Tribhuvan University

Faculty of Humanities and Social Sciences

A Final Year Project

On

“SIGN LANGUAGE DETECTION”

Submitted to:

Department of Information Technology

Kathford International College of Engineering and Management
Balkumari, Lalitpur

In partial fulfillment of the requirements for the Bachelor in Computer Application

Submitted by

Bishal Maharjan (6-2-456-213-2019)

Roshan Ghimire (6-2-456-230-2019)

Under the Supervision of

Suwas Karki

September, 2024

ACKNOWLEDGEMENT

We would like to express our deepest appreciation to all those who provided us with the possibility to complete this project. Firstly, we would like to thank our Project Supervisor **Er. Suwas Karki** and our **HOD Mr. Mukesh Prasad Chaudhary** who has coordinated with us and has shared suggestions and ideas related to our project. This project would have been impossible to achieve without their constant and guidance.

Finally, we would like to thank all the members of the IT Department and the “Kathford International College of Engineering and Management” for providing a sound environment to do our project. We would also like to extend our deepest gratitude to all those who have directly and indirectly guided us in this project.

ABSTRACT

The project focuses on the development of a Sign Language Detection System aimed at bridging communication gaps between individuals with hearing impairments and the wider community. The system utilizes advanced image processing and machine learning techniques to accurately recognize and translate sign language gestures into spoken or written language. Key features of the system include a comprehensive sign language database, real-time gesture recognition, personalized learning modules, and user-friendly interfaces. By leveraging cutting-edge technology, the system seeks to enhance communication accessibility, inclusivity, and overall user experience.

Keywords**:** ***sign language, detection system, communication, accessibility, machine learning, user experience, technology***

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATION

- **ANN - Artificial Neural Network**
- **API - Application Programming Interface**
- **ASL - American Sign Language**
- **CNN - Convolutional Neural Network**
- **CSV - Comma-Separated Values**
- **DNN - Deep Neural Network**
- **ReLU - Rectified Linear Unit**
- **tflite - TensorFlow Lite**

Chapter 1: Introduction

1.1 Introduction

American Sign Language (ASL) is the visual-general language used by deaf people in the United States and parts of Canada. There is no universal sign language, different sign languages are used in different countries or regions. No person or committee invented ASL the exact beginning of ASL is not clear, but some suggest that it arose more than 200 years ago from the intermixing of local sign languages and French sign language. The movements of hands and face express ASL It is the primary language of many North Americans who are deaf and hard of hearing, and it is used by many hearing people as well. ASL has a large area of scope. Fingerspelling is part of ASL and is used to spell out English words. In the figure-spelled alphabet, each letter corresponds to a distinct handshape. Fingerspelling is often used to indicate a proper name or an English word for something. ASL provides a set of gesture signs named an American manual alphabet that can be cast-off to spell out many of the English words available. The set of gestures and signs of English words are shown below:

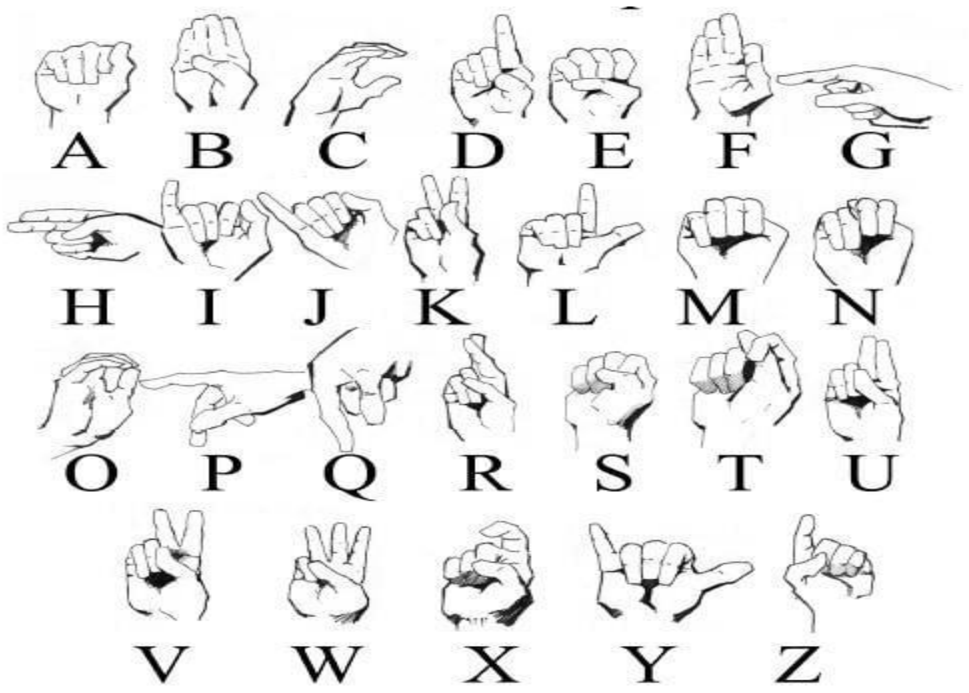


Figure 1: Sign Language Alphabet [1]

1.2 Problem Statement

Speech-impaired individuals used hand signs and gestures to communicate, leading to difficulties for hearing individuals in understanding their language. This highlighted the need for a system that recognized different signs and gestures and conveyed information to those who do not understand sign language. Such a system would bridge the gap between physically challenged individuals and the hearing population.

For instance, in environments like airports, staff members often encountered challenges in understanding the language of speech-impaired individuals during tasks such as verification and checking. Conversely, mute or deaf individuals also faced difficulties comprehending the communication of staff members. While knowledge of sign language could serve as a solution, it is not feasible for everyone involved. Therefore, a medium was required to facilitate communication between these two groups. Many people used hand signs to communicate, emphasizing the necessity of a system capable of converting these signs into a language comprehensible to all.

1.3 Objective

- To develop a desktop application that predicts hand gestures in real time, displays the corresponding words, allows selection from word suggestions, and converts the selected word into speech, enabling seamless and effective communication.

1.4 Scope and Limitation

Here are some Scope of sign language detection:

- **Real-time Gesture Recognition**: The system recognize and translate hand gestures in real-time using a webcam. It focus on the alphabet gestures from ASL, enabling users to spell out words and phrases.

- **Machine Learning Integration**: The project integrates a trained Convolutional Neural Network (CNN) model that accurately identifies hand gestures by extracting features from real-time video input. This model is built to recognize all ASL alphabet gestures with high accuracy.

- **User-Friendly Interface**: A simple and intuitive user interface allow users to interact with the system easily. Users able to gesture recognition, view the results, and listen to the translated gestures using text-to-speech (TTS) features.

Limitation:

Despite the system's innovative approach, certain limitations need to be acknowledged:

- **Detection Accuracy in Complex Environments**: The system's performance may degrade in environments with poor lighting, cluttered backgrounds, or when there are multiple moving objects in the camera's view. These factors can affect the accuracy of gesture recognition.
- **Limited Gesture Dataset**: The system focus on detecting the ASL alphabet, meaning it will not support more complex sentence structures or non-alphabetical gestures. Future expansion to detect a broader range of gestures will require additional datasets and training.
- **Hand Gesture Occlusion**: The model may struggle to accurately detect gestures if parts of the hand are occluded or not fully visible to the camera. This could happen if gestures are performed too quickly or if the user's hands are not correctly positioned.
- **Language Support**: The system is initially designed to support English as the output language. It does not currently offer multi-language support for non-English speakers, although this could be added in future iterations.
- **Learning Curve**: Users unfamiliar with sign language may need time to learn the correct hand gestures, especially in ensuring proper alignment with the camera. Inconsistent or incorrect gestures may lead to incorrect translations.

1.5 Development Methodology

The Sign Language Detection System project utilizes the Agile Development Methodology, which promotes flexibility and iterative progress, making it ideal for evolving projects like this one, where machine learning components require continual refinement based on user feedback.

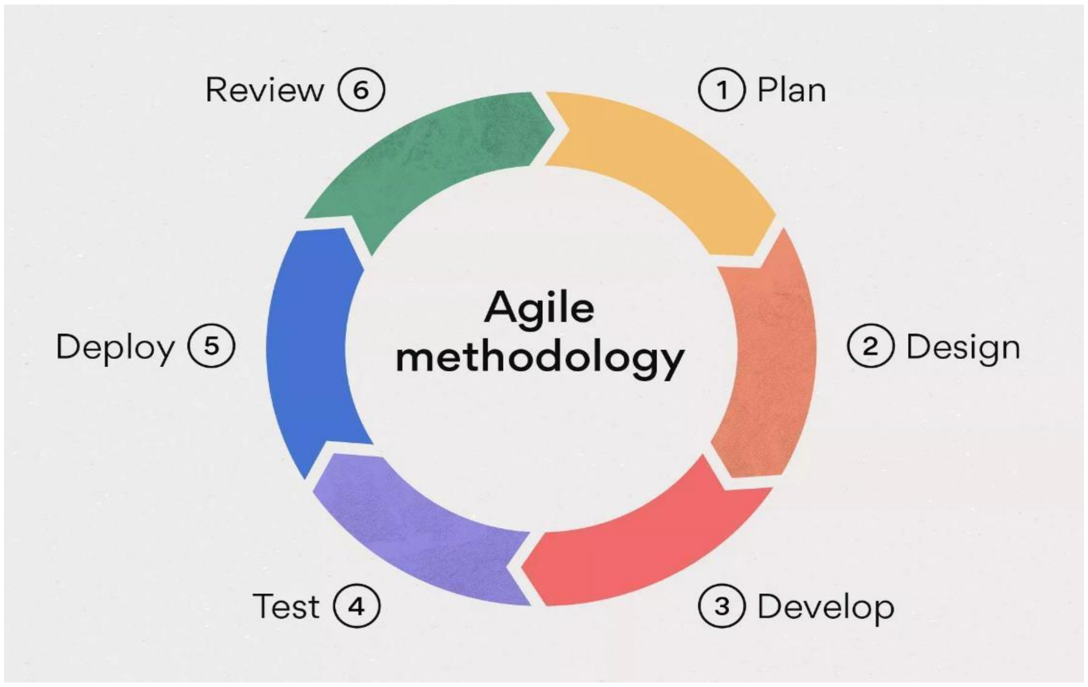


Figure 2: Agile Methodology [2]

Sprint 1: Requirement Analysis and System Architecture Design

- Deliverable: Core functionalities and architecture diagrams.

Sprint 2: Image Capture Module Development

- Deliverable: Webcam integration and image preprocessing.

Sprint 3: CNN Model Training and Integration

- Deliverable: Trained CNN model for gesture recognition.

Sprint 4: User Interface and Feedback Mechanism

- Deliverable: User-friendly interface for interaction and feedback.

Sprint 5: Testing and Refinement

- Deliverable: Fully functional system with real-time gesture detection and text/audio output. Extensive testing ensures performance across various conditions.

1.6 Report Organization

The report of the project is based on the following format:

Chapter 1: Introduction: This is the primary section of the report where the overview, problem statement, objectives, scope and limitation, and methodology have been discussed.

Chapter 2: Background study and literature review: In this section the description of fundamental theories from intellectual personnel and general concepts related to the project has been discussed.

Chapter 3: System analysis: This section describes the general architecture and flow mechanism of the system. It describes the hardware and software needed to build the system and the system's feasibility analysis

Chapter 4: System Design: This section describes the diagrammatic description of the system with a suitable diagram to describe the system-designed use case diagram, class diagram, sequence diagram, and activity diagram.

Chapter 5: Implementation and testing: This section defines the tools used and the implementation details of the project. It also consists of different test cases used to test different units of the project.

Chapter 6: Conclusion and Future Recommendation: This section describes the conclusion of the project and the future plans for the project.

Chapter 2: Background Study and Literature Review

2.1 Background Study

The Sign Language Detection System leverages advanced machine learning and image processing techniques to recognize and interpret American Sign Language (ASL) gestures. The foundation of the project lies in deep learning models, particularly Convolutional Neural Networks (CNNs), which are well-suited for image recognition tasks. Below is a brief overview of the key concepts and technologies used in this project.

2.1.1 Artificial Neural Networks (ANN)

Artificial Neural Networks (ANNs) are computational models inspired by the human brain's neural networks. They consist of interconnected layers of nodes (neurons) where each layer transforms input data into more abstract representations. ANNs are commonly used in pattern recognition tasks but can struggle with high-dimensional data, making them less optimal for complex image recognition tasks like gesture detection.

In the context of this project, ANNs serve as a theoretical foundation but are not directly applied due to limitations in handling spatial hierarchies and visual data. Instead, a more advanced model, CNN, is utilized for enhanced performance.

2.1.2 Deep Learning Model: CNN

Convolutional Neural Networks (CNNs) are specialized deep learning models designed to process visual data. They are highly effective in identifying patterns within images, such as edges, textures, and shapes, making them ideal for gesture recognition.

In this project, CNNs are employed for real-time gesture recognition. The CNN model processes images from the webcam feed, applies filters to detect hand shapes and movements, and classifies these patterns into corresponding ASL gestures. The architecture involves multiple layers, including:

- **Convolutional Layers:** Extract features from the input image.
- **Pooling Layers:** Reduce dimensionality while retaining important features.
- **Fully Connected Layers:** Classify the extracted features into specific gestures.

The CNN model is trained on a dataset of ASL gestures to ensure high accuracy in recognizing signs.

2.1.3 HDF5 (h5) File Format

The HDF5 (h5) file format is a data storage format commonly used in machine learning to save model weights and structures. This format allows for efficient storage and retrieval of complex data, making it suitable for saving trained models in machine learning projects.

In this project, the h5 file format is used to store the trained CNN model. Once the model is trained on the ASL gesture dataset, it is saved as an h5 file, facilitating easy deployment and future use in the system without the need for retraining the model from scratch.

2.1.4 TensorFlow Lite (tflite) Model

TensorFlow Lite (tflite) is a lightweight version of TensorFlow designed for mobile and embedded devices. It allows machine learning models to run efficiently on devices with limited computing power, such as smartphones or Raspberry Pi.

In this project, the tflite format is considered for deploying the trained model on mobile platforms. This enables real-time gesture recognition on devices with limited processing power, making the system more accessible to users on various platforms.

2.2 Literature Review

U. Hari Priya and team [3] developed an American Sign Language (ASL) recognition system using Convolutional Neural Networks (CNNs). The study compared various methods, including logistic regression and CNN, to enhance communication between signers and non-signers. They utilized a custom ASL dataset, capturing hand gestures via a webcam, and included images of the first 18 English alphabets (excluding J) with various gesture variations for model training. Additionally, they used a Kaggle dataset with 72,025 images. Gestures were converted to grayscale images and processed through the CNN model. The CNNs, designed to emulate biological neural connectivity, underwent training with steps including convolution, Rectified Linear Unit (ReLU) activation, pooling, and classification. The model employed cross-entropy loss to measure performance, which increases when predicted probabilities diverge from actual labels. The CNN model demonstrated high accuracy and ease of debugging, facilitating real-time communication with minimal delay.

Kang and his team [4] developed a real-time sign-finger spelling recognition system using Convolutional Neural Networks (CNNs) from depth maps. They collected 31,000 depth maps, with 1,000 images for each class, using the Creative Senz3D camera. The dataset included 31 different hand signs from five individuals, covering all finger-spelling letters and numbers, excluding the J and Z letters requiring additional classification data. They used hand segmentation to create bounding boxes of uniform size (256 by 256), assuming that the hand would be close to the camera. The CNN architecture employed was based on CaffeNet, consisting of three max-pooling layers, three fully connected layers, and a total of five layers. Their model achieved accuracies of 83.58% and 85.49% for fine-tuning and 75.18% and 78.39% for training. This approach relied solely on depth maps captured by the Creative Senz3D camera, which is relatively expensive and inaccessible.

Bagby, Braden, and their team [5] in 2021 simplified sign language detection for smart home devices using Google MediaPipe. They proposed a novel approach to the challenges of capturing hand gestures in American Sign Language (ASL). They created a custom dataset comprising CSV files with hand landmarks for different gestures, derived from an original set of 15,000 images evenly representing all 26 ASL letters using MediaPipe. They reserved 20% of this dataset for testing.

They developed a prototype application that separated the resource-intensive parts of the system from the camera input and command output. Initially, video frames were captured via OpenCV in the client module, with each frame stored as a 2D numpy array of size 86,400 bytes (480x60 pixels with 3 RGB values). Custom application layers were written to transmit these frames to a server. A Python framework built on top of MediaPipe was integrated with hand landmark detection code. The system supported two types of input: single images defined by command line arguments and JPEG image streams sent over TCP.

Chapter 3: System Analysis and Design

3.1 System Analysis

To analyze the performance of a sign language recognition system, several parameters can be measured, such as accuracy, precision, recall, and F1 score. These metrics can be used to evaluate the system's performance on a set of test data.

Another important aspect of system analysis is identifying areas for improvement. In the case of a sign language recognition system, potential areas for improvement may include improving the accuracy of the pattern recognition component, enhancing the input devices to capture more nuanced gestures, or developing better user interfaces to improve the user experience.

3.1.1 Requirement Analysis

Requirement analysis is one of the initial tasks performed in our project. It is the process of precisely identifying, defining, and documenting the various requirements that are related to the Sign Language Detection System. There are two types of requirements:

i. Functional Requirement

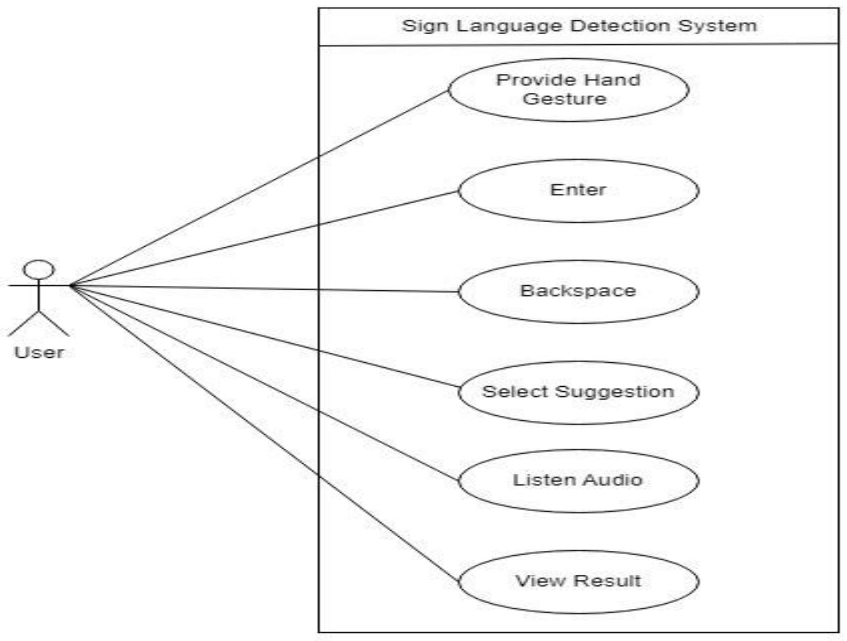


Figure 3: Use Case Diagram of Sign Language Detection System

The above diagram illustrates the functionalities of the Sign Language Detection System and the interactions between different user and the system.

Actors:

1. User: The only actor, who interacted with the system by providing input (hand gestures) and controlling the feedback received from the system.

Use Cases:

1. Provide a hand gesture: The user performed a hand gesture that the system recognized and processed.
2. Entered: The user confirmed the recognized gesture by enter, finalizing the input.
3. Backspaced: The user undid or deleted the last recognized gesture if they made a mistake or wanted to make changes.
4. Selected Suggestion: After processing the gesture, the system offered suggestions. The user selected the correct suggestion when multiple options were available.
5. Listen Audio: The user had the option to listen to the audio output of the translated gesture.
6. View Result: The system displayed the final translated result for the user to review.

ii. Non- Functional Requirement

- Availability**:** The system achieve 99.9% uptime, accounting for scheduled maintenance.

- Real-Time Performance**:** The system recognize signs and translate them to text within 1 second to ensure seamless user interaction.

- Environmental Adaptability: The system provide real-time recognition with a accuracy in low-light conditions (defined as lighting levels below 100 lux).

- Usability: The system support natural interaction, aiming for a user satisfaction score of 80% or higher in usability testing.

3.1.2 Feasibility Analysis

1. Technical feasibility: The technical feasibility of an American Sign Language (ASL) recognition system depended on the availability and effectiveness of the technology needed to develop and operate the system. This included hardware, such as cameras or sensors, and software, such as machine learning algorithms and user interfaces. The system needed to have sufficient processing power and speed to accurately recognize and interpret ASL gestures in real time.
2. Operational Feasibility: The operational feasibility of an ASL recognition system depended on whether the system could function effectively in the intended environment and meet the needs of its users. This included considerations such as the system's accuracy, reliability, and user-friendliness. The system needed to handle variations in gesture form and style and should have integrated with other communication technologies, such as text-to-speech or speech-to-text systems.
3. Economic Feasibility: The economic feasibility of an ASL recognition system depended on the costs of developing and maintaining the system, as well as the potential benefits it offered. This included factors such as the initial development costs, ongoing maintenance and support costs, and any potential revenue streams.
4. Schedule: The project was scheduled for full development in four months. The total project consisted of nine development tasks, subdivided into various phases with an allocated time schedule as per requirements. The project was expected to be completed on time, as outlined in the Gantt chart and other scheduling concepts.

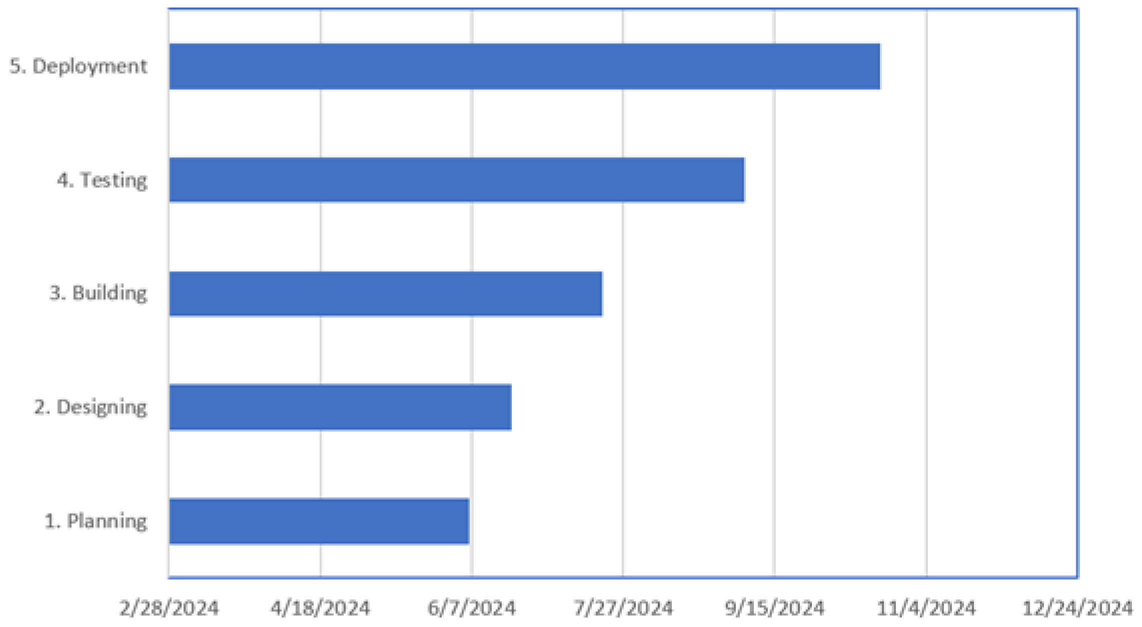


Figure 4: Gantt Chart

3.1.3 Object Modeling Using Class and Object Diagram

Object modeling represents a system's objects, actions, and associated attributes. This involves creating a static model of the system that captures the objects that constitute it and their relationships. The goal of object modeling is to identify the objects within the system, their attributes and behaviors, and the relationships among them. Object and class diagrams are utilized for the object modeling of the system.

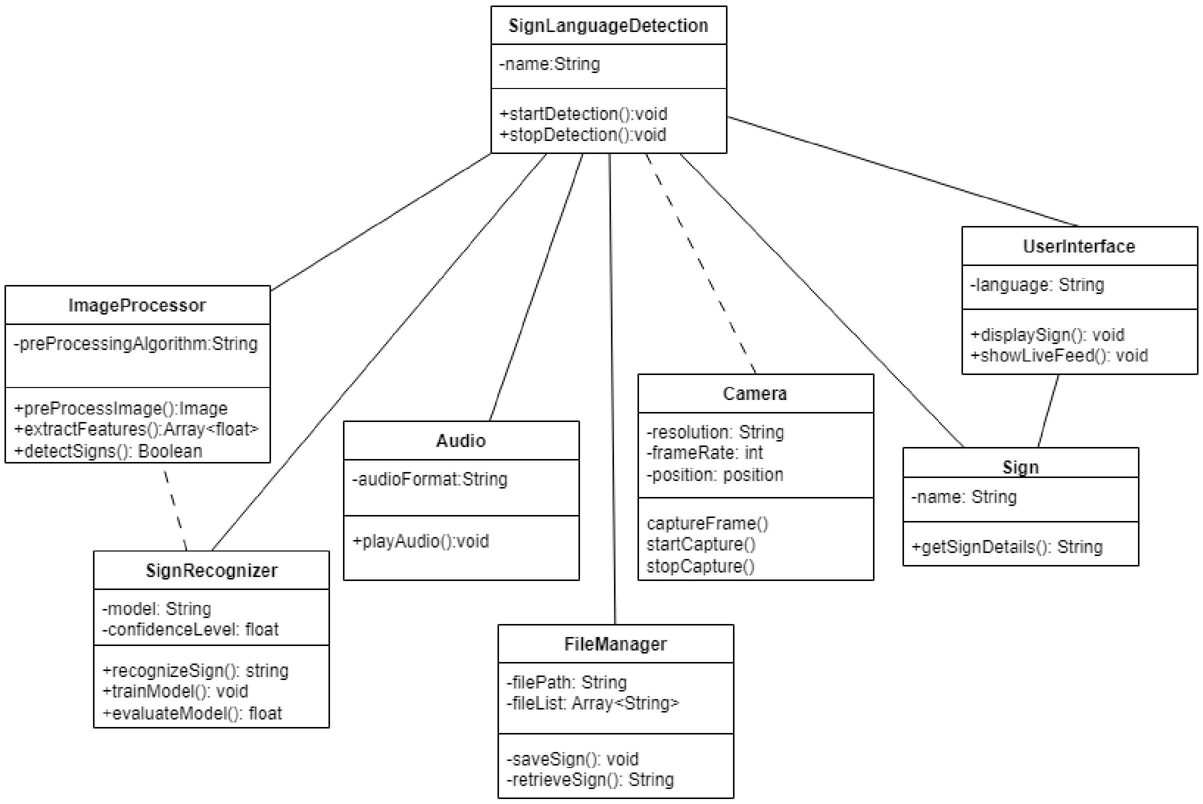


Figure 5: Class Diagram of Sign Language Detection System

1. SignLanguageDetection

- **Attributes:**
  - name: Likely the name of the system or detection instance.
- **Methods:**
  - startDetection(): Starts the detection process.
  - stopDetection(): Stops the detection process.
- **Relationships:**
  - **Associations:** Connected to multiple other classes (like Camera, SignRecognizer, etc.), indicating that it coordinates the overall detection process.

2. ImageProcessor

- **Attributes:**
  - preProcessingAlgorithm: Represents the algorithm used for preprocessing images.
- **Methods:**
  - preProcessImage(): Handles the image preprocessing.

  - extractFeatures(): Extracts relevant features from the preprocessed image.
  - detectSigns(): Detects potential signs from the image.
- **Relationships:**
  - **Association:** Connected to SignRecognizer, indicating that processed images are passed on to the sign recognition module.

3. SignRecognizer

- **Attributes:**
  - model: Represents the machine learning model used for recognition.
  - confidenceLevel: Indicates the confidence level of the recognition.
- **Methods:**
  - recognizeSign(): Recognizes a sign from the processed image.
  - trainModel(): Trains the recognition model.
  - evaluateModel(): Evaluates the performance of the model.
- **Relationships:**
  - **Association:** Connected to ImageProcessor for receiving processed images.

4. Camera

- **Attributes:**
  - resolution: The resolution of the captured images or video.
  - frameRate: The frame rate at which images or video are captured.
  - position: The position of the camera.
- **Methods:**
  - captureFrame(): Captures a single frame.
  - startCapture(): Starts continuous capture.
  - stopCapture(): Stops continuous capture.
- **Relationships:**
  - **Association:** Connected to SignLanguageDetection, indicating it provides the input images or video feed for the system.

5. UserInterface

- **Attributes:**
  - language: The language in which the interface is displayed.
- **Methods:**

    - displaySign(): Displays the detected sign to the user.
    - showLiveFeed(): Displays the live video feed from the camera.
- **Relationships:**
    - **Association:** Connected to SignLanguageDetection, indicating it interacts with the user to display information.

6. Sign

- **Attributes:**
    - name: The name or identifier of the sign.
- **Methods:**
    - getSignDetails(): Retrieves details about the sign.
- **Relationships:**
    - **Association:** Connected to UserInterface, likely to provide information about detected signs.

7. FileManager

- **Attributes:**
    - filePath: The path where files are stored.
    - fileList: A list of stored files.
- **Methods:**
    - saveSign(): Saves detected signs to storage.
    - retrieveSign(): Retrieves stored signs.
- **Relationships:**
    - **Association:** Connected to SignLanguageDetection, indicating it handles the storage and retrieval of detected signs.

8. Audio

- **Attributes:**
    - audioFormat: The format of the audio output.
- **Methods:**
    - playAudio(): Plays an audio message corresponding to the detected sign.
- **Relationship:**
    - **Association**: Connected to SignLanguageDetection to provide audio feedback for the recognized signs.

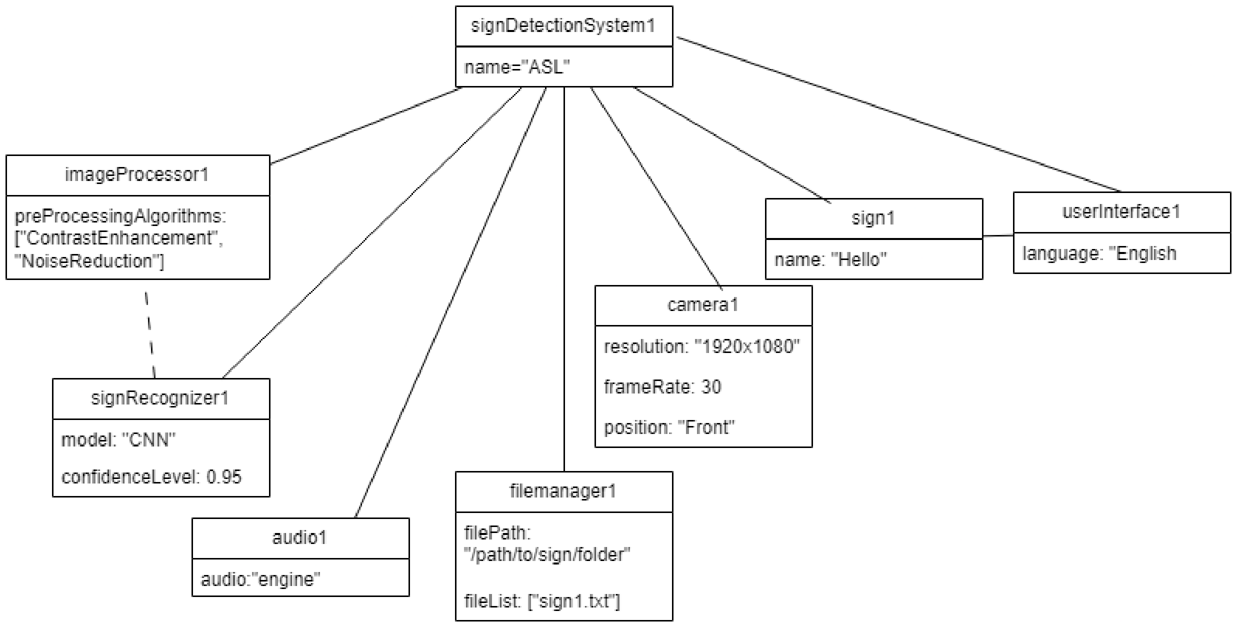


Figure 6: Object Diagram of Sign Language Detection System

1. **signDetectionSystem1**:

   - **name**: "ASL" (indicating that this system is configured for American Sign Language).
   - This object represents the overall detection system coordinating the other components.

2. **imageProcessor1**:

   - **preProcessingAlgorithms**: ["ContrastEnhancement", "NoiseReduction"]
   - This object shows the specific image processing algorithms used for enhancing and cleaning the input images before further analysis.

3. **signRecognizer1**:

   - **model**: "CNN" (Convolutional Neural Network used for sign recognition).
   - **confidenceLevel**: 0.95 (indicating high confidence in the recognition results).
   - This object represents the trained model used to recognize signs with a high level of accuracy.

4. **camera1**:

- **resolution**: "1920x1080" (Full HD resolution for capturing clear images).
- **frameRate**: 30 (frames per second, indicating smooth video capture).
- **position**: "Front" (likely indicates that the camera is positioned at the front, facing the user).
- This object represents the camera settings being used to capture the video feed.

5. **userInterface1**:

- **language**: "English"
- This object represents the user interface configured to display information in English.

6. **sign1**:

- **name**: "Hello"
- This object represents a specific sign detected or being used, which is the word "Hello".

7. **fileManager1**:

- **filePath**: "/path/to/sign/folder" (directory where signs are stored).
- **fileList**: ["sign1.txt"] (list of files stored, here containing one file).
- This object manages the storage and retrieval of detected sign information.

8. **audio1**:

- **audio:** engine
- This object represents the audio system, which plays the corresponding audio for the detected sign in a specific format and for a set duration.

3.1.4 Dynamic Modeling Using State and Sequence Diagram

Dynamic modeling of our system (Sign Language Detection System) is an object-oriented approach that involves creating a dynamic model that represents the behavior of the system at runtime.

This involves identifying the objects that make up the system, their attributes, and the methods that they can perform. Here we have shown the state and sequence diagram of our system.

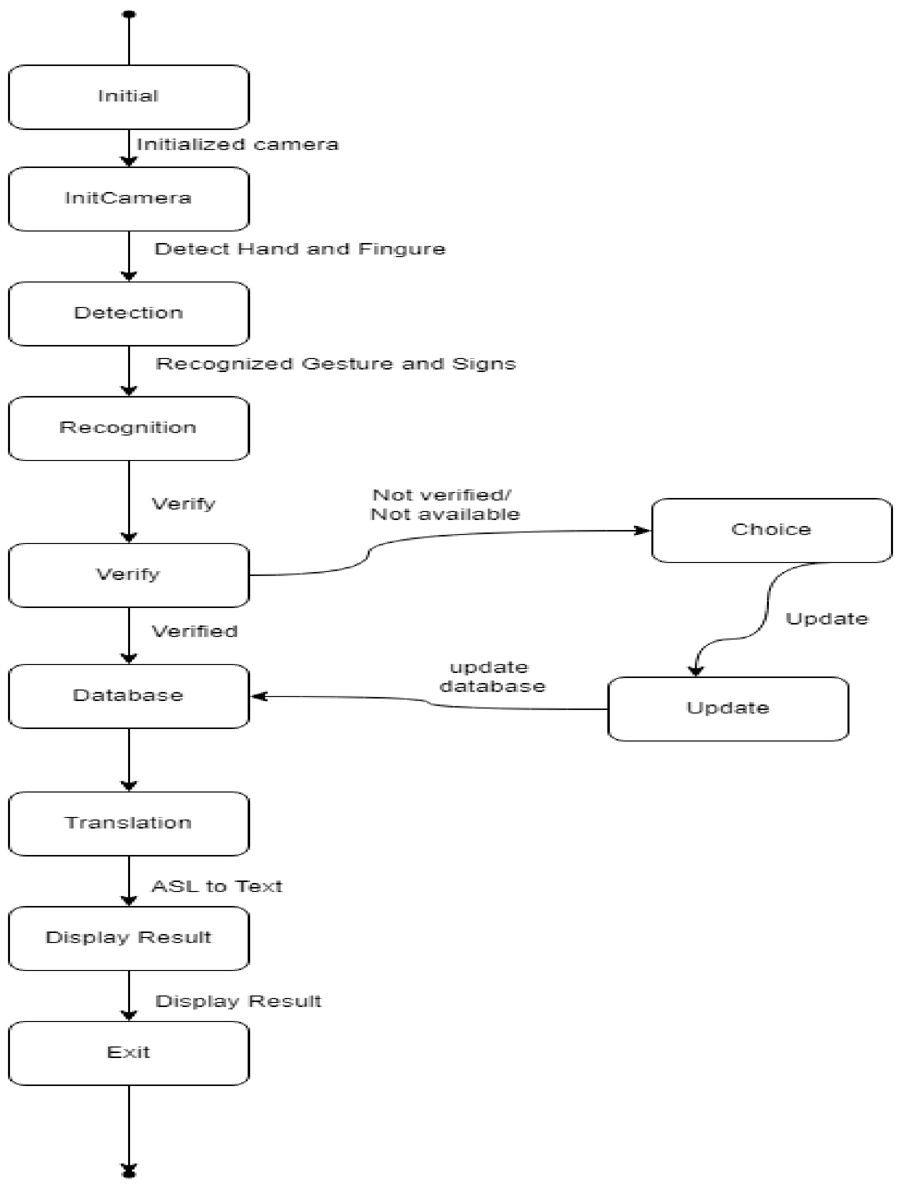


Figure 7: State Diagram of Sign Language Detection System

1. Initial

- The system begins its operation.

2. InitCamera

- The camera is initialized. This step prepares the camera to start detecting hand and finger movements.

3. Detection

- The system detects the hand and finger movements. This step involves identifying gestures and signs from the captured video feed.

4. Recognition

- Detected gestures and signs are recognized. The system matches these detected signs against a known set of signs or gestures.

5. Verify

- The system verifies whether the recognized sign is correct or available in the database.
- **If verified**: The system proceeds to the next step.
- **If not verified/not available**: The user is given a choice to either update the database or proceed without the sign.

6. Choice

- If the sign is not verified or available, the user can choose to update the database.

7. Update

- The user can update the system's database with the new or corrected sign, ensuring that it will be recognized correctly in the future.

8. Database

- After verification, the recognized sign is stored or updated in the system's database for future reference.

9. Translation

- The system translates the recognized American Sign Language (ASL) sign into text.

10. Display Result

- The translated text result is displayed to the user.

11. Exit

- The system completes the process and exits.

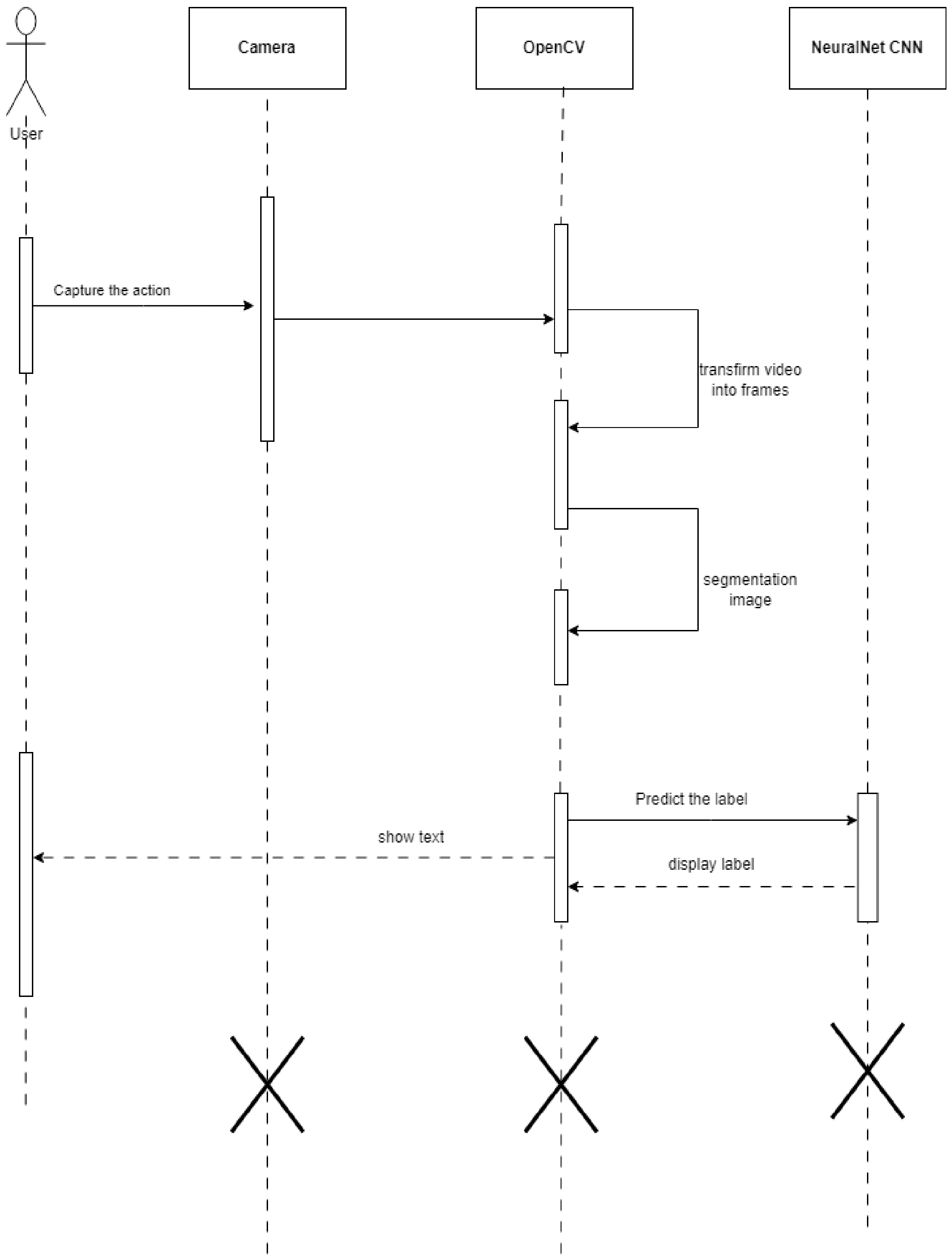


Figure 8: Sequence Diagram of Sign Lanaguage Detection System

1. User:

- The user initiates the process by performing a sign or gesture.

2. Camera:

- **Capture the action**: The camera captures the user's sign or gesture as a video stream.

3. OpenCV:

- **Transform video into frames**: OpenCV, a computer vision library, processes the video by dividing it into individual frames.
- **Segmentation image**: OpenCV performs image segmentation to isolate the relevant parts of the image, such as the hand and fingers, from the background.

4. Neural Net CNN:

- **Predict the label**: The segmented image frames are passed to the CNN, which analyzes the image to predict the corresponding sign or label.
- **Display label**: The predicted label (which could be a word or letter) is sent back to OpenCV.

5. User:

- **Show text**: The recognized sign or label is displayed as text for the user to see.

3.1.5 Process Modeling Activity Diagram

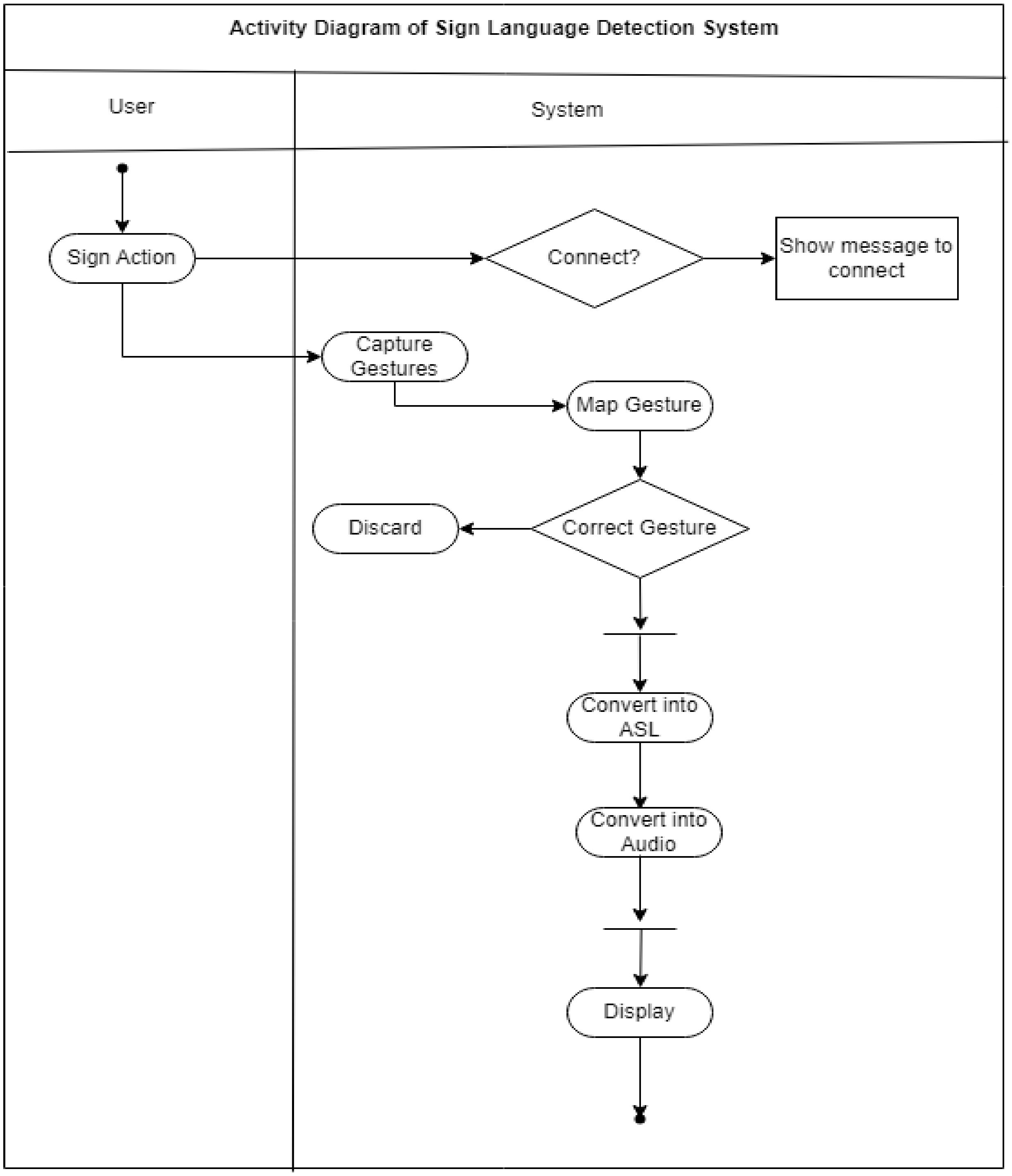


Figure 9: Activity Diagram of Sign Language Detection System

Swimlanes:

- **User**: The person performing the sign language.
- **System**: The sign language detection system.

Workflow:

1. **Connect? (Decision Node)**:
   - The system starts by checking if it is connected.
   - If not connected, a message is shown to prompt the user to connect.
2. **Sign Action (Action)**:
   - The user performs a sign language action.
3. **Capture Gestures (Action)**:
   - The system captures the gestures performed by the user.
4. **Map Gesture (Action)**:
   - The system maps the captured gestures to the corresponding sign language gestures.
5. **Correct Gesture? (Decision Node)**:
   - The system checks if the mapped gesture is correct.
   - If the gesture is incorrect, it is discarded, and the process may loop back to capture new gestures.
6. **Convert into ASL (Action)**:
   - If the gesture is correct, it is converted into American Sign Language (ASL).
7. **Convert into Audio (Action)**:
   - The ASL gesture is then converted into audio.
8. **Display (Action)**:
   - The system displays the converted information, either visually or in audio format.

3.2 System Design

3.2.1 Refinement of Class, Object, State, Sequence and Activity Diagrams

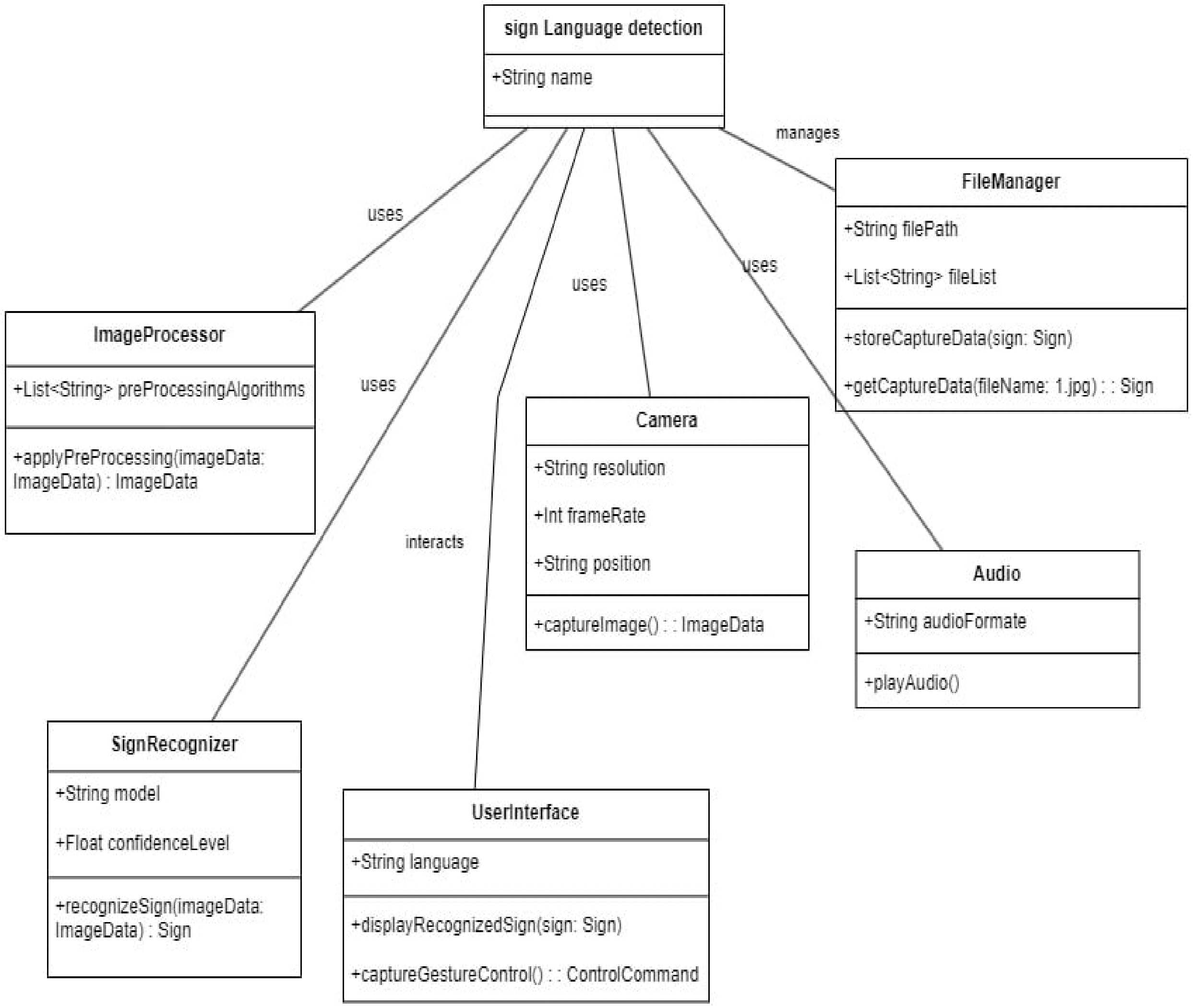

sign Language detection
+String name
manages
FileManager
+String filePath
+List<String> fileList
+storeCaptureData(sign: Sign)
+getCaptureData(fileName: 1.jpg) : : Sign
uses
uses
uses
uses
ImageProcessor
+List<String> preProcessingAlgorithms
+applyPreProcessing(imageData: ImageData) : ImageData
Camera
+String resolution
+Int frameRate
+String position
+captureImage() : : ImageData
interacts
Audio
+String audioFormate
+playAudio()
SignRecognizer
+String model
+Float confidenceLevel
+recognizeSign(imageData: ImageData) : Sign
UserInterface
+String language
+displayRecognizedSign(sign: Sign)
+captureGestureControl() : : ControlCommand

Figure 10: Refinement of Class diagram

1. Sign Language Detection:

- **Attributes**:
  - String name: The name of the sign language detection system.
- **Relationships**:
  - This class interacts with multiple other classes in the system, such as ImageProcessor, Camera, SignRecognizer, UserInterface, and FileManager.

2. **ImageProcessor**:
   - **Attributes**:
     - List<String> preProcessingAlgorithms: A list of preprocessing algorithms applied to image data.
   - **Operations**:
     - applyPreProcessing(imageData: ImageData) : ImageData: Applies preprocessing algorithms to the captured image data and returns the processed image data.
   - **Relationships**:
     - This class is used by the Sign Language Detection class to process images captured by the camera.
3. **Camera**:
   - **Attributes**:
     - String resolution: The resolution of the camera.
     - Int frameRate: The frame rate of the camera.
     - String position: The position of the camera.
   - **Operations**:
     - captureImage() : ImageData: Captures an image and returns the image data.
   - **Relationships**:
     - The Camera class is used by the Sign Language Detection system to capture gestures performed by the signer.
4. **SignRecognizer**:
   - **Attributes**:
     - String model: The model used for sign recognition.
     - Float confidenceLevel: The confidence level of the recognition.
   - **Operations**:
     - recognizeSign(imageData: ImageData) : Sign: Recognizes a sign from the processed image data and returns the recognized sign.
   - **Relationships**:
     - The SignRecognizer class is used by the Sign Language Detection system to recognize signs from the processed image data.
5. **UserInterface**:

- **Attributes**:
    - String language: The language setting for the user interface.
- **Operations**:
    - displayRecognizedSign(sign: Sign): Displays the recognized sign to the user.
    - captureGestureControl() : ControlCommand: Captures control commands from the user, such as gestures to be recognized.

- **Relationships**:
    - The UserInterface class interacts with the Sign Language Detection system to display recognized signs and capture user input.

6. **FileManager**:
    - **Attributes**:
        - String filePath: The path where sign data files are stored.
        - Array<String> fileArray: A list of files managed by the system.
    - **Operations**:
        - storeSignData(sign: Sign): Stores the recognized sign data in a file.
        - readSignData(fileName: String) : Sign: Reads sign data from a file.
    - **Relationships**:
        - The FileManager class is managed by the Sign Language Detection system and handles file operations related to sign data.
7. **Audio**:
    - **Attributes:**
        - String audio: Play the audio related to sign language.
    - **Operations:**
        - playAudio() : void: Plays the audio file or stream related to the sign language recognition process.
    - **Relationship:**
        - The Audio class is linked to the Sign Language Detection system and operates as an optional output for playing audio associated with the recognized signs.

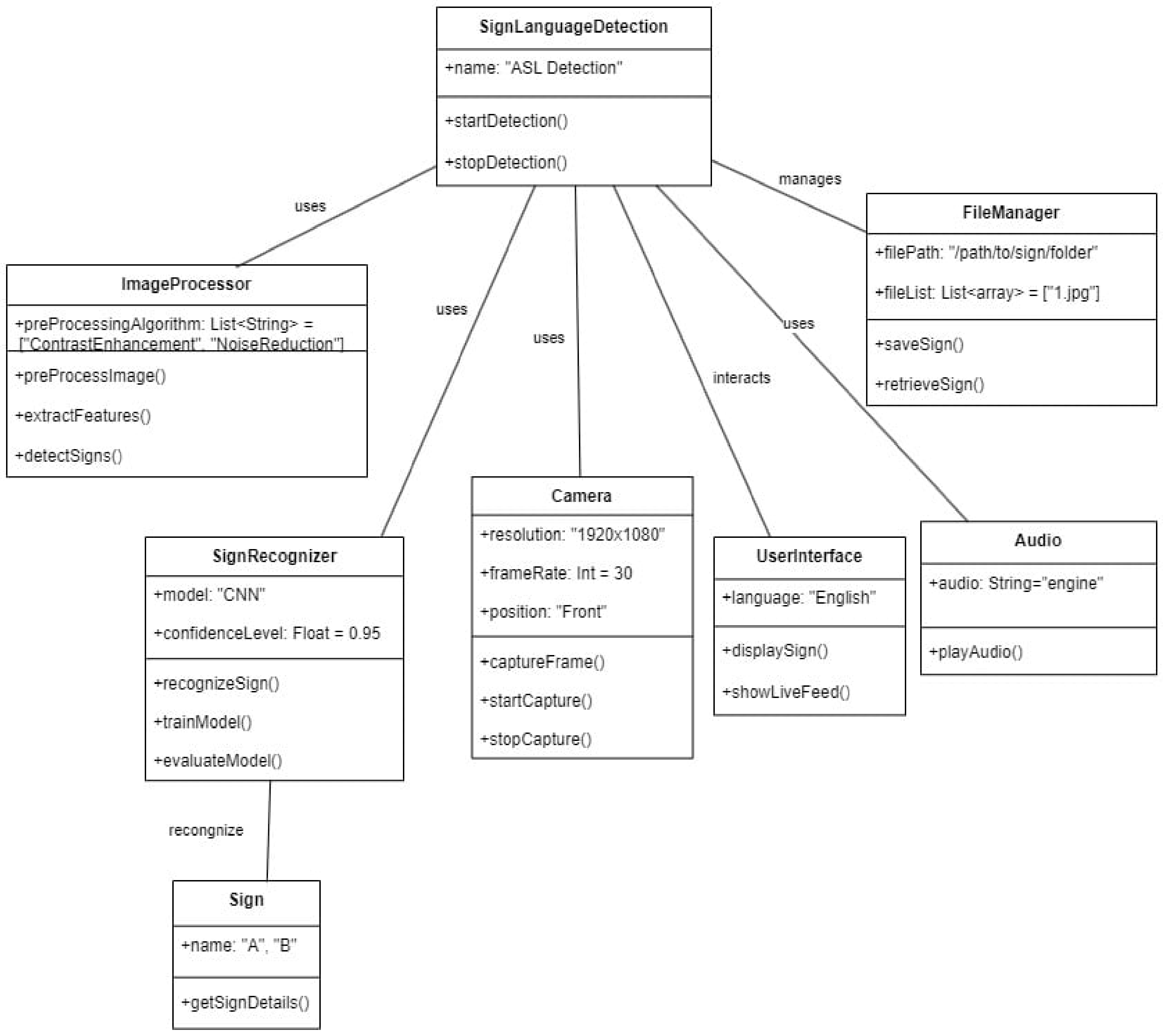


Figure 11: Refinement of Object Diagram

1. **SignLanguageDetection**:

- **Attributes**:
  - name: "ASL Detection": The name of the system, specifying that it detects American Sign Language (ASL).
- **Operations**:
  - startDetection(): Starts the sign language detection process.
  - stopDetection(): Stops the detection process.

- **Relationships**:
    - The central class, which uses other components to perform detection, interacts with the user interface, and manages file storage.

2. **ImageProcessor**:

- **Attributes**:
    - preProcessingAlgorithm: List<String> = ["ContrastEnhancement", "NoiseReduction"]: A list of preprocessing algorithms used to enhance image data.
- **Operations**:
    - preProcessImage(): Applies preprocessing techniques to the image data.
    - extractFeatures(): Extracts relevant features from the image data for further processing.
    - detectSigns(): Detects signs in the preprocessed image data.
- **Relationships**:
    - Used by the SignLanguageDetection system to process images captured by the camera.

3. **Camera**:

- **Attributes**:
    - resolution: "1920x1080": The camera's resolution.
    - frameRate: Int = 30: The frame rate of the camera.
    - position: "Front": The position of the camera.
- **Operations**:
    - captureFrame(): Captures a single frame of image data.
    - startCapture(): Begins capturing a sequence of frames.
    - stopCapture(): Stops capturing frames.
- **Relationships**:
    - The Camera class is used to capture images for processing.

4. **SignRecognizer**:

- **Attributes**:
    - model: "CNN": The model used for sign recognition, in this case, a Convolutional Neural Network (CNN).
    - confidenceLevel: Float = 0.95: The confidence level of the recognition.

- **Operations**:
    - recognizeSign(): Recognizes signs from processed image data.
    - trainModel(): Trains the sign recognition model.
    - evaluateModel(): Evaluates the performance of the recognition model.
- **Relationships**:
    - The SignRecognizer class is responsible for identifying signs and interacts with the Sign class.

5. **Sign**:
    - **Attributes**:
        - name: "A", "B": Represents recognized signs, for instance, the letters "A" and "B".
    - **Operations**:
        - getSignDetails(): Retrieves details of the recognized sign.
    - **Relationships**:
        - The Sign class represents the signs recognized by the SignRecognizer.
6. **UserInterface**:
    - **Attributes**:
        - language: "English": The language setting for the interface.
    - **Operations**:
        - displaySign(): Displays the recognized sign to the user.
        - showLiveFeed(): Shows a live feed of the camera capturing gestures.
    - **Relationships**:
        - The UserInterface class interacts with the user to display information and provide control options.
7. **FileManager**:
    - **Attributes**:
        - filePath: "/path/to/sign/folder": The directory path where sign data is stored.
        - fileList: Array<String> = ["sign1.jpg"]: A list of files storing recognized signs.
    - **Operations**:
        - saveSign(): Saves the recognized sign data to a file.
        - retrieveSign(): Retrieves sign data from a file.
    - **Relationships**:

- The FileManager class is responsible for managing the storage and retrieval of sign data.

8. **Audio:**

- **Attributes**:
    - audio: "engine": Play the audio related to gesture.
- **Operations**:
    - playAudio(): Plays the audio file related to the recognized sign.
- **Relationships**:
    - The Audio class interacts with the SignLanguageDetection system to provide auditory feedback to the user, complementing the visual display on the UserInterface.

3.2.2 Deployment Diagram

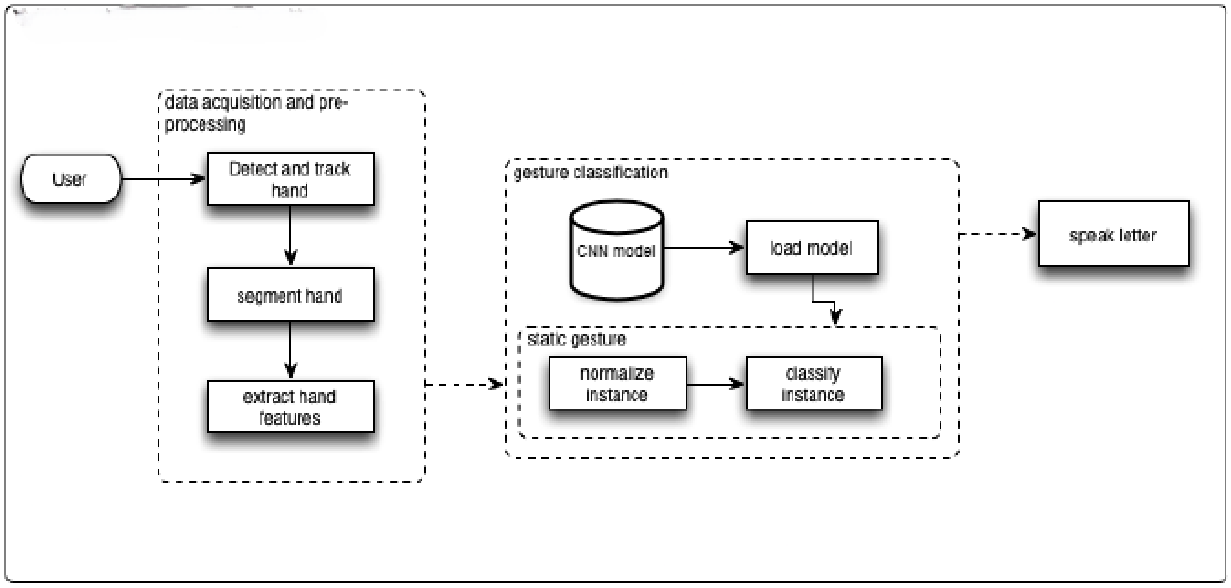


Figure 12: Deployment Diagram

3.3 Algorithm Details

Convolutional Neural Network Model: Convolutional Neural Network (CNN) is a class of deep, feed-forward artificial neural networks, most commonly applied to analyze visual imagery. CNNs use a variation of multilayer perceptron designed to require minimal preprocessing. CNN has proved to be a significant milestone in the area of detection and classification of images.

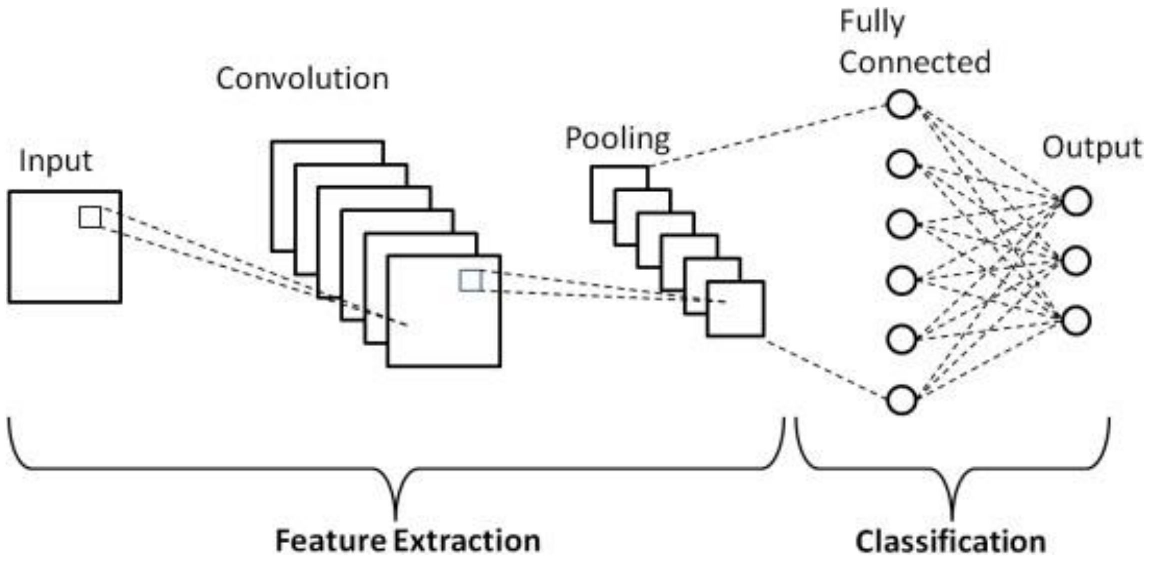


Figure 13: Systematic Diagram of basic CNN [6]

The architecture of model consists of two convolutional layer, a pooling layer, and one Artificial Neural Network (ANN). The convolutional layer and pooling layer are used to extract features and ANN is used to classify the features. In convolution layers, 32 feature maps are sampled of the input to generate a feature map using convolution operation which is further sampled by pooling matrix using max pooling technique. The convolution and max pooling operations are further repeated once. The output is then attended into a 1D vector and supplied to the ANN as an input. Then, the ANN uses the input and the weight which are generated from training, and the 128 neurons in the fully connected (hidden) layer for the image classification into one of the classes.

1. Convolution Operation

Convolutional Layer:

- **Input:** An image represented as a matrix of pixel values (e.g., height × width × depth).
- **Filter (Kernel):** A smaller matrix (e.g., 3×3 or 5×5) used to detect features in the image.

Mathematical Formula:

The convolution operation involves sliding the filter across the input image and performing element-wise multiplication followed by summation. For a given input image I and filter K:

$$(I * K)(x,y) = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} I(x+i, y+j).K(i,j)$$

Where:

- (x,y) are the coordinates of the top-left corner of the filter position.
- (i,j) are the coordinates within the filter.
- m and n are the dimensions of the filter.

2. Activation Function

ReLU (Rectified Linear Unit):

The ReLU activation function introduces non-linearity into the model. For each element xxx in the feature map:

$$\boldsymbol{ReLU}(\boldsymbol{x}) = max(\boldsymbol{0}, \boldsymbol{x})$$

Sigmoid Function (for binary classification):

$$\boldsymbol{\sigma}(\boldsymbol{x}) = \frac{\mathbf{1}}{\mathbf{1} + \boldsymbol{e}^{-x}}$$

Softmax Function (for multi-class classification):

$$\boldsymbol{Softmax}(\boldsymbol{zi}) = \frac{e^{z_j}}{\sum_j e^{z_j}}$$

3. Pooling Operation

Max Pooling:

Reduces the spatial dimensions by taking the maximum value in each region. For a 2×2 max pooling operation:

$$\boldsymbol{MaxPool}(\boldsymbol{x}, \boldsymbol{y}) = max\{\boldsymbol{I}(\boldsymbol{x}, \boldsymbol{y}), \boldsymbol{I}(\boldsymbol{x} + \mathbf{1}, \boldsymbol{y}), \boldsymbol{I}(\boldsymbol{x}, \boldsymbol{y} + \mathbf{1}), \boldsymbol{I}(\boldsymbol{x} + \mathbf{1}, \boldsymbol{y} + \mathbf{1})\}$$

Average Pooling:

Reduces the spatial dimensions by taking the average value in each region:

$$\boldsymbol{AvgPool(x,y) = 41(I(x,y) + I(x+1,y) + I(x,y+1) + I(x+1,y+1))}$$

4. Fully Connected Layers

After convolution and pooling, the feature maps are flattened into a vector and passed through fully connected (dense) layers.

Fully Connected Layer:

If z is the input vector and W is the weight matrix, the output y is computed as:

$y{=}W{\cdot}\, z + b$

Where b {b}b is the bias vector.

5. Loss Function

Cross-Entropy Loss (for classification):

Measures the difference between the predicted probability distribution and the true distribution:

$$\boldsymbol{Loss} = \sum_i yi.\log(\hat{y}_i)$$

Where:

- y_i is the true label for class i.
- $\hat{y}$ is the predicted probability for class iii.

6. Backpropagation and Optimization

Gradient Descent: The weights and biases of the network are updated using the gradients computed through backpropagation. For weight WWW and learning rate α\alphaα:

$$\boldsymbol{W \leftarrow W - \alpha \frac{\alpha Loss}{\alpha W}}$$

Where $\frac{\alpha Loss}{\alpha W}$ is the gradient of the loss with respect to the weight.

Chapter 4: Implementation and Testing

4.1 Implementation

To develop the system, several software development tools, programming languages, and libraries were utilized. The selection of tools was made based on the specific requirements of the Sign Language Detection System, including image processing, machine learning, and user interface functionalities.

4.1.1 Tool Used

- **CASE Tools**:
  - o Unified Modeling Language (UML) Tools: Used for designing the system architecture and object models. Examples include Visual Paradigm and Lucidchart.
  - o PyCharm: Python IDE used for the development and debugging of the system.
- **Programming Languages**:
  - o Python: The primary programming language used in the project for its rich support in image processing (OpenCV) and machine learning (TensorFlow/Keras).
  - o HTML/CSS: Used for the user interface design when creating a web-based interface for the system.
- **Libraries**:
  - o OpenCV: A library used for image and video processing, crucial for capturing and pre-processing sign language gestures.
  - o TensorFlow/Keras: Used for building and training the Convolutional Neural Network (CNN) model for sign recognition.
  - o PyAudio: Used for handling the audio playback system.
- **Database Platform**:
  - o FileManager: Instead of a traditional database, a file-based system is used for storing detected sign information and associated metadata in text files. This simplifies the management of data for this project.

4.1.2 Implementation Details of Modules

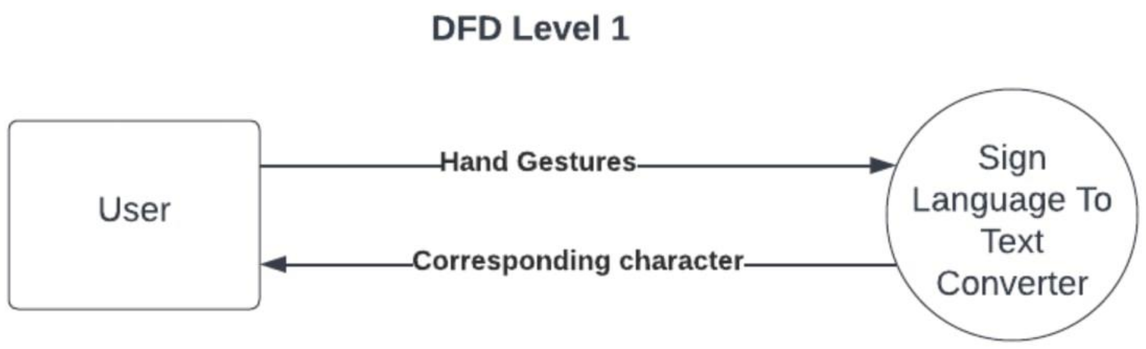


Figure 14: DFD level 1 of Sign Language Detection System

DFD Level 1

The Level 1 DFD provided an overview of the interaction between the user and the system, which included the sign language detection process.

User: The user performed hand gestures that represented specific characters in sign language.

Sign Language to Text Converter: This process received the hand gestures from the user and converted them into corresponding characters, which could then be used for communication.

Flow:

- **Input:** The user provided hand gestures.
- **Process:** The system interpreted the hand gestures using the Sign Language to Text Converter.
- **Output:** The converter returned the corresponding character to the user, completing the process of converting sign language into text.

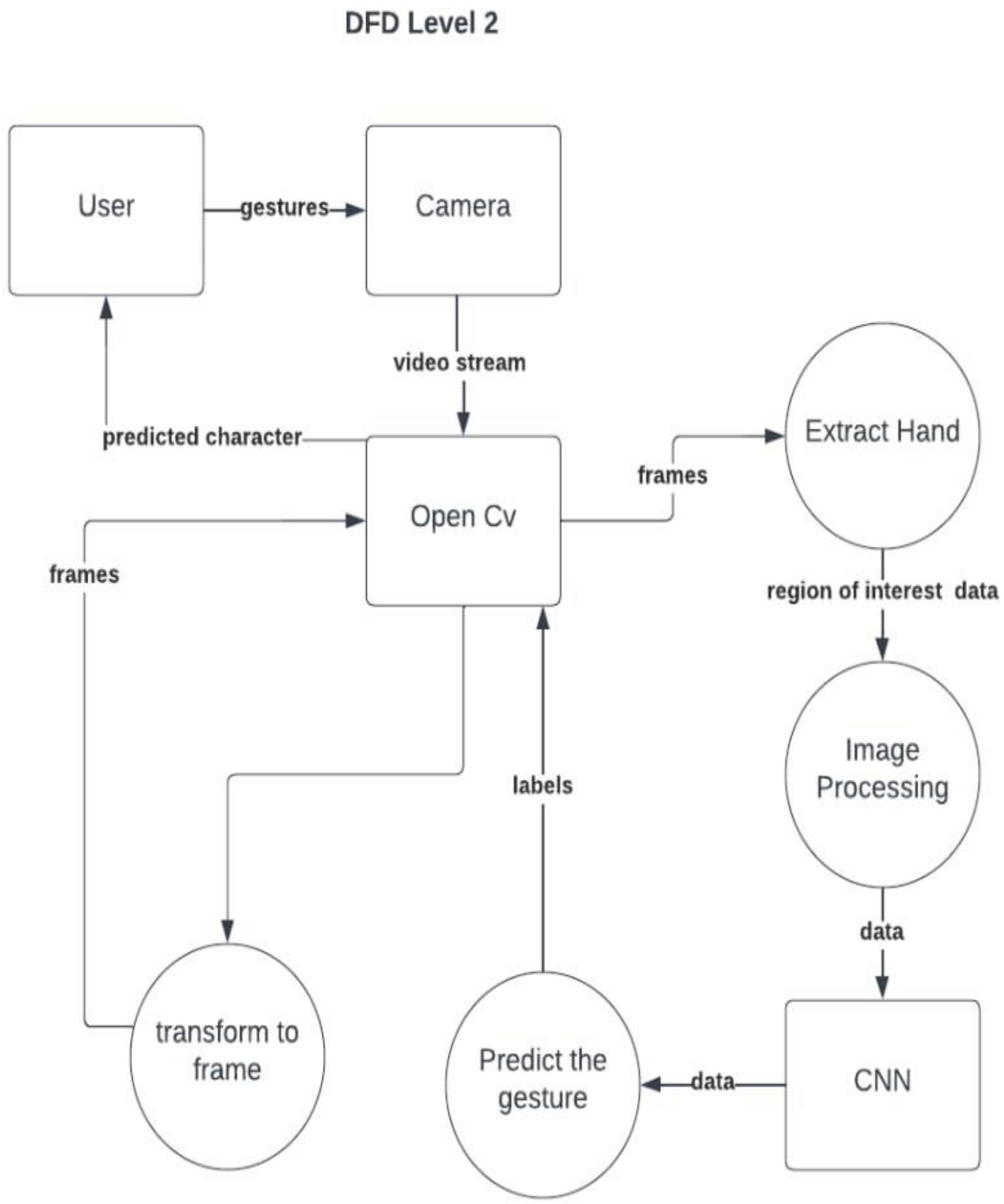


Figure 15: DFD Level 2 of Sign Language Detection System

DFD Level 2

The Level 2 DFD provided a more detailed view of the processes involved in converting the hand gestures into text. It broke down the internal workings of the "Sign Language to Text Converter" into more specific components and explained how they interacted.

User and Camera: The user performed gestures, which were captured by the camera as a video stream.

OpenCV: The OpenCV module processed the video stream to extract individual frames of the hand gestures.

Extract Hand: From these frames, OpenCV sent the relevant frames to the Extract Hand process to locate the region of interest—the user's hand.

Image Processing and Convolutional Neural Network (CNN): The region of interest data was then passed on to the Image Processing unit to further refine and prepare the data. The processed data was then input to a CNN (Convolutional Neural Network), which analyzed the features and classified the gesture.

Prediction and Transformation:

Predict the Gesture: The CNN outputted labels corresponding to specific characters, which were then used by the Predict the Gesture process to determine the user's gesture.

Transform to Frame: The predicted character was transformed into a frame that represented the output, and it was fed back to the user as the predicted character.

Flow:

- The camera captured the gestures from the user and sent the video stream to OpenCV.
- OpenCV processed the frames, extracted the region of interest, and sent it for Image Processing.
- The CNN was used to analyze and predict the gesture based on the processed images.
- Finally, the predicted character was returned to the user, completing the conversion of the gesture into text.

4.1.3 Implementation Details of Algorithm

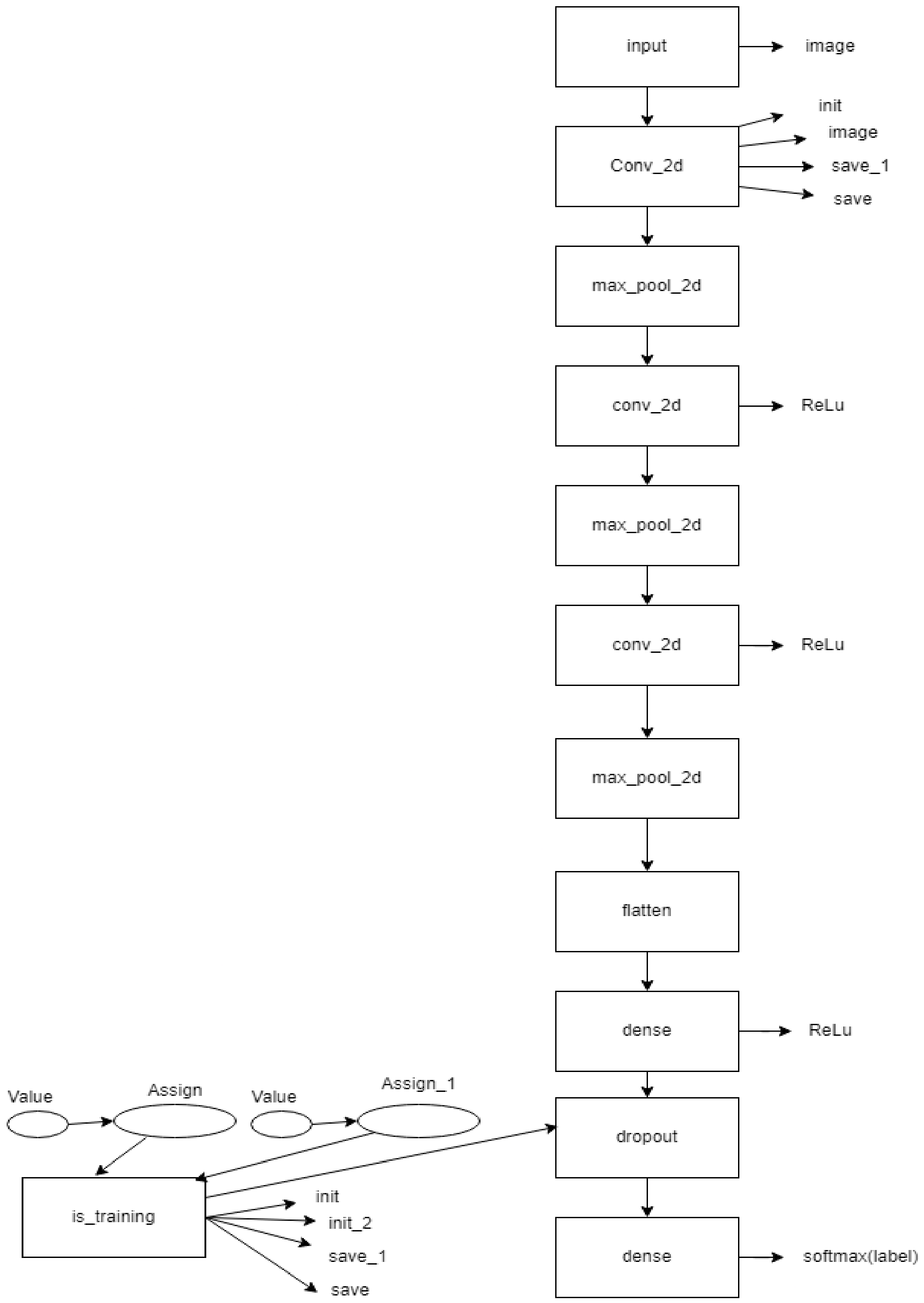


Figure 16: Flow diagram of CNN

First Convolutional Block: A Conv2D layer with 32 filters (3x3), using ReLU, takes 400x400 RGB images, followed by MaxPooling to downsample.

Second Convolutional Block: Adds a Conv2D layer with 64 filters and MaxPooling to capture more complex features.

Third Convolutional Block: Another Conv2D layer with 128 filters and MaxPooling for deeper feature learning.

Flattening Layer: Flattens the output from the convolutional layers into a 1D vector for the dense layer.

Dense Layer: A fully connected layer with 512 neurons using ReLU to combine learned features.

Dropout Layer: A Dropout layer (50%) to reduce overfitting.

Output Layer: A Dense layer with softmax activation for classifying 26 classes (A-Z). Model

Summary: Prints the architecture details, including layers and parameters.

4.2 Testing

Table 1: Black Box Testing

Test Case ID	**Test Scenario**	**Expected Result**	**Actual Result**	**Pass/Fail**
0	Check for the letter "A"	The letter "A" should be displayed on the screen.	Fluctuation between "A" and "Z"	Fail
1	Check for the letter "B"	The letter "B" should be displayed on the screen.	As Expected	Pass

2	Check for the letter "C"	The letter "C" should be displayed on the screen.	As Expected	Pass
3	Check for the letter "D"	The letter "D" should be displayed on the screen.	As Expected	Pass
4	Check for the letter "E"	The letter "E" should be displayed on the screen.	As Expected	Pass
5	Check for the letter "F"	The letter "F" should be displayed on the screen.	As Expected	Pass

Table 2: Unit Testing

Test Case ID	Test Scenario	Expected Result	Actual Result	Pass/Fail
0	Check for the camera opening function	The camera should be opened	As Expected	Pass
1	Check for the prepared dataset function	It should create the .csv and store the landmark in .csv file	As Expected	Pass

2	Check for the landmark detection	It should detect the landmarks given by the users.	As Expected	Pass

4.3 Result Analysis

4.3.1 Epoch vs Loss Graph

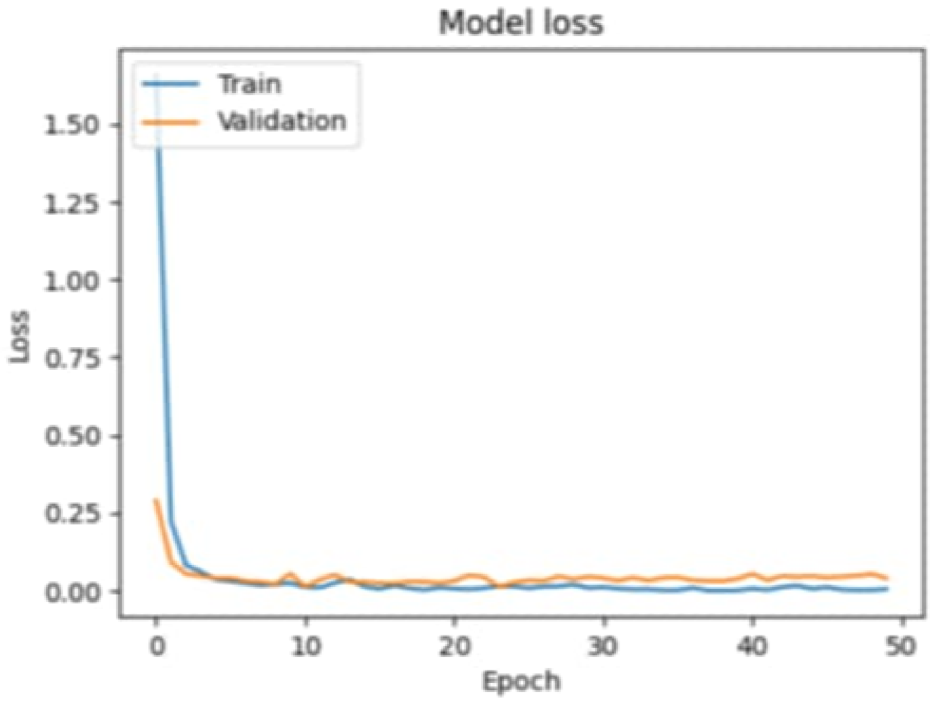


Figure 17: Epoch vs Loss Graph

The first quantity that is useful to track during training and testing is the loss, evaluated on individual batches during the forward pass. The figure shows the model loss versus epochs during both training (orange curve) and testing (blue curve). As the number of epochs increases, the loss for both curves decrease, and at the final point, the training loss is much lower than the testing loss.

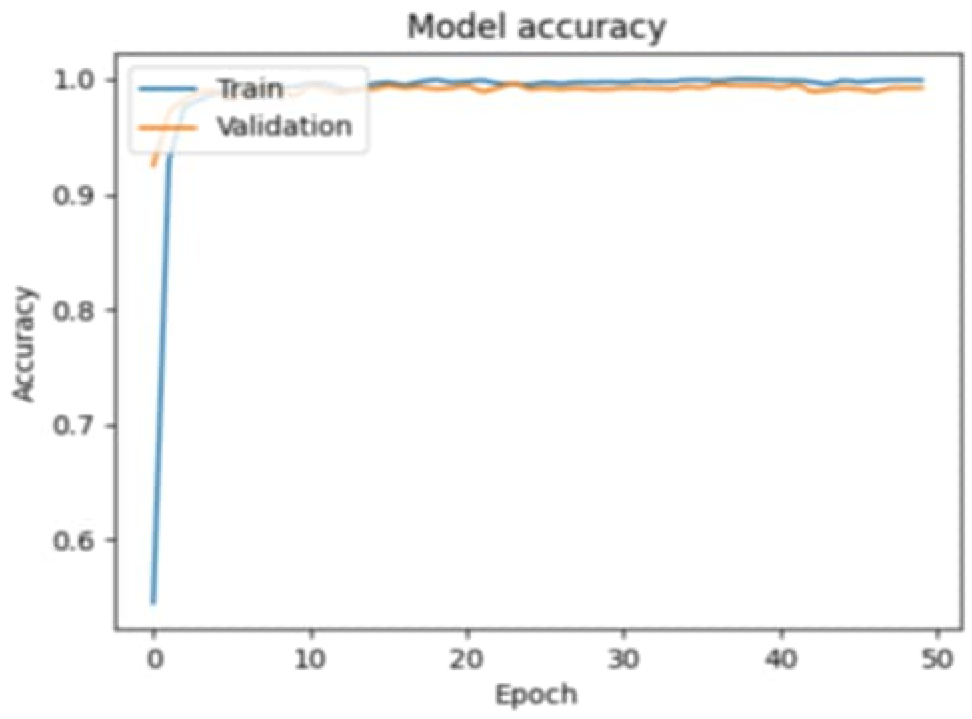


Figure 18: Epoch vs Accuracy Graph

Another commonly used curve to understand the progress of the model is the accuracy curve. The figure shows the epoch versus accuracy graph. With an increase in the number of epochs, both the training accuracy and test accuracy increase. By the final point, there is a significant gap between the training curve and the test curve, with the training accuracy being much higher than the testing accuracy. This indicates that the model is not overfitting and performs well, achieving an accuracy of about 93%.

Chapter 5: Conclusion and Future Recommendations

5.1 Conclusion

The project successfully developed a Sign Language Detection System that addressed the communication challenges faced by speech-impaired individuals. By leveraging advanced image processing and machine learning techniques, particularly a Convolutional Neural Network (CNN), the system was able to recognize American Sign Language gestures in real-time. It effectively converted these gestures into text and speech, facilitating seamless communication between speech-impaired individuals and the hearing population. The results demonstrated that the system achieved a high level of accuracy and usability, making it a valuable tool in environments where effective communication is essential.

5.2 Future Recommendations

Even if this project doesn't incorporate all the gestures, the procedure can be applied to train more gestures. It opens the door to a new and more accessible way to recognize sign language. In other words, it is not possible to recognize all of ASL yet; however, it encourages further study of this approach in order to succeed.

This project is an initial step toward reaching an effective solution for daily concerns. It can be extended in multiple ways in the future, such as:

- **Android Application:** Users will also be able to communicate through a mobile app.
- **Adding New Gestures:** Users will be able to add their own gestures to the dataset.
- **Multi-Language Support**: Incorporating support for different sign languages and spoken languages to broaden usability.
- **Pre-Image Recognition**: Allowing users to upload images for sign recognition without requiring live translation.

Reference

[1] M. Selli, "european commission," 31 5 2021. [Online]. Available: https://epale.ec.europa.eu/en/blog/learning-training-guide-learning-basic-sign-language-phrases-using-them-field-tourism.

[2] Laoyan, "Revolutionizing Software Delivery: How Test Automation Elevates Agile Methodologies," 2024. [Online]. Available: https://www.smartdev.com/revolutionizing-software-delivery-how-test-automation-elevates-agile-methodologies/.

[3] P. UH, P. SK and J. MM, "American sign language recognition using CNN," *International Journal of Research in Engineering, Science and Management,* vol. 3, no. 7, pp. 333-336, 2020.

[4] K. Byeongkeun, S. Tripathi and T. Q. Nguyen, "Real-time sign language fingerspelling recognition using convolutional neural networks from depth map," *3rd IAPR Asian Conference on Pattern Recognition,* pp. 136-140, 2015.

[5] B. B, G. D, H. R, L. Z, and S. R, "Simplifying sign language detection for smart home devices using Google MediaPipe," 2021.

[6] "Schematic diagram of a basic convolutional neural network (CNN) architecture," [Online]. Available:https://www.researchgate.net/publication/336805909_A_High_Accuracy_Model_Average_Ensemble_of_Convolutional_Neural_Networks_for_Classification_of_Cloud_Image_Patches_on_Small_Datasets/figures? =1.

Appendix

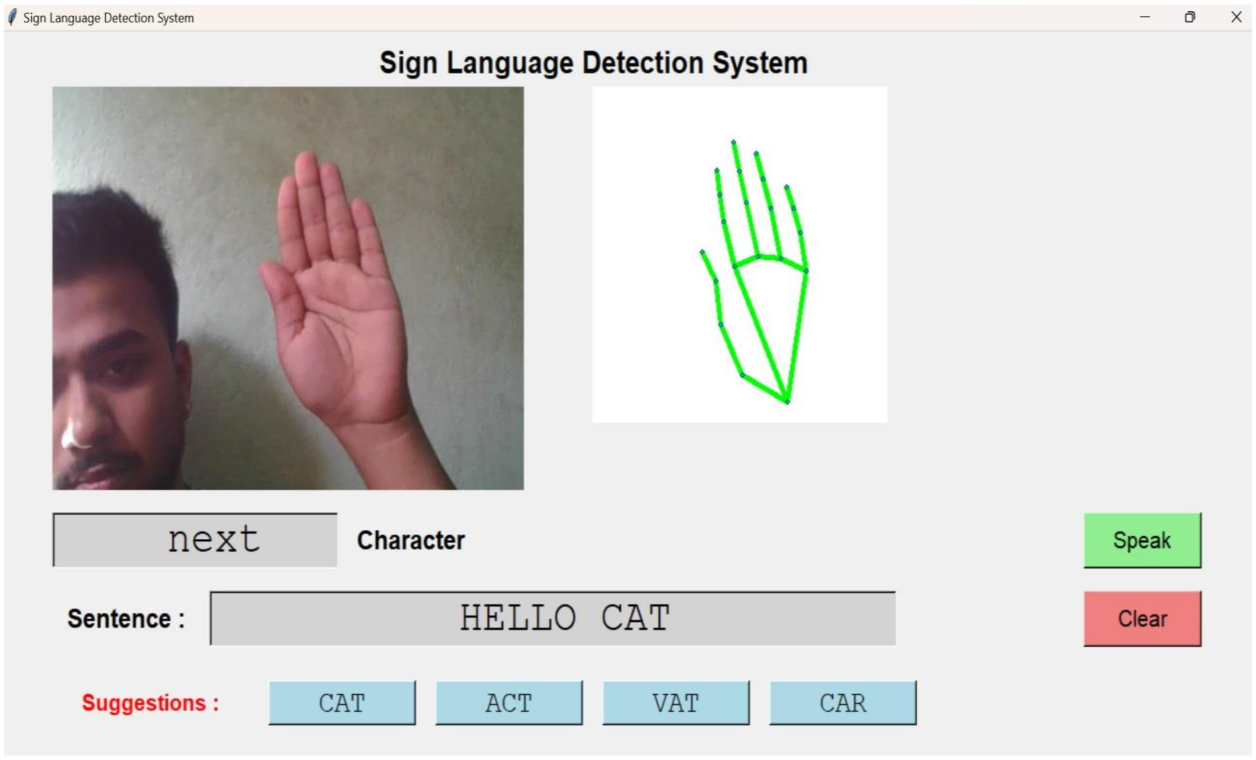
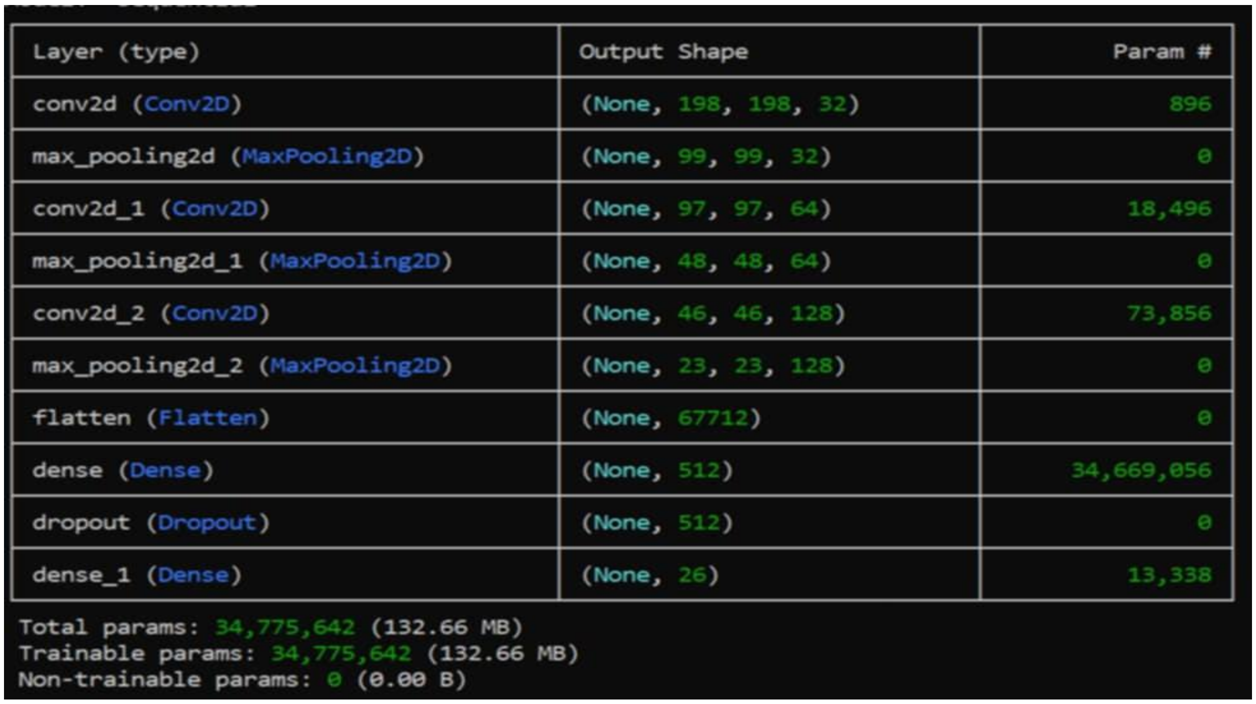
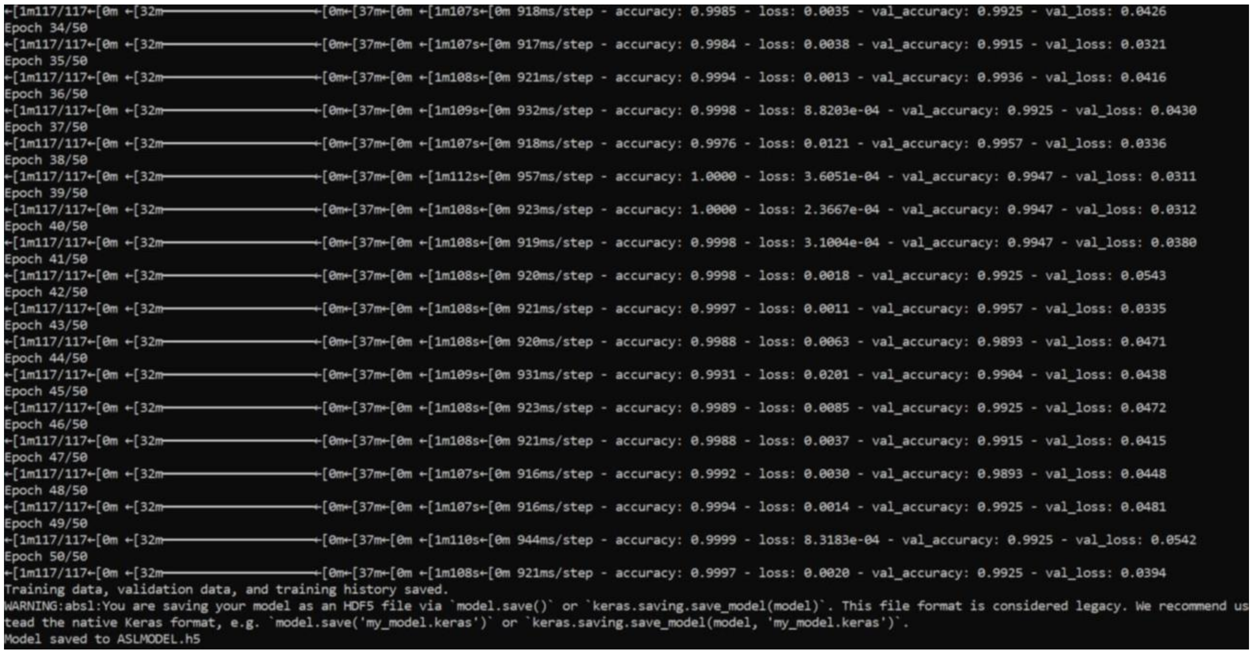


Figure 19: Output



```
Layer (type)                      Output Shape                  Param #
conv2d (Conv2D)                   (None, 198, 198, 32)              896
max_pooling2d (MaxPooling2D)      (None, 99, 99, 32)                  0
conv2d_1 (Conv2D)                 (None, 97, 97, 64)             18,496
max_pooling2d_1 (MaxPooling2D)    (None, 48, 48, 64)                  0
conv2d_2 (Conv2D)                 (None, 46, 46, 128)            73,856
max_pooling2d_2 (MaxPooling2D)    (None, 23, 23, 128)                 0
flatten (Flatten)                 (None, 67712)                       0
dense (Dense)                     (None, 512)                34,669,056
dropout (Dropout)                 (None, 512)                         0
dense_1 (Dense)                   (None, 26)                     13,338
Total params: 34,775,642 (132.66 MB)
Trainable params: 34,775,642 (132.66 MB)
Non-trainable params: 0 (0.00 B)
```

Figure 20: Model training



```
←[1m117/117←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m107s←[0m 918ms/step - accuracy: 0.9985 - loss: 0.0035 - val_accuracy: 0.9925 - val_loss: 0.0426
Epoch 34/50
←[1m117/117←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m107s←[0m 917ms/step - accuracy: 0.9984 - loss: 0.0038 - val_accuracy: 0.9915 - val_loss: 0.0321
Epoch 35/50
←[1m117/117←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m108s←[0m 921ms/step - accuracy: 0.9994 - loss: 0.0013 - val_accuracy: 0.9936 - val_loss: 0.0416
Epoch 36/50
←[1m117/117←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m109s←[0m 932ms/step - accuracy: 0.9998 - loss: 8.8203e-04 - val_accuracy: 0.9925 - val_loss: 0.0430
Epoch 37/50
←[1m117/117←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m107s←[0m 918ms/step - accuracy: 0.9976 - loss: 0.0121 - val_accuracy: 0.9957 - val_loss: 0.0336
Epoch 38/50
←[1m117/117←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m112s←[0m 957ms/step - accuracy: 1.0000 - loss: 3.6051e-04 - val_accuracy: 0.9947 - val_loss: 0.0311
Epoch 39/50
←[1m117/117←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m108s←[0m 923ms/step - accuracy: 1.0000 - loss: 2.3667e-04 - val_accuracy: 0.9947 - val_loss: 0.0312
Epoch 40/50
←[1m117/117←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m108s←[0m 919ms/step - accuracy: 0.9998 - loss: 3.1004e-04 - val_accuracy: 0.9947 - val_loss: 0.0380
Epoch 41/50
←[1m117/117←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m108s←[0m 920ms/step - accuracy: 0.9998 - loss: 0.0018 - val_accuracy: 0.9925 - val_loss: 0.0543
Epoch 42/50
←[1m117/117←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m108s←[0m 921ms/step - accuracy: 0.9997 - loss: 0.0011 - val_accuracy: 0.9957 - val_loss: 0.0335
Epoch 43/50
←[1m117/117←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m108s←[0m 920ms/step - accuracy: 0.9988 - loss: 0.0063 - val_accuracy: 0.9893 - val_loss: 0.0471
Epoch 44/50
←[1m117/117←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m109s←[0m 931ms/step - accuracy: 0.9931 - loss: 0.0201 - val_accuracy: 0.9904 - val_loss: 0.0438
Epoch 45/50
←[1m117/117←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m108s←[0m 923ms/step - accuracy: 0.9989 - loss: 0.0085 - val_accuracy: 0.9925 - val_loss: 0.0472
Epoch 46/50
←[1m117/117←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m108s←[0m 921ms/step - accuracy: 0.9988 - loss: 0.0037 - val_accuracy: 0.9915 - val_loss: 0.0415
Epoch 47/50
←[1m117/117←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m107s←[0m 916ms/step - accuracy: 0.9992 - loss: 0.0030 - val_accuracy: 0.9893 - val_loss: 0.0448
Epoch 48/50
←[1m117/117←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m107s←[0m 916ms/step - accuracy: 0.9994 - loss: 0.0014 - val_accuracy: 0.9925 - val_loss: 0.0481
Epoch 49/50
←[1m117/117←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m110s←[0m 944ms/step - accuracy: 0.9999 - loss: 8.3183e-04 - val_accuracy: 0.9925 - val_loss: 0.0542
Epoch 50/50
←[1m117/117←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m108s←[0m 921ms/step - accuracy: 0.9997 - loss: 0.0020 - val_accuracy: 0.9925 - val_loss: 0.0394
Training data, validation data, and training history saved.
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend us
tead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
Model saved to ASLMODEL.h5
```

Figure 21: Save Model

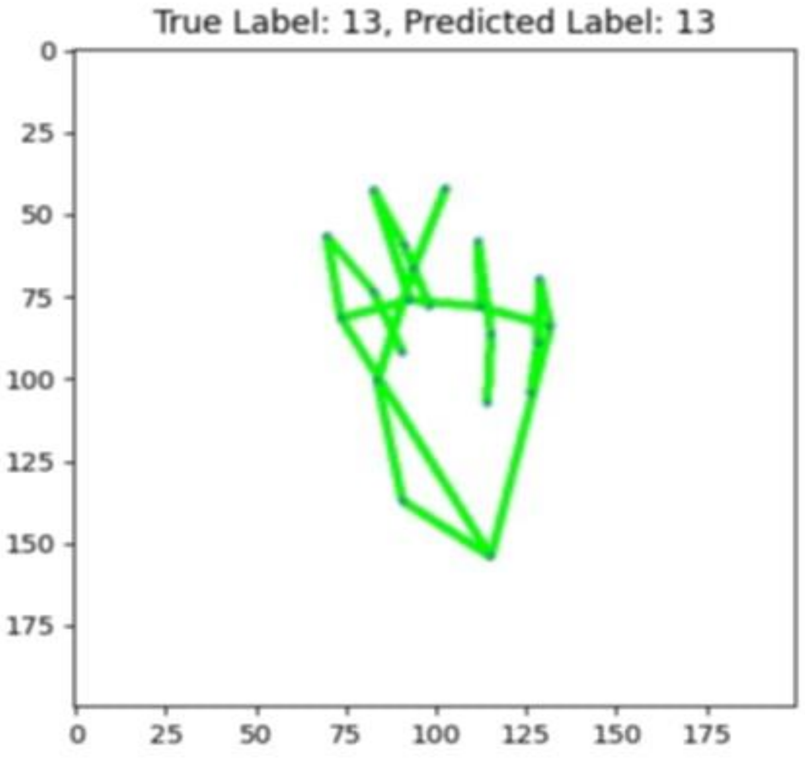


Figure 22: Predicted Label