CS 763: Task 03: Masked Training

- Announced: 02 Mar. Due: Mar 09, 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 almost a repeat of Task02. Sort of.

In Task02 you used a "cooked" `face_recognition` module that barely exposed the underpinnings of the heavy lifting. Most of you were unhappy with either the computation time involved, or the detection of people wearing masks, or, the recognition — despite the frontal face, and high quality images taken with modern cameras.

The obvious solution is to train on images with masks, if we are going to test on these kinds of images. (without using the `face_recognition` module of course).

Onward ho!

[Hint: Get started soon. Training takes time.]

2 Age and Gender

But first, let's be sure we can use a pre-trained artificial neural network with a vanilla `opencv` package without being mollycoddled with a wrapper package. In this sub task, we'll predict the (apparent) age and gender based on the facial features in the image using a pre-trained model. Obviously you cannot use the `face_recognition` module but you can use any function call in the `opencv` package. Here, roughly are the steps (but we expect you to fill in the missing steps, or ask for help). For simplicity, assume gender classification as a binary classification (M, F). Further, bucket the ages into [('child' (0-13), 'teens' [13-19], 'iitians' (19-28], 'middle' (28-40], 'boomers' (40-55], 'downhill' (55-65], 'senior' (65-75], 'super' (75-∞)].

1. (Background) Pick a pre-trained `caffe` model from this link: ethz.ch (OpenCV can digest `caffe` models).
2. (Data) (a) Use any of your previous images with three members in the group (use the name `01.jpg`) in `data/captured/q2/01.jpg` and (b) `arnold` from Task02.
3. (Process) Detect and extract the face from the input image, and feed the extracted image as input to the Deep Learning model to predict the apparent age and gender.
4. (Results) Draw a bounding box around the detected faces and annotate it with the gender and age (e.g., `M, teen`). Save the resulting image in `results/age-gender/` with the same name as the input image.

```
Example: python age-gender.py -i ../data/arnold.jpg
```

3 Training

In this subtask, we'll learn to train custom models for face detection. Training can be intense and impose unnecessary computation demands (with consequential climate change issues!!). Therefore, we will work with smaller sized toy inputs provided to you in the `data/masked` folder. (You will recognize some of these images.) We will also use the `dlib` package which as you saw was used in the `face_recognition` package. `dlib` and `opencv` are written in `C++` for efficiency considerations; you will essentially use some of the `C++` code in the `examples` directory in (link) this dlib distribution. In summary, here are the steps:

1. Background
   (a) Download using `git` the `dlib` distribution.
   (b) Compile, by following the instructions in the file `CMakeLists.txt` a single `cpp` file in the (note: This is not in `python`) `examples` directory . This compilation is best[1] done by commenting out 100 lines and keeping one line. Any file is ok, but consider `face_detection_ex.cpp`.
   (c) Run the example just to make sure you are good with the code-compile-run cycle.
2. (Detection) The computer-vision-ml cycle
   (a) Annotate the data set (see below) to prepare for training.
   (b) Modify appropriate (provided in `dlib`) `cpp` files to train.
   (c) Modify appropriate (provided in `dlib`) `cpp` files to test.
3. (Recognize) Finally, modify appropriate (provided in `dlib`) file to do face recognition on images with masks.

Report results/observations in all cases generously.

Please note this process will involve some digging around, an essential task in taking a graduate-level course. We provide hints, but we are not baby-sitting you.

3.1 Annotation

We are not telling you what to annotate but the training code requires an XML file. To create this annotated data you will need to use the `imglab` tool included with `dlib`. It is located in the `tools/imglab` folder. Let's assume you have a folder containing images called `images`. The bare minimum steps are

```
imglab -r -c training.xml images  # create xml listing images in folder

imglab training.xml  # now annotate all files listed earlier
```

A window (See Fig. 1) will appear showing all the images. Go to `help` in the menu to figure out more.

3.2 Tasks

Use `training.xml` for the steps mentioned here.

[1] We recommend this especially if you have a low power machine, with a small amount of memory.

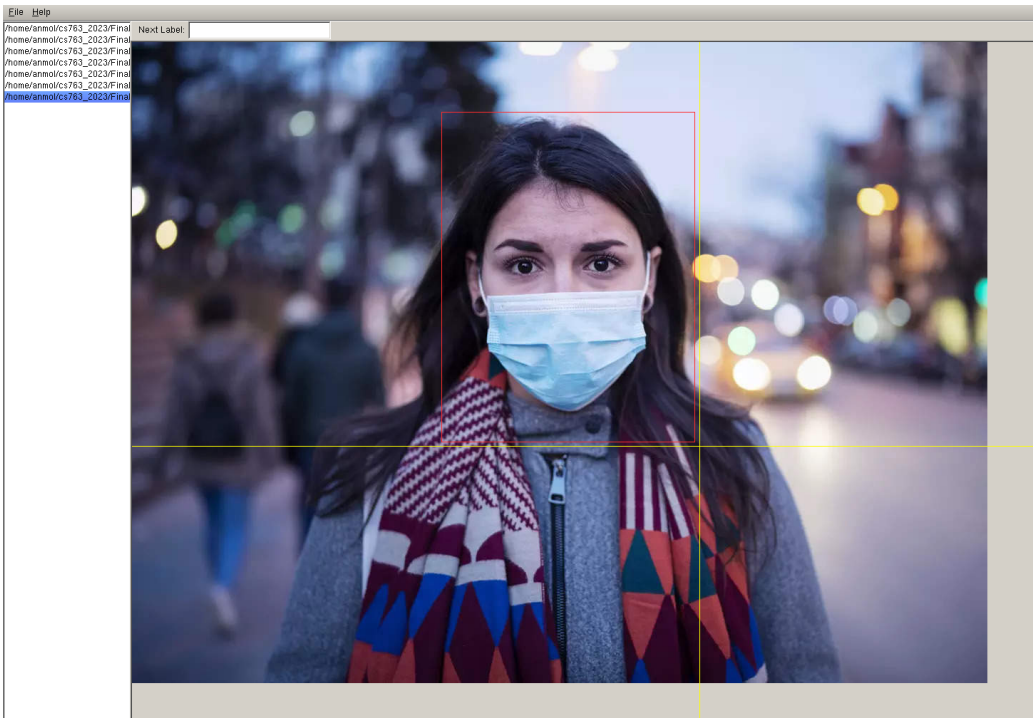


Figure 1: Using imglab for basic annotation.

1. (SVM) The face detector you used in Task 02 (`hog` option) made use of a support vector machine (SVM) to classify. Use HoG for mask-based faces using the dataset provided in `masked` folder. Save the model as `masked01.svm` in the `q3` folder. If you have multiple models, add `01` and so on (e.g. `masked02.svm`). (Why would you have multiple models? The training goal should be good to detect faces, ideally, if only the eyes are visible! See Fig. 2.)

2. (Report) Make sure you report training time, convergence rate (a graph would be nice), cross-validation parameters, and validation accuracy. Don't restrict yourself to just these things, feel free to report other observations.

3. (Test) Write code to test on any given image and write the output.

4. (ANN) The face detector you used in Task 02 (`cnn` option) made use of an artificial neural network (ANN). Use the same annotations to train a specific convolutional network given in `dlib`.

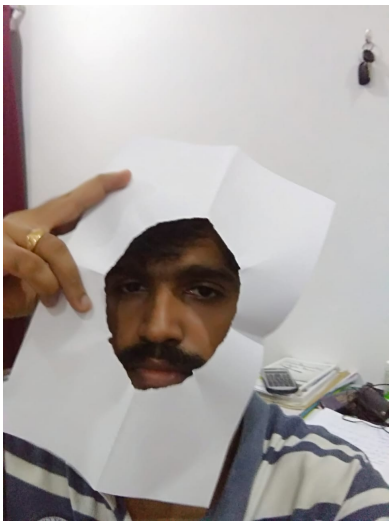
   ```
   net_type  = loss_mmod
   <con<1,6,6,1,1,rcon3<rcon3<rcon3
   <downsampler<input_rgb_image_pyramid<pyramid_down<6>>>>>>>>;
   ```

   There is nothing fancy here, it is just a less (compared to the earlier one you used) powerful ANN. However, this net illustrates an important learning viz. the connection between a generic ANN used in machine learning (ML) and an ANN used in ML for CV. It uses a pyramid structure common in CV, with other typical convolution, batch normalization and relu layers. It also uses a fast loss module `loss_mmod` for a quicker max-margin across various sliding windows (admittedly specific to `dlib`).

   Your submitted version will (eventually) save the final model as `ann.dat` (but you may want to periodically save in your local debugging session, but not submission, `temp_model_0x.dat` epoch models). You may also want to think of your last temporary saved model as your final model. See also Sec. 3.4.

(a) Input Image

(b) Detected Faces



(c) Challenge if the paper covered the mouth!

Figure 2: Typical face detection using our trained model. Thicken the lines in (b)!

5. (Report) As usual.
6. (Test) Write code to test on any given image and write output.

3.3 Synthesis

You might have noticed that the ANN model in Sec. 3.2 can be problematic. ANN is thirsty for data. It is common in computer vision to generate data.

1. Use `imglab` to annotate your mask. See Fig. 3. Save your annotations to `landmarks.xml`.
2. Write code `create.py` to lineate and fill (using only `opencv`). See Fig. 4. Do not upload your images in your submission, but do write code to process all images in `landmarks.xml` and save them in `data/captured/synthesis` folder. Take a rgb color parameter for the filling in your code.
3. (Shape Predictor) Now that you have a few images, modify appropriate (provided in `dlib`) `cpp` file to train a shape predictor.
4. (Report) As usual on the training process.
5. (Shape Creation) Run the shape predictor on un-annotated images to get landmarks for each such image.
6. Loop above steps to synthesize more images with masks.
7. Re-train your ANN in Sec. 3.2
8. (Report) As usual.
9. (Test) Write code to test on any given image and write output.

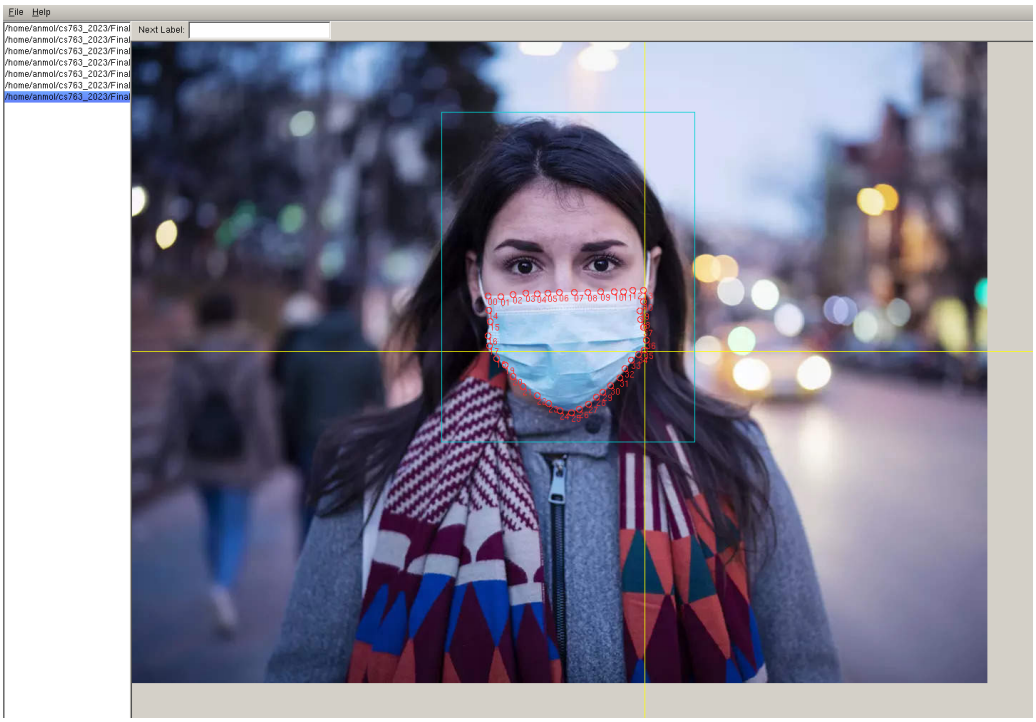


Figure 3: Using imglab for advanced annotations. Compare with the Fig. 1

.

3.4 Recognize

Finally write code `recognize.cpp` to perform face recognition as in Sec.3 of Task02 for masked images. Follow the steps mentioned there. Make sure you use both the ANN and SVM models.

4 Submission Guidelines

1. Only the following Python packages are permitted.

   `opencv, numpy, os, sys, argparse`

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

2. Please clean up your folders before submission.

3. As usual, report answers to questions in `answers.pdf`. Use section names such as report-svm.pdf, report-ann01.pdf, report-shape.pdf, and report-ann02.pdf to delineate your various answers.

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

(a) Lines

(b) Filled

Figure 4: Lineate, fill and generate images.

7. The data folder is NOT empty.
8. Anonymity must be maintained only in `readme.txt`
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).
11. The top assignment folder should look something like

```
130010009_140076001_150050001_Task0X
├── ReflectionEssay.pdf
├── code
│   ├── age-gender-detect.py
│   ├── train-svm.cpp
│   ├── train-dnn.cpp
│   ├── train-shape.cpp
│   ├── detect-svm.cpp
│   ├── detect-dnn.cpp
│   ├── detect-shape.cpp
│   ├── CMakeLists.txt
│   ├── create.py
│   └── recognize.cpp
├── data
│   ├── masked
│   └── captured
│       ├── q2
│       │   └── 01.jpg
│       └── q3
│           ├── synthesis
│           ├── training.xml
│           └── landmarks.xml
├── results
├── answers.pdf
├── convincingDirectory
└── readme.txt
```