Applied Machine Learning *(CSAI2016P)*

Final Machine Learning Project Report

Presented by GROUP-1 | Group Name - *'Singers'*

Facial Recognition System
Supervised Learning Approach

Using `OpenCV`'s `Haar Cascade Classifier` and `LBPH Face Recognizer` Models

Access Project Files Online - https://github.com/dedrituz/Face_Recognition_Python_sem4

Group Members:
Agamvir Singh Gill
Sakshi Tewari
Siddhant Gupta

Introduction - Face Recognition System

This project illustrates a functional and modular face recognition system as an example of simple supervised learning principles. We live in a period of increasing reliance on intelligent automation. The ability of a machine to identify individuals accurately and efficiently is a priceless utility and holds significant value across diverse applications, from security access control to personalised human-computer interactions.

Here, we showcase a system built using accessible open-source tools and libraries, demonstrating core concepts of supervised learning through registering individuals, training a model on their facial features, and subsequently recognising them in real-time.

We can use this to provide a clear and intuitive understanding of how labelled data drives the learning process in supervised machine learning, enabling accurate classification and prediction tasks in a real-world scenario. This example serves as a tangible way to understand the power and potential of supervised learning techniques.

Problem Statement - Looks Matter

In security and access control, face recognition offers a robust and non-intrusive method for identity verification - most of us tend to have facial data accessible easily, and we do not need expensive sensors to verify unique features. This streamlines entry processes for authorised personnel and bolsters security against unauthorised access in sensitive areas, as a simple example. “Sensitive areas” here may refer to corporate offices, data centres, airports, government facilities, and may extend to more intangible ‘areas’, like our phones, etc.

Another example occurs in law enforcement and public safety, where face recognition helps identify suspects, missing people, and potential threats, accelerating investigations and overall contributing to safer communities.

Security is not the only field where this technology shows its versatility - It can also enhance user experiences by making specific repetitive processes faster and pain-free. In the retail sector, it enables personalised marketing, advertising, and payment systems. Healthcare can use these methods to identify patients to ensure accurate record-keeping and medication administration. Even educational institutions can benefit - attendance taking is made easier, and secure examination protocols are aided using face recognition. We see the latter more commonly in competitive/entrance examinations.

The face recognition ability of many devices and software will provide swift, accurate, and often contactless identification in order to solve complex problems and create more efficient and secure environments across multiple domains, as the technology matures and improves.

Methodology - Design and Algorithm

We aim to teach the system to recognise people using their labelled faces during offline training. This new knowledge is then used to identify unseen faces in real time.

Basic design stages:

- **Registration** - data acquisition and preparation
  Collect and organize labeled facial images for each distinct user.
- **Offline learning** - model training
  Learn the unique facial features of users based on the collected data for all users.
- **Real-time operation** - face detection and recognition
  Identify individuals from a live video stream utilising the trained model from the previous phase.

Key features of the application's design:

- **Modular** - the system is designed with distinct stages/phases to make the project more manageable and understandable.
- **Data-driven learning** - performance of the final output relies heavily on the quality and quantity of the labelled data collected during the 'registration' phase.
- **Feature extraction** - **`Local Binary Pattern (LBP)`** is used to train on the data, and is a crucial, intentional choice, as it is suited to extracting robust features that are less sensitive to variations in lighting.
- **Model-based recognition** - **`Local Binary Pattern Histogram (LBPH)`** model, once trained, encapsulates all the learned knowledge in an `XML (.xml)` file, which can then be used for efficient, real-time facial recognition applications.
- **Thresholding** - confidence score and defined threshold values help in deciding whether a face is correctly detected, as well as if the detected face matches a known individual.
- **Persistence** - people.csv file and recognition_model.xml ensure that the registered users and the trained model, respectively, stay persistent across application sessions.
- **Error-handling** - we include basic error handling to make the project more robust.

We shall now create a well-defined algorithm that goes in-depth on all of the above-mentioned stages and features before we take a look at the code to give us a more implementation-centric perspective of this approach.

Note - it is assumed that the entire application is run in a loop using the `tkinter.Tk.mainloop()` method starting from Step-5 until the user either exits the process or closes the `root` window.

Algorithm:

Input - live video stream (webcam) and a user-provided name/tag
Output - detected faces labelled as either known (using given names/tags) or *'UNKNOWN'* for unidentified faces
Steps:

Step-1 : START

Initialisation…

Step-2 : Load face detection classifier (`Haar cascade`), and initialise `LBPH (Local Binary Pattern Histograms)` face recogniser.

Step-3 : Load registered user data from `people.csv` into a dictionary variable `labels_dict.`

Step-4: If the trained model file `recognition_model.xml` exists, load it.

Step-5 : If the user clicks on the *'Detect'* button, requesting initialisation of the detection and recognition process, but the `people.csv` file contains no entries, then continue. Otherwise, skip to Step-24.

Step-6 : Display error message, preventing the user from continuing and leaving only the *'Register'* option as valid.

Registration…

Step-7 : If the user clicks on the *'Register'* button, prompt for name.

Step-8 : The system assigns a unique ID to all new users.

Step-9 : Activate the camera and start displaying the live camera feed.

Step-10 : If the *'Cancel'* button is pressed, re-display the main window, and return to Step-5.

Training…

Step-11 : If the *'Train'* button is pressed, for each frame of the video, detect faces using the `Haar cascade` classifier, and for each face successfully detected, continue after initialising variable `image_count` to 1.

Step-12 : Create a cropped image from each face centered on the face.

Step-13 : Convert the image to grayscale.

Step-14 : Resize the grayscale image to a *standard size (200x200)*.

Step-15 : Save the processed image to the data directory using the format '`p_id.image_count.jpg`'.

Step-16 : Display the image in a separate window.

Step-17 : Increment the image counter (`image_count`).

Step-18: If variable image_count has reached the value 200, continue to Step-19 after releasing the camera; otherwise, go back and continue from Step-12.

Step-19 : Load the images from the data directory.

Step-20 : For each image, read the image, then extract **`L`**`ocal` **`B`**`inary` **`P`**`attern` **`(LBP)`** features.

Step-21 : Train the `LBPH` face recogniser using the extracted features and their corresponding `p_id` labels.

Step-22 : Save the trained model to `recognition_model.xml`.

Step-23 : Append the new user's `p_id` and user_name to the `people.csv` file.

Recognition/Detection…

Step-24 : Activate the camera and display the live camera feed.

Step-25 : For each frame of the stream, detect faces using the `Haar cascade classifier`, and for each detected face, continue from Step-26.

Step-26 : Create a cropped image from each face centered on the face.

Step-27 : Convert the image to grayscale.

Step-28 : Resize the grayscale image to a *standard size (200x200)*.

Step-29 : Extract `LBP` features from the processed face image.

Step-30 : Use the trained `LBPH` model to predict the `p_id` and `confidence` score.

Step-31 : If the confidence score is below a specified threshold, continue to Step-32, else, skip to Step-34.

Step-32 : Retrieve the corresponding `user_name` from `labels_dict` using the predicted `p_id`.

Step-33 : Display the `user_name` and a bounding box around the detected face in the camera feed.

Step-34 : Label the face as *'UNKNOWN'* and display it with a bounding box.

Step-35 : Display the processed video frame.

Step-36 : Release the camera when it is no longer needed and close all windows.

Step-37 : STOP

Code - Implementation

Here is the implementation of the Facial Recognition System in Python using `tkinter` for the GUI and `OpenCV`:

```
import tkinter as tk
from tkinter import messagebox, simpledialog, Toplevel, Label, Button
import csv
import cv2
from PIL import Image, ImageTk
import os
import numpy as np

class Application:
    def __init__(self, master):
        self.master = master
        self.master.title("Face Recognition System")
        self.master.geometry("300x200")
        self.face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
        self.p_id = None
        self.user_name = None
        self.data_dir = "data"
        self.video_capture = None
        self.is_streaming = False
        self.is_detecting = False
        self.recognizer = cv2.face.LBPHFaceRecognizer_create()
        self.load_model()
        self.labels_dict = self.load_labels()

        self.cam_width = None
        self.cam_height = None
        self.video_canvas = None

        self.detection_params = {
            'scaleFactor': 1.2,
            'minNeighbors': 6
        }


        self.create_main_widgets()

    def load_model(self):
        if os.path.exists("recognition_model.xml"):
            self.recognizer.read("recognition_model.xml")

    def load_labels(self):
        labels = {}
        csv_path = "people.csv"
        if not os.path.exists(csv_path):
            return labels
        try:
            with open(csv_path, newline='', encoding='utf-8') as csvfile:
                reader = csv.reader(csvfile)
                next(reader)
                for row in reader:
                    if len(row) >= 2 and row[0].isdigit():
```

```
                        labels[int(row[0])] = row[1]
        except Exception as e:
            messagebox.showerror("Error", f"Failed to load labels from people.csv: {e}")
        return labels

    def create_main_widgets(self):
        self.clear_master()

        self.detect_button = tk.Button(self.master, text="Detect", command=self.login, width=20)
        self.detect_button.pack(pady=(40, 10), anchor='center')

        self.register_button = tk.Button(self.master, text="Register",
command=self.register_prompt, width=20)
        self.register_button.pack(pady=10, anchor='center')

        self.master.geometry("300x200")

    def clear_master(self):
        for widget in self.master.winfo_children():
            widget.destroy()

    def login(self):
        csv_path = "people.csv"
        try:
            with open(csv_path, newline='', encoding='utf-8') as csvfile:
                reader = csv.reader(csvfile)
                rows = list(reader)
                if len(rows) <= 1:
                    messagebox.showwarning("No Faces Found", "No known faces available. Please
register first.")
                    return
        except FileNotFoundError:
            messagebox.showwarning("No Faces Found", "No known faces available. Please register
first.")
            return

        self.load_model()
        self.labels_dict = self.load_labels()
        self.open_detection_window()

    def open_detection_window(self):
        self.clear_master()

        label = tk.Label(self.master, text="Face Detection", font=("Helvetica", 12))
        label.pack(pady=5)

        if self.video_capture is None or not self.video_capture.isOpened():
            self.video_capture = cv2.VideoCapture(0)
            if not self.video_capture.isOpened():
                messagebox.showerror("Camera Error", "Cannot open webcam.")
                self.create_main_widgets()
                return

        frame_width = int(self.video_capture.get(cv2.CAP_PROP_FRAME_WIDTH))
        frame_height = int(self.video_capture.get(cv2.CAP_PROP_FRAME_HEIGHT))

        self.cam_width = frame_width // 2
        self.cam_height = frame_height // 2
```

```
        canvas_width = self.cam_width + 200
        canvas_height = self.cam_height + 200

        total_height = canvas_height + 150
        total_width = canvas_width + 20
        self.master.geometry(f"{total_width}x{total_height}")

        self.video_canvas = tk.Canvas(self.master, width=canvas_width, height=canvas_height)
        self.video_canvas.pack()

        back_button = tk.Button(self.master, text="Back", command=self.cancel_detection,
width=20)
        back_button.pack(pady=10)

        self.is_detecting = True
        self.is_streaming = False
        self.update_video_stream_detect()

    def update_video_stream_detect(self):
        if not self.is_detecting or self.video_capture is None:
            return

        ret, frame = self.video_capture.read()
        if not ret:
            self.stop_video_stream()
            messagebox.showerror("Camera Error", "Failed to capture video frame.")
            return

        frame = cv2.flip(frame, 1)
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        faces = self.face_cascade.detectMultiScale(gray, **self.detection_params)

        for (x, y, w, h) in faces:
            face_img = gray[y:y+h, x:x+w]
            face_img_resized = cv2.resize(face_img, (200, 200))

            try:
                label, confidence = self.recognizer.predict(face_img_resized)
                if confidence < 35:
                    name = self.labels_dict.get(label, "UNKNOWN")
                    color = (255, 255, 255)
                    label_text = f"{name}"
                else:
                    name = "UNKNOWN"
                    color = (0, 0, 255)
                    label_text = "UNKNOWN"
            except:
                name = "UNKNOWN"
                color = (0, 0, 255)
                label_text = "UNKNOWN"

            cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)
            cv2.putText(frame, label_text, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.7, color, 2)

        small_frame = cv2.resize(frame, (self.cam_width, self.cam_height))
        cv2image = cv2.cvtColor(small_frame, cv2.COLOR_BGR2RGBA)
```

```
        img = Image.fromarray(cv2image)
        imgtk = ImageTk.PhotoImage(image=img)

        x_coord = (self.cam_width + 200) // 2
        y_coord = ((self.cam_height + 200) // 4) + 100

        self.video_canvas.delete("all")
        self.video_canvas.create_image(x_coord, y_coord, anchor=tk.CENTER, image=imgtk)
        self.video_canvas.imgtk = imgtk

        self.master.after(30, self.update_video_stream_detect)

    def stop_video_stream(self):
        self.is_detecting = False
        self.is_streaming = False
        if self.video_capture:
            self.video_capture.release()
            self.video_capture = None

    def cancel_detection(self):
        self.stop_video_stream()
        self.create_main_widgets()

    def register_prompt(self):
        csv_path = "people.csv"
        next_p_id = 1
        try:
            with open(csv_path, newline='', encoding='utf-8') as csvfile:
                reader = csv.reader(csvfile)
                rows = list(reader)
                if len(rows) > 1:
                    existing_ids = [int(row[0]) for row in rows[1:] if row and row[0].isdigit()]
                    if existing_ids:
                        next_p_id = max(existing_ids) + 1
        except FileNotFoundError:
            next_p_id = 1

        self.p_id = next_p_id

        prompt = f"Your assigned ID is {self.p_id}.\nPlease enter your name:"
        name = simpledialog.askstring("Register", prompt, parent=self.master)

        if name and name.strip():
            self.user_name = name.strip()
            if not os.path.exists(self.data_dir):
                os.makedirs(self.data_dir)
            self.show_camera_feed()
        else:
            messagebox.showwarning("Input Error", "Name cannot be empty. Registration
cancelled.")

    def show_camera_feed(self):
        self.clear_master()

        label = tk.Label(self.master, text=f"Registering {self.user_name} (ID: {self.p_id})",
font=("Helvetica", 12))
        label.pack(pady=5)
```

```
        self.video_capture = cv2.VideoCapture(0)
        if not self.video_capture.isOpened():
            messagebox.showerror("Camera Error", "Cannot open webcam.")
            self.create_main_widgets()
            return

        frame_width = int(self.video_capture.get(cv2.CAP_PROP_FRAME_WIDTH))
        frame_height = int(self.video_capture.get(cv2.CAP_PROP_FRAME_HEIGHT))

        self.cam_width = frame_width // 2
        self.cam_height = frame_height // 2

        canvas_width = self.cam_width + 200
        canvas_height = self.cam_height + 200

        total_height = canvas_height + 150
        total_width = canvas_width + 20
        self.master.geometry(f"{total_width}x{total_height}")

        self.video_canvas = tk.Canvas(self.master, width=canvas_width, height=canvas_height)
        self.video_canvas.pack()

        train_button = tk.Button(self.master, text="Train", command=self.open_capture_window,
width=20)
        train_button.pack(pady=10)

        cancel_button = tk.Button(self.master, text="Cancel", command=self.cancel_registration,
width=20)
        cancel_button.pack(pady=5)

        self.is_streaming = True
        self.update_video_stream()

    def update_video_stream(self):
        if not self.is_streaming or self.video_capture is None:
            return

        ret, frame = self.video_capture.read()
        if not ret:
            self.stop_video_stream()
            messagebox.showerror("Camera Error", "Failed to capture video frame.")
            return

        frame = cv2.flip(frame, 1)

        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        faces = self.face_cascade.detectMultiScale(gray, **self.detection_params)

        for (x, y, w, h) in faces:
            cv2.rectangle(frame, (x,y), (x+w, y+h), (0, 255, 255), 2)

        small_frame = cv2.resize(frame, (self.cam_width, self.cam_height))

        cv2image = cv2.cvtColor(small_frame, cv2.COLOR_BGR2RGBA)
        img = Image.fromarray(cv2image)
        imgtk = ImageTk.PhotoImage(image=img)

        x_coord = (self.cam_width + 200) // 2
```

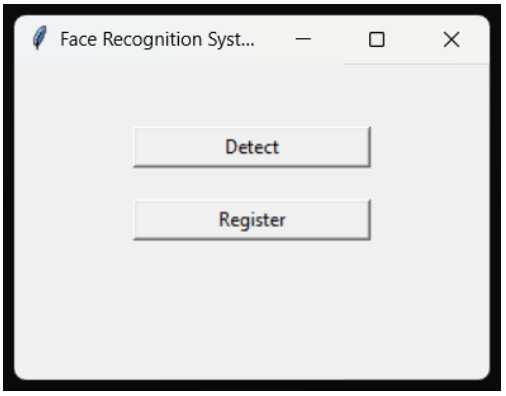
```
        y_coord = ((self.cam_height + 200) // 4) + 100

        self.video_canvas.delete("all")
        self.video_canvas.create_image(x_coord, y_coord, anchor=tk.CENTER, image=imgtk)
        self.video_canvas.imgtk = imgtk

        self.master.after(30, self.update_video_stream)

    def cancel_registration(self):
        self.stop_video_stream()
        self.create_main_widgets()

    def open_capture_window(self):
        self.stop_video_stream()
        self.capture_popup = Toplevel(self.master)
        self.capture_popup.title("Capture and Train Faces")
        self.capture_popup.geometry("350x420")
        self.capture_popup.transient(self.master)
        self.capture_popup.grab_set()

        label = Label(self.capture_popup, text=f"Capturing images for {self.user_name} (ID:
{self.p_id})", font=("Helvetica", 12))
        label.pack(pady=10)

        self.capture_img_label = Label(self.capture_popup)
        self.capture_img_label.pack()

        self.progress_label = Label(self.capture_popup, text="Captured 0 / 200 images",
font=("Helvetica", 10))
        self.progress_label.pack(pady=5)

        self.stop_capture_button = Button(self.capture_popup, text="Stop Capture",
command=self.stop_capture, width=20)
        self.stop_capture_button.pack(pady=10)

        self.capture_video = cv2.VideoCapture(0)
        if not self.capture_video.isOpened():
            messagebox.showerror("Camera Error", "Cannot open webcam.")
            self.capture_popup.destroy()
            self.create_main_widgets()
            return

        self.capture_count = 0
        self.max_capture = 200
        self.is_capturing = True
        self.capture_images_loop()

        self.capture_popup.protocol("WM_DELETE_WINDOW", self.on_capture_popup_close)

    def capture_images_loop(self):
        if not self.is_capturing:
            return

        ret, frame = self.capture_video.read()
        if not ret:
            messagebox.showerror("Camera Error", "Failed to capture video frame.")
            self.stop_capture()
            return
```

```
        frame = cv2.flip(frame, 1)

        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        faces = self.face_cascade.detectMultiScale(gray, **self.detection_params)

        if len(faces) > 0 and self.capture_count < self.max_capture:
            (x, y, w, h) = faces[0]
            face_img = gray[y:y+h, x:x+w]
            face_img = cv2.resize(face_img, (200, 200))

            cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 255), 2)

            img_filename = f"{self.p_id}.{self.capture_count + 1}.jpg"
            img_path = os.path.join(self.data_dir, img_filename)
            cv2.imwrite(img_path, face_img)

            self.capture_count += 1
            self.progress_label.config(text=f"Captured {self.capture_count} / {self.max_capture}
images")

            face_color = cv2.cvtColor(face_img, cv2.COLOR_GRAY2RGBA)
            img = Image.fromarray(face_color)
            imgtk = ImageTk.PhotoImage(image=img)
            self.capture_img_label.imgtk = imgtk
            self.capture_img_label.configure(image=imgtk)

            if self.capture_count >= self.max_capture:
                self.is_capturing = False
                self.capture_video.release()
                self.capture_video = None
                self.train_model()
                return
        else:
            for (x, y, w, h) in faces:
                cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 255), 2)

        frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGBA)
        frame_img = Image.fromarray(frame_rgb)
        imgtk_full = ImageTk.PhotoImage(image=frame_img)

        self.capture_img_label.after(30, self.capture_images_loop)
        self.capture_img_label.master.update()

    def train_model(self):
        images, labels = [], []
        for filename in os.listdir(self.data_dir):
            if filename.endswith(".jpg"):
                try:
                    img = cv2.imread(os.path.join(self.data_dir, filename), cv2.IMREAD_GRAYSCALE)
                    if img is None:
                        continue
                    p_id_str = filename.split(".")[0]
                    if not p_id_str.isdigit():
                        continue
                    label = int(p_id_str)
                    images.append(img)
                    labels.append(label)
```

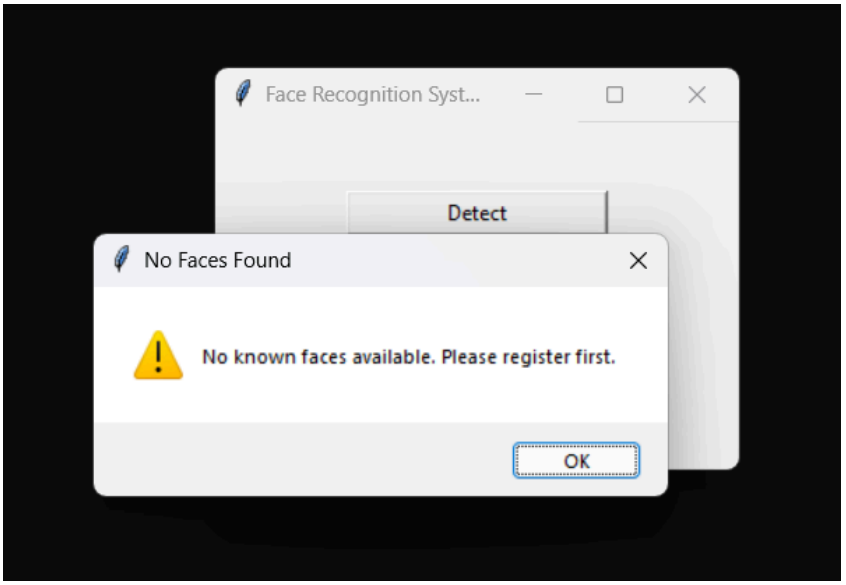
```
                except Exception:
                    continue

        if len(images) == 0 or len(labels) == 0:
            messagebox.showerror("Training Error", "No images found for training.")
            self.capture_popup.destroy()
            self.create_main_widgets()
            return

        recognizer = cv2.face.LBPHFaceRecognizer_create()
        recognizer.train(images, np.array(labels))
        recognizer.write("recognition_model.xml")

        self.update_people_csv()

        messagebox.showinfo("Training Complete", f"User '\{self.user_name}\' registered and model
trained successfully.")
        self.capture_popup.destroy()
        self.create_main_widgets()

    def update_people_csv(self):
        csv_path = "people.csv"
        file_exists = os.path.exists(csv_path)
        try:
            with open(csv_path, 'a', newline='', encoding='utf-8') as csvfile:
                writer = csv.writer(csvfile)
                if not file_exists:
                    writer.writerow(["p_id", "name"])
                writer.writerow([self.p_id, self.user_name])
        except Exception as e:
            messagebox.showerror("File Error", f"Failed to update people.csv: {e}")

    def stop_capture(self):
        self.is_capturing = False
        if self.capture_video:
            self.capture_video.release()
            self.capture_video = None
        messagebox.showinfo("Capture Stopped", "Image capture stopped.")
        if self.capture_popup:
            self.capture_popup.destroy()
        self.create_main_widgets()

    def on_capture_popup_close(self):
        if messagebox.askokcancel("Quit", "Capture in progress. Do you want to stop?"):
            self.stop_capture()

def main():
    root = tk.Tk()
    root.eval('tk::PlaceWindow . center')
    app = Application(root)
    root.mainloop()

if __name__ == "__main__":
    main()
```

Output - Final Look

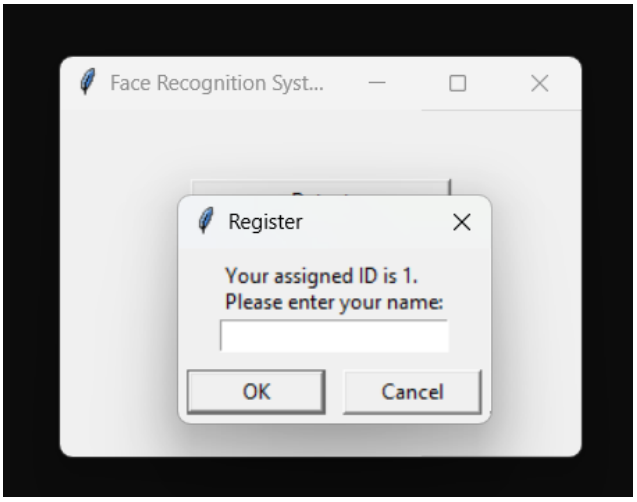
On startup:



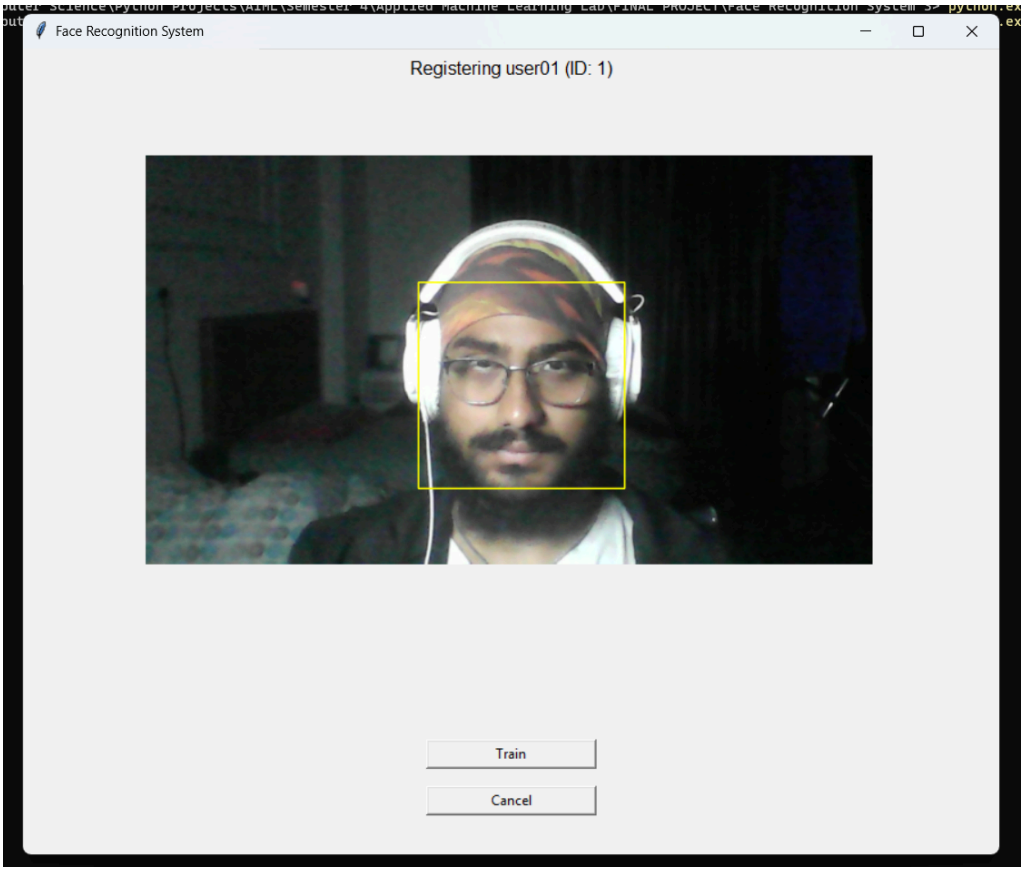
Triggered due to pressing *'Detect'* while the dataset is empty:



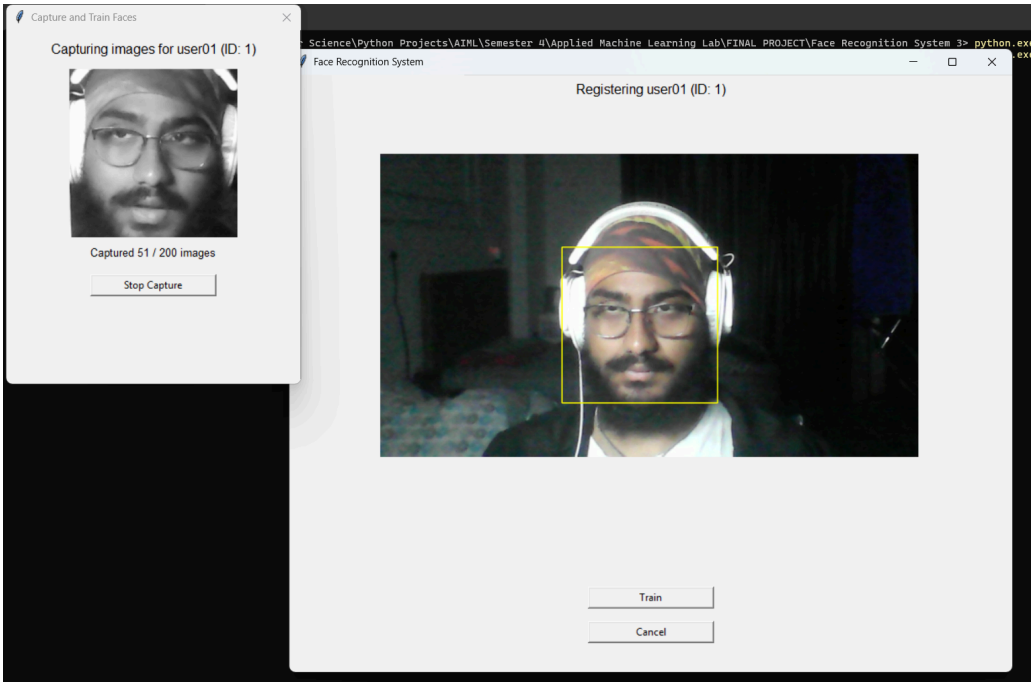
Always triggered on pressing *'Register'*:



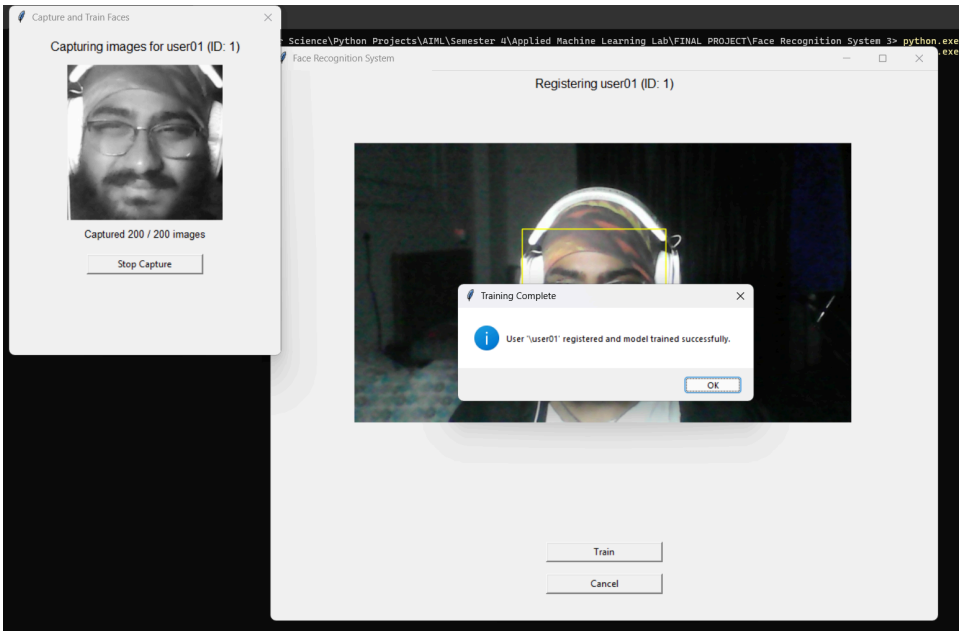
The window changes after the user enters a valid name/tag:



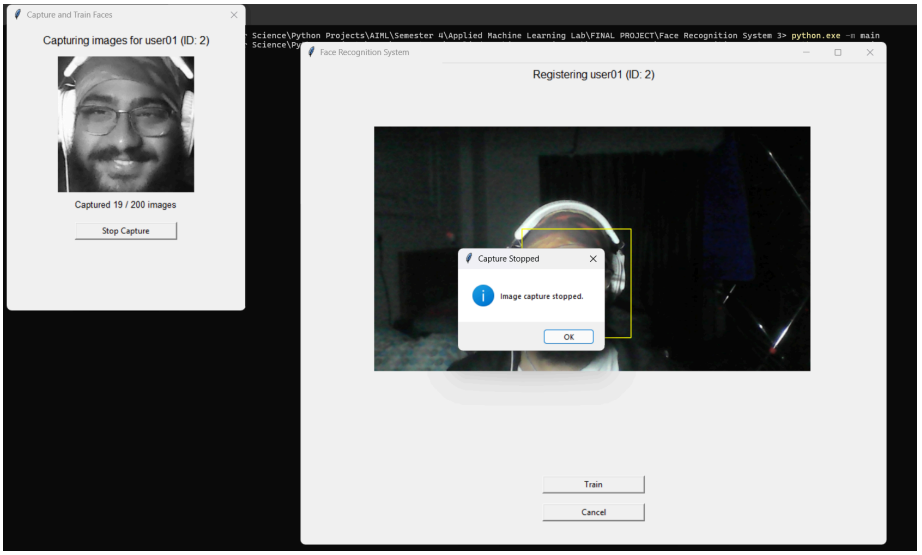
A new window with a processing preview opens after pressing *'Train'* (Main feed is paused) (*'Cancel'* results in return to main window without model training/data entry into `people.csv`):



Successful training message:

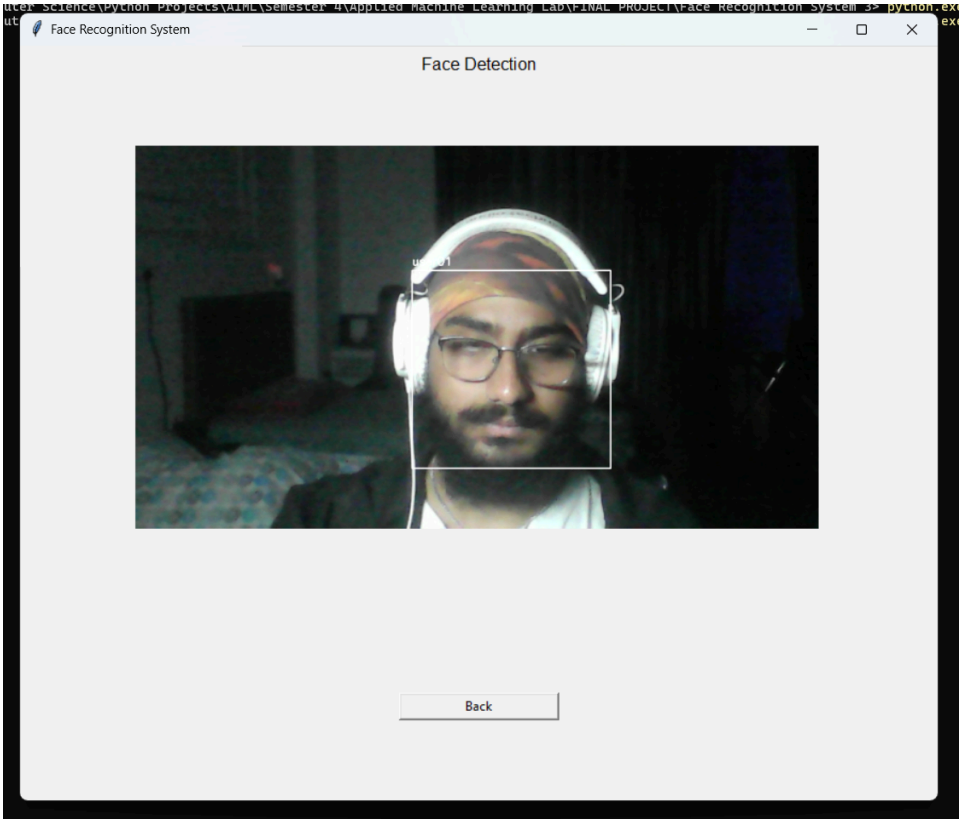


Error/info message if image capture is stopped in the middle of the training process (facilitates return to main window without model training/data entry into `people.csv`):

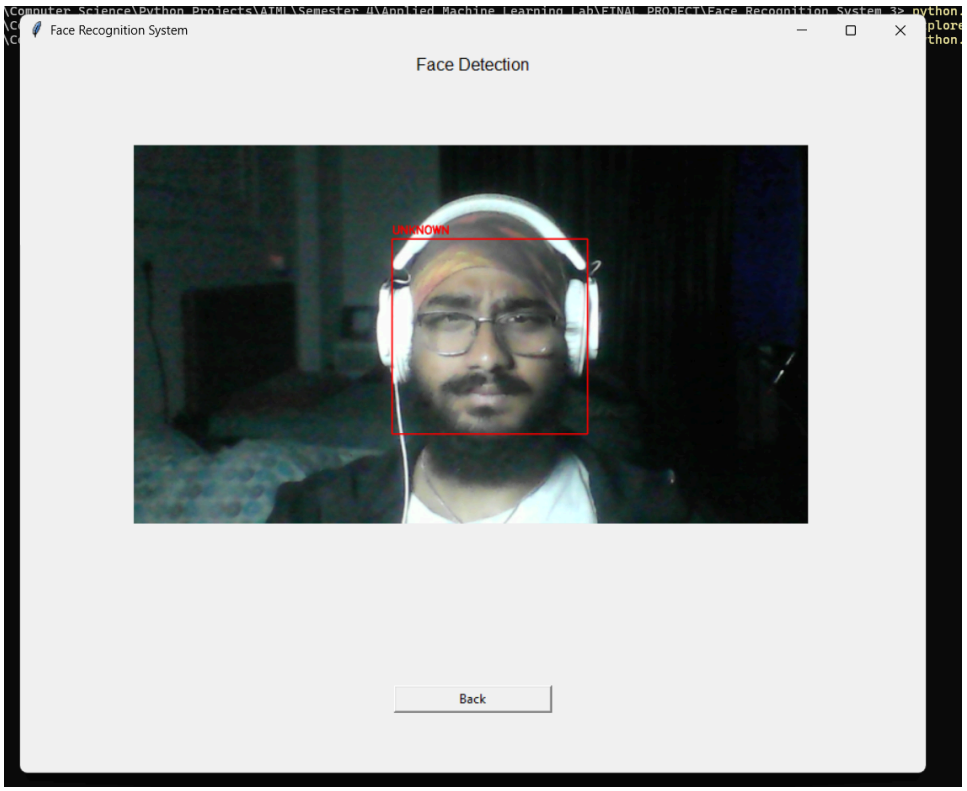


On pressing *'Detect'* with at least one valid entry in the `people.csv` file:

Known user *'user01'*:



'UNKNOWN' user:



Note - pressing *'Back'* on the *'Detect'* window takes you back to the first window.

Conclusion - Facing the Flaws

Here, we were able to demonstrate the implementation of a supervised LBP-based facial recognition system, highlighting the key stages of data acquisition, model training, and real-time detection, and we were also able to provide a clear image of how easy it is for machines to learn to identify individuals with reasonable accuracy.

Furthermore, the project underlines the technology's potential as a practical application of ML, where we capture, process, and learn from labelled face data, showcasing the capacity of supervised ML techniques in solving real-world identification challenges.

On reflection, further work on this system, or things to keep in mind while creating better systems may include boosting the system's robustness (protection against crashes, bugs, and better error handling code), accuracy (tuning parameters and adjusting model choice, or implementing a CNN), and scalability (present system is best for handling a handful of users and would not fare well for hundreds of thousands of users, as is the case for major corporations), as well as addressing ethical considerations surrounding its deployment.

References

1. https://docs.python.org/3/
2. https://anzeljg.github.io/rin2/book2/2405/docs/tkinter/
3. https://python-adv-web-apps.readthedocs.io/en/latest/csv.html
4. https://docs.opencv.org/4.x/index.html
5. https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html
6. https://pillow.readthedocs.io/en/stable/
7. https://numpy.org/doc/
8. https://docs.python.org/3/library/os.html
9. https://docs.super.ai/docs/how-to-format-csv
10. https://stackoverflow.com/questions/729692/why-should-text-files-end-with-a-newline
11. https://www.geeksforgeeks.org/reading-csv-files-in-python/
12. https://stackoverflow.com/questions/17262256/how-to-read-one-single-line-of-csv-data-in-python
13. https://www.geeksforgeeks.org/python-os-path-join-method/
14. https://www.geeksforgeeks.org/python-os-path-split-method/
15. https://www.geeksforgeeks.org/python-pil-image-convert-method/
16. https://stackoverflow.com/questions/53357877/usage-of-ordq-and-0xff
17. https://www.youtube.com/playlist?list=PLiWNvnK7PSPGPDmrdo3jhi_7hvkGrkFlN
18. https://github.com/ageitgey/face_recognition *(Uses dlib and online face detection dataset)*
19. https://www.techtarget.com/searchdatacenter/tip/XML-vs-YAML-Compare-configuration-file-formats
20. https://www.geeksforgeeks.org/face-recognition-with-local-binary-patterns-lbps-and-opencv/
21. https://stackoverflow.com/questions/66763823/cv2-face-lbphfacerecognizer-create-is-not-working-even-though-i-have-installed
22. https://pypi.org/project/opencv-python/
23. https://stackoverflow.com/questions/45655699/attributeerror-module-cv2-face-has-no-attribute-createlbphfacerecognizer
24. https://www.geeksforgeeks.org/how-to-show-webcam-in-tkinter-window-python/
25. https://stackoverflow.com/questions/11420748/setting-camera-parameters-in-opencv-python
26. https://docs.opencv.org/3.4/d8/dfe/classcv_1_1VideoCapture.html