CS 763: Task 02: Face It

- Announced: 16 Feb. Due: Feb 23 10PM
- Please write (only if true) the honor code. You can find the honor code on the web page. If you used any source (person or thing) explicitly state it.
- **This is NOT an individual assignment**

1 Overview

This task is a follow up on the content we covered in class on "faces" and computer vision. In this task, you detect, identify, and have some fun performing makeup on faces.

You will use `face_recognition` from `https://github.com/ageitgey/face_recognition`; it's not part of `opencv` but provides a user friendly package. However, you may notice that installing this can be a tad tricky.

Read the submission instructions carefully.

2 Face Detection

2.1 Images

(a) Input Image (b) Detected Face 1 (c) Detected Face 2

Figure 1: Extracted faces from an image. Notice that the third potential face is not shown.

This subtask is divided in the following steps. See Fig. 1 for context.

1. From the sample face-detection image(`data/samples/arnold-sylvester`), detect and extract individual faces. The output should include the bounding box coordinates of each detected face (on the standard output), which in-turn will be used to extract all faces as sub-images. Save resultant images (see below for nomenclature).
2. Use the dataset from the previous task (Task01) with faces of your group members, consisting of at least 2 images per person, and make sure all the members in your group are present in at least 1 image. Store the images in `data/captured` as before (for example, `01.jpg, 02.jpg`). You are not allowed to change the existing images that you already submitted. However, you can add to the dataset if absolutely necessary.
3. Repeat Step 1 on your dataset.

4. Store outputs with `face` followed by your input file name followed by (if necessary for multiple faces) `suffix` in `results/faceDetection/` (For example `face01suffix01.jpg, face02.jpg, and so on`.) Comment on what worked, what didn't in your answers.

```
python face-detection.py --data path-to-image \
--faces path-to-save-faces --type 1
```

```
Example: python face-detection.py --data ../data/captured/01.jpg \
--faces ../results/faceDetection/ --type 1
```

2.1 Question: Comment on desired or undesired results. What can you do to detect the third face in Fig. 1? Make sure you document parameters.

2.2 Video

Repeat the previous subtask 2.1 on videos. Write code to perform the following:

1. Read the path of the input video. Provided as `data/samples/arnold.mp4`.
2. For each frame in the video, draw a bounding box (red border coloured) for every frame detected. A video with face bounding boxes should be saved as `results/faceDetection/arnold.mp4`. A sample output frame from the output video is shown in Fig. 2.
3. Repeat for the videos your captured in Task01. Store videos as `data/captured/faceVideo0?.mp4`. Do not change the content, but keep the output size to 240p. The final video with face bounding boxes should be saved as `results/faceDetection/faceVideoOutput0?.mp4`.
4. Report accuracy percentage of frames with successful detection. Comment as usual as before.

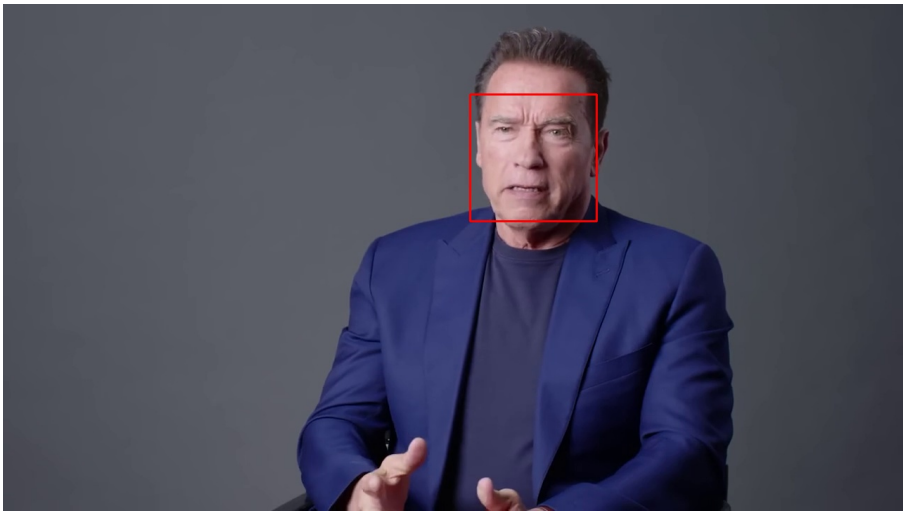

Figure 2: Output frame for face detection on sample video.

```
python face-detection.py --data path-to-video \
--faces path-to-save-video --type 2
```

```
Example: python face-detection.py --data ../data/captured/faceVideo.mp4 \
--faces results/faceDetection/ --type 2
```

2.3 Viola-Jones vs HoG

As discussed in class the *Viola Jones (VJ)* framework is a classical method which detects faces using **Haar** features. In this subtask, the goal is to compare two methods, VJ built into `opencv` vs HOG built into `face_recognition` for face detection as follows:

1. Copy (recall pickle format) the bounding box information that you had for the dataset you created in Task01 into `data/captured`. Assume these as ground truth.
2. Perform face detection on your dataset using VJ using the pre-trained classifier file `haarcascade_frontalface_alt.xml` and output the bounding boxes (recall pickle format).
3. Write `compare.py` which compares the two face detection results, the second one using the method in Section 2.1. Compare the results obtained in step 2 & 3 by using **intersection-over-union** metric. Output score should be printed on the terminal.

```
Example: python compare.py --data ../data/samples/arnold-sylvester.jpg
```

2.3 Question:

1. Why have we used **intersection-over-union** as a metric to compare results?
2. Discuss your results in a paragraph. Make sure you write pro of VJ vs con of VJ (which automatically implies con of HoG vs pro of HoG).

3 Face Recognition

After detecting the faces in an image, the next task is to recognize faces from a set of known images. This task draws motivation from facial bio-metric security systems, automated class attendance systems, etc.

3.1 Images

In this sub-task we will attempt to recognize known faces in an image containing multiple faces as follows.

1. Assume Arnold and Sylvester as known. From (`data/samples/arnold-sylvester-unknown`) detect all the faces and draw a blue bounding box around each face detected. This is similar to Sec. 2.1.
2. Annotate the detected faces with the name of recognized person and save the result in `results/faceDetection/sampleRecognized0?.jpg`. If a detected face is not known then annotate the detection as 'Unknown') (see Fig. 3).
3. Repeat the above experiment with your data and consider the two lowest roll number in your group as known and the third as unknown. In this case the query images are two: the image with all three group members, and `arnold-sylvester-unknown.jpg`. (In the latter case, the expected results are Unknown, Unknown and Unknown).

(a) Input Image (b) Annotated Image

Figure 3: Recognizing Sylvester and Arnold. The output on the right satisfies content, but not form. Thicken the lines, and make the text readable. Trim long names. Be elegant and creative.

```
python face-recognition.py -i path-to-sample-file \
-o path-to-save-recognitions --type 1 or 2
```

```
Example: python face-recognition.py \
-i ../data/samples/arnold-sylvester-unknown.jpg \
-o ../results/faceRecognition/sampleRecognized0?.jpg --type 1
```

```
Example: python face-recognition.py \
-i ../data/captured/0?.jpg \
-o ../results/faceRecognition/sampleRecognized0?.jpg --type 2
```

3.1 Question: Faces in the known dataset and those in the query image can be in different poses. Explain briefly how the system used for recognition resolves this issue by referencing the relevant function hierarchy (e.g. python stacktrace) for about 3 informative levels.

3.2 Videos

1. Read the path of the input video. Provided as `data/samples/arnold.mp4`. Consider Arnold as known.
2. For each frame, detect, recognize, and annotate faces. Save the output video in `results/faceRecognition/` as `sampleOutput.mp4`.
3. Now repeat these steps for your video (from Task01). As before consider the two lowest roll numbers as known. Save output to `results/faceRecognition/faceVideoOutput0?.mp4`.

```
python face-recognition.py \
-i path-to-sample-video-file -o path-to-save-recognitions --type 3
```

```
Example: python face-recognition.py -i ../data/samples/arnold.mp4 \
-o ../results/faceRecognition/faceVideoOutput.mp4 --type 3
```

3.3 Images with Face Masks

In recent times we have seen how masks have played an essential part in our daily lives. Our face recognition systems must be robust enough to adapt to this situation.

1. Update your dataset. Create a test image set (3 images) one for each member with face masks. Variety is expected (e.g. surgical mask, n95, color, cost etc.). Store this set as `data/captured/masked01.jpg, masked02.jpg, etc.`
2. Perform face recognition as before using the same code. Recall that only two members are considered as known.
3. Show the results as in the image section.

3.2 Question: Report your observations while performing face recognition on masked face images.

4 Fun With Images

Often times we are required to manipulate facial data in the images. In this section, you will learn to manipulate the image in order to apply makeup to the face. In order to do this, we need to identity various facial landmarks such as, eyes, eye-brows, lips, nose etc. Later, using this information we can manipulate facial features to apply makeup to faces.

1. Detect facial landmarks for the input image (`data/samples/arnold-sylvester`). Store the results in `results/drawing` as `peopleLandmarks`. Sample output can be seen in Fig. 4.
2. Now repeat the experiment on `data/samples/angelina` and using the facial landmarks detected, apply makeup to the face.
3. Save the output image with face makeup as `results/drawing/angelinaBeautiful`. Sample output can be seen in Fig. 5.

```
Example landmarks: python face-makeup.py -i ../data/samples/arnold.jpg \
-o ../results/drawing/peopleLandmarks.jpg --type 1
```

```
Example makeup: python face-makeup.py -i ../data/samples/angelina.jpg \
-o ../results/drawing/angelinaBeautiful.jpg --type 2
```

(a)

Figure 4: Features of Arnold and Sylvester highlighted.

(a) Input image

(b) Output image

Figure 5: From an artistic point of view, which one do you think will be preferred?

5 Submission Guidelines

1. The top assignment folder should look something like

```
130010009_140076001_150050001_Task02
├── ReflectionEssay.pdf
├── code
│   ├── compare.py
│   ├── face-detection.py
│   ├── face-recognition.py
│   └── face-makeup.py
├── data
│   ├── captured
│   │   ├── 01.jpg
│   │   ├── 02.jpg
│   │   ├── 03.jpg
│   │   ├── faceVideo01.mp4
│   │   ├── faceVideo02.mp4
│   │   ├── masked01.jpg
│   │   ├── masked02.jpg
│   │   └── masked03.jpg
│   └── samples
├── results
├── answers.pdf
├── convincingDirectory
└── readme.txt
```

2. Only the following Python packages are permitted.

   `face_recognition, opencv, numpy, os, sys, argparse`

   Your code should run in a virtual environment that is likely to not have any other packages.

3. Please note the following and clean up your folders before submission. Make sure to resize your images to XGA (or lower), and videos to 240p or lower. As usual, report answers to questions in `answers.pdf`.

4. Note that we expect you to think about every question, implicit or explict, and reflect upon them in your `reflectionEssay`. The primary purpose is to amplify your knowledge (student vs test-taker) since this document is not peer-graded (but is required). Reflection essay is not expected to be anonymous.

5. The results directory is EMPTY. It is simply a placeholder when the grader runs your code on `data`. Generally, the output of your program has the same file name as the input (i.e., if not explicitly mentioned otherwise), and is in the `results` directory.

6. The samples folder is EMPTY. The grader is expected to copy these files from the problem statement directory.

7. The data folder is NOT empty.

8. Anonymity must be maintained in readme.txt and answers.pdf. As is obvious, photos are not anonymous but should be included regardless as necessary.

9. Include a `readme.txt` (See this piazza post). **Note:** Now the `readme.txt` doesn't includes the student's screen name.

10. Other items: (See this piazza post).