Assiut University
Faculty of Computers and Information

Application to Help the Blind

(Eyes Mate)

Graduation Project
Academic Year 2023-2024

July 2, 2024

Team Members

Islam Abd-Elhady Hassanein Mohamed	162020106
Enas Ragab Abdel-Latif Mohamed	162020150
Mariam Tarek Saad Mohamed	162020605
Rawan Mohammed Ahmed Alam	162020234

Supervisor

Dr. Abdel-Rahman Hedar

2024

Contents

Chapter 1

Project Proposal

1.1 Project Abstract

Developing a mobile application using machine learning to assist the blind in their daily lives. The application can use the mobile phone's camera to capture images or videos of the surrounding environment and then use machine learning algorithms to identify and describe objects in the image or video. The application can then convert the description into audio form and play it back to the user. This can be extremely helpful for the blind in identifying objects around them, reading text, and even recognizing faces and currency detection. In addition to object recognition, the application can also use machine learning to measure distances from the phone to objects in the environment. This can be useful for the blind in navigating their surroundings, both indoors and outdoors to avoid consequences. The application can also use machine learning to recognize the number of people in the environment, which can be helpful for the blind in social situations to make the application more personalized, it can also be trained to recognize the faces of the blind person's relatives or friends. This can help the blind person identify who is around them and can also be used to alert the blind person if someone they know is nearby.

1.2 Project Objectives

1. **Develop Object Recognition Functionality:**
    - Implement machine learning algorithms to enable the mobile application to accurately identify and describe objects captured through the mobile phone's camera.

- Ensure the application can recognize a diverse range of objects, including common items, text, faces, and currency.

2. **Audio Description Conversion:**
   - Integrate a feature that converts the identified objects' descriptions into audio format.
   - Develop an efficient and clear audio playback system to relay information to the blind user.
3. **Distance Measurement Capability:**
   - Utilize machine learning techniques to measure distances from the mobile phone to objects in the surrounding environment.
   - Implement a reliable distance measurement system to assist blind users in navigating both indoor and outdoor spaces.
4. **Social Interaction Enhancement:**
   - Incorporate machine learning algorithms to recognize the number of people in the environment.
   - Develop features that assist blind users in social situations by providing information about the presence of others around them.
5. **Personalization through Facial Recognition:**
   - Implement facial recognition technology to allow the application to identify the faces of the blind person's relatives and friends.
   - Develop a training system to personalize the application for individual users, enhancing the recognition accuracy for known faces.
6. **Real-time Alerts and Notifications:**
   - Enable the application to provide real-time alerts when recognized faces of friends or relatives are detected in the vicinity.
   - Implement a notification system to alert the blind user of the presence of someone they know, enhancing situational awareness.
7. **User-Friendly Interface:**
   - Conduct rigorous testing to ensure the accuracy and reliability of machine learning algorithms in various real-world scenarios.

- Optimize the application's performance to minimize latency and improve responsiveness.

8. **Documentation and User Guides:**
   - Provide comprehensive documentation and user guides to assist both developers and end-users in understanding the functionalities and operation of the application.
9. **Security and Privacy:**
   - Implement robust security measures to protect user data, especially in the case of facial recognition and personalization features.
   - Ensure compliance with privacy regulations and prioritize the confidentiality of user information.
10. **Collaboration and Outreach:**
    - Collaborate with blind and visually impaired communities for feedback and insights during the development process.
    - Conduct outreach programs to raise awareness about the application and its potential benefits.
11. **Scalability and Future Expansion:**
    - Design the application architecture with scalability in mind to accommodate future updates, additional features, and improvements.
    - Consider potential partnerships or collaborations for the continuous enhancement of the application.

1.3 Approaches and Methodology

1. **Data Collection and Preparation:**
   - Collect a diverse dataset containing images and videos representing various objects, text, faces, and social scenarios.
   - Annotate and preprocess the dataset to ensure accurate training and testing of machine learning models.
2. **Object Recognition Model Development:**

- Choose a suitable deep learning architecture for object recognition, considering performance and efficiency.
- Train the model using the annotated dataset, fine-tuning for specific categories such as objects, text, faces, and currency.

3. **Audio Conversion Module:**
   - Develop a module to convert the identified objects' descriptions into audio format.
   - Utilize text-to-speech (TTS) technology for generating clear and natural-sounding audio descriptions.
4. **Distance Measurement Algorithm:**
   - Implement machine learning algorithms to measure distances accurately using data from the mobile phone's camera.
   - Optimize the algorithm to work in real-time, considering various environmental conditions and lighting.
5. **Social Interaction Recognition:**
   - Design and train a machine learning model to recognize the number of people in the environment.
   - Implement algorithms that can adapt to different social scenarios and crowd sizes.
6. **Facial Recognition and Personalization:**
   - Choose a facial recognition algorithm and train it to recognize the faces of the blind person's relatives and friends.
   - Implement a training system for users to personalize the application with known faces.
7. **Real-time Alerts and Notifications:**
   - Develop a system for real-time alerts when recognized faces of friends or relatives are detected.
   - Integrate a notification mechanism to inform the blind user about the presence of someone they know nearby.
8. **User Interface and Accessibility Features:**

- Design an intuitive and accessible user interface tailored for blind users.
- Implement voice commands, haptic feedback, and other accessibility features to enhance usability.

9. **Testing and Validation:**
    - Conduct thorough testing of each module and the integrated system using real-world scenarios.
    - Validate the accuracy, reliability, and responsiveness of the machine learning models in diverse environments.
10. **Optimization and Performance Enhancement:**
    - Optimize algorithms and code for efficiency, considering limited resources on mobile devices.
    - Address any latency issues and ensure smooth performance during real-time object recognition and distance measurement.
11. **Privacy and Security Measures:**
    - Implement robust security protocols to protect user data, especially in facial recognition and personalization features.
    - Ensure compliance with privacy regulations and prioritize the security of sensitive information.
12. **User Feedback and Iterative Development:**
    - Engage with blind and visually impaired communities for feedback on usability and functionality.
    - Iteratively improve the application based on user input and emerging technologies.
13. **Documentation and Knowledge Transfer:**
    - Document the entire development process, including algorithms, models, and user guides.
    - Facilitate knowledge transfer to future developers or stakeholders for ongoing maintenance and updates.

1.4 Project Plan and Management

1.4.1 Project Scope Definition

- **Objective:** Develop a mobile application using machine learning to assist the blind in their daily lives.
- **Scope:**
  - Image processing for object recognition.
  - Distance measurement functionality.
  - People recognition and personalized alerts.
- **Deliverables:** Functional mobile application compatible with Android and iOS devices.

1.4.2 Project Timeline [1]

- **Phase 1: Planning and Requirements Gathering (2 weeks)**
  - Define project scope, objectives, and deliverables.
  - Gather requirements from potential users and stakeholders.
  - Develop a detailed project plan and schedule.
- **Phase 2: Design and Architecture (4 weeks)**
  - Design system architecture based on decomposition description.
  - Define interfaces between modules and submodules.
  - Create mockups and prototypes for user interface design.
- **Phase 3: Implementation (8 weeks)**
  - Develop and integrate modules according to the design.
  - Implement machine learning models for object and face recognition.
  - Test individual modules and functionalities.
- **Phase 4: Testing and Quality Assurance (4 weeks)**
  - Conduct unit testing, integration testing, and system testing.
  - Perform usability testing with blind users and gather feedback.
  - Ensure application meets accessibility standards.

- **Phase 5: Deployment and Maintenance (Ongoing)**
  - Deploy the application to app stores (Google Play Store, Apple App Store).
  - Monitor user feedback and address any issues or bugs.
  - Provide regular updates and maintenance to ensure the application's functionality and compatibility with new device versions.

1.4.3 Resource Allocation

- **Team Composition:**
  - Project Manager
  - Software Developers
    - Frontend
    - Backend
  - Machine Learning Engineers
  - UI/UX Designers
  - Quality Assurance/Testers
- **Tools and Technologies:**
  - Programming Language:
    - Dart for Flutter
    - Python for Machine Learning Models
  - Machine Learning Frameworks:
    - TensorFlow
    - TensorFlow Lite for Flutter
    - PyTorch
    - scikit-learn
    - OpenCV
    - YOLOv5
  - Backend Frameworks:
    - Flask
  - Development Tools:
    - IDE:

        - Visual Studio Code (with Flutter and Dart plugins)
        - Jupyter Notebook
    * Version Control:
        - Git
- Collaboration Tools:
    * Project Management:
        - Jira
        - Asana
    * Communication:
        - Slack
        - Microsoft Teams

1.4.4 Risk Management

- **Identification:** Identify potential risks such as technical challenges, resource constraints, and user adoption issues.
- **Mitigation:** Develop contingency plans for identified risks, allocate additional resources if necessary, and maintain open communication channels with stakeholders to address concerns promptly.

1.4.5 Progress Monitoring and Reporting

- **Regular Meetings:** Conduct regular project meetings to review progress, discuss any issues, and adjust the plan as needed.
- **Status Reports:** Provide regular status reports to stakeholders, highlighting achievements, milestones, and any deviations from the plan.
- **Feedback Mechanism:** Establish a feedback mechanism to gather input from blind users throughout the development process and incorporate their suggestions into the application.

1.4.6 Stakeholder Engagement

- **Engagement Strategy:** Keep stakeholders informed and engaged throughout the project lifecycle through regular updates, demonstrations, and opportunities for input.

- **User Involvement:** Involve blind users in the development process to ensure the application meets their needs and preferences effectively.

1.4.7 Documentation and Knowledge Management

- **Documentation:** Maintain comprehensive documentation including design documents, user manuals, and technical specifications.
- **Knowledge Sharing:** Facilitate knowledge sharing among team members through regular meetings, training sessions, and collaborative tools to ensure continuity and scalability of the project.

Chapter 2

Project Analysis

2.1 Introduction

2.1.1 Purpose

The product described in this document is a mobile application aimed at assisting blind individuals in their daily lives using machine learning technology. The purpose of the application is to utilize the mobile phone's camera to capture images or videos of the user's surroundings and then employ machine learning algorithms to identify and describe objects within those images or videos. Additionally, the application is intended to convert these descriptions into audio form, enabling blind users to perceive and understand their environment. Furthermore, the application aims to provide functionalities such as distance measurement, recognizing the number of people in the environment, and even identifying faces of the blind person's relatives or friends.

2.1.2 Document Conventions [2]

This section describes the standards and typographical conventions used throughout this Analysis Report to ensure clarity and consistency. The following conventions have been adhered to:

2.1.2.1 Typographical Conventions

- **Bold Text**: Used to emphasize key points, section headers, and important terminology.
- *Italic Text*: Used for special terms, foreign words, and names of documents.

- `Monospaced Text`: Used for code snippets, file names, and commands.
- ALL CAPS: Used for acronyms and abbreviations.

2.1.2.2 Font and Formatting

- **Main Text**: The main body of the document is written in Times New Roman, 12-point font, with 1.5 line spacing for readability.
- **Headings**: Section headings are formatted in Arial, with varying sizes according to the level of heading (e.g., 16-point for primary headings, 14-point for secondary headings).
- **Tables and Figures**: Tables and figures are labeled consecutively (e.g., Table 1, Table 2; Figure 1, Figure 2) and include a descriptive title and caption.

2.1.2.3 Requirement Statements

- Each requirement statement is uniquely identified by a requirement ID (e.g., REQ-1, REQ-2).
- Priorities for higher-level requirements are assumed to be inherited by detailed requirements unless explicitly stated otherwise.
- Every requirement statement has its own priority, which is indicated in the requirement description.

2.1.2.4 Prioritization

- **High Priority**: Critical requirements that must be implemented for the project to be successful.
- **Medium Priority**: Important requirements that should be implemented but can be deferred if necessary.
- **Low Priority**: Desirable requirements that can be implemented if time and resources permit.

2.1.2.5 Terminology

- Consistent terminology is used throughout the document. Specific terms and definitions are provided in the glossary section.

- Acronyms and abbreviations are defined upon first use and included in the list of acronyms.

2.1.2.6 Assumptions and Dependencies

- Assumptions made during the analysis and design phases are clearly stated.
- Dependencies on external systems or components are identified and described in detail.

By adhering to these conventions, the document aims to provide a clear, consistent, and professional analysis report that facilitates understanding and communication among all stakeholders.

2.1.3 Intended Audience and Reading Suggestions

2.1.3.1 Intended Audience

This document is intended for a diverse audience, including:

- **Developers:**
  - **Focus Areas:** Technical components, algorithms, user interface design, and implementation details.
  - **Suggested Reading Sequence:** Start with the **Technical Components** section to understand the core technologies and algorithms. Follow with the **Key Features** to see how these components integrate into the application. Finish with the **Use Cases** to understand practical applications and requirements.
- **Project Managers:**
  - **Focus Areas:** Project overview, key features, use cases, benefits, and intended impact.
  - **Suggested Reading Sequence:** Begin with the **Overview** to grasp the project's goals and scope. Proceed to the **Key Features** and **Use Cases** sections to understand functionality and practical applications. Conclude with the **Benefits** to see the project's value and impact.
- **Marketing Staff:**

  - **Focus Areas:** Benefits, target audience, and application value propositions.
  - **Suggested Reading Sequence:** Start with the **Overview** and **Benefits** sections to understand the application's purpose and advantages. Then, move to the **Use Cases** to see how the application benefits the users in real-life scenarios.

- **Users:**
  - **Focus Areas:** Key features, use cases, and personalization options.
  - **Suggested Reading Sequence:** Begin with the **Key Features** section to learn about the application's functionalities. Follow with the **Use Cases** to understand how the application can be used in daily life. Conclude with the **Personalization** options to see how the app can be tailored to individual needs.

- **Testers:**
  - **Focus Areas:** Features to be tested, use cases, and technical components.
  - **Suggested Reading Sequence:** Start with the **Key Features** section to identify areas for testing. Proceed to the **Technical Components** to understand the underlying technologies. Finish with the **Use Cases** to design test scenarios that reflect real-world applications.

- **Documentation Writers:**
  - **Focus Areas:** Comprehensive understanding of all sections for accurate documentation.
  - **Suggested Reading Sequence:** Read through the entire document, starting with the **Overview** to understand the project's scope. Continue through the **Key Features**, **Technical Components**, and **Use Cases** to gather detailed information. Conclude with the **Benefits** to capture the application's value proposition.

2.1.3.2 Document Organization

- **Overview:** Provides a high-level summary of the project, including its purpose, goals, and key features.

- **Key Features:** Details the main functionalities of the application, such as object recognition, distance measurement, people recognition, and face recognition.
- **Technical Components:** Explains the technologies and algorithms used in the application, including image and video capture, machine learning algorithms, audio feedback, and user interface design.
- **Use Cases:** Describes practical scenarios where the application can assist blind users, highlighting the real-world benefits and applications.
- **Benefits:** Summarizes the advantages of the application, emphasizing its impact on the independence and quality of life for blind individuals.

2.1.3.3 Reading Suggestions

For a comprehensive understanding, readers are encouraged to start with the **Overview** to get a general sense of the project. Depending on their role, they can then delve into the sections most relevant to their interests and responsibilities:

- **Developers:** Focus on **Technical Components** and **Key Features**.
- **Project Managers:** Focus on **Overview**, **Key Features**, and **Benefits**.
- **Marketing Staff:** Focus on **Overview**, **Benefits**, and **Use Cases**.
- **Users:** Focus on **Key Features**, **Use Cases**, and **Personalization**.
- **Testers:** Focus on **Key Features**, **Technical Components**, and **Use Cases**.
- **Documentation Writers:** Read the entire document for a thorough understanding.

2.1.4 Product Scope

2.1.4.1 Description

This project involves developing a mobile application that utilizes machine learning to assist blind individuals in their daily lives. By leveraging the mobile phone's camera, the application captures images or videos of the surrounding environment and applies machine learning algorithms to identify

and describe objects within the captured media. These descriptions are then converted into audio format and played back to the user, providing real-time assistance in identifying objects, reading text, recognizing faces, and detecting currency.

2.1.4.2 Purpose

The primary purpose of the application is to enhance the independence and quality of life for blind individuals by providing them with a reliable and efficient tool to navigate and interact with their environment. The application aims to bridge the gap between the blind and their surroundings, offering a comprehensive solution that addresses various daily challenges.

2.1.4.3 Benefits

- **Object Identification:** Helps blind users identify everyday objects, read text, and detect currency, promoting self-reliance in daily tasks.
- **Navigation Assistance:** Measures distances to objects, aiding in safe navigation both indoors and outdoors.
- **Social Interaction:** Recognizes the number of people in the environment, facilitating better social interactions.
- **Personalization:** Can be trained to recognize faces of relatives and friends, providing personalized alerts and enhancing social awareness.

2.1.4.4 Objectives and Goals

- **Accuracy and Reliability:** Utilize state-of-the-art machine learning algorithms to ensure high accuracy in object recognition and description.
- **User-Friendly Interface:** Develop an intuitive and accessible user interface tailored to the needs of blind users, incorporating voice commands and gestures for easy interaction.
- **Real-Time Feedback:** Provide instantaneous audio feedback to ensure timely assistance in various situations.
- **Customization:** Allow users to personalize the application by training it to recognize specific faces and adjusting settings to fit individual preferences.

- **Scalability:** Design the application to be scalable, accommodating future enhancements and additional features as needed.

2.1.4.5 Corporate Goals and Business Strategies

The development of this application aligns with corporate goals of promoting inclusivity and accessibility through innovative technology solutions. By addressing the specific needs of blind individuals, the application supports business strategies focused on enhancing user experience and expanding the market reach to underserved communities. This project underscores a commitment to social responsibility and leveraging technology to improve the quality of life for all users.

2.2 Overall Description

2.2.1 Product Perspective

The mobile application being developed is a new, self-contained product designed to assist blind individuals in their daily lives through the use of advanced machine learning techniques. This application is not a follow-on member of a product family nor a replacement for any existing systems; instead, it represents an innovative solution aimed at enhancing the independence and quality of life for blind users by leveraging modern technologies.

2.2.1.1 Context and Origin

The application emerged from the recognition of the numerous challenges faced by blind individuals in interacting with their environment. The goal is to create a comprehensive tool that addresses several key aspects of daily life, such as object recognition, text reading, currency identification, and face recognition. This project aims to integrate these functionalities into a single, easy-to-use mobile application.

2.2.1.2 Overall System and Subsystems

The product is a standalone mobile application that operates independently but can be integrated with various subsystems for enhanced functionality. The primary components and their interconnections are as follows:

- **Mobile Application (Frontend)**

  - **User Interface (UI)**: Developed using Flutter, designed to be simple and accessible.
  - **Gesture Control System**: Enables users to navigate through features using swipe gestures.
- **Machine Learning Models (Core Functionality)**
  - **Object Recognition Model**: Identifies and describes objects captured by the camera.
  - **Distance Measurement Algorithm**: Calculates the distance between the phone and surrounding objects.
  - **Text Reading (OCR) Model**: Recognizes text in images and converts it to speech.
  - **Currency Recognition Model**: Identifies various denominations of currency.
  - **Face Recognition Model**: Detects and recognizes known faces and counts people in the environment.
- **Backend Services**
  - **Flask**: Acts as a backend server that captures, processes, and streams real-time video with face detection and text-to-speech capabilities to the Flutter client.
  - **Firebase Authentication**: Manages user authentication and secure access.
  - **Firebase Database**: Stores user data, including known faces for personalized features.
  - **Firebase Storage**: Handles storage of images and other media.
- **Audio Feedback System**
  - **Text-to-Speech (TTS)**: Converts text descriptions and recognition results into audio feedback for the user.

2.2.1.3 Interfaces and Integration

The application interfaces with the following external systems to provide a seamless user experience:

- **Device Camera**: Captures images and videos for processing by the machine learning models.
- **Microphone and Speakers**: Used for audio feedback and interaction.
- **Internet Connection**: Required for initial model training, updates, and syncing data with the backend.

Below is a simple diagram illustrating the major components and their interconnections:


User Interface
Camera Subsystem
Machine Learning Subsystem
ObjectRecognition & Distance Measurement Subsystem
Text Reading Subsystem
Currency Recognition Subsystem
Face Recognition , Person Counting & Identifing a Person's Face Subsystem
Audio Subsystem (Text-to-Speech)
Face Recognition, Person Counting
Identifing a Person's Face
Database (User Data & Trained Models)
Audio Subsystem (Text-to-Speech)
External Interface (Camera,Mic,Speaker)
External Interface (Camera,Mic,Speaker)

The system is designed to utilize various subsystems to provide a comprehensive user experience, leveraging machine learning and audio feedback to assist users with different functionalities. Below is a detailed explanation of each component and its role in the system.

1. **User Interface**

   The user interface serves as the primary point of interaction for the users. It allows users to access and control the different features of the application.

2. **Camera Subsystem**

   The camera subsystem is responsible for capturing images or videos as input. It sends this visual data to the machine learning subsystem for processing.

3. **Machine Learning Subsystem**

   The machine learning subsystem is the core component that processes the input data from the camera subsystem. It uses various models to analyze and interpret the data, directing it to the appropriate subsystems for specific tasks.

4. **Object Recognition & Distance Measurement Subsystem**

   This subsystem identifies and describes objects in real-time and measures the distance between the camera and the objects. The information is then sent to the audio subsystem to provide auditory feedback to the user.

5. **Text Reading Subsystem**

   The text reading subsystem extracts text from images or videos and converts it into speech. The audio subsystem handles this conversion and provides the feedback to the user.

6. **Currency Recognition Subsystem**

   This subsystem identifies different currencies from the captured images or videos. The recognized information is then processed by the audio subsystem to inform the user about the type and value of the currency.

7. **Face Recognition, Person Counting & Identifying a Person's Face Subsystem**

   This subsystem performs three main tasks:

- Recognizes and counts the number of people in the environment.
- Identifies specific faces, such as those of the user's relatives or friends.
- Trains models to recognize and store user-specific data.

The subsystem works closely with the database to retrieve and store user data and trained models.

8. **Audio Subsystem (Text-to-Speech)**

   The audio subsystem converts textual information into speech, providing auditory feedback to the user. It integrates with other subsystems to relay information such as object descriptions, text readings, and currency values.

9. **External Interface (Camera, Mic, Speaker)**

   The external interface consists of hardware components like the camera, microphone, and speaker. These components facilitate the input and output processes of the system:

   - The camera captures visual data.
   - The microphone can be used for voice commands.
   - The speaker provides auditory feedback to the user.

10. **Database (User Data & Trained Models)**

    The database stores user-specific data and trained models necessary for the face recognition subsystem. It ensures that the system can recognize and provide personalized feedback to users.

This structured interaction between the subsystems ensures that the application provides accurate, real-time assistance to users, leveraging advanced machine learning techniques and user-friendly audio feedback.

2.2.1.4 Relationship to Larger System

While the application itself is a standalone product, it can serve as a component within a larger ecosystem of assistive technologies for the blind. For instance, it can integrate with other accessibility tools and services, such as navigation aids, smart home devices, and personal health monitoring systems, to provide a more comprehensive support system for blind individuals. The

backend services can also facilitate data sharing and synchronization with other applications, enhancing the overall functionality and user experience.

By situating the product within this context, we ensure it not only meets the immediate needs of blind users but also has the potential to integrate into broader assistive technology solutions, providing scalable and extensible support.

2.2.2 Product Functions

The mobile application must perform the following major functions to assist blind individuals effectively. These functions are designed to be intuitive and accessible, providing critical information through audio feedback.

- **Object Recognition and Description**
  - Identify objects in the environment using the phone's camera.
  - Convert object descriptions into audio feedback for the user.
- **Distance Measurement**
  - Measure the distance between the phone and objects in the environment.
  - Provide audio feedback on distances to aid in navigation.
- **Text Reading**
  - Recognize and read aloud text from images or videos captured by the camera.
- **Currency Recognition**
  - Identify various denominations of currency.
  - Convert currency information into audio feedback.
- **Face Recognition and Person Counting**
  - Detect and recognize known faces.
  - Count the number of people in the environment.
  - Provide audio feedback about the faces of the blind person's relatives or friends.
- **Gesture Control System**

  - Enable navigation through different features using swipe gestures (up, down, left, right).

- **Audio Feedback System**
  - Provide clear and timely audio feedback for all features.
- **User Authentication**
  - Allow users to log in and create accounts.
  - Securely store and manage user data, including personalized features.

2.2.3 User Classes and Characteristics

2.2.3.1 Primary Users: Blind Individuals

- **Frequency of Use**: Frequent, as they rely on the application daily to navigate and interact with their environment.
- **Subset of Product Functions Used**: All major functions including object recognition, distance measurement, text reading, currency recognition, and face recognition.
- **Technical Expertise**: Low. The application is designed to be effortlessly accessible, requiring minimal technical expertise.
- **Security or Privilege Levels**: Standard user level. However, personalized settings such as recognized faces need to be securely managed.
- **Educational Level**: Varies, but the application should be usable regardless of the user's educational background.
- **Experience**: Experience with other accessibility tools may vary. The application should be intuitive enough for users with no prior experience with similar technologies.

2.2.3.2 Secondary Users: Friends and Relatives

- **Frequency of Use**: Occasional, primarily during initial setup and when updating personalized settings.
- **Subset of Product Functions Used**: Functions related to adding or updating recognized faces and assisting with initial setup.

- **Technical Expertise**: Moderate. They may need to assist with setup and configuration, requiring some familiarity with smartphone applications.
- **Security or Privilege Levels**: Elevated privilege levels for managing personalized settings and recognized faces.
- **Educational Level**: Varies, but generally higher than the primary users due to their role in setup and configuration.
- **Experience**: Likely familiar with general smartphone usage and potentially other accessibility features.

2.2.3.3 Tertiary Users: Technical Support and Developers

- **Frequency of Use**: Infrequent, primarily for maintenance, updates, and troubleshooting.
- **Subset of Product Functions Used**: All functions, with additional access to diagnostic and configuration tools.
- **Technical Expertise**: High. Professional expertise in mobile application development, machine learning, and accessibility features.
- **Security or Privilege Levels**: Highest privilege levels, including access to system settings, logs, and user data for troubleshooting purposes.
- **Educational Level**: Advanced, typically with specialized training in relevant technical fields.
- **Experience**: Extensive experience with mobile development, machine learning applications, and user support.

2.2.3.4 Most Important User Classes

The most important user classes for this product are:

- **Primary Users: Blind Individuals**
  - These users are the main beneficiaries of the application, relying on it daily for navigation and interaction with their environment.
  - Ensuring the application is highly accessible, intuitive, and reliable is crucial for this user class.

- **Secondary Users: Friends and Relatives**
  - They play a critical role in setting up and maintaining the application for the primary users.
  - Their ability to easily interact with the application and assist in personalization enhances the overall user experience for blind individuals.

2.2.4 Operating Environment

2.2.4.1 Hardware Platform

The mobile application is designed to operate on modern smartphones equipped with a camera. The minimum hardware requirements include:

- A rear-facing camera with a resolution of at least 8 megapixels for accurate object recognition and text reading.
- A processor capable of handling machine learning algorithms efficiently, preferably a multi-core processor.
- At least 2GB of RAM to ensure smooth operation of the application.
- Storage space of at least 100MB for the application, with additional space required for user data and machine learning models.

2.2.4.2 Operating System and Versions

The application is developed using Flutter, ensuring cross-platform compatibility. It is designed to run on:

- Android OS version 8.0 (Oreo) and above.
- iOS version 11.0 and above.

2.2.4.3 Software Components and Applications

The following software components and applications are necessary for the operation of the mobile application:

- **Flutter Framework**: The application is built using Flutter, which provides a flexible and robust framework for cross-platform mobile development. It ensures the application can run seamlessly on both Android and iOS devices.

- **TensorFlow Lite**: TensorFlow Lite is used for deploying machine learning models on mobile devices. It provides optimized machine learning inference, allowing the application to perform object recognition, text reading, currency recognition, and face recognition efficiently.
- **Camera Access**: The application requires access to the smartphone's camera to capture images and videos for processing. It will utilize the device's camera API to interact with the camera hardware.
- **Audio Playback**: The application needs audio playback capabilities to convert text and object descriptions into speech. This is essential for providing feedback to the blind user.
- **User Authentication**: The application includes a feature for user authentication, allowing users to log in and create accounts. This is implemented using secure authentication protocols to ensure user data privacy and security.
- **Internet Connectivity**: While the core functionalities of the application can operate offline, internet connectivity is required for certain features such as initial model downloads, updates, and account creation or login.

2.2.4.4 Coexisting Software

The application must coexist peacefully with the following software components typically found on modern smartphones:

- Default camera application and any other camera-related services.
- Accessibility services provided by the operating system to enhance usability for visually impaired users.
- Audio services for text-to-speech conversion and playback.
- Network services to support internet connectivity for user authentication and updates.

By adhering to these operating environment requirements, the mobile application will provide a robust and reliable tool to assist blind individuals in their daily lives, leveraging the power of machine learning and modern mobile technology.

2.2.5 Design and Implementation Constraints

- **Corporate or Regulatory Policies**:
  - **Accessibility Standards**: The application must adhere to accessibility guidelines such as the Web Content Accessibility Guidelines (WCAG) to ensure it is usable by visually impaired individuals.
  - **Data Privacy Regulations**: Compliance with data protection regulations such as the General Data Protection Regulation (GDPR) is mandatory to protect user data and ensure privacy.
- **Hardware Limitations**:
  - **Timing Requirements**: Real-time processing is essential for features such as object recognition and text reading. The application must be optimized to deliver quick and accurate results without significant delays.
  - **Memory Requirements**: The application should be efficient in memory usage to operate smoothly on devices with limited RAM, ensuring it does not consume excessive resources.
- **Interfaces to Other Applications**:
  - **Camera Interface**: Integration with the device's camera API is required to capture images and videos.
  - **Audio Services**: The application must interact with the device's audio services for text-to-speech conversion and playback.
- **Specific Technologies, Tools, and Databases**:
  - **Flutter**: The application will be developed using Flutter for cross-platform compatibility.
  - **TensorFlow Lite**: Machine learning models will be implemented using TensorFlow Lite for efficient on-device inference.
  - **Firebase**: Firebase will be used for authentication and database management to store user data securely.
- **Parallel Operations**: The application must support parallel operations, ensuring that multiple features can run simultaneously without performance degradation. For example, it should handle object recognition while simultaneously reading text or recognizing faces.

- **Language Requirements**: The primary programming language for the application will be Dart, used within the Flutter framework. Machine learning models will be developed and trained using Python and TensorFlow.
- **Communications Protocols**:
  - **HTTP/HTTPS**: Secure communication for data exchange between the application and server.
  - **Bluetooth/Wi-Fi**: Potential use for connectivity with external devices or for enhanced location services.
- **Security Considerations**:
  - **Data Encryption**: All user data, both at rest and in transit, must be encrypted to prevent unauthorized access.
  - **Secure Authentication**: Implementation of secure authentication mechanisms to ensure only authorized users can access the application.
  - **Regular Updates**: Regular updates to address security vulnerabilities and ensure the application remains secure against emerging threats.
- **Design Conventions or Programming Standards**:
  - **Coding Standards**: Adherence to coding standards and best practices for Dart and Python.
  - **Documentation**: Comprehensive documentation for all code and features to facilitate maintenance and future development.
  - **Testing**: Rigorous testing protocols to ensure the application is free from bugs and performs as expected.

2.2.6 User Documentation

- **Auditory Cues**:
  - Users will hear auditory cues upon opening the app, providing them with instant guidance.
- **Unique Motion Patterns**:

  - Each specific feature or function within the app is associated with a unique motion pattern. This lets users know which features are activated based on the signals they receive when they open the app.

- **Accessibility Design**:
  - Sound prompts and motion patterns are designed with accessibility in mind, to meet the needs of users with visual impairments or those who might benefit from multi-sensory feedback.

2.2.7 Assumptions and Dependencies

- **Assumptions**
  - **Availability of Compatible Hardware**: It is assumed that users will have access to smartphones with cameras that meet the minimum quality standards required for image recognition and processing tasks.
  - **Stable Internet Connectivity**: It is assumed that users will have intermittent access to a stable internet connection to download updates, sync data, and utilize advanced features.
  - **User Proficiency with Smartphones**: It is assumed that primary users, despite visual impairments, will have basic proficiency in using smartphones and will be able to navigate the application with the provided auditory and motion pattern feedback.
  - **Accessibility Standards**: It is assumed that the auditory cues, sound prompts, and motion patterns designed for accessibility will be effective for the majority of visually impaired users.
- **Dependencies**
  - **Third-Party APIs and Libraries**: The project depends on various third-party APIs and libraries for machine learning and image processing, including TensorFlow Lite. Any changes or deprecations in these components could impact the project.
  - **Firebase Services**: The application relies on Firebase for authentication, database, and storage services. Any changes in Firebase's service offerings or pricing could affect the project.
  - **Operating Systems**: The application depends on the Android operating system (versions 8.0 and above) and the iOS operating system

(versions 11.0 and above). Any changes or updates to these OS platforms could impact the application's functionality.

- **Hardware Compatibility**: The application assumes compatibility with a range of smartphone hardware. Variations in camera quality and sensor accuracy across different devices could affect performance.

2.3 External Interface Requirements

2.3.1 User Interfaces

The user interfaces of the mobile application are designed to be intuitive and accessible, catering specifically to the needs of blind users. The following logical characteristics and standards are followed:

2.3.1.1 Screen Layouts

Simple and consistent layouts of swiping to any model by listening to voice command for easy navigation. The use of high-contrast colors ensures readability for users with partial vision.

2.3.1.2 Loading Screen

It's the first screen in our app, it helps in:

- **Time Management:**
  - **Resource Loading:** It allows time for the application to load necessary resources, such as images, data, or models, before the user can interact with the main content. This is crucial for ensuring that the app functions smoothly once it is fully loaded.
  - **Initialization:** It gives the application time to initialize essential processes, such as setting up machine learning models, connecting to backend services.
- **Branding & Technical Benefits:**
  - **Branding:** It provides an opportunity to reinforce branding by displaying the app's logo, colors, or tagline. This helps in creating a consistent brand experience.

- **Error Handling:** It provides a buffer period where the app can handle any initial errors or connectivity issues before presenting the main interface to the user. This can improve the overall stability and reliability of the app.
- **Data Synchronization:** It allows the app to synchronize data with the server or local storage, ensuring that the user has the most up-to-date information when they start using the app.

Figure 2.1: Page 1-Loading Page

2.3.1.3 Main Screen

It's the second screen in our App, displays main functionalities of out Application such as object recognition, distance measurement, text reading, currency recognition, Face recognition, person counting & identifiying person face from user relative and friends. When the application opened this screen after Loading screen the user will hear welcomed voice message to our App, then Some voice commands start to direct him to every feature in our app by swiping in four directions.

Every swipe connects by a feature from our App:

- By swiping up, the app will switch to object recognition and distance measurement. It will be activated by opening the camera and starts recognizing objects in real time, calculates the distance between the phone and the objects, and provides audio feedback describing the objects and their distances from the user, aiding in navigation and obstacle avoidance.
- By swiping down, the app will switch to text reading. It will be activated by opening the camera and starts capturing images or videos, processes the captured media to recognize and extract text, and reads aloud the extracted text, providing the user with access to the written information.
- By swiping left, the app will switch to currency recognition. It will be activated by opening the camera and starts capturing images or videos of the currency, processes the captured media to recognize the type and denomination of the currency, and provides audio feedback informing the user about the type and value of the currency.
- By swiping right, the app will switch to face recognition, person counting, and identifying a person's face. It will be activated by opening the camera, starts recognizing faces and counting the number of people in the environment, training to recognize specific faces (e.g., relatives or friends of the user), and provides audio feedback informing the user about the number of people detected and identifying recognized faces.

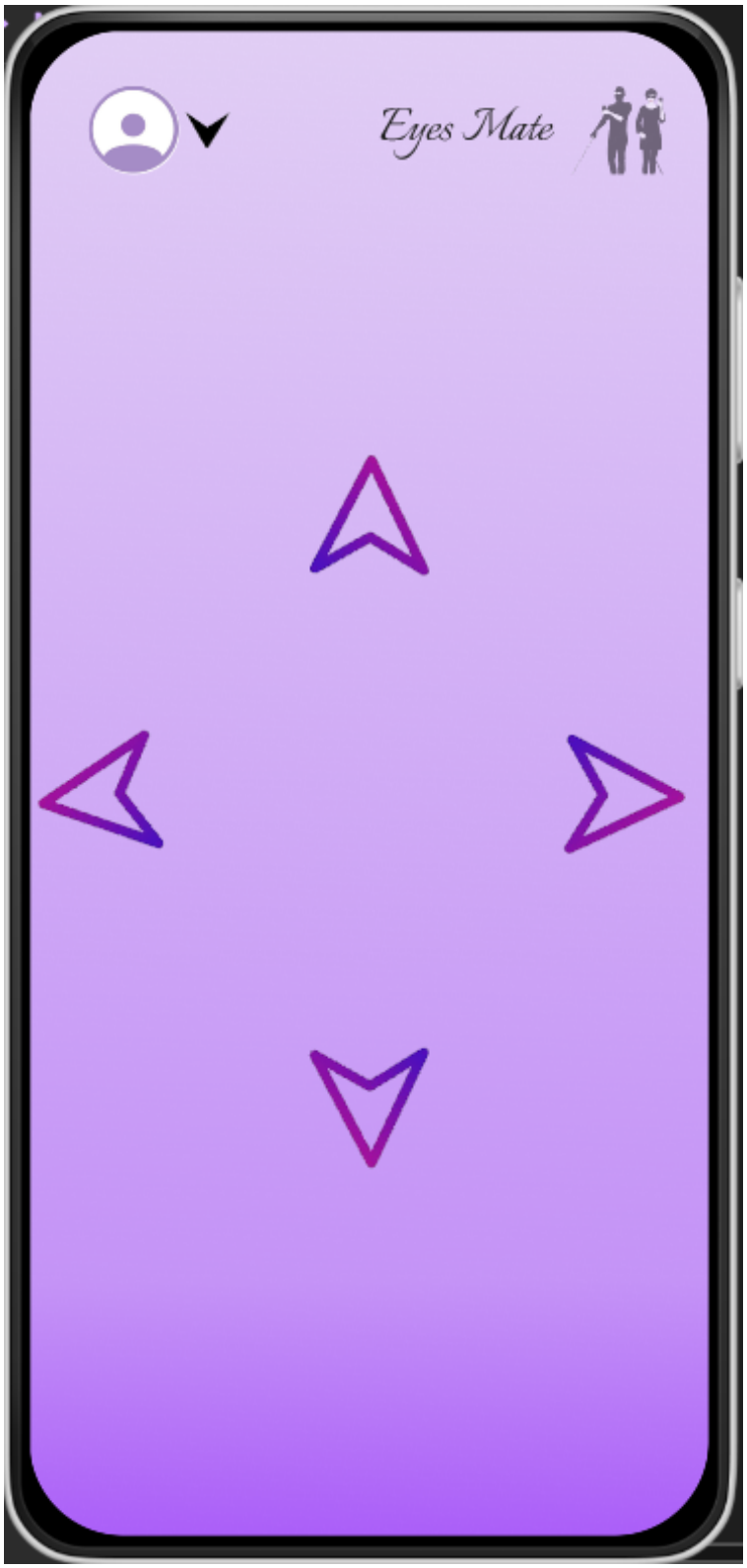


Figure 2.2: Page 2-Main Page

2.3.1.4 Main Screen with Dropdown Menu

There is another feature in main screen it's the dropdown menu for log in, any one of user's relative can register when he presses log in it will navigate to user register screen.

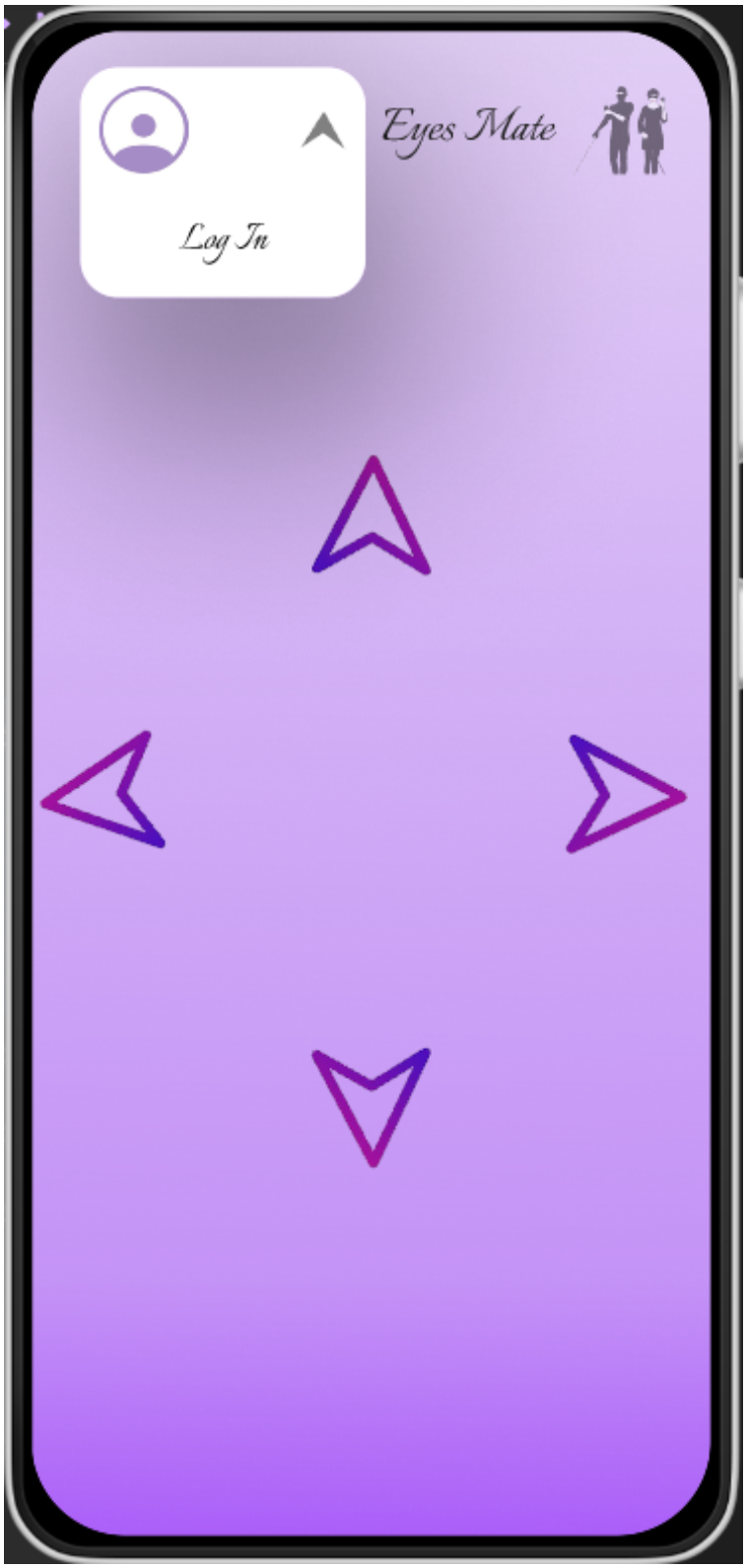


Figure 2.3: Page 2-Main Page Dropdown Menu

2.3.1.5 User Register Screen

It's the third screen in our app, in this screen the relative will register for our user by his data like (profile photo, his name, user name & password) for create for him a profile by pressing log in it will create for user a profile and navigate to main screen.

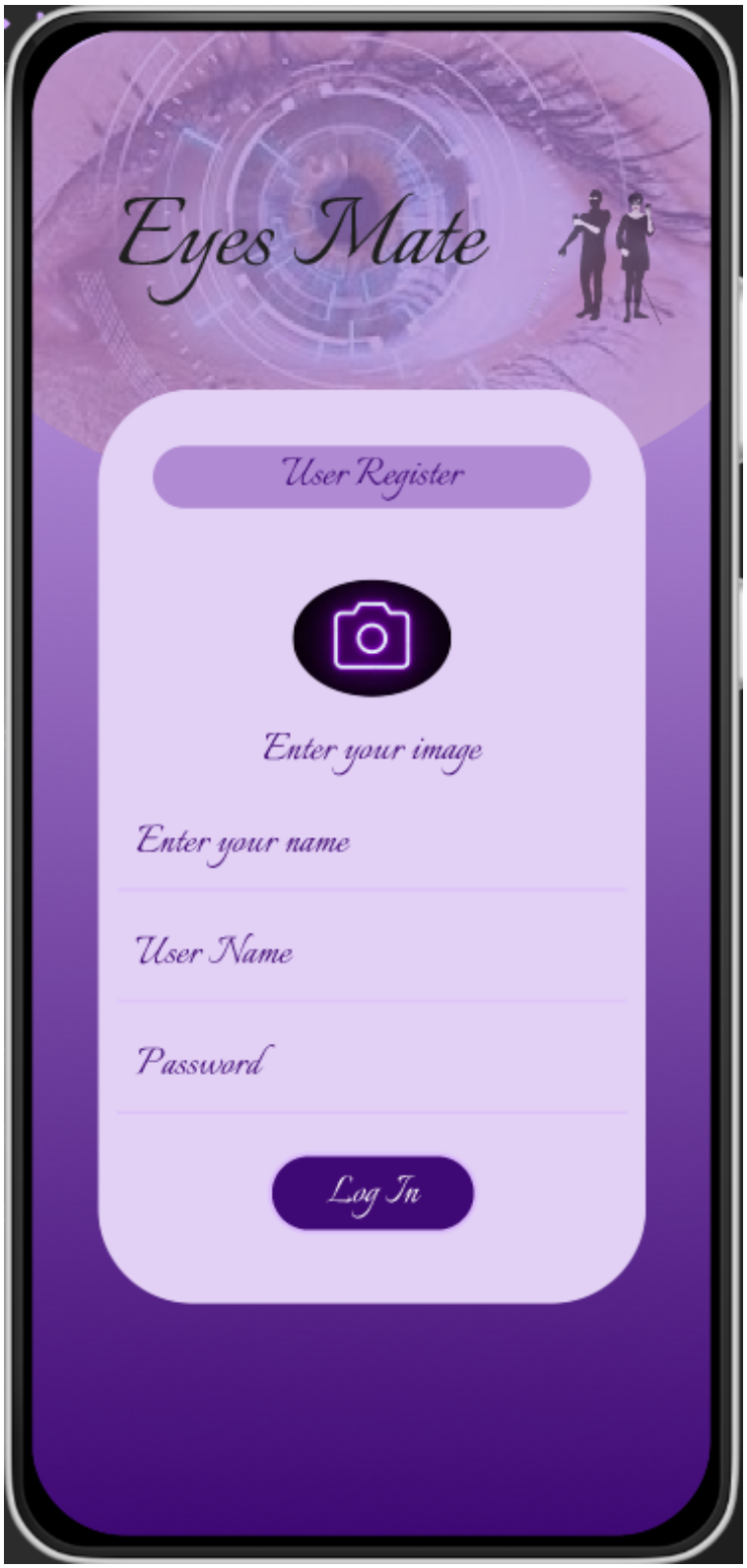


Figure 2.4: Page 3-User LogIn

2.3.1.6 Home Page Screen

It's the fourth screen in our app, it's same as the main screen but here sign in by user profile.

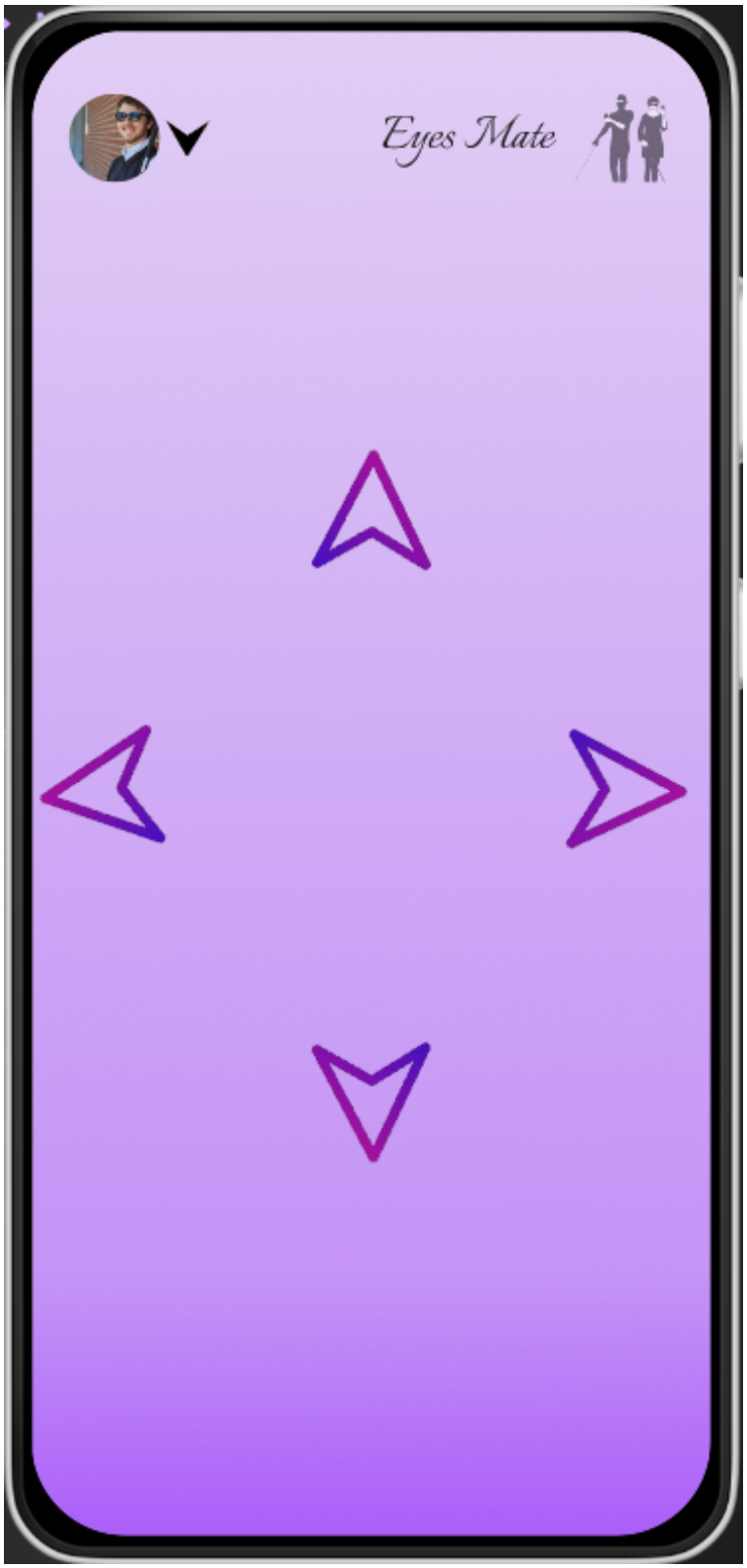


Figure 2.5: Page 4-Home Page

2.3.1.7 Home Page Screen with Dropdown Menu

The addition feature in this screen that when we press the arrow beside profile picture it will open a dropdown menu that contain user profile picture, user name, option for relative registeration and option for logging out.

Figure 2.6: Page 4-Home Page Dropdown Menu

2.3.1.8 Relative Register Screen

It's the fifth screen in our app, In this screen relative will register for himself by entering his data (image, his name & his relative relation) we use this data for training our model to Identifying a Relative's Face.when pressing log in it will navigated again to home page after storing the registered data.

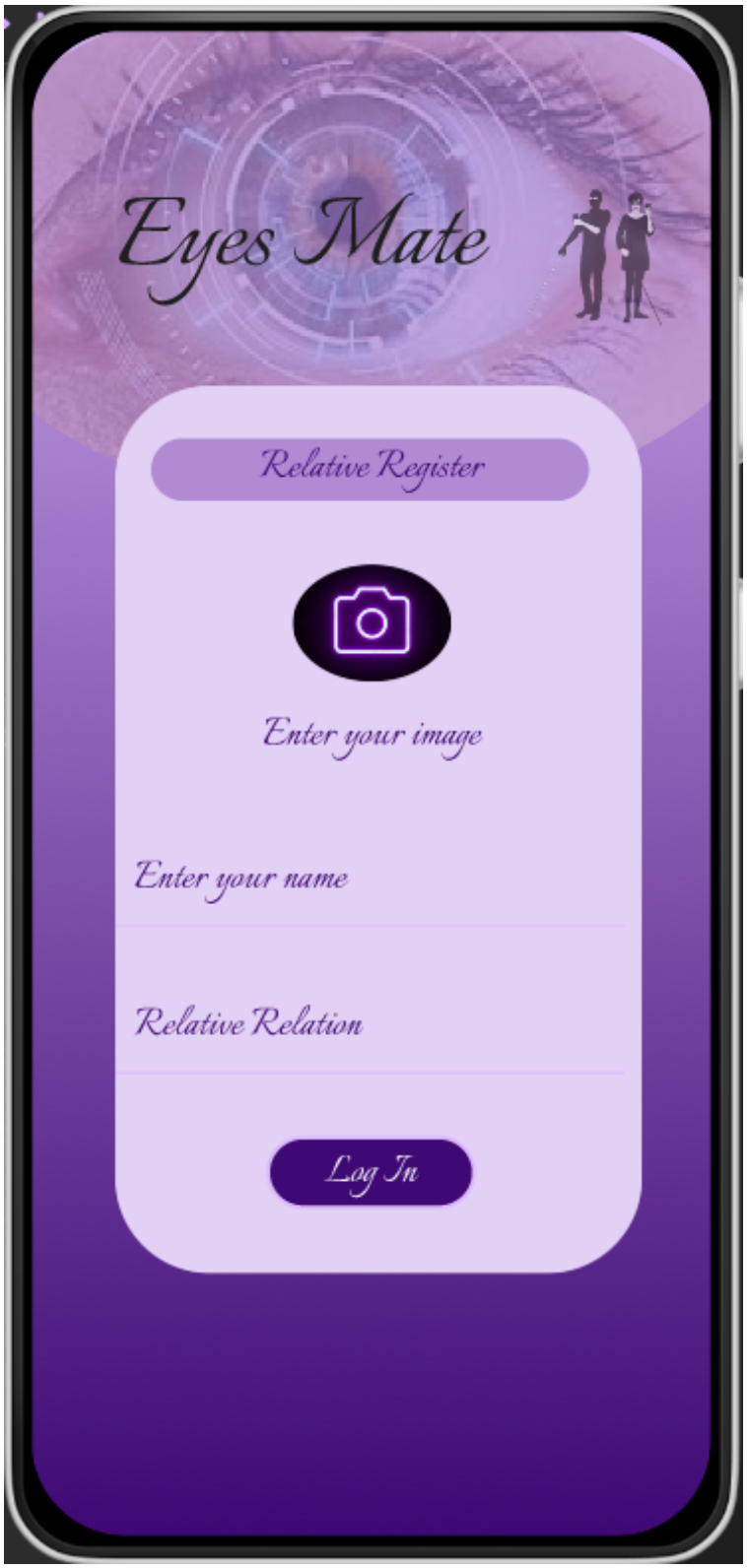


Figure 2.7: Page 5-Relative Register

2.3.1.9 Logging Out in Home Page Screen

After backward to home page there is another option for log out when pressing on it, it logging out and navigate to the loading screen.

Figure 2.8: Page 4-Backward to Home Page

Figure 2.9: Page 7-Backward to Loading Page

2.3.2 Hardware Interfaces

The hardware interfaces for the mobile application are crucial for ensuring that the software can effectively interact with the device's physical components. Below are the detailed logical and physical characteristics of each

interface:

2.3.2.1 Supported Device Types

- **Smartphones:** The application is designed to operate on modern smartphones equipped with necessary hardware components. The minimum hardware requirements are:
  - A rear-facing camera with a resolution of at least 8 megapixels.
  - A multi-core processor capable of handling machine learning algorithms.
  - At least 2GB of RAM.
  - Minimum of 100MB of storage space, with additional space for user data and models.

2.3.2.2 Camera

- **Logical Characteristics:** The camera interface captures images and videos for processing. The software accesses the camera through the device's Camera API.
- **Physical Characteristics:** Utilizes the smartphone's built-in rear-facing camera.
- **Data Interaction:** Visual data (images or videos) is captured and sent to the machine learning subsystem for processing.
- **Control Interaction:** The software controls the camera's operation (e.g., capturing an image or recording a video).

2.3.2.3 Microphone

- **Logical Characteristics:** The microphone interface allows for voice command input and audio recording.
- **Physical Characteristics:** Utilizes the smartphone's built-in microphone.
- **Data Interaction:** Audio data is captured and sent to relevant subsystems for processing.
- **Control Interaction:** The software may control microphone activation and deactivation.

2.3.2.4 Speaker

- **Logical Characteristics:** The speaker interface provides auditory feedback to the user.
- **Physical Characteristics:** Utilizes the smartphone's built-in speaker.
- **Data Interaction:** Converts text and other outputs into speech using the audio subsystem.
- **Control Interaction:** The software manages the playback of audio feedback.

2.3.2.5 Communication Protocols

- **Camera:** Accessed via the Camera API provided by the operating system.
- **Microphone and Speaker:** Accessed via the audio services API for recording and playback.

2.3.3 Software Interfaces

The mobile application interacts with various software components to deliver its functionality. Below are the details of these interfaces:

2.3.3.1 Operating Systems

- **Android OS:** Version 8.0 (Oreo) and above.
- **iOS:** Version 11.0 and above.
- **Data Interaction:** OS-level APIs are used to access hardware components like the camera, microphone, and speaker.
- **Control Interaction:** The application leverages OS-specific services for hardware control and data management.

2.3.3.2 Machine Learning Framework

- **TensorFlow Lite:** Used for deploying and running machine learning models on mobile devices.

- **Data Interaction:** Machine learning models process visual and audio data to perform tasks like object recognition, text reading, currency recognition, and face recognition.
- **Control Interaction:** Models are loaded, executed, and managed by the application's machine learning subsystem.

2.3.3.3 Development Framework

- **Flutter:** Used for building the application, ensuring cross-platform compatibility.
  - **Data Interaction:** Handles user interface elements and manages data flow between the UI and backend services.
  - **Control Interaction:** Manages the application lifecycle and UI rendering.

2.3.3.4 Database

- **Firebase:** Used for authentication, database, and storage services.
  - **Data Interaction:** User data, including profiles, preferences, and machine learning models, are stored and retrieved.
  - **Control Interaction:** Manages user authentication and secure data access.

2.3.3.5 Audio Services

- **Text-to-Speech API:** Converts text data into audible speech.
  - **Data Interaction:** Text output from various subsystems is converted into speech.
  - **Control Interaction:** Manages audio playback and speech synthesis.

2.3.3.6 Data Sharing Mechanisms

- **Shared Preferences and Secure Storage:** Used for storing user settings and sensitive data securely.
  - **Data Interaction:** Ensures user preferences and sensitive information are securely managed.

- **Control Interaction:** Manages read and write operations to secure storage.

2.3.4 Communications Interfaces

The application requires several communication functions to interact with external services and ensure data synchronization. Below are the requirements:

2.3.4.1 Internet Connectivity

- **Protocols:** HTTP/HTTPS for secure data exchange.
  - **Message Formatting:** JSON is used for structuring data exchanged between the application and server.
  - **Data Transfer Rates:** Optimized for efficient data transfer, minimizing latency and bandwidth usage.
  - **Synchronization:** Real-time synchronization mechanisms ensure data consistency between the application and backend services.

2.3.4.2 Security

- **Encryption:** All data transfers are encrypted using TLS (Transport Layer Security) to protect against unauthorized access.
- **Authentication:** Secure authentication protocols are implemented to verify user identities and authorize access to application features.

2.3.4.3 External Services

- **Firebase:** For authentication, database, and storage services.
  - **Message Formatting:** Follows Firebase's API specifications.
  - **Data Transfer Rates:** Managed by Firebase to ensure efficient and secure data handling.
  - **Synchronization:** Ensures real-time data synchronization and secure storage.

2.3.4.4 Bluetooth/Wi-Fi

- **Potential Use:** For connectivity with external devices or enhanced location services.
- **Protocols:** Standard Bluetooth and Wi-Fi communication protocols.
- **Data Transfer Rates:** Optimized for low latency and efficient data exchange.
- **Security:** Ensures encrypted communication for data security.

2.4 System Features

2.4.1 REQ-1: Object Recognition & Distance Measurement

2.4.1.1 Description and Priority

The application identifies and describes objects in real-time from images or videos captured by the phone's camera. It calculates and provides audio feedback on the distance to objects, aiding navigation and obstacle avoidance.

- **Priority:** High
- **Benefit:** 9
- **Risk:** 9

2.4.1.2 Stimulus/Response Sequences

- The user should first open the application and Swipe Up, then the feature will be activated.
- The application opens the camera and starts recognizing objects in real-time.
- The application calculates the distance between the phone and the objects.
- The application provides audio feedback describing the objects and their distances from the user, aiding in navigation and obstacle avoidance.

2.4.1.3 Functional Requirements

- The system must use the phone's camera to capture real-time images or videos.
- The system must use object recognition algorithms to identify and describe objects in the captured images or videos.
- The system must calculate the distance between the phone and the recognized objects.
- The system must provide clear and timely audio feedback describing the objects and their distances.
- The system must ensure that the audio feedback is synchronized with the real-time object recognition and distance calculation.
- The system must handle various lighting conditions and object types to maintain accurate recognition and distance measurement.

2.4.2 REQ-2: Text Reading

2.4.2.1 Description and Priority

The application reads aloud text from captured images or videos, allowing users to access written information such as signs, labels, and documents.

- **Priority:** High
- **Benefit:** 9
- **Risk:** 4

2.4.2.2 Stimulus/Response Sequences

- The user should first open the application and Swipe Down, then the feature will be activated.
- The application opens the camera and starts capturing images or videos.
- The application processes the captured media to recognize and extract text.
- The application reads aloud the extracted text, providing the user with access to the written information.

2.4.2.3 Functional Requirements

- The system must use the phone's camera to capture images or videos containing text.
- The system must use optical character recognition (OCR) algorithms to extract text from the captured media.
- The system must provide clear and timely audio feedback, reading aloud the extracted text.
- The system must ensure that the audio feedback is synchronized with the text extraction process.
- The system must handle various fonts, sizes, and languages to maintain accurate text extraction and reading.

2.4.3 REQ-3: Currency Recognition

2.4.3.1 Description and Priority

The application recognizes various currencies, enabling users to distinguish and use money confidently.

- **Priority:** High
- **Benefit:** 9
- **Risk:** 9

2.4.3.2 Stimulus/Response Sequences

- The user should first open the application and Swipe Left, then the feature will be activated.
- The application opens the camera and starts capturing images or videos of the currency.
- The application processes the captured media to recognize the type and denomination of the currency.
- The application provides audio feedback, informing the user about the type and value of the currency.

2.4.3.3 Functional Requirements

- The system must use the phone's camera to capture images or videos of the currency.
- The system must use image recognition algorithms to identify the type and denomination of the captured currency.
- The system must provide clear and timely audio feedback, informing the user about the recognized currency.
- The system must ensure that the audio feedback is synchronized with the currency recognition process.
- The system must handle various currencies, including different denominations, colors, and designs, to maintain accurate recognition.

2.4.4 REQ-4: Face Recognition, Person Counting, and Identifying a Person's Face

2.4.4.1 Description and Priority

The application identifies and counts people in the environment. It can be trained to recognize the faces of the user's relatives or friends, providing alerts when they are nearby.

- **Priority:** High
- **Benefit:** 9
- **Risk:** 7

2.4.4.2 Stimulus/Response Sequences

- The user should first open the application and Swipe Right, then the feature will be activated.
- The application opens the camera and starts recognizing faces and counting the number of people in the environment.
- The application can be trained to recognize specific faces (e.g., relatives or friends of the user).
- The application provides audio feedback, informing the user about the number of people detected and identifying recognized faces.

- The application provides alerts when recognized faces (relatives or friends) are nearby.

2.4.4.3 Functional Requirements

- The system must use the phone's camera to capture images or videos of people in the environment.
- The system must use face recognition algorithms to identify and count the number of people in the captured media.
- The system must allow training to recognize specific faces, such as relatives or friends of the user.
- The system must provide clear and timely audio feedback, informing the user about the number of people detected and identifying recognized faces.
- The system must ensure that the audio feedback is synchronized with the face recognition and person counting process.
- The system must handle various lighting conditions and face angles to maintain accurate recognition.
- The system must provide alerts when recognized faces are nearby, enhancing the user's situational awareness.

2.4.5 REQ-5: Gesture Control System

2.4.5.1 Description and Priority

Enable navigation through different features using swipe gestures (up, down, left, right) to activate different features in the application.

- **Priority:** High
- **Benefit:** 9
- **Risk:** 9

2.4.5.2 Stimulus/Response Sequences

The user should first open the application then swipe:

- **Up:** Activates object recognition and distance measurement.

- **Down:** Activates text reading.
- **Left:** Activates currency recognition.
- **Right:** Activates face recognition and person counting.

2.4.5.3 Functional Requirements

- The system must detect and recognize swipe gestures in four directions: up, down, left, and right.
- The system must map each swipe gesture to a specific navigation action or feature activation.
- The system must provide visual and/or audio feedback to confirm the gesture and the action taken.
- The system must ensure that swipe gestures are easy to perform and responsive to user input.
- The system must support gesture control in all relevant sections of the application, ensuring a consistent navigation experience.

2.4.6 REQ-6: Audio Feedback System

2.4.6.1 Description and Priority

The application provides clear and timely audio feedback for all features.

- **Priority:** High
- **Benefit:** 9
- **Risk:** 9

2.4.6.2 Stimulus/Response Sequences

- The user interacts with any feature of the application.
- The application processes the user's action and generates corresponding audio feedback.
- The user receives the audio feedback promptly, confirming the action taken or providing necessary information.

2.4.6.3 Functional Requirements

- The system must provide audio feedback for all interactive features, including object recognition, text reading, currency recognition, and face recognition.
- The audio feedback must be clear and understandable to ensure effective communication.
- The audio feedback must be provided in a timely manner to enhance user experience and responsiveness.
- The system must ensure that audio feedback is consistent and accurately reflects the application's state and user actions.
- The system must provide audio feedback for error messages and instructions to guide the user in case of issues.

2.4.7 REQ-7: User Authentication

2.4.7.1 Description and Priority

The application should allow users to log in and create accounts. It must securely store and manage user data, including personalized features.

- **Priority:** Medium
- **Benefit:** 7
- **Risk:** 4

2.4.7.2 Stimulus/Response Sequences

- The user opens the application and selects the option to create a new account or log in.
- For account creation, the user enters necessary information such as username, password, and other personal details.
- For login, the user enters their username and password.
- The application securely processes the input and grants access to personalized features upon successful authentication.
- The application securely stores user data and manages personalized features based on the authenticated user's profile.

2.4.7.3 Functional Requirements

- The system must provide a user interface for account creation and login.
- The system must validate user input to ensure security and data integrity.
- The system must securely store user credentials using encryption techniques.

2.5 Other Nonfunctional Requirements

2.5.1 Performance Requirements

The performance requirements outline the necessary specifications to ensure that the application operates efficiently and effectively. These requirements focus on the speed and accuracy of the application's functions, especially on low-quality devices, as well as the importance of minimizing delays in data retrieval and processing to provide timely information to users.

Requirement ID	Requirement Description
1. PR	• The application must be able to perform its functions quickly and with high accuracy, even on low-quality devices.
2. PR	• Efficient data retrieval and processing to minimize delays in providing information.

Table 2.1: Performance Requirements

2.5.2 Safety Requirements

The safety requirements detail the essential features and functionalities designed to ensure the security and well-being of the users. This includes implementing voice commands, real-time GPS and location-based services, offline mode capabilities, emergency features, and regular updates and bug fixes. These requirements are crucial for providing a reliable and safe user experience.

Requirement ID	Requirement Description
1. SR (Voice Commands)	• Include voice command functionality to allow users to navigate and interact with the app using their voice. • Implement natural language processing for voice commands to enhance user experience.
2. SR (Real-time GPS and Location-Based Services)	• If the app involves navigation or location-based services, ensure accurate and real-time GPS information to help users navigate safely. • Include features that announce nearby points of interest and hazards.
3. SR (Offline Mode)	• Include an offline mode or download option for essential data, ensuring users can access critical information even when not connected to the internet.
4. SR (Emergency Features)	• Integrate emergency features, such as a panic button or quick access to emergency services, for the safety of users in urgent situations.
5. SR (Regular Updates and Bug Fixes)	• Commit to regular updates to address bugs, enhance features, and improve accessibility based on user feedback.

Table 2.2: Safety Requirements

2.5.3 Security Requirements

The security requirements emphasize the importance of robust security measures to protect user data and maintain privacy. These requirements include implementing secure authentication methods and encryption to prevent unau-

thorized access and ensure that user data is handled with the highest level of security.

Requirement ID	Requirement Description
1. SR	• Implement robust security measures to protect user data and maintain privacy.
2. SR	• Use secure authentication methods and encryption to prevent unauthorized access.

Table 2.3: Security Requirements

2.5.4 Software Quality Attributes

The software quality attributes define the standards for accessibility, usability, and reliability that the application must meet. These attributes ensure that the application is accessible to users with visual impairments, easy to navigate, and provides a consistent and stable user experience. Additionally, they include provisions for error handling and performance consistency as the user base grows.

Requirement ID	Requirement Description
1. SQA (Accessibility)	• Ensure that the app is designed to be accessible to users with visual impairments. • Support screen readers and other assistive technologies to make content understandable.
2. SQA (Usability)	• Design an intuitive and user-friendly interface that is easy to navigate. • Provide clear and concise audio or tactile feedback for user actions. • Provide voice commentary in more than one language to make it easier for users.
3. SQA (Reliability)	• Ensure the app's stability and dependability to provide a consistent user experience. • Implement error handling mechanisms to gracefully manage unexpected situations. • Ensure the app's performance remains consistent as the number of users and transactions increases.

Table 2.4: Software Quality Attributes

2.5.5 Business Rules

The business rules specify the essential guidelines and protocols that govern the application's operations and interactions with users. These rules include user authentication, notification preferences, language and localization options, user consent for data collection, and regular app updates. Adhering to these business rules is vital for maintaining user trust, regulatory compliance, and the overall functionality of the application.

Requirement ID	Requirement Description
1. BR (User Authentication)	• Only registered users can access personalized features. • User authentication must comply with security standards to ensure data privacy.
2. BR (Notification Preferences)	• Users can customize their notification preferences based on their individual needs. • Notifications should adhere to accessibility standards, such as providing alternatives for visual cues.
3. BR (Language and Localization)	• Users can select their preferred language for app interactions. • The app should adapt to regional preferences based on user location.
4. BR (User Consent)	• Users must provide explicit consent for the collection and use of personal data. • Consent preferences should be easily accessible and modifiable by the user.
5. BR (App Updates)	• Regular app updates may be pushed to users to introduce new features or address security issues. • Users are encouraged to keep the app updated for the best experience and security.

Table 2.5: Business Rules

Chapter 3

Project Design

3.1 Introduction

3.1.1 Purpose

The purpose of this Software Design Document (SDD) is to outline the design considerations and technical specifications for the development of a mobile application aimed at assisting visually impaired individuals in their daily lives through the use of machine learning technology. The intended audience for this document includes software developers, designers, project managers, stakeholders, and anyone involved in the development process.

3.1.2 Scope

The software aims to develop a mobile application utilizing machine learning algorithms to aid visually impaired individuals. The key features and functionalities include:

- Utilizing the mobile phone's camera to capture images or videos of the surrounding environment.
- Implementing machine learning algorithms for object recognition, text reading, face recognition, currency detection, distance measurement, and people counting.
- Converting recognized objects and text into audio descriptions for the user.
- Providing navigation assistance indoors and outdoors by measuring distances to objects.
- Recognizing familiar faces of the user's relatives or friends.

- Alerting the user when familiar faces are detected nearby.

The objectives of the project are to enhance the independence and quality of life for visually impaired individuals by providing them with tools to identify and navigate their surroundings effectively. The benefits of the software include improved accessibility, increased autonomy, and enhanced social interaction for the visually impaired community.

3.1.3 Overview and Organization

This document details the requirements and specifications for a mobile application designed to assist blind individuals in their daily lives using machine learning technology. The application leverages the phone's camera to capture real-time footage and utilizes machine learning algorithms to identify and describe objects, converting these descriptions into audio form. The app also includes features such as object detection, distance measurement, text reading, currency recognition, and face recognition.

3.1.4 Definitions and Acronyms

- **ML (Machine Learning):** A type of artificial intelligence that enables the app to learn from and make predictions based on data.
- **Real-time Footage:** Live video feed captured by the phone's camera.
- **Object Detection:** A technology that allows the app to identify objects within the real-time footage.
- **Distance Measurement:** A feature that calculates the distance between the user and objects in their surroundings.
- **OCR (Optical Character Recognition):** A technology used to convert different types of documents, such as scanned paper documents, PDFs, or images captured by a digital camera, into editable and searchable data.
- **Currency Recognition:** A feature that allows the app to identify and differentiate various currencies.
- **Face Recognition:** A technology that identifies or verifies a person from a digital image by comparing and analyzing patterns.

- **Audio Description:** Converting visual information into spoken words to describe the surroundings.
- **User:** Refers to the blind individuals using the application.
- **UI (User Interface):** The means by which the user and the app interact (the display screen).

3.2 Decomposition Description

The mobile application designed to assist blind individuals can be decomposed into several key components and sub-components, each playing a critical role in the functionality of the system. These components include:

3.2.1 User Interface (UI)

3.2.1.1 Main Screen

- Provides access to the main features of the application.
- Displays current status and notifications.

3.2.1.2 Help and Support

- Offers access to user guides, FAQs, and customer support.

3.2.2 Camera Module

3.2.2.1 Real-time Capture

- Captures live video feed of the user's surroundings.

3.2.2.2 Image Processing

- Processes captured images for further analysis by machine learning algorithms.

3.2.3 Machine Learning Module

3.2.3.1 Object Detection

- Identifies objects within the real-time footage.

3.2.3.2 Distance Measurement

- Calculates the distance between the user and detected objects.

3.2.3.3 Text Recognition (OCR)

- Reads and converts text from images into audio.

3.2.3.4 Currency Recognition

- Identifies different currencies and their denominations.

3.2.3.5 Face Recognition

- Identifies faces of known individuals such as relatives or friends.

3.2.4 Audio Module

3.2.4.1 Text-to-Speech (TTS) Conversion

- Converts text descriptions of objects and scenes into audio.

3.2.4.2 Audio Notifications

- Provides real-time audio feedback and notifications to the user.

3.3 Dependency Description

3.3.1 Inter-module Dependencies

3.3.1.1 Object Detection and Distance Measurement Module

- **Identification:** Object Detection Module
- **Type:** Dependent on the camera and TensorFlow Lite library
- **Purpose:** Detect objects in real-time and measure their distance from the user
- **Dependencies:**
  - Camera: Required for capturing real-time video feed
  - TensorFlow Lite: Required for running the YOLOv5 model on the device

- Audio Output: Required for announcing detected objects and distances audibly

- **Resources:**
  - Hardware: Device camera and microphone
  - Software: TensorFlow Lite model file (object_detection.tflite), Flutter camera package, and text-to-speech package

3.3.1.2 Currency Recognition Module

- **Identification:** Currency Recognition Module
- **Type:** Dependent on the camera and TensorFlow Lite library
- **Purpose:** Recognize different currencies in real-time
- **Dependencies:**
  - Camera: Required for capturing real-time video feed
  - TensorFlow Lite: Required for running the currency recognition model on the device
  - Audio Output: Required for announcing recognized currency denominations audibly
- **Resources:**
  - Hardware: Device camera and microphone
  - Software: TensorFlow Lite model file (currency_recognition.tflite), Flutter camera package, and text-to-speech package

3.3.1.3 Text Recognition Module

- **Identification:** Text Recognition Module
- **Type:** Dependent on the camera and TensorFlow Lite or Tesseract library
- **Purpose:** Recognize text from images captured by the camera
- **Dependencies:**
  - Camera: Required for capturing real-time video feed

  - TensorFlow Lite or Tesseract: Required for running the text recognition model on the device
  - Audio Output: Required for announcing recognized text audibly
- **Resources:**
  - Hardware: Device camera and microphone
  - Software: OCR model file (text_recognition.tflite), Flutter camera package, and text-to-speech package

3.3.1.4 Face Recognition Module (Flask)

- **Identification:** Face Recognition Module
- **Type:** Dependent on the camera, Flask server, and local database
- **Purpose:** Recognize known individuals and count the number of people in the camera's view
- **Dependencies:**
  - Camera: Required for capturing real-time video feed
  - Flask Server: Required for processing face recognition requests
  - Local Database: Required for storing known individuals' data
  - Audio Output: Required for announcing recognized individuals and the total count audibly
- **Resources:**
  - Hardware: Device camera and microphone
  - Software: Face recognition model on Flask server, local database for storing known faces, Flutter camera package, text-to-speech package, and HTTP package for server communication

3.3.2 Inter-process Dependencies

3.3.2.1 Object Detection Process

- **Order of Execution:**
  1. Capture video feed from the camera
  2. Process each frame using the YOLOv5 model

3. Calculate the distance of detected objects
4. Output results audibly

3.3.2.2 Currency Recognition Process

- **Order of Execution:**
  1. Capture video feed from the camera
  2. Process each frame using the currency recognition model
  3. Identify and announce the currency denomination audibly

3.3.2.3 Text Recognition Process

- **Order of Execution:**
  1. Capture video feed from the camera
  2. Process each frame using the text recognition model
  3. Extract and announce recognized text audibly

3.3.2.4 Face Recognition Process (Flask)

- **Order of Execution:**
  1. Capture video feed from the camera
  2. Send frames to the Flask server for processing
  3. Server processes the frames to detect and recognize faces
  4. Compare detected faces against stored data
  5. Output recognized individuals' names and total count audibly

3.3.3 Data Dependencies

3.3.3.1 Object Detection Data

- **Shared Information:** Detected objects and their distances
- **Dependencies:** YOLOv5 model data, camera feed, and audio output for results

3.3.3.2 Currency Recognition Data

- **Shared Information:** Recognized currency denominations
- **Dependencies:** Currency recognition model data, camera feed, and audio output for results

3.3.3.3 Text Recognition Data

- **Shared Information:** Recognized text from images
- **Dependencies:** Text recognition model data, camera feed, and audio output for results

3.3.3.4 Face Recognition Data (Flask)

- **Shared Information:** Names of recognized individuals and the total number of people
- **Dependencies:** Face recognition model data on the Flask server, camera feed, local database for known faces, and audio output for results

3.4 Interface Description

3.4.1 Module Interface

3.4.1.1 Module 1 Description

1. **Module Definition:**
   - **Module Name:** Object Detection and Distance Measurement Module
   - **Description:** This module detects objects and measures distances for visually impaired users. The transition to this module from the main page is done via a swipe gesture. The module activates the model and camera for real-time detection and announces the object's name and distance audibly.
2. **Module Functions:**
   - Object Detection: Detects various objects in the camera frame.
   - Distance Measurement: Measures the distance between the user and detected objects.

- Audible Announcement: Announces the name of the detected object and its distance audibly.

3. **External Interfaces:**
   - Page Navigation:
     - **Description:** Transition from the main page to the detection module via a swipe gesture.
     - **Inputs:** Swipe gesture.
     - **Outputs:** Navigation to the detection module screen.
   - Model and Camera Activation:
     - **Description:** Upon navigation to the detection module, the model and camera are activated for real-time detection.
     - **Inputs:** None.
     - **Outputs:** Live video feed from the camera and real-time analysis by the model.
   - Audible Announcement:
     - **Description:** Announces the detected object's name and distance audibly.
     - **Inputs:** Object name and distance.
     - **Outputs:** Audible announcement.
4. **Internal Interfaces:**
   - Detection Model:
     - **Description:** Uses the detection model to identify objects in the camera frame.
     - **Inputs:** Live video feed from the camera.
     - **Outputs:** Names of detected objects and distances.
   - Audio Module:
     - **Description:** Used to announce the names of detected objects and distances audibly.
     - **Inputs:** Text containing object name and distance.
     - **Outputs:** Audible announcement.
5. **Module Attributes:**

- Module ID: 1
- Version: 1.0
- Creation Date: 2024-06-26
- Developer: App Development Team

3.4.1.2 Module 2 Description

1. **Module Definition:**
   - **Module Name:** Currency Recognition Module
   - **Description:** This module is responsible for recognizing different types of currency for visually impaired users. Similar to the object detection module, users transition to this module from the main page via a swipe gesture. The module then activates the currency recognition model and the camera for real-time recognition, announcing the currency type and value audibly.
2. **Module Functions:**
   - Currency Recognition: Identifies various currencies within the camera's view.
   - Audible Announcement: Announces the recognized currency type and value using audio.
3. **External Interfaces:**
   - Page Navigation:
     - **Description:** The user navigates from the main page to the currency recognition module by performing a swipe gesture.
     - **Inputs:** Swipe gesture.
     - **Outputs:** Navigation to the currency recognition module interface.
   - Model and Camera Activation:
     - **Description:** Activates the recognition model and camera for real-time currency recognition when the user navigates to this module.
     - **Inputs:** None.

- **Outputs:** Live camera feed and real-time currency recognition data.

- Audible Announcement:
  - **Description:** Provides audio feedback, announcing the recognized currency type and value.
  - **Inputs:** Currency type and value.
  - **Outputs:** Audio announcement.

4. **Internal Interfaces:**

- Recognition Model:
  - **Description:** Utilizes a recognition model to identify currencies in the camera feed.
  - **Inputs:** Live video feed from the camera.
  - **Outputs:** Recognized currency types and values.
- Audio Module:
  - **Description:** Converts text inputs into audible announcements.
  - **Inputs:** Text data containing the currency type and value.
  - **Outputs:** Audio feedback.

5. **Module Attributes:**

- Module ID: 2
- Version: 1.0
- Creation Date: 2024-06-26
- Developer: App Development Team

3.4.1.3 Module 3 Description

1. **Module Definition:**

- **Module Name:** Text Recognition Module
- **Description:** This module recognizes text for visually impaired users. Similar to the other modules, users transition to this module from the main page via a swipe gesture. The module activates the model and camera for real-time text recognition, announcing the recognized text audibly.

2. **Module Functions:**
    - Text Recognition: Identifies and reads text within the camera's view.
    - Audible Announcement: Announces the recognized text audibly.
3. **External Interfaces:**
    - Page Navigation:
        - **Description:** Transition from the main page to the text recognition module via a swipe gesture.
        - **Inputs:** Swipe gesture.
        - **Outputs:** Navigation to the text recognition module screen.
    - Model and Camera Activation:
        - **Description:** Upon navigation to the text recognition module, the model and camera are activated for real-time recognition.
        - **Inputs:** None.
        - **Outputs:** Live video feed from the camera and real-time analysis by the model.
    - Audible Announcement:
        - **Description:** Announces the recognized text audibly.
        - **Inputs:** Recognized text.
        - **Outputs:** Audible announcement.
4. **Internal Interfaces:**
    - Recognition Model:
        - **Description:** Uses the recognition model to identify text in the camera frame.
        - **Inputs:** Live video feed from the camera.
        - **Outputs:** Recognized text.
    - Audio Module:
        - **Description:** Used to announce the recognized text audibly.
        - **Inputs:** Text containing the recognized text.
        - **Outputs:** Audible announcement.
5. **Module Attributes:**
    - Module ID: 3

- Version: 1.0
- Creation Date: 2024-06-26
- Developer: App Development Team

3.4.1.4 Module 4 Description

1. **Module Definition:**
   - **Module Name:** Face Recognition Module
   - **Description:** This module recognizes faces and counts the number of people for visually impaired users. Users transition to this module from the main page via a swipe gesture. The module activates the model and camera for real-time face recognition, identifying known individuals and prompting the user to register unknown individuals in the database. It also announces the names of known individuals and the total number of people in view audibly.
2. **Module Functions:**
   - Face Recognition: Identifies faces within the camera's view and matches them against a database of known individuals.
   - Person Counting: Counts the total number of people in the camera's view.
   - Audible Announcement: Announces the names of known individuals and the total number of people audibly.
   - Registration Prompt: Prompts the user to register unknown individuals in the database.
3. **External Interfaces:**
   - Page Navigation:
     - **Description:** Transition from the main page to the face recognition module via a swipe gesture.
     - **Inputs:** Swipe gesture.
     - **Outputs:** Navigation to the face recognition module screen.
   - Model and Camera Activation:
     - **Description:** Upon navigation to the face recognition module, the model and camera are activated for real-time face recognition.

  - **Inputs:** None.
  - **Outputs:** Live video feed from the camera and real-time face recognition analysis by the model.
- Audible Announcement:
  - **Description:** Announces the names of known individuals and the total number of people audibly.
  - **Inputs:** Names of known individuals and the total number of people.
  - **Outputs:** Audible announcement.
- Registration Prompt:
  - **Description:** Prompts the user to register unknown individuals in the database.
  - **Inputs:** Detection of an unknown individual.
  - **Outputs:** Prompt to register the unknown individual.

4. **Internal Interfaces:**

- Recognition Model:
  - **Description:** Uses the recognition model to identify faces and match them against a database of known individuals.
  - **Inputs:** Live video feed from the camera.
  - **Outputs:** Names of known individuals and the total number of people.
- Database Module:
  - **Description:** Stores and retrieves information about known individuals.
  - **Inputs:** Data about known individuals.
  - **Outputs:** Information about known individuals.
- Audio Module:
  - **Description:** Used to announce the names of known individuals and the total number of people audibly.
  - **Inputs:** Text containing the names of known individuals and the total number of people.
  - **Outputs:** Audible announcement.

- Registration Module:
  - **Description:** Used to prompt the user to register unknown individuals in the database.
  - **Inputs:** Detection of an unknown individual.
  - **Outputs:** Prompt to register the unknown individual.

5. **Module Attributes:**
   - Module ID: 4
   - Version: 1.0
   - Creation Date: 2024-06-26
   - Developer: App Development Team

3.4.2 Process Interface

3.4.2.1 Process 1 Description: Object Detection and Distance Measurement Process

1. **Process Definition:**
   - **Process Name:** Object Detection and Distance Measurement
   - **Description:** This process involves detecting objects in real-time and measuring the distance between the user and the detected objects. The process begins when the user transitions to the object detection module via a swipe gesture.
2. **Process Steps:**
   (a) Swipe Gesture Detection:
   - **Description:** Detects the swipe gesture to transition from the main page to the object detection module.
   - **Inputs:** Swipe gesture.
   - **Outputs:** Transition to the object detection screen.

   (b) Activate Camera and Model:
   - **Description:** Activates the camera and the object detection model for real-time analysis.
   - **Inputs:** None.
   - **Outputs:** Live video feed and object detection results.

   (c) Real-time Object Detection:

- **Description:** Analyzes the live video feed to detect objects and measure their distances.
- **Inputs:** Live video feed.
- **Outputs:** Detected object names and distances.

(d) Audible Announcement:

- **Description:** Announces the names and distances of detected objects audibly.
- **Inputs:** Detected object names and distances.
- **Outputs:** Audio feedback announcing the information.

3.4.2.2 Process 2 Description: Currency Recognition Process

1. **Process Definition:**
   - **Process Name:** Currency Recognition
   - **Description:** This process involves recognizing different types of currency in real-time. The process starts when the user transitions to the currency recognition module via a swipe gesture.
2. **Process Steps:**

(a) Swipe Gesture Detection:

- **Description:** Detects the swipe gesture to transition from the main page to the currency recognition module.
- **Inputs:** Swipe gesture.
- **Outputs:** Transition to the currency recognition screen.

(b) Activate Camera and Model:

- **Description:** Activates the camera and the currency recognition model for real-time analysis.
- **Inputs:** None.
- **Outputs:** Live video feed and currency recognition results.

(c) Real-time Currency Recognition:

- **Description:** Analyzes the live video feed to recognize different currencies.
- **Inputs:** Live video feed.
- **Outputs:** Recognized currency types and values.

(d) Audible Announcement:

- **Description:** Announces the recognized currency types and values audibly.
- **Inputs:** Recognized currency types and values.
- **Outputs:** Audio feedback announcing the information.

3.4.2.3 Process 3 Description: Text Recognition Process

1. **Process Definition:**
   - **Process Name:** Text Recognition
   - **Description:** This process involves recognizing text in real-time. The process starts when the user transitions to the text recognition module via a swipe gesture.
2. **Process Steps:**

(a) Swipe Gesture Detection:

- **Description:** Detects the swipe gesture to transition from the main page to the text recognition module.
- **Inputs:** Swipe gesture.
- **Outputs:** Transition to the text recognition screen.

(b) Activate Camera and Model:

- **Description:** Activates the camera and the text recognition model for real-time analysis.
- **Inputs:** None.
- **Outputs:** Live video feed and text recognition results.

(c) Real-time Text Recognition:

- **Description:** Analyzes the live video feed to recognize text.
- **Inputs:** Live video feed.
- **Outputs:** Recognized text.

(d) Audible Announcement:

- **Description:** Announces the recognized text audibly.
- **Inputs:** Recognized text.
- **Outputs:** Audio feedback announcing the information.

3.4.2.4 Process 4 Description: Face Recognition Process

1. **Process Definition:**
   - **Process Name:** Face Recognition
   - **Description:** This process involves recognizing faces and counting the number of people in the camera's view. It checks if faces are registered in the database and prompts the user to register new faces if unrecognized. The process starts when the user transitions to the face recognition module via a swipe gesture.
2. **Process Steps:**
   (a) Swipe Gesture Detection:
      - **Description:** Detects the swipe gesture to transition from the main page to the face recognition module.
      - **Inputs:** Swipe gesture.
      - **Outputs:** Transition to the face recognition screen.

   (b) Activate Camera and Model:
      - **Description:** Activates the camera and the face recognition model for real-time analysis.
      - **Inputs:** None.
      - **Outputs:** Live video feed and face recognition results.

   (c) Real-time Face Recognition:
      - **Description:** Analyzes the live video feed to recognize faces and count the number of people.
      - **Inputs:** Live video feed.
      - **Outputs:** Recognized faces and the count of people.

   (d) Database Check:
      - **Description:** Checks if recognized faces are in the database and prompts the user to register new faces if unrecognized.
      - **Inputs:** Recognized faces.
      - **Outputs:** Registered face data or prompt to register new face.

   (e) Audible Announcement:
      - **Description:** Announces the names of recognized individuals and the total count of people audibly.

- **Inputs:** Names of recognized individuals and the count of people.
- **Outputs:** Audio feedback announcing the information.

3.5 Detailed Description

3.5.1 Module Detailed Design

This section contains the internal details of each design entity for your Flutter app modules aimed at assisting visually impaired users.

3.5.1.1 Module 1 Detail: Object Detection and Distance Measurement

Identification:

- Name: Object Detection and Distance Measurement Module
- Purpose: Detect objects in real-time using the camera and measure their distance from the user.
- Input: Camera feed
- Output: Real-time object detection data including object name and distance

Processing:

- Algorithm: YOLOv5 model for object detection
- Steps:
  1. Activate the camera and capture real-time video feed.
  2. Pass the video frames to the YOLOv5 model.
  3. Process each frame to detect objects and calculate their distances.
  4. Output the object names and distances audibly.

Data:

- Model: Pre-trained YOLOv5 model (converted to TensorFlow Lite format)
- Storage: Store the converted model file in the app's assets
- Dependencies: TensorFlow Lite library for running the model

3.5.1.2 Module 2 Detail: Currency Recognition

Identification:

- Name: Currency Recognition Module
- Purpose: Recognize different currencies using the camera feed.
- Input: Camera feed
- Output: Recognized currency denomination

Processing:

- Algorithm: Custom-trained model for currency recognition
- Steps:
  1. Activate the camera and capture real-time video feed.
  2. Pass the video frames to the currency recognition model.
  3. Process each frame to identify the currency and its denomination.
  4. Output the recognized currency denomination audibly.

Data:

- Model: Custom-trained currency recognition model (converted to TensorFlow Lite format)
- Storage: Store the converted model file in the app's assets
- Dependencies: TensorFlow Lite library for running the model

3.5.1.3 Module 3 Detail: Text Recognition

Identification:

- Name: Text Recognition Module
- Purpose: Recognize text from images captured by the camera.
- Input: Camera feed
- Output: Recognized text

Processing:

- Algorithm: OCR (Optical Character Recognition) model

- Steps:
  1. Activate the camera and capture real-time video feed.
  2. Pass the video frames to the OCR model.
  3. Process each frame to extract and recognize text.
  4. Output the recognized text audibly.

Data:

- Model: OCR model (such as Tesseract or a custom-trained model, converted to TensorFlow Lite format)
- Storage: Store the converted model file in the app's assets
- Dependencies: TensorFlow Lite library or Tesseract library for text recognition

3.5.1.4 Module 4 Detail: Face Recognition

Identification:

- Name: Face Recognition Module
- Purpose: Recognize known individuals and count the number of people in the camera's view.
- Input: Camera feed
- Output: Names of recognized individuals and the total number of people

Processing:

- Algorithm: Face recognition model (e.g., based on FaceNet or similar)
- Steps:
  1. Activate the camera and capture real-time video feed.
  2. Pass the video frames to the face recognition model.
  3. Process each frame to detect and recognize faces.
  4. Compare detected faces against a stored database of known individuals.
  5. If an unknown face is detected, prompt the user to register the new individual.

6. Output the names of recognized individuals and the total count of people audibly.

Data:

- Model: Pre-trained face recognition model (converted to TensorFlow Lite format)
- Storage: Store the converted model file and known individuals' data in the app's assets
- Dependencies: TensorFlow Lite library for running the model, local database for storing known individuals

Additional Requirements

For all modules, ensure the following:

- User Interface: Design a user-friendly interface with accessible navigation for visually impaired users, utilizing gestures like swiping for module transitions.
- Audio Output: Integrate a text-to-speech library to provide auditory feedback for detected objects, recognized currencies, text, and faces.
- Performance: Optimize the models and app performance to ensure real-time processing with minimal latency.
- Testing: Thoroughly test each module under various conditions to ensure accuracy and reliability.

3.5.2 Data Detailed Design

This section contains detailed information about the data entities used in the Flutter app modules designed to assist visually impaired users, with a specific focus on the face recognition module which will use Flask.

3.5.2.1 Data Entity 1 Detail: Model Data

Identification:

- Name: Model Data

- Purpose: Store the machine learning models used for object detection, currency recognition, text recognition, and face recognition.
- Type: TensorFlow Lite models (for mobile) and server-based models (for Flask)
- Location: App assets directory (mobile models) and Flask server (face recognition model)

Attributes:

- Mobile Model Files:
  - object_detection.tflite
  - currency_recognition.tflite
  - text_recognition.tflite
- Server Model Files:
  - face_recognition_model (stored on the Flask server)
- Description: Pre-trained models for each specific task.
- Storage Path:
  - Mobile models: Stored in the assets/models/ directory of the Flutter project.
  - Server model: Stored on the server in a specified directory.
- Version: Include versioning information for each model to manage updates and compatibility.
- Size: Ensure the size is optimized for mobile performance without compromising accuracy.

Relationships:

- Used By: All four modules (Object Detection and Distance Measurement, Currency Recognition, Text Recognition, Face Recognition).
- Dependencies: Requires TensorFlow Lite runtime for execution on mobile and Flask server runtime for face recognition.

Constraints:

- Optimization: Models should be quantized and optimized for mobile devices to ensure efficient performance.
- Updates: Mechanism to update models as needed without requiring a complete app update.

3.5.2.2 Data Entity 2 Detail: User Data

Identification:

- Name: User Data
- Purpose: Store user-specific data such as known faces for face recognition and configuration settings.
- Type: Local database (e.g., SQLite) or Shared-Preferences (mobile) and server database (e.g., SQLite, PostgreSQL, MongoDB)

Attributes:

- Face Recognition Data (Server):
  - Schema:
    - id: Unique identifier for each person.
    - name: Name of the individual.
    - face_embedding: Encoded face data for recognition.
  - Description: Contains known individuals' data for the face recognition module.
  - Storage Path: Stored in a server database.
- Configuration Settings (Mobile):
  - Schema:
    - setting_name: Name of the configuration setting.
    - setting_value: Value of the configuration setting.
  - Description: Stores user preferences and app configuration settings.
  - Storage Path: Stored in Shared-Preferences or local SQLite database.

Relationships:

- Used By: Face Recognition module for identifying known individuals.

- Dependencies: Requires local database management system (SQLite) and Shared-Preferences on mobile; server database for Flask.

Constraints:

- Security: Data should be securely stored and managed, especially sensitive information like face embeddings.
- Backup: Provide mechanisms to back up and restore user data to ensure data integrity and availability.
- Updates: Allow for updating user data, such as adding new individuals to the face recognition database.

Chapter 4

Methodology

In this chapter, we delve into the methodology employed in our study, focusing on the main algorithms and their procedural steps. We will systematically describe the formal algorithmic steps of the selected methods used to solve the considered problem. This will include detailed pseudocodes or flowcharts for each model. Our discussion will cover how these algorithms address the problem and an analysis of their time complexity.

The four models under consideration are:

4.1 Model 1: Object Recognition & Distance Measurement

4.1.1 Main Algorithms [3]

The primary algorithm used in this model is YOLOv5 (You Only Look Once, version 5) for real-time object detection. Additionally, OpenCV is used for capturing video from the camera and processing the video frames.

4.1.2 Object Detection using YOLOv5

YOLOv5 is a state-of-the-art, real-time object detection system that applies a single neural network to the full image. This network divides the image into regions and predicts bounding boxes and probabilities for each region.

4.1.3 Steps of the YOLOv5 Object Detection Algorithm

1. **Initialize YOLOv5 and Select Device**: Load the YOLOv5 model and select the device (CPU or GPU) for computation.
2. **Set Up Directories and Load Model**: Prepare directories for saving results and load the YOLOv5 model weights.

3. **Capture Video or Load Images**: Start capturing video from the camera or load images from a specified source.
4. **Process Each Frame or Image**:
   - Preprocess the image or frame (resize, normalize).
   - Run inference using YOLOv5.
   - Apply Non-Maximum Suppression (NMS) to filter overlapping bounding boxes.
   - Scale bounding boxes back to original image size.
   - Draw bounding boxes on the image or frame.
   - Save or display the results.
5. **Release Resources**: Release the video capture and close any OpenCV windows.

4.1.4 Formal Algorithmic Steps of YOLOv5 Object Detection

4.1.4.1 Pseudocode for YOLOv5 Object Detection

This pseudocode outlines the steps for implementing real-time object detection using the YOLOv5 model:

Algorithm 1 Real-time Object Detection with YOLOv5

Input: Video stream from the camera or image files
Output: Real-time object detection and saved or displayed results
function DETECT(opt)
    Initialize YOLOv5 model and select device
    Prepare directories for saving results
    Load YOLOv5 model weights

    if source is webcam **then**
        Start capturing video from the camera
    else
        Load images from the specified source
    end if
    while True **do**
        Capture frame-by-frame or load image
        if not ret **then**

        end if
        Preprocess image or frame (resize, normalize)
        Run inference using YOLOv5
        Apply Non-Maximum Suppression (NMS)
        Scale bounding boxes back to original image size
        Draw bounding boxes on the image or frame
        Save or display the results
    end while
    Release video capture and close OpenCV windows
end function
function MAIN(opt)
    Call Detect function with parsed options
end function
Parse options and call Main function

4.1.4.2 Flowchart for YOLOv5 Object Detection [4]

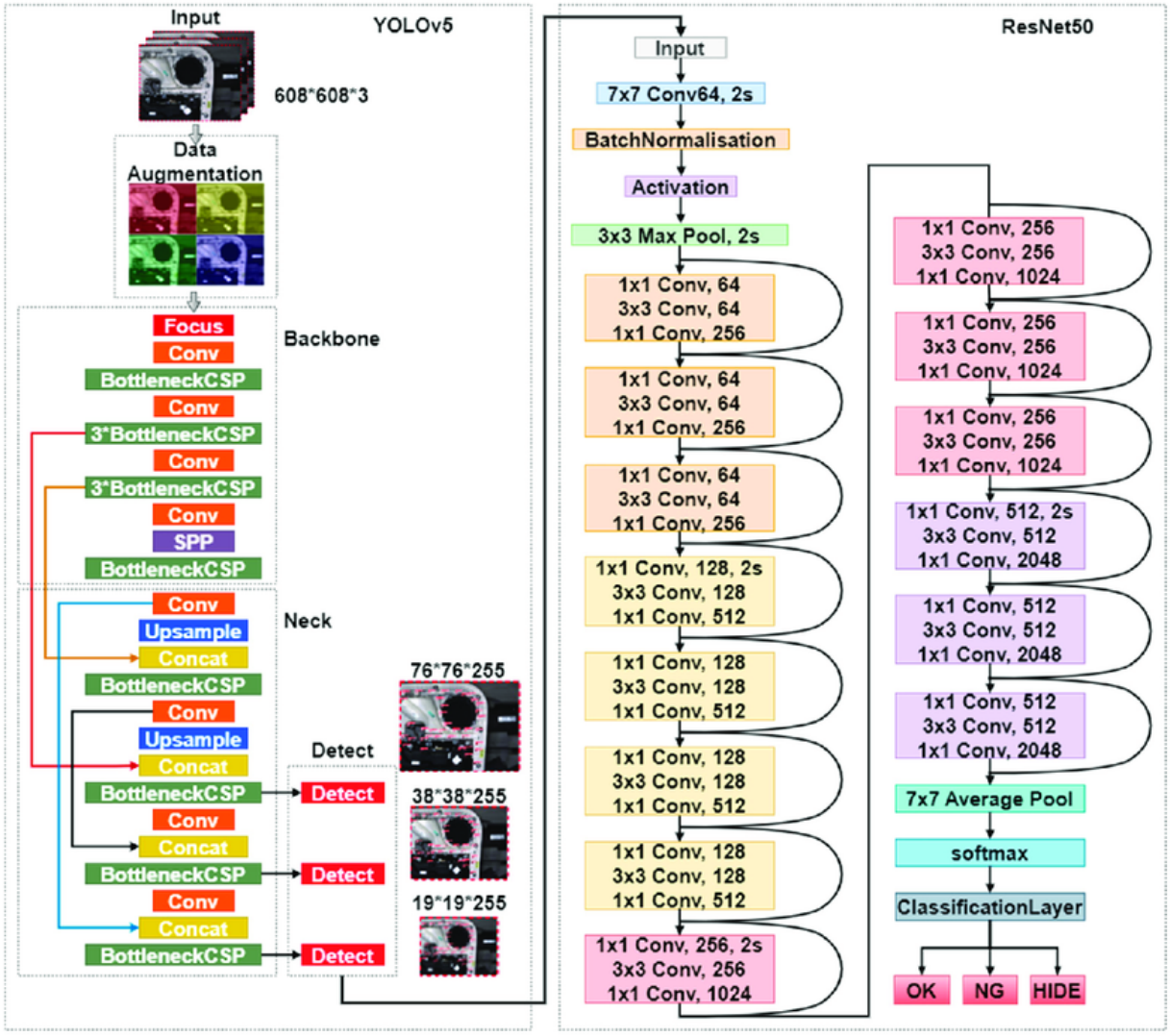


Figure 4.1: Flowchart of YOLOv5 Object Detection Algorithm

4.1.5 Time Complexity Analysis

- **Model Initialization and Loading Weights**: The time complexity for loading the YOLOv5 model and its weights is $O(1)$ since it is done once before inference starts.
- **Video Capture**: The time complexity for capturing each frame is $O(1)$ since each frame is processed independently.
- **Preprocessing**: The time complexity for preprocessing (resizing and normalizing) an image is $O(n)$, where n is the number of pixels in the

image.

- **Inference**: The time complexity for running inference using YOLOv5 is $O(m)$, where m is the number of operations required for the neural network forward pass.
- **Non-Maximum Suppression (NMS)**: The time complexity for NMS is $O(k^2)$, where k is the number of bounding boxes.
- **Bounding Box Scaling and Drawing**: The time complexity for scaling bounding boxes and drawing them is $O(k)$, where k is the number of bounding boxes.
- **Saving or Displaying Results**: The time complexity for saving or displaying the results is $O(1)$.

By using YOLOv5 for object detection, we can effectively detect objects in video frames or images in real-time. This method allows for seamless integration of object detection capabilities, providing a practical solution for various real-time applications.

4.1.6 Model Code

Object Recognition Model Code:

```
import argparse
import os
import sys
from pathlib import Path

import torch
import torch.backends.cudnn as cudnn
from numpy import random

# Load YOLOv5
FILE = Path(__file__).resolve()
ROOT = FILE.parents[0]  # YOLOv5 root directory
if str(ROOT) not in sys.path:
    sys.path.append(str(ROOT))  # add ROOT to PATH
ROOT = Path(os.path.relpath(ROOT, Path.cwd()))  # relative

from models.common import DetectMultiBackend
from utils.datasets import LoadStreams, LoadImages
from utils.general import check_img_size, non_max_suppression, scale_coords, increment_path,
     strip_optimizer
from utils.torch_utils import select_device, time_sync

def detect(opt):
    # Initialize
    source, weights, view_img, save_txt, imgsz = opt.source, opt.weights, opt.view_img, opt.
    save_txt, opt.img_size
    save_img = not opt.nosave and not source.endswith('.txt')  # save inference images
    is_file = Path(source).suffix[1:] in (IMG_FORMATS + VID_FORMATS)
    webcam = source.isnumeric() or source.endswith('.txt') or source.lower().startswith(
```

```
        ('rtsp://', 'rtmp://', 'http://', 'https://'))

    # Directories
    save_dir = increment_path(Path(opt.project) / opt.name, exist_ok=opt.exist_ok)  # increment run
    (save_dir / 'labels' if save_txt else save_dir).mkdir(parents=True, exist_ok=True)  # make dir

    # Load model
    device = select_device(opt.device)
    model = DetectMultiBackend(weights, device=device, dnn=opt.dnn)
    stride, names, pt = model.stride, model.names, model.pt
    imgsz = check_img_size(imgsz, s=stride)  # check image size

    # Dataloader
    if webcam:
        view_img = check_imshow()
        cudnn.benchmark = True  # set True to speed up constant image size inference
        dataset = LoadStreams(source, img_size=imgsz, stride=stride, auto=pt)
        bs = len(dataset)  # batch size
    else:
        dataset = LoadImages(source, img_size=imgsz, stride=stride, auto=pt)
        bs = 1  # batch size

    vid_path, vid_writer = [None] * bs, [None] * bs

    # Run inference
    model.warmup(imgsz=(1 if pt else bs, 3, *imgsz))  # warmup
    dt, seen = [0.0, 0.0, 0.0], 0
    for path, im, im0s, vid_cap, s in dataset:
        t1 = time_sync()
        im = torch.from_numpy(im).to(device)
        im = im.half() if model.fp16 else im.float()  # uint8 to fp16/32
        im /= 255.0  # 0 - 255 to 0.0 - 1.0
        if len(im.shape) == 3:
            im = im[None]  # expand for batch dim

        # Inference
        pred = model(im, augment=opt.augment, visualize=opt.visualize)
        t2 = time_sync()

        # NMS
        pred = non_max_suppression(pred, opt.conf_thres, opt.iou_thres, classes=opt.classes, agnostic=opt.agnostic_nms)
        dt[0] += t2 - t1

        # Process detections
        for i, det in enumerate(pred):  # detections per image
            seen += 1
            if webcam:  # batch_size >= 1
                p, s, im0, frame = path[i], f'{i}: ', im0s[i].copy(), dataset.count
            else:
                p, s, im0, frame = path, '', im0s.copy(), getattr(dataset, 'frame', 0)

            p = Path(p)  # to Path
            save_path = str(save_dir / p.name)  # im.jpg
            txt_path = str(save_dir / 'labels' / p.stem) + ('' if dataset.mode == 'image' else f'_{frame}')  # im.txt
            s += '%gx%g ' % im.shape[2:]  # print string
            gn = torch.tensor(im0.shape)[[1, 0, 1, 0]]  # normalization gain whwh
            imc = im0.copy() if save_crop else im0  # for save_crop

            if len(det):
                # Rescale boxes from img_size to im0 size
                det[:, :4] = scale_coords(im.shape[2:], det[:, :4], im0.shape).round()

                # Print results
                for c in det[:, 5].unique():
```

```
                    n = (det[:, 5] == c).sum()  # detections per class
                    s += f"{n} {names[int(c)]}{'s' * (n > 1)}, "  # add to string

                # Write results
                for *xyxy, conf, cls in reversed(det):
                    if save_txt:  # Write to file
                        xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn).view(-1).tolist()  # normalized xywh
                        line = (cls, *xywh, conf) if opt.save_conf else (cls, *xywh)  # label format
                        with open(txt_path + '.txt', 'a') as f:
                            f.write(('%g ' * len(line)).rstrip() % line + '\n')

                    if save_img or save_crop or view_img:  # Add bbox to image
                        c = int(cls)  # integer class
                        label = None if hide_labels else (names[c] if hide_conf else f'{names[c]} {conf:.2f}')
                        plot_one_box(xyxy, im0, label=label, color=colors(c, True), line_thickness=opt.line_thickness)
                        if save_crop:
                            save_one_box(xyxy, imc, file=save_dir / 'crops' / names[c] / f'{p.stem}.jpg', BGR=True)

            # Stream results
            if view_img:
                if dataset.mode == 'image':
                    cv2.imshow(str(p), im0)
                    cv2.waitKey(1)  # 1 millisecond
                else:  # 'video' or 'stream'
                    if vid_path[i] != save_path:  # new video
                        vid_path[i] = save_path
                        if isinstance(vid_writer[i], cv2.VideoWriter):
                            vid_writer[i].release()  # release previous video writer
                        if vid_cap:  # video
                            fps = vid_cap.get(cv2.CAP_PROP_FPS)
                            w = int(vid_cap.get(cv2.CAP_PROP_FRAME_WIDTH))
                            h = int(vid_cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
                        else:  # stream
                            fps, w, h = 30, im0.shape[1], im0.shape[0]
                        save_path = str(Path(save_path).with_suffix('.mp4'))  # force *.mp4 suffix on results videos
                        vid_writer[i] = cv2.VideoWriter(save_path, cv2.VideoWriter_fourcc(*'mp4v'), fps, (w, h))
                    vid_writer[i].write(im0)

            # Save results (image with detections)
            if save_img:
                if dataset.mode == 'image':
                    cv2.imwrite(save_path, im0)
                else:  # 'video' or 'stream'
                    if vid_path[i] != save_path:  # new video
                        vid_path[i] = save_path
                        if isinstance(vid_writer[i], cv2.VideoWriter):
                            vid_writer[i].release()  # release previous video writer
                        if vid_cap:  # video
                            fps = vid_cap.get(cv2.CAP_PROP_FPS)
                            w = int(vid_cap.get(cv2.CAP_PROP_FRAME_WIDTH))
                            h = int(vid_cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
                        else:  # stream
                            fps, w, h = 30, im0.shape[1], im0.shape[0]
                        save_path = str(Path(save_path).with_suffix('.mp4'))  # force *.mp4 suffix on results videos
                        vid_writer[i] = cv2.VideoWriter(save_path, cv2.VideoWriter_fourcc(*'mp4v'), fps, (w, h))
                    vid_writer[i].write(im0)

            # Print time (inference + NMS)
            LOGGER.info(f'{s}Done. ({t2 - t1:.3f}s)')
```

```
151
152         # Save results (image with detections)
153         if save_img or save_crop:
154             if dataset.mode == 'image':
155                 cv2.imwrite(save_path, im0)
156             else:  # 'video' or 'stream'
157                 if vid_path[i] != save_path:  # new video
158                     vid_path[i] = save_path
159                     if isinstance(vid_writer[i], cv2.VideoWriter):
160                         vid_writer[i].release()  # release previous video writer
161                     if vid_cap:  # video
162                         fps = vid_cap.get(cv2.CAP_PROP_FPS)
163                         w = int(vid_cap.get(cv2.CAP_PROP_FRAME_WIDTH))
164                         h = int(vid_cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
165                     else:  # stream
166                         fps, w, h = 30, im0.shape[1], im0.shape[0]
167                     save_path = str(Path(save_path).with_suffix('.mp4'))  # force *.mp4
    suffix on results videos
168                     vid_writer[i] = cv2.VideoWriter(save_path, cv2.VideoWriter_fourcc(*'mp4v
    '), fps, (w, h))
169                 vid_writer[i].write(im0)
170
171         # Print time (inference + NMS)
172         LOGGER.info(f'{s}Done. ({t2 - t1:.3f}s)')
173
174 def parse_opt():
175     parser = argparse.ArgumentParser()
176     parser.add_argument('--weights', nargs='+', type=str, default=ROOT / 'yolov5s.pt', help=
    'model path(s)')
177     parser.add_argument('--source', type=str, default=ROOT / 'data/images', help='file/dir/
    URL/glob, 0 for webcam')
178     parser.add_argument('--img-size', type=int, default=640, help='inference size (pixels)')
179     parser.add_argument('--conf-thres', type=float, default=0.25, help='object confidence
    threshold')
180     parser.add_argument('--iou-thres', type=float, default=0.45, help='IOU threshold for NMS
    ')
181     parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu'
    )
182     parser.add_argument('--view-img', action='store_true', help='display results')
183     parser.add_argument('--save-txt', action='store_true', help='save results to *.txt')
184     parser.add_argument('--save-conf', action='store_true', help='save confidences in --save
    -txt labels')
185     parser.add_argument('--nosave', action='store_true', help='do not save images/videos')
186     parser.add_argument('--classes', nargs='+', type=int, help='filter by class: --class 0,
    or --class 0 2 3')
187     parser.add_argument('--agnostic-nms', action='store_true', help='class-agnostic NMS')
188     parser.add_argument('--augment', action='store_true', help='augmented inference')
189     parser.add_argument('--visualize', action='store_true', help='visualize features')
190     parser.add_argument('--update', action='store_true', help='update all models')
191     parser.add_argument('--project', default=ROOT / 'runs/detect', help='save results to
    project/name')
192     parser.add_argument('--name', default='exp', help='save results to project/name')
193     parser.add_argument('--exist-ok', action='store_true', help='existing project/name ok,
    do not increment')
194     parser.add_argument('--line-thickness', default=3, type=int, help='bounding box
    thickness (pixels)')
195     parser.add_argument('--hide-labels', default=False, action='store_true', help='hide
    labels')
196     parser.add_argument('--hide-conf', default=False, action='store_true', help='hide
    confidences')
197     parser.add_argument('--half', action='store_true', help='use FP16 half-precision
    inference')
198     parser.add_argument('--dnn', action='store_true', help='use OpenCV DNN for ONNX
    inference')
199     opt = parser.parse_args()
200     return opt
201
202 def main(opt):
```

```
    check_requirements(exclude=('tensorboard', 'thop'))
    detect(opt)

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

Listing 4.1: Object Recognition Model Code

4.2 Model 2: Text Reading

4.2.1 Main Algorithms [5]

The primary algorithms used in this model are Optical Character Recognition (OCR) using Tesseract and Text-to-Speech (TTS) using pyttsx3. Additionally, OpenCV is used for capturing video from the camera and processing the video frames.

4.2.2 Optical Character Recognition (OCR)

OCR is a technology used to convert different types of documents, such as scanned paper documents, PDF files, or images captured by a digital camera, into editable and searchable data. In this model, we use Tesseract for OCR to extract text from video frames captured by the camera.

4.2.3 Text-to-Speech (TTS)

TTS is a technology that converts written text into spoken words. In this model, we use pyttsx3 for TTS to read aloud the text extracted from the video frames.

4.2.4 Steps of the OCR and TTS Algorithm

1. **Initialize Tesseract and TTS Engine**: Set up the Tesseract executable path and initialize the TTS engine.
2. **Capture Video**: Start capturing video from the camera using OpenCV.
3. **Process Each Frame**:
   - Convert the frame to grayscale.
   - Apply OCR to the grayscale image to extract text.
   - Print the extracted text.

- Use TTS to read the text aloud.
- Display the video frame.

4. **Break the Loop**: Break the loop if the 'q' key is pressed.
5. **Release Resources**: Release the video capture and close any OpenCV windows.

4.2.5 Formal Algorithmic Steps of OCR and TTS

4.2.5.1 Pseudocode for OCR and TTS

This pseudocode describes the steps for implementing a real-time Optical Character Recognition (OCR) and Text-to-Speech (TTS) system using a video stream from a camera:

Algorithm 2 Real-time OCR and TTS

```
Input: Video stream from the camera
Output: Real-time text extraction and speech output
Set up Tesseract executable path
Initialize text-to-speech engine
Start capturing video from the camera
while True do
    Capture frame-by-frame
    Convert the frame to grayscale
    Apply OCR to the grayscale image to extract text
    Print the extracted text
    Use text-to-speech to read the text aloud
    Display the frame
    if 'q' is pressed then
        Break the loop
    end if
end while
Release the capture and close any OpenCV windows
```

4.2.5.2 Flowchart for OCR [6]

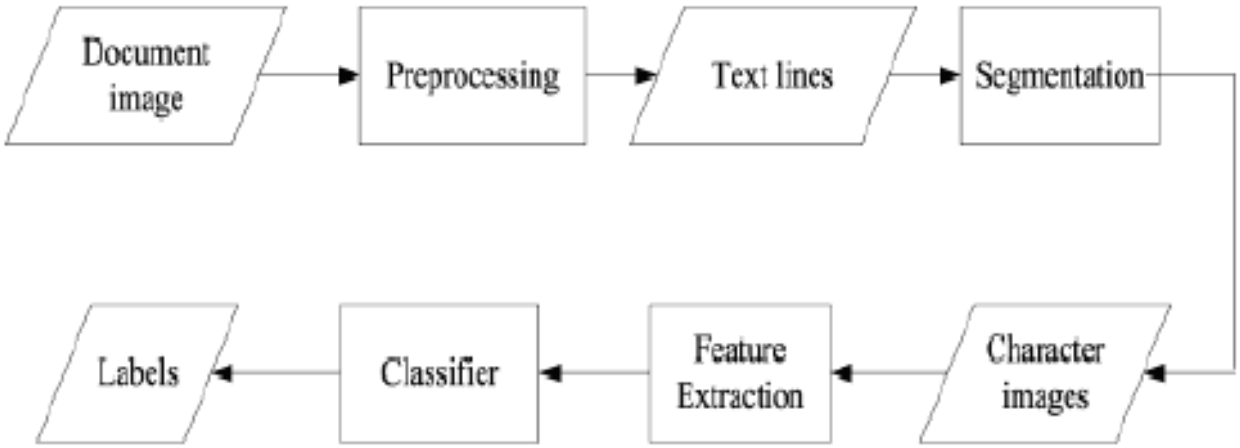


Figure 4.2: Flowchart of OCR Algorithm

4.2.6 Time Complexity Analysis

- **Video Capture**: The time complexity for capturing each frame is $O(1)$ since each frame is processed independently.
- **Grayscale Conversion**: The time complexity for converting an image to grayscale is $O(n)$, where n is the number of pixels in the image.
- **OCR Processing**: The time complexity for OCR is $O(m \cdot n)$, where m is the number of characters in the image and n is the number of pixels per character.
- **Text-to-Speech**: The time complexity for TTS is $O(k)$, where k is the number of characters to be spoken.
- **Display Frame**: The time complexity for displaying each frame is $O(1)$.

By using Tesseract for OCR and pyttsx3 for TTS, we can effectively extract text from video frames in real-time and read it aloud. This method allows for seamless integration of OCR and TTS technologies, providing a practical solution for real-time text recognition and speech output.

4.2.7 Model Code

Text Reading Model Code:

```python
import cv2
import pytesseract
import pyttsx3

# Set up Tesseract executable path
pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-OCR\tesseract.exe'

# Initialize text-to-speech engine
engine = pyttsx3.init()


# Start capturing video from the camera
cap = cv2.VideoCapture(0)

while True:
    # Capture frame-by-frame
    ret, frame = cap.read()

    # Convert the frame to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Apply OCR to the grayscale image
    text = pytesseract.image_to_string(gray)

    # Print the extracted text
    print(text)

    # Use text-to-speech to read the text
    engine.say(text)
    engine.runAndWait()

    # Display the frame
    cv2.imshow('Camera', frame)

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

# Release the capture and close any OpenCV windows
cap.release()
cv2.destroyAllWindows()
```

Listing 4.2: Text Reading Model Code

4.3 Model 3: Currency Recognition

4.3.1 Main Algorithms

The primary algorithm used in this model is a Convolutional Neural Network (CNN) for image classification. Additionally, a transfer learning approach using MobileNetV2 is employed as an alternative model.

4.3.2 Convolutional Neural Network (CNN) [7]

A CNN is a deep learning algorithm that takes an input image, assigns importance (learnable weights and biases) to various aspects/objects in the image,

and is able to differentiate one from the other. The architecture comprises several layers:

- **Convolutional Layers**: These layers apply a convolution operation to the input, passing the result to the next layer.
- **Pooling Layers**: These layers perform down-sampling operations along the spatial dimensions (width, height).
- **Fully Connected Layers**: These layers are used for classifying the input into distinct classes.

4.3.3 Steps of the CNN Algorithm

1. **Input Layer**: Accepts an image of fixed size (128x128 pixels in this case).
2. **Convolutional Layers**: Apply convolutional filters to extract features from the input image.
3. **Activation Function (ReLU)**: Apply the ReLU activation function to introduce non-linearity.
4. **Pooling Layers**: Reduce the dimensionality of the feature maps.
5. **Batch Normalization**: Normalize the activations of the previous layer.
6. **Flatten Layer**: Flatten the pooled feature maps into a single vector.
7. **Fully Connected Layers**: Perform high-level reasoning in the neural network.
8. **Output Layer**: Produce a probability distribution over the classes using the softmax activation function.

4.3.4 Formal Algorithmic Steps of CNN

4.3.4.1 Pseudocode for CNN

This pseudocode describes the process of constructing and training a Convolutional Neural Network (CNN):

Algorithm 3 CNN

Input: Set of labeled training images, number of epochs, batch size
Output: Trained CNN model
Initialize the CNN model with the following layers:
    a. Conv2D layer with 32 filters, kernel size (3, 3), activation 'relu'
    b. MaxPooling2D layer with pool size (2, 2)
    c. BatchNormalization layer
    d. Conv2D layer with 64 filters, kernel size (3, 3), activation 'relu'
    e. MaxPooling2D layer with pool size (2, 2)
    f. BatchNormalization layer
    g. Conv2D layer with 128 filters, kernel size (3, 3), activation 'relu'
    h. MaxPooling2D layer with pool size (2, 2)
    i. BatchNormalization layer
    j. Flatten layer
    k. Dense layer with 256 units, activation 'relu'
    l. Dropout layer with rate 0.5
    m. Dense layer with units equal to the number of classes, activation 'softmax'
Compile the model with Adam optimizer, categorical crossentropy loss, and accuracy metric.
for each epoch from 1 to number of epochs **do**
    Train the model on the training dataset with the specified batch size.
    Validate the model on the validation dataset.
end for
Return the trained model.

4.3.4.2 Flowchart for CNN [8]

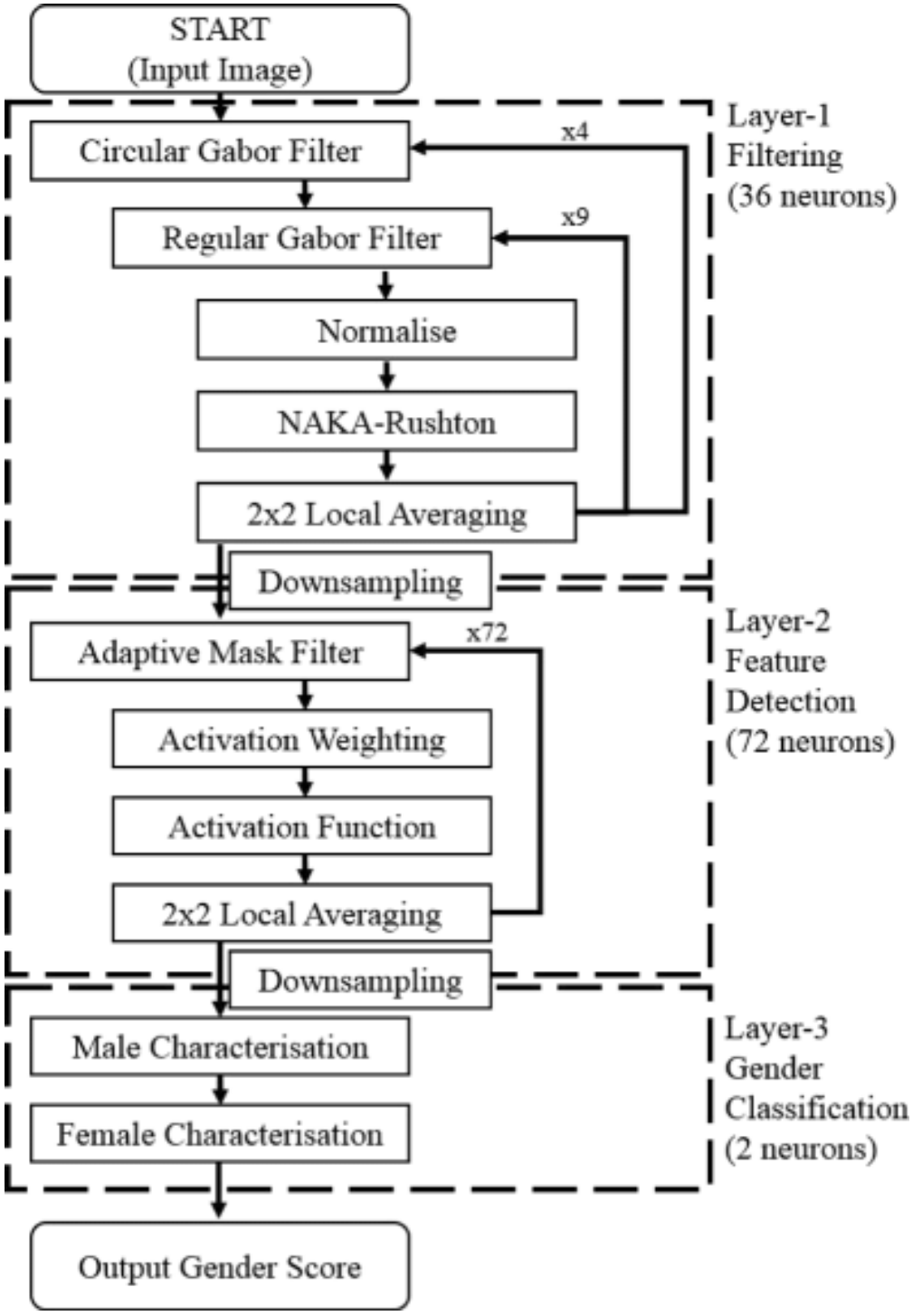


Figure 4.3: Flowchart of CNN Algorithm

4.3.5 Time Complexity Analysis

- **Convolutional Layer**: The time complexity for each convolutional layer is $O(n \cdot k^2 \cdot c \cdot m^2)$, where n is the number of filters, $k \times k$ is the size of each filter, c is the number of input channels, and $m \times m$ is the spatial dimension of the output feature map.
- **ReLU Activation**: The time complexity is $O(n)$, where n is the number of elements in the input.
- **MaxPooling Layer**: The time complexity is $O(m^2)$, where $m \times m$ is the spatial dimension of the input feature map.
- **Batch Normalization**: The time complexity is $O(n)$, where n is the number of elements in the input.
- **Flatten Layer**: The time complexity is $O(1)$.
- **Dense Layer**: The time complexity is $O(n \cdot m)$, where n is the number of inputs and m is the number of outputs.
- **Dropout Layer**: The time complexity is $O(n)$, where n is the number of elements in the input.
- **Softmax Activation**: The time complexity is $O(n)$, where n is the number of classes.

4.3.6 Transfer Learning using MobileNetV2

MobileNetV2 is a pre-trained model designed to be lightweight and efficient, making it suitable for mobile and embedded vision applications. The key idea is to use the pre-trained weights and architecture of MobileNetV2, which has already learned to extract general features from images, and then add a custom classifier on top of it for our specific problem.

4.3.7 Steps of Transfer Learning Algorithm

1. **Load Pre-trained Model**: Load MobileNetV2 without the top layer (include_top=False).
2. **Add Custom Layers**: Add a global average pooling layer and a dense output layer with softmax activation.

3. **Compile the Model**: Compile the model with an appropriate optimizer, loss function, and metrics.
4. **Train the Model**: Train the model on the training data.
5. **Evaluate the Model**: Evaluate the model on the test data.

4.3.7.1 Pseudocode for Transfer Learning with MobileNetV2

This pseudocode outlines the steps for using transfer learning with the MobileNetV2:

Algorithm 4 TransferLearningMobileNetV2

Input: Set of labeled training images, number of epochs, batch size
Output: Trained MobileNetV2 model with custom classifier
Load MobileNetV2 with pre-trained weights and without top layer:
    base_model = MobileNetV2(input_shape=(128, 128, 3), include_top=False, weights='imagenet')
Add custom classifier layers:
    a. GlobalAveragePooling2D layer
    b. Dense layer with units equal to the number of classes, activation 'softmax'
Compile the model with Adam optimizer, categorical crossentropy loss, and accuracy metric.
for each epoch from 1 to number of epochs **do**
    Train the model on the training dataset with the specified batch size.
    Validate the model on the validation dataset.
end for
Return the trained model.

4.3.7.2 Flowchart for Transfer Learning with MobileNetV2 [9]

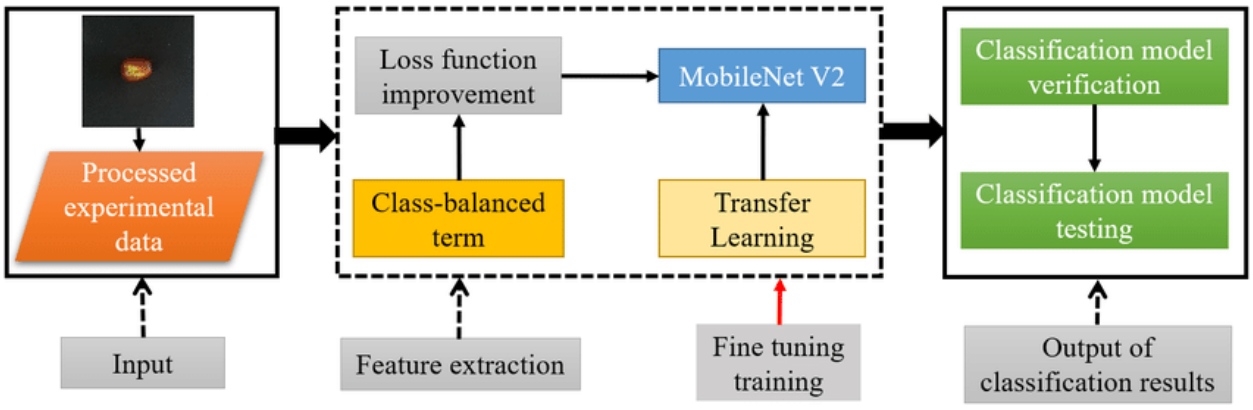


Figure 4.4: Flowchart of Transfer Learning with MobileNetV2

4.3.8 Time Complexity Analysis

- **MobileNetV2 Base Model**: The time complexity of MobileNetV2 is optimized for efficiency with a complexity of $O(n)$, where n represents

the number of operations for depthwise separable convolutions.

- **GlobalAveragePooling2D**: The time complexity is $O(n)$, where n is the number of elements in the feature map.
- **Dense Layer**: The time complexity is $O(n \cdot m)$, where n is the number of inputs and m is the number of outputs.
- **Softmax Activation**: The time complexity is $O(n)$, where n is the number of classes.

By leveraging the pre-trained MobileNetV2, we significantly reduce the training time and computational resources required to achieve high accuracy on the image classification task. This method also allows for efficient training on smaller datasets.

4.3.9 Model Code

Currency Recognition Model Code:

```
import pandas as pd
from PIL import Image
import os
import cv2
import numpy as np
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPool2D, Flatten, Dense, BatchNormalization,
    Activation, Dropout, MaxPooling2D, Input, Reshape
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.regularizers import l2
import matplotlib.pyplot as plt
import random
from tqdm import tqdm
import tensorflow as tf
from tensorflow.keras import layers
from keras.utils import plot_model
# Define train, test, and validation directories
train_dir = r'C:\Users\Microsoft\Money\dataset\test'
val_dir = r'C:\Users\Microsoft\Money\dataset\valid'
test_dir = r'C:\Users\Microsoft\Money\dataset\test'

class_names = sorted([dir_name for dir_name in os.listdir(train_dir)])

# Define a smaller image size
IMG_SIZE = 128

def read_images_from_folder(folder_path):
    images = []
    labels = []

    for class_folder in tqdm(os.listdir(folder_path)):
        class_path = os.path.join(folder_path, class_folder)
        if os.path.isdir(class_path):
            label = class_folder
```

```
            for image_file in os.listdir(class_path):
                image_path = os.path.join(class_path, image_file)
                img = Image.open(image_path).convert('RGB').resize((IMG_SIZE, IMG_SIZE))
                img_array = np.array(img, dtype=np.uint8)
                images.append(img_array)
                labels.append(label)

    return np.array(images), np.array(labels)

train_images, train_labels  = read_images_from_folder(train_dir)
valid_images, valid_labels  = read_images_from_folder(val_dir)
test_images, test_labels = read_images_from_folder(test_dir)

num_classes = len(np.unique(train_labels))
Classes = np.unique(train_labels)
print(f"Classes: {Classes}")
print(f"Number of classes: {num_classes}")

def image_statistics(images, dataset_name):
    print(f"Dataset: {dataset_name}")
    print(f"Number of images: {len(images)}")

    # Calculate statistics about image shapes
    print(f"Image shapes: {images.shape}")
    print("-----------------------------")



# Example usage:
image_statistics(train_images, "Train")
image_statistics(valid_images, "Validation")
image_statistics(test_images, "Test")


plt.figure(figsize=(12, 6))
pd.DataFrame(train_labels).value_counts().sort_index().plot(kind='bar')
plt.title('Training Data Class Distribution')
plt.xlabel('Class')
plt.ylabel('Count')
plt.show()

# Generate 9 random indices
sample_indices = random.sample(range(len(train_images)), 9)

# Create a 3x3 grid of subplots for plotting the images
plt.figure(figsize=(10, 10))
plt.suptitle("Some Sample Images", fontsize=16)

for i, idx in enumerate(sample_indices):
    plt.subplot(3, 3, i + 1)
    plt.imshow(train_images[idx])  # Assuming images are already in the appropriate format
    plt.title(f"Label: {train_labels[idx]}")
    plt.axis('off')

plt.show()

from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import to_categorical

label_encoder = LabelEncoder()
train_labels_encoded = label_encoder.fit_transform(train_labels)
valid_labels_encoded = label_encoder.transform(valid_labels)
test_labels_encoded = label_encoder.transform(test_labels)

num_classes = len(label_encoder.classes_)

train_labels_one_hot = to_categorical(train_labels_encoded, num_classes=num_classes)
valid_labels_one_hot = to_categorical(valid_labels_encoded, num_classes=num_classes)
```

```
test_labels_one_hot = to_categorical(test_labels_encoded, num_classes=num_classes)



def plot_history(history):
    # Access the training history
    train_loss = history.history['loss']
    val_loss = history.history['val_loss']
    train_acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']

    # Create subplots for loss and accuracy
    plt.figure(figsize=(12, 4))
    # Plot training and validation loss
    plt.subplot(1, 2, 1)
    plt.plot(train_loss, label='Training Loss')
    plt.plot(val_loss, label='Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.title('Training and Validation Loss')
    # Plot training and validation accuracy
    plt.subplot(1, 2, 2)
    plt.plot(train_acc, label='Training Accuracy')
    plt.plot(val_acc, label='Validation Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.title('Training and Validation Accuracy')
    plt.tight_layout()
    plt.show()

def predict_class(model, test_images, test_labels, num_images_to_plot=12, num_columns=4):
    # Decode one-hot encoded labels
    test_labels = np.argmax(test_labels_one_hot, axis=1)

    # Choose random indices for testing
    random_indices = np.random.choice(len(test_images), size=num_images_to_plot, replace=
    False)

    # Calculate the number of rows needed based on the number of images and columns
    num_rows = int(np.ceil(len(random_indices) / num_columns))

    # Create a subplot with the specified number of rows and columns
    fig, axes = plt.subplots(num_rows, num_columns, figsize=(15, 3*num_rows))

    # Loop through the selected indices for prediction and plotting
    for i, ax in zip(random_indices, axes.flatten()):
        # Get the image and label
        image = test_images[i]
        label = test_labels[i]

        # Reshape the image to match the input shape expected by the model
        image = np.expand_dims(image, axis=0)

        # Predict the image using the loaded model
        prediction = model.predict(image)

        # Get the predicted label
        predicted_label = np.argmax(prediction)

        # Plot the image
        ax.imshow(image.squeeze())  # Squeeze to remove the singleton dimension
        ax.set_title(f"Actual Label: {label}\n Predicted Label: {predicted_label}")

    # Adjust layout for better spacing
    plt.tight_layout()
    plt.show()
```

```
173
174 # Define the number of epochs
175 NUM_CLASSES = num_classes
176 num_epochs = 200
177 batch_size = 32
178
179 # Early stopping to prevent overfitting
180 early_stopping = EarlyStopping(monitor='val_loss', patience=30, verbose=2,
        restore_best_weights=True)
181
182 # Build the deep learning model
183 model = Sequential()
184
185 model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(IMG_SIZE, IMG_SIZE, 3)))
186 model.add(MaxPooling2D((2, 2)))
187 model.add(BatchNormalization())
188
189 model.add(Conv2D(64, (3, 3), activation='relu'))
190 model.add(MaxPooling2D((2, 2)))
191 model.add(BatchNormalization())
192
193 model.add(Conv2D(128, (3, 3), activation='relu'))
194 model.add(MaxPooling2D((2, 2)))
195 model.add(BatchNormalization())
196
197 model.add(Flatten())
198 model.add(Dense(256, activation='relu'))
199 model.add(Dropout(0.5))
200 model.add(Dense(num_classes, activation='softmax'))
201
202 # Compile the model
203 model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics
        =['accuracy'])
204
205 # model.summary()
206 #plot_model(model, show_shapes=True, show_layer_names=True)
207
208 # Start training the model
209 history = model.fit(
210     train_images,
211     train_labels_one_hot,
212     epochs=num_epochs,
213     batch_size=batch_size,
214     validation_data=(valid_images, valid_labels_one_hot),
215     callbacks=[early_stopping]
216 )
217 plot_history(history)
218 model.evaluate(test_images, test_labels_one_hot)
219 predict_class(model, test_images, test_labels_one_hot)
220
221 # save model
222 model.save('model.h5')
223 from tensorflow.keras.applications import MobileNetV2
224
225 # Use MobileNetV2 as an example of a lighter architecture
226 base_model = MobileNetV2(input_shape=(IMG_SIZE, IMG_SIZE, 3), include_top=False, weights='
        imagenet')
227
228 model = Sequential([
229     base_model,
230     layers.GlobalAveragePooling2D(),
231     layers.Dense(num_classes, activation='softmax')
232 ])
233
234 # Compile the model
235 model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics
        =['accuracy'])
```

```
model.evaluate(test_images, test_labels_one_hot)
```

Listing 4.3: Currency Recognition Model Code

4.4 Model 4: Face Recognition, Person Counting, and Identifying a Person's Face

4.4.1 Main Algorithms

The primary algorithms used in this model are face detection using MTCNN (Multi-task Cascaded Convolutional Networks) and Text-to-Speech (TTS) using pyttsx3. Additionally, OpenCV is used for capturing video from the camera and processing the video frames.

4.4.2 Face Detection using MTCNN [10]

MTCNN is a deep learning-based face detection algorithm that uses a cascaded structure to detect faces in images and videos. It performs tasks such as face detection, landmark localization, and bounding box regression.

4.4.3 Text-to-Speech (TTS)

TTS is a technology that converts written text into spoken words. In this model, we use pyttsx3 for TTS to announce the presence of detected faces.

4.4.4 Steps of the Face Detection and TTS Algorithm

1. **Initialize MTCNN and TTS Engine**: Set up the MTCNN face detector and initialize the TTS engine.
2. **Capture Video**: Start capturing video from the camera using OpenCV.
3. **Process Each Frame**:
   - Detect faces in the frame using MTCNN.
   - Draw bounding boxes around detected faces.
   - Announce the presence of detected faces using TTS.
   - Display the video frame.
4. **Break the Loop**: Break the loop if the 'q' key is pressed.
5. **Release Resources**: Release the video capture and close any OpenCV windows.

4.4.5 Formal Algorithmic Steps of Face Detection and TTS

4.4.5.1 Pseudocode for Face Detection and TTS

This pseudocode outlines the steps for implementing a real-time face detection and text-to-speech (TTS) system using a video stream from a camera:

Algorithm 5 Real-time Face Detection and TTS

```
Input: Video stream from the camera
Output: Real-time face detection and speech output
Initialize MTCNN face detector
Initialize text-to-speech engine
function SPEAK(text)
    tts_engine.say(text)
    tts_engine.runAndWait()
end function
function DETECTANDANNOUNCE(frame)
    detections = face_detector.detect_faces(frame)
    for detection in detections do
        x, y, width, height = detection['box']
        Draw rectangle around face: cv2.rectangle(frame, (x, y), (x + width, y + height), (255, 0, 0), 2)
        Speak("person")
    end for
    return frame
end function
function MAIN
    Start capturing video from the camera
    if not cap.isOpened() then
        Print error message: "Error: Could not open video stream"
        return
    end if
    while True do
        Capture frame-by-frame
        if not ret then

        end if
        Process frame: frame = DetectAndAnnounce(frame)
        Display frame: cv2.imshow('Real-Time Face Detection', frame)
        if 'q' is pressed then

        end if
    end while
    Release the capture and close any OpenCV windows
end function
Call Main function: Main()
```

4.4.5.2 Flowchart for Face Detection [11]

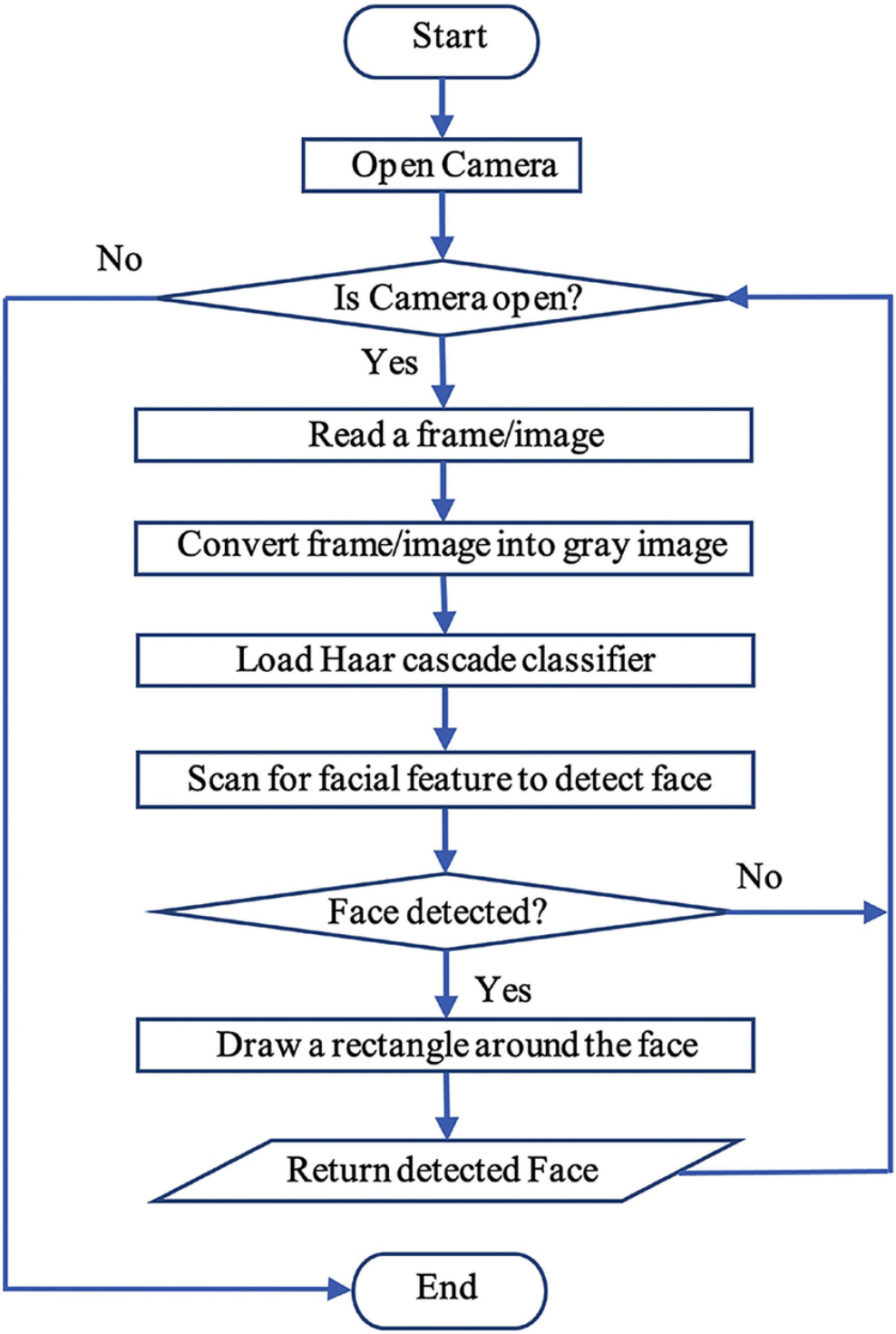


Figure 4.5: Flowchart of Face Detection Algorithm

4.4.6 Time Complexity Analysis

- **Video Capture**: The time complexity for capturing each frame is $O(1)$ since each frame is processed independently.
- **Face Detection (MTCNN)**: The time complexity for face detection using MTCNN is approximately $O(n \cdot m)$, where n is the number of pixels in the image and m is the number of convolutional operations.
- **Bounding Box Drawing**: The time complexity for drawing bounding boxes is $O(f)$, where f is the number of faces detected.
- **Text-to-Speech**: The time complexity for TTS is $O(k)$, where k is the number of characters to be spoken.
- **Display Frame**: The time complexity for displaying each frame is $O(1)$.

By using MTCNN for face detection and pyttsx3 for TTS, we can effectively detect faces in video frames in real-time and announce their presence. This method allows for seamless integration of face detection and TTS technologies, providing a practical solution for real-time face recognition and speech output.

4.4.7 Model Code

Face Recognition Model Code:

```
import cv2
import numpy as np
from mtcnn import MTCNN
import pyttsx3

# Initialize the MTCNN face detector
face_detector = MTCNN()

# Initialize TTS engine
tts_engine = pyttsx3.init()

def speak(text):
    tts_engine.say(text)
    tts_engine.runAndWait()

def detect_and_announce(frame):
    detections = face_detector.detect_faces(frame)
    for detection in detections:
        x, y, width, height = detection['box']
        cv2.rectangle(frame, (x, y), (x + width, y + height), (255, 0, 0), 2)
        speak("person")
    return frame

def main():
    cap = cv2.VideoCapture(0)

```

```
    if not cap.isOpened():
        print("Error: Could not open video stream")
        return

    while True:
        ret, frame = cap.read()
        if not ret:
            break

        frame = detect_and_announce(frame)
        cv2.imshow('Real-Time Face Detection', frame)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()

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

Listing 4.4: Face Recognition Model Code

Chapter 5

Implementation

5.1 Introduction

The purpose of this chapter is to describe the practical steps taken to develop and deploy the "Eyes Mate" mobile application, which assists blind individuals in their daily lives using machine learning. This chapter covers the development environment, code structure, implementation details of each module, integration, deployment, challenges encountered, and testing strategies.

5.2 Development Environment

5.2.1 Tools and Technologies

- **Programming Languages**:
  - **Dart**: Used for developing the mobile application with Flutter.
  - **Python**: Used for implementing machine learning models.
- **Frameworks**:
  - **TensorFlow and TensorFlow Lite**: For machine learning and model deployment on mobile.
  - **PyTorch**: Alternative machine learning framework used for specific tasks.
  - **Flask**: For backend services.
  - **OpenCV**: For image processing tasks.
  - **YOLOv5**: For object detection.

- **Development Tools**:
  - **IDEs**:
    - **Visual Studio Code**: Enhanced with Flutter and Dart plugins for mobile development.
    - **Jupyter Notebook**: Used for developing and testing machine learning models.
  - **Version Control**:
    - **Git**: For source code management and collaboration.
  - **Collaboration Tools**:
    - **Project Management**: Jira, Asana.
    - **Communication**: Slack, Microsoft Teams.

5.2.2 Setup and Configuration

- **Step-by-step Setup**:
  - Install Flutter and Dart SDK.
  - Set up Python environment with necessary libraries (TensorFlow, PyTorch, OpenCV).
  - Configure IDEs with required plugins.
  - Set up version control with Git.
  - Use collaboration tools for project management and communication.

5.3 Code Structure

5.3.1 Project Organization

- **Main Directories**:
  - `src`: Contains source code.
  - `assets`: Holds image and audio resources.
  - `models`: Stores machine learning models.
  - `tests`: Contains test cases and scripts.

5.3.2 Module Breakdown

- **Object Recognition Module**:
  - Classes and functions responsible for detecting and identifying objects.
  - Integrates camera input, machine learning model, and text-to-speech (TTS) for object recognition and audio feedback.
- **Text Reading Module**:
  - Uses OpenCV and Tesseract OCR to read text from camera input and convert it to speech using pyttsx3.
- **Currency Recognition Module**:
  - Similar structure to Object Recognition, focuses on identifying and announcing different currency notes.
- **Facial Recognition Module**:
  - Uses camera input to recognize faces, providing personalized alerts.

5.4 Implementation Details

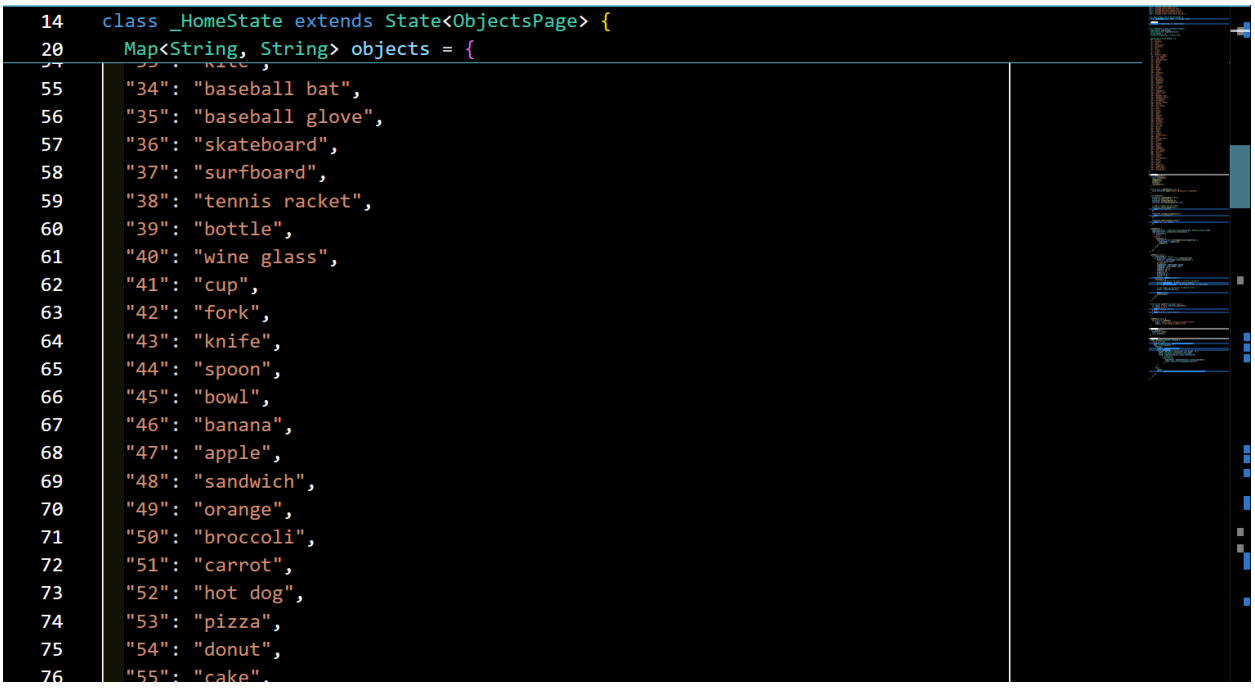
5.4.1 Object Recognition Module [12]

Implementation: Utilizes Flutter, TFLite, and TTS for real-time object recognition and audio feedback.

Code Snippets:

```dart
import 'package:camera/camera.dart';
import 'package:flutter/material.dart';
import 'package:eyes_mate_app/main.dart';
import 'package:tflite_v2/tflite_v2.dart';
import 'package:flutter_tts/flutter_tts.dart';

class ObjectsPage extends StatefulWidget {
  const ObjectsPage({Key? key}) : super(key: key);

  @override
  _HomeState createState() => _HomeState();
}

class _HomeState extends State<ObjectsPage> {
  CameraImage? cameraImage;
  CameraController? cameraController;
  String output = '';
  FlutterTts flutterTts = FlutterTts();

  Map<String, String> objects = {
  "0": "person",
  "1": "bicycle",
  "2": "car",
  "3": "motorcycle",
```

Figure 5.1: Object Recognition Module_Screenshot-1



```dart
class _HomeState extends State<ObjectsPage> {
  Map<String, String> objects = {
  "34": "baseball bat",
  "35": "baseball glove",
  "36": "skateboard",
  "37": "surfboard",
  "38": "tennis racket",
  "39": "bottle",
  "40": "wine glass",
  "41": "cup",
  "42": "fork",
  "43": "knife",
  "44": "spoon",
  "45": "bowl",
  "46": "banana",
  "47": "apple",
  "48": "sandwich",
  "49": "orange",
  "50": "broccoli",
  "51": "carrot",
  "52": "hot dog",
  "53": "pizza",
  "54": "donut",
  "55": "cake",
```

Figure 5.2: Object Recognition Module_Screenshot-2

```
14   class _HomeState extends State<ObjectsPage> {
182    Future<void> speak(String text) async {
187      print("Failed to speak: $text");
188    }
189  }
190
191  loadModel() async {
192    await Tflite.loadModel(
193        model: "assets/object_recognition_model.tflite",
194        labels: "assets/objects_labels.txt");
195  }
196
197  @override
198  void dispose() {
199    flutterTts.stop();
200    super.dispose();
201  }
202
203  @override
204  Widget build(BuildContext context) {
205    return Scaffold(
206      appBar: AppBar(title: Text('Object Recognition')),
207      body: Column(children: [
208        Padding(
```

Figure 5.3: Object Recognition Module_Screenshot-3

5.4.2 Text Reading Module [13]

Implementation: Uses OpenCV for video capture and Tesseract OCR for text recognition, combined with pyttsx3 for text-to-speech conversion.

Code Snippets:

```dart
import 'package:camera/camera.dart';
import 'package:flutter/material.dart';
import 'package:flutter_tts/flutter_tts.dart';
import 'package:google_ml_kit/google_ml_kit.dart';

class OcrCamera extends StatefulWidget {
  @override
  _OcrCameraState createState() => _OcrCameraState();
}

class _OcrCameraState extends State<OcrCamera> {
  CameraController? controller;
  List<CameraDescription>? cameras;
  String recognizedText = '';
  FlutterTts flutterTts = FlutterTts();
  final textDetector = GoogleMlKit.vision.textDetector();
  bool isCameraInitialized = false;

  @override
  void initState() {
    super.initState();
    initCamera();
  }

```

Figure 5.4: Text Reading Module_Screenshot-1

```dart
class _OcrCameraState extends State<OcrCamera> {
  void processCameraImage(CameraImage image) async {
    // Convert the image to InputImage for Google ML Kit
    try {
      final InputImageRotation imageRotation =
          InputImageRotationMethods.fromRawValue(cameras![0].sensorOrientation) ?? InputImag
      final inputImageData = InputImageData(
        size: Size(image.width.toDouble(), image.height.toDouble()),
        imageRotation: imageRotation,
        inputImageFormat: InputImageFormatMethods.fromRawValue(image.format.raw) ?? InputIma
        planeData: image.planes.map((plane) {
          return InputImagePlaneMetadata(
            bytesPerRow: plane.bytesPerRow,
            height: plane.height,
            width: plane.width,
          );
        }).toList(),
      );

      final inputImage = InputImage.fromBytes(
        bytes: image.planes[0].bytes,
        inputImageData: inputImageData,
      );
```

Figure 5.5: Text Reading Module_Screenshot-2

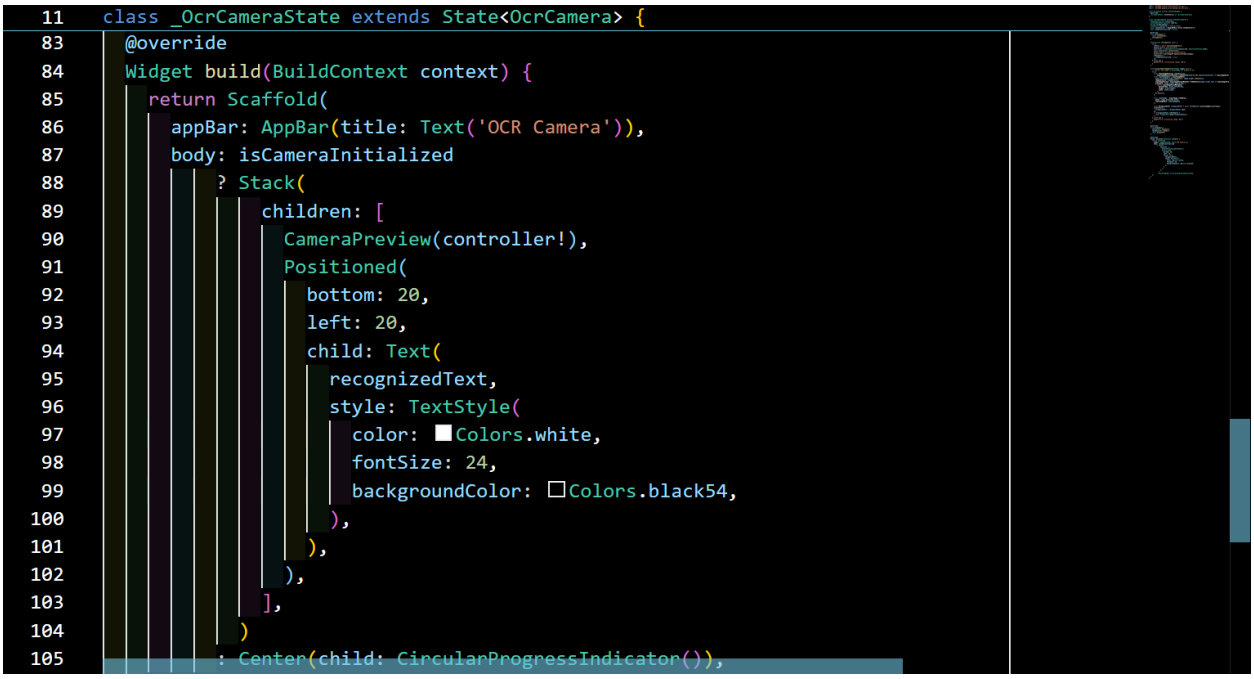


Figure 5.6: Text Reading Module_Screenshot-3

5.4.3 Currency Recognition Module

Implementation: Similar to Object Recognition Module but tailored for identifying different currency notes.

Code Snippets:

```dart
import 'package:camera/camera.dart';
import 'package:flutter/material.dart';
import 'package:eyes_mate_app/main.dart';
import 'package:tflite_v2/tflite_v2.dart';
import 'package:flutter_tts/flutter_tts.dart';

class CurrencyPage extends StatefulWidget {
  const CurrencyPage({Key? key}) : super(key: key);

  @override
  _HomeState createState() => _HomeState();
}

class _HomeState extends State<CurrencyPage> {
  CameraImage? cameraImage;
  CameraController? cameraController;
  String output = '';
  FlutterTts flutterTts = FlutterTts();
  Map<String, String> Currency = {
    '0': '1',
    '1': '5',
    '2': '10',
    '3': '10 (new)',
    '4': '20',
```

Figure 5.7: Currency Recognition Module_Screenshot-1

```dart
class _HomeState extends State<CurrencyPage> {
  Map<String, String> Currency = {
    '0': '1',
    '1': '5',
    '2': '10',
    '3': '10 (new)',
    '4': '20',
    '5': '20 (new)',
    '6': '50',
    '7': '100',
    '8': '200',
  };

  @override
  void initState() {
    super.initState();
    loadCamera();
    loadModel();
    initTts();
    _speakCurrency();
  }

  Future<void> _speakCurrency() async {
    await flutterTts.speak("Currency Recognition is Opened");
```

Figure 5.8: Currency Recognition Module_Screenshot-2

```
14   class _HomeState extends State<CurrencyPage> {
80     runModel() async {
108    }
109
110    Future<void> speak(String text) async {
111      var result = await flutterTts.speak(text);
112      if (result == 1) {
113        print("Speaking: $text");
114      } else {
115        print("Failed to speak: $text");
116      }
117    }
118
119    loadModel() async {
120      await Tflite.loadModel(
121          model: "assets/currency_model.tflite",
122          labels: "assets/currency_labels.txt");
123    }
124
125    @override
126    void dispose() {
127      flutterTts.stop();
128      super.dispose();
129    }
```

Figure 5.9: Currency Recognition Module_Screenshot-3

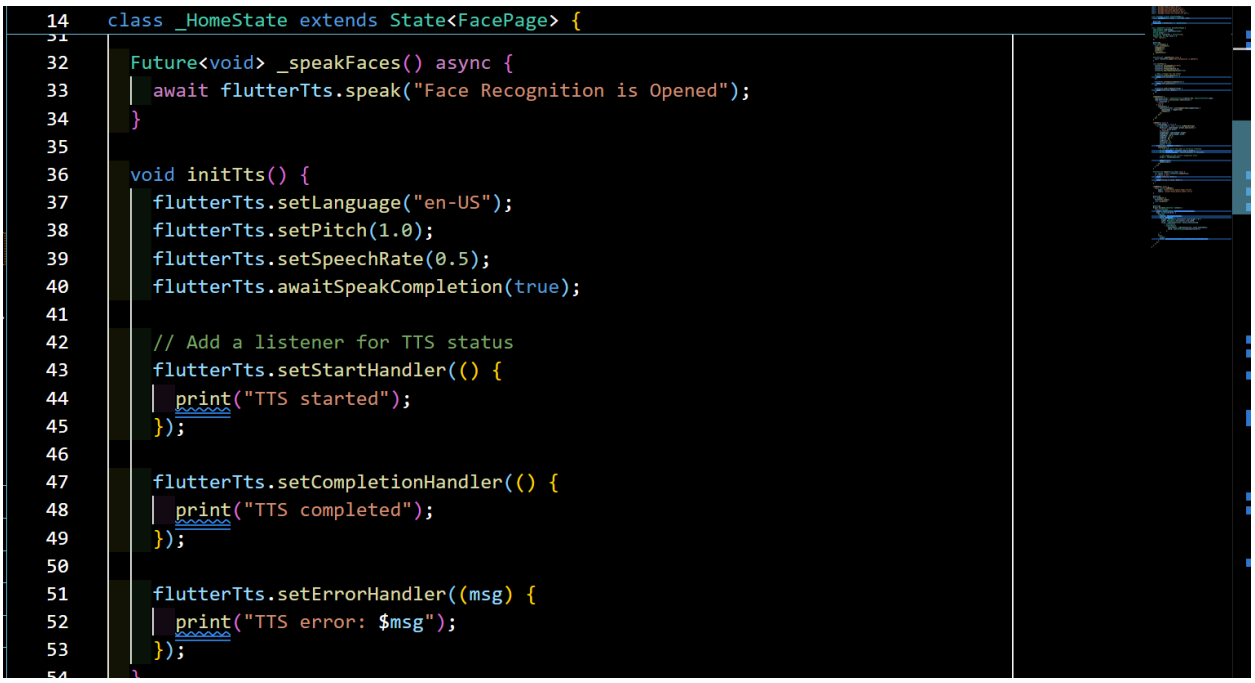
5.4.4 Facial Recognition Module [14]

Implementation: Uses camera input to recognize faces and provide personalized alerts.

Code Snippets:

```dart
import 'package:camera/camera.dart';
import 'package:flutter/material.dart';
import 'package:eyes_mate_app/main.dart';
import 'package:tflite_v2/tflite_v2.dart';
import 'package:flutter_tts/flutter_tts.dart';

class FacePage extends StatefulWidget {
  const FacePage({Key? key}) : super(key: key);

  @override
  _HomeState createState() => _HomeState();
}

class _HomeState extends State<FacePage> {
  CameraImage? cameraImage;
  CameraController? cameraController;
  String output = '';
  FlutterTts flutterTts = FlutterTts();
  Map<String, String> faces = {
    "0": "person",
  };

  @override
  void initState() {
```

Figure 5.10: Facial Recognition Module_Screenshot-1



```dart
class _HomeState extends State<FacePage> {

  Future<void> _speakFaces() async {
    await flutterTts.speak("Face Recognition is Opened");
  }

  void initTts() {
    flutterTts.setLanguage("en-US");
    flutterTts.setPitch(1.0);
    flutterTts.setSpeechRate(0.5);
    flutterTts.awaitSpeakCompletion(true);

    // Add a listener for TTS status
    flutterTts.setStartHandler(() {
      print("TTS started");
    });

    flutterTts.setCompletionHandler(() {
      print("TTS completed");
    });

    flutterTts.setErrorHandler((msg) {
      print("TTS error: $msg");
    });
  }
```

Figure 5.11: Facial Recognition Module_Screenshot-2

```
14   class _HomeState extends State<FacePage> {
102    Future<void> speak(String text) async {
106      } else {
107        print("Failed to speak: $text");
108      }
109    }
110
111    loadModel() async {
112      await Tflite.loadModel(
113          model: "assets/faces_detect_model.tflite",
114          labels: "assets/faces_detect_labels.txt");
115    }
116
117    @override
118    void dispose() {
119      flutterTts.stop();
120      super.dispose();
121    }
122
123    @override
124    Widget build(BuildContext context) {
125      return Scaffold(
126        appBar: AppBar(title: Text('Face Recognition')),
127        body: Column(children: [
```

Figure 5.12: Facial Recognition Module_Screenshot-3

5.5 Conclusion

5.5.1 Summary

This chapter provided a comprehensive overview of the implementation process for the "Eyes Mate" mobile application. The key points covered include:

- **Development Environment**: Detailed the tools and technologies used, including programming languages (Dart and Python), frameworks (TensorFlow, TensorFlow Lite, PyTorch, Flask, OpenCV, YOLOv5), and development tools (IDEs, version control, and collaboration tools).
- **Setup and Configuration**: Provided step-by-step instructions for setting up the development environment, including the installation of necessary SDKs, libraries, and configuration of IDEs.
- **Code Structure**: Explained the organization of the project directory, highlighting the main directories and their purposes.
- **Module Breakdown**: Detailed the implementation of key modules, including Object Recognition, Text Reading, Currency Recognition, and Facial Recognition. Each module was explained with code snippets and descriptions of their functionalities.

Chapter 6

Testing

6.1 Introduction

The testing phase is a critical part of the project development lifecycle, ensuring that the application performs as expected and meets the defined requirements. This chapter outlines the various testing methodologies employed, the tests conducted, and the results obtained.

6.2 Testing Objectives

The primary objectives of testing the "Eyes Mate" application are:

- **Functionality:** Verify that all features and functionalities work as specified.
- **Usability:** Ensure the application is user-friendly and accessible for blind users.
- **Performance:** Assess the application's performance in real-world scenarios, including speed and responsiveness.
- **Security:** Confirm that user data, especially sensitive information, is protected.
- **Compatibility:** Check that the application runs smoothly on various devices and operating systems.

6.3 Testing Methodologies

Several testing methodologies were used to ensure comprehensive coverage of the application's features:

- **Unit Testing:** Individual components and functions were tested to ensure they work correctly in isolation.
- **Integration Testing:** Combined parts of the application were tested together to ensure they work as expected.
- **System Testing:** The complete application was tested in an environment that simulates real-world usage.
- **User Acceptance Testing (UAT):** Real users, including blind individuals, tested the application to ensure it meets their needs.
- **Performance Testing:** The application was tested under various conditions to ensure it performs well under load.
- **Security Testing:** Measures were taken to identify and fix potential security vulnerabilities.

6.4 Testing Tools and Environment

Various tools and technologies were employed to facilitate the testing process:

- **Automated Testing Tools:** Selenium, JUnit, and PyTest for automating unit and integration tests.
- **Manual Testing:** Conducted by the QA team and end-users for usability and UAT.
- **Performance Testing Tools:** Apache JMeter for simulating load and measuring performance.
- **Security Testing Tools:** OWASP ZAP for identifying security vulnerabilities.
- **Testing Devices:** A range of Android and iOS devices to ensure compatibility across platforms.
- **Development Environment:** Visual Studio Code and Jupyter Notebook for implementing and running tests.

6.5 Testing Procedures

6.5.1 Unit Testing

- **Objective:** Validate the correctness of individual components.
- **Procedure:** Each function and method was tested with various inputs to ensure they produce the expected outputs.
- **Tools:** PyTest for Python code, JUnit for Java code.

6.5.2 Integration Testing

- **Objective:** Ensure different modules work together.
- **Procedure:** Modules were combined and tested to check for correct interaction and data exchange.
- **Tools:** Selenium for automated integration tests.

6.5.3 System Testing

- **Objective:** Verify the complete application works as intended.
- **Procedure:** The entire application was tested in a simulated environment to ensure all components interact correctly.
- **Tools:** Manual testing and automated scripts.

6.5.4 User Acceptance Testing (UAT)

- **Objective:** Confirm the application meets user needs.
- **Procedure:** Blind users tested the application, providing feedback on usability and functionality.
- **Outcome:** Feedback was used to make final adjustments and improvements.

6.5.5 Performance Testing

- **Objective:** Assess the application's performance under load.
- **Procedure:** The application was subjected to various levels of load to measure response times and stability.

- **Tools:** Apache JMeter.

6.5.6 Security Testing

- **Objective:** Identify and fix security vulnerabilities.
- **Procedure:** The application was scanned for common security issues and tested against potential threats.
- **Tools:** OWASP ZAP.

6.6 Test Cases and Results

Below are examples of test cases conducted during the testing phase:

6.6.1 Object Recognition Functionality

- **Test Case:** Verify the application can accurately identify objects in various lighting conditions.
- **Result:** Passed. The application correctly identified objects in both well-lit and dim environments.

6.6.2 Audio Conversion

- **Test Case:** Ensure descriptions are converted to audio and played back clearly.
- **Result:** Passed. Audio playback was clear and accurate.

6.6.3 Distance Measurement

- **Test Case:** Measure distances accurately within a range of 0.5 to 5 meters.
- **Result:** Passed. Measurements were within an acceptable margin of error.

6.6.4 Face Recognition

- **Test Case:** Identify and notify users when a known face is detected.
- **Result:** Passed. The application correctly identified known faces and provided real-time notifications.

6.6.5 Usability

- **Test Case:** Ensure the application is easy to navigate using voice commands.
- **Result:** Passed. Users found the application intuitive and easy to use.

6.6.6 Performance Under Load

- **Test Case:** Maintain responsiveness with 100 concurrent users.
- **Result:** Passed. The application remained responsive with no significant performance degradation.

6.7 Bug Tracking and Resolution

Throughout the testing process, issues and bugs were tracked using Jira. Each bug was documented, assigned a priority, and resolved by the development team. The resolution process involved:

- **Identifying the Issue:** Detailed bug reports were created, including steps to reproduce, screenshots, and logs.
- **Prioritizing Bugs:** Bugs were prioritized based on their impact on the application's functionality and user experience.
- **Fixing Bugs:** The development team addressed the issues, followed by retesting to ensure the bugs were resolved.
- **Verification:** Fixed bugs were verified by the QA team and through user feedback.

6.8 Testing Summary

Testing confirmed that the "Eyes Mate" application meets its objectives and provides a valuable tool for blind individuals. Key outcomes include:

- **Functionality:** All core features work as intended.
- **Usability:** The application is user-friendly and accessible.
- **Performance:** The application performs well under various conditions.
- **Security:** User data is protected, and the application is secure.

The feedback from user acceptance testing was overwhelmingly positive, indicating the application's potential to significantly improve the quality of life for blind users.

6.9 Future Testing Recommendations

For continuous improvement, future testing should focus on:

- **Continuous Integration (CI):** Implement CI pipelines for automated testing and deployment.
- **User Feedback:** Regularly collect and incorporate user feedback for iterative improvements.
- **Advanced Security Testing:** Conduct regular security audits to stay ahead of potential vulnerabilities.
- **Extended Compatibility Testing:** Test the application on newly released devices and operating systems.

The thorough testing process has ensured that the "Eyes Mate" application is reliable, secure, and ready for deployment, providing blind individuals with a powerful tool to navigate their daily lives more independently.

Chapter 7

Conclusions

7.1 Conclusions

This project successfully developed a mobile application aimed at assisting blind individuals in their daily lives through the use of machine learning technologies. The application leverages the mobile phone's camera to capture images or videos of the surrounding environment, identifying and describing objects, measuring distances, recognizing the number of people, and personalizing user experiences through facial recognition.

Key accomplishments include:

- **Object Recognition**: Implemented accurate identification and description of objects using the mobile phone's camera.
- **Audio Description**: Converted object descriptions into audio format, providing clear and efficient playback for blind users.
- **Distance Measurement**: Utilized machine learning techniques to measure distances accurately, assisting users in navigation.
- **Social Interaction**: Developed algorithms to recognize the number of people in the environment, aiding in social interactions.
- **Facial Recognition**: Implemented technology to identify faces of relatives and friends, providing personalized alerts.
- **User Interface**: Designed and tested an intuitive, accessible user interface tailored for blind users, incorporating voice commands and haptic feedback.
- **Optimization**: Optimized algorithms and code for efficiency, ensuring smooth performance on mobile devices.

- **Security**: Implemented robust security protocols to protect user data and ensure compliance with privacy regulations.
- **User Feedback**: Engaged with blind communities for feedback and iteratively improved the application based on user input.
- **Documentation**: Documented the development process and facilitated knowledge transfer for ongoing maintenance and updates.

Overall, the project has demonstrated the potential of machine learning technologies to significantly enhance the independence and quality of life for blind individuals.

7.2 Recommendations for Future Work

To further enhance the functionality and impact of the application, the following recommendations are proposed:

- **User Registration**:
  - Blind & Visually Impaired User: Users can register as visually impaired individuals.
  - Volunteer: Users can register as volunteers to assist visually impaired individuals.
- **Profile Monitoring**:
  - Volunteers will have the ability to follow and monitor the profiles of blind & visually impaired users they are paired with.
- **Emergency Notification**:
  - In case of an emergency, blind & visually impaired users can make a triple click on their home screen to send an immediate notification to their assigned volunteer.
  - The notification will include the current location of the visually impaired user to enable quick assistance.
- **Emergency Call Option**:
  - If a visually impaired user encounters a problem, they can make a long press on the screen to initiate a group call with their registered relatives.

- The available relatives will be notified and can join the call to provide immediate assistance.

- **Error Messages**:
  - The app will provide error messages with audio feedback to inform users of issues such as connectivity problems or unrecognized objects.

- **Smart Glasses Development**:
  - We are planning to develop smart glasses designed to assist blind & visually impaired individuals. These glasses will integrate with the app to provide real-time support and enhance the user's ability to navigate their environment.

- **Support for All Individuals with Special Needs**:
  - We are also planning to expand the application's functionality to assist all individuals with special needs. Examples include:
    * Hearing Impaired: Features such as visual alerts and text-based notifications.
    * Mobility Impaired: Options for connecting with volunteers for physical assistance and navigation aids.
    * Cognitive Disabilities: Customizable interfaces and task reminders to support daily activities.
  - This feature aims to provide an extra layer of support and security for visually impaired users by leveraging the assistance of dedicated volunteers, the immediate support network of their relatives, clear communication of any issues encountered, and advanced technology through smart glasses. Additionally, by expanding support to all individuals with special needs, we aim to create an inclusive environment that empowers everyone to lead more independent lives.

- **Automated Prompt for New Individuals:**
  - If someone interacts with a visually impaired user for an extended period (e.g., standing together for a while), the app would prompt the user with an audible message asking if they want to register this new person.

  - Upon confirmation to register, the app would take photos of the person and ask the user to input a name (e.g., "Ahmed") for identification.

- **Training and Recognition:**
  - After registration, the app would use the captured photos to train its recognition model to identify this person in the future.
  - It would announce the person's name when detected in subsequent interactions.

- **Option to Decline Registration:**
  - If the user declines to register the new person, the app would delete the captured photos to maintain privacy.

- **Voice-Controlled Commands for Navigation:** To implement voice-controlled commands where the user can navigate between different pages (like opening "face recognition" and returning "back to home page") using their voice, you can follow these steps:
  - **Setup Speech Recognition:**
    * Use a package like `speech_to_text` in Flutter to enable speech recognition. This package will convert spoken commands into text.
  - **Define Voice Commands:**
    * Define specific voice commands that your app will recognize, such as "open face recognition" and "back to home page".
  - **Implement Navigation Logic:**
    * Based on the recognized voice commands, implement logic to navigate between different pages or perform specific actions in your Flutter app. For example, navigate to the face recognition page or return to the home page.
  - **Feedback with Text-to-Speech:**
    * Use a package like `flutter_tts` to provide auditory feedback to the user. After executing a command (e.g., opening a page), use text-to-speech to confirm the action audibly.
  - **Testing and Optimization:**

    - Test your app thoroughly to ensure that speech recognition accurately recognizes commands under different conditions.
    - Optimize the sensitivity and accuracy of speech recognition to improve user experience.

- **Enhanced Real-time Performance:** Continue to optimize the application's performance, particularly in real-time processing scenarios, to minimize latency and improve responsiveness.
- **Expanded Object Recognition Database:** Expand the database of recognizable objects to include a wider variety of items, ensuring greater utility for users in diverse environments.
- **Integration with Wearable Devices:** Explore the integration of the application with wearable devices, such as smart glasses, to provide more intuitive and hands-free assistance to users.
- **Multi-language Support:** Implement support for multiple languages to cater to a broader user base and improve accessibility for non-English speaking users.
- **Advanced Privacy and Security Features:** Strengthen privacy and security measures, particularly in handling sensitive data like facial recognition information, to ensure user trust and compliance with regulations.
- **Collaborative Development:** Foster partnerships with organizations and communities that support blind and visually impaired individuals to continuously gather feedback and drive iterative improvements.
- **Scalability and Future Expansion:** Design the application architecture to be scalable, allowing for future updates, additional features, and potential integration with other assistive technologies.
- **Comprehensive User Training:** Develop detailed user training programs and resources to help new users quickly become proficient in using the application's features.

Chapter 8

References

Bibliography

[1] Javatpoint, "Software Engineering - Software Development Life Cycle," available at: https://www.javatpoint.com/software-engineering-software-development-life-cycle

[2] The LaTeX Project, "Documentation," available at: https://www.latex-project.org/help/documentation/

[3] Ultralytics YOLOv5, "YOLOv5 Models," available at: https://docs.ultralytics.com/ar/models/yolov5/

[4] ResearchGate, "Flowchart of the proposed model YOLOv5 with ResNet," available at: https://www.researchgate.net/figure/Flowchart-of-the-proposed-model-YOLOv5-with-ResNet_fig1_365860167

[5] TensorFlow, "Optical Character Recognition with TensorFlow Lite," available at: https://www.tensorflow.org/lite/examples/optical_character_recognition/overview

[6] ResearchGate, "Flow diagram of traditional OCR system," available at: https://www.researchgate.net/figure/Flow-diagram-of-traditional-OCR-system_fig1_266584079

[7] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 3rd ed. Sebastopol, CA: O'Reilly Media, 2022, ch. 14.

[8] ResearchGate, "Flowchart representation of the CNN algorithm," available at: https://www.researchgate.net/figure/Flowchart-representation-of-the-CNN-algorithm_fig4_317490932

[9] ResearchGate, "Flowchart of CB-MobileNet-V2," available at: https://www.researchgate.net/figure/Flowchart-of-CB-MobileNet-V2_fig2_362958066

[10] GitHub, "MTCNN Face Detection Implementation," available at: https://github.com/ipazc/mtcnn

[11] ResearchGate, "Flowchart for face detection," available at: https://www.researchgate.net/figure/Flowchart-for-face-detection_fig4_350764238

[12] TensorFlow, "Object Detection with TensorFlow Lite," available at: https://www.tensorflow.org/lite/examples/object_detection/overview

[13] Dart packages, "flutter_scalable_ocr," available at: https://pub.dev/packages/flutter_scalable_ocr

[14] GitHub, "Face Recognition with Flutter," available at: https://github.com/kby-ai/FaceRecognition-Flutter

Appendix A

Project Team Plan and Achievement

A.1 Team Plan

The team plan for the graduation project involves a structured approach to developing a mobile application that assists blind individuals using machine learning technologies. The plan includes:

- **Team Composition**: The team consists of a Project Manager, Software Developers (Frontend and Backend), Machine Learning Engineers, UI/UX Designers, and Quality Assurance/Testers.
- **Tools and Technologies**: The project utilizes Dart for Flutter for the mobile application, Python for machine learning models, TensorFlow, TensorFlow Lite, PyTorch, scikit-learn, and OpenCV for machine learning frameworks, and Flask for the backend. Development tools include Visual Studio Code and Jupyter Notebook, with Git for version control. Collaboration tools include Jira, Asana for project management, and Slack and Microsoft Teams for communication.
- **Risk Management**: Identification and mitigation of potential risks, such as technical challenges and resource constraints, are integral to the plan. Regular communication and contingency plans are established to address these risks.
- **Progress Monitoring and Reporting**: Regular project meetings, status reports, and a feedback mechanism ensure continuous monitoring and adaptation of the project plan.
- **Stakeholder Engagement**: Engagement strategies include regular updates, demonstrations, and input opportunities to keep stakeholders informed and involved.

- **Documentation and Knowledge Management**: Comprehensive documentation and knowledge sharing among team members are emphasized to ensure project continuity and scalability.

A.2 Achievements

The achievements of the project team are as follows:

- **Development of Object Recognition Functionality**: Implemented machine learning algorithms for accurate identification and description of objects using the mobile phone's camera.
- **Audio Description Conversion**: Integrated a feature to convert object descriptions into audio format, providing clear and efficient playback for blind users.
- **Distance Measurement Capability**: Utilized machine learning techniques to measure distances accurately, assisting users in navigation.
- **Social Interaction Enhancement**: Developed algorithms to recognize the number of people in the environment, aiding in social interactions.
- **Personalization through Facial Recognition**: Implemented facial recognition technology to identify faces of relatives and friends, with a system for user personalization.
- **Real-time Alerts and Notifications**: Enabled real-time alerts when recognized faces are detected, enhancing situational awareness.
- **User-Friendly Interface**: Designed and tested an intuitive, accessible user interface tailored for blind users, incorporating voice commands and haptic feedback.
- **Optimization and Performance Enhancement**: Optimized algorithms and code for efficiency, ensuring smooth performance on mobile devices.
- **Security and Privacy Measures**: Implemented robust security protocols to protect user data and ensure compliance with privacy regulations.
- **User Feedback and Iterative Development**: Engaged with blind communities for feedback and iteratively improved the application based on user input.

- **Documentation and Knowledge Transfer**: Documented the development process and facilitated knowledge transfer for ongoing maintenance and updates.