A project on

Implementing a Smart attendance system based on Face recognition

Team Members:
Anandakrishnan A 21BCE1682

Abstract:

This project presents a Python-based system for real-time face detection and recognition using the OpenCV library. The system offers functionalities for capturing images from a webcam, saving them with associated names, and subsequently detecting and recognizing faces in live video streams. This system can be used for attendance logging by institutions that have large populations. The user interface is built using tkinter, providing the appropriate functionalities to manage and access the database. Known faces are stored along with their corresponding encodings, facilitating efficient comparison during real-time detection. The system enables seamless integration of new faces into the recognition database, enhancing its adaptability and usability. With its straightforward interface and robust face recognition capabilities, this project can also serve as a versatile tool for various other applications, including security systems, payroll tracking, and personalized user experiences.

Introduction:

Facial Recognition is one of the most popular forms of biometric authentications. Due to its high resistance against fraud and its high level of security and effectiveness it is a very reliable form of authentication. This paper explores the integration of deep transfer learning into face recognition systems for smart attendance applications. This work will pave the way for widespread adoption of face recognition in attendance management, leading to increased efficiency, reduced errors, and a more secure learning environment. This paper aims to evaluate the effectiveness of deep transfer learning for face recognition and presents a comprehensive approach encompassing face detection, feature extraction, and classification. It will also discuss the implications for practical implementation and future research directions. Use the enter key to start a new paragraph. The appropriate spacing and indent are automatically applied.

Literature Survey

J. Nandre, S. Rai and B. R. Kanawade, "Comparative Analysis of Transfer Learning CNN for Face Recognition," 2022 2nd International Conference on Intelligent Technologies (CONIT), Hubli, India, 2022, pp. 1-6, doi: 10.1109/CONIT55038.2022.9847946.

This paper explores its potential for secure authentication, highlighting its advantages over traditional methods like passwords and fingerprints. It compares three deep learning approaches for face recognition: training a model from scratch, using pre-trained models with transfer learning, and using pre-trained models for feature extraction. The study utilizes three pre-trained CNN architectures - MobileNet, VGG16, and ResNet50 - and evaluates them on a dataset. Transfer learning with MobileNet

proved fastest and most accurate, achieving 100% accuracy with a batch size of 32. All three models performed well in real-time face recognition tasks, demonstrating the efficacy of transfer learning for this application.

Adithama, S. P., Maslim, M., & Gunawan, R. (2023). Implementation of face recognition for patient identification using the transfer learning method. TEM Journal, 12(2), 775-784. doi:https://doi.org/10.18421/TEM122-22

In this study, the authors investigate the use of face recognition for patient identification in hospitals, aiming to improve patient safety by reducing identification errors. They focus on the deep learning technique of CNNs and specifically employ the VGGFace2 pre-trained model and SENet 50 architecture and their dataset leverages one-shot learning. Their findings indicate high accuracy, with registration achieving 90%-100% and verification reaching 100%. However, further research is needed to assess the generalizability of these results and address potential concerns regarding data privacy and ethical implications of using facial recognition in healthcare settings.

A. Phornchaicharoen and P. Padungweang, "Face Recognition using Transferred Deep Learning for Feature Extraction," 2019 Joint International Conference on Digital Arts, Media and Technology with ECTI Northern Section Conference on Electrical, Electronics, Computer and Telecommunications Engineering (ECTI DAMT-NCON), Nan, Thailand, 2019, pp. 304-309, doi: 10.1109/ECTI-NCON.2019.8692306.

This paper investigates deep transfer learning for face recognition on datasets with numerous classes but limited images per class. Comparing two approaches - traditional feature extraction/classification versus transfer learning with pre-trained CNN - the study finds significant improvements in both accuracy and efficiency with transfer learning. The pre-trained CNN effectively extracts features, minimizing overfitting due to limited data, as evidenced by lower cross-entropy loss and higher accuracy on unseen data. Future work plans to fine-tune the pre-trained CNN, explore other models, and refine the CK+ dataset for improved performance.

S. Khan, E. Ahmed, M. H. Javed, S. A. A Shah and S. U. Ali, "Transfer Learning of a Neural Network Using Deep Learning to Perform Face Recognition," 2019 International Conference on Electrical, Communication, and Computer Engineering (ICECCE), Swat, Pakistan, 2019, pp. 1-5, doi: 10.1109/ICECCE47252.2019.8940754.

This paper explores facial recognition using convolutional neural networks, highlighting its advantages over past methods. The authors propose a system utilizing the pre-trained AlexNet CNN for transfer learning, achieving 97.95% accuracy on a dataset of 4,000 images across four classes. Compared to past techniques like Fisherfaces and LBPH, they find improved efficiency and accuracy. The model uses key features extracted by the CNN for recognition include eye distance, jawline length,

and nose width. They acknowledge the need for larger datasets for further accuracy gains and conclude that CNN-based approach is a reliable and effective solution for facial recognition.

G. NaliniPriya, R. Meenakshi, R. Mythili, B. Mathiyazhagi, E. Harini and S. B. Harini, "A Face Recognition Security Model Using Transfer Learning Technique," 2022 8th International Conference on Smart Structures and Systems (ICSSS), Chennai, India, 2022, pp. 1-6, doi: 10.1109/ICSSS54381.2022.9782238.

This paper explores a novel method of facial recognition using deep learning. The proposed system leverages the pre-trained AlexNet model to extract potent features, enabling highly accurate and faster face recognition compared to existing models. This translates to robust user authorization based on facial identification, effectively barring unauthorized access and safeguarding confidential information. However, improved organization and the inclusion of more comprehensive results could significantly enhance the paper's clarity and showcase the research contribution more effectively. This approach positions the proposed system as a promising solution for bolstering security measures.

Methodology

OpenCV:

OpenCV, short for Open Source Computer Vision Library, is an open-source computer vision and machine learning software library. It provides a wide range of functionalities for image and video analysis, including object detection, face recognition, feature detection, and image processing. OpenCV is written in C++ and offers interfaces for various programming languages, including Python, making it accessible to a broad community of developers. Within the Python ecosystem, the cv2 module serves as a popular interface to the OpenCV library, providing seamless integration and access to its extensive suite of tools and algorithms. With its comprehensive capabilities and cross-platform compatibility, OpenCV and its Python interface, cv2, have become foundational components in numerous computer vision projects, powering applications across diverse domains such as robotics, augmented reality, medical imaging, and surveillance.

Tkinter:

Tkinter is a standard Python library for creating graphical user interfaces (GUIs). It provides a simple and intuitive way to develop desktop applications with graphical components such as buttons, labels, entry fields, and more. Tkinter is based on the Tk GUI toolkit, which is widely used and well-supported across multiple platforms. With Tkinter, developers can create visually appealing and interactive applications with minimal effort, thanks to its easy-to-use syntax and extensive documentation. Despite its simplicity, Tkinter offers enough flexibility to create complex GUIs for a variety of applications, from simple utilities to sophisticated desktop software. As a core part of

the Python Standard Library, Tkinter is widely adopted and enjoys strong community support, making it an excellent choice for developing cross-platform desktop applications in Python.

face_recognition:
This library is used for face detection, face recognition, and face manipulation tasks.
It provides pre-trained models and methods to detect faces in images, compare faces, and extract facial features.
In the code, it's primarily used for loading images, detecting faces, encoding faces, and comparing them with known faces.

csv:
The csv module provides classes and functions for reading and writing CSV (Comma Separated Values) files.
Although it's not explicitly used in the provided code snippet, it might be intended for tasks like saving/loading data in CSV format, which is a common format for storing tabular data.
datetime:
The datetime module provides classes and functions for working with dates and times in Python.
It allows for creating, manipulating, and formatting date and time objects.
In the code, it's not directly used but imported for potential future use, perhaps for timestamping or handling date-related tasks.

Although not implemented in the provided code snippet, the csv module can be used for saving proof or logging data to a CSV file.
After capturing and saving images along with the corresponding names, the code could potentially log this information to a CSV file for record-keeping purposes.
This CSV file could contain columns for the captured image names and corresponding timestamps, providing a log of when each image was captured.
By utilizing the csv.writer or csv.DictWriter classes, the code could append new entries to an existing CSV file or create a new file if it doesn't exist.
Storing proof in a CSV file could serve various purposes, such as maintaining a record of captured images for security applications or for analysis purposes in research studies.

Implementation

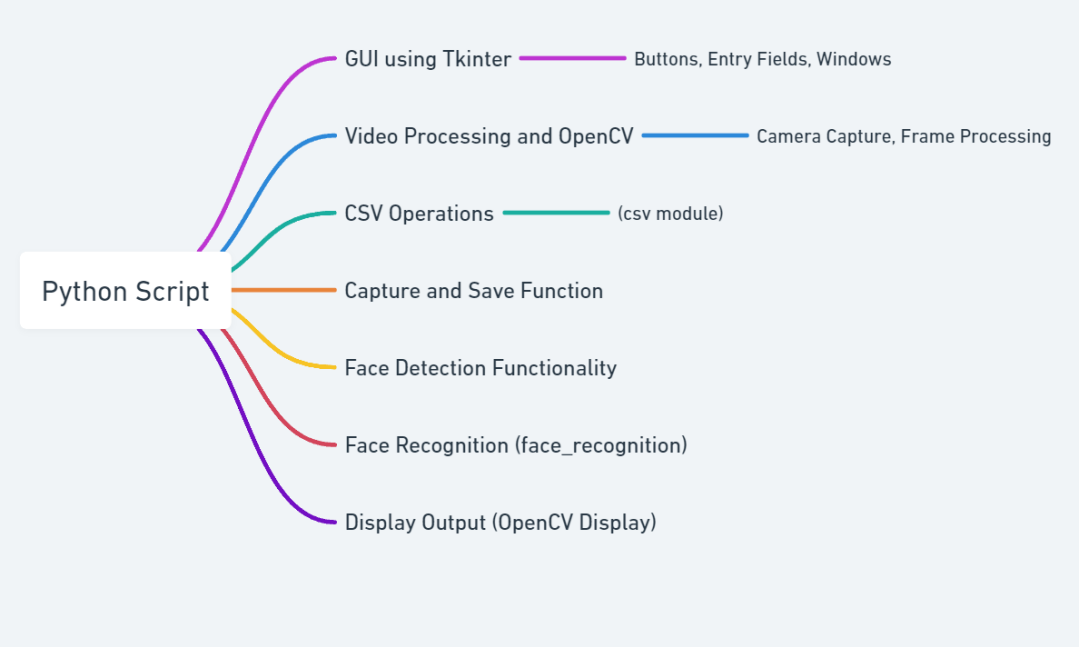
Environment Setup:
Install Python: Ensure that Python is installed on your system.
Install OpenCV: Install the OpenCV library (cv2) using pip, a package manager for Python.
Install face_recognition: Install the face_recognition library using pip.
Verify installations: Test that OpenCV and face_recognition are installed correctly by importing them in a Python script.

Image Capture and Saving:
Write a function to capture images from a camera device using OpenCV.
Utilize tkinter to create a GUI window with an entry field for entering the name associated with the captured image. This is used to enter new faces into the database.
Add relevant interface features to store the record.
Implement functionality to save the captured image as a file and display success/error messages using tkinter messagebox.

Face recognition model training:
Define a function to reload known faces from saved images.
Store paths to known face images along with theircorresponding names..
Use face_recognition library to load images, encode faces, and append encodings and names to lists.

Real-time face detection:
Create a function to detect faces in real-time video streams captured from the camera device.
Utilize OpenCV to read frames from the video stream and convert them to RGB format.
Use face_recognition library to locate faces in the frames and encode their features.
Compare the encoded features with known facesin the database to accurately identify them.
Highlight the face detected and display the user's information if identified.

Flow of Various Process:



Conclusion

In conclusion, the developed project represents a significant step towards implementing real-time face detection and recognition systems using readily available tools and libraries in the Python ecosystem. By leveraging OpenCV and the face_recognition library, the project offers a functional solution for capturing, training, and recognizing faces from webcam feeds. The system's ability to detect and recognize known faces in live video streams opens up a wide range of potential applications, including security monitoring, access control, and personalized user experiences. However, while the project demonstrates the feasibility of such systems, there are several areas for improvement to enhance its performance, accuracy, and usability.

One notable aspect for improvement is the system's performance, which may suffer from latency or reduced frame rates in real-time scenarios or with large datasets. Optimizations such as multi-threading, GPU acceleration, or model quantization could help alleviate these performance issues and improve the system's responsiveness. Additionally, enhancing the accuracy of face detection and recognition algorithms is crucial for reliable operation across diverse environments and conditions. Techniques such as data augmentation, model fine-tuning, and integrating advanced recognition models could enhance the system's accuracy and robustness.

Furthermore, the user interface of the project, while functional, could benefit from enhancements to improve user interaction and experience. Incorporating features such as real-time feedback, progress indicators, error handling, and customization options would make the system more intuitive and user-friendly. Moreover, integration with

other platforms and applications could extend the project's functionality and reach, enabling seamless integration into larger software ecosystems and workflows.

Overall, while the project provides a solid foundation for real-time face detection and recognition, there is ample room for improvement and further development. By addressing the identified shortcomings and implementing enhancements, the project can evolve into a more robust, accurate, and user-friendly solution with broader applications and impact in various domains.

Appendix

Source Code:

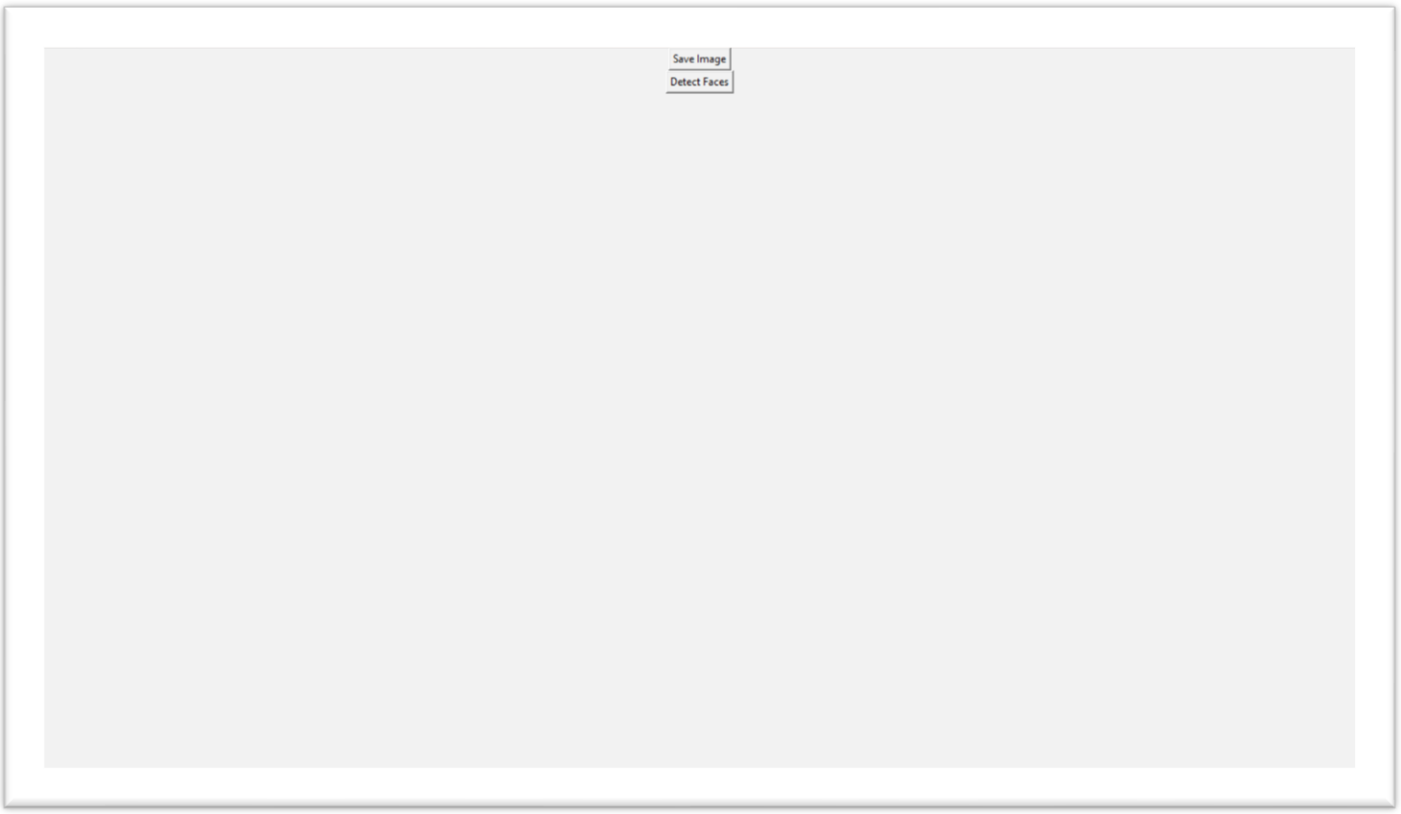
```
import face_recognition
import cv2
import csv
from datetime import datetime
import tkinter as tk
from tkinter import filedialog, messagebox

# Function to capture and save image with entered name
def capture_and_save():
    def capture_frame():
        nonlocal frame_saved
        ret, frame = capture.read()
        if ret:
            cv2.imshow("Capture", frame)
            frame_saved = frame.copy()

    def save_image():
        nonlocal frame_saved
        global known_face_encodings
        global known_faces_names

        name = name_entry.get()
        if name.strip() == "":
            messagebox.showerror("Error", "Please enter a name.")
            return

        if frame_saved is not None:
            cv2.imwrite(f"{name}.jpg", frame_saved)
            messagebox.showinfo("Success", f"Image saved as {name}.jpg")
```

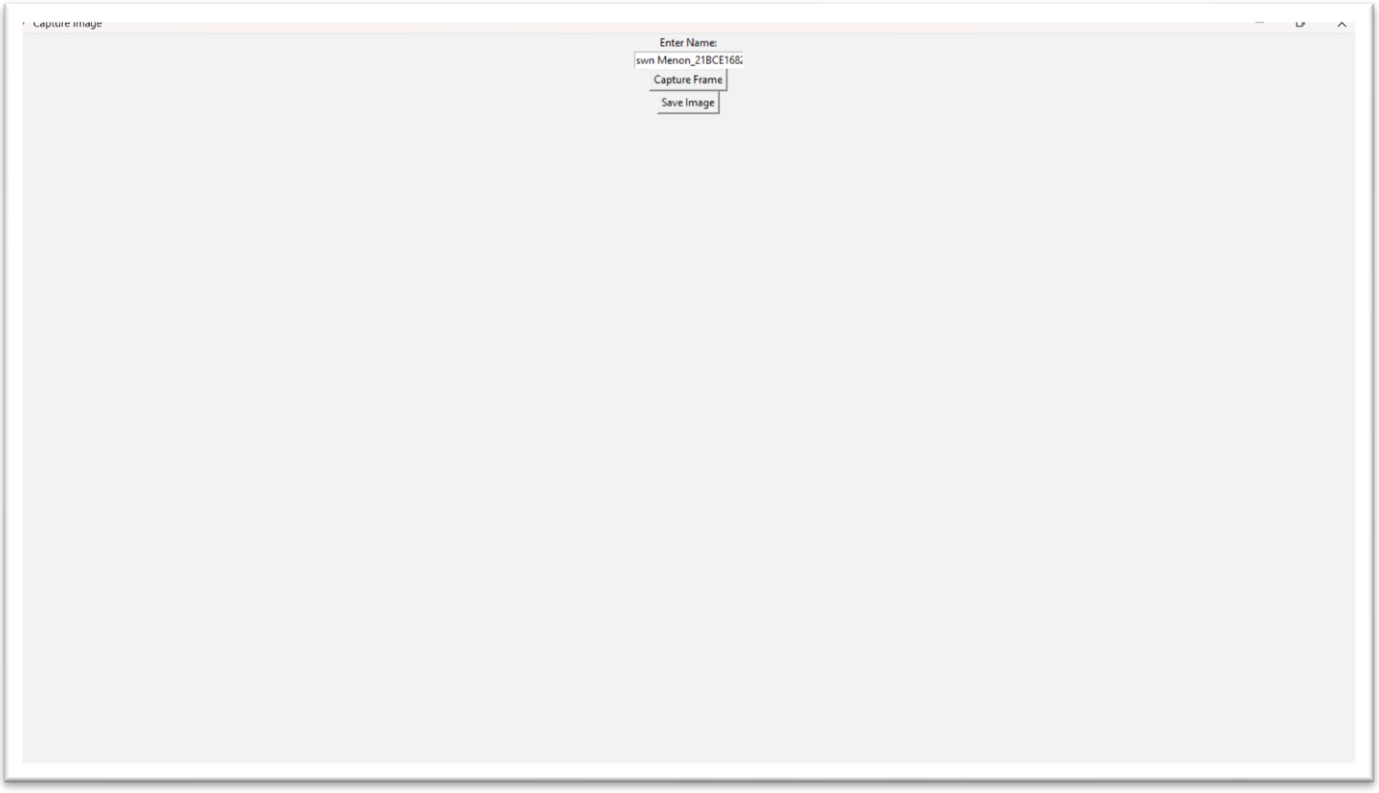
```
            # Reload known faces list
            image = face_recognition.load_image_file(f"{name}.jpg")
            encoding = face_recognition.face_encodings(image)[0]
            known_face_encodings.append(encoding)
            known_faces_names.append(name)

    # Create new window for image capture
    new_window = tk.Toplevel(root)
    new_window.title("Capture Image")

    # Label and entry for name input
    name_label = tk.Label(new_window, text="Enter Name:")
    name_label.pack()
    name_entry = tk.Entry(new_window)
    name_entry.pack()

    # Initialize video capture
    capture = cv2.VideoCapture(0)
    frame_saved = None

    # Button to capture image
    capture_button = tk.Button(new_window, text="Capture Frame",
command=capture_frame)
    capture_button.pack()

    # Button to save image
    save_button = tk.Button(new_window, text="Save Image", command=save_image)
    save_button.pack()

    # Function to close window when finished
    def on_closing():
        capture.release()
        cv2.destroyAllWindows()
        new_window.destroy()

    new_window.protocol("WM_DELETE_WINDOW", on_closing)

# Function to reload known faces list
def reload_known_faces():
    global known_face_encodings
    global known_faces_names

    # Clear existing known face encodings and names
    known_face_encodings.clear()
```

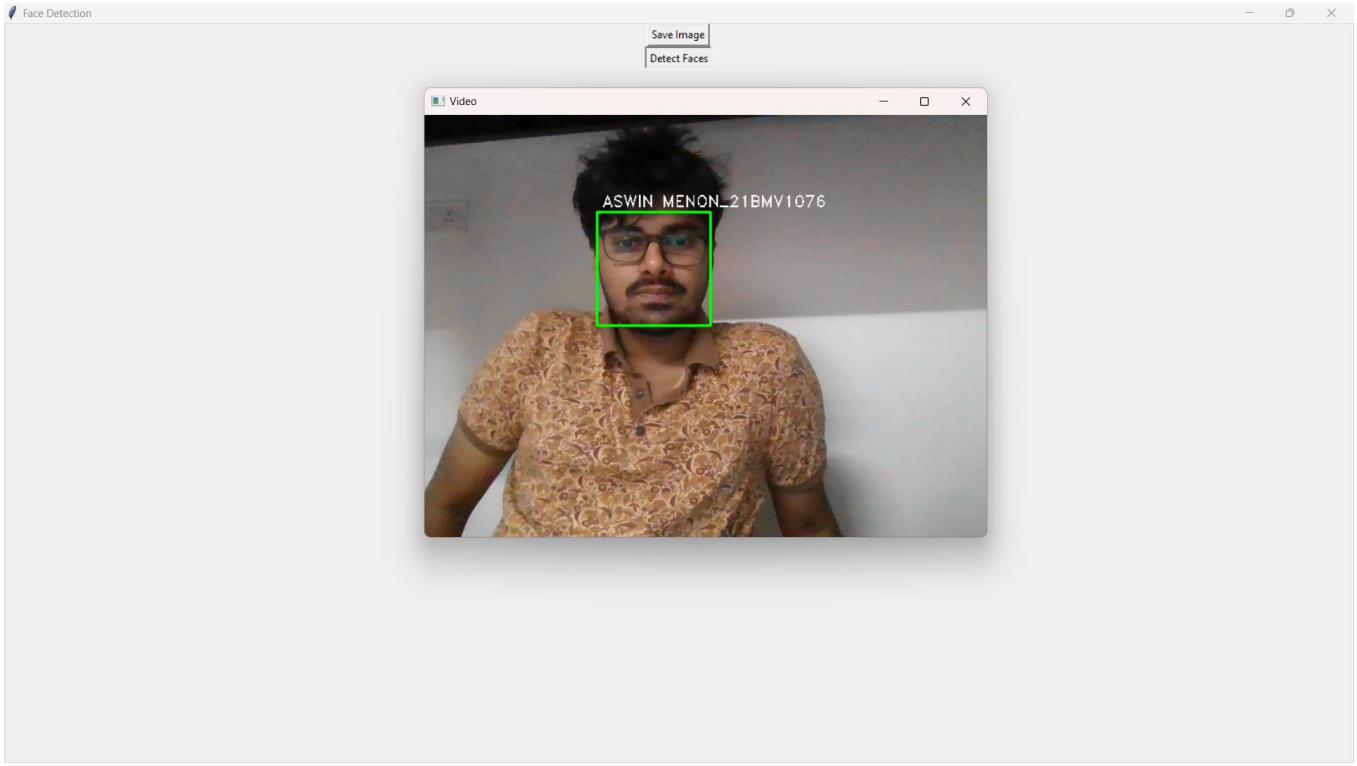
```
    known_faces_names.clear()

    # Load known face encodings
    for name, path in known_faces_paths.items():
        image = face_recognition.load_image_file(path)
        encoding = face_recognition.face_encodings(image)[0]
        known_face_encodings.append(encoding)
        known_faces_names.append(name)

# Function to perform face detection on updated data
def detect_faces():
    # Initialize variable to track whether any known face is detected
    known_face_detected = False

    while True:
        # Read frame from video capture
        ret, frame = video_capture.read()

        # Convert frame to RGB format
        rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

        # Find face locations
        face_locations = face_recognition.face_locations(rgb_frame)

        # Encode faces in the current frame
        face_encodings = face_recognition.face_encodings(rgb_frame, face_locations)

        # Iterate over detected faces
        for face_encoding, (top, right, bottom, left) in zip(face_encodings, face_locations):
            # Compare face encoding with known face encodings
            matches = face_recognition.compare_faces(known_face_encodings, face_encoding)
            if True in matches:
                known_face_detected = True

                # Get the name of the detected face
                name = known_faces_names[matches.index(True)]

                # Draw a rectangle and text around the face
                cv2.rectangle(frame, (left, top), (right, bottom), (0, 255, 0), 2)
                cv2.putText(frame, name, (left + 6, top - 6), cv2.FONT_HERSHEY_DUPLEX, 0.6, (255, 255, 255), 1)

        # Display the resulting frame
```

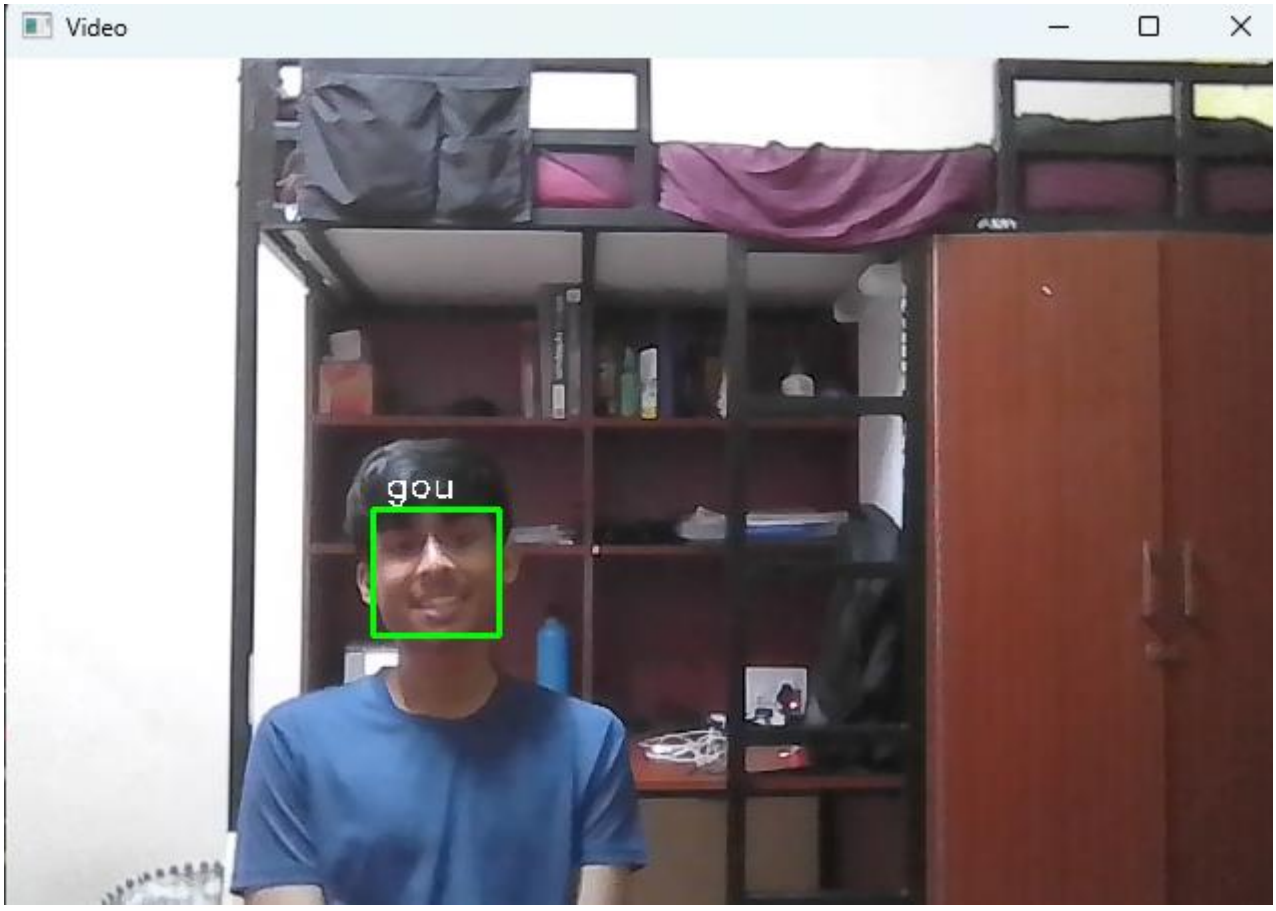
```
        cv2.imshow('Video', frame)

        # Break the loop if 'q' is pressed
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

# Initialize video capture
video_capture = cv2.VideoCapture(0)

# Load known face encodings and names
known_face_encodings = []
known_faces_names = []

# Define paths to known faces
known_faces_paths ={
}

# Load known face encodings
reload_known_faces()

# GUI
root = tk.Tk()
root.title("Face Detection")
root.geometry("300x200")

# Button to open image capture window
capture_button = tk.Button(root, text="Save Image", command=capture_and_save)
capture_button.pack()

# Button to start face detection
detect_button = tk.Button(root, text="Detect Faces", command=detect_faces)
detect_button.pack()

root.mainloop()

# Release video capture
video_capture.release()
cv2.destroyAllWindows()
```

UI


Save Image
Detect Faces


Capture Image
Enter Name:
swn Menon_21BCE168
Capture Frame
Save Image


Face Detection
Save Image
Detect Faces
Video
ASWIN MENON_21BMV1076


Video
gou