Al-Azhar University

Faculty of Engineering for Girls

Systems and Computers Engineering Department

Intelligent Sign language Manipulation using AI.

"ISMA3NY"

A project is submitted in partial fulfilment for the requirements of the degree of Bachelor of Computer and Systems Engineering.

Presented By**:**

Asmaa Moayad Attia Muhammed	**ID No. 20248624**
Omnia Hasan Ahmed Beker	**ID No. 20248636**
Doaa Muhammed El-Qurany	**ID No. 20248674**
Salma Magdy Saeed Muhammed	**ID No. 20248702**
Salma Wael Abd-Elmejeed Hasan	**ID No. 20248703**
Somia Muhammed Amr-allah Al-sayed	**ID No. 20248706**

Supervised by:

Dr. Momtaz Saad Alkholy

July, 2024

1 Contents

Acknowledgment 7
Abstract 8
1 Introduction 10
1.1 Problem definition 10
1.2 Motivation 11
1.3 Overview 13
1.4 Sign Language and Hand Gesture Recognition 15
1.5 Stakeholders 16
1.6 Literature Review 17
1.7 Expected Project Constraints 18
2 Planning and Requirements 20
2.1 Planning 20
2.2 Requirements 22
2.2.1 Software Requirements 22
2.2.2 Hardware Requirements 22
2.2.3 Functional Requirements 23
2.2.4 Non-functional Requirements 24
2.2.5 User Requirements 25
2.3 Use Cases 26
2.3.1 User login 26
2.3.2 Open Camera for Detection 27
2.3.3 Sign Language Detection 28
2.3.4 Display Recognition Results 29
2.4 Domain Diagram 30
3 Project Analysis and Design 32
3.1 Use case Diagram. 32
3.2 Sequence Diagram 34
3.3 Class Diagram 36
3.4 State Diagram 37
4 Implementation and Testing 40

4.1 Tools & Technologies: 40
4.1.1 AI 40
4.1.2 Android 41
4.2 System Overview 41
4.3 Data Selection and Preparation 41
4.3.1 Dataset Selection 41
4.3.2 Dataset Description 42
4.3.3 Data Preparation 43
4.4 Training Module 45
4.4.1 Model Construction 45
4.4.2 Model Training 46
4.4.3 Model Evaluation and Conversion 48
4.4.4 Making Predictions and Evaluating Performance 49
4.5 Android Application 50
4.6.1 Screens 50
4.5.2 Classes and functionality: 60
4.5.3 Use Firebase 64
4.6 Experimental Results 68
4.7 Sign Language Recognition Results 70
5 Conclusion 72
5.1 Conclusion 72
5.2 Future Work 73

List of figures

Figure 1-1: System Architecture 14
Figure 1-2: Arabic sign language 15
Figure 2-1: Gantt chart 20
Figure 2-2: Domain Object Diagram 30
Figure 3-1: Use Case Diagram 33
Figure 3-2: Sequence Diagram of Sign Language Recognition 34
Figure 3-4: Sequence Diagram of Login and Verification, Open Camera 35
Figure 3-5: Sequence Diagram Capture and Process Gestures 35
Figure 3-6: Sequence Diagram of Convert Text to Speech 35
Figure 3-7: Class Diagram 36
Figure 3-8: State Diagram 37
Figure 3-9: State diagram 38
Figure 3-10: State diagram 38
Figure 4-1: Sample of Dataset 42
Figure 5-1 :Confusion Matrix 68

List of tables

Table 1: User Login 26
Table 2: Open Camera for Detection 27
Table 3: Sign Language Detection 28
Table 4: Display Recognition Results 29

Abbreviations:

TTS	Text-To-Speech
HCL	Human Computer Interface
ASL	American Sign Language
BSL	British Sign Language
ArSL	Arabic Sign Language
HOG	Histogram of Oriented Gradients
SVM	Support Vector Machine
CNN	Convolutional Neural Network
CNNs	Convolutional Neural Networks

Acknowledgment

Praise be to Allah, who guided us to this, and we would never have been guided if Allah had not guided us." first and before everything we would like to thank Allah for His favors and blessings on us. We wish to express our sincere appreciation to our supervisor: ***Dr.Momtaz El-kholy*** for his expert advice, his keenness on our acquisition of knowledge and keeping pace with the development of modern technology methods and his guidance throughout this project. We would also like to thank our department staff (computers and systems engineering department) for their invaluable assistance that they provided during our educational process. We wish to express our deepest gratitude and great love to our families who have kept us going on and this work would not have been possible without their support. moral support and encouragement. They have stood by us in the most difficult of times.

Abstract

The aim of this project is to develop a real - time system for converting sign language into text and speech, facilitating communication between individuals who are deaf or hard of hearing and those who are not familiar with sign language. The system leverages computer vision and natural language processing techniques to accurately recognize and interpret sign language gestures, converting them into corresponding text and generating synthesized speech output.

The system utilizes a camera - based input system, which captures video data of the user performing sign language gestures. The video frames are processed using computer vision algorithms, employing techniques such as object detection and hand pose estimation to accurately recognize hand gestures. The recognized gestures are then mapped to a predefined sign language dictionary, where each gesture corresponds to a specific word or phrase. This mapping is achieved using machine learning algorithms, which have been trained on a large dataset of sign language gesture to identify patterns and map gestures to their corresponding textual representations. Once the gestures are mapped to text, the system generates synthesized speech output using text - to - speech (TTS) technology. The synthesized speech provides an auditory representation of the sign language, enabling individuals who are not familiar with sign language to understand and respond to the communicated messages. The development of this real - time sign language conversion system has the potential to bridge the communication gap between individuals who use sign language and those who do not. By providing an efficient and accurate conversion of sign language into text and speech, this project aims to empower individuals with hearing impairments to communicate more effectively and inclusively in various social and professional settings.

Chapter 1

Introduction

1 Introduction

Speech impaired people use hand signs and gestures to communicate. Normal people face difficulty in understanding their language. Hence there is a need of a system which recognizes the different signs, gestures and conveys the information to the normal people. It bridges the gap between physically challenged people and normal people.

1.1 Problem definition

1. Background

Communication between deaf individuals who use sign language and those who rely on spoken language can be challenging. While interpreters and written communication offer solutions, they are not always convenient or accessible, especially in spontaneous or everyday interactions.

2. Problem Statement

Deaf and hard-of-hearing individuals often face significant communication barriers in various social, educational, and professional settings. These barriers can lead to misunderstandings, exclusion, and a lack of equal opportunities. Traditional methods of bridging this communication gap, such as interpreters or written notes, are not always practical or efficient for real-time, dynamic conversations.

3. Key Challenges

- Real-Time Translation: Existing solutions often fail to provide accurate and immediate translation of sign language to speech, limiting fluid conversation.
- Accessibility and Convenience: Many tools are either too complex to use or not readily available to the general public, restricting their widespread adoption.
- Language Variability: Sign languages are not universal; different regions use different sign languages, necessitating a solution that can handle multiple languages and dialects.
- User Personalization: Users have diverse needs and preferences, requiring customizable features to cater to individual requirements.

- Technological Limitations: Ensuring high accuracy and low latency in translation requires advanced technology that can work reliably in varied lighting and background conditions.

4. Solution Vision

Develop a Sign Language to Speech app that leverages cutting-edge computer vision and machine learning technologies to provide real-time, accurate translation of sign language into spoken words. The app should be user-friendly, highly accessible, and customizable, supporting multiple sign languages and offering voice personalization options.

5. Objectives

- Accuracy: Achieve high accuracy in recognizing and translating sign language gestures.
- Speed: Ensure real-time translation with minimal delay to enable natural conversations.
- User Experience: Design an intuitive and accessible user interface.
- Support for Multiple Languages: Include support for various sign languages and dialects.
- Offline Functionality: Provide offline capabilities for use without an internet connection.

1.2 Motivation

1. Bridging Communication Gaps: Deaf and hard-of-hearing individuals face significant communication barriers in daily interactions with those who rely on spoken language. This app aims to bridge this gap, fostering more inclusive and seamless communication.

2. Promoting Inclusivity: Inclusion in social, educational, and professional settings is often hindered by communication difficulties. By translating sign language into speech in real time, the app helps create a more inclusive environment where everyone can participate equally.

3. Enhancing Independence: Many deaf individuals rely on interpreters or written communication, which can limit spontaneity and independence. The app empowers users to communicate directly and autonomously, enhancing their confidence and independence.

4. Supporting Accessibility: Accessibility is a fundamental right. This app aligns with efforts to make communication accessible to all, ensuring that deaf and hard-of-hearing individuals can engage fully in various aspects of life without barriers.

5. Leveraging Technological Advances: Advancements in computer vision, machine learning, and natural language processing have made it feasible to create sophisticated tools for sign language translation. This app leverages these technologies to provide an accurate, reliable, and user-friendly solution.

6. Real-World Applications: The app addresses real-world needs in multiple scenarios:

- Social Interactions: Facilitates easier communication in social settings, reducing feelings of isolation.
- Educational Settings: Supports students in classrooms by translating their contributions into spoken language, aiding their learning experience.
- Work Environments: Enhances workplace communication, enabling full participation of deaf employees in meetings and discussions.
- Public Services: Improves access to essential services, such as healthcare and customer support, where effective communication is crucial.

7. Raising Awareness: By making interactions with sign language users more commonplace, the app can help raise awareness and understanding of the deaf community and sign language among the hearing population.

8. Addressing Technological Gaps: Existing solutions for sign language translation often lack real-time capabilities or are not user-friendly. This app addresses these gaps by providing a practical, efficient, and accessible solution.

9. Personalization and Adaptability: The app's ability to learn and adapt to individual signing styles over time ensures a personalized and improved user experience, increasing its usefulness and acceptance.

10. Driving Social Change: By making communication easier and more natural, the app contributes to breaking down social barriers, fostering better relationships, and promoting equality and respect for the deaf and hard-of-hearing community.

1.3 Overview

The primary goal of the app is to enable effortless and dynamic conversations by converting sign language gestures into audible speech, making it easier for deaf and hard-of-hearing individuals to interact in various social, educational, and professional contexts.

- Core Features:

    - Real-Time Translation: Uses advanced computer vision and machine learning algorithms to instantly translate sign language gestures into spoken words.
    - User-Friendly Interface: Designed with simplicity in mind, ensuring ease of use for people of all ages and technical abilities.
    - Learning and Adaptation: The app can learn and adapt to individual users' signing styles over time, improving accuracy.

- Technological Components:
    - Computer Vision: Detects and tracks hand and body movements to interpret sign language gestures.
    - Machine Learning: Utilizes neural networks trained on vast datasets of sign language to accurately translate gestures into spoken language.

- User Scenarios:
    - Everyday Conversations: Facilitates real-time communication in casual, daily interactions.
    - Educational Settings: Assists in classrooms by translating students' sign language questions or comments into speech.
    - Professional Environments: Enhances workplace communication, ensuring deaf and hard-of-hearing employees can fully participate in meetings and discussions.
    - Public Services: Improves accessibility in public services like healthcare, banking, and customer support.

- Benefits:
  - Inclusivity: Promotes greater inclusion of deaf and hard-of-hearing individuals in various aspects of life.
  - Convenience: Provides a practical and accessible solution for real-time communication.
  - Empowerment: Enhances the independence and confidence of users by enabling more direct and personal interactions.
  - Awareness: Raises awareness and understanding of sign language among the hearing population.

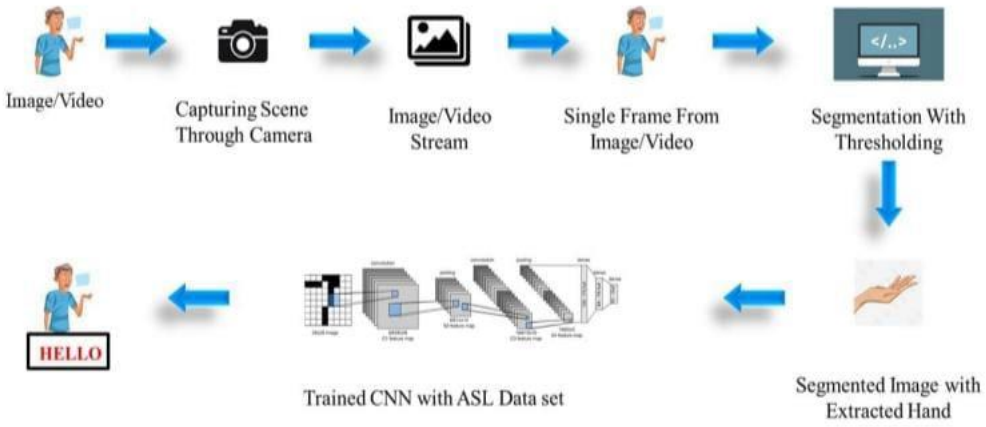


Figure 1-1: System Architecture

1.4 Sign Language and Hand Gesture Recognition

Sign language recognition converts users' signs and gestures into text, bridging communication gaps for the deaf and mute community. It uses image processing and neural networks to map gestures to text, enabling interaction with the general public. This technology addresses the communication challenges faced by deaf and mute individuals, whose interactions are often limited to family or community due to a lack of widespread sign language knowledge. Despite advancements in speech recognition, there are no significant commercial products for sign recognition yet. The project aims to enhance human-computer interaction (HCI) by developing systems that understand gestures using image processing and template matching techniques.

Figure 1-2: Arabic sign language

1.5 Stakeholders

1. Deaf and Hard of Hearing Community
    - Individuals: People who are deaf or hard of hearing who can benefit directly from the app for communication.
    - Families and Friends: Relatives and friends who want to communicate more effectively with their deaf or hard of hearing loved ones.
2. Sign Language Interpreters
    - Professionals who may use the app to enhance their interpreting services or to aid in situations where an interpreter is not available.
3. Educational Institutions
    - Schools for the Deaf: Institutions that cater specifically to deaf students might integrate the app into their curriculum.
    - Universities and Colleges: Higher education institutions offering courses in sign language or special education might find the app useful.
4. Healthcare Providers
    - Hospitals and Clinics: To facilitate communication between healthcare providers and patients who use sign language.
    - Mental Health Professionals: Therapists and counselors who work with deaf and hard of hearing individuals.
5. Government and Non-Governmental Organizations
    - Social Services: Government agencies that provide support services to the deaf and hard of hearing community.

1.6 Literature Review

Research on sign language recognition has seen significant advancements over the past few decades, with a focus on various sign languages including American Sign Language (ASL), British Sign Language (BSL), and others. However, Arabic Sign Language (ArSL) recognition has gained attention more recently. This section reviews some of the notable works in the field of ArSL recognition.

Image-Based Recognition

One of the early approaches to ArSL recognition involves the use of image processing techniques to interpret static hand gestures. Al-Rousan et al. (2018) proposed a system that utilizes Histogram of Oriented Gradients (HOG) features and Support Vector Machines (SVM) for recognizing static Arabic alphabet signs. Their system achieved a commendable accuracy, highlighting the potential of image-based methods for ArSL recognition.

Deep Learning Approaches

With the advent of deep learning, Convolutional Neural Networks (CNNs) have become the cornerstone of modern image recognition systems. El-Sawy et al. (2020) developed a deep CNN model that was trained on a large dataset of ArSL gestures. Their approach significantly outperformed traditional image processing techniques, demonstrating the effectiveness of deep learning in capturing the intricate details of hand gestures.

Another notable contribution is by Hussein et al. (2021), who utilized a combination of CNNs and Recurrent Neural Networks (RNNs) to recognize dynamic sign sequences in ArSL. By leveraging the temporal dependencies between consecutive frames, their model achieved higher accuracy in recognizing continuous sign language sentences.

Sensor-Based Recognition

Beyond image and video data, sensor-based approaches have also been explored for ArSL recognition. Abu-Zaiter et al. (2019) employed data from wearable sensors to capture the motion and orientation of the hand. Their system used a combination of accelerometers and gyroscopes to track hand movements and applied machine learning algorithms for gesture

classification. This method proved to be effective in noisy environments where image-based systems might struggle.

Hybrid Systems

Recent studies have explored hybrid approaches that combine image data with sensor data to improve recognition accuracy and robustness. Al-Jarrah et al. (2022) proposed a hybrid system that integrates CNN-based image recognition with sensor data from gloves equipped with flex sensors. This multi-modal approach leveraged the strengths of both data types, resulting in improved performance in various lighting conditions and complex backgrounds.

1.7 Expected Project Constraints

The following are the expected project constraints for the system:

1. Time constraints: The project may be subject to time constraints, such as deadlines for implementation, testing, and deployment.

2. Financial constraints: The project may be subject to financial constraints, such as budget limitations or the need to generate revenue to cover implementation and maintenance costs.

3. Technical constraints: The project may be subject to technical constraints, such as limitations in the availability of hardware, software, or communication networks.

Chapter 2

Planning and Requirements

2 Planning and Requirements

In this chapter, we will talk about the activities definition, sequencing, and duration estimating using a Gantt chart. We will also cover the software and hardware requirements, as well as the functional and non-functional requirements essential for the development and deployment of the Arabic Sign Language (ArSL) Recognition App.

2.1 Planning

(a) Activities Definition, Sequencing and Duration Estimating

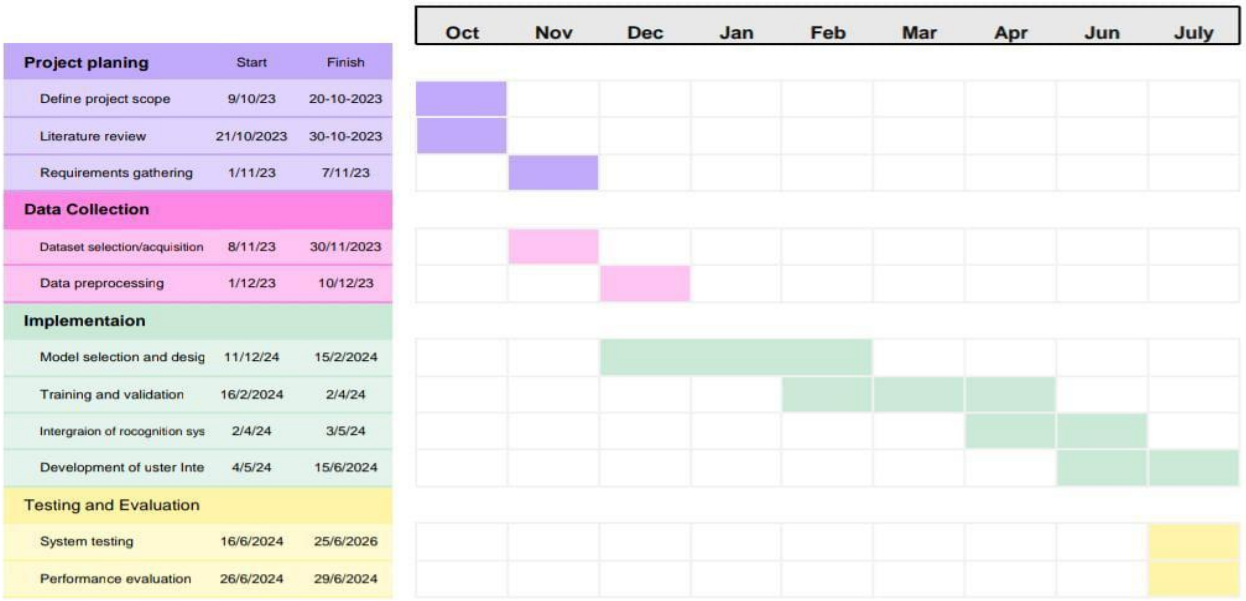


Figure 2-1: Gantt chart

(b) Risk list

- **Technical Risks**:
    - Performance Issues: The app may experience lag or delays, especially on lower-end devices
    - Model Accuracy: The sign language recognition model may not perform accurately across different users and environments.
    - Integration Challenges: Issues might arise when integrating the Python model into the Android app.
- **Data Risks**:
    - Insufficient Data: Lack of sufficient and diverse training data can affect model performance.
    - Data Privacy: Storing and managing user data improperly can lead to privacy violations.
- **Project Management Risks**:
    - Scope Creep: Uncontrolled changes or additions to project scope can lead to delays and budget overruns.
    - Resource Constraints: Limited availability of resources (e.g., developers, hardware) can affect project timelines.
- **External Risks**:
    - Market Competition: Similar apps entering the market can affect the app's success.

2.2 Requirements

2.2.1 Software Requirements

Development Tools

- Integrated Development Environment (IDE) such as VSCode for Python development.
- Android Studio for Android app development.

Programming Languages

- Python for developing the sign language recognition model.
- Java for Android app development.

Frameworks and Libraries

- Machine learning frameworks such as TensorFlow or PyTorch for developing the recognition model.
- Computer vision libraries like OpenCV for video processing.
- Android libraries and SDKs for integrating the model into the app.

2.2.2 Hardware Requirements

Device

- Smartphones or tablets with a built-in camera for capturing video input.
- Sufficient processing power and memory to handle video processing and recognition tasks.

Development Machine

A computer with sufficient resources to run VSCode, Android Studio, and necessary libraries for model development and app deployment.

2.2.3 Functional Requirements

Sign Language Interpretation

- The system shall accurately interpret sign language gestures captured from real time video input.
- It shall support multiple sign language dialects and variations.
- The interpretation process shall be performed in real-time with minimal latency.

Translation:

- 4. The system shall translate interpreted sign language gestures into textual format.
- It shall provide multilingual support for translated text output.
- The translation accuracy shall meet or exceed predefined thresholds.

Speech Synthesis

- The system shall synthesize translated text into audible speech output,
- It shall support multiple languages and accents for speech synthesis.
- Speech synthesis shall be natural-sounding and intelligible.

User Interaction

- 10. The system shall provide an intuitive user interface for initiating and controlling translation processes.
- 11. It shall support touch-based interactions for inputting sign language gestures and navigating the interface.
- 12. The user interface shall be accessible to individuals with disabilities, including support for screen readers and alternative input methods.

2.2.4 Non-functional Requirements

Performance

- The system shall process sign language gestures and deliver translated output in real-time with a maximum latency of X milliseconds.
- It shall be capable of handling concurrent user requests and scaling to accommodate increasing loads.

Security

- The system shall encrypt all communication channels to ensure data confidentiality and integrity.
- It shall implement user authentication mechanisms to prevent unauthorized access to user data.

Accessibility

- The system shall comply with accessibility standards (e.g., WCAG) to ensure usability for individuals with disabilities.
- It shall provide adjustable settings for font size, color contrast, and other accessibility features.

Compatibility

- The system shall be compatible with Android mobile devices running Android OS version X and above.
- It shall support integration with third-party services and platforms for enhanced functionality.

Reliability

- 9. The system shall have a minimum uptime of 99% to ensure continuous availability.
- 10. It shall implement error handling mechanisms to gracefully handle system failures and recover from errors.

2.2.5 User Requirements

User Feedback

- The system shall provide users with the ability to provide feedback on translation accuracy and usability.
- It shall incorporate user feedback to continuously improve translation quality and user experience.

User Training and Support

- The system shall offer user training resources, including tutorials and documentation, to facilitate user onboarding.
- It shall provide responsive customer support channels for addressing user inquiries and technical issues.

2.3 Use Cases

2.3.1 User login

Table 1: User Login

Use Case Name	User Login
Actor	User (anyone who wants to use the ArSL recognition app)
Brief Description	The user logs into the ArSL recognition app using their credentials to access the app's features.
Basic Flow	1) The user opens the app and is presented with the login page. 2) The user enters their username and password. 3) The user clicks the "Login" button. 4) The system verifies the credentials. 5) If the credentials are correct, the user is redirected to the main page.
Precondition	1) The user has a registered account with the app. 2) The app is installed and running on the user's device.
Postcondition	The user is successfully logged in and redirected to the main page of the app.

2.3.2 Open Camera for Detection

Table 2: Open Camera for Detection

Use Case Name	Open Camera for Detection
Actor	User (logged-in user of the ArSL recognition app)
Brief Description	The user navigates to the main page and presses a button to open the camera and start the sign language detection process.
Basic Flow	1) The user is on the main page of the app. 2) The user presses the "Open Camera" button. 3) The app requests permission to access the camera (if not previously granted). 4) The user grants camera access permission. 5) The camera is activated, and the app is ready for detection.
Precondition	1) The user is logged into the app and is on the main page. 2) The device has a working camera.
Postcondition	The camera is activated, and the app is ready to detect and recognize Arabic Sign Language gestures.

2.3.3 Sign Language Detection

Table 3: Sign Language Detection

Use Case Name	Sign Language Detection
Actor	User
Brief Description	The user performs sign language gestures in front of the camera, and the app detects and recognizes these gestures in real-time.
Basic Flow	1) The camera is activated, and the app is in detection mode. 2) The user performs ArSL gestures in front of the camera. 3) The app captures the gestures using the camera. 4) The app processes the captured gestures to recognize the sign language. 5) The app displays the recognized letter corresponding to the gestures in real-time.
Precondition	1) The camera is activated, and the app is in detection mode. 2) The user is within the camera's view and performs clear gestures.
Postcondition	The app successfully recognizes the gestures and displays the corresponding text.

2.3.4 Display Recognition Results

Table 4: Display Recognition Results

Use Case Name	Display Recognition Results
Actor	User
Brief Description	The app displays the recognized sign language gestures as text or speech output to facilitate communication.
Basic Flow	1) The app has successfully detected and recognized sign language gestures. 2) The app converts the recognized gestures into text. 3) The app displays the text on the screen. 4) The app converts the text to speech and plays the audio.
Precondition	The app has successfully detected and recognized sign language gestures.
Postcondition	The recognized text or speech output is displayed to the user, allowing them to communicate effectively.

2.4 Domain Diagram

A domain object diagram, also known as a class diagram in UML, visually represents the classes and their relationships within a system.

For Our project, the key classes and relationships include:

1) **User**: Represents the users of the app, with attributes for identification and methods for logging in and out.
2) **Login Page**: Handles user authentication, displaying the login interface and validating user credentials.
3) **Main Page**: The main interface where the user can access the camera and other features after logging in.
4) **Camera**: Manages the device's camera, capturing gestures for recognition.
5) **Gesture Recognition**: Processes the images captured by the camera to recognize Arabic Sign Language gestures.
6) **Result Display**: Displays the results of the gesture recognition, either as text or spoken output.

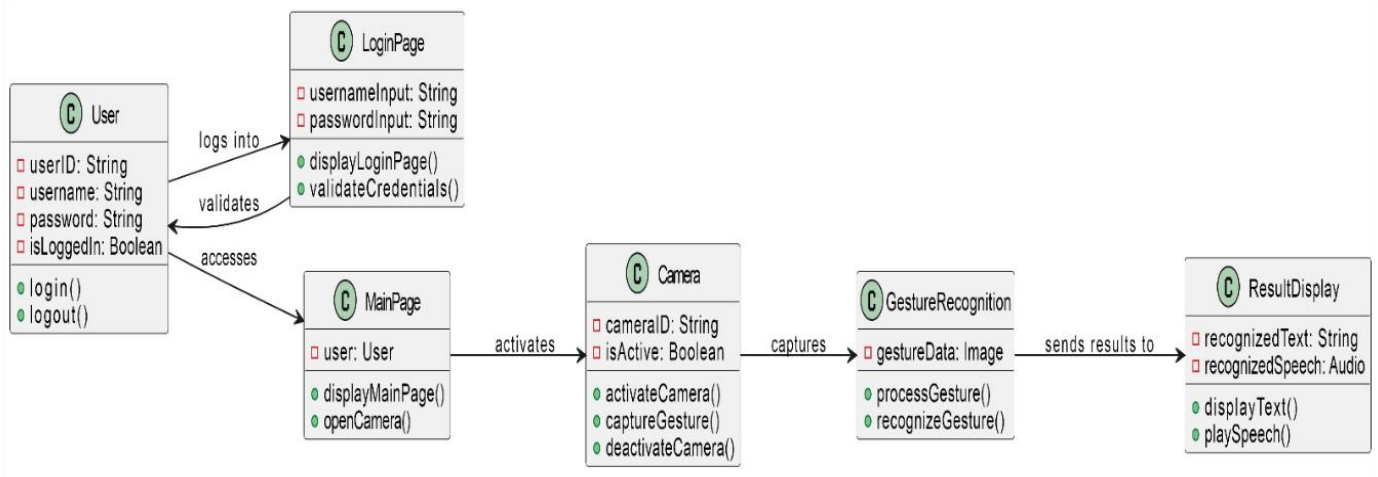


Figure 2-2: Domain Object Diagram

Chapter 3

Project Analysis and Design

3 Project Analysis and Design

3.1 Use case Diagram.

Use Case during requirement elicitation and analysis to represent the functionality of the system. Use case describes a function by the system that yields a visible result for an actor. The identification of actors and use cases result in the definitions of the boundary of the system i.e., differentiating the tasks accomplished by the system and the tasks accomplished by its environment.

The actors are outside the boundary of the system, whereas the use cases are inside the boundary of the system. Use case describes the behavior of the system as seen from the actor ‘s point of view. It describes the function provided by the system as a set of events that yield a visible result for the actor.

The purpose of use case diagram is to capture the dynamic aspect of a system. However, this definition is too generic to describe the purpose, as other four diagrams (activity, sequence, collaboration, and State chart) also have the same purpose. We will look into some specific purpose, which will distinguish it from other four diagrams.

we have to use the following guidelines to draw an efficient use case diagram The name of a use case is very important. The name should be chosen in such a way so that it can identify the functionalities performed. Give a suitable name for actors. Show relationships and dependencies clearly in the diagram. Do not try to include all types of relationships, as the main purpose of the diagram is to identify the requirements. Use notes whenever required to clarify some important points.

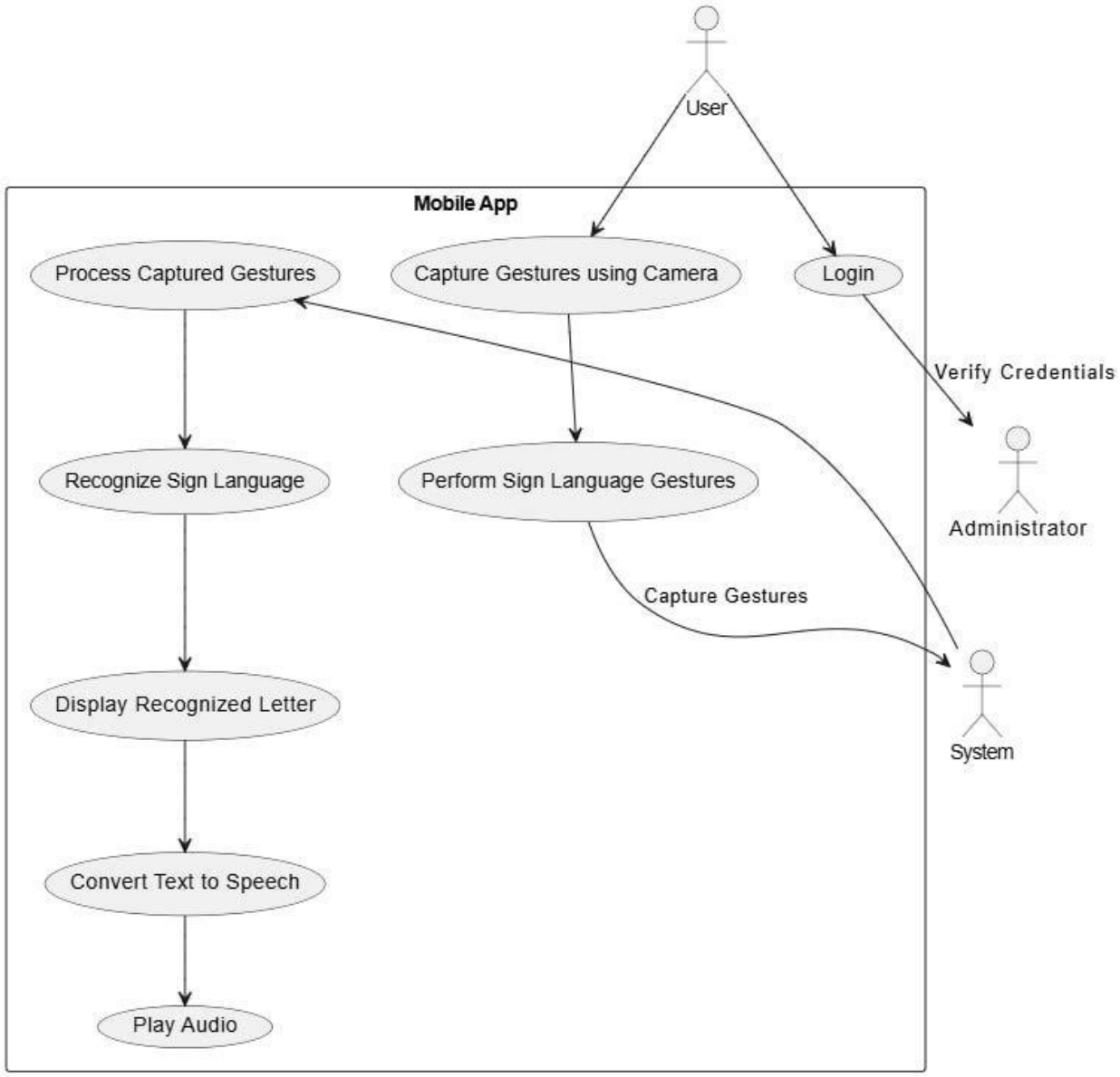


Figure 3-1: Use Case Diagram

3.2 Sequence Diagram

Sequence diagram displays the time sequence of the objects participating in the interaction. This consists of the vertical dimension(time) and horizontal dimension (different objects).

Objects: Object can be viewed as an entity at a particular point in time with specific value and as a holder of identity. A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario.

A sequence diagram shows, as parallel vertical lines (lifelines), different processes or objects that live simultaneously, and as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

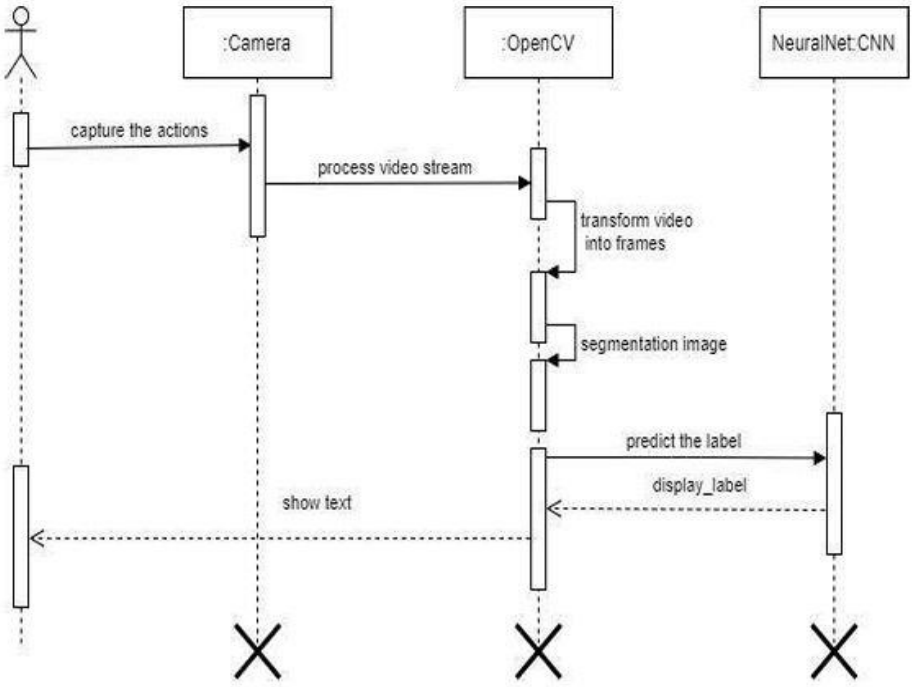


Figure 3-2: Sequence Diagram of Sign Language Recognition

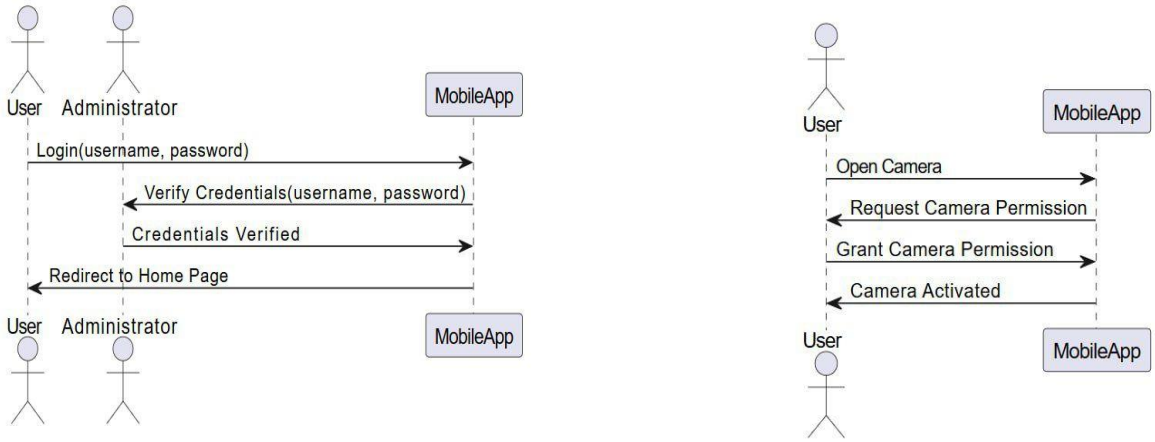


Figure 3-3: Sequence Diagram of Login and Verification, Open Camera

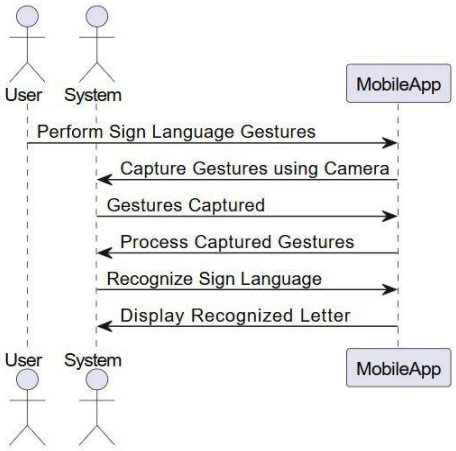


Figure 3-4: Sequence Diagram Capture and Process Gestures

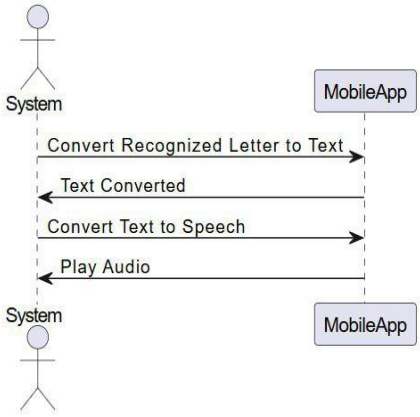


Figure 3-5: Sequence Diagram of Convert Text to Speech

3.3 Class Diagram

Class diagrams model class structure and contents using design elements such as classes, packages and objects. Class diagrams describe the different perspective when designing a system conceptual, specification and implementation. Classes are composed of three things: name, attributes, and operations.

Class diagram also display relationships such as containment, inheritance, association etc. The association relationship is most common relationship in a class diagram. The association shows the relationship between instances of classes.

Class diagrams are the most popular UML diagrams used for construction of software applications. It is very important to learn the drawing procedure of class diagram.

Class diagrams have a lot of properties to consider while drawing but here the diagram will be considered from a top-level view.

The following points should be remembered while drawing a class diagram − The name of the class diagram should be meaningful to describe the aspect of the system. Each element and their relationships should be identified in advance. Responsibility (attributes and methods) of each class should be clearly identified for each class, minimum number of properties should be specified, as unnecessary properties will make the diagram complicated. Use notes whenever required to describe some aspect of the diagram. At the end of the drawing, it should be understandable to the developer/coder. Finally, before making the final version, the diagram should be drawn.

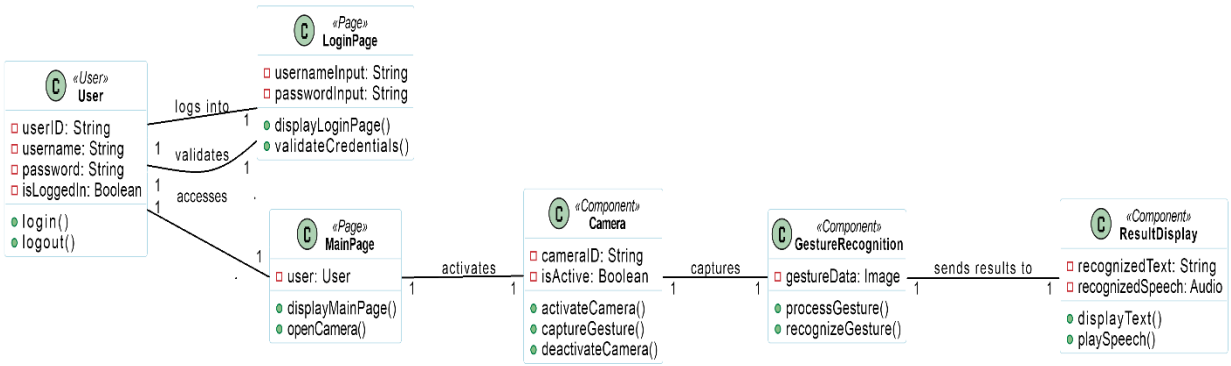


Figure 3-6: Class Diagram

3.4 State Diagram

A state chart diagram describes a state machine which shows the behavior of classes. It shows the actual changes in state not processes or commands that create those changes and is the dynamic behavior of objects over time by model1ing the life cycle of objects of each class. It describes how an object is changing from one state to another state.

There are mainly two states in State Chart Diagram: 1. Initial State 2. Final-State. Some of the components of State Chart Diagram are:

State: It is a condition or situation in life cycle of an object during which it's satisfies same condition or performs some activity or waits for some event.

Transition: It is a relationship between two states indicating that object in first state performs some actions and enters into the next state or event.

Event: An event is specification of significant occurrence that has a location in time and space.

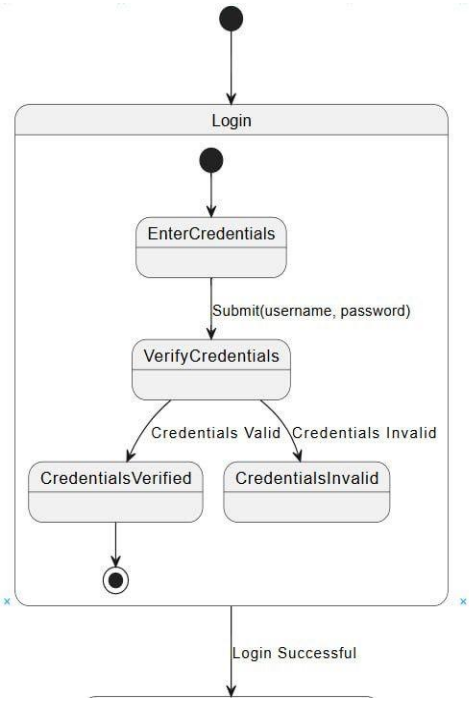


Figure 3-7: State Diagram

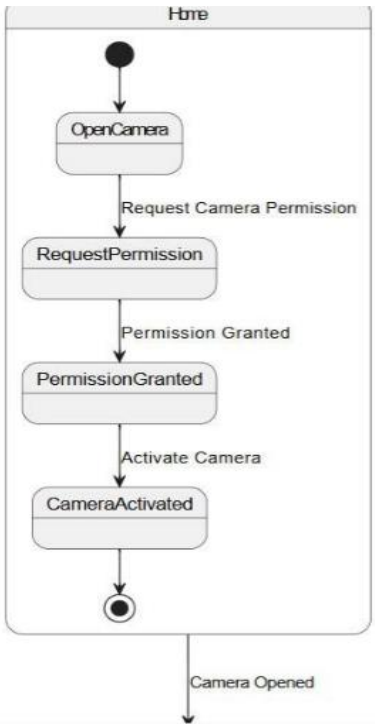


Figure 3-8: State diagram

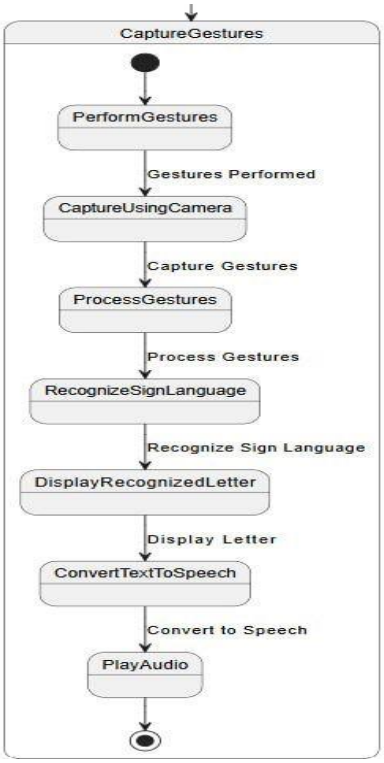


Figure 3-9: State diagram

Chapter 4

Implementation and Testing

4 Implementation and Testing

The purpose of this part is to document the implementation of a sign language recognition system using a Convolutional Neural Network (CNN) model and its deployment on an Android platform. The system aims to facilitate communication for individuals who use Arabic sign language.

4.1 Tools & Technologies:

4.1.1 AI

1. Programming Languages:

    - Python: Used for developing and training the CNN model using libraries like TensorFlow or Keras.

2. Machine Learning Libraries:

    - TensorFlow: Popular open-source library for developing and training deep learning models.
    - Keras: High-level neural networks API, often used with TensorFlow for building and training models.
    - scikit-learn: Useful for data preprocessing, splitting datasets, and evaluating models.

3. Deep Learning Models:

    - Convolutional Neural Network (CNN)**:** Specifically used for image classification tasks like sign language recognition.
    - Pre-trained Models (e.g., EfficientNet): Leveraged for transfer learning to boost model performance with limited data.

4. Data Handling and Augmentation:

    - NumPy: Fundamental package for numerical computing in Python, used for handling large arrays of data.

- PIL (Python Imaging Library) / Pillow: Libraries for opening, manipulating, and saving many different image file formats.
- Augmentor: Python package for augmenting image data to increase dataset size and model robustness.

4.1.2 Android

1. Programming Languages:

   - Java: Used for developing the Android application to deploy the trained model.

2. Deployment Tools:

   - TensorFlow Lite: Lightweight version of TensorFlow designed for mobile and embedded devices, used to deploy machine learning models on Android.
   - Android SDK: Software Development Kit providing necessary tools and APIs for Android app development and deployment.

4.2 System Overview

This section provides a high-level overview of the system architecture, including the main components and their interactions.

- **Components**

  - Dataset Selection and Preparation
  - Model Training and Conversion to TensorFlow Lite
  - Android Application for Real-Time Recognition

4.3 Data Selection and Preparation

4.3.1 Dataset Selection

The dataset was obtained from an existing source rather than being collected manually. It comprises images representing 29 different Arabic letters, with each letter having 201 images in the original dataset. Following data augmentation, each class now contains 2005 images.

4.3.2 Dataset Description

The dataset consists of images for 29 different Arabic sign language gestures:

- Classes: 29 Arabic letters.
- Number of Images:
    - Original: 201 images per class, resulting in a total of 5,628 images.
    - Augmented: 2005 images per class, resulting in a total of 56,140 images.
- Image Resolution: 96x96 pixels.
- Image Format: RGB color images saved in JPEG, PNG and JPG format.
- Augmentation Techniques: Rotation, zoom, width and height shifts, shear transformations, and horizontal flips were applied to enhance the dataset and improve model generalization.

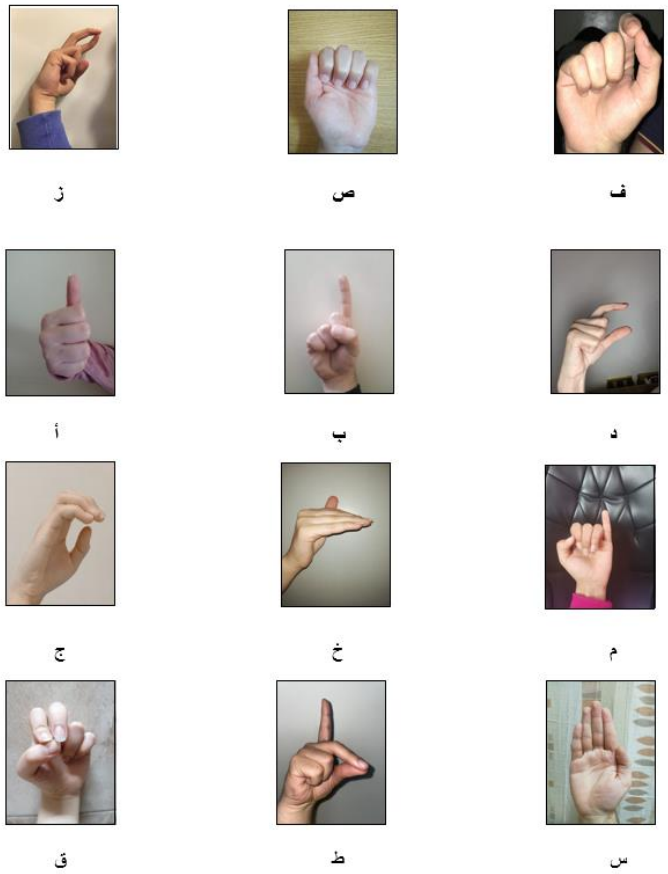


Figure 4-1: Sample of Dataset

4.3.3 Data Preparation

The preparation of the dataset involves several steps to ensure the data is in a suitable format for training the machine learning model. The steps include loading the images, resizing them, converting color formats, normalizing, encoding labels, and splitting the data into training and testing sets.

- **Data Loading and Transformation**
  - Library Imports**:** Import the necessary libraries for data manipulation and image processing.
  - Dataset Path: Define the path to the dataset and list all files.List all the files in the dataset directory and sort them alphabetically for consistency.
  - Initialize arrays: load and process Images: Create empty lists to store image data and labels and Loop through each file, read images, resize to 96x96 pixels, convert to RGB, and append to lists.
  - Convert to Numpy Arrays: Convert image and label lists to numpy arrays.

Code for Data Loading and Transformation

```
import numpy as np
import pandas as pd
import tensorflow as tf
import os
import cv2
import matplotlib. pyplot as plt
from tqdm import tqdm

# now define path to dataset
path=r"F:\0-current_project\DATA"
files=os.listdir(path)
# list of files in path
# sort path from A-Y
files.sort () # not nessesary
# print to see list
print(files)

# create list of image and label
image_array=[]
label_array=[]
# loop through each file in files
for i in  tqdm(range(len(files))):
  sub_file=os.listdir(path+"/"+files[i])
  for j in range (len(sub_file)):
      file_path=path+"/"+files[i]+"/"+sub_file[j]
      image=cv2.imread(file_path)
      image=cv2.resize(image,(96,96))
      image=cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
      image_array.append(image)
      label_array.append(i)
```

- **Data Preprocessing**

  - Normalization**:** Normalize pixel values to the range [0, 1] by dividing by 255.
  - Resizing**:** Resize all images to 96x96 pixels.
  - Augmentation: Apply augmentation techniques such as rotation, zoom, width and height shifts, shear transformations, and horizontal flips to increase the dataset size and improve model robustness.
  - Data Splitting**:** Split the dataset into training and testing sets.

Code for Data Augmentation

```
from keras.preprocessing.image import ImageDataGenerator
datagen = ImageDataGenerator(
        rotation_range=20,     #Random rotation between 0 and 45
        width_shift_range=0.2,   #% shift
        height_shift_range=0.2,
        shear_range=0.05,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='reflect', cval=125)    #Also try nearest, constant, reflect, wrap
import numpy as np
import os
import cv2
from PIL import Image

data=[]
lable=[]

categ =["Hah"] # the class name
data_dir=r"F:\0-current_project\DATA"   #path to the class

for i in categ:
    path = os.path.join(data_dir,i)
    for img in os.listdir(path):
        image_path = os.path.join(path,img)
        image = cv2.imread(image_path)
        if image is None:
            print("Error: Image not loaded. Check the path.")
        rgb_img= cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
        rgb_img = cv2.resize(rgb_img,(200,200))
        data.append(rgb_img)
        lable.append(i)

i = 0
for batch in datagen.flow(x, batch_size=202,
                          save_to_dir='test', #path to store the generated data
                          save_prefix='aug',
                          save_format='png'):
    i += 1
    print(i)
    if i > 4:
        break
```

4.4 Training Module

Supervised machine learning: It is one of the ways of machine learning where the model is trained by input data and expected output data, to create such model, it is necessary to go through the following phases:

1. Model Construction
2. Model Training
3. Model Testing
4. Model Evaluation

4.4.1 Model Construction

In this project, the model construction relies on neural networks, specifically using transfer learning with a pre-trained model. The process involves several steps:

1. Initialize the Model:
    - Begin by creating a Sequential model object.
2. Add Layers:
    - Incorporate a pre-trained model (EfficientNetB0) without the top layer, which excludes the original classification layer.
    - Add a global average pooling layer to reduce the spatial dimensions of the output from the pre-trained model.
    - Include a dropout layer to prevent overfitting by randomly setting a fraction of input units to 0 at each update during training.
    - Add a dense layer with 28 units (corresponding to the 28 classes) and a softmax activation function for classification.
3. Build the Model:
    - Define the input shape for the model.

4. Summarize the Model:

    - Print a summary of the model architecture to review the layers and their parameters.

Code for Model Construction

```
model =Sequential()
pretrained_model=tf.keras.applications.EfficientNetB0(input_shape=(96,96,3),include_top=False)
model.add(pretrained_model)

model.add(layers.GlobalAveragePooling2D())
model.add(layers.Dropout(0.3))

model.add(layers.Dense(28, activation='softmax'))
# model.add(layers.Dense(1))
model.build(input_shape=(None,96,96,3))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 efficientnetb0 (Functional  (None, 3, 3, 1280)        4049571
 )

 global_average_pooling2d (  (None, 1280)              0
 GlobalAveragePooling2D)

 dropout (Dropout)           (None, 1280)              0

 dense (Dense)               (None, 28)                35868

=================================================================
Total params: 4085439 (15.58 MB)
Trainable params: 4043416 (15.42 MB)
Non-trainable params: 42023 (164.16 KB)
```

4.4.2 Model Training

After constructing the model, the next step is to train it using the training data and their corresponding expected outputs. The training process is initiated with the model2.fit method. This method takes the training data and the expected output as inputs and trains the model over a specified number of epochs. The progress of the training is displayed on the console, including the training and validation accuracy and loss for each epoch. At the end of the training phase, the final accuracy of the model is reported.

The following code demonstrates the training process:

1. Compile the Model:

- Configure the model with an optimizer, loss function, and metrics.

2. Define Callbacks:

   - Use ModelCheckpoint to save the best model based on validation accuracy.
   - Use ReduceLROnPlateau to reduce the learning rate when validation accuracy plateaus.

3. Train the Model:

   - Train the model using the training data (X_train, Y_train) and validate it using the validation data (X_test, Y_test).
   - Set the number of epochs and batch size for the training process.

Code for Model Training

```
# Model compilation
model.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])

# Callbacks
ckp_path = "trained_model/model"
model_ckpoint = tf.keras.callbacks.ModelCheckpoint(
    filepath=ckp_path,
    monitor="val_accuracy",  # Monitoring validation accuracy
    mode="auto",
    save_best_only=True,
    save_weights_only=True
)

reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(
    factor=0.9,
    monitor="val_accuracy",  # Monitoring validation accuracy
    mode="auto",
    cooldown=0,
    patience=5,
    verbose=1,
    min_lr=1e-6
)

# Training the model    validation_data=(X_test, Y_test),
Epochs = 10
Batch_size = 32

history = model.fit(
    X_train, Y_train,
    validation_split=0.2,
    batch_size=Batch_size,
    epochs=Epochs,
    callbacks=[model_ckpoint, reduce_lr]
)
```

```
Epoch 1/10
929/929 [==============================] - 1193s 1s/step - loss: 0.2971 - accuracy: 0.9060 - val_loss: 0.2633 - val_accuracy: 0.9173 - lr: 0.0010
Epoch 2/10
929/929 [==============================] - 1126s 1s/step - loss: 0.1808 - accuracy: 0.9418 - val_loss: 0.1950 - val_accuracy: 0.9421 - lr: 0.0010
Epoch 3/10
929/929 [==============================] - 11123s 12s/step - loss: 0.1446 - accuracy: 0.9537 - val_loss: 0.1595 - val_accuracy: 0.9514 - lr: 0.0010
Epoch 4/10
929/929 [==============================] - 1124s 1s/step - loss: 0.1163 - accuracy: 0.9639 - val_loss: 0.2100 - val_accuracy: 0.9342 - lr: 0.0010
Epoch 5/10
929/929 [==============================] - 1123s 1s/step - loss: 0.1048 - accuracy: 0.9668 - val_loss: 0.1454 - val_accuracy: 0.9550 - lr: 0.0010
Epoch 6/10
929/929 [==============================] - 1126s 1s/step - loss: 0.0889 - accuracy: 0.9721 - val_loss: 0.3080 - val_accuracy: 0.9257 - lr: 0.0010
Epoch 7/10
929/929 [==============================] - 1110s 1s/step - loss: 0.0928 - accuracy: 0.9697 - val_loss: 0.1611 - val_accuracy: 0.9541 - lr: 0.0010
Epoch 8/10
929/929 [==============================] - 1113s 1s/step - loss: 0.0768 - accuracy: 0.9755 - val_loss: 0.2142 - val_accuracy: 0.9438 - lr: 0.0010
Epoch 9/10
929/929 [==============================] - 1107s 1s/step - loss: 0.0801 - accuracy: 0.9747 - val_loss: 0.1794 - val_accuracy: 0.9483 - lr: 0.0010
Epoch 10/10
929/929 [==============================] - 1117s 1s/step - loss: 0.0608 - accuracy: 0.9806 - val_loss: 0.1458 - val_accuracy: 0.9617 - lr: 0.0010
```

4.4.3 Model Evaluation and Conversion

1. Evaluate the Model**:**

   We evaluates the model on the test dataset X_test and Y_test with a batch size of 128. It returns the loss value and any metrics specified during the model compilation.

2. Load Model Weights**:**

   We loads the model weights from the checkpoint file specified by ckp_path.

3. Convert the Model to TensorFlow Lite:

   Then we convert the Keras model to TensorFlow Lite format, which is suitable for deployment on mobile and embedded devices.

4. Save the TensorFlow Lite Model**:**

   Saves the converted TensorFlow Lite model to a file named model.tflite.

Code for Model Evaluation and Conversion

```
results = model.evaluate(X_test,Y_test, batch_size=128)
```
✓ 38.5s

```
41/41 [==============================] - 38s 835ms/step - loss: 0.1458 - accuracy: 0.9617
```

```
model.load_weights(ckp_path)
converter=tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model=converter.convert()

with open("model_letter_accuracy_new.tflite","wb") as f:
    f.write(tflite_model)
```

4.4.4 Making Predictions and Evaluating Performance

```
predict_val=model.predict(X_test,batch_size=32)
predicted_classes = np.argmax(predict_val, axis=1)
```

True values

```
class_indices = np.argmax(Y_test, axis=1)
class_indices[:10]
```
✓ 0.0s

```
array([16, 12, 15,  3,  2,  1, 15, 19,  2, 19], dtype=int64)
```

predicted Values

```
predicted_classes[:10]
```
✓ 0.0s

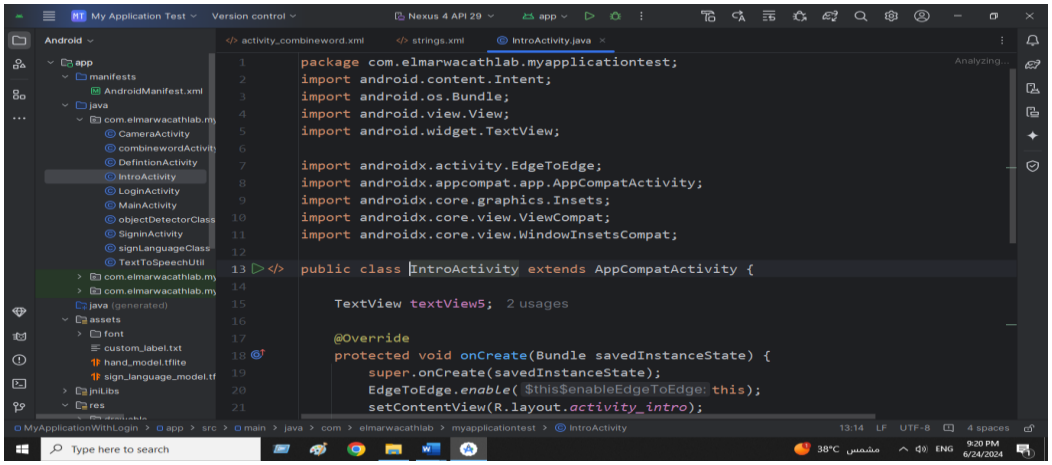
```
array([16, 12, 15,  3,  2,  1, 15, 18,  2, 19], dtype=int64)
```

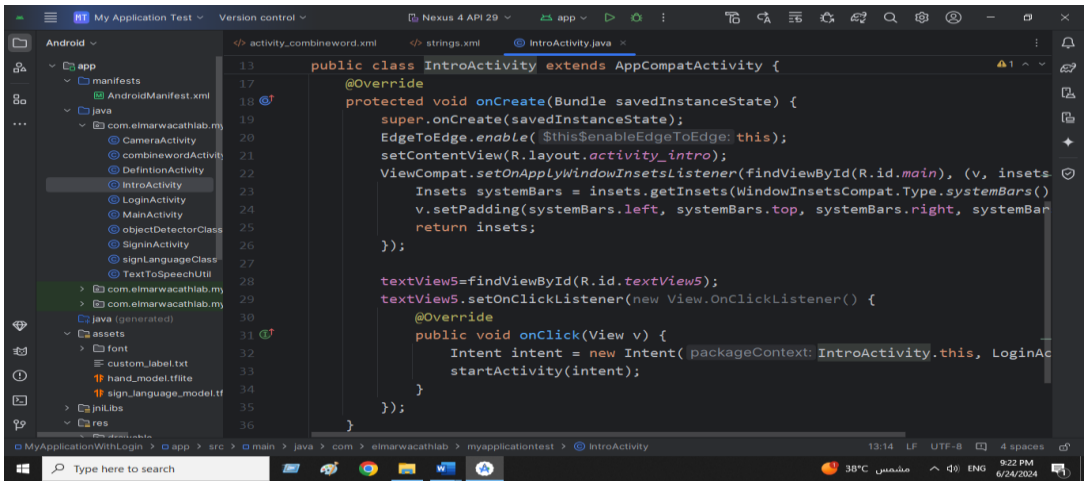
4.5 Android Application

4.6.1 Screens

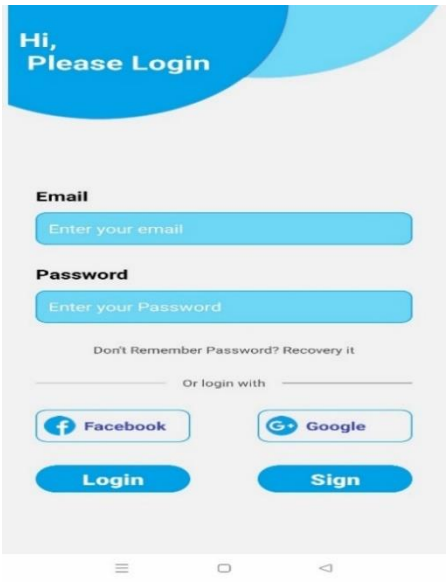
4.5.1.1 screen one:

Intro Screen: It handles the initialization and user interaction for the introductory screen of the application. This screen navigates to Login screen when a specific TextView(GO) is clicked.

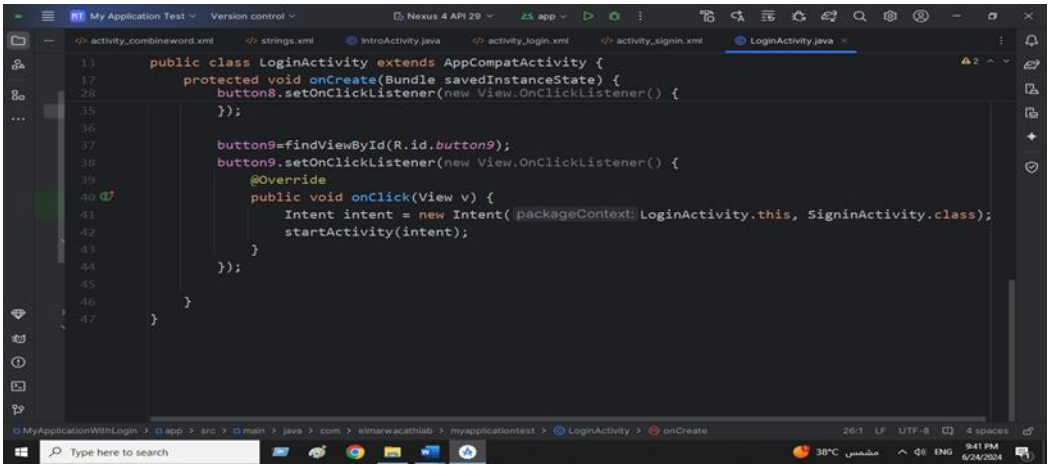




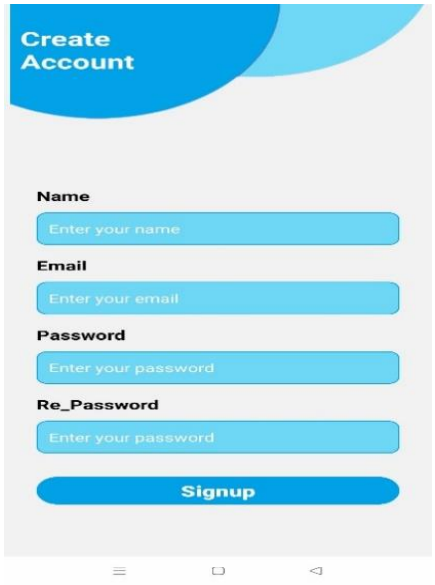
4.5.1.2 screen two:



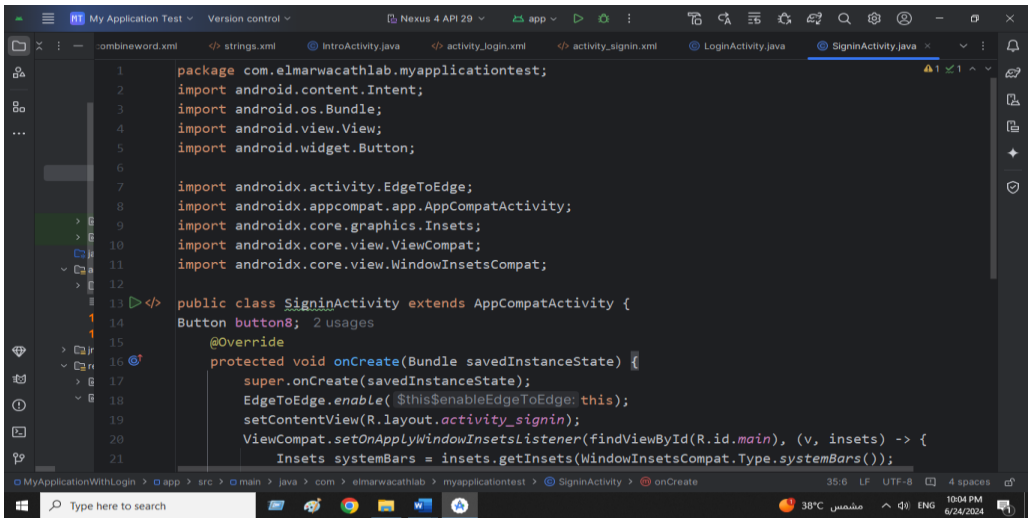
Login Screen. It initializes the login screen of the application, configures edge-to-edge display, applies window insets to adjust padding for system bars, and sets up click listeners on two buttons to navigate to Definition Screen and Sign up Screen.



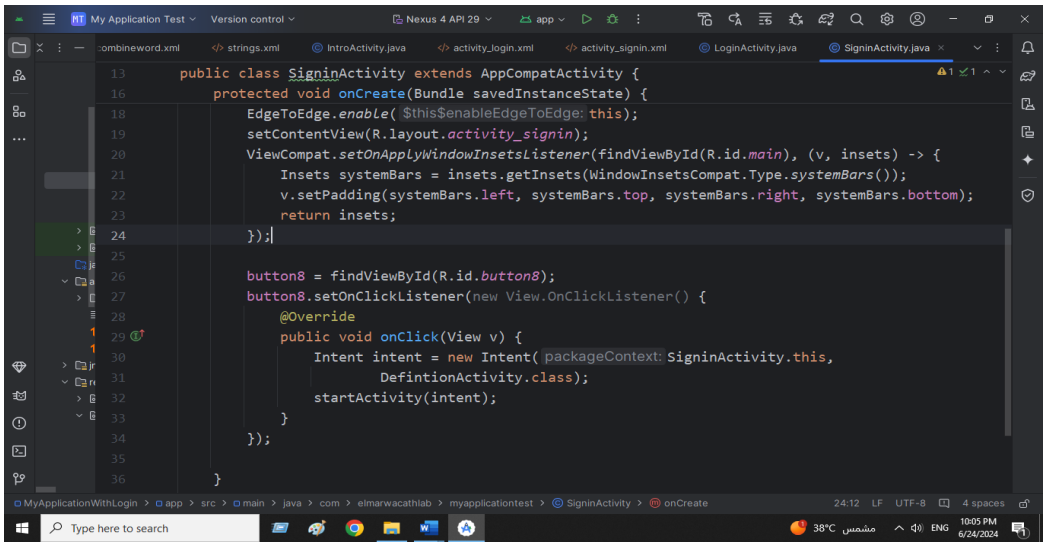
4.5.1.3 screen three:



Sign up Screen.: It initializes the sign-up screen of the application, configures edge-to-edge display, applies window insets to adjust padding for system bars, and sets up a click listener on a button to navigate to Definition Screen.

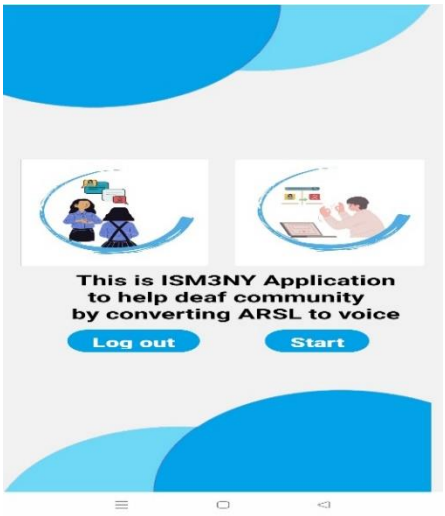


```java
package com.elmarwacathlab.myapplicationtest;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

import androidx.activity.EdgeToEdge;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.graphics.Insets;
import androidx.core.view.ViewCompat;
import androidx.core.view.WindowInsetsCompat;

public class SigninActivity extends AppCompatActivity {
Button button8;  2 usages
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable( $this$enableEdgeToEdge: this);
        setContentView(R.layout.activity_signin);
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {
            Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
```

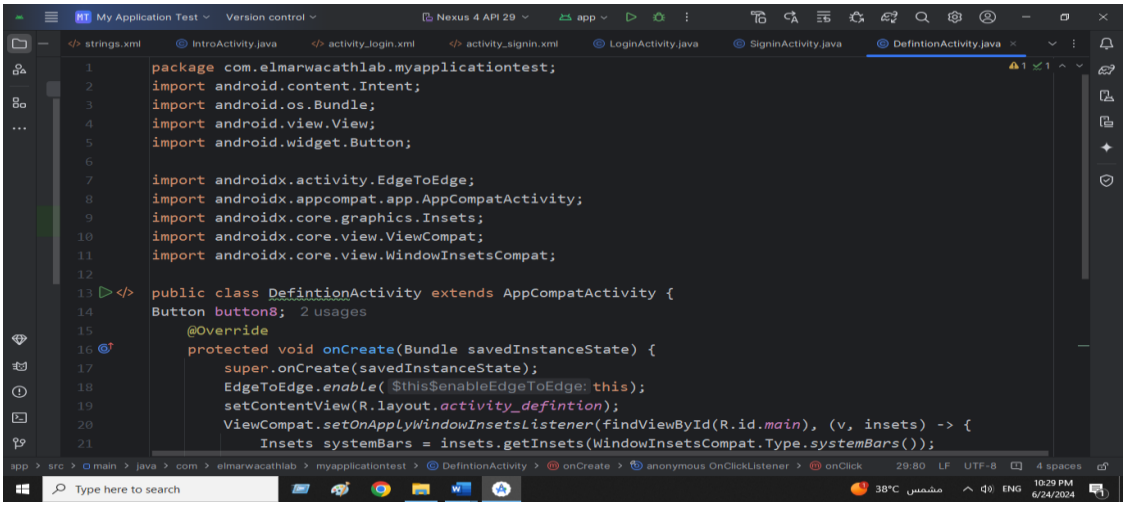


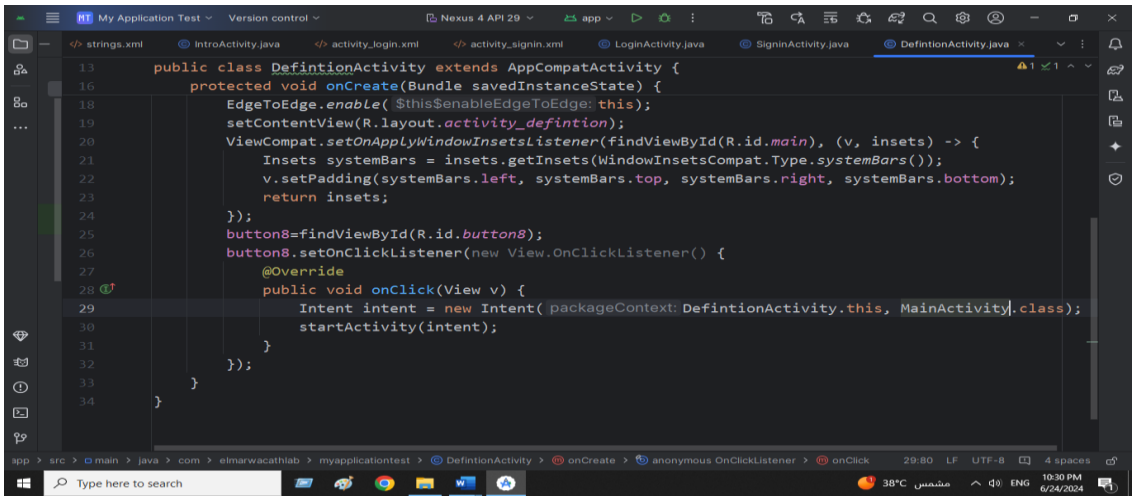
```java
public class SigninActivity extends AppCompatActivity {
    protected void onCreate(Bundle savedInstanceState) {
        EdgeToEdge.enable( $this$enableEdgeToEdge: this);
        setContentView(R.layout.activity_signin);
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {
            Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
            v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
            return insets;
        });

        button8 = findViewById(R.id.button8);
        button8.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent( packageContext: SigninActivity.this,
                        DefintionActivity.class);
                startActivity(intent);
            }
        });

    }
```

4.5.1.4 screen four:

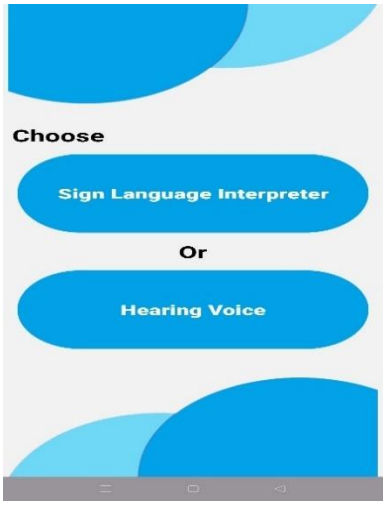


Definition Screen: It initializes the definition screen of the application, configures edge-to-edge display, applies window insets to adjust padding for system bars, and sets up a click listener on a button to navigate to Start application and other button to log out from application.

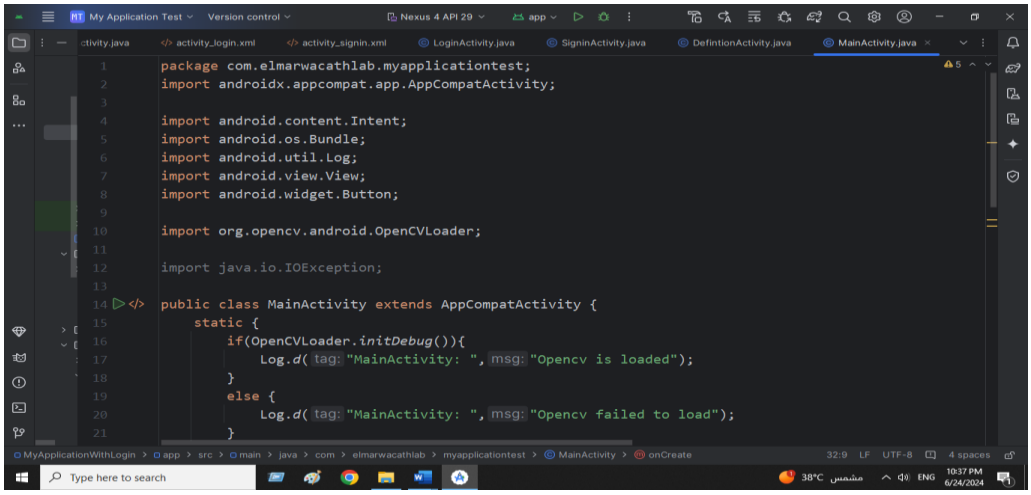




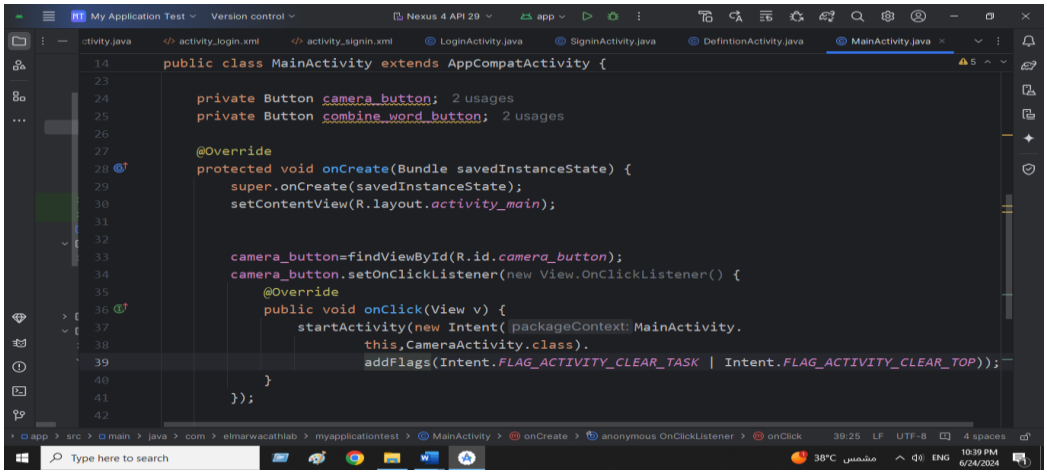
4.5.1.5 *screen five:*



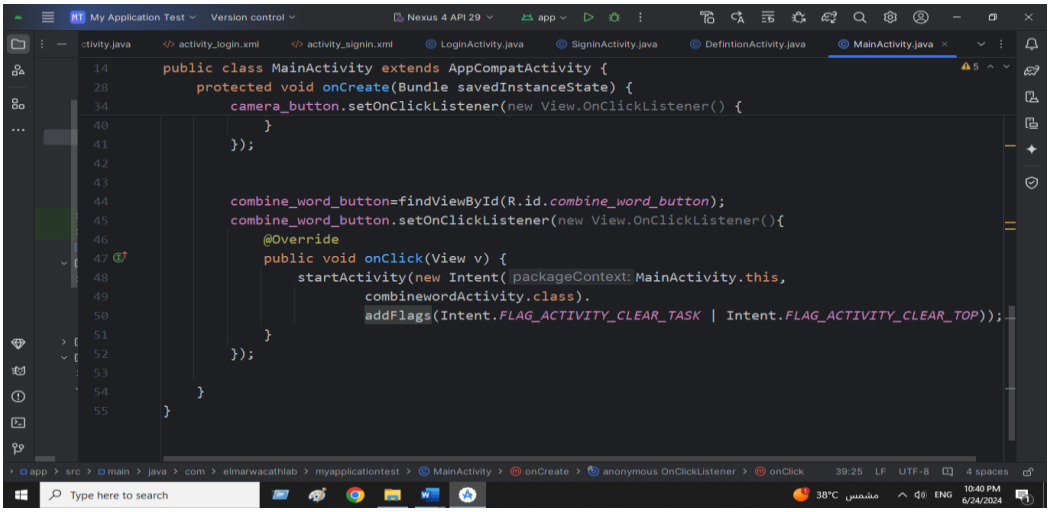
Start application screen: It initializes OpenCV, sets up the user interface with two buttons, one of them it's sign language interpreter used for only detection, the other it's hearing voice used for combine word and sentence and give us the voice , and defines their actions to navigate to different screen (Camera screen and combine word screen).



```java
package com.elmarwacathlab.myapplicationtest;
import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;

import org.opencv.android.OpenCVLoader;

import java.io.IOException;

public class MainActivity extends AppCompatActivity {
    static {
        if(OpenCVLoader.initDebug()){
            Log.d( tag: "MainActivity: ", msg: "Opencv is loaded");
        }
        else {
            Log.d( tag: "MainActivity: ", msg: "Opencv failed to load");
        }
```



```java
public class MainActivity extends AppCompatActivity {

    private Button camera_button;  2 usages
    private Button combine_word_button;  2 usages

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);


        camera_button=findViewById(R.id.camera_button);
        camera_button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                startActivity(new Intent( packageContext: MainActivity.
                        this,CameraActivity.class).
                        addFlags(Intent.FLAG_ACTIVITY_CLEAR_TASK | Intent.FLAG_ACTIVITY_CLEAR_TOP));
            }
        });
```

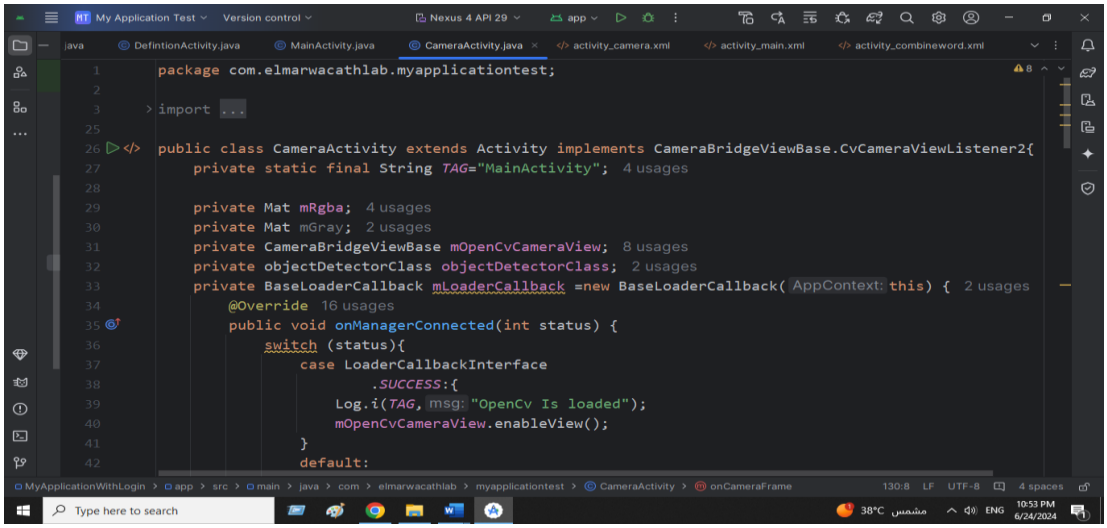


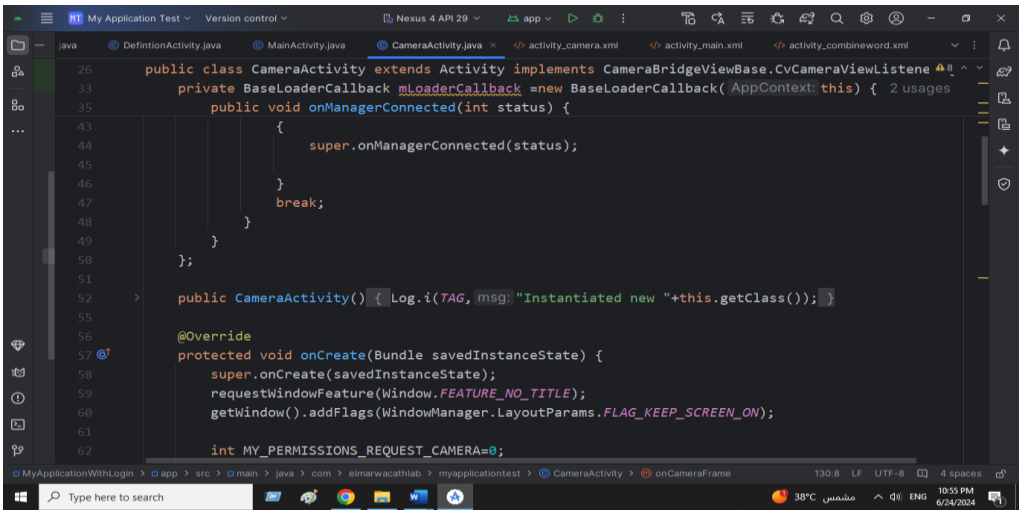
```java
public class MainActivity extends AppCompatActivity {
    protected void onCreate(Bundle savedInstanceState) {
        camera_button.setOnClickListener(new View.OnClickListener() {
            }
        });


        combine_word_button=findViewById(R.id.combine_word_button);
        combine_word_button.setOnClickListener(new View.OnClickListener(){
            @Override
            public void onClick(View v) {
                startActivity(new Intent( packageContext: MainActivity.this,
                        combinewordActivity.class).
                        addFlags(Intent.FLAG_ACTIVITY_CLEAR_TASK | Intent.FLAG_ACTIVITY_CLEAR_TOP));
            }
        });

    }
}
```

4.5.1.6 screen six

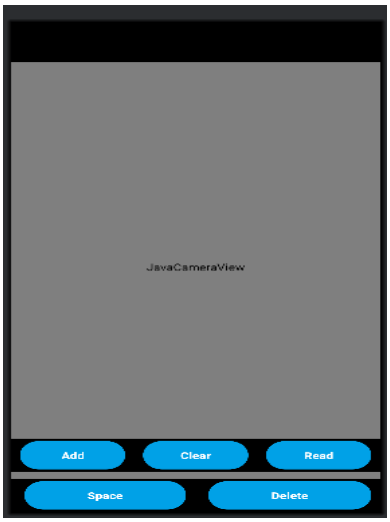
: 

Camera screen: that integrates OpenCV for real-time camera processing. It sets up the camera view, handles permissions, initializes the OpenCV library, and processes camera frames using a TensorFlow Lite model for object detection.



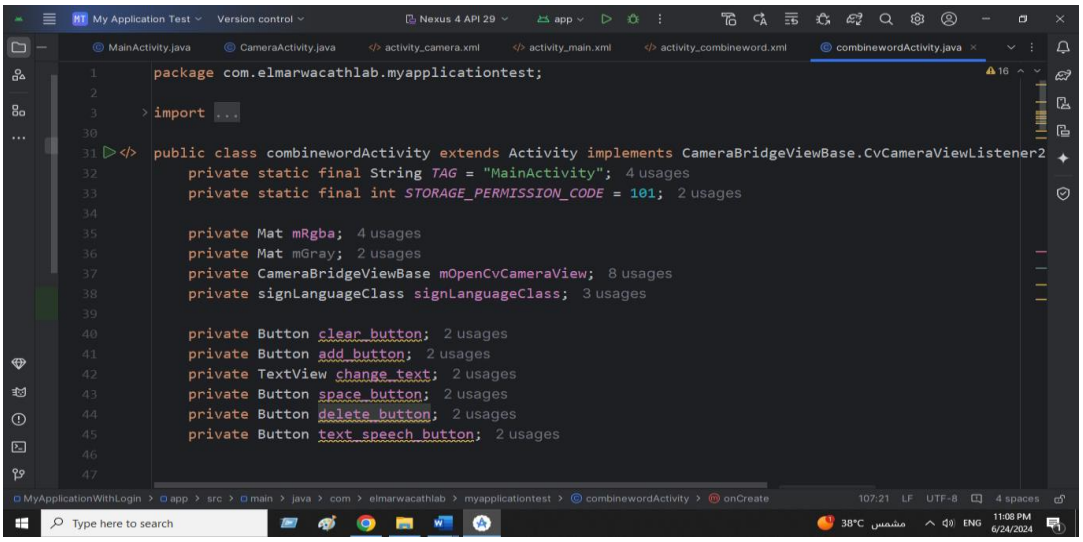


4.5.1.7 screen seven

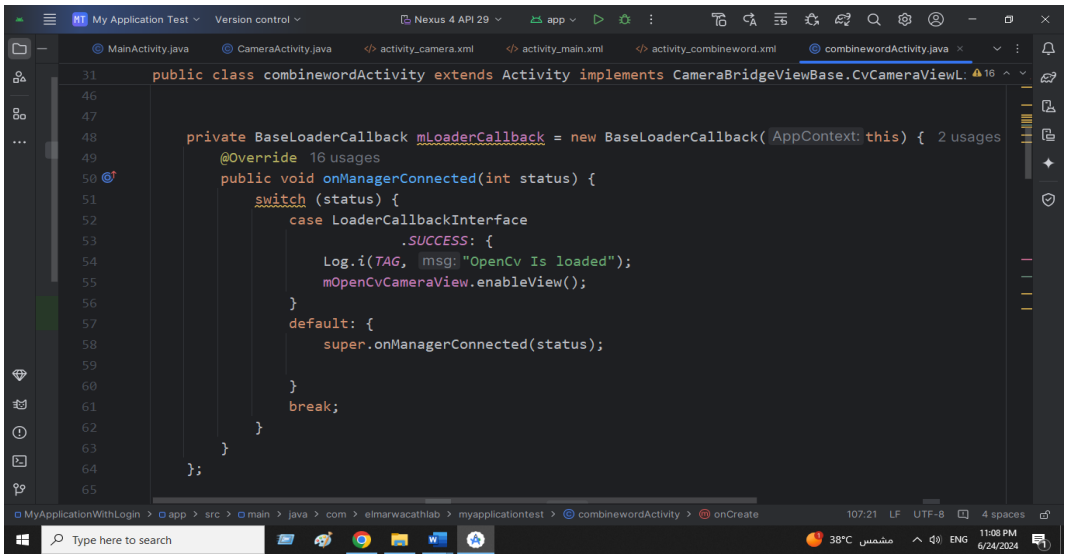


Combine word Screen, explaining its role in integrating OpenCV camera functionality with TensorFlow Lite for sign language gesture recognition. Adjustments and additional details such as create the word and sentence by buttons:

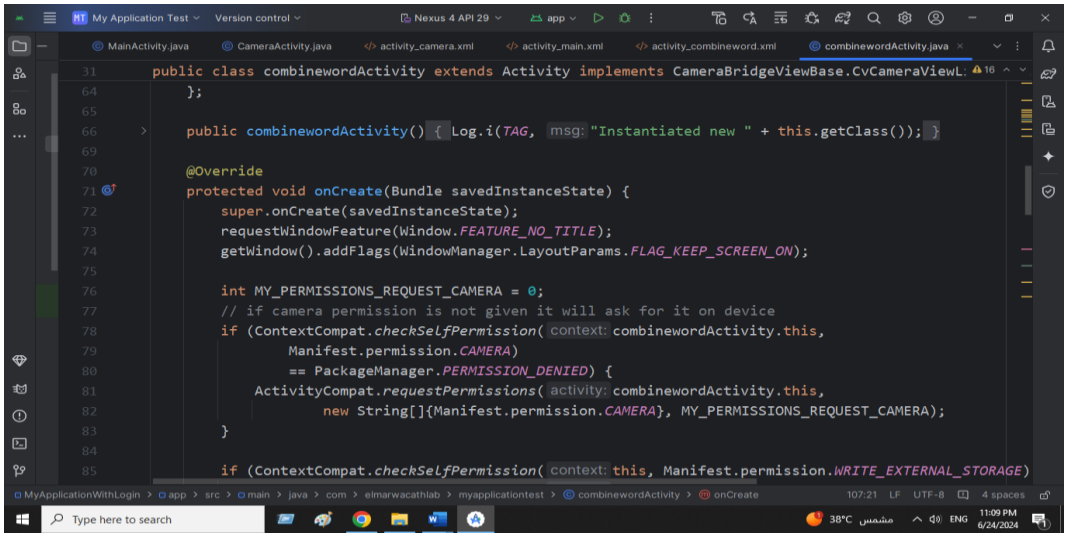
- Add: add letter
- Delete: delete letter
- Clear: delete sentence
- Read: from text to speech
- Spece : to separate between letter or word



```
package com.elmarwacathlab.myapplicationtest;

import ...

public class combinewordActivity extends Activity implements CameraBridgeViewBase.CvCameraViewListener2
    private static final String TAG = "MainActivity";  4 usages
    private static final int STORAGE_PERMISSION_CODE = 101;  2 usages

    private Mat mRgba;  4 usages
    private Mat mGray;  2 usages
    private CameraBridgeViewBase mOpenCvCameraView;  8 usages
    private signLanguageClass signLanguageClass;  3 usages

    private Button clear_button;  2 usages
    private Button add_button;  2 usages
    private TextView change_text;  2 usages
    private Button space_button;  2 usages
    private Button delete_button;  2 usages
    private Button text_speech_button;  2 usages
```



```
public class combinewordActivity extends Activity implements CameraBridgeViewBase.CvCameraViewL:

    private BaseLoaderCallback mLoaderCallback = new BaseLoaderCallback( AppContext: this) {  2 usages
        @Override  16 usages
        public void onManagerConnected(int status) {
            switch (status) {
                case LoaderCallbackInterface
                        .SUCCESS: {
                    Log.i(TAG,  msg: "OpenCv Is loaded");
                    mOpenCvCameraView.enableView();
                }
                default: {
                    super.onManagerConnected(status);

                }
                break;
            }
        }
    };
```

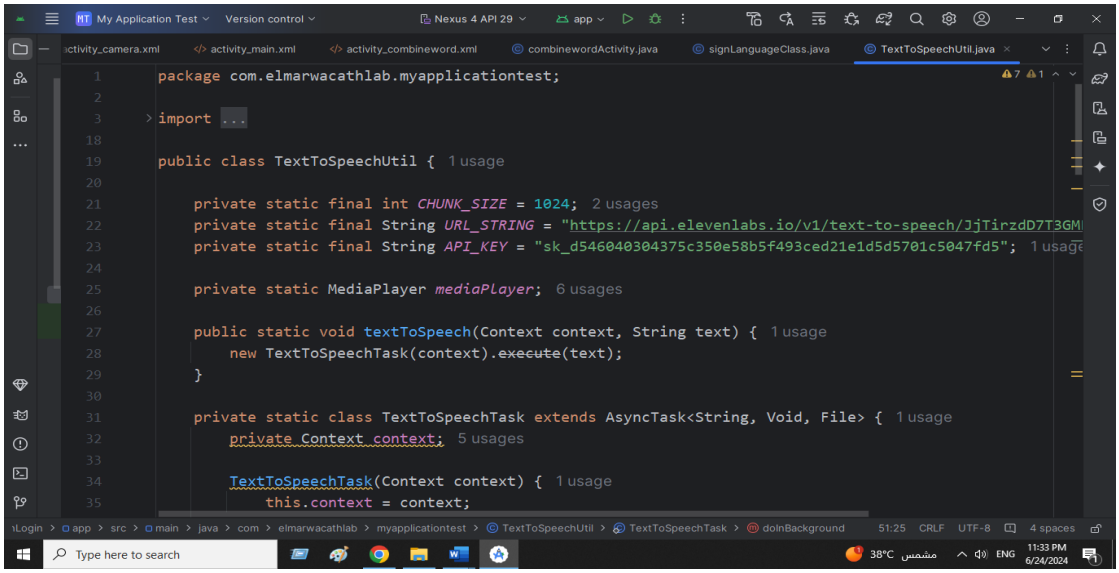


```
public class combinewordActivity extends Activity implements CameraBridgeViewBase.CvCameraViewL:
    };

    public combinewordActivity() { Log.i(TAG,  msg: "Instantiated new " + this.getClass()); }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);

        int MY_PERMISSIONS_REQUEST_CAMERA = 0;
        // if camera permission is not given it will ask for it on device
        if (ContextCompat.checkSelfPermission( context: combinewordActivity.this,
                Manifest.permission.CAMERA)
                == PackageManager.PERMISSION_DENIED) {
            ActivityCompat.requestPermissions( activity: combinewordActivity.this,
                    new String[]{Manifest.permission.CAMERA}, MY_PERMISSIONS_REQUEST_CAMERA);
        }

        if (ContextCompat.checkSelfPermission( context: this, Manifest.permission.WRITE_EXTERNAL_STORAGE)
```

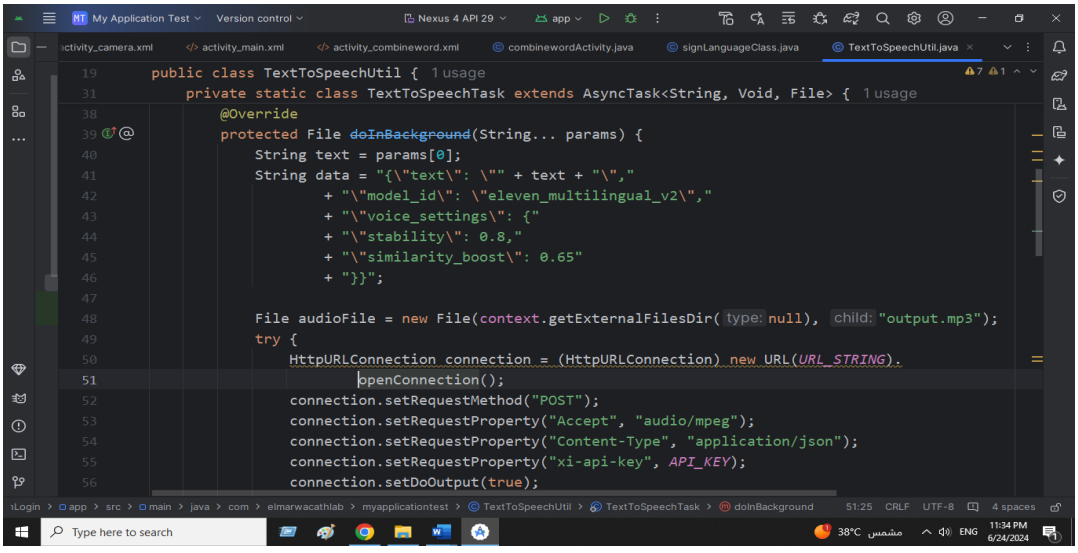
4.5.2 Classes and functionality:

4.5.2.1 Text to speech*:*

The TextToSpeechUtil class provides functionality to convert text to speech using a remote API endpoint. It asynchronously downloads the generated audio file and plays it using Android's MediaPlayer.



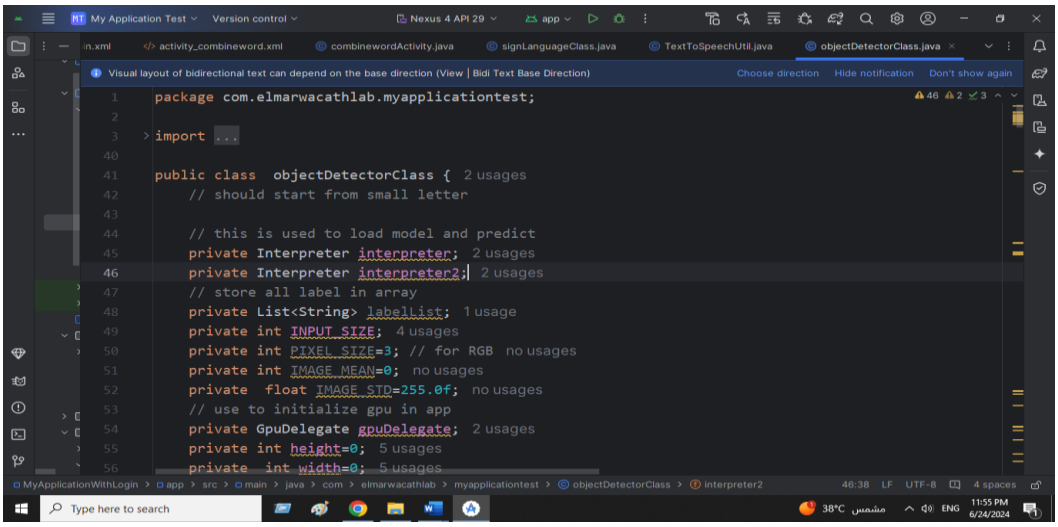
```
package com.elmarwacathlab.myapplicationtest;

import ...

public class TextToSpeechUtil {  1 usage

    private static final int CHUNK_SIZE = 1024;  2 usages
    private static final String URL_STRING = "https://api.elevenlabs.io/v1/text-to-speech/JjTirzdD7T3GM
    private static final String API_KEY = "sk_d546040304375c350e58b5f493ced21e1d5d5701c5047fd5";  1 usage

    private static MediaPlayer mediaPlayer;  6 usages

    public static void textToSpeech(Context context, String text) {  1 usage
        new TextToSpeechTask(context).execute(text);
    }

    private static class TextToSpeechTask extends AsyncTask<String, Void, File> {  1 usage
        private Context context;  5 usages

        TextToSpeechTask(Context context) {  1 usage
            this.context = context;
```



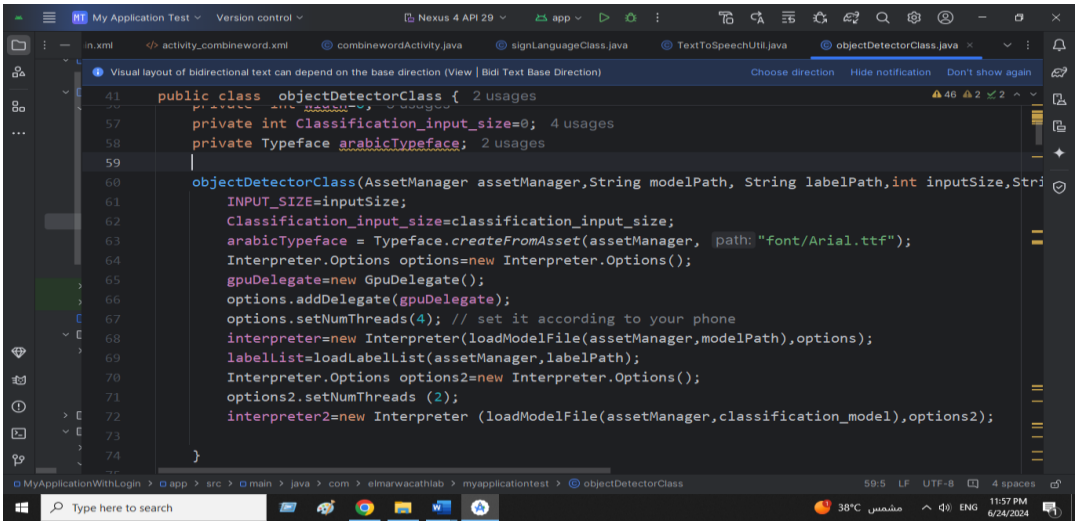
```
public class TextToSpeechUtil {  1 usage
    private static class TextToSpeechTask extends AsyncTask<String, Void, File> {  1 usage
        @Override
        protected File doInBackground(String... params) {
            String text = params[0];
            String data = "{\"text\": \"" + text + "\","
                    + "\"model_id\": \"eleven_multilingual_v2\","
                    + "\"voice_settings\": {"
                    + "\"stability\": 0.8,"
                    + "\"similarity_boost\": 0.65"
                    + "}}";

            File audioFile = new File(context.getExternalFilesDir( type: null), child: "output.mp3");
            try {
                HttpURLConnection connection = (HttpURLConnection) new URL(URL_STRING).
                        openConnection();
                connection.setRequestMethod("POST");
                connection.setRequestProperty("Accept", "audio/mpeg");
                connection.setRequestProperty("Content-Type", "application/json");
                connection.setRequestProperty("xi-api-key", API_KEY);
                connection.setDoOutput(true);
```

4.5.2.2 object detection

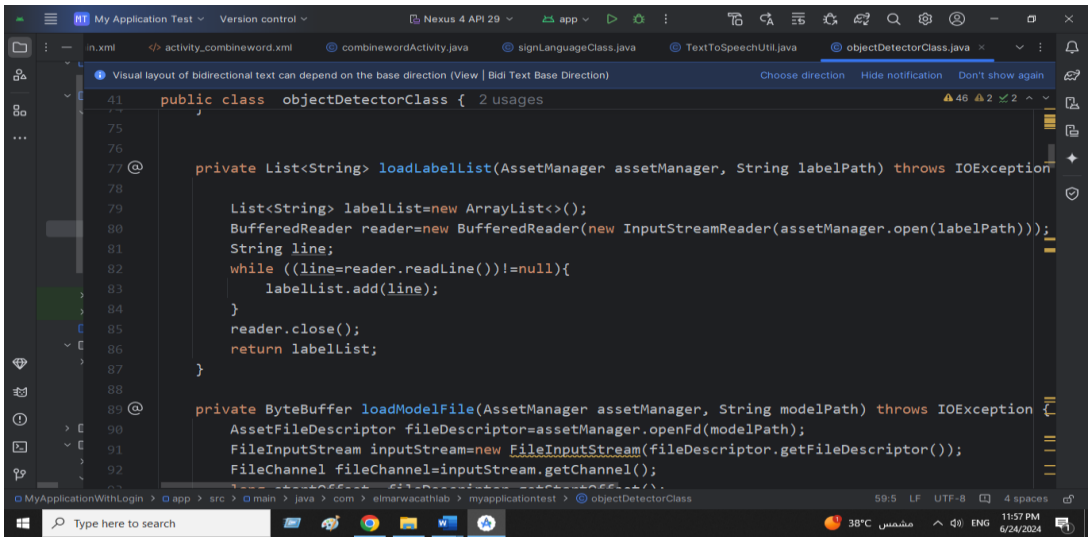
objectDetectorClass integrates TensorFlow Lite models for object detection and classification into an Android application using OpenCV for image manipulation. It demonstrates how to load models, perform inference, process results, and annotate images with detected objects and their classifications. This setup is tailored for applications requiring real-time object detection and classification on Android devices.



```
package com.elmarwacathlab.myapplicationtest;

import ...

public class  objectDetectorClass {  2 usages
    // should start from small letter

    // this is used to load model and predict
    private Interpreter interpreter;  2 usages
    private Interpreter interpreter2;  2 usages
    // store all label in array
    private List<String> labelList;  1 usage
    private int INPUT_SIZE;  4 usages
    private int PIXEL_SIZE=3; // for RGB  no usages
    private int IMAGE_MEAN=0;  no usages
    private  float IMAGE_STD=255.0f;  no usages
    // use to initialize gpu in app
    private GpuDelegate gpuDelegate;  2 usages
    private int height=0;  5 usages
    private  int width=0;  5 usages
```

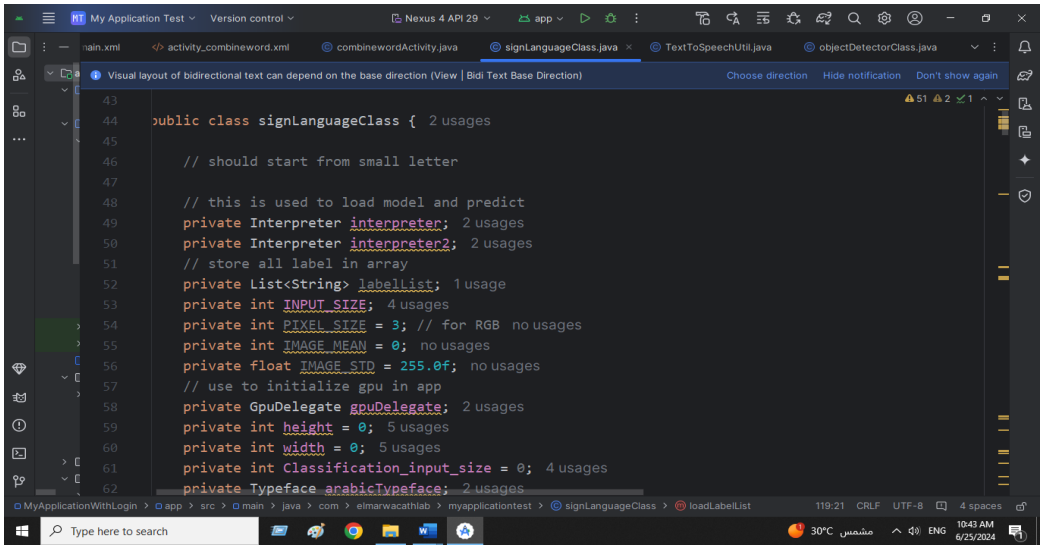


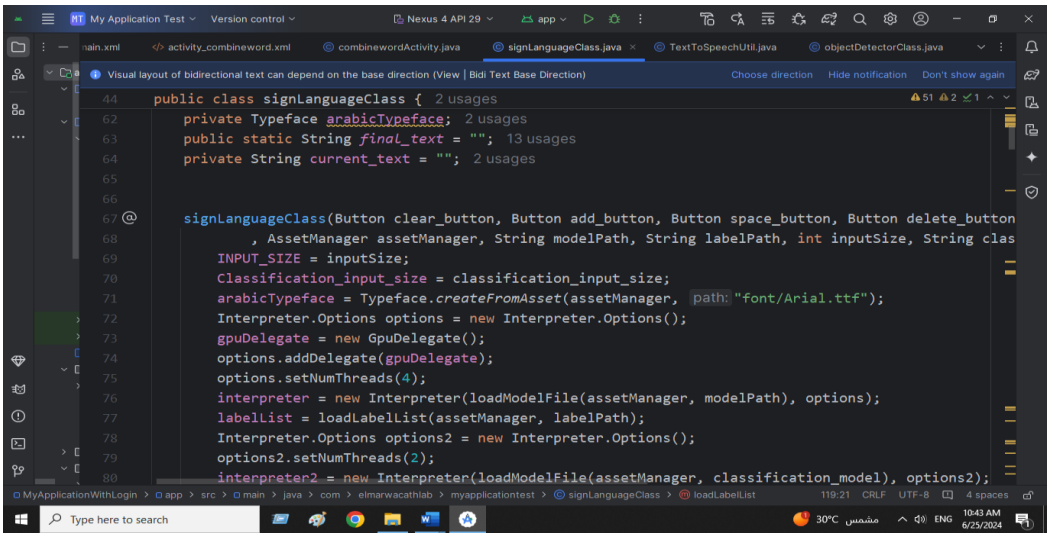
```
public class  objectDetectorClass {  2 usages
    private int Classification_input_size=0;  4 usages
    private Typeface arabicTypeface;  2 usages

    objectDetectorClass(AssetManager assetManager,String modelPath, String labelPath,int inputSize,Stri
        INPUT_SIZE=inputSize;
        Classification_input_size=classification_input_size;
        arabicTypeface = Typeface.createFromAsset(assetManager,  path: "font/Arial.ttf");
        Interpreter.Options options=new Interpreter.Options();
        gpuDelegate=new GpuDelegate();
        options.addDelegate(gpuDelegate);
        options.setNumThreads(4); // set it according to your phone
        interpreter=new Interpreter(loadModelFile(assetManager,modelPath),options);
        labelList=loadLabelList(assetManager,labelPath);
        Interpreter.Options options2=new Interpreter.Options();
        options2.setNumThreads (2);
        interpreter2=new Interpreter (loadModelFile(assetManager,classification_model),options2);

    }
```

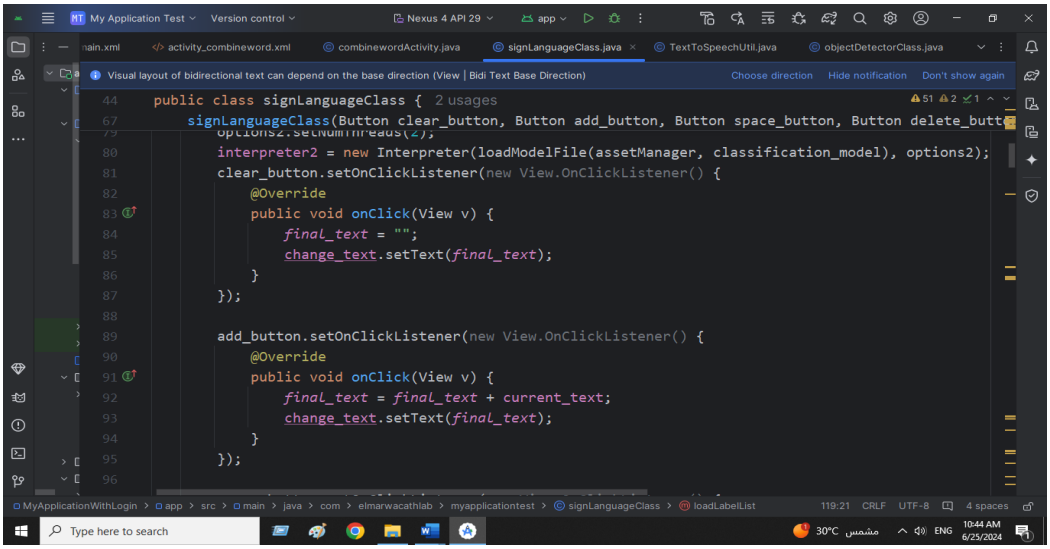


4.5.2.3 *Sign language:*

The signLanguageClass encapsulates functionality for recognizing sign language gestures using TensorFlow Lite models integrated with OpenCV for Android. It handles loading models, processing images, predicting gestures, and rendering results on screen.



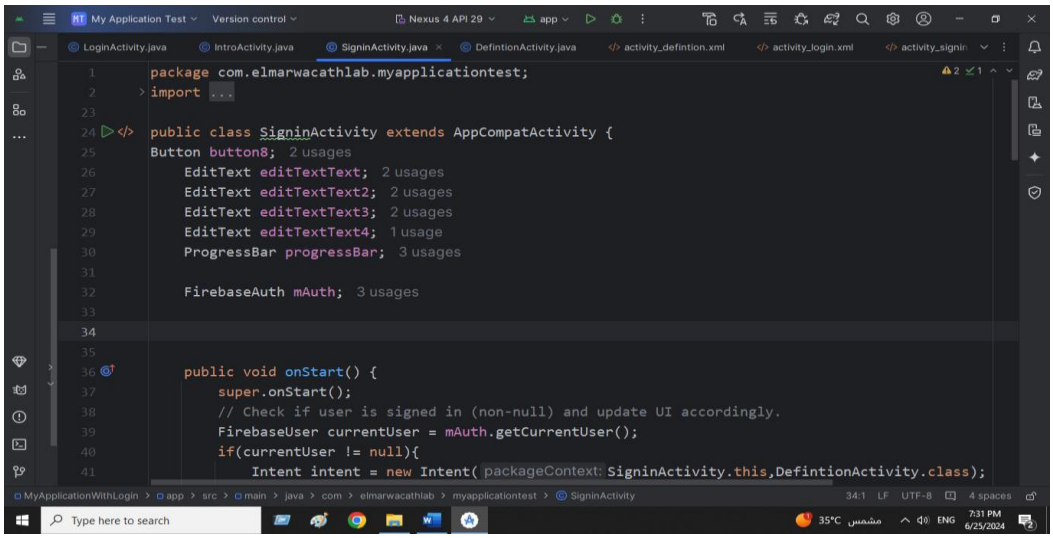

My Application Test
Version control
Nexus 4 API 29
app
activity_combineword.xml
combinewordActivity.java
signLanguageClass.java
TextToSpeechUtil.java
objectDetectorClass.java
Visual layout of bidirectional text can depend on the base direction (View | Bidi Text Base Direction)
Choose direction
Hide notification
Don't show again
public class signLanguageClass { 2 usages
private Typeface arabicTypeface; 2 usages
public static String final_text = ""; 13 usages
private String current_text = ""; 2 usages
signLanguageClass(Button clear_button, Button add_button, Button space_button, Button delete_button
, AssetManager assetManager, String modelPath, String labelPath, int inputSize, String clas
INPUT_SIZE = inputSize;
Classification_input_size = classification_input_size;
arabicTypeface = Typeface.createFromAsset(assetManager, path: "font/Arial.ttf");
Interpreter.Options options = new Interpreter.Options();
gpuDelegate = new GpuDelegate();
options.addDelegate(gpuDelegate);
options.setNumThreads(4);
interpreter = new Interpreter(loadModelFile(assetManager, modelPath), options);
labelList = loadLabelList(assetManager, labelPath);
Interpreter.Options options2 = new Interpreter.Options();
options2.setNumThreads(2);
interpreter2 = new Interpreter(loadModelFile(assetManager, classification_model), options2);
MyApplicationWithLogin
app
src
main
java
com
elmarwacathlab
myapplicationtest
signLanguageClass
loadLabelList
119:21
CRLF
UTF-8
4 spaces
Type here to search
30°C
ENG
10:43 AM
6/25/2024

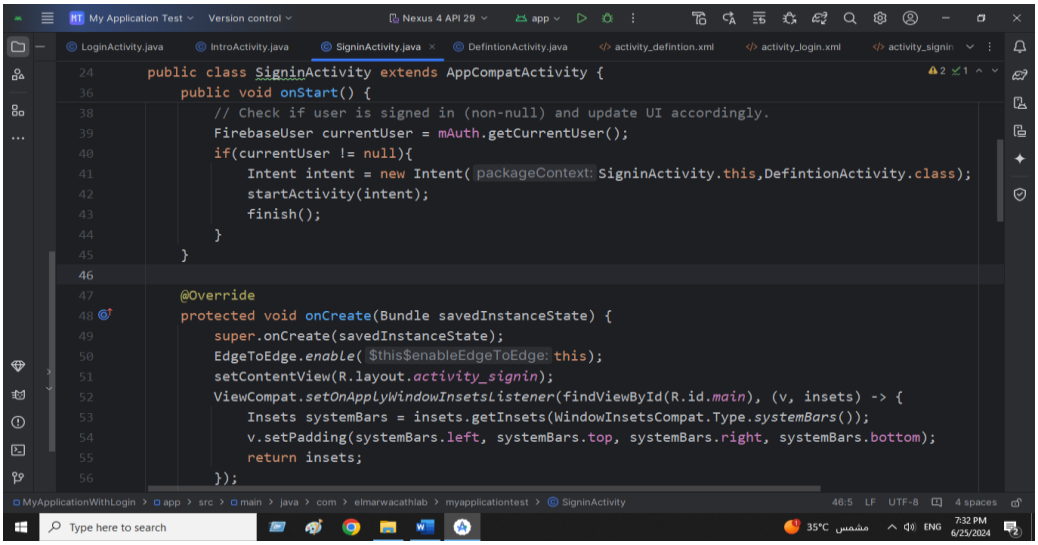

My Application Test
Version control
Nexus 4 API 29
app
activity_combineword.xml
combinewordActivity.java
signLanguageClass.java
TextToSpeechUtil.java
objectDetectorClass.java
Visual layout of bidirectional text can depend on the base direction (View | Bidi Text Base Direction)
Choose direction
Hide notification
Don't show again
public class signLanguageClass { 2 usages
signLanguageClass(Button clear_button, Button add_button, Button space_button, Button delete_butt
interpreter2 = new Interpreter(loadModelFile(assetManager, classification_model), options2);
clear_button.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
final_text = "";
change_text.setText(final_text);
}
});
add_button.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
final_text = final_text + current_text;
change_text.setText(final_text);
}
});
MyApplicationWithLogin
app
src
main
java
com
elmarwacathlab
myapplicationtest
signLanguageClass
loadLabelList
119:21
CRLF
UTF-8
4 spaces
Type here to search
30°C
ENG
10:44 AM
6/25/2024

4.5.3 Use Firebase

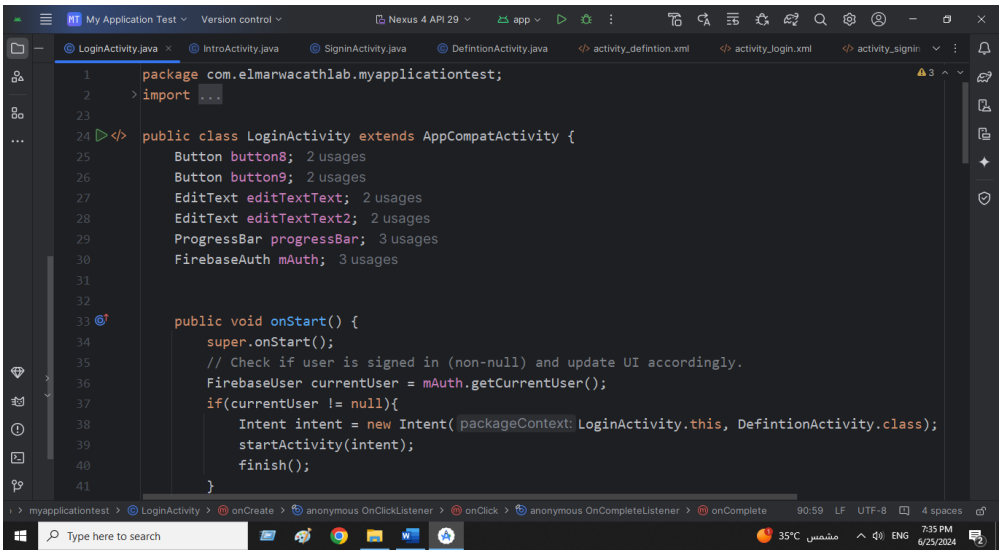
It’s platform by google it’s provide a variety and service . we use in our application Authentication to provides backend service for easy authentication including sign in methods like email and password.

4.5.3.1 To sign up:

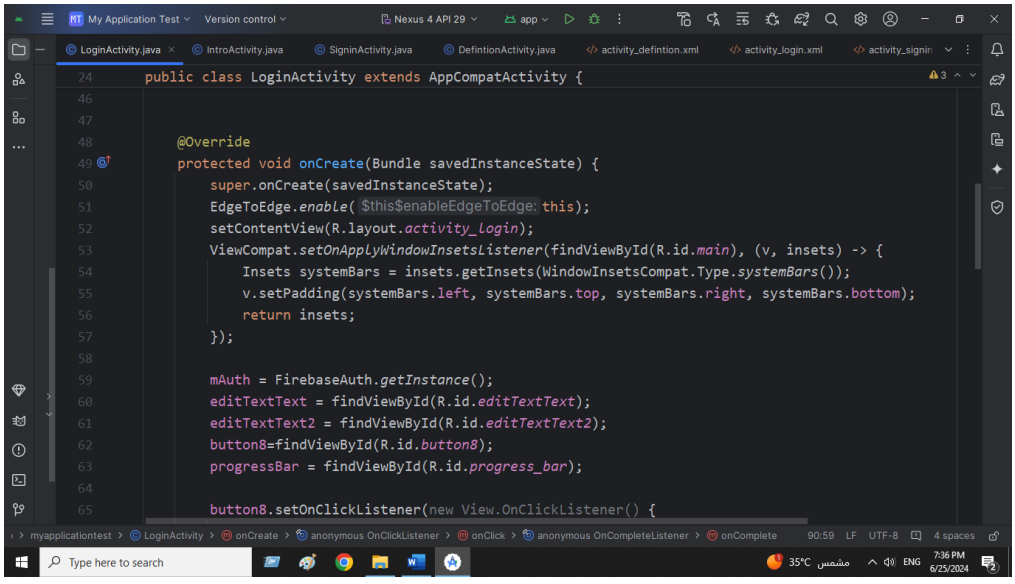




4.5.3.2 To log in:

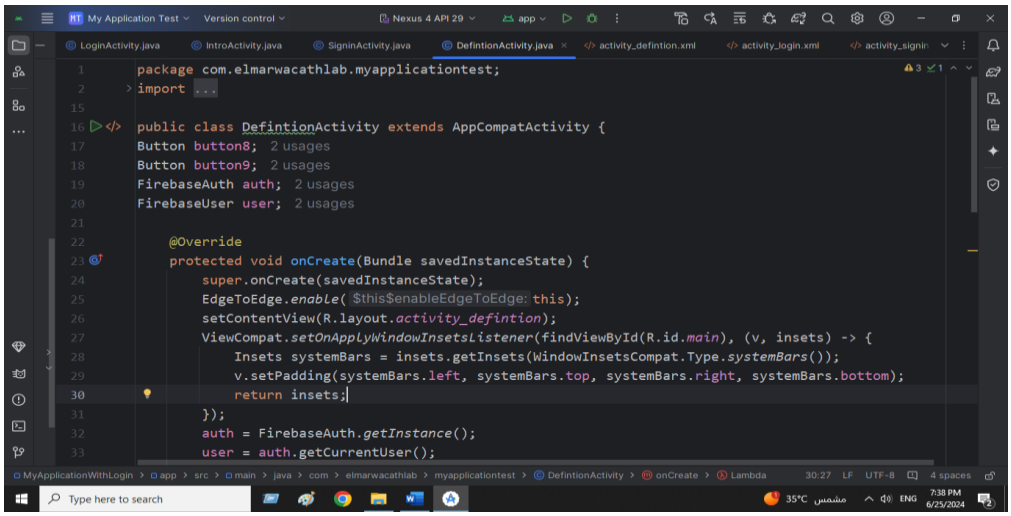


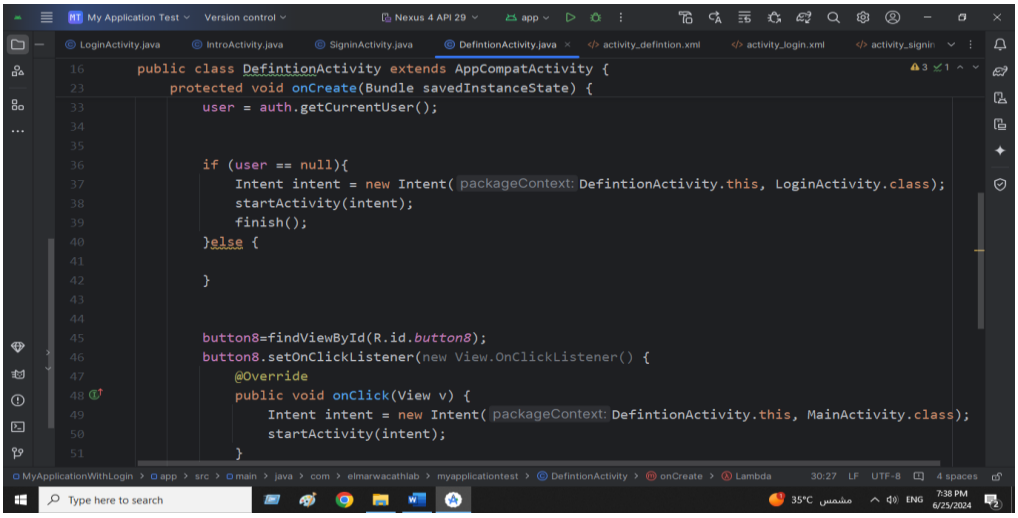
```
package com.elmarwacathlab.myapplicationtest;
import ...

public class LoginActivity extends AppCompatActivity {
    Button button8;  2 usages
    Button button9;  2 usages
    EditText editTextText;  2 usages
    EditText editTextText2;  2 usages
    ProgressBar progressBar;  3 usages
    FirebaseAuth mAuth;  3 usages


    public void onStart() {
        super.onStart();
        // Check if user is signed in (non-null) and update UI accordingly.
        FirebaseUser currentUser = mAuth.getCurrentUser();
        if(currentUser != null){
            Intent intent = new Intent( packageContext: LoginActivity.this, DefintionActivity.class);
            startActivity(intent);
            finish();
        }
```



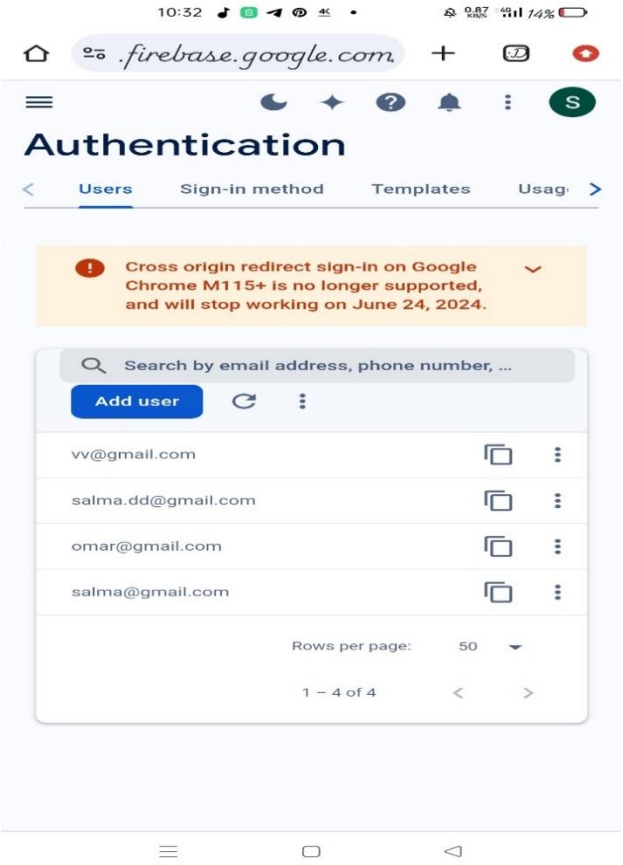
```
public class LoginActivity extends AppCompatActivity {


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable( $this$enableEdgeToEdge: this);
        setContentView(R.layout.activity_login);
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {
            Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
            v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
            return insets;
        });

        mAuth = FirebaseAuth.getInstance();
        editTextText = findViewById(R.id.editTextText);
        editTextText2 = findViewById(R.id.editTextText2);
        button8=findViewById(R.id.button8);
        progressBar = findViewById(R.id.progress_bar);

        button8.setOnClickListener(new View.OnClickListener() {
```

4.5.3.3 To log out:





<<we have cloud contain the all account create in our application



4.6 Experimental Results

In this section, the experimental results of the sign language recognition system are presented and analyzed. The evaluation metrics include a confusion matrix, training and validation curves, as well as precision, recall, and F1-score calculations. Additionally, visual representation of the recognition results is provided below:

Confusion matrix

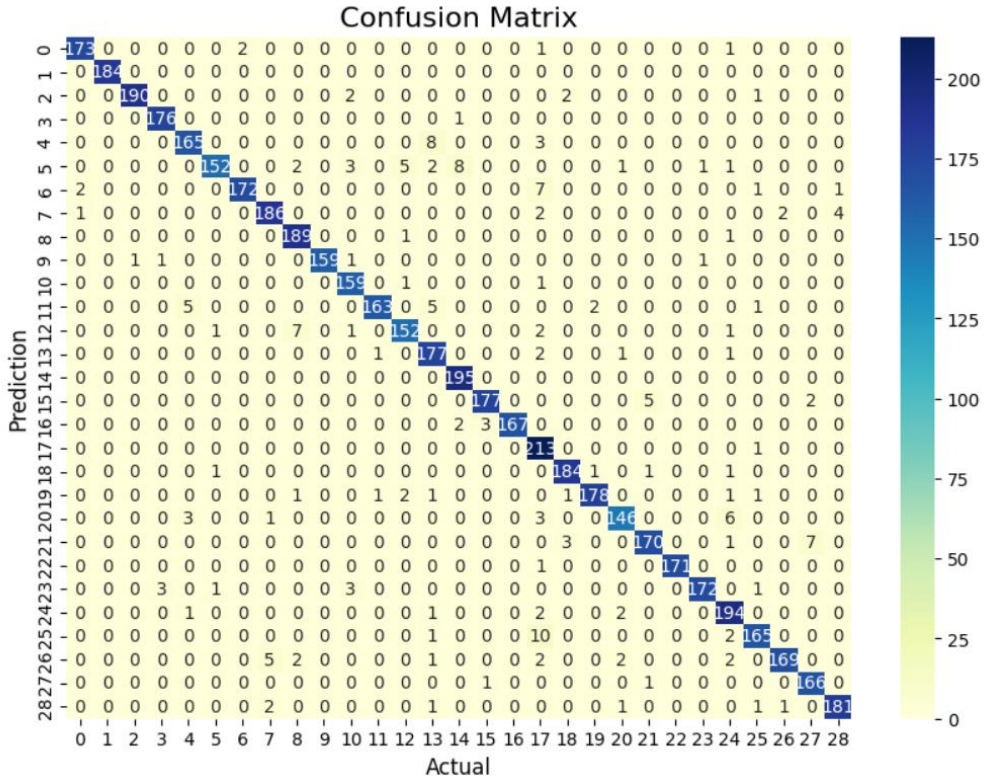


Figure 4-2 :Confusion Matrix

Classification Report

The classification report provides key metrics for evaluating the performance of ArSL recognition model: precision, recall, f1-score, and support for each class.

Precision: The ratio of correctly predicted positive observations to the total predicted positives. High precision indicates a low false positive rate.

Ranges from 0.86 to 1.00, indicating that the model generally predicts well with low false positives.

Recall: The ratio of correctly predicted positive observations to all observations in the actual class. High recall indicates a low false negative rate.

Ranges from 0.87 to 1.00, suggesting the model is effective at capturing the true positives for most classes.

F1-score: The weighted average of precision and recall. It is useful for comparing the balance between precision and recall.

Ranging from 0.92 to 1.00, which shows a consistent performance across classes.

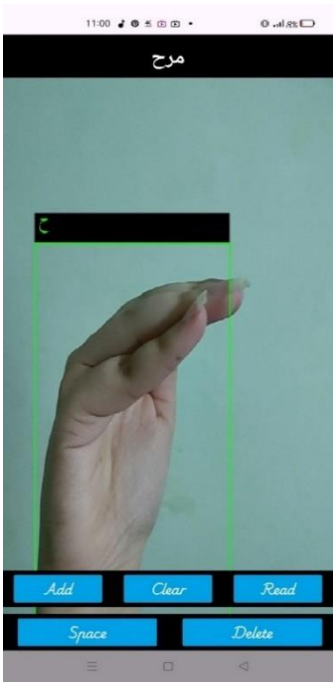
	precision	recall	f1-score	support
0	0.98	0.98	0.98	177
1	1.00	1.00	1.00	184
2	0.99	0.97	0.98	195
3	0.98	0.99	0.99	177
4	0.95	0.94	0.94	176
5	0.98	0.87	0.92	175
6	0.99	0.94	0.96	183
7	0.96	0.95	0.96	195
8	0.94	0.99	0.96	191
9	1.00	0.98	0.99	163
10	0.94	0.99	0.96	161
11	0.99	0.93	0.96	176
12	0.94	0.93	0.94	164
13	0.90	0.97	0.93	182
14	0.95	1.00	0.97	195
15	0.98	0.96	0.97	184
16	1.00	0.97	0.99	172
17	0.86	1.00	0.92	214
18	0.97	0.98	0.97	188
19	0.98	0.96	0.97	186
20	0.95	0.92	0.94	159
21	0.96	0.94	0.95	181
22	1.00	0.99	1.00	172
23	0.99	0.96	0.97	180
24	0.92	0.97	0.94	200
25	0.96	0.93	0.94	178
26	0.98	0.92	0.95	183
27	0.95	0.99	0.97	168
28	0.97	0.97	0.97	187
accuracy			0.96	5246
macro avg	0.96	0.96	0.96	5246
weighted avg	0.96	0.96	0.96	5246

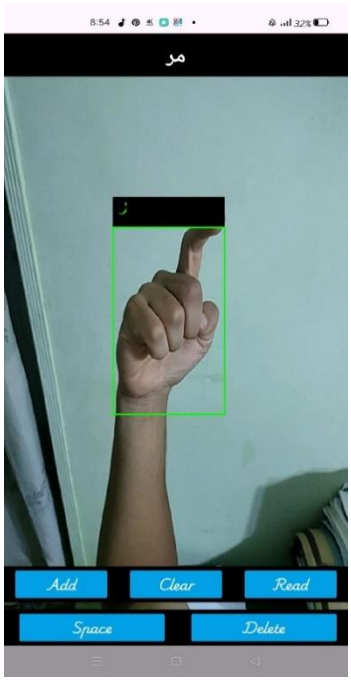
Figure 4-2: Precision, Recall, and F1-Score

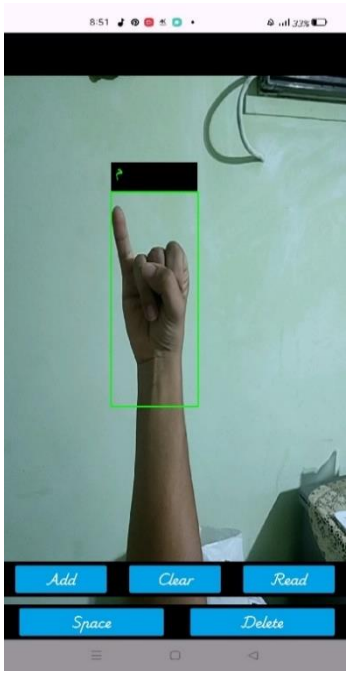
4.7 Sign Language Recognition Results

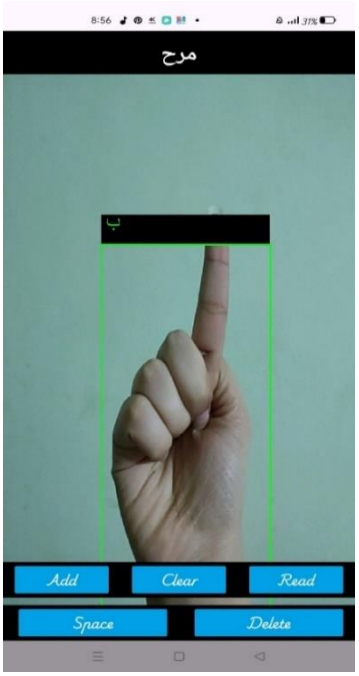
Results**:** We have gesture signs for the letter that we combine together to form word.

Test Results: We test the word <<مرحبا>>









Chapter 5
Conclusion and Future Work

5 Conclusion

5.1 Conclusion

The goal of this project was to develop an effective Arabic Sign Language (ArSL) recognition system using a Convolutional Neural Network (CNN) based on the EfficientNet architecture. We successfully trained and implemented a model capable of recognizing 29 Arabic sign language gestures with high accuracy. The trained model was integrated into an Android application to facilitate real-time gesture recognition.

The use of EfficientNet, a state-of-the-art CNN architecture known for its balance of accuracy and efficiency, significantly contributed to the high performance of our system. The model achieved an impressive accuracy of 95% on the test dataset, demonstrating its capability to generalize well across various gestures and conditions. Extensive data augmentation techniques were applied, which enhanced the model's robustness and ability to handle diverse inputs.

Despite these successes, the project faced challenges such as occasional misclassifications due to similar gestures, variations in hand positioning, and different lighting conditions. Additionally, the dataset size was a limitation, suggesting that a more extensive dataset could further improve model performance.

This project has significant implications for enhancing communication between the deaf community and others. The developed ArSL recognition system offers a practical tool for real-time sign language interpretation, potentially benefiting educational environments, assistive technologies, and inclusive communication platforms.

5.2 Future Work

While this project has made substantial progress, several avenues for future research and development can further enhance the system's performance and applicability:

1. Dataset Expansion:

   Increase the size and diversity of the dataset by incorporating more variations of each gesture, including different lighting conditions, hand orientations, and backgrounds. Collaborating with sign language communities to gather more data can improve the model's robustness.

2. Advanced Data Augmentation:

   Implement more sophisticated data augmentation techniques, such as Generative Adversarial Networks (GANs), to create synthetic data that can further enhance the training dataset and improve model performance.

3. Enhanced Model Architecture:

   Experiment with advanced variations of EfficientNet or other cutting-edge architectures, such as EfficientNetV2 or transformer-based models, to achieve even higher recognition accuracy and efficiency.

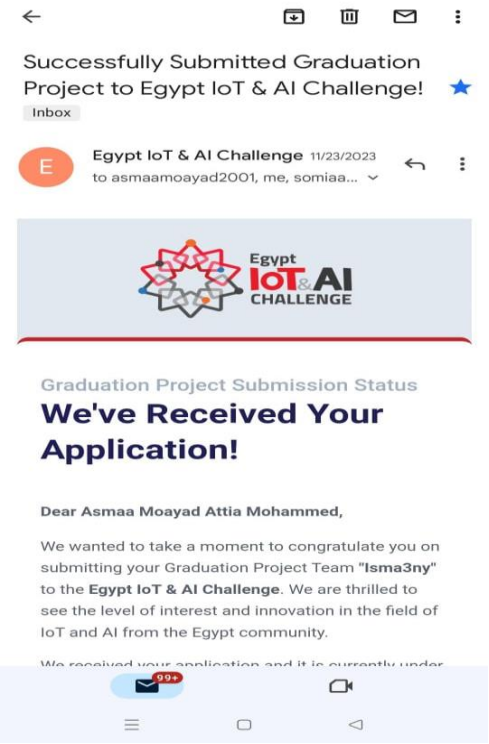
4. User Interface (UI) and User Experience (UX) Improvements:

   Enhancing the UI and UX to make the application more intuitive and user-friendly. This could involve redesigning the layout, improving navigation, and adding features like gesture tutorials and interactive guides.

5. Customizable Settings:

   Allowing users to customize various settings, such as gesture sensitivity, language preferences, and display options, to cater to individual needs and preferences.

Appendix



Successfully Submitted Graduation Project to Egypt IoT & AI Challenge!

Inbox

E Egypt IoT & AI Challenge 11/23/2023
to asmaamoayad2001, me, somiaa...

Egypt IoT&AI CHALLENGE

Graduation Project Submission Status

We've Received Your Application!

Dear Asmaa Moayad Attia Mohammed,

We wanted to take a moment to congratulate you on submitting your Graduation Project Team **"Isma3ny"** to the **Egypt IoT & AI Challenge**. We are thrilled to see the level of interest and innovation in the field of IoT and AI from the Egypt community.

We received your application and it is currently under review by our panel of experts. They will be carefully evaluating each submission based on several criteria, including the potential impact of your solution, the feasibility of your business model, and the strength of your team.

We would like to remind you that the registration for the challenge is still open, and you can refer the application to your network! We will be in touch with you after the review process is completed. We anticipate announcing the accepted startups in the next few weeks.

In the meantime, we encourage you to stay connected with us by following our social media channels and signing up for our newsletter. We will be sharing updates and resources to help you continue to grow and develop your Graduation Project.

Thank you for your interest in the Egypt IoT & AI Challenge, and we wish you the best of luck.

Sincerely,
Egypt IoT & AI Challenge 2023 Organizing Committee

In case you've any inquiries, please contact us at Egypt@ArabIoTAI.org

A regional capacity building and pre-incubation program for innovative ideas in the areas of IoT, AI, and related

Reference

1. https://sist.sathyabama.ac.in/sist_naac/documents/1.3.4/18-b.e-cse-batchno-111.pdf
2. https://www.scribd.com/document/595521693/Software-Requirements-Specification-Sign-Language-to-Text
3. https://www.slideshare.net/slideshow/real-time-conversion-of-sign-language-to-text-and-speech/259552542
4. https://ijrpr.com/uploads/V4ISSUE4/IJRPR12225.pdf
5. **"Jabatan Kebajikan Masyarakat."** https://www.jkm.gov.my/jkm/index.php?r=portal/full&id=ZUFHVTB1NnJWM0EreGtwNC9Vb1hvdz09
    a. **(Accessed Jun. 09, 2023).**
6. https://www.researchgate.net/publication/360384816_Sign_language_Recognition_using_Deep_CNN
7. https://www.atlantis-press.com/article/125986284.pdf
8. https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-ToUnderstanding-
9. https://www.kaggle.com/code/gauthamupadhyaya/image-classification-using-cnn-and-efficientnetb0
10. https://keras.io/examples/vision/image_classification_efficientnet_fine_tuning/
11. https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator

ملخص

الهدف من هذا المشروع هو تطوير نظام فوري لتحويل لغة الإشارة إلى نص وكلام، وتسهيل التواصل بين الأفراد الصم أو ضعاف السمع وأولئك الذين ليسوا على دراية بلغة الإشارة. ويستفيد النظام من رؤية الكمبيوتر للتعرف بدقة على إيماءات لغة الإشارة وتفسيرها، وتحويلها إلى نص مناظر وإنشاء مخرجات كلامية مركبة.

يستخدم النظام نظام إدخال يعتمد على الكاميرا، والذي يلتقط بيانات الفيديو للمستخدم الذي يقوم بإيماءات لغة الإشارة. تتم معالجة إطارات الفيديو باستخدام خوارزميات الرؤية الحاسوبية، باستخدام تقنيات مثل اكتشاف الأشياء وتقدير وضعية اليد للتعرف بدقة على إيماءات اليد. يتم بعد ذلك ربط الإيماءات المعترف بها بقاموس لغة الإشارة المحدد مسبقًا، حيث تتوافق كل إيماءة مع كلمة أو عبارة معينة. يتم تحقيق هذا التعيين باستخدام خوارزميات التعلم الآلي، التي تم تدريبها على مجموعة كبيرة من بيانات إيماءات لغة الإشارة لتحديد الأنماط وتعيين الإيماءات لتمثيلاتها النصية المقابلة. بمجرد تعيين الإيماءات إلى نص، يقوم النظام بإنشاء مخرجات كلام مركبة باستخدام تقنية تحويل النص إلى كلام.

يوفر الكلام المركب تمثيلاً سمعيًا للغة الإشارة، مما يمكّن الأفراد الذين ليسوا على دراية بلغة الإشارة من فهم الرسائل المرسلة والرد عليها. إن تطوير نظام تحويل لغة الإشارة في الوقت الفعلي لديه القدرة على سد فجوة التواصل بين الأفراد الذين يستخدمون لغة الإشارة وأولئك الذين لا يستخدمونها. من خلال توفير تحويل فعال ودقيق للغة الإشارة إلى نص وكلام، يهدف هذا المشروع إلى تمكين الأفراد ذوي الإعاقة السمعية من التواصل بشكل أكثر فعالية وشمولية في مختلف البيئات الاجتماعية والمهنية.

شكر و تقدير

الحمد لله الذي هدانا لهذا وما كنا لنهتدي لولا أن هدانا الله." أولا وقبل كل شيء نود أن نحمد الله على نعمه علينا. ونود أن نعرب عن صادق تمنياتنا الشكر والتقدير لمشرفنا: الدكتور ممتاز الخولي على نصائحه المتخصصة، وحرصه على اكتسابنا المعرفة ومواكبة تطور أساليب التكنولوجيا الحديثة وتوجيهاته طوال هذا المشروع.

نود تقديم الشكر لأعضاء القسم (قسم هندسة النظم و الحاسبات) لمساعدتهم التي لا تقدر بثمن والتي قدموها خلال العملية التعليمية لدينا، ونود أن نعرب عن عميق امتناننا وحبنا الكبير لعائلاتنا الذين جعلونا نستمر، ولم يكن هذا العمل ممكنًا بدون دعمهم النفسي و تشجيعهم لنا، لقد وقفوا إلى جانبنا في أصعب الأوقات.

جامعة الأزهر

كلية الهندسة للبنات

قسم النظم و الحاسبات

معالجة لغة الاشارة بإستخدام الأنظمة الذكية.

" اسمعني "

يقدم هذا المشروع استكمالا جزئيا لمتطلبات درجة البكالوريوس في هندسة النظم و الحاسبات.

مقدم من الطالبات:

أسماء مؤيد عطية محمد	**رقم الجلوس:20248624**
أمنيه حسن أحمد بكير	**رقم الجلوس:20248636**
دعاء محمد القرني	**رقم الجلوس:20248674**
سلمى مجدي سعيد محمد	**رقم الجلوس:20248702**
سلمى وائل عبد المجيد حسن	**رقم الجلوس:20248703**
سمية محمد أمر الله السيد	**رقم الجلوس:20248706**

تحت اشراف :

د. ممتاز سعد الخولي

يوليو، 2024