School of Electrical Engineering and Computer Sciences
National University of Science and Technology

MACHINE LEARNING

Assignment No. 1

K-Nearest Neighbour

Muhammad Hamza Shakoor
Reg # 255633
BEE 10
ML Group - 01
SEECS NUST

Abstract:

In this assignment we were given a dataset of 50 images of celebrities and we had to compare them with our picture to see who is our celebrity look alike, using K-Nearest Neighbour technique.
Coding for this assignment is done in “python” programming language and the IDE we used is “Google Colaboratory”.

Tutorial of Program:

Before running this block, make sure to upload "data.mat" in the google drive.

```
[ ]  #Enter your path of dataset from google drive
     import scipy.io as sio
     GOOGLE_COLAB = True
     path = ""
     if GOOGLE_COLAB:
         from google.colab import drive, files
         drive.mount('/content/drive/')
         path = "/content/drive/My Drive/Machine Learning/"
```

- **We imported our libraries that we needed to mount our drive with google colaboratory.**
- **Path is set to our folder on the drive where all the data is stored.**

```
dataset = path + "data.mat"

#Enter path of your test image
test_image=path+"test.jpg"
mat_contents = sio.loadmat(dataset)
mat_contents
images = mat_contents['images']
label = mat_contents['C']
images.shape
```

- **data.mat is our dataset file, so we set our path for dataset in "dataset" variable.**
- **Similarly path for test image is set.**
- **The next few lines of code load our dataset into the program**
- **We set images from dataset in "images variable"**
- **And content (data) of those images in "label" variable.**
- **Then we are just printing the shape of those images**

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pf

Enter your authorization code:
```

- **At first try, google colab asks for permission to mount google drive, click on this link, then copy the code they provide in that box.**

```
Mounted at /content/drive/
(50, 3072)
```

```
import numpy as np
images= np.transpose(images)
images.shape
im = np.reshape(images, [ 32, 32, 3, -1], order="F")
```

- **Numpy is imported as np**
- **We are taking the transpose of our dataset images**
- **Transpose, from (50, 3072) to (3072, 50)**
- **The 3 dimensional (32x32x3) vector represents color images**
- **Then we are reshaping the images vector into a single 1D vector.**
- **Order 'F' in np.reshape() is just optional, it means to read / write the elements using Fortran-like index order, with the first index changing fastest, and the last index changing slowest.**

```
from matplotlib import pyplot as plt
#import cv2

plt.imshow(im[:,:,:,40])
print(im[:,:,:,0].shape)
a= im[:,:,:,40]
plt.imshow(a)
a.shape
```

- **After importing matplotlib, we are storing 41th image from the dataset into the variable "a". ':' means that we are including every element present in array at that place.**
- **Then we are just plotting that image and printing its shape size.**

```
(32, 32, 3)
(32, 32, 3)
```


```
from scipy import misc
import cv2
from math import sqrt
import numpy as np
from numpy import ndarray
```

- **Importing necessary libraries and functions**

```
for i in range(50):
    G = im[:,:,:,i]
    G = np.reshape(G,[-1], order="F")
```

- **Reshaping all the images of the dataset just like we did before**

```
    #Read your image here
####### Your code here #######
from PIL import Image
image = Image.open('drive/My Drive/Machine Learning/hamza smile.jpg')
image.show()
```

- **Reading my test image into the program using PIL library**
- **I used PIL because I was already aware of it instead of CV2.**

```
# The file format of the source file.
print(image.format) # Output: JPEG

# Image size, in pixels. The size is given as a 2-tuple (width, height).
print(image.size)
############################
    #Resize your image
####### Your code here #######
new_image = image.resize((32, 32))
new_image.save('image_32.jpg')

print(image.size)
print(new_image.size) # Output: (32, 32)
new_im = np.reshape(new_image, [ 32, 32, 3, -1], order="F")
plt.imshow(new_im[:,:,:,0])
print(new_im[:,:,:,0].shape)
new_a= new_im[:,:,:,0]
plt.imshow(new_a)
new_a.shape
```

- **Then we are just printing the format and size of the test image to check the dimensions.**
- **After that we are resizing and reshaping the image to 32*32*3 size**
- **We are just converting our image to a single vector of length 32x3 = 96**
- **we converted the dataset images into a single vector of length 3072**

```
############################
    #Calculate Eucleadian distance between your image and dataset
####### Your code here #######
i1=new_a
i2=im[:,:,:,40]
def calculateDistance(i1, i2):
    return (np.sum((i1-i2)**2))**0.5
calculateDistance(i1,i2)
############################
```

- **Now we are finding the Eucleadian distance using this formula:**

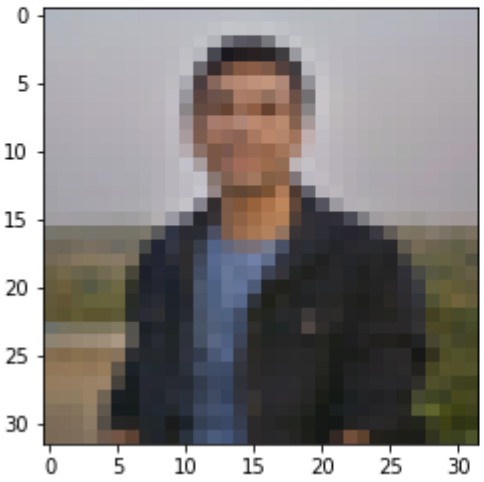
$$d(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

p, q = two points in Euclidean n-space

q_i, p_i = Euclidean vectors, starting from the origin of the space (initial point)

n = n-space

```
JPEG
(769, 769)
(769, 769)
(32, 32)
(32, 32, 3)
569.2749774933025
```



- **569.2749 is the Euclidean distance between my test image and 41th image of the dataset**

Code for 1NN:

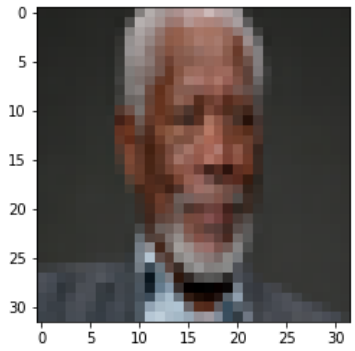
```
# Write code for 1 NN
#Find min distance
#Find at which point min value exists

####### Your code here #######
i1=new_a
d=[]
for index in range(50):
  i2=im[:,:,:,index]
  n=calculateDistance(i1,i2)
  d.append(n)
print (d)
```

- **Now we calculating euclidean distance between our test image and all the images of our data set using for loop**
- **And appending all those distances in a new list, d [ ]**

```
#min distance
min_d=min(d)
print(min_d)
#index of minimum value in list
min_index=d.index(min_d)
print(min_index)
##############################
print("your image matches with this actor ")
plt.imshow(im[:,:,:,min_index])
print(im[:,:,:,min_index].shape)
```

- **Now we are finding minimum distance, which is the minimum element of list d.**
- **After that we are finding the index of that distance value.**
- **Index is used to plot the relative image on the screen.**

```
[549.5707415792803, 575.8089961089528, 563.0896909019024, 573.7307730983235, 543.581640602403,
543.581640602403
4
your image matches with this actor
(32, 32, 3)
```



- **4th index (5th image was our closest match)**

Code for 3NN:

```
[13] #3 NN
     #Write code for 3 NN
     #Find 3 min distances
     #Find their instances
     #hint: Use for Loop

     ######## Your code here ########
     #i1=new_a
     d=[]
     for index in range(50):
       i2=im[:,:,:,index]
       n=calculateDistance(i1,i2)
       d.append(n)
     print (d)

     #3 min distances
     def lowest_three(a, n):
       return np.partition(a, n-1)[:n]
     min_three_list=lowest_three(d, 3)
     print(min_three_list)
```

- **Similary finding the distance as we did before**
- **Only new thing here is, now we are finding 3 least number from that list of eucliadean distances.**

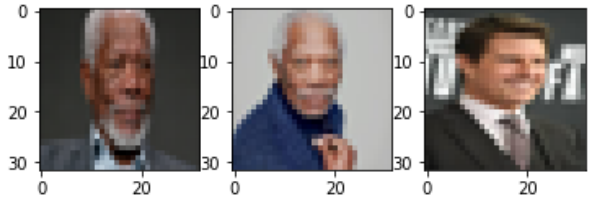
```
######################
min_index_list=[]
#index of 3 minimum values in list
for i in range(3):
  min_index=d.index(min_three_list[i])
  min_index_list.append(min_index)
print(min_index_list)
##############################
```

- **These 3 numbers provide the index to those related images.**

```
print("your image matches with these 3 actors: ")
################################
f, axarr = plt.subplots(1,3)
axarr[0].imshow(im[:,:,:,min_index_list[0]])
axarr[1].imshow(im[:,:,:,min_index_list[1]])
axarr[2].imshow(im[:,:,:,min_index_list[2]])
print(im[:,:,:,min_index_list[2]].shape)
```

- **We are plotting 3 matching images using subplots**

```
[549.5707415792803, 575.8089961089528, 563.0896909019024, 573.7307730983235, 543.581640602403
[543.5816406  549.57074158 553.08498443]
[4, 0, 13]
your image matches with these 3 actors:
(32, 32, 3)
```



Code for 5NN:

```
[14] #Write code for 5 NN
     #Find 5 min distances
     #Find their instances

     ####### Your code here #######
     d=[]
     for index in range(50):
       i2=im[:,:,:,index]
       n=calculateDistance(i1,i2)
       d.append(n)
     print (d)

     #5 min distances
     def lowest_five(a, n):
       return np.partition(a, n-1)[:n]
     min_five_list=lowest_five(d, 5)
     print(min_five_list)
```

- **Everything is similar to 3NN code, now we are just finding 5 least numbers from that list of eucliadean distances.**

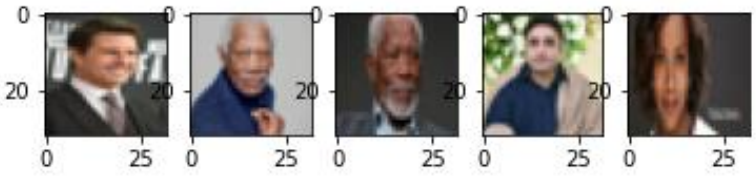
```
######################
min_index_list=[]
#index of 3 minimum values in list
for i in range(5):
  min_index=d.index(min_five_list[i])
  min_index_list.append(min_index)
print(min_index_list)
```

- **And getting images through their indexes and plotting them using the code given below:**

```
##############################
print("your image matches with these 5 actors: ")
##############################
f, axarr = plt.subplots(1,5)
axarr[0].imshow(im[:,:,:,min_index_list[0]])
axarr[1].imshow(im[:,:,:,min_index_list[1]])
axarr[2].imshow(im[:,:,:,min_index_list[2]])
axarr[3].imshow(im[:,:,:,min_index_list[3]])
axarr[4].imshow(im[:,:,:,min_index_list[4]])
print(im[:,:,:,min_index_list[4]].shape)
##############################
```

- **This code prints out 5 images that are closely matching to our test image**

```
[549.5707415792803, 575.8089961089528, 563.0896909019024, 573.7307730983
[553.08498443 549.57074158 543.5816406  553.75626407 556.70458953]
[13, 0, 4, 27, 34]
your image matches with these 5 actors:
(32, 32, 3)
```



- **Final result of 5NN**