



CSE 573

## Project Report

Name: Aditya Athota

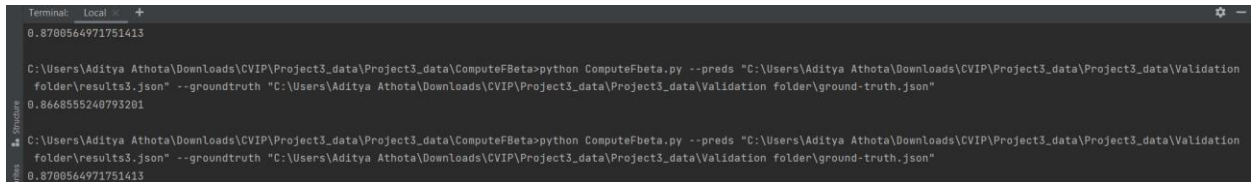
UBIT Name: adityaat

UB Person Number: 50368003

### Task I

- > I am performing the Face Detection using the OpenCV's deep neural network module.
- > The pre-requirements needed for this module are:
  - a. ". prototxt" file defines the model architecture
  - b. ". caffemodel" file which has the actual nodes of neural network
- > Initially, I have imported the required libraries such as cv2, numpy, json that are required for the task.
- > Read the image paths that are present in the validation folder and assign those paths to the variable "imagePaths".
- > We load the model using the pre-requirement files that are mentioned above and store it in the variable "d\_n\_net".
- > Now, read each image from the "imagePaths" list by using the "for loop".
- > In the "for loop", at first, read the image and store the image dimensions.
- > Next, we pass the image into "cv2.dnn.blobFromImage" where the data is pre-processed such as setting the dimensions and normalization and assign this to the variable named "blob".
- > We pass the variable blob into the "d\_n\_net" in order to detect the faces.
- > After that, we will start a "for loop" to draw the boxes.
- > Before starting the loop, we assign the detected faces in an image to a variable "confidence". In the loop, before drawing the box, we compare this "confidence" value with a threshold value that we give manually. The boxes will be drawn after this condition is met.
- > After, the threshold condition is met, we calculate (x, y) co-ordinates of the bounding box and formulate the box.
- > The co-ordinates of this box are then converted into type "int" and then assigned each co-ordinate into different variable.
- > Using these variables, the number of boxes that were formed in the image and the co-ordinates of the box are stored in the variable "element" which then appended into the "json\_list" array.
- > After storing the boxes of all images into the "json\_list", create a json file and dump this list in the file.

- > We then compare our json file with the "ground-truth.json" with the help of "ComputeFBeta.py" to get the F1 score.
- > Following is the F1 score that has been achieved: "0.8700564971751413".



```

Terminal: Local
0.8700564971751413

C:\Users\Aditya Athota\Downloads\CVIP\Project3_data\Project3_data\ComputeFBeta>python ComputeFBeta.py --preds "C:\Users\Aditya Athota\Downloads\CVIP\Project3_data\Project3_data\Validation
folder\results3.json" --groundtruth "C:\Users\Aditya Athota\Downloads\CVIP\Project3_data\Project3_data\Validation folder\ground-truth.json"
0.866855240793201

C:\Users\Aditya Athota\Downloads\CVIP\Project3_data\Project3_data\ComputeFBeta>python ComputeFBeta.py --preds "C:\Users\Aditya Athota\Downloads\CVIP\Project3_data\Project3_data\Validation
folder\results3.json" --groundtruth "C:\Users\Aditya Athota\Downloads\CVIP\Project3_data\Project3_data\Validation folder\ground-truth.json"
0.8700564971751413

```

## Task 2

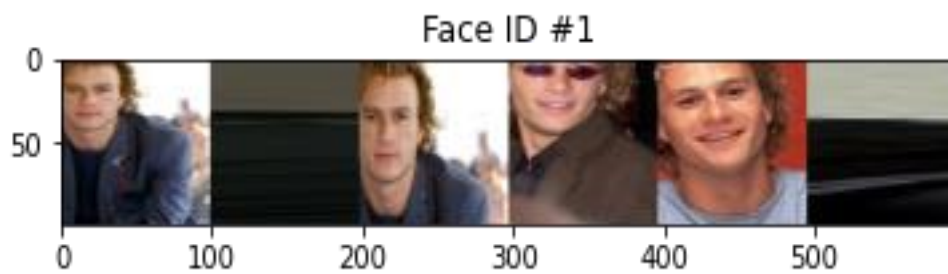
- > This task is about clustering the similar images using the k-means algorithm.
- > Initially, the required libraries were loaded and image paths were stored which is similar to the task 1.
- > The number of clusters that are required were taken from the given path.
- > Now, we begin the loop over the "imagePaths" array where all the images paths were stored.
- > Here in the loop, initially, we read and load the image and change the image color using "rgb".
- > Now, we recognize the face of each image using "face\_recognition" API.
- > Inside the face\_recognition API, we are passing CNN face detector to get better results.
- > Now, we compute each image into 128-dimensional vector and stored it in the variable "encodings".
- > The resulted encodings of each image is then stored in an array "data".
- > This "data" is then passed into the "KMeans" class where the clusters are formed.
- > In the "KMeans" class, initially, we pass the number of clusters that were given in the image path and the number of iterations required.
- > After the initialization, we select random samples in the data using "np.random" and initialize the centroids.
- > Now, we form a loop over the number of iterations that were given manually. In the loop, we get the clusters by calculating the Euclidean distance on the centroids and assigning the nearest centroid that were obtained.
- > Now, in the clusters that were formed, we find the new centroid and repeat the same process as mentioned above to get the new nearest clusters.
- > This process is repeated until all the iterations that we assigned manually are done in the "for loop".
- > Finally, based on the nearest clusters, the cluster labels were assigned to each vector i.e., each image.
- > We then find out the labels of clusters that were formed and assign it to the variable "labelIDs".

- > Now, we start a loop over the "labelIDs", and in that we read each image, find out the image box that needs to be displayed, resize the image and concatenate the similar images.
- > Following are the results of the cluster images that were formed:

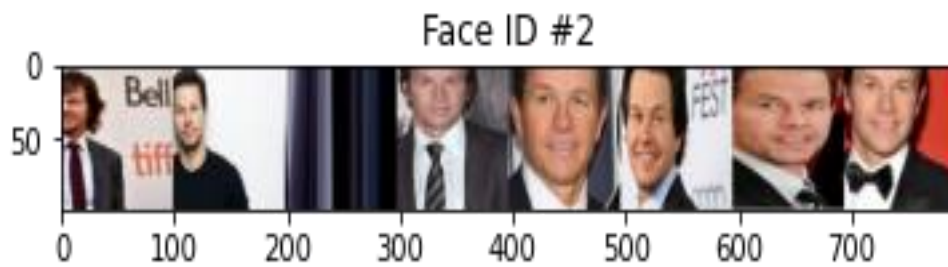
Cluster 0:



Cluster 1:



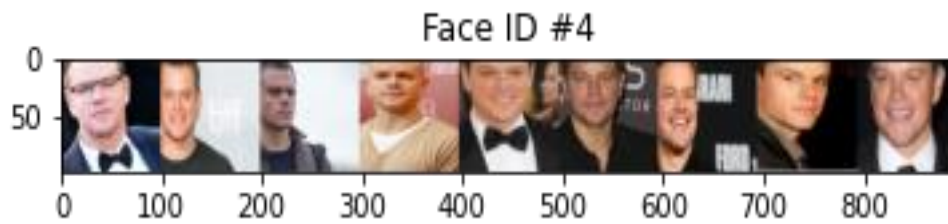
Cluster 2:



Cluster 3:



Cluster 4:



- > At last, the number of images that were formed for each cluster is stores in the variable data which is then being appended by the "json\_list" array. This "json\_list" is then dumped in the json library to get the "cluster.json" file.