University at Buffalo
Department of Computer Science and and Engineering
CSE 473/573 - Computer Vision and Image Processing

Naman Pundir
50373843
namanpun@buffalo.edu

****PROJECT-3****

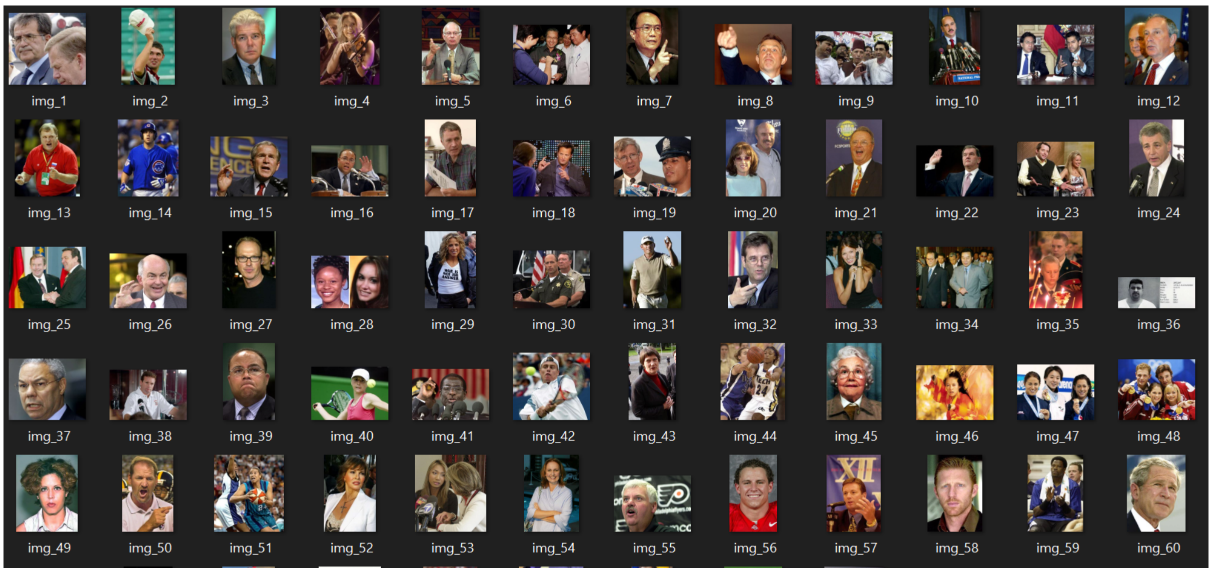
Face Detection in the Wild

TASK-1

Objective: Face Detection on the images given in a folder.

Dataset: Validation folder/images and Test folder/images

Images:



Face Detection: Face detection is a computer vision technology to identify human faces in digital images or digital videos.

Algorithms Tried: I tried the following OpenCV algorithms:

haarcascade_frontalface_default.xml

haarcascade_frontalface_alt.xml

haarcascade_profileface.xml

lbpcascade_frontalface.xml

Lbpcascade_profileface.xml

Source: https://github.com/opencv/opencv/tree/master/data

Final Algorithm: haarcascade_frontalface_default.xml

After trying all the above xml files in my code I got maximum accuracy with **haarcascade_frontalface_default.xml.**

HaarCascade: Haar is an Object Detection Algorithm which is used to detect faces in a digital image or digital video. Haar algorithm uses edge or line detection features proposed by **Viola and Jones** in their research paper "Rapid Object Detection using a Boosted Cascade of Simple Features" published in 2001.

Haarcascade_frontalface_default.xml: This particular haar xml algorithm is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

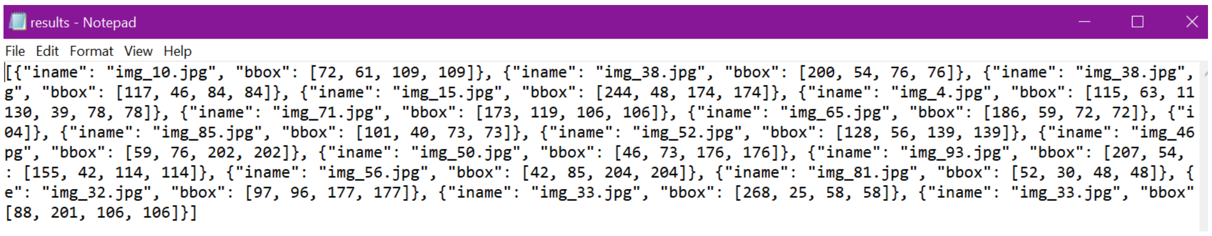
Results:

Accuracy with Validation images and ComputeFBeta.py file:

F1 score: 82.3%

All the detected faces were dumped in a json file "results.json" which contain all the bounding boxes of the detected faces.

results.json:



```
[{"iname": "img_10.jpg", "bbox": [72, 61, 109, 109]}, {"iname": "img_38.jpg", "bbox": [200, 54, 76, 76]}, {"iname": "img_38.jpg",
g", "bbox": [117, 46, 84, 84]}, {"iname": "img_15.jpg", "bbox": [244, 48, 174, 174]}, {"iname": "img_4.jpg", "bbox": [115, 63, 11
130, 39, 78, 78]}, {"iname": "img_71.jpg", "bbox": [173, 119, 106, 106]}, {"iname": "img_65.jpg", "bbox": [186, 59, 72, 72]}, {"i
04]}, {"iname": "img_85.jpg", "bbox": [101, 40, 73, 73]}, {"iname": "img_52.jpg", "bbox": [128, 56, 139, 139]}, {"iname": "img_46
pg", "bbox": [59, 76, 202, 202]}, {"iname": "img_50.jpg", "bbox": [46, 73, 176, 176]}, {"iname": "img_93.jpg", "bbox": [207, 54,
: [155, 42, 114, 114]}, {"iname": "img_56.jpg", "bbox": [42, 85, 204, 204]}, {"iname": "img_81.jpg", "bbox": [52, 30, 48, 48]}, {
e": "img_32.jpg", "bbox": [97, 96, 177, 177]}, {"iname": "img_33.jpg", "bbox": [268, 25, 58, 58]}, {"iname": "img_33.jpg", "bbox"
[88, 201, 106, 106]}]
```

Accuracy was not too much because some images contain just profile faces or side profile faces and haar cascade frontal default is not that good with side profile faces. On the contrary when I used a haar cascade profile it was good with profile faces but overall it was not as good as the frontal default algorithm.

Implementation Challenges: The only challenge I faced in this task was to determine the correct parameters like "**scaleFactor" and "minNeighbors "** for my code. **I ended up with "1.2" and "5".**

I tested my code with the Test folder and some random images from the internet also. I noticed that random images which have a group of people but all facing front were giving 100% accuracy while some images having just side faces like half face, looking sky etc. were giving low accuracy results.

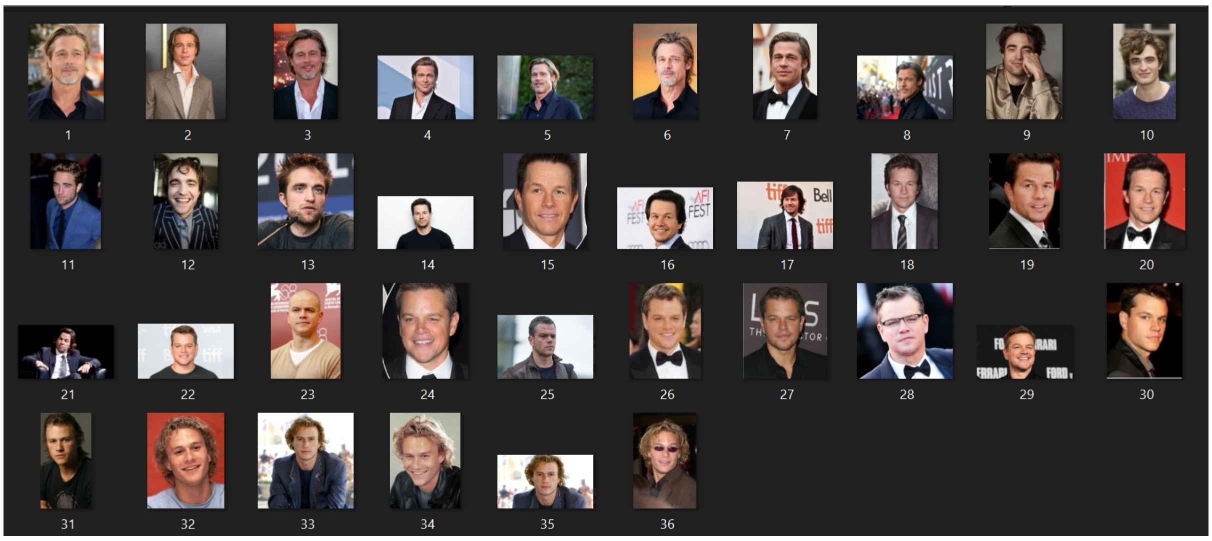
****

TASK-2

Objective: Face Detection and clustering on the images given in a folder.

Dataset: faceCluster_5 : images of 5 hollywood actors.

Images:



Clustering: Clustering is a machine learning algorithm an unsupervised machine learning method which groups together the same data from a dataset. All the like data will be grouped together. It is an unsupervised learning method which means a program learns and understands itself without the need of any supervisor.

Algorithm Tried:

K-Means

K-Nearest Neighbour

Final Algorithm:

K-Means Algorithm

After using the above two algorithms I ended up with the K-means algorithm because I was getting 100% accuracy in grouping the images and implementing K-means was less complex then KNN.

K-Means Algorithm:

Intuitive working:

1. Select K random points
2. Calculate centroid for all the points and assign points to closest centroid
3. Repeat till converge

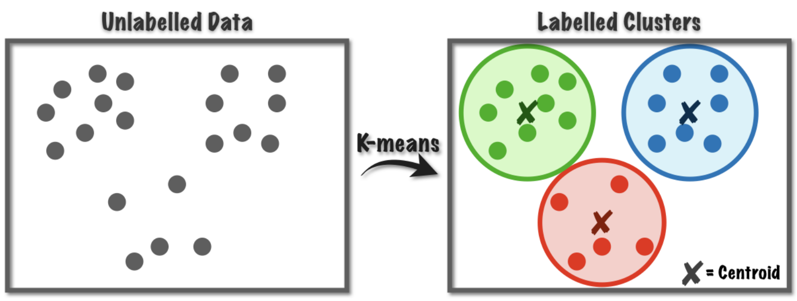
Random initialization problem:

1. Random initialization doesn't always work

2. To solve this problem use KMeans++

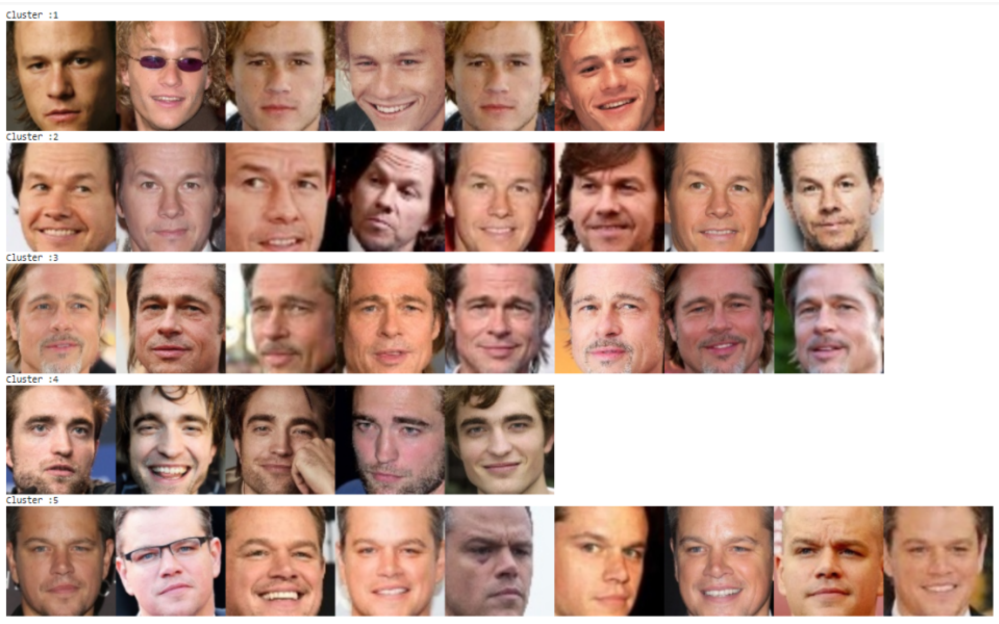
Selecting number of clusters:

Within Cluster Sum of Squares is the sum of squares of the distance of every point to the nearest cluster. Initially it will be very high and then it will lower and lower and will eventually reach 0. We have to make a call and decide what is the best value of K depending on the slope obtained on increasing the cluster. If at a point, an increase in the number of clusters doesn't decrease a substantial amount of WCSS then that's the ideal number of clusters.

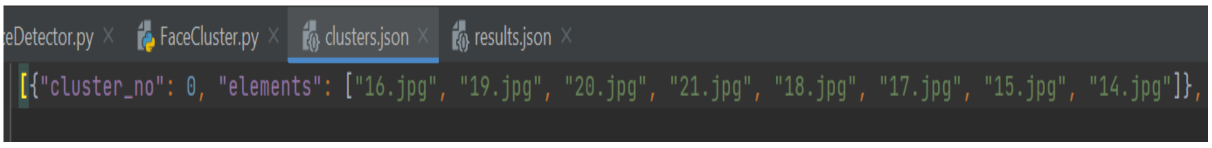


Results:

After applying my code to give faceCLuster_5 dataset I am getting **100% accuracy** i.e all the clusters are perfect.



Final clusters were dumped in a json file “clusters.json”:



```
[{"cluster_no": 0, "elements": ["16.jpg", "19.jpg", "20.jpg", "21.jpg", "18.jpg", "17.jpg", "15.jpg", "14.jpg"]},
```

Accuracy was **100%** that means all the grouped images were perfect and I also tried my code with some internet images it was also giving **100%** accuracy.

The basic steps I used to complete this task were:

1. Uploading the images from the path.
2. Using task 1 code I detected the faces in all the given images.
3. I cropped the detected faces and stored them in a different folder.
4. Uploaded the cropped faces images.
5. Encoding of all the cropped faces.
6. Applying k means algorithm to these encoded faces.
7. Labeling of all the images
8. Grouping the same labels together.
9. Dumping the groups to clusters.json

References I used to complete Project-3:

1.https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_objdetect/py_face_detection/py_face_detection.html

2.https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html

3.https://pypi.org/project/face-recognition/

4. https://realpython.com/k-means-clustering-python/

5. https://numpy.org/doc/stable/reference/generated/numpy.hstack.html

****