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

Fall 2020
Project #3
Due Date: 12/11/20, 11:59PM

1 Face Detection in the Wild (8 Points)

The goal of this project is to have you utilize the face detection algorithms available in opencv library (preferably opencv version > 4.0).

1.1 Project Description

Given a face detection dataset composed of hundreds of images, the goal is to detect faces contained in the images. The detector should be able to locate all the faces in any testing image. Figure 1 shows an example of performing face detection. We will use a subset of FDDB [1] as the dataset for this project. You can use any face detection modules available in OpenCV.

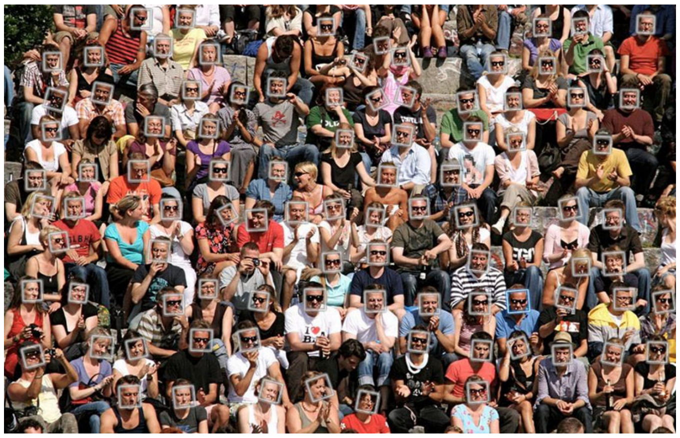


Figure 1: An example of performing face detection. The detected faces are annotated using gray bounding boxes.

1.2 Libraries permitted and prohibited

- Any API provided by OpenCV.

1.3 Data and Evaluation

You will be given 100 images along with ground-truth annotations (validation folder). You can evaluate your model performance on this validation set or use it to further improve your detection.

During testing, you need report results on another 100 images (test folder) without ground truth annotations.

The following bash script will be used to evaluate the performance of the face detector you created:

```
unzip FaceDetection.zip
cd [FaceDetection]
python3 FaceDetector.py [data-directory]
python3 ComputeFBeta.py ./results.json [ground-truth.json]
```

`FaceDetector.py` contains YOUR python code that will able to detect faces in all the images in `[data-directory]` and generate a json file, *i.e.*, `results.json`, that stores all the bounding boxes of the detected faces. The generated json file shall be located in the same folder where the images are located. The bounding boxes of the detected faces should be stored in a list using the following format:

```
[{"iname": "img.jpg", "bbox": [x, y, width, height]}, ...]
```

`"img.jpg"` is an example of the name of an image; `x` and `y` are the row index and column index of the top-left corner of the bounding box; `width` and `height` are the width and height of the bounding box, respectively. `x`, `y`, `width` and `height` should be integers.

We will provide `ComputeFBeta.py`, a python code that computes f_β using `results.json` and the ground truth in `[ground-truth.json]`.

You can refer to the sample json provided to you for more information. There is also a piece of example code regarding how to create json file.

1.4 Evaluation Rubric

We will be awarding 5 points for F1 score computed, 1 point for code and 2 points for report.
For F1 score computed using ComputeFBeta.py. (5 points)

- F1 > 0.5: 5 points
- F1 > 0.35: 4 points
- F1 > 0.2: 3 points
- F1 > 0.05: 2 points
- F1: > 0.01: 1 points
- F1: < 0.01: 0 points

Code:

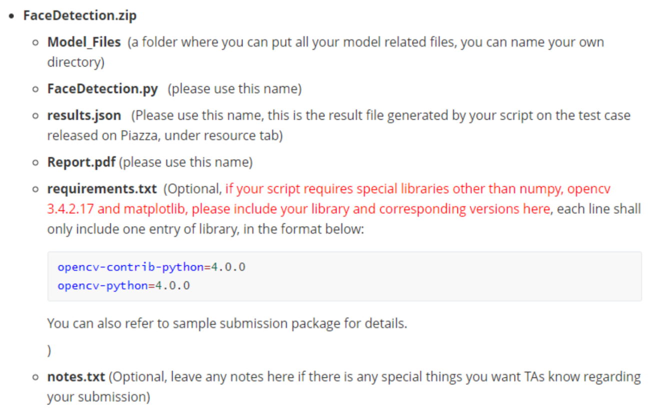
- Submitted code is reasonable and interpretable (1 point)

Report:

- Concise description of your implementation (1 point)
- Discussion of the results and and implementation issues (1 point)

1.5 Code and Report

Submission package requirements:



- **FaceDetection.zip**
  - **Model_Files** (a folder where you can put all your model related files, you can name your own directory)
  - **FaceDetection.py** (please use this name)
  - **results.json** (Please use this name, this is the result file generated by your script on the test case released on Piazza, under resource tab)
  - **Report.pdf** (please use this name)
  - **requirements.txt** (Optional, if your script requires special libraries other than numpy, opencv 3.4.2.17 and matplotlib, please include your library and corresponding versions here, each line shall only include one entry of library, in the format below:

    ```
    opencv-contrib-python=4.0.0
    opencv-python=4.0.0
    ```

    You can also refer to sample submission package for details.

)
  - **notes.txt** (Optional, leave any notes here if there is any special things you want TAs know regarding your submission)

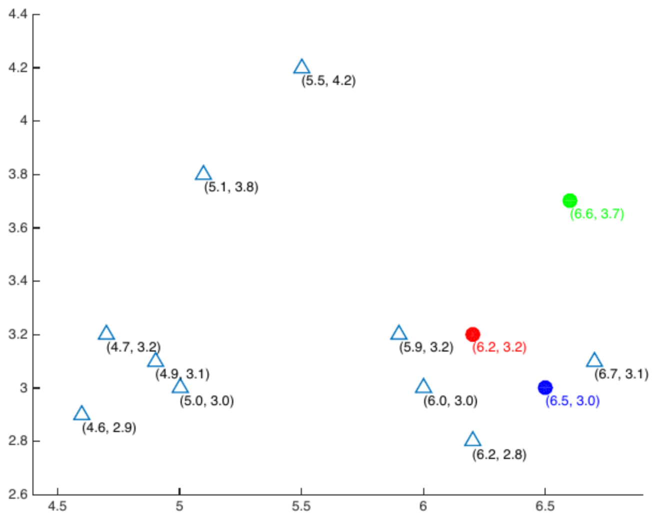
As shown in 1.5, please create a FaceDetection.zip accordingly. Since this project allows more flexibility in the use of libraries, please also remember to include a file with your resources your code requires to make it easier on the grader. This should be in the form of a txt file named “requirements.txt”. The file named “requirements.txt” should specify the libraries you used and the version of the libraries, and our grader should be able to install all the libraries you used by using the command: pip install -r requirements.txt. Note: Please submit your code with title as ‘FaceDetector.py’ (ignore the name shown in above image).

In addition to your code, a report is required with this project. It should contain

- Your name and your UBID.
- A description of your implementation,

The report should be a pdf file. You will be graded based on the report, code and an evaluation on the 100 images in test folder. You only need to upload the results.json in the test folder and do not need to upload any image results during submission.

2 K-means Clustering (7 points)



$$X = \begin{bmatrix} 5.9 & 3.2 \\ 4.6 & 2.9 \\ 6.2 & 2.8 \\ 4.7 & 3.2 \\ 5.5 & 4.2 \\ 5.0 & 3.0 \\ 4.9 & 3.1 \\ 6.7 & 3.1 \\ 5.1 & 3.8 \\ 6.0 & 3.0 \end{bmatrix}$$

Given the matrix X whose rows represent different data points, you are asked to perform a k-means clustering on this dataset using the Euclidean distance as the distance function. Here k is chosen as 3. All data in X were plotted in above Figure. The centers of 3 clusters were initialized as $\mu_1 = (6.2, 3.2)$ (red), $\mu_2 = (6.6, 3.7)$ (green), $\mu_3 = (6.5, 3.0)$ (blue).

Implement the k-means clustering algorithm **(you are only allowed to use the basic numpy routines such as numpy.mean(), numpy.sum(), etc., to implement the algorithm. You are not allowed to use any numpy or openCV libraries that have 'knn', 'kmeans', 'cluster' in the function name.)**.

- Classify $N = 10$ samples according to nearest $\mu_i (i = 1, 2, 3)$. Plot the results by coloring the empty triangles in red, blue or green. Include the classification vector and the classification plot (`task2_iter1_a.jpg`) in the report. (1 point)

  Hint: Using **plt.scatter** with edgecolor, facecolor, marker and `plt.text` to plot the figure.

  Note: Classification vector is a vector of size 10 (since N=10) with values of 0 (Red), 1 (Green) or 2 (Blue), which indicate the cluster a data point belongs to.

- Recompute μ_i. Plot the updated μ_i in solid circle in red, blue, and green respectively. Include the updated values and the plot in the report (`task2_iter1_b.jpg`) (1 point).

- For a second iteration, plot the classification plot and updated μ_i plot for the second iteration. Include the classification vector and updated μ_i values and these two plots (`task2_iter2_a.jpg`, `task2_iter2_b.jpg`) in the report. (1 point)

Figure 2: baboon.png

- **Color Quantization** (4 points) Apply k-means to image color quantization. Using only k colors to represent the image `baboon.png` (shown in Figure. 2). Include the color quantized images for $k = 3, 5, 10, 20$ (`task2_baboon_3.jpg, task2_baboon_5.jpg, task2_baboon_10.jpg, task2_baboon_20.jpg`).

2.1 Code and Report

Place the code 'task2.py' and report 'task2_report.pdf' in k-means.zip folder. You need to write a separate report for task 2 with all the above mentioned plots/images, along with a description on the implementation of your algorithm.
Note: Color quantization code should finish within few minutes ($<$ 10 minutes).

3 Submission Guidelines

- Unlimited number of submissions is allowed and only the latest submission will be used for grading. Add FaceDetection.zip and k-means.zip into a folder and create 'UBID_project3.zip'. Please upload this to UBlearns.
- Not following submission guidelines strictly will result in large penalty.
- Identical code will be treated as plagiarism. Please work it out independently.
- For codes raising "RuntimeError", the grade will be 0 for this task.
- Late submissions within 24 hours are allowed and will result in a 40% penalty. After one day, submissions will not be accepted.

References

[1] V. Jain and E. L. Miller, “Fddb: a benchmark for face detection in unconstrained settings,” 2010.