CS 763: Task 01: Log in

- Announced: 28 Jan. Due: Feb 02 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

In this task we'll get started with some basic (i.e., not expected to be exhaustive) image processing tasks using Python 3 bindings. In addition to `opencv, numpy, os, sys`, only the following Python packages are permitted. Your code should run in a virtual environment that is likely to not have any other packages.

`pickle, argparse`

2 Bounding Boxes

Almost any deep learning technique requires a large amount of data for training. These datasets are constructed manually by annotators carefully labeling parts of the image according to the task. This subtask will help you partially understand the world of annotators.

2.1 Annotation

In this subtask, you will learn to annotate input images with rectangular bounding boxes, boxes that is expected to cover every face in the input image. Write code to perform the following:

1. Construct a dataset with faces of your group members, consisting of at least 2 images per person, and making sure all the members in your group are present in some image or the other. Do not exceed a total of 20 images. Some images will have multiple faces of all members. Make sure you have images of various sizes starting from 24x24, and resolutions of type svga, xga, wxga, HD, and ending at no more than 'Full HD resolution' (whatever that means :)

(a) Input image (b) Bounding box (c) Anonymized

Figure 1: In the rightmost picture, we cannot easily tell the identity of the people.

2.1 Question: First, list what diversity you have obtained, and next the ways in which you have obtained the variety (a line for each image).

2. Obtain the coordinates of the faces in all the input images of the directory by selecting the coordinates with mouse clicks for each image. (**Hint:** See `setMouseCallback()`.)

3. Save the coordinates for each face as a `pickle` file, a file containing bounding box information for each face in an image.

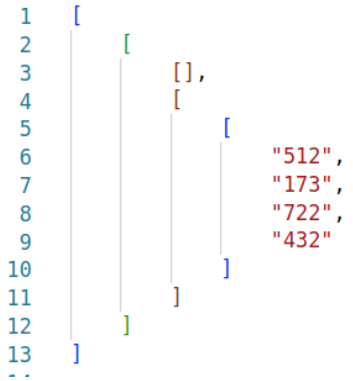

Figure 2: Example annotation pickle file format.

Our Pickle Format: If the input is a video, treat each frame as an image.

- The final pickled file will have one list which contains faces bounding box coordinates for each frame. Thus its length will be equal to the number of frames in the input.
- Each frame bounding boxes are in turn stored as a list of length 2 where the first element of that list is kept empty, and the second element is XX (see below).
- XX is another list which contains the bounding box extents, i.e., it is a list whose length is equal to the number of faces in the frame, and where each element is another list that contains geometry information of the appropriate face (top left and bottom right (u_1, v_1, u_2, v_2) OpenCV coordinate rectangle). In the example above, there is only 1 frame, and only 1 face.

A sample annotation file is provided in `data/video`.

```
python bounding-box.py --data path-to-directory-containing-images \
--annotation path-to-save-annotation --type 1
```

```
Example: python bounding-box.py --data ../data/captured/ \
--annotation ../data/captured-annotation/ --type 1
```

2.2 Drawing Boxes

In this subtask, you will read the annotation details provided to you and use them to draw rectangular boxes around (hopefully) faces present in the image.

Write code to perform the following:

1. Read the path of images and annotation files.
2. Load each image in the directory and annotate the faces with rectangular boxes.
3. Move to the next image on pressing 'n' on the keyboard and to the previous image on pressing 'p' in a circular fashion. The key 'q' should quit the program. No other bindings are expected.

```
$ python bounding-box.py --data path-to-directory-containing-images \
  --annotation path-to-annotation-folder --type 2
```

```
  Example: python bounding-box.py --data ../data/captured/ \
--annotation ../data/captured-annotation/ --type 2
```

2.3 Blur Faces

In this subtask, you will learn to hide a part of the image. For the images you have annotated already with bounding boxes for each face, hide the faces in the image using `Gaussian blur`.

Write code to apply `Gaussian blur` only within these rectangular regions. The desired visual appearance is to be able to just make out that there is a face within the rectangle, not a black box! In this regard, Fig.1(c) is an over-correcting blur.

As before the keyboard actions should work.

```
$ python bounding-box.py --data path-to-directory-containing-images \
  --annotation path-to-annotation-folder --type 3
```

```
Example: python bounding-box.py --data ../data/captured/ \
--annotation ../data/captured-annotation/ --type 3
```

2.4 Videos

You have so far experimented with annotating images with face coordinates and blurring faces in them. However, it will be useful if this can be extended to video. Prepare a video with one/two of your team members in it that spans for at least 10 seconds.

You have to write code to perform the following:

1. Load the video and annotate it with rectangular regions for faces present in the video for each frame and save it in the pickle format discussed already.

2. Load the video and `Gaussian blur` the rectangular regions specified by the pickle file.

3. Write the output to a new file with the same name in the `results` directory.

The blurred video output for the input video will be found in the **results** folder. There is no keyboard action involved and the video will thus be processed in 'batch' mode. When the type argument is passed as 4, read the video and save the annotations in the annotation path. When the type argument passed is 5, blur the faces in the video and save it to results folder.

Run example:

```
$ python bounding-box.py --data path-to-video \
  --annotation path-to-annotation-file --type 4
```

```
python bounding-box.py ../data/video/sample.mp4  \
../data/video/annotation.txt --type 4
```

3 Edge and Contour Detection

Manually detecting faces and creating annotations take a lot of time and effort. In an ideal world, we'll want to do as little of this as possible. Detecting edges and contours in an image may sometimes be useful inputs for a learning system.

3.1 Canny

1. Perform a so-called edge detection of a specified image. Use the particular method, known as "Canny" Edge Detection for this purpose (See this description). The desired output is written to the `results` folder with the name `edges`. Try getting results as close to this as possible, or better.

   (**Hint:** Try converting the image to grayscale, and then use appropriate Gaussian blur.)

2. Write the output with the same image name in the `results` folder.

```
Example: python detect-edge.py -i ../data/butterfly.jpg
```

(a) Input image

(b) Output Image

Figure 3: Observe that the gradients in the image has disappeared.

3.1 Question: Where, if at all, does `DeepFace` use ideas like these in recognizing faces? Explain briefly.

3.2 Contour

Can you we use edge detection to find the contour of a person? See example below.

1. Create a contour for the tall person (and only the tall person) in the provided image. For this task, you'll need to look into the API `findContours()`.Write the output with the same image name in the `results` folder.

2. Collect one image with more than 1 member of your team and perform the same task on that image. After clicking the image, you can post process any which way you want manually.

```
Example: python detect-contour.py -i ../data/shortvstall.jpg
```

(a) Input image (b) Output Image

Figure 4: Contour Detection

4 Submission Guidelines

1. The top assignment directory should look something like

```
130010009_140076001_150050001_Task01
├── ReflectionEssay.pdf
├── code
│   ├── bounding-box.py
│   ├── detect-contour.py
│   └── detect-edge.py
├── data
│   ├── captured
│   │   ├── img1.jpg
│   │   ├── img2.jpg
│   │   └── img3.jpg
│   ├── captured-annotation
│   │   ├── img1.txt
│   │   ├── img2.txt
│   │   └── img3.txt
│   └── video
│       ├── sample.mp4
│       └── annotation.txt
├── results
├── answers.pdf
├── convincingDirectory
└── readme.txt
```

Please note the following

(a) Include a `readme.txt` (See this piazza post).

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

(c) Final Submission. Very very important.

i. Submit on **Moodle**.

ii. The lexicographic smallest roll number in the group should submit the entire payload (with all the technical stuff).

iii. All other roll numbers who want marks should submit only their individual `readme.txt` (which ought to be different from that of the lowest roll number) and no other information.