FPT Education

FPT UNIVERSITY

FPT UNIVERSITY HCMC

Face Recognition and Tracking in Video

Computer Vision - SU24

Instructor: Mr. Huynh Van Thong

Student conducted:

Dao Khang— SE183427

Huynh Khanh Quang— SE183777

Table of Contents

List of Figures

1 Introduction

Face recognition and tracking have become integral components of many modern applications, from security systems to social media filters. These technologies enable computers to detect, identify, and follow human faces within a video stream, providing a foundation for various advanced functionalities.

1.1 Face Recognition

Face recognition involves identifying or verifying a person from a digital image or a video frame. The process typically includes several key steps:

- **Face Detection**: The initial step where the system locates faces within an image or video frame. This is usually done using algorithms like the Viola-Jones object detection framework or more advanced deep learning models like the ones based on Convolutional Neural Networks (CNNs).
- **Feature Extraction**: Once a face is detected, the system extracts distinguishing features from it. This might involve identifying key facial landmarks (eyes, nose, mouth) or using deep learning models to encode the facial features into a vector representation.
- **Face Recognition**: The extracted features are compared against a database of known faces. This comparison can be done using various techniques.
- **Verification and Identification**: Verification checks if the face matches a claimed identity, while identification determines who the person is from a set of known individuals.

1.2 Face Tracking

Face tracking goes a step further by continuously monitoring the position and movement of detected faces in a video stream. This process includes:

- **Security and Surveillance**: Identifying and tracking individuals in public spaces or restricted areas.
- **Authentication**: Unlocking devices or granting access to services based on face recognition.
- **Human-Computer Interaction**: Enhancing user experience in gaming, virtual reality, and social media through real-time face tracking.

1.3 Applications

Face recognition and tracking are used in various applications, such as:

- **Detection in Each Frame**: Similar to face detection, but performed on a continuous sequence of frames.
- **Association**: Linking detected faces across frames to ensure the system is tracking the same individual over time. This is often done using algorithms like the Kalman filter, particle filter, or deep learning-based trackers.
- **Handling Occlusions and Motion**: Effective face tracking systems can handle partial occlusions (when part of the face is covered) and variations in lighting, expression, and angles.

1.4 Challenges

Despite significant advancements, face recognition and tracking still face several challenges:

- **Variability in Appearance**: Differences in lighting, pose, expression, and occlusion can affect accuracy.
- **Scalability**: Managing large databases of faces and real-time processing in high-resolution videos requires substantial computational resources.
- **Privacy Concerns**: Ethical and legal considerations around the collection and use of biometric data.

2 Scope and Problem Definition

2.1 Scope

The scope of face recognition and tracking in video encompasses a broad range of areas, each with its own set of challenges and applications. This includes:

- **Face Detection**: Developing algorithms that can accurately and efficiently detect faces in varying conditions, such as different lighting, angles, and occlusions.
- **Feature Extraction and Recognition**: Creating robust methods for extracting distinctive features from faces and matching them against a database for recognition, accounting for variations in appearance due to aging, makeup, and facial hair.
- **Face Tracking**: Implementing techniques to continuously follow detected faces across multiple frames in a video, handling motion, occlusions, and changes in orientation.
- **Integration and Application Developmen**: Building complete systems that integrate face detection, recognition, and tracking into practical applications such as security systems, biometric authentication, and user experience enhancement tools.

2.2 Problem Definition

Face recognition and tracking in video face several key challenges and problems that need to be addressed to improve their effectiveness and reliability:

- **Variability in Conditions**:
  +) **Lighting Variations**: Changes in lighting conditions can significantly affect the accuracy of face detection and recognition.
  +) **Pose and Angle**: Faces appear different from various angles, making consistent recognition difficult.
  +) **Occlusions**: Objects such as glasses, masks, or other faces can partially obscure faces, complicating detection and tracking.

- **Scalability and Performance**:
  +) **Real-time Processing**: Achieving real-time performance while maintaining high accuracy requires significant computational power and efficient algorithms.
  +) **Large Databases**: Managing and searching through large databases of faces quickly and accurately is a challenging task.

- **Accuracy and Reliability**:
  +) **False Positives and Negatives**: Minimizing incorrect identifications and missed detections is crucial for reliable performance.
  +) **Feature Distinction**: Extracting features that are distinctive yet robust to variations in appearance.

- **Ethical and Privacy Issues**:
  +) **Data Security**: Ensuring the security of stored facial data to prevent unauthorized access and misuse.
  +) **Consent and Transparency**: Addressing concerns about the use of facial recognition technology without explicit consent and ensuring transparency in its use.

- **Integration with Existing Systems**:
  +) **Data Security**: Ensuring the security of stored facial data to prevent unauthorized access and misuse.
  +) **Consent and Transparency**: Addressing concerns about the use of facial recognition technology without explicit consent and ensuring transparency in its use.

3 Method

3.1 List of functions

3.1.1 def get_samples

- **Purpose**: Captures images of the user's face and stores them in a specified directory for training.
- **Modules Used**: cv2, os
  - **cv2.VideoCapture()**: Captures video from the webcam.
  - **cv2.flip()**: Flips an image around the specified axis.
  - **cv2.imshow()**: Displays an image in a window.
  - **cv2.rectangle()**: Draws a rectangle on an image.
  - **cv2.imwrite()**: Saves an image to a specified file.
  - **os.makedirs()**: Creates a directory.

3.1.2 def recognition

- **Purpose**: Trains the face recognizer with the captured face samples.
- **Modules Used**: cv2, os, numpy
  - **cv2.face.LBPHFaceRecognizer_create()**: Creates an LBPH face recognizer.
  - **cv2.CascadeClassifier()**: Loads the Haar cascade file for face detection.
  - **cv2.imread()**: Reads an image from a file.
  - **os.listdir()**: Returns a list containing the names of the entries in the directory.
  - **numpy.array()**: Creates an array.
  - **numpy.uint8()**: Converts data to 8-bit unsigned integers.

3.1.3 def recognize_in_video

- **Purpose**: Recognizes faces in real-time from webcam feed.
- **Modules Used**: cv2, os
  - **cv2.VideoCapture()**: Captures video from the webcam.
  - **cv2.face.LBPHFaceRecognizer_create()**: Creates an LBPH face recognizer.
  - **cv2.CascadeClassifier()**: Loads the Haar cascade file for face detection.
  - **cv2.cvtColor()**: Converts an image from one color space to another.
  - **cv2.rectangle()**: Draws a rectangle on an image.
  - **cv2.putText()**: Puts text on an image.
  - **os.path.exists()**: Returns True if path refers to an existing path or open file descriptor.

3.1.4 def update_position

- **Purpose**: Handles the event when the trackbar value is changed.
- **Modules Used**: cv2
  - **cv2.getTrackbarPos()**: Gets the current position of the trackbar.
  - **cv2.setTrackbarPos()**: Sets the position of the trackbar.

3.1.5 def get_samples_display

- **Purpose**: GUI function to capture face samples from the user.
- **Modules Used**: tkinter
  - **tkinter.messagebox.showerror()**: Displays an error message box.
  - **tkinter.Entry.get()**: Retrieves the text from an entry widget.

3.1.6 def recognition_display

- **Purpose**: GUI function to start the training of the face recognizer.
- **Modules Used**: tkinter
  - **tkinter.messagebox.showinfo()**: Displays an informational message box.

3.1.7 def recognize_in_video_display

- **Purpose**: GUI function to start face recognition from the webcam.
- **Modules Used**: tkinter
  - **tkinter.messagebox.showinfo()**: Displays an informational message box.

3.1.8 main

- **Purpose**: Sets up and starts the main GUI application.
- **Modules Used**: tkinter
  - **tkinter.Tk()**: Creates the main window.
  - **tkinter.Frame()**: Creates a frame widget.
  - **tkinter.Label()**: Creates a label widget.
  - **tkinter.Entry()**: Creates an entry widget.
  - **tkinter.Button()**: Creates a button widget.
  - **tkinter.Tk.mainloop()**: Starts the Tkinter event loop.

3.2 Algorithm

3.2.1 Haar Cascade Classifier

- **Definition**: Essentially, the algorithm uses Haar-like features and applies them through multiple stages (cascade) to create a robust recognition system.

- **How It Works**:
  +) Haar-like features are simple patterns used to detect contrasts in image intensity. They include edge features, line features, and four-rectangle features.

  +) Using these features, we can calculate the values of the Haar-Like feature as the difference between the sum of the pixels of the black and white regions as in the following formula:

  $$f(x) = \text{Sum of black region} - \text{Sum of white region} \tag{1}$$

  +) When calculating Haar-Like features, there will be pixel regions that are repeated many times. Therefore, to optimise the process, we apply the Integral Image. Viola and Jones introduced the concept of the Integral Image, which is a 2D array with the same dimensions as the image for which the Haar-Like features are to be calculated. Each element of this array is computed by summing the pixel values above (row-1) and to the left (column-1) of it.

  +) AdaBoost: Operates on the principle of combining weak classifiers linearly to form a strong classifier by weighting the weak classifiers.

  +) Cascade: The AdaBoost as described above can only capture basic features. To recognize a face, we need around 6000 such features, and Cascade helps optimise this process. The Cascade is designed as follows: Among the 6000+ features, they are divided into many stages. Each time the sliding window moves over a region of the image, each stage is processed sequentially: if stage 1 detects a face, we move on to stage 2; if not, we discard that region and slide the window to a different position. If a region passes all stages of face detection, then that window contains a face.

3.2.2 LBPH (Local Binary Patterns Histogram) Face Recognizer

- **Definition**: LBPHFaceRecognizer is a face recognition method based on the analysis of local binary patterns of facial images. It is one of the simplest and most effective facial recognition methods.

4 Results

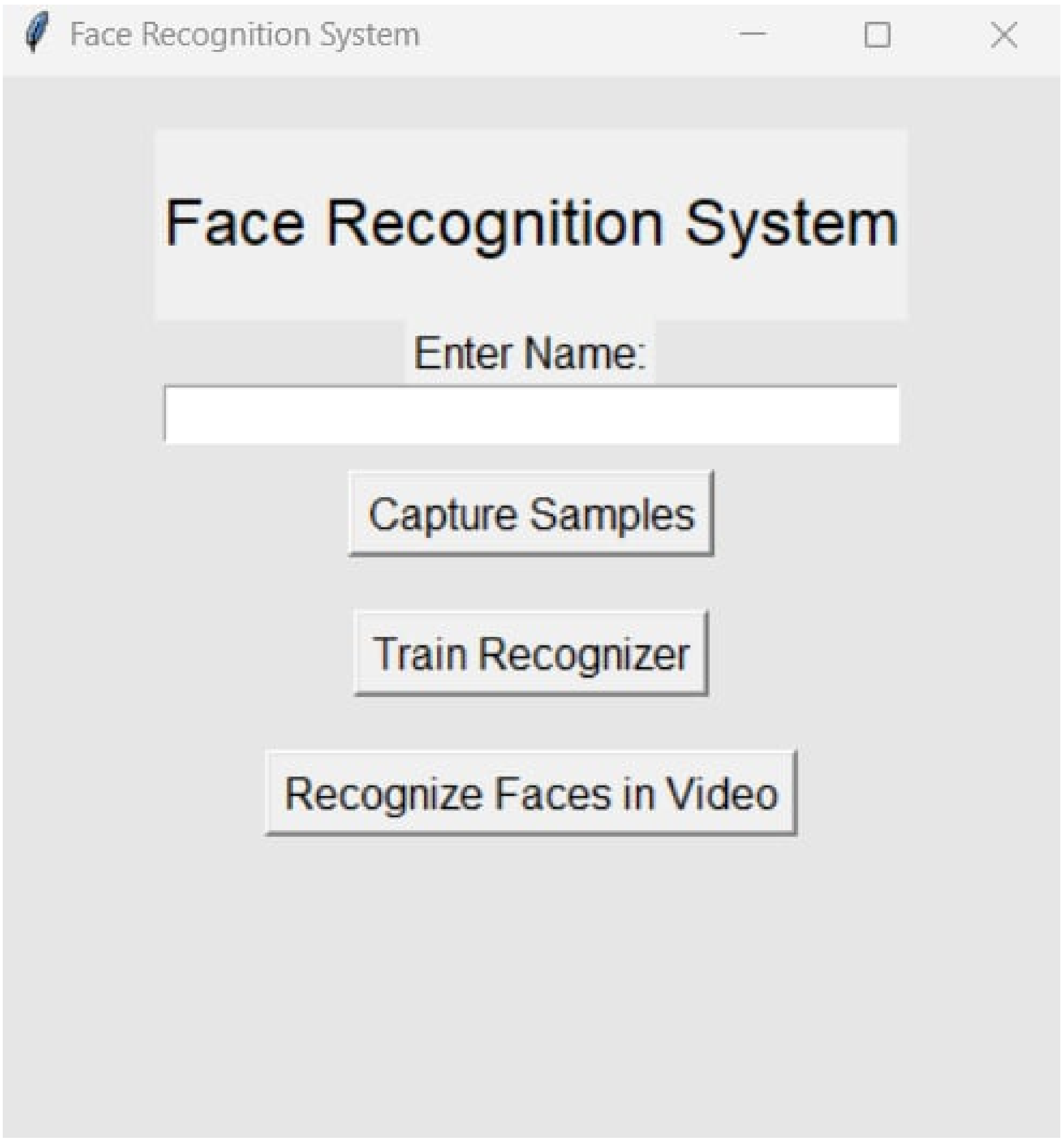


Figure 1: GUI

- Graphical User Interface

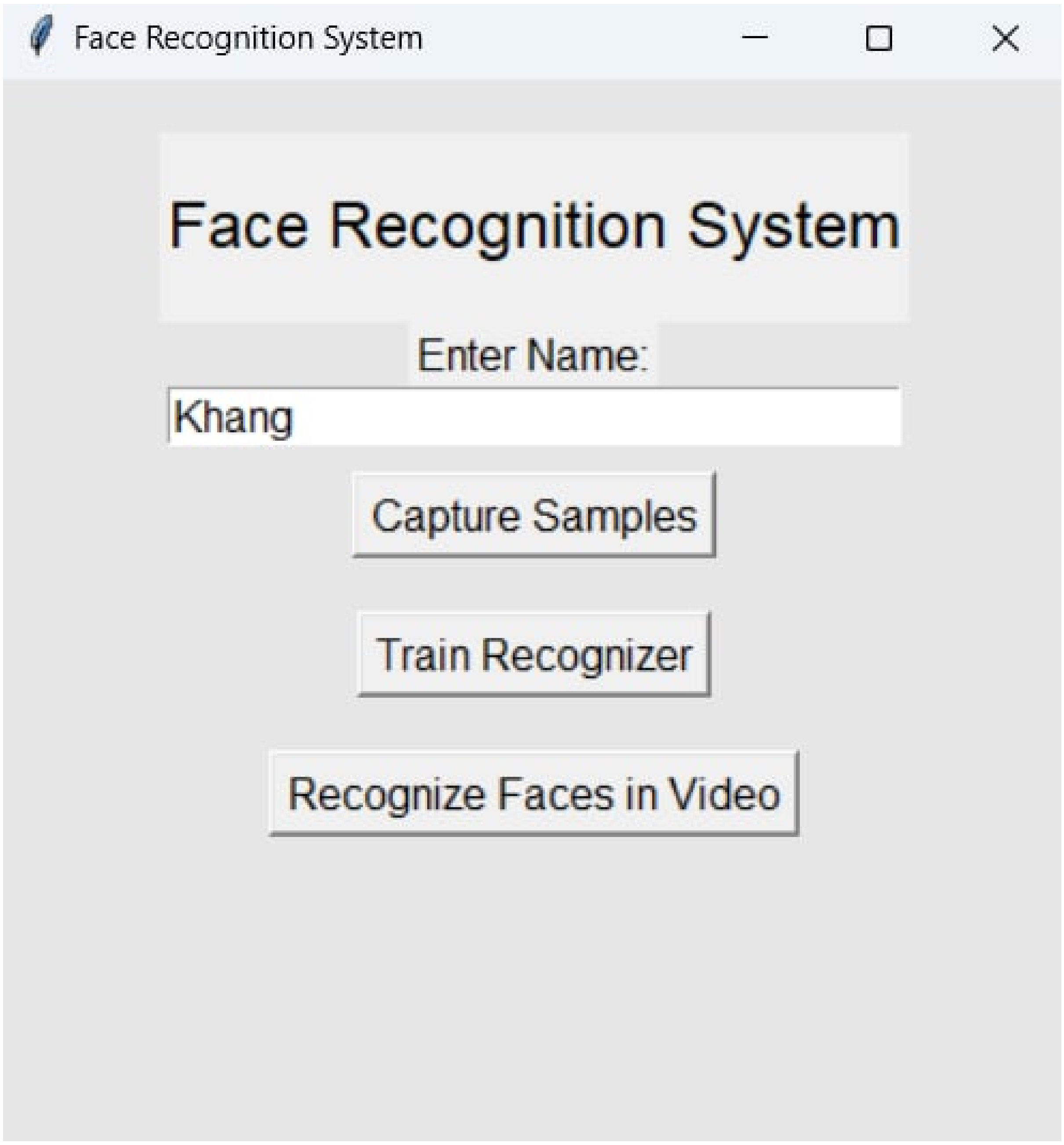


Figure 2: Fill the name and ready to Capture Samples

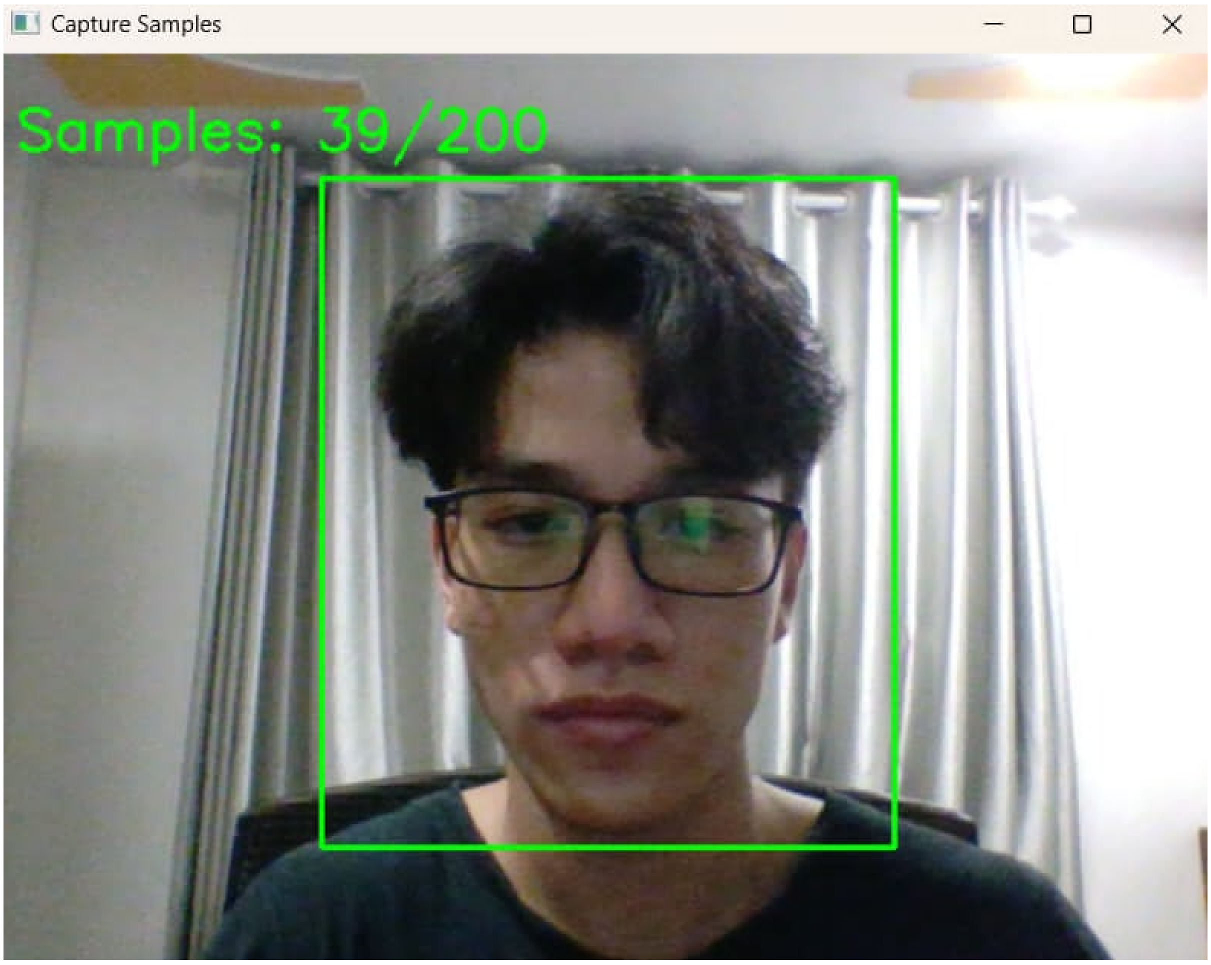
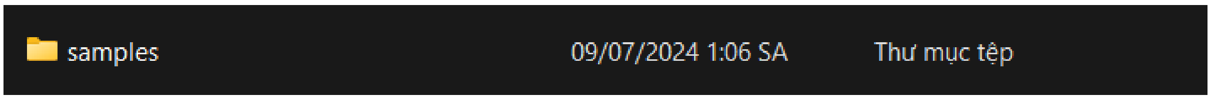


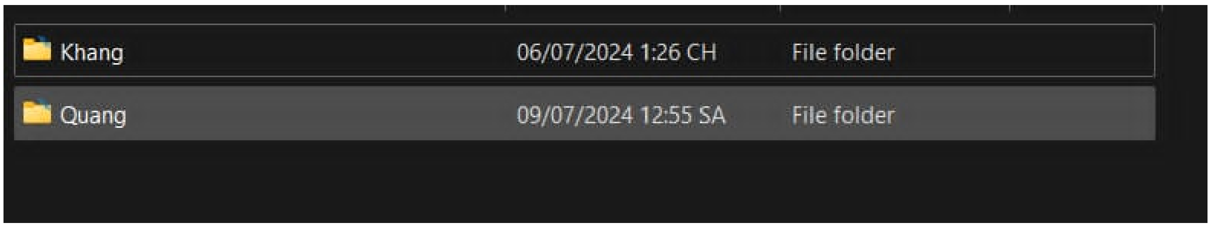
Figure 3: Capture Samples and Start generating Dataset

- press and hold 'c' to start Capture Samples
- press 'q' to quit Capture Samples



samples 09/07/2024 1:06 SA Thư mục tệp

Figure 4: Datasets's folders



Khang	06/07/2024 1:26 CH	File folder
Quang	09/07/2024 12:55 SA	File folder

Figure 5: Samples's folders

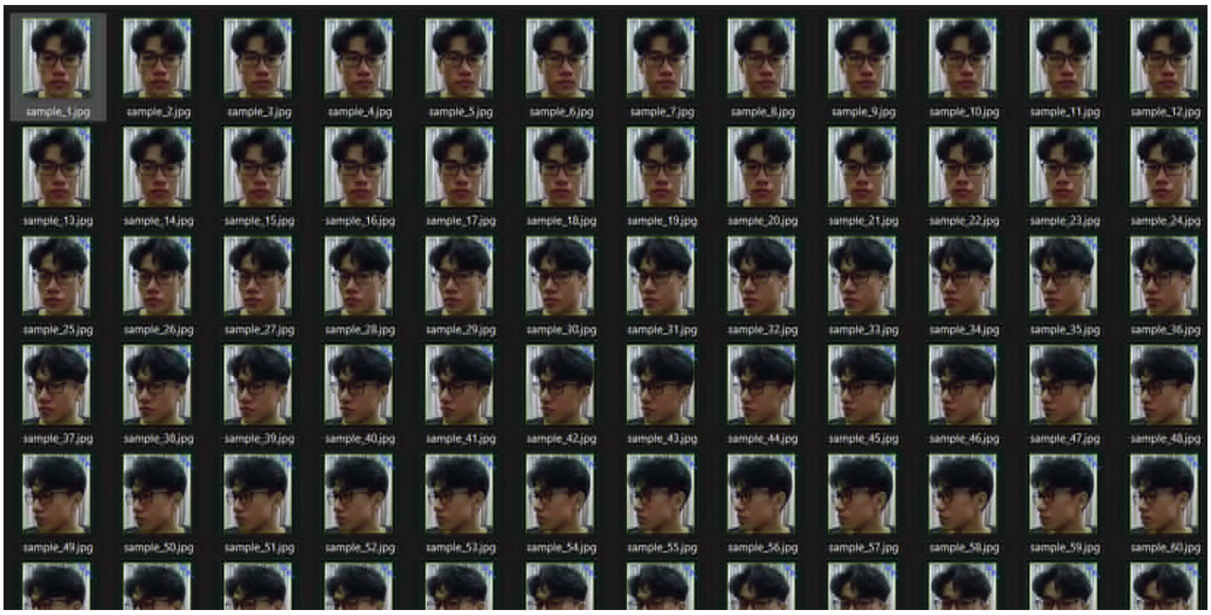


Figure 6: Khang's folders

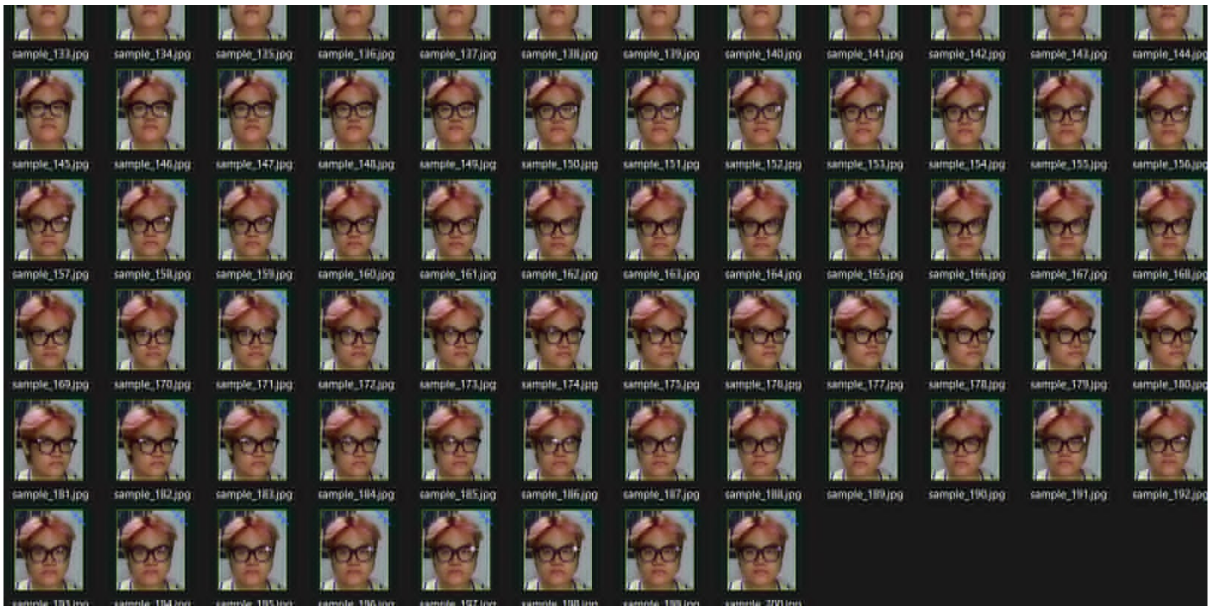


Figure 7: Quang's folders

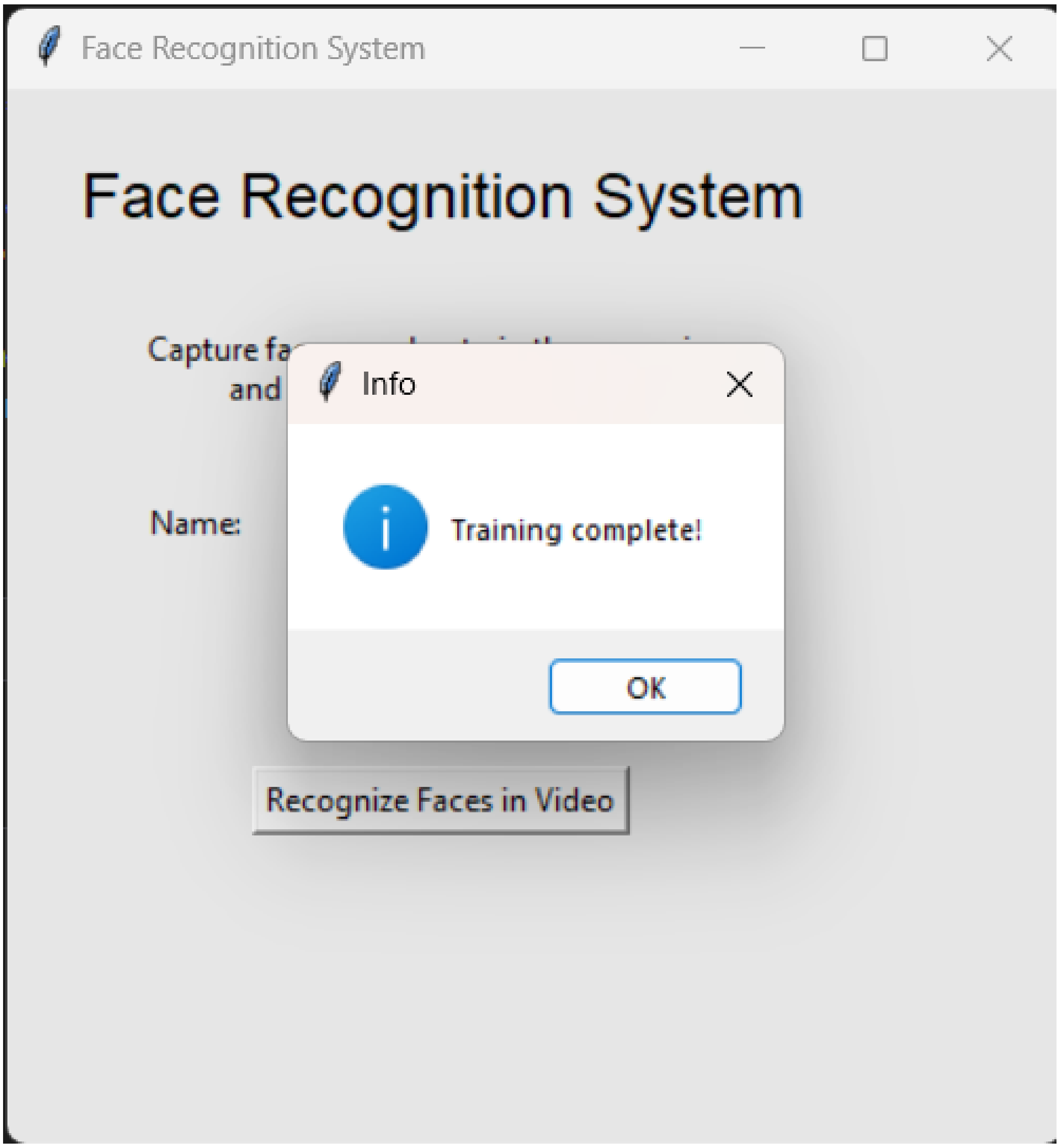


Figure 8: Training completed

- A box shows up to notify that the training datasets was completed!

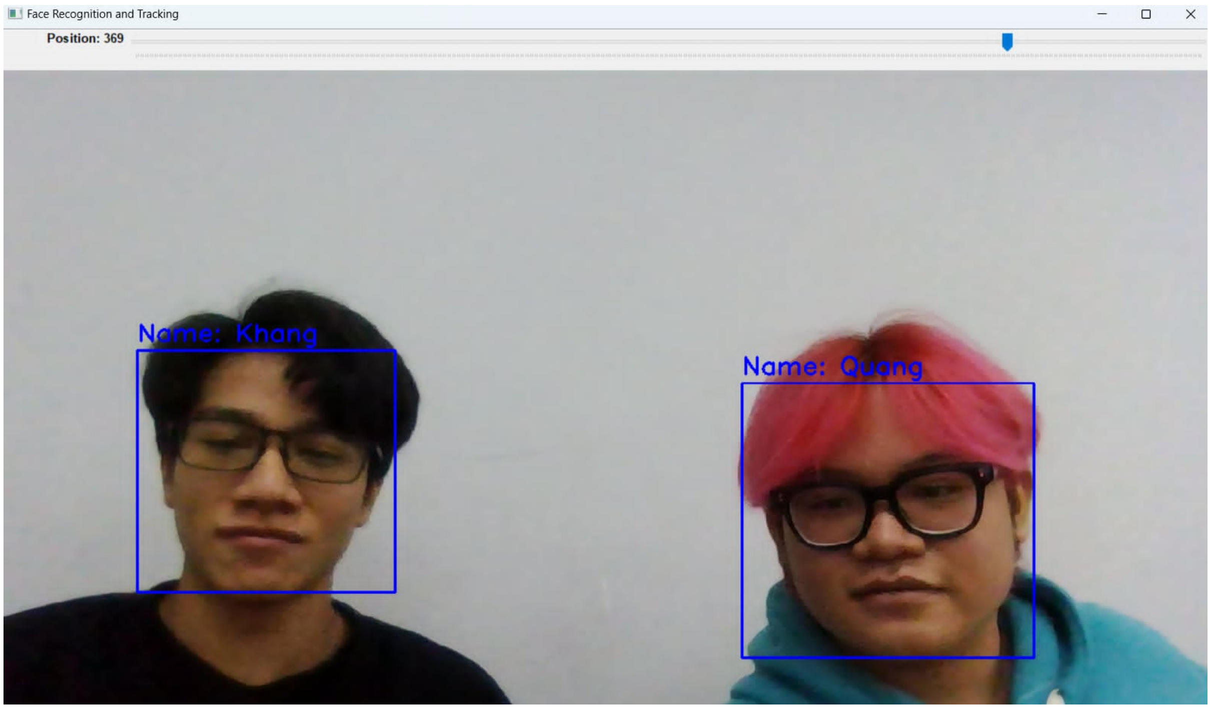


Figure 9: Result

5 Task Table

Member	Job	Percent Task
Dao Khang	slide, code, content, report	100%
Huynh Khanh Quang	slide, code, content, report	100%

Table 1: Task Table

6 Code

Code available at:
`https://github.com/quanggne/Facial-Recognition-and-Tracking-Video`