OPENFADER - A SIMPLE GUI FOR FACE RECOGNITION

Gianluca Boleto - Gianluca Sommariva
University of Genoa

Overview

OPENFADER (which stands for OPEN source FAce Detection, Expression and Recognition) is an interactive graphical user interface written in Python [6] that allows user to perform three different algorithms, via webcam or by selecting a source input, that are precisely Face Detection, Face Recognition and Facial Expression (or Emotion) Recognition.

Model

Since both recognition problems take advantage of detection, Face Detection is the first part of our pipeline. We have used the python library *FER* [8] that we found very accurate in detecting faces. This library scans the input image and returns the bounding box coordinates of all detected faces using the ***Histogram of Oriented Gradients (HOG)*** [1], that is, turning an image into black and white and then breaking it down into squares of 16x16 pixels each. Then every square will be replaced by an arrow with the same direction as that which the greatest number of single gradients were pointing. Once this is done all that remains is to find the part of the image that most closely resembles a previously encoded HOG pattern that schematically represent the average structure of any human face (shown in Fig. 1).

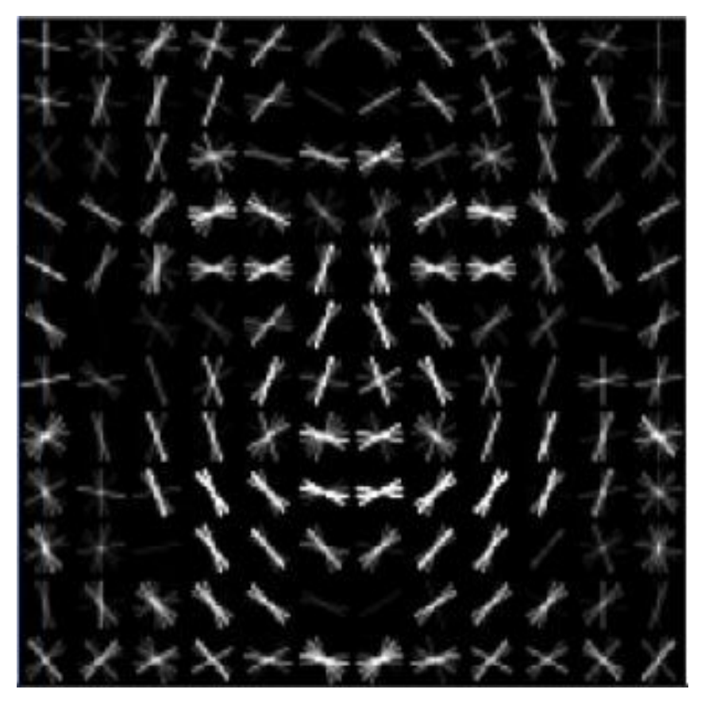


Fig. 1: HOG pattern

For the recognition problem, we used the *face-recognition* [3] library. For this purpose (that is to compare features of the input face with those saved in our database in order to obtain a match) this library uses an algorithm called ***Face Landmarks Estimation*** [4]. The basic idea is that on each face there are 68 specific points called points of interest, or landmarks (the top of the chin, the outer edge of each eye, the inner edge of each eyebrow, etc..). Once that is done, the function provides an *affine transformation* to center the eyes and mouth in the same position at the center of the image, in order to preserve the parallelism between the lines. Based on this, the algorithm performs 128 measurements, called ***Embeddings*** [7], with the aim of easily comparing two images containing faces. These measurements are fortunately performed by a *TensorFlow* [9] pre trained ***convolutional neural network***. To develop such a precise tool, each image containing a face was compared with two other images (as shown in Fig. 2), one containing the face of the same person and one with a different one, in order to slightly modify the weight on each node and therefore to have better precision.

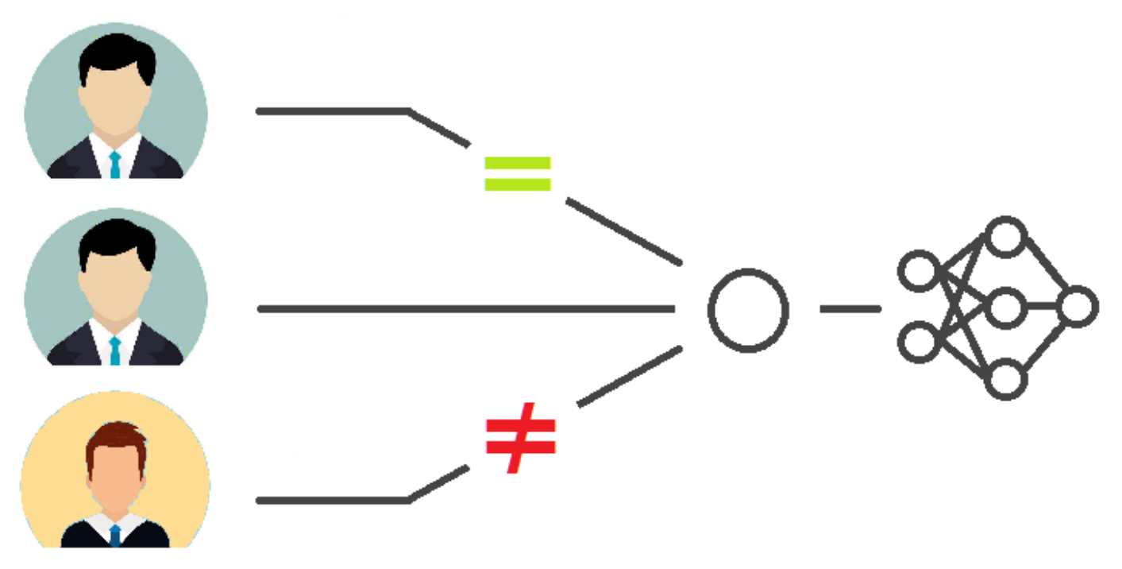


Fig. 2: CNN training

At last, for facial emotion recognition problem, we exploited once again the FER a convolutional neural network. The result of the algorithm is the most high percentage value match between those that are defined the ***Universal facial expressions of emotion*** [2]: disgust, sadness, happiness,fear,anger and surprise.

Graphical User Interface

Once the detection and the two recognition problem were implemented, the next step was to develop an interactive graphical user interface to allow user to choose the source and the algorithm to perform. Since the model was implemented in Python, we decided to design gui with *Tkinter* [10] library, due to its documentation and minimal content. First of all, we needed to view the source options with which to run our model. Below the setup window:

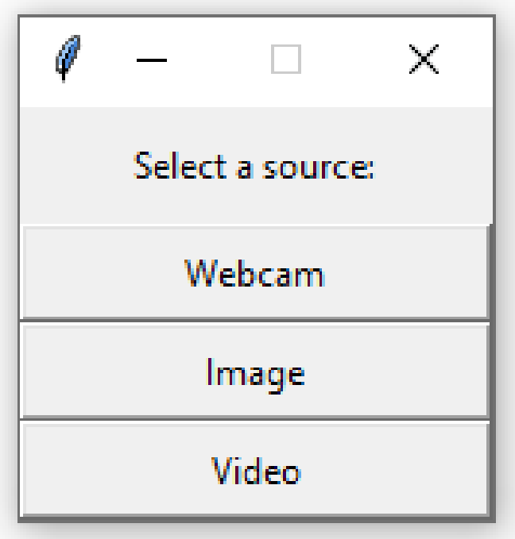
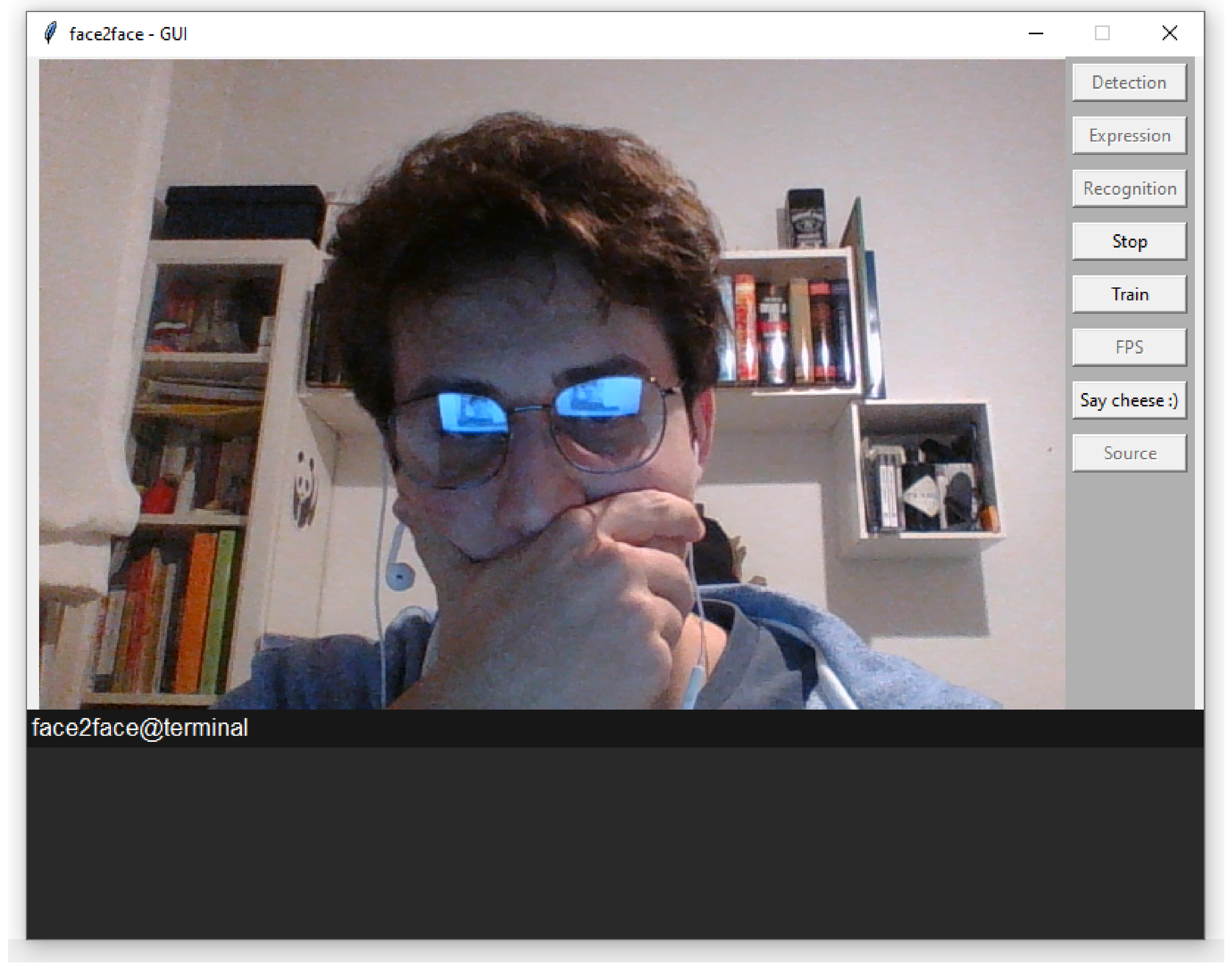


Fig. 3: Gui allows to select the source

So, after the setup window (Fig. 3), our program starts automatically the gui (Fig. 4), that was designed as follows:



```
face2face@terminal
```

Fig. 4: OpenFader GUI

The top section was divided between a "Source Box", where the source selected (or webcam, image, video) is located and a "Pushbutton List", that allows user to select different actions, combined with an "Only-Output Terminal" (the bottom section). These action are:

- **Detect** - as the name suggests, it runs the face detection algorithm, displaying (if any) a box around the detected face.
- **Expression** - as the name suggests, it runs the face expression algorithm, displaying (if any) the most high percentage value match on the terminal. While the process is running, any changes to the source can actually change the result on the terminal in real time.
- **Recognition** - as the name suggests, it runs the face recognition algorithm, displaying (if any) a "polaroid-similar" box around the detected face, with the name (if matched) on the white space below the box.
- **Source** - as the name suggests, it runs again the setup window, to allow the user to change the source without closing the gui and then rerun the program.
- **Train** - it allows to browse and save a media in db. By then, it will be possible to do a recognition analysis with regards to this new trained sample.

Depending on setup choice, other additional buttons are present in the interface:

- **Say cheese** (only for Webcam mode) - as the name suggests, it instantly takes a picture via webcam in order to add to database a new profile (photo id and full name); thanks to this, later it is possible to successfully use the face recognition mode.
- **Browse** (only for Image and Video mode) - as the name suggests, it allows user to browse and select the path of the source it intends to analyze. Once the image (or video) is displayed in the "Source Box", user can perform any algorithm on it, as described before.

Remarks

Two were the main problems encountered during the application development; first of all, once both model and gui were developed and then merged together, there was a lag trying to run each "face-something" algorithm due to Tkinter buttons interrupting all other activity. So it was necessary to manage the threads and by doing this, adding a new **Stop** button, to stop the running algorithm without starting a new one during the execution of the same tkinter button command. And at last, once this was fixed, we opted for the development of a button to manage the **Fps** (frame per second) video, in order to adapt the gui to different kind of laptop (or environment) in which it will be executed.

References

[1] N. Dalal and B. Triggs. “Histograms of oriented gradients for human detection”. In: *IEEE* (2005).

[2] P. Ekman. “Universal Facial Expression of Emotion”. In: *California Mental Health Research Digest* (1970).

[3] Adam Geitgey. *Face Recognition from Python.* URL: `https://pypi.org/project/face-recognition`.

[4] V. Kazemi and J. Sullivan. “One Millisecond Face Alignment with an Ensemble of Regression Trees”. In: *IEEE* (2014).

[5] *Keras.* URL: `https://keras.io`.

[6] *Python.* URL: `https://www.python.org`.

[7] F. Schroff, D. Kalenichenko, and J. Philbin. “FaceNet: A Unified Embedding for Face Recognition and Clustering”. In: *IEEE* (2015).

[8] Justin Shenk. *Facial Expression Recognition from Python.* URL: `https://pypi.org/project/fer`.

[9] *Tensorflow.* URL: `https://www.tensorflow.org`.

[10] *Tkinter from Python.* URL: `https://docs.python.org/3/library/tkinter.html`.