General Installation Information - Windows

1. Microsoft Visual Studio 2015 or newer: (https://visualstudio.microsoft.com/downloads/)
2. CMake for Windows: https://github.com/Kitware/CMake/releases/download/v3.16.2/cmake-3.16.2-win64-x64.msi (other options available at: https://cmake.org/download/). During installation check to add CMake to PATH for all users (leave everything else default).
3. Restart Windows (simply starting new cmd window so it will include changes in PATH should be enough, restart is a safer option though).
4. In cmd/powershell: pip install face_recognition

Implementation Logic

The program uses a combination of **Haar Cascade** (haarcascade_frontalface_default) and **face_recognition** package to identify the location of the face in a frame and match it with an encoding array containing 128 elements that correspond to face features.

The entire application consists of the following files:

1. **main.py** – This is to be run by the user first, to begin execution. The remaining flow of control is quite self-explanatory from the instructions provided in the terminal window.
2. **captureFace.py** – This program trains the model by capturing the user's face and calculating its corresponding encoding data, from photos taken by the user from their webcam's output.
3. **recogniseFace.py** – This program compares the face in the real-time webcam output with the encoding data previously saved. If a match is found, the label of the face is printed under the bounding box drawn around the face.
4. **haarcascade_frontalface_default.xml** – This is the Haar Cascade data file that is used to detect the location of a frontal face.

The **main.py** program provides the following options:

1. Train Model – Capture face data from webcam.
2. Test Model – Test the trained model to check if a face is detected.
3. Delete trained data – Delete the pickle files (only) that contain previously calculated encoding data.
4. Train from directory – Iterate through every folder in known_faces directory to train the model with the pictures saved there.
5. Remove directory – Delete the directory known_faces, and the subfolders inside it containing all the (photos of) faces captured.
6. Exit – Terminate program execution.

A known_faces folder will be created in the root directory. Inside this, there will be sub-folders with names that correspond to the labels of faces previously saved. Inside these sub-folders there will be photos of the person, whose face label is the name of the subfolder itself. For e.g., known_faces/X_Æ_A-12/0.jpg, 1.jpg, 2.jpg ... will be how the photos of X_Æ_A-12 are saved.

The encoding data is calculated and stored in pickle files: **faces** and **names** so that every time the program runs, the photos need not be re-iterated to calculate the encoding values again.

The face location is determined by Haar Cascade as this was the fastest option (favouring performance). Other alternatives were CNN (Convolutional Neural Network) or HOG (Histogram of Oriented Gradients) which gave better, but slower results.

While capturing faces in **captureFace.py**, every time the user takes a picture, the face location is found and the encoding data is calculated and stored in a list. If a face is not captured, the user will be prompted about the same, and that frame will not be saved.

Inside **recogniseFace.py**, the compare_faces function in the face_recognition package compares the encoding values of the real-time input from the webcam with the elements of the list that were previously stored from **captureFace.py**, with a certain tolerance value, say 't', set as a parameter, where $0 \le t \le 1$ (0 - strict, 1 - relaxed). It draws a yellow bounding box around the face that matches and the matched label is printed under it.