**Python Project Report: Face & Object Recognition App**

**Name:** Samarth Birdawade

**SRN:** 31240165

**Class:** MCA – FY (Sem 2)

**Div:** A

**Guide / Instructor:** Prof. Khushbu Kushwaha

**Introduction**

The Face and Object Recognition App is a Python-based application designed to provide users with the ability to register and recognize faces, with the potential for future expansion to include object recognition capabilities. This application utilizes a combination of powerful libraries, including OpenCV for image and video processing, face\_recognition for face detection and recognition, and Tkinter for creating a user-friendly graphical interface.

The app's core functionalities are divided into two main processes: face registration and face recognition. Face registration involves capturing multiple images of a user's face from various angles, extracting unique facial encodings, and storing this data for future recognition. Face recognition, on the other hand, involves capturing a live video stream, detecting faces within the stream, and comparing their encodings against the stored encodings to identify individuals.

The graphical user interface (GUI) is constructed using Tkinter and enhanced with the CustomTkinter library for additional styling options. The GUI provides a straightforward interface for users to interact with the application, featuring buttons to initiate face registration and recognition, labels to display instructions and recognized names, and a designated area to display the live video feed.

The codebase is structured into three primary files: app.py, face\_registration\_script.py, and face\_recognition\_script.py. This modular design promotes code reusability and maintainability, separating the core application logic, face registration process, and face recognition process into distinct modules.

While the current version of the app focuses primarily on face recognition, there is significant potential for future development. This includes the integration of object recognition capabilities, enhancements to the GUI to incorporate features such as multi-camera support and user profiles, performance optimization for real-time processing, robust error handling, and security considerations to protect user data.

The Face and Object Recognition App, while currently in its early stages of development, demonstrates the potential of Python and its associated libraries in creating applications with practical uses in various fields, including security, identification, and human-computer interaction. With further development and refinement, this application can become a valuable tool for both personal and professional use.

**Core Functionalities**

The app is designed to perform two primary functions:

1. **Face Registration:** This process involves capturing images of a new user's face from a webcam, extracting facial encodings using the ‘face\_recognition’ library, and storing these encodings along with the user's name in a pickle file (encodings.pkl).
2. **Face Recognition:** This process involves capturing a video stream from the webcam, detecting faces in each frame, extracting facial encodings, and comparing them with the encodings stored in the encodings.pkl file. If a match is found, the corresponding name is displayed along with a bounding box around the recognized face in the video stream.

**Libraries Used**

The project utilizes several Python libraries to achieve its functionalities:

* **OpenCV (cv2):** Used for image and video processing, including capturing video streams, reading and writing images, and performing image manipulation.
* **face\_recognition:** Provides high-level functions for face detection and recognition, built on top of the dlib library.
* **NumPy (np):** Used for numerical operations, particularly for working with arrays and matrices.
* **Pillow (PIL):** Used for image manipulation, including opening, resizing, and converting images.
* **Tkinter (tk):** Provides the foundation for building the graphical user interface (GUI).
* **CustomTkinter:** Extends Tkinter with additional GUI elements and styling options.
* **TTKBootstrap (ttk):** Offers modern UI styling for Tkinter widgets.

**GUI Design**

The GUI is designed using a combination of Tkinter and CustomTkinter to provide a user-friendly interface. The main components of the GUI include:

* **Title Bar:** A custom title bar with minimize, maximize, and close buttons.
* **Buttons:** Buttons for triggering face registration and recognition functionalities.
* **Labels:** Labels for displaying instructions, prompts, and recognized names.
* **Video Feed:** A label for displaying the video stream from the webcam.

The GUI\_Layout.py file defines the layout and styling of the GUI, including theme colors, font styles, and button designs. This separation of concerns allows for easier customization and maintenance of the GUI.

**Code Structure**

The code is structured into three main files:

1. **app.py:** Contains the main application logic, including the GUI setup, button functionalities, and functions to launch the face registration and recognition scripts. It uses the subprocess module to run these scripts in separate processes.
2. **face\_registration\_script.py:** Handles the face registration process, including capturing images, extracting encodings, and storing them in the encodings.pkl file.
3. **face\_recognition\_script.py:** Handles the face recognition process, including capturing the video stream, detecting faces, extracting encodings, and comparing them with the stored encodings.

This modular structure promotes code reusability and maintainability.

**In-depth Code Analysis**

**app.py**

This file serves as the entry point for the application. It begins by importing necessary modules and retrieving theme colors from GUI\_Layout.py. The code then initializes a cv2.VideoCapture object to access the default webcam.

The GUI is built using Tkinter, with a custom title bar implemented using a tk.Frame. The title bar includes buttons for closing, maximizing, and minimizing the window, as well as a label for the title. The main content area features two buttons: "Register Face" and "Recognize Face," which trigger the respective functionalities.

The register\_face function launches the face\_registration\_script.py script using subprocess.run, passing the camera index as an argument. Similarly, the face\_recognition\_app function launches the face\_recognition\_script.py script.

The GUI also includes an image label displaying a skull icon at the bottom right corner. The window is centered on the screen, and various event bindings are set up for window dragging, button hover effects, and keyboard shortcuts.

**face\_registration\_script.py**

This file handles the face registration process. It starts by importing necessary modules and retrieving theme colors from GUI\_Layout.py. The code then defines constants for the directories where face images and encodings are stored.

The GUI for face registration is built using Tkinter, with a similar custom title bar as in app.py. The main content area includes a label prompting the user to enter their name, an entry field for the name, and a button to start the registration process.

The process\_new\_faces function iterates through the face images in the FACES\_DIR directory, extracts facial encodings using face\_recognition.face\_encodings, and saves the encodings along with the corresponding names in the ENCODINGS\_FILE.

The start\_registration function retrieves the user's name from the entry field, creates a directory for the user's face images, and hides the name entry and start button. It then calls the update\_frame function to start capturing and displaying the video feed.

The update\_frame function continuously updates the video feed in the GUI and displays instructions to the user for capturing images in different poses. The capture\_image function captures the current frame from the webcam and saves it as an image in the user's directory. After capturing six images, it calls the process\_new\_face\_encodings function to update the known encodings.

The process\_new\_face\_encodings function calls the process\_new\_faces function to update the encodings and displays a completion message. The show\_completion\_buttons function displays buttons for returning to the app or registering another face.

The return\_to\_app function closes the registration window and releases the webcam. The reset\_registration function resets the registration UI for a new user.

**face\_recognition\_script.py**

This file handles the face recognition process. It starts by importing necessary modules and retrieving theme colors from GUI\_Layout.py. The code then defines constants for the directories where face images and encodings are stored.

The GUI for face recognition is built using Tkinter, with a similar custom title bar as in app.py. The main content area includes a label for displaying the video feed.

The load\_known\_faces function loads the known face encodings and names from the ENCODINGS\_FILE. The process\_faces function runs in a separate thread and continuously processes the video stream, detecting faces and comparing their encodings with the known encodings. If a match is found, the corresponding name is stored in the recognized\_face dictionary along with the face's bounding box coordinates.

The update\_feed function continuously updates the video feed in the GUI and draws bounding boxes around recognized faces, displaying their names. The update\_frame function converts the OpenCV frame to a Tkinter-compatible format for display.

**Potential Areas for Improvement**

While the app provides basic face recognition functionality, there are several areas for potential improvement:

* **Object Recognition:** Implement object recognition capabilities by training models on suitable datasets. This would involve adding functionalities to register and recognize objects, similar to the face recognition features.
* **GUI Enhancements:** Improve the GUI with additional features and customization options, such as support for multiple cameras, user profiles, and settings configuration. This could involve adding menus, dialog boxes, and more interactive elements to the GUI.
* **Performance Optimization:** Optimize the code to improve performance and reduce resource consumption, particularly for real-time video processing. This could involve using more efficient algorithms, reducing image sizes, and optimizing the GUI update loop.
* **Error Handling:** Implement robust error handling to gracefully handle exceptions and provide informative feedback to the user. This could involve using try-except blocks to catch potential errors and display error messages to the user.
* **Security:** Consider security implications, such as protecting the stored facial encodings and preventing unauthorized access to the app. This could involve encrypting the encodings file and implementing user authentication.

**Conclusion**

The Face and Object Recognition App provides a basic implementation of face recognition using Python and various libraries. The code is well-structured and utilizes appropriate libraries for its functionalities. However, there are several areas for potential improvement, particularly in terms of object recognition, GUI enhancements, performance optimization, error handling, and security. By addressing these areas, the app can be made more robust, feature-rich, and suitable for real-world applications.