Brief Report

**1. Design Choices**

1.1. **Face Detection and Landmark Extraction:** For face detection and landmark extraction, dlib was selected as the primary tool due to its proven accuracy and efficiency in detecting faces and facial landmarks. "Dlib's” HOG (Histogram of Oriented Gradients) and CNN (Convolutional Neural Network) based face detectors provide high accuracy, even in challenging conditions such as partial occlusions or varied lighting. The face landmark extraction is based on the 68 landmark points model, which helps in detecting key facial features like the eyes, nose, mouth, and jawline, allowing for better alignment and recognition of faces.

**Why dlib?**

1. **Accuracy:** “dlib” provides one of the most accurate face detection methods.
2. **Speed:** Despite being accurate, the library is efficient enough for real-time face detection.
3. **Flexibility:** “dlib” offers both traditional machine learning methods (HOG) and deep learning-based methods (CNN) for face detection, enabling a balance between performance and speed depending on the hardware.

1.2. **Face Recognition Model (VGGFace - ResNet50):** For face recognition, the VGGFace model, specifically using the ResNet50 architecture, was selected. VGGFace is a powerful pre-trained model for face recognition that was trained on a large dataset of celebrity faces and is capable of extracting robust features from face images. The ResNet50 architecture, a deep residual network, provides a good trade-off between speed and accuracy.

**Why VGGFace with ResNet50?**

1. **Pre-trained on a large dataset:** VGGFace has been trained on millions of face images, ensuring high accuracy and generalizability.
2. **Robust Feature Extraction:** It extracts robust and distinctive features that help in identifying individuals with high precision.
3. **Optimized for Speed:** ResNet50, while deep, is optimized for faster inference without compromising too much on performance.

By using this model, we ensure that face recognition works effectively even in challenging real-time environments.

1.3. **Database Integration (MySQL):** For storing and managing face embeddings, MySQL was chosen as the database management system. MySQL is a widely-used, reliable relational database that is well-suited for handling structured data, such as face embeddings, which are stored as large arrays (binary large objects or BLOBs). The database allows for fast and efficient storage, retrieval, and manipulation of these embeddings, which are essential for comparison during face recognition.

**Why MySQL?**

1. **Reliability:** MySQL has a long-standing reputation for being a stable and robust database system.
2. **Efficient Data Handling:** Storing face embeddings as BLOBs enables efficient retrieval and comparison, ensuring smooth recognition.
3. **Ease of Connectivity:** Python's mysql-connector library provides a simple and effective way to interact with MySQL, making integration into the Python-based recognition system seamless.

The database also supports CRUD (Create, Read, Update, Delete) operations, making it easy to manage the stored embeddings and update records as needed.

1.4. **Real-Time Face Recognition System (OpenCV):** To implement real-time face recognition, OpenCV was used. OpenCV is a powerful library for computer vision and image processing tasks. It allows us to handle the live webcam feed, perform face detection, process frames for recognition, and display results efficiently.

**Why OpenCV?**

1. **Real-time Processing:** OpenCV is optimized for real-time computer vision tasks, making it an ideal choice for live face detection and recognition.
2. **Integration with dlib:** OpenCV works seamlessly with dlib, which is essential for face detection and tracking in real-time.
3. **Cross-platform:** OpenCV is cross-platform and works across various operating systems, ensuring that the system works on most devices.

OpenCV's VideoCapture function is used to capture video frames from the webcam, and the frames are then processed to detect faces and compare them with the stored embeddings in the database for recognition.

1.5. **Graphical User Interface (GUI) - Tkinter:** The Tkinter library was used to build the graphical user interface (GUI) of the face recognition system. Tkinter is the standard GUI toolkit for Python and is well-suited for creating simple, interactive applications.

**Why Tkinter?**

1. **Simplicity:** Tkinter is simple to use and allows quick development of desktop GUI applications without the need for complex frameworks.
2. **Lightweight:** Tkinter is a lightweight library that does not require much overhead, making it a good choice for a system that needs to run smoothly without demanding too many resources.
3. **Ease of Integration:** Tkinter integrates easily with Python and other libraries such as OpenCV and MySQL, enabling the creation of interactive components like buttons, labels, and image displays.

The Tkinter GUI allows users to interact with the face recognition system, displaying live webcam feeds, recognition results, and providing controls to upload images and manage the database.

**2. Implementation Details**

2.1. **Face Embedding Collection and Database Integration:** The face embeddings are extracted using the VGGFace model, which processes the face images and converts them into a compact numerical representation. These embeddings are then stored in the MySQL database as binary large objects (BLOBs) for later use during recognition. The database enables efficient comparison of embeddings during the recognition process, ensuring that the system can accurately identify individuals.

2.2. **Real-Time Face Recognition:** The system uses the webcam feed to continuously capture frames, detect faces, extract embeddings, and compare them against the stored embeddings in the database. If a match is found, the name of the recognized person is displayed; otherwise, the system returns "Unknown." This process is repeated for each frame, providing real-time recognition.

2.3. **Tkinter GUI Integration:** The Tkinter-based GUI provides an intuitive interface for interacting with the system. The live webcam feed is displayed in a Tkinter window, and the user can start recognition, upload new images, and view the stored face embeddings in the database. Buttons and labels allow the user to interact with the system efficiently.

3. Conclusion and Future Work

The system successfully integrates face detection, recognition, and database management into a cohesive solution for real-time face recognition. The use of dlib for face detection, VGGFace for recognition, OpenCV for real-time processing, MySQL for data storage, and Tkinter for the GUI creates a robust and efficient application.

**Future work could include:**

1. **Improving the Database Structure:** Implementing more sophisticated database management techniques to handle larger datasets.
2. **Enhancing Accuracy:** Using a more advanced or custom-trained face recognition model to improve accuracy under challenging conditions.
3. **Scalability:** Implementing distributed systems to scale the recognition system across multiple devices or users.