Vaibhav Rao
50375332

# **REPORT PROJECT 3 = CV-CSE573**

## For Task 1:

For task 1, I used the OpenCV API CascadeClassifier for detection of faces in an Image. This classifier uses an XML file already trained on multiple datasets. I used the File for Frontal face detection as it gave me the best accuracy for the given dataset.

Incase I wanted to increase my accuracy, and none of the said files would have been appropriate, I would have to train a model and create a new xml file that could have been able to detect frontal + side faces which I feel could have helped increase accuracy.

I tried the following and got some of the below results with hit-and-trial method to select the best parameters.

File: haarcascade_frontalface_alt.xml, Scaling: 1.05: ComputeFbeta score : 0.8256880733944955
File: haarcascade_frontalface_alt.xml, Scaling: 1.06: ComputeFbeta score : 0.8338557993730408

File: haarcascade_frontalface_alt2.xml – max accuracy around 0.7817589576547231

File: haarcascade_frontalface_default.xml Maximum accuracy around 81% and 82%

Most of these models Performed well with around 1.2 scaling and 5 neighbours. Increasing scaling made it difficult to find faces.
The only challenge was most of these files were not trained on all the datasets. Apart from frontal face, I tried files for Profile Faces, Side Faces (Which required rotation to capture some more side faces) and LBP Cascade Classifier. However I got the best results from frontal face since most of the images had this.

## For TASK 2:

For task 2 I used the same portion as task 1 to identify the faces in the image. Here since there was only 1 face per image (As mentioned in the description, I have just extracted the 1st Bounding box result from detectMultiScale.

After that, I do a face-encoding for the image inside the BBOX(Face of our target) and then I applied a K-Means Clustering algorithm which was the best fit for this scenario.

Vaibhav Rao
50375332

Since we knew the number of clusters and did not have much information about our target variable (apart from human judgement) this unsupervised clustering algorithm worked best in this scenario.

For the algorithm K-means, This works with the idea to pick random centres from our dataset (Number of points = number of clusters)

Then these centres are evaluated with all points in the dataset. The points closest to each centre (Euclidian Distance) is labelled to belong to that cluster.

After this to assess, Each point of this labelled class is taken and we get a mean value of the entire class cluster to analyze if the centre is closest to the mean. If not new centres our chosen as the best centres would actually be centrally placed within a cluster

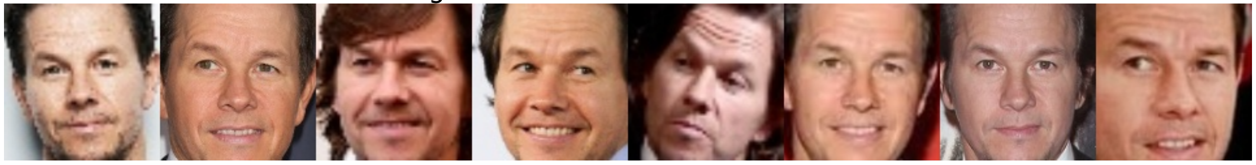
We extract the centres(Which I locally used for plotting using matplotlib to understand visually how the algorithm was performing) We also extract Label information of this data (Once the centres have converged) I have also seeded this so that the random prediction somehow stays unique during different runs as this gave me the best results.

We use the labels of out train data extract the predicted class cluster and use this information to create our cluster.json file.

Some file-processing, sorting and creating the json file logic had to be applied.

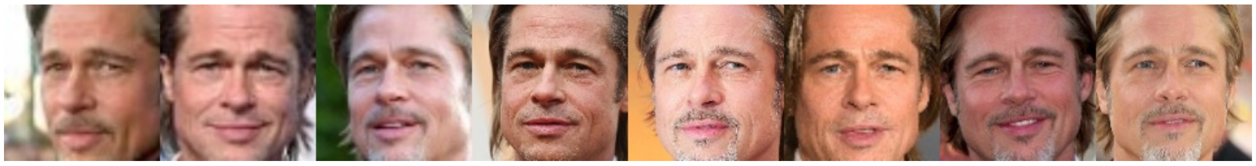
Clusters:

Cluster 0 = Mark Wahlberg



The algorithm was able to correctly identify all images in this cluster.

Cluster 1 = Brad Pit



The algorithm was able to correctly identify all images in this cluster.

Vaibhav Rao
50375332

Cluster 2: = Robert Pattinson

The algorithm was able to correctly identify all images in this cluster.

Cluster 3: = Heath Ledger

The algorithm was able to correctly identify all images in this cluster.

Cluster 4 = Matt Damon

The algorithm was able to correctly identify all images in this cluster.