Task 1:

Algorithm:

- Haar Cascading was implemented.
- OpenCV repository was referred for the xml file that is consumed to detect faces.
- Default file was used, which is capable of detecting frontal faces.
- Other files are also available for improving the accuracy and detecting side faces as well.
- Verification was done with the default and 2 other xml files(haarcascade_frontalface_alt and haarcascade_frontalface_alt2) however the default xml provided better f1 score comparatively.
- Tuning was also done for scalefactor. After implementing various values the scale factor was finalized to be 1.2 with an f1 score of around 0.8064 on the validation set.
- A method to crop detected images along with specifications for task 1 was created to be consumed for Task 2 as well.

Results & Challenges:

- As per my evaluation for F1 score, the objective of getting an acceptable score was achieved after various experimentation with the parameters.
- An autorun main method, that performed the clustering,and created the json file for f1 computation was implemented.
- Implementing system arguments was a bit of a challenge, due to lack of my experience with them.
- Another challenge was reading all image filenames. For this a very basic solution was implemented. The directory was changed to the image folder and with the help of listdir() the names were extracted. The directory was then reverted to previous location. This was required to get image names corresponding to all clusters that were detected.
- Json creation also proved challenging, due to data type related issues, which were a result of my implementation approach. To mitigate this issue, the required results were first read into a dataframe before conversion to json.

Task 2:

Algorithm:

- K means clustering was implemented.
- Firstly, cropped images were generated using get_faces method from Task 1, these were on to the encoder to obtain image vectors of size 128 each.
- K cluster centers were randomly selected at first.
- Based on these points, the sample image vectors were then clustered using a simple sum of squared distances based implementation.
- The newly generated clusters were then used to generate means which acted as new cluster centers.
- This was repeated unless the centers computed were the same for the current run.
- Based on the clusters generated images were concatenated using opencv's horizontal concatenation based implementation.
- Based on the clusters generated, the output was populated in clusters.json

Results & Challenges:

- Due to random choice of cluster points, it was observed that in certain runs, the cluster would turn out to be empty, due to which a warning was given (RuntimeWarning: Mean of empty slice). In such a scenario, a rerun of the code would usually lead to acceptable results. The root cause could not be identified for this due to prior commitments.
- Based on the below images, the clusters seem to be acceptable.
- All 36 faces were detected, cropped and clustered (Refer below figures).

Figure: Cluster_0

Figure: Cluster_1

Figure: Cluster_2

Figure: Cluster_3

Figure: Cluster_4