Tribhuvan University

Institute of Science and Technology

Bhaktapur Multiple Campus, Bhaktapur

Project Report On

GestureSpeak: Real-Time Sign Language Recognition System

Under Supervision of:

Bishwa Nath Lal Karn

Department of CSIT, BMC

Submitted by:

Adarsha Rimal (25716/077)

Sagar Gautam (25758/077)

Shiva Ram Dhakal (25767/077)

Submitted to:

Tribhuvan University

Institute of Science and Technology

February, 2025

Tribhuvan University

Institute of Science and Technology

Bhaktapur Multiple Campus, Bhaktapur

SUPERVISOR'S RECOMMENDATION

I hereby recommend that the project work report entitled **GestureSpeak: Real-Time Sign Language Recognition System** submitted by **Mr. Adarsha Rimal** (25716/077), **Mr. Sagar Gautam** (25758/077) and **Mr. Shiva Ram Dhakal**(25767/077) of **Bhaktapur Multiple Campus, Doodhpati, Bhaktapur** is prepared under my supervision as per the procedure and format requirement laid by the Faculty of B.Sc. CSIT, Tribhuvan University, in the partial fulfillment of the requirements for the Bachelor's Degree in Computer Science and Information Technology (B. Sc. CSIT). I, therefore, recommend the project work report for evaluation.

.....

Bishwa Nath Lal Karn

Bhaktapur Multiple Campus

B.Sc. CSIT Department

Tribhuvan University

Institute of Science and Technology

Bhaktapur Multiple Campus, Bhaktapur

LETTEROFAPPROVAL

We certify that we have read this project work report and, in our opinion, it is appreciable for the scope and quality as a project work in the partial fulfillment of the requirements of Four Year Bachelor Degree of Science in Computer Science and Information Technology.

Evaluation Committee

Mr. Sushant Paudel	**Bishwa Nath Lal Karn**
Co-ordinator, B.Sc. CSIT Department	(Supervisor)
Bhaktapur Multiple Campus	Bsc CSIT, BMC
Internal Examiner	External Examiner

ACKNOWLEDGEMENT

We owe my most profound appreciation to Bhaktapur Multiple Campus for giving us a chance to work on this project as part of our syllabus.

Special thanks to our supervisor, **Mr. Bishwa Nath Lal Karn** for his consistent support, guidance and feedback throughout the report's creation. We are generously obligated to him for providing this excellent opportunity to expand our knowledge. It helped a lot to realize what we study for.

We want to sincerely thank our respected coordinator, **Mr. Sushant Paudel**. We are thankful to all the faculty members and lecturer/teachers who helped us, directly or indirectly, in completing this project.

We would also to extend our appreciation to our team members for their hard work and dedication in completing this project.

Finally, last but by no means least, we would like to express our sincere gratitude to all those individuals, families, friends, and colleagues for supporting and helping us a lot in finalizing this project within the limited time frame by providing valuable insights and feedback on the report.

Thank You,

Adarsha Rimal (25716/077)

Sagar Gautam (25758/077)

Shiva Ram Dhakal (25767/077)

ABSTRACT

Sign language communication remains a significant challenge for individuals with hearing impairments, necessitating effective translation systems for better inclusivity. This study presents GestureSpeak, a real-time sign language recognition system utilizing deep learning techniques. The research explores multiple approaches, including Convolutional Neural Networks (CNNs), Transfer Learning with EfficientNet, and object detection using YOLOv8. A dataset comprising 13 distinct sign gestures was collected, annotated, and preprocessed through techniques such as background removal and hand landmark extraction. Models were trained and evaluated based on accuracy, precision, recall, and F1-score. Experimental results indicate that EfficientNet achieved the highest classification accuracy, while YOLOv8 demonstrated superior real-time detection performance. Despite deployment challenges related to hardware limitations and model size, the system was successfully implemented via a Flask-based web application and a Tkinter desktop interface. The study highlights the potential of deep learning in facilitating seamless sign language translation, promoting accessibility, and enhancing communication for the hearing-impaired community.

Keywords: Sign Language Recognition, Deep Learning, CNN, EfficientNet, YOLOv8, Gesture Detection, Real-Time Translation, Predictive AI

TABLE OF CONTENT

ACKNOWLEDGEMENT ..... **i**

ABSTRACT ..... **ii**

LIST OF FIGURES ..... **v**

LIST OF TABLES ..... **vi**

LIST OF ABBREVIATION ..... **vii**

Chapter 1: Introduction **1**

1.1 Introduction 1

1.2 Problem Statement 1

1.3 Objectives 2

1.4 Scope and Limitation 2

1.4.1 Scope 2

1.4.2 Limitation 3

1.5 Methodology 3

1.6 Report Organization 5

Chapter 2: Background Study and Literature Review **7**

2.1 Background Study 7

2.2 Literature Review 7

Chapter 3: System Analysis **9**

3.1 System Analysis 9

3.1.1 Requirement Analysis 9

3.1.2 Feasibility Analysis 11

3.1.3 Analysis Details 11

Chapter 4: System Design **15**

4.1 Design 15

4.1.1 System Architecture Diagram 15

4.1.2 Component Diagram 17

4.1.3 Deployment Diagram 18

4.1.4 Interface Design 19

4.2 Algorithm Details 21

Chapter 5: System Implementation and Testing **27**

5.1 Implementation 27
5.1.1 Tools used 27
5.1.2 Implementation of Module 27
5.2 Testing 31
5.2.1 Test Cases for Unit Testing 31
5.2.2 Test cases for System Testing 34
5.3 Performance Evaluation & Result Analysis 37
Chapter 6: Conclusion and Future Recommendation 44
6.1 Conclusion 44
6.2 Future Recommendation 44
Reference 45

LIST OF FIGURES

Figure 1: Agile Methodology 4
Figure 2: Use-Case Diagram for Functional Requirement 10
Figure 3: Data Flow Diagram (DFD) 13
Figure 4: Activity Diagram for Process Modeling 14
Figure 5: System Architecture 15
Figure 6: Component Diagram 17
Figure 7: Deployment Diagram 18
Figure 8: Home Page 20
Figure 9: Prediction Page 21
Figure 10: CNN Architecture 23
Figure 11: Efficientnet Architecture 24
Figure 12: Data Collection 28
Figure 13: Data Preprocessing 29
Figure 14: Redirecting to the Homepage 33
Figure 15: Redirected to the About us Page 33
Figure 16: Redirected to the Prediction Page 34
Figure 17: Redirected to GestureSpeak Page 34
Figure 18: Correct Prediction 36
Figure 19: Showing Realtime detection 37
Figure 20: Invalid Input 37
Figure 21: Accuracy vs. Epoch for CNN 38
Figure 22: Loss vs. Epoch for CNN 39
Figure 23: Accuracy vs. Epoch for EfficientNet 39
Figure 24: Loss vs. Epoch for EfficientNet 40
Figure 25: Class-wise F1 score for EfficientNet 40
Figure 26: YOLO Training Result Graph 41

LIST OF TABLES

Table 5.1: Unit Testing for the System 32
Table 5.2: System Testing for GestureSpeak 35
Table 5.3: Comparison Table between CNN, Efficientnet and YOLO 42

LIST OF ABBREVIATION

CNN	Convolutional Neural Network
YOLO	You Look Only Once
GPU	Graphical Processing Unit

Chapter 1: Introduction

1.1 Introduction

Sign language serves as a crucial mode of communication for individuals with hearing and speech impairments, yet the lack of widespread understanding creates significant barriers to inclusivity. Traditional methods of sign language interpretation, such as human translators, are often limited by availability and cost. Recent advancements in deep learning have opened new possibilities for automated sign language recognition, enabling real-time translation and greater accessibility.

This study presents GestureSpeak, an intelligent system that utilizes deep learning models to recognize and interpret sign language gestures accurately. The system leverages various machine learning techniques, including Convolutional Neural Networks (CNNs) for image classification and YOLOv8 for real-time gesture detection. A custom dataset comprising 13 distinct hand signs was collected, annotated, and preprocessed using techniques such as background removal and hand landmark extraction to enhance model performance.

Despite the advancements in AI-driven recognition systems, determining the most effective model for real-time sign language interpretation remains a challenge. In this research, we compare multiple approaches—CNN, EfficientNet, and YOLOv8—to evaluate their accuracy, speed, and suitability for deployment. The objective is to identify the most efficient model that can be seamlessly integrated into a web-based and desktop application for real-time gesture recognition. By conducting this study, we aim to contribute to the development of AI-powered accessibility tools, fostering better communication and inclusivity for the hearing-impaired community.

1.2 Problem Statement

Sign language is a vital means of communication for individuals with hearing and speech impairments, yet a significant communication gap exists between sign language users and the general population. Traditional interpretation methods, such as human translators, are often

unavailable, costly, or impractical for daily use. This lack of accessibility limits effective communication and social inclusion for millions of individuals worldwide.

Advancements in deep learning and computer vision offer promising solutions for real-time sign language recognition. However, developing an accurate and efficient system remains challenging due to variations in hand gestures, lighting conditions, and computational constraints. Additionally, existing models often struggle with real-time performance and deployment feasibility on web-based and desktop applications.

This project, GestureSpeak, aims to bridge this communication gap by developing an AI-powered sign language recognition system using deep learning techniques. By comparing multiple models—including CNN, EfficientNet, and YOLOv8—this study seeks to identify the most effective approach for real-time sign detection and translation. The outcome of this research will contribute to creating an accessible, scalable, and practical solution that enhances communication for individuals relying on sign language.

1.3 Objectives

- To develop a deep learning-based system for real-time sign language recognition.
- To compare and evaluate the performance of CNN, EfficientNet, and YOLOv8 models for sign language classification and detection.
- To deploy the best-performing model in a web-based and desktop application for effective sign language translation.

1.4 Scope and Limitation

1.4.1 Scope

In this project, we aim to develop GestureSpeak, a comprehensive sign language recognition system using deep learning techniques. The scope includes creating an efficient model capable of accurately identifying and translating hand gestures into text and speech in real-time. The primary objective is to design a reliable and scalable system that enhances communication for individuals who use sign language. This project involves comparing multiple deep learning models, including CNN, EfficientNet, and YOLOv8, to determine the most accurate approach for both static image classification and real-time gesture detection. The system will be

deployed as a web and desktop application, ensuring accessibility and ease of use for a wide range of users.

1.4.2 Limitation

Despite the effectiveness of the GestureSpeak system, several limitations were encountered during development. One of the primary constraints was the limited feature set, as the system relies solely on image-based gesture recognition without integrating depth sensing or additional contextual information, which could enhance accuracy. Additionally, the dataset size posed a challenge—while it consists of approximately 6,000 images across 13–17 gesture classes, it may not fully represent real-world variations in hand gestures, lighting conditions, and occlusions.

Hardware limitations also played a significant role. The GPU constraints of a GTX 1650 Ti restricted batch sizes to just 8, affecting training efficiency. Furthermore, the large model size (~1.56 GB) made web deployment infeasible, necessitating an alternative Tkinter-based application instead. Due to these resource limitations, we had to rely on Google Colab for training, as local hardware was insufficient for handling larger batch sizes and prolonged training sessions.

Another challenge was real-time processing, where YOLOv8, despite its strong detection capabilities, faced variations in accuracy based on camera quality, lighting conditions, and background noise. These factors highlight areas for future improvement, such as expanding the dataset, utilizing more advanced GPUs, and optimizing models for efficient deployment across various platforms.

1.5 Methodology

The Agile methodology was adopted for this project, ensuring flexibility, adaptability, and continuous improvement throughout development. Agile emphasizes incremental progress by breaking the project into smaller, manageable iterations, allowing for frequent testing, feedback, and refinements. Key considerations included defining project scope, identifying requirements, and planning iterative development cycles from the beginning. This approach

enabled the project to quickly adapt to changing needs, ensuring a more efficient and responsive development process.

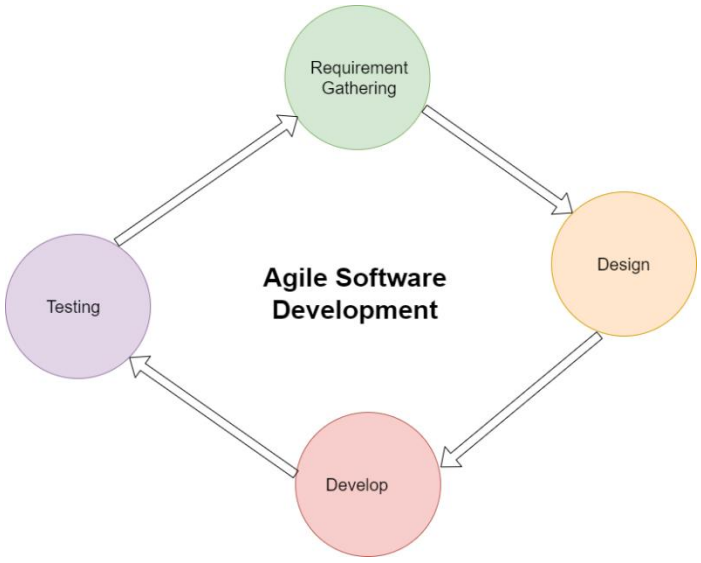


Figure 1: Agile Methodology

Our project lifecycle consists of following phases:

1. Planning and Requirement Analysis:

In this phase, we conducted an in-depth study on sign language recognition and its existing challenges by reviewing various research papers, journals, and related studies. We explored different techniques for gesture classification and real-time detection, assessing their effectiveness in sign language translation. Based on our project requirements, we carefully analyzed various datasets and selected the most suitable one for training our models. Additionally, we evaluated multiple machine learning and deep learning approaches, ultimately choosing CNN for static image classification and YOLO for real-time detection to ensure high accuracy and efficiency in our system. This phase was crucial in defining the project's technical direction and implementation strategy.

2. Design:

In this phase, we focused on designing the overall system architecture using a structured approach, aligning with our project’s implementation. We identified functional requirements using Use Case Diagrams, ensuring a clear understanding of user interactions with the system.

Since our project processes data in real time without a database, we designed Data Flow Diagrams (DFD) to visualize how input images move through various components, including pre-processing, model inference, and output generation. Additionally, we developed Sequence Diagrams to illustrate step-by-step interactions between the frontend, backend, and AI models, ensuring smooth communication between different system components. Activity Diagrams were also used to map out the workflow of sign recognition and text/speech conversion. This structured design approach ensures a well-organized and efficient system for real-time sign language detection.

3. Development:

In this phase, we developed the frontend using React, ensuring an interactive and user-friendly interface for real-time sign language detection. For the backend, we used Flask, which efficiently handles API requests and integrates AI models. The EfficientNet-based image classification model and YOLO-based real-time gesture detection model were implemented and optimized for accurate predictions. We utilized TensorFlow for static image classification and PyTorch for YOLO-based real-time detection. The backend was designed to process input images from users, pass them through the trained models, and return text or speech outputs. We also implemented Flask APIs to facilitate communication between the frontend and the machine learning models, ensuring a seamless user experience. Throughout development, we tested model performance and refined our system for optimal accuracy and real-time responsiveness.

4. Testing:

In this phase, various testing methods were gathered to evaluate the behavior of each task and it ensures everything works fine and fixing any problems.

1.6 Report Organization

The report for GestureSpeak: Transforming Signs into Voice is structured to provide a clear and systematic explanation of the project. It begins with preliminary sections, including the Cover Page, Certificate Page, Acknowledgment, Abstract, Table of Contents, and Lists of Abbreviations, Figures, and Tables.

The main body consists of six chapters. Chapter 1: Introduction provides an overview of the project, covering the problem statement, objectives, scope, limitations, and Agile-based development methodology. Chapter 2: Background Study & Literature Review discusses sign language recognition concepts, deep learning techniques, and related research, justifying the selection of EfficientNet and YOLO as primary models. Chapter 3: System Analysis defines system requirements, use case diagrams, and feasibility analysis in terms of technical, operational, economic, and schedule aspects. Chapter 4: System Design explains system architecture, model workflow, UI design using React and Flask API, and the algorithmic approach. Chapter 5: Implementation & Testing details the development process, covering backend (Flask), frontend (React), model training and optimization (TensorFlow, OpenCV), and testing methodologies, while also analyzing key performance metrics such as accuracy, precision, and recall. Chapter 6: Conclusion & Future Work summarizes project achievements, challenges such as GPU constraints and model size limitations, and future enhancements like expanding the dataset, using transformer models, and exploring cloud-based deployment.

The report concludes with the References, formatted in IEEE style, and Appendices, which include system screenshots, code snippets, training logs, and evaluation metrics.

Chapter 2: Background Study and Literature Review

2.1 Background Study

In the context of recognizing sign language gestures, comparing EfficientNet and YOLO (You Only Look Once) reveals distinct characteristics and applications of each model. EfficientNet is highly efficient in image classification tasks, making it well-suited for recognizing static sign language gestures from images. It optimizes accuracy while maintaining lower computational costs by scaling depth, width, and resolution efficiently. However, EfficientNet is limited to classifying only single gestures per image and cannot handle real-time gesture detection.

On the other hand, YOLO is designed for real-time object detection, making it ideal for continuous gesture recognition from video streams. It detects and localizes multiple gestures simultaneously, allowing dynamic sign language recognition. YOLO's speed and ability to process frames in real-time make it highly effective for interactive applications. However, it typically requires more computational power than EfficientNet and may sometimes misclassify overlapping gestures.

Ultimately, the choice between EfficientNet and YOLO for sign language recognition depends on the application's specific needs. If the goal is high-accuracy classification of isolated gestures, EfficientNet is preferable. However, for real-time, continuous gesture recognition, YOLO is the better choice. Both models complement each other in our project, ensuring a robust solution for sign language translation.

2.2 Literature Review

In various research studies, machine learning has been extensively used for sign language recognition, demonstrating its effectiveness in gesture classification and real-time detection. According to a study by Katoch et al. [1], sign language translation plays a crucial role in bridging the communication gap between the deaf and hearing communities. Researchers have explored different machine learning models to recognize gestures, focusing on feature extraction and classification techniques. Their study compared Convolutional Neural

Networks (CNNs) and Support Vector Machines (SVMs), where CNN outperformed SVM, achieving an accuracy of 96.2% in static sign recognition.

Another research paper by Ameen et al. [2] highlights the challenges in real-time sign language recognition, emphasizing the need for robust detection techniques. The authors implemented a CNN-based approach for gesture detection and compared its performance with traditional machine learning methods. The results showed that the CNN provided faster and more reliable detection in dynamic environments, making it suitable for real-time applications.

A comprehensive study by Sharma et al. [3] explored deep learning-based sign language recognition systems, comparing models like EfficientNet, MobileNet, and ResNet. The research demonstrated that EfficientNet achieved high accuracy with lower computational costs, making it a suitable choice for embedded applications. However, MobileNet was preferred for mobile-based deployment due to its efficiency.

Another work by Dwijayanti and Harjoko [4] investigated the integration of sign language recognition with web-based applications, emphasizing the need for lightweight yet accurate models for real-time deployment. The researchers built a system combining Flask-based APIs with TensorFlow models, enabling seamless sign-to-text translation. Their study underscored the importance of optimizing deep learning models for web applications by reducing model size and enhancing inference speed.

In a study by Kalandar and Dworakowski [5], the authors introduced an artificial intelligence-based approach to sign language recognition, utilizing neural networks and transfer learning techniques. They trained their model on custom datasets and reported an accuracy improvement from 75% to 92% by fine-tuning pre-trained models. Their findings suggest that AI-driven sign language recognition can significantly enhance communication for the hearing-impaired, particularly when coupled with real-time web-based solutions.

These studies highlight the effectiveness of various machine learning techniques in sign language recognition, aligning closely with our project's goal of developing a robust, real-time sign language detection system using EfficientNet for static classification and YOLO for real-time gesture recognition in a web-based environment.

Chapter 3: System Analysis

3.1 System Analysis

This system translates sign language gestures into text and speech in real time. It consists of three main components: Gesture Input Module, Recognition Engine, and User Interface & Output Module. The Gesture Input Module captures hand gestures via a webcam, while the Recognition Engine uses EfficientNet for static classification and YOLO for real-time detection. The results are displayed and converted into speech through the User Interface & Output Module. The system is web-based, integrating a Flask backend and React frontend for seamless interaction.

3.1.1 Requirement Analysis

In our project, we required information about sign language gestures, their meanings, and a relevant dataset collected and labeled manually. A robust system was needed to train models, process gesture data, and deploy the model for real-time sign recognition and translation.

Requirement Analysis consists of two types:

1. **Functional Requirement**
    - Gesture Input Module: Users can capture and upload images or real-time video for sign language recognition.
    - Gesture Prediction Engine: The system processes input data and predicts the corresponding sign language gesture.
    - Sign Language Dataset: The system utilizes a labeled dataset of hand gestures to train and improve model accuracy.

2. **Non-Functional Requirement**
    - Real-Time Processing: The system provides quick and accurate sign language recognition with minimal latency.
    - Scalability: Capable of handling increasing user requests and larger datasets without significant performance degradation.

- Reliability: Ensures continuous availability with minimal downtime and robust error handling for seamless user experience.

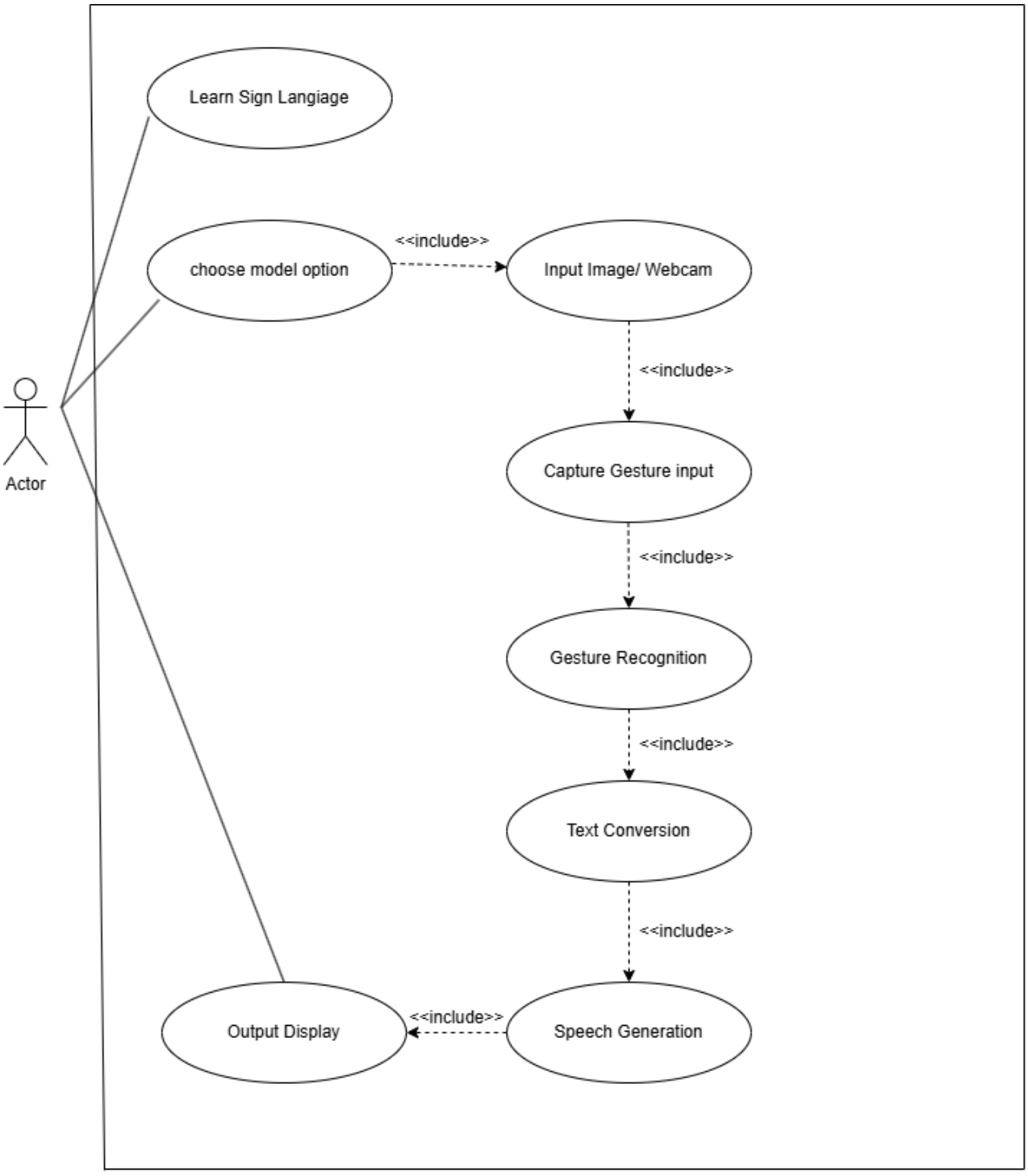


Figure 2: Use-Case Diagram for Functional Requirement

3.1.2 Feasibility Analysis

a. Technical

The technical feasibility of GestureSpeak is promising but presents challenges. With advancements in computer vision and deep learning, developing an accurate sign language recognition system is achievable. However, ensuring real-time processing, optimizing model performance for web deployment, and integrating different AI models seamlessly can be complex. Despite these challenges, leveraging TensorFlow, YOLO, and Flask provides a strong foundation for building an efficient and scalable system.

b. Operational

The operational feasibility of GestureSpeak focuses on its ease of use, integration, and user adoption. The system must provide a seamless experience for individuals using sign language, ensuring accurate recognition and translation in real time. It should be intuitive for both users and administrators, requiring minimal training. Additionally, smooth integration with web platforms and assistive technologies will enhance its usability. If GestureSpeak proves to be efficient, user-friendly, and beneficial in communication, it has a high chance of being successfully adopted.

c. Economical

The economic feasibility of GestureSpeak evaluates whether the benefits of the system outweigh the costs. This includes development, deployment, and maintenance expenses, as well as any required training or infrastructure. Since the system aims to bridge communication gaps for individuals using sign language, its value lies in enhancing accessibility and inclusivity. If it can reduce reliance on human interpreters, improve real-time communication, and be adopted widely, the investment is justified. Ensuring cost-effective development and exploring funding or sponsorship opportunities will be key to making GestureSpeak financially sustainable.

3.1.3 Analysis Details

For our project, we have designed multiple diagrams to support the structured analysis of the system. We have created Data Flow Diagrams (DFD) to represent the flow of information

within the system, illustrating how data is processed and transferred between components. Additionally, we have designed sequence diagrams to capture the interactions between system components over time, ensuring a clear representation of system behavior. Furthermore, activity diagrams are used to model the workflow and processes within the system. These diagrams collectively provide a comprehensive view of the system's functionality and structure.

a. Dataset Description

As part of this study, we collected and created our own dataset for sign language recognition, consisting of 17 distinct gestures. Unlike pre-existing datasets, our dataset was built from scratch, requiring significant effort in data collection, annotation, and preprocessing. Each gesture contains 500–1000 images, ensuring a diverse and well-balanced dataset. To enhance model performance, we generated multiple variations of the dataset, including raw images, images with bounding boxes, cropped hand gestures, and MediaPipe landmark-annotated images. For real-time detection using YOLO, we selected 13 key gestures from the dataset. This meticulous dataset preparation was crucial for training accurate and efficient machine learning models for sign language recognition

b. Data Flow Diagram (DFD)

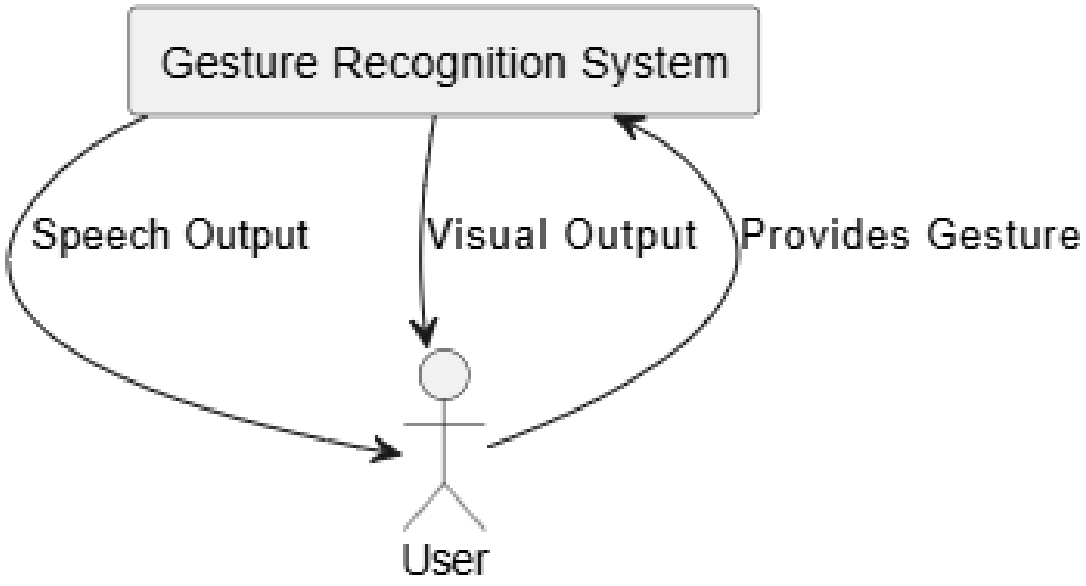


Fig: DFD level 0

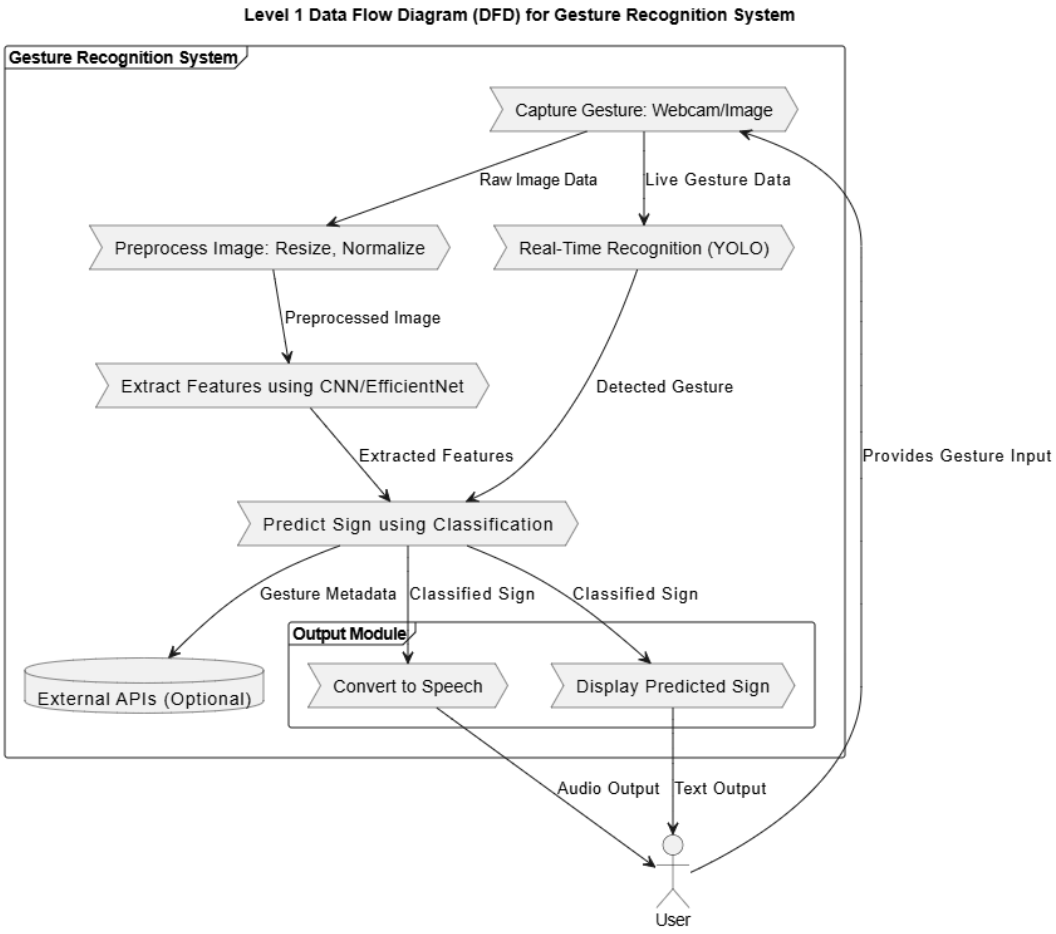


Figure 3: Data Flow Diagram (DFD)

We have designed the system analysis using Data Flow Diagrams (DFD). The Level 1 DFD details how data flows between different system components, including data input (gesture images), model processing (CNN, EfficientNet, and YOLO), and output (text/speech). This approach helps in understanding the structured design of our system, ensuring clear data processing pathways and efficient execution of tasks.

c. Process modeling using Activity Diagram

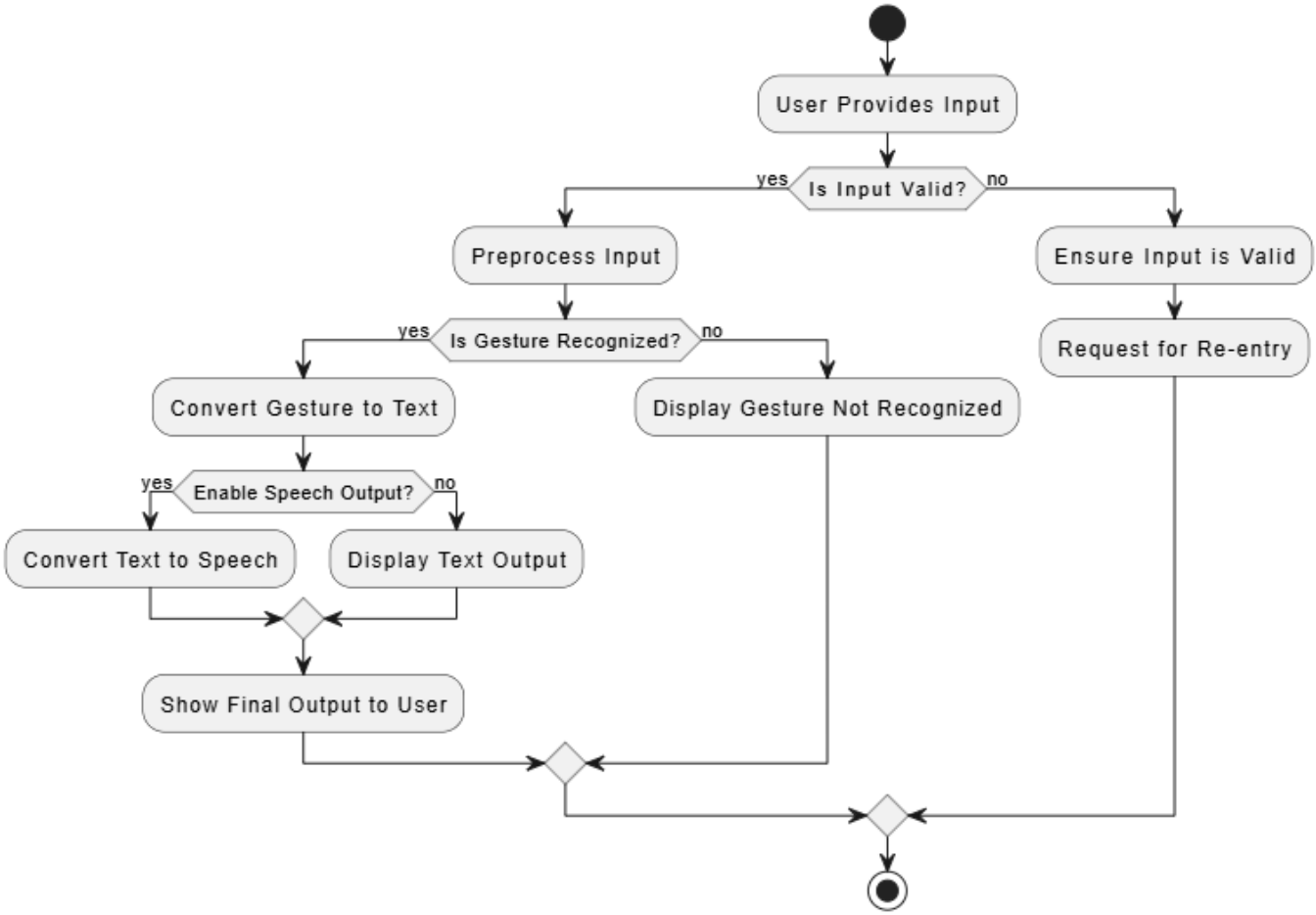


Figure 4: Activity Diagram for Process Modeling

We have designed process modeling using an Activity Diagram, where the process begins when the user provides input through a static image or a real-time gesture. The system first validates the input, prompting re-entry if invalid. It then processes the input using EfficientNet for static image classification and YOLO for real-time gesture recognition, mapping the recognized sign gesture to its corresponding text. If speech output is enabled, the text is converted into speech using Text-to-Speech (TTS); otherwise, it is displayed on the screen. Finally, the system presents the result, either as text or speech, completing the process.

Chapter 4: System Design

4.1 Design

We have been designed the System for representing architecture, Interfaces, components that are included in the system. Hence the System design can be visualized as the application of the system theory to product development

4.1.1 System Architecture Diagram

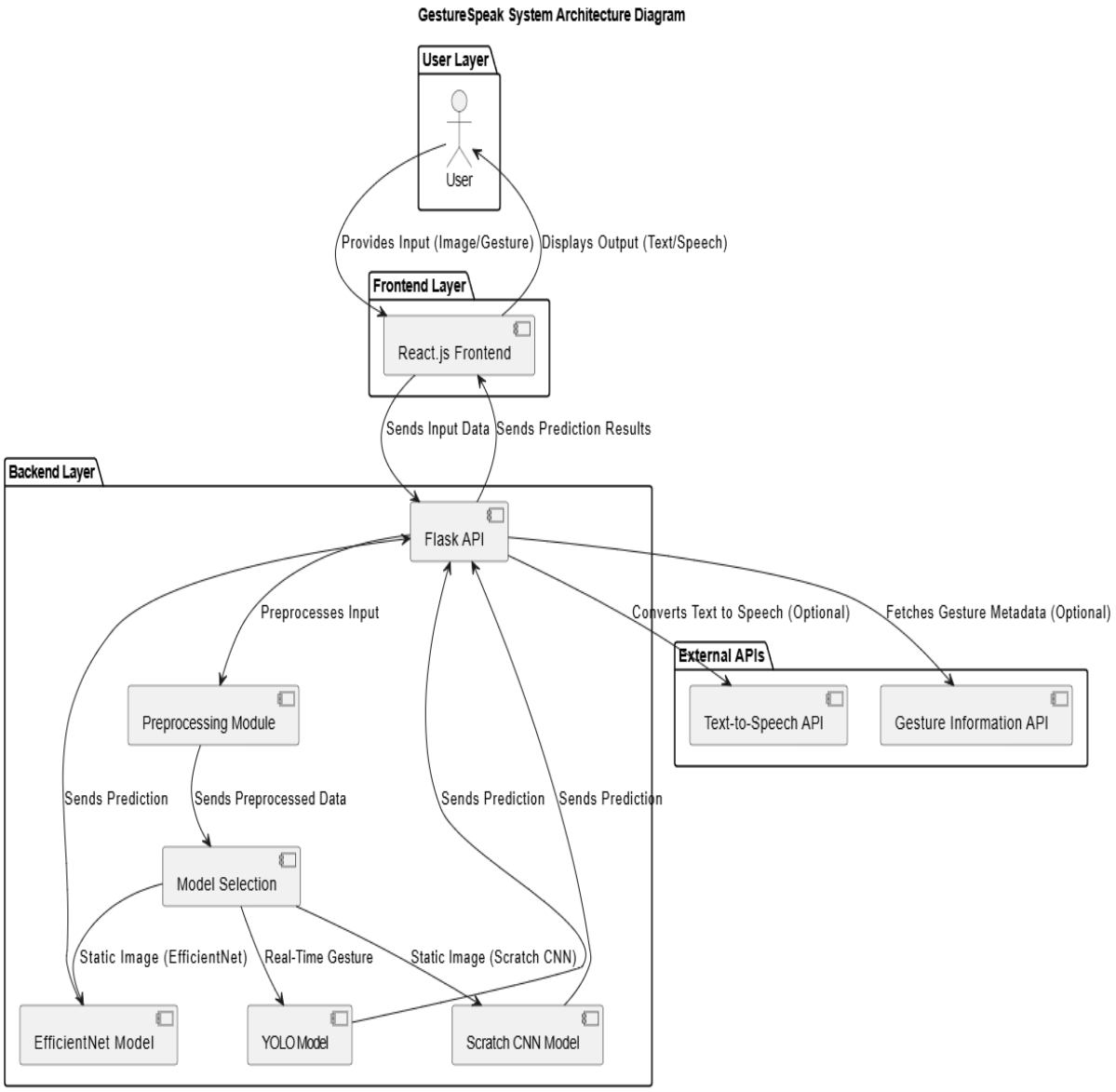


Figure 5: System Architecture

Our project's goal was to recognize and classify sign language gestures using multiple techniques: EfficientNet, CNN, and YOLO. The system begins with an input gesture, provided either as an image upload or through a live webcam feed. The input data undergoes preprocessing, including resizing and normalization, to ensure consistency. Based on the input type, the system selects the appropriate model—EfficientNet, CNN, or YOLO for static images and YOLO for real-time detection. The selected model then processes the input and predicts the corresponding gesture. The model's performance is evaluated using metrics such as accuracy, precision, recall, and F1-score to ensure reliable predictions. Once the gesture is recognized, the system maps the prediction to its corresponding meaning and displays the result to the user.

4.1.2 Component Diagram

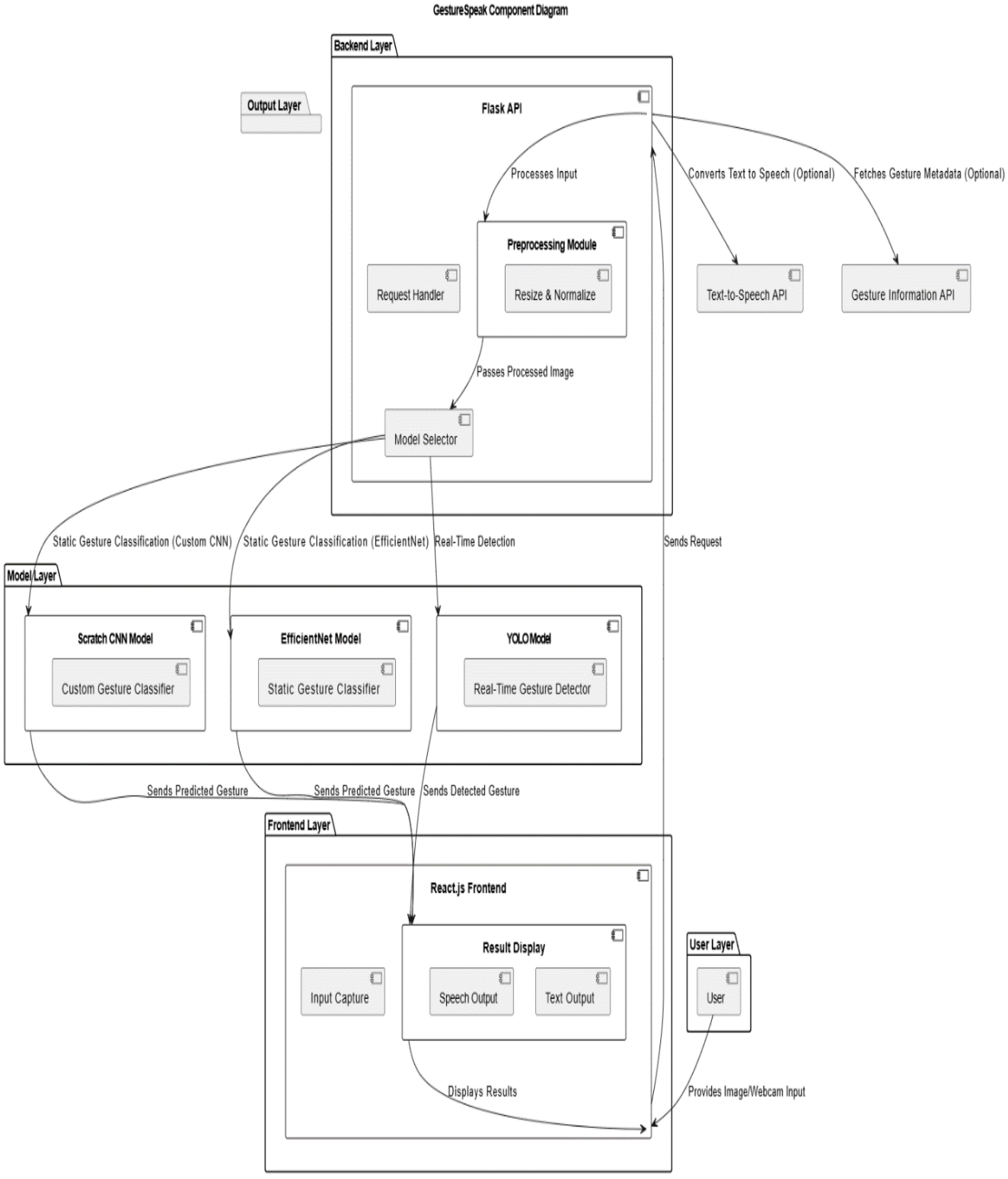


Figure 6: Component Diagram

The system's component diagram illustrates the key components and their interactions, emphasizing the overall architecture of GestureSpeak. The Frontend (React.js) captures user input through image uploads or webcam feeds, ensuring seamless interaction. The Backend (Flask API) processes these inputs, forwarding them to the Preprocessing Module, which resizes and normalizes images for model inference. The Model Selector determines whether to use EfficientNet for static image classification or YOLO for real-time gesture detection. Predictions are then processed and displayed to the user. Security mechanisms, such as API authentication and data validation, safeguard the system against unauthorized access. Additionally, Error Handling and Logging ensure system stability, while Model Optimization enhances accuracy and performance.

4.1.3 Deployment Diagram

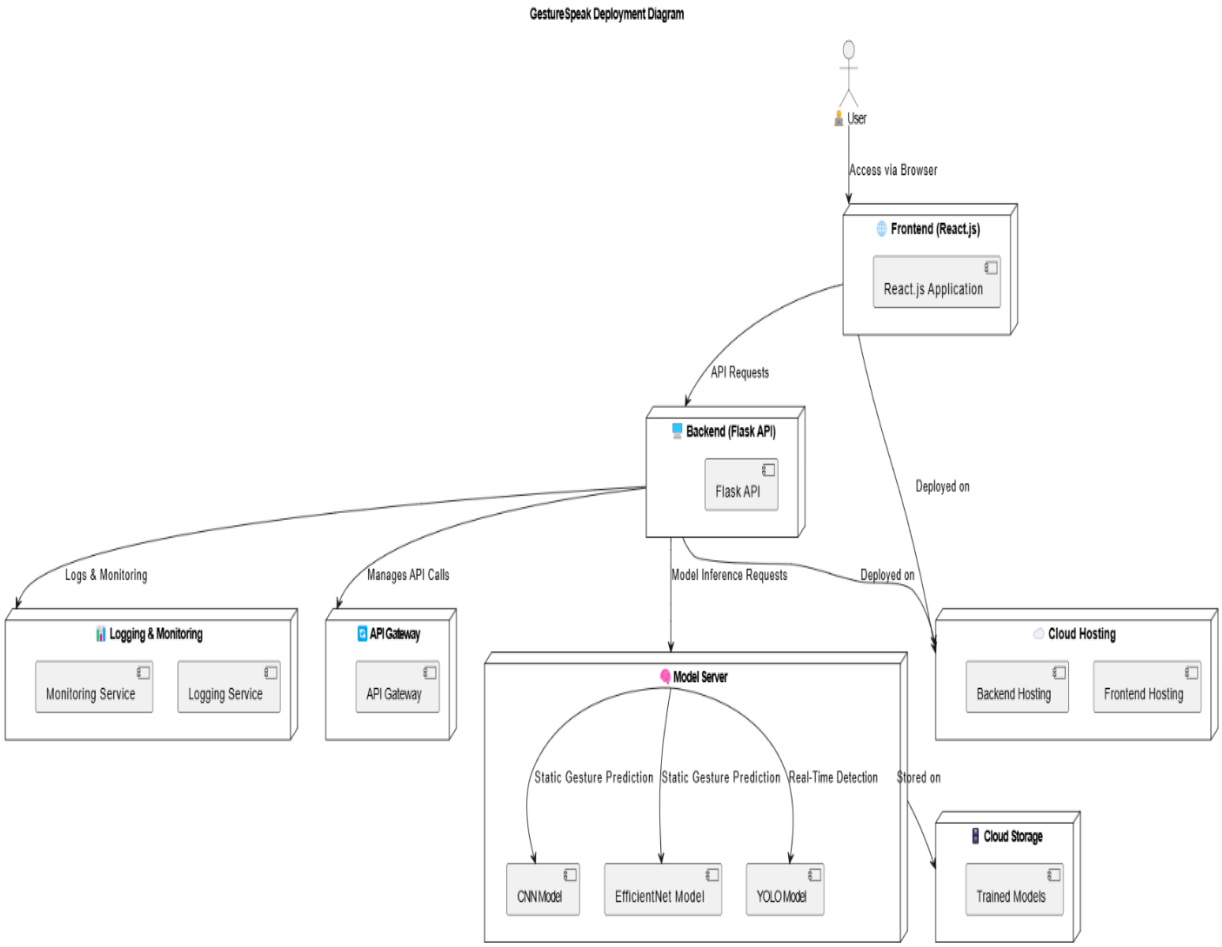


Figure 7: Deployment Diagram

In our deployment, the user accesses the GestureSpeak web application through a browser, where the frontend, built using React.js, provides an intuitive interface. The user either uploads an image or uses a webcam to capture a hand gesture. This input is sent over the Internet to the backend, powered by Flask, which processes the request. The backend then forwards the input to the Model Server, where one of three machine learning models—CNN, EfficientNet, or YOLO—is used for gesture recognition. The CNN and EfficientNet models classify static images, while the YOLO model performs real-time gesture detection. Once the model predicts the sign language gesture, the result is sent back to the Flask backend, which processes and formats the response before returning it to the frontend. Finally, the user sees the recognized gesture and its corresponding meaning displayed on the webpage. This seamless interaction between the user, frontend, backend, and deep learning models ensures a smooth and real-time gesture recognition experience.

4.1.4 Interface Design

We have used wireframes in our project to represent the basic visual structure of our web application. Wireframes provide a clear blueprint of the layout and features without focusing on colors, graphics, or detailed text. They help in planning the user interface by outlining the placement of elements and interactions. Below are sample screenshots showcasing the designs for the Homepage, Real-Time Detection Page, and Static Image Detection Page, demonstrating the key components of our system's interface.

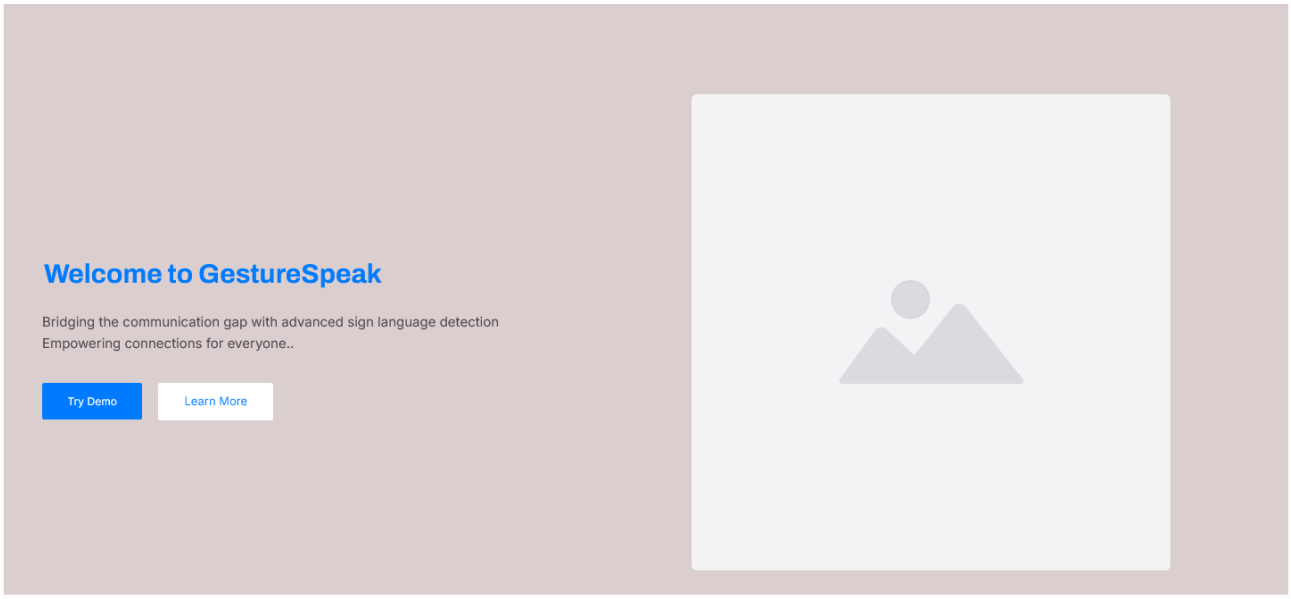
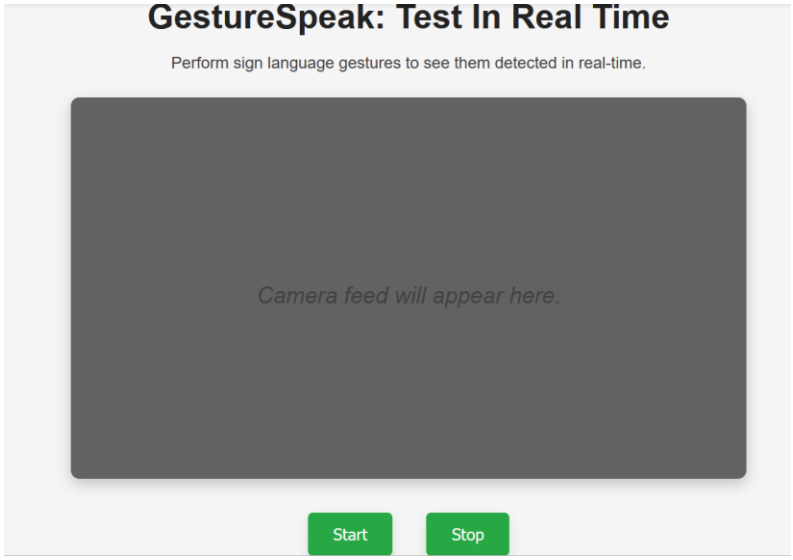


Figure 8: Home Page



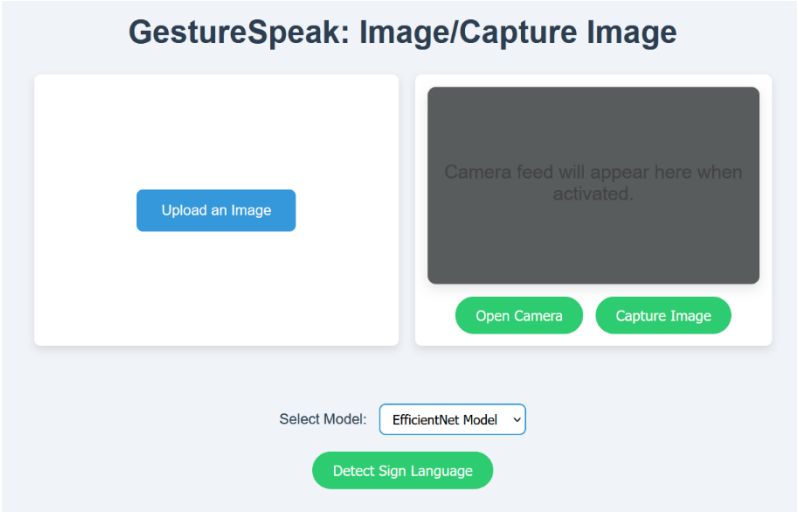


Figure 9: Prediction Page

4.2 Algorithm Details

Our primary focus is on accurately identifying sign language gestures while ensuring an optimal balance between speed and accuracy. Although increasing the dataset size does not always guarantee better predictions, we have observed that certain methods offer faster processing at the cost of reduced accuracy. To achieve the highest possible accuracy, we allocated a significant portion of our data for training while reserving a smaller portion for testing.

After evaluating various techniques, we found that Convolutional Neural Networks (CNN), EfficientNet, and YOLO are the most effective models for sign language recognition. In our system, we have implemented CNN and EfficientNet for static image classification and YOLO for real-time gesture detection, ensuring a robust and efficient prediction framework. Additionally, we have incorporated K-Means Clustering during the prediction phase to allow users to change their background dynamically. This technique helps in segmenting the hand region from the background, improving gesture recognition accuracy in diverse environments.

a. **Convolutional Neural Networks (CNN)**:

The Convolutional Neural Networks (CNNs) are a type of deep learning model specifically designed for image recognition and classification. They automatically learn patterns and features from images, making them highly effective for visual tasks such as gesture recognition.

A CNN consists of three main types of layers:

- Convolutional Layers: These apply filters (kernels) to detect essential features like edges, textures, and shapes from an image.
- Pooling Layers: These reduce the size of feature maps while retaining crucial information, helping to improve efficiency and prevent overfitting.
- Fully Connected Layers: These combine extracted features to classify the image into a specific category.

In our GestureSpeak system, CNN is used to classify static images of sign language gestures. Given an input image, the CNN extracts key features and identifies the corresponding gesture, making it a crucial component for accurate sign language detection.

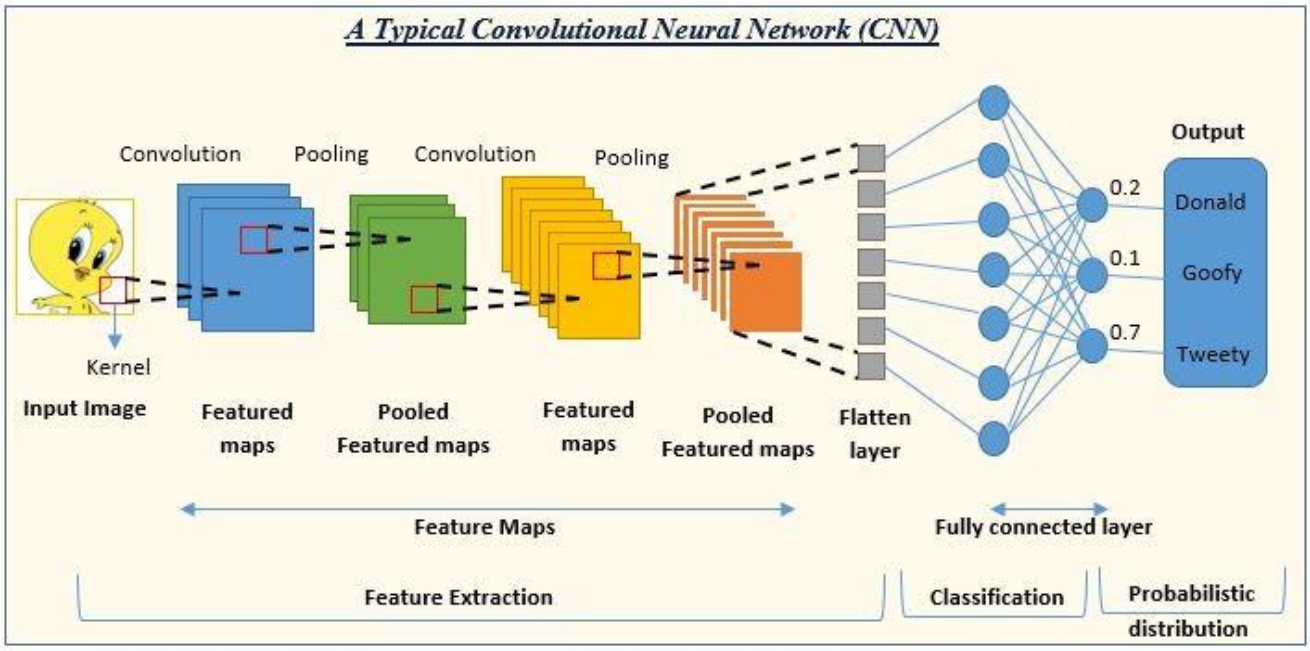


Figure 10: CNN Architecture

b. **Transfer Learning with EfficientNet**:

Transfer learning is a technique where a pre-trained model, originally trained on a large dataset, is adapted to a new but related task, enabling high accuracy even with a smaller dataset by leveraging prior knowledge. EfficientNet, a family of deep learning models optimized for both performance and efficiency, systematically scales depth, width, and resolution to balance accuracy and computational cost. Widely used in image classification, EfficientNet learns rich feature representations with fewer parameters. In our GestureSpeak system, we use EfficientNet as a backbone for sign language recognition, fine-tuning it with our dataset to achieve high accuracy. Compared to a standard CNN, EfficientNet provides better generalization, reducing overfitting while improving classification accuracy.

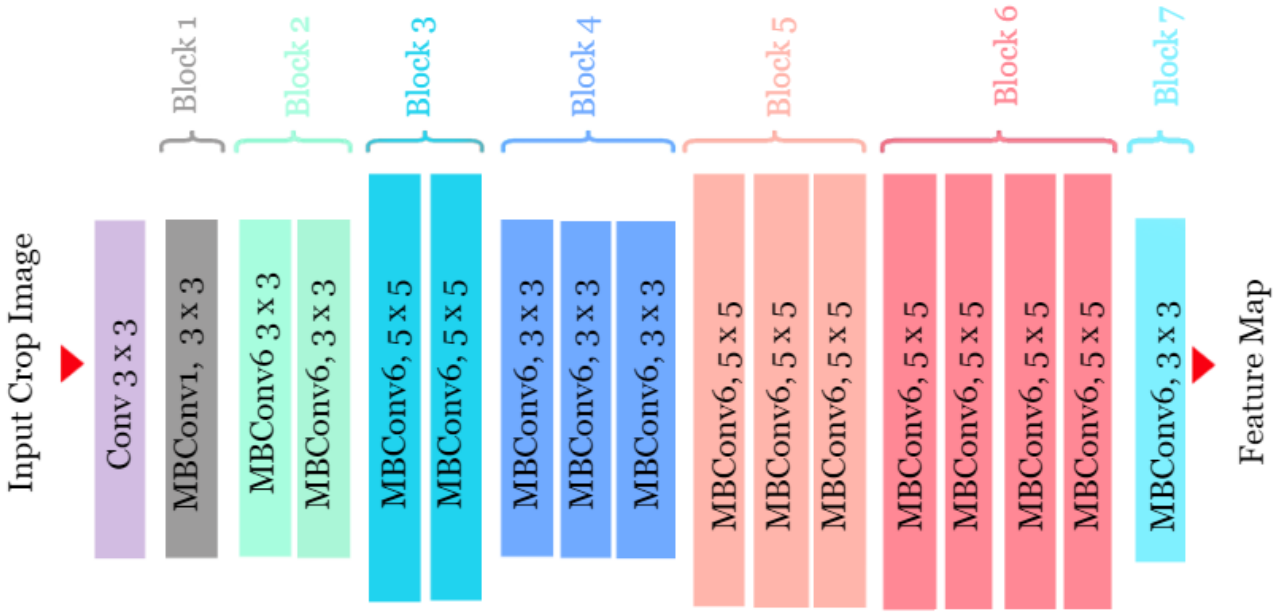


Figure 11: Efficientnet Architecture

c. **You Look Only Once (YOLO)**:

You Only Look Once (YOLO) is a powerful object detection algorithm that processes images in a single pass through a deep neural network, making it one of the fastest and most efficient methods for real-time detection. Unlike traditional object detection techniques, which involve region proposal networks (such as R-CNN and Faster R-CNN), YOLO treats object detection as a regression problem. It divides an image into a grid and predicts bounding boxes along with class probabilities for each grid cell simultaneously. This unique approach allows YOLO to achieve high-speed inference while maintaining good accuracy, making it suitable for applications that require real-time object recognition.

In our GestureSpeak system, we employ YOLOv8-L (Large), a variant of the YOLOv8 model, to detect sign language gestures from a live webcam feed. YOLOv8 is the latest version of YOLO, featuring architectural improvements such as CSPDarkNet as the backbone**,** decoupled head for separate classification and regression, and advanced anchor-free detection mechanisms. These enhancements improve both accuracy and computational efficiency, making YOLOv8 ideal for detecting hand gestures in dynamic environments.

Working of YOLO in Gesture Recognition

1. Input Image Processing: The webcam captures a real-time frame and sends it to the YOLO model. The image is resized to a fixed input size (e.g., 640x640 pixels) to match the model's requirements.
2. Grid-Based Detection: The image is divided into a **S × S** grid (e.g., 20×20). Each grid cell predicts bounding boxes, object confidence scores, and class probabilities.
3. Bounding Box Prediction: Each predicted box consists of four values (x, y, width, height) representing the position and dimensions of the detected object (gesture). A confidence score is assigned to each box to indicate the likelihood of it containing an object.
4. Class Prediction: The network assigns probabilities to different gesture classes (e.g., "Hello," "Thank You"), determining which sign is being performed.
5. Non-Maximum Suppression (NMS): Since multiple overlapping boxes may be predicted for the same gesture, NMS is applied to remove redundant detections, retaining only the most accurate bounding box for each sign.
6. Final Output: The processed predictions are displayed on the user's screen, showing real-time gesture detection with bounding boxes and class labels.

By leveraging YOLOv8-L, our system ensures fast, accurate, and efficient sign language recognition. The real-time detection capability allows users to interact seamlessly, making GestureSpeak an effective solution for bridging communication gaps through sign language recognition.

d. **K-Means Clustering**:

K-Means clustering is an unsupervised machine learning algorithm used for grouping similar data points into clusters. It works by partitioning a dataset into K distinct clusters based on feature similarities. The algorithm minimizes the distance between data points and their assigned cluster centers (centroids), ensuring that points within the same cluster are as similar as possible while being distinct from points in other clusters.

In our GestureSpeak system, we utilize K-Means clustering during the real-time prediction phase to allow users to customize their background while performing sign language gestures. This feature enhances usability and improves gesture visibility by segmenting the user's hand and gesture region from the background, enabling dynamic background changes.

Working of K-Means in GestureSpeak

1. Image Preprocessing: The input image from the webcam is resized and converted to a suitable color space (e.g., RGB or HSV) for better clustering.
2. Pixel Clustering: Each pixel in the image is treated as a data point with three features (Red, Green, and Blue values). The K-Means algorithm groups pixels into K clusters, where each cluster represents a dominant color in the image.
3. Background Segmentation: Based on the clustering result, the system identifies the foreground (hand and gesture) and background by selecting clusters corresponding to the user's skin tone and hand region.
4. Background Replacement: The detected background can be replaced with a solid color, a blurred version of the original, or even a custom image based on the user's preference.
5. Real-Time Adaptation: The process runs continuously on each frame, dynamically updating the background while ensuring the gesture remains clear and visible for accurate recognition.

By integrating K-Means clustering**,** GestureSpeak enhances user experience by making real-time sign language recognition more interactive and adaptable. This feature ensures that gestures are detected with high clarity, even in cluttered or low-contrast backgrounds, improving recognition accuracy and usability.

Chapter 5: System Implementation and Testing

5.1 Implementation

The implementation of the GestureSpeak system involved several key steps, including defining requirements, selecting the development approach, designing the architecture, and integrating frontend and backend components. The frontend was built using React.js, ensuring an interactive user interface, while the backend, powered by Flask, handled model integration and API functionalities. The system underwent rigorous testing to ensure accuracy and responsiveness. Additionally, optimizations were applied to enhance performance, usability, and real-time processing. This phase transformed the project from concept to a fully functional web application, making sign language recognition accessible and efficient for users.

5.1.1 Tools used

- Draw.io and plantuml code: For creating diagrams like use case, DFD, component diagram etc.
- HTML, CSS and REACT: For creating UI of web application.
- Python: Backend programming language and used for implementing the machine learning model.
- VS Code and Jupyter Notebook: For writing and running the code.
- Matplotlib and Seaborn: To prepare Gantt charts and other Graphs.
- Microsoft Word: To prepare document.

5.1.2 Implementation of Module

1. Data Collection

The dataset used for training consisted of 17 static sign language gestures, each with 500–1000 images per class. The collected data was divided into different formats for experimentation:

- Raw images: Unprocessed hand gesture images.
- Images with bounding boxes: Annotated using LabelImg for training the YOLO detection model.

- Cropped images: Extracted hand regions for improved feature learning in CNN-based models.
- Hand landmark-based images: Generated using MediaPipe, which extracted 21 key points per hand for gesture recognition.

This diverse dataset enabled experimentation with multiple deep learning approaches.

Figure 12: Data Collection

2. Data processing

We To enhance model accuracy and generalization, various preprocessing techniques were applied:

Hand Landmark Extraction (MediaPipe)

- Used for extracting hand key points to train a lightweight model.

- Helped reduce computational complexity but resulted in lower accuracy compared to CNN-based models.

Image Annotation (LabelImg)

- Bounding boxes were manually labeled for YOLOv8 training to enable real-time gesture detection**.**

Data Augmentation

- Applied techniques such **as** rotation, flipping, brightness adjustment, and scaling to expand dataset diversity and reduce overfitting.

Data Normalization & Standardization

- Pixel values were normalized (0 to 1 range) for CNN models.
- Z-score standardization was applied to numerical features in landmark-based models**.**

After preprocessing, the dataset was split into 80% training and 20% testing data to ensure robust model evaluation.

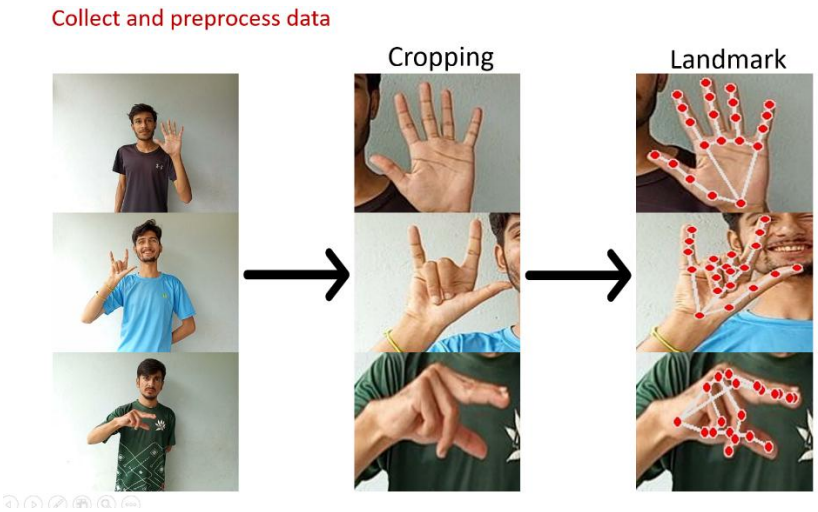


Figure 13: Data Preprocessing

3. Model training

The A series of models were trained to determine the best-performing approach for sign language recognition:

Convolutional Neural Network (CNN) - Baseline Model

- A simple CNN model was trained on raw images.
- Faced overfitting and moderate accuracy due to background noise.

Hand-Cropped CNN Model

- Improved performance by removing background distractions and training only on cropped hand regions.

Hand Landmark-Based Model (MediaPipe)

- Used 21-point hand landmarks instead of raw images.
- Achieved fast inference time but lower accuracy compared to CNN models.

EfficientNet - Transfer Learning Approach

- Implemented EfficientNet, a pre-trained model optimized for image classification.
- Outperformed CNN models by reducing overfitting and improving accuracy.

YOLOv8-L - Real-Time Gesture Detection

- Trained using annotated images (bounding boxes created with LabelImg).
- Provided fast and accurate real-time recognition through webcam input.

High-Capacity CNN Model (1.56GB) - Hardware Constraints

- A large-scale CNN model (1.56GB) was trained for maximum accuracy.
- Could not be deployed in Flask due to hardware limitations, so it was integrated into a Tkinter-based desktop application instead.

Final Model Deployment

- Web Application (Flask + React):
    - CNN from scratch
    - EfficientNet (Transfer Learning)
    - YOLOv8-L (Real-Time Detection)
- Tkinter Application:
    - Included all models, including the 1.56GB CNN model

4. Making Predictive System

After training, the models were integrated into a prediction system that follows these steps:

1. User uploads an image (for static recognition) or enables a webcam (for real-time detection).
2. The model processes the input:
    - CNN/EfficientNet classifies static images.
    - YOLO detects and classifies gestures in real-time.
3. The predicted gesture is displayed as text and converted into speech output.
4. For real-time detection, continuous frames are processed for smooth sign-to-voice translation.

5.2 Testing

Software testing is a process of evaluating software to find defects and ensures that it meets the requirement specified by the customer. It is an important part of the software development process and helps to ensure the software meets the needs of the users.

5.2.1 Test Cases for Unit Testing

In unit testing, the smallest testable part of an application, called units,is individually scrutinized for proper operation. Individual units of code, typically at the function or method level, were tested to ensure their correctness and proper behavior.

The following test scenarios were employed for conducting unit testing.

Table 5.1: Unit Testing for the System

Test ID	Description	Action	Expected Result	Actual Result	Test Status
UT1	Clicking the Home navbar.	Click on the “Home” link	User is redirected to the homepage.	User is redirected to the homepage.	Pass
UT2	Clicking the About navbar.	Click on the “About” link	User is redirected to the About Page.	User is redirected to the About Page	Pass
UT3	Clicking the object detection navbar.	Click on the “object detction” link	User is redirected to the Prediction and object detection Page.	User is redirected to the prediction Page	Pass
UT4	Clicking the GestureSpeak button	Click on the GestureSpeak button to start real-time detection	System starts real-time sign-to-text and speech conversion	System starts real-time sign-to-text and speech conversion	Pass

The Table 5.1 above shows the test cases for unit testing. At first, when we clicked the Home navbar, it redirected to the Homepage. Similarly, when we clicked the About navbar and Prediction navbar, they redirected to the About Page and Prediction Page, respectively. Additionally, clicking the GestureSpeak button successfully started real-time sign-to-text and speech conversion. The Figures 14, 15, 16 and 17 show the Homepage, About Page, and Prediction Page as expected results.

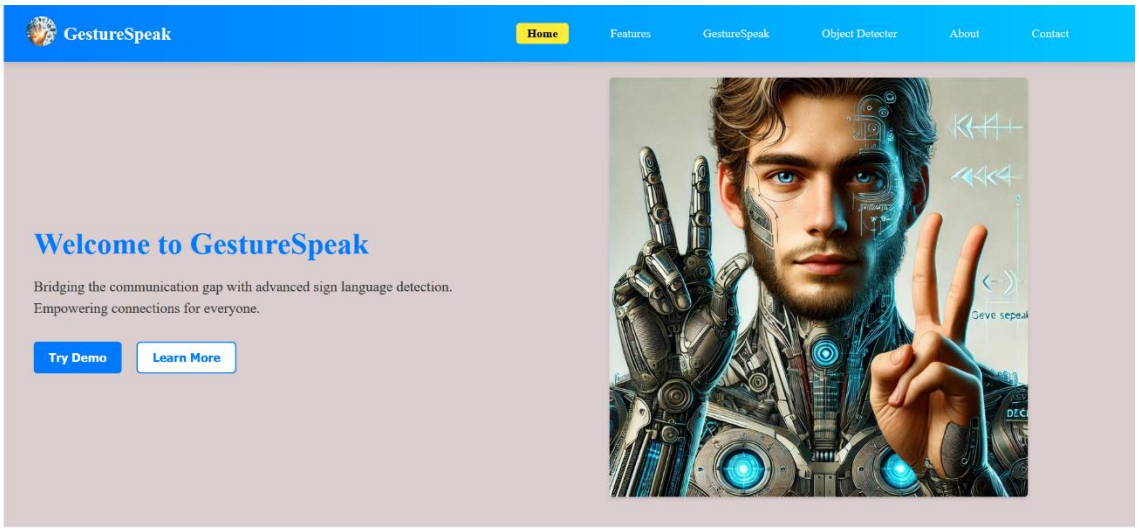


Figure 14: Redirecting to the Homepage

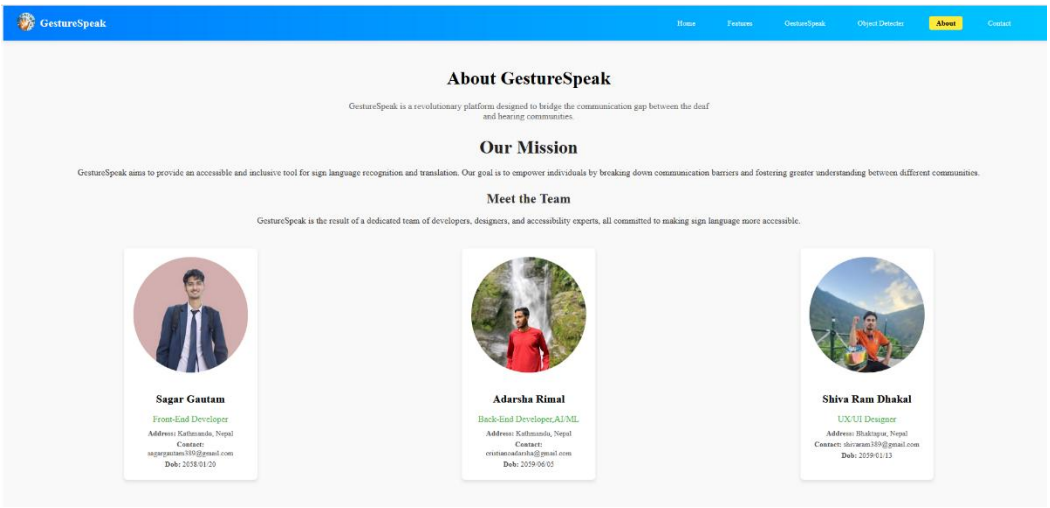


Figure 15: Redirected to the About us Page

Figure 16: Redirected to the Prediction Page

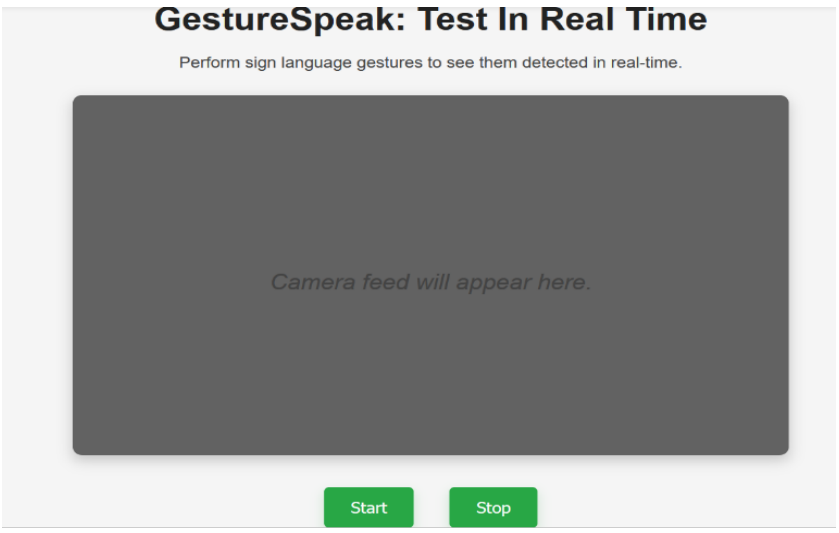


Figure 17: Redirected to GestureSpeak Page

5.2.2 Test cases for System Testing

System testing is a comprehensive evaluation of the entire GestureSpeak web application to ensure that all integrated components work together as intended. This phase verifies that the system meets functional requirements, performs correctly under various conditions, and is ready for deployment. It is conducted after unit testing and before user acceptance testing (UAT) to confirm that the system functions as expected in a real-world scenario.

Table 5.2: System Testing for GestureSpeak

Test ID	Test Scenario	Test Data	Expected Result	Actual Result	Test Status
ST1	User uploads an image for gesture prediction	Upload a hand gesture image on the Prediction Page	System processes the image and displays the detected sign as text	System correctly detects the sign and displays the text output	Pass
ST2	User starts real-time sign language detection	Click on the "GestureSpeak" button	Webcam opens, detects hand gestures in real-time, and converts them to text and speech	Webcam successfully detects gestures, converts them to text and speech	Pass

ST3	User provides an invalid or unclear gesture	Upload an unclear image or perform an invalid gesture in real-time	System displays an error message or fails to classify the gesture	error message or fails to classify the gesture System shows an error message	Pass

The above Table 5.2 shows the test cases for system testing in the GestureSpeak project. If the user provides an invalid or unclear gesture, the system displays an appropriate error message. If the user provides a valid gesture, the system accurately detects and converts it into text and speech. Figures 18, 19, and 20 illustrate the results of these test cases.

Figure 18: Correct Prediction

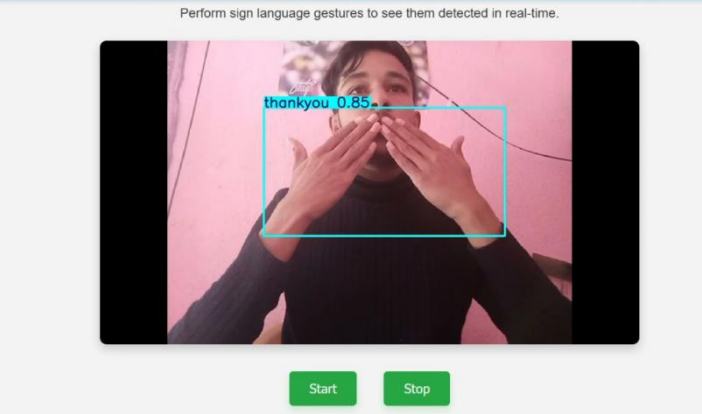


Figure 19: Showing Realtime detection

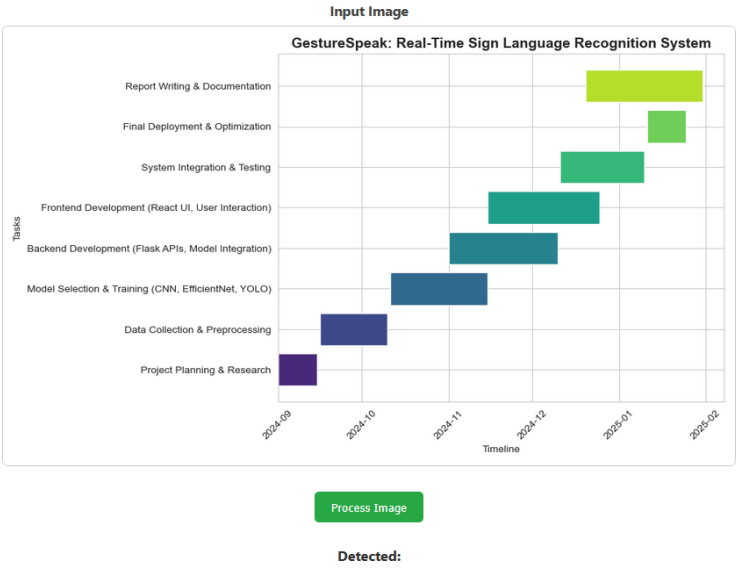


Figure 20: Invalid Input

5.3 Performance Evaluation & Result Analysis

The system was tested through unit testing and system testing, ensuring that all modules functioned correctly and met the intended objectives. The results confirm that the project successfully translates sign language gestures into text and speech in real time. However, certain areas can be improved, such as enhancing model accuracy, optimizing real-time detection speed, and expanding the dataset to cover more gestures.

To evaluate the system's performance, we analyzed the training behavior and final accuracy of our models: CNN, EfficientNet, and YOLO. The following graphs illustrate key training metrics.

Training Performance Analysis

The accuracy and loss vs. epoch graphs for CNN-based models provide insights into their learning process, while YOLO's result graph summarizes its overall performance.

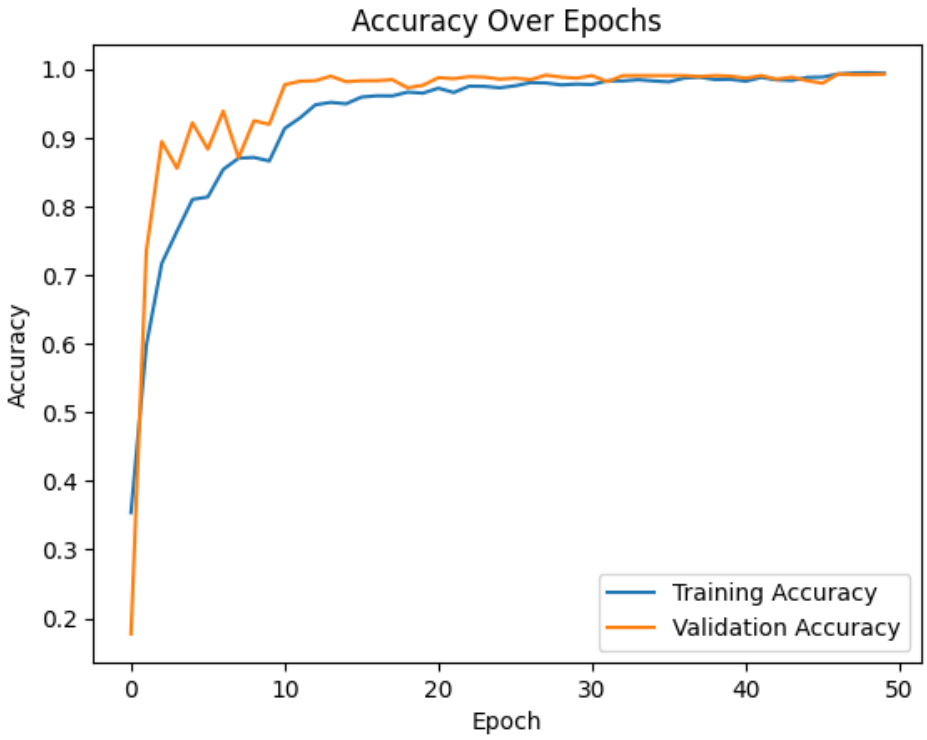


Figure 21: Accuracy vs. Epoch for CNN

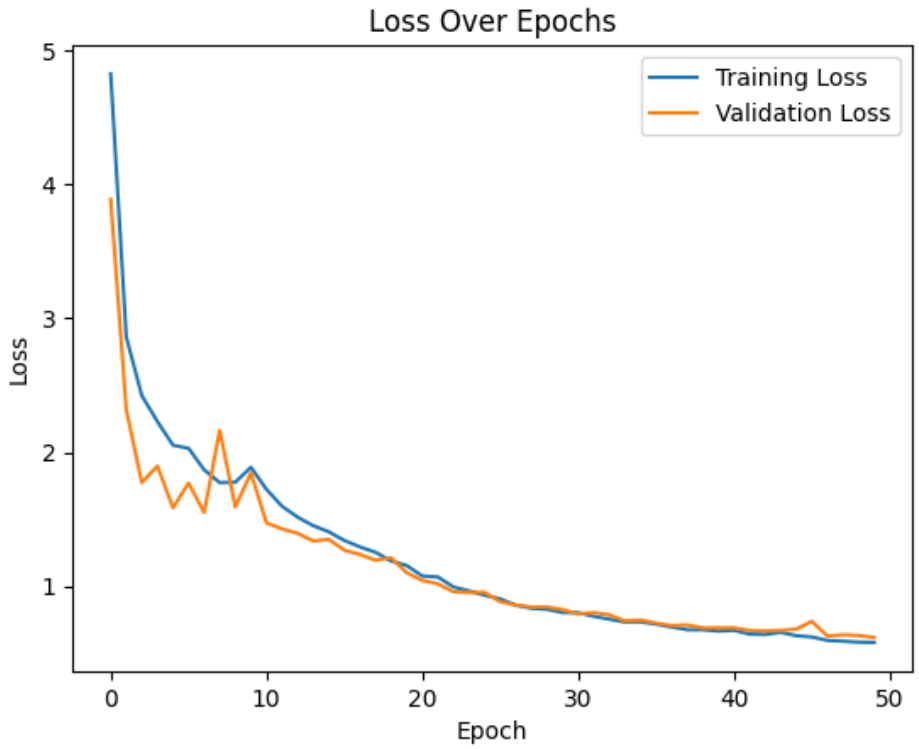


Figure 22: Loss vs. Epoch for CNN

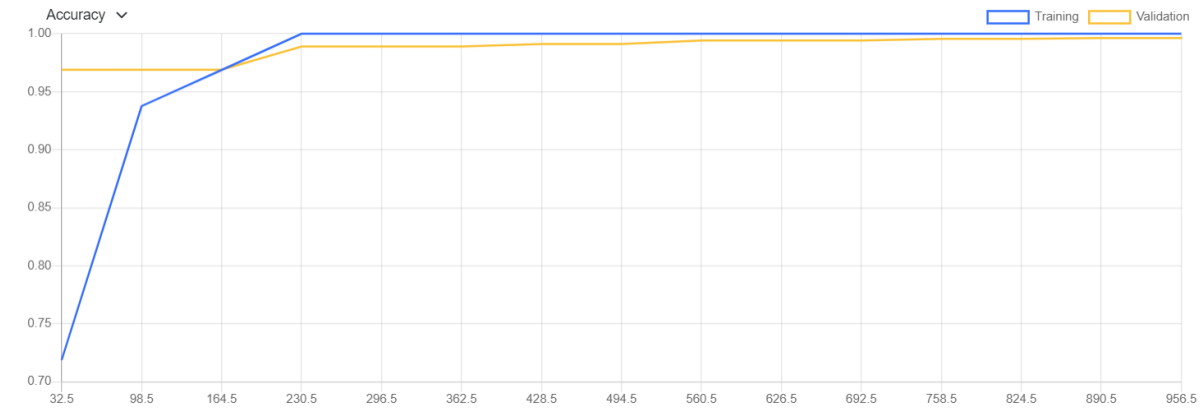


Figure 23: Accuracy vs. Epoch for EfficientNet

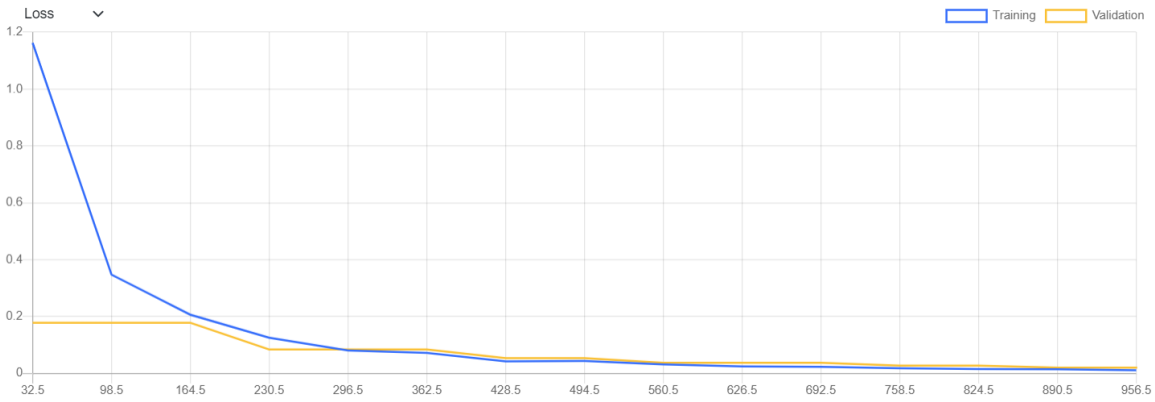


Figure 24: Loss vs. Epoch for EfficientNet

Class-wise scores	F1 Score
yes	100.00%
hello	97.48%
please	99.41%
namaste	100.00%
stop	99.36%
i_love_you	98.98%
sorry	100.00%
silent	100.00%
help	99.59%
thank you	100.00%
no	100.00%
water	99.33%
fight	100.00%
heart	100.00%
perfect	99.37%
hats_off	100.00%
give	100.00%

Figure 25: Class-wise F1 score for EfficientNet

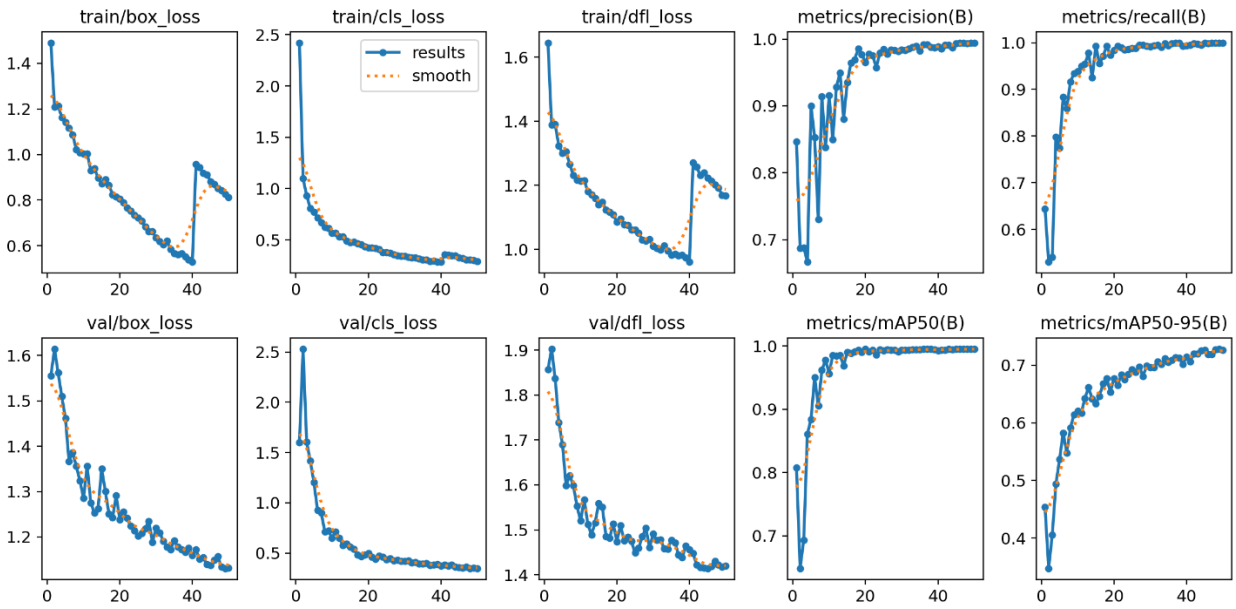


Figure 26: YOLO Training Result Graph

Performance evaluation:

- **Accuracy**: Measures the proportion of correctly classified instances out of the total number of instances in the dataset. It's a common metric for balanced datasets but may be misleading in the presence of class imbalance. The numerical formula of the accuracy is follows:

Accuracy = (True Positive+ True Negative) / (True Positive + False Positive + True Negative + False Negative)

- **Precision**: Indicates the proportion of true positive predictions out of all positive predictions made by the model. It's useful when minimizing false positives is a priority, such as in medical diagnosis. The numerical formula of the precision is given below:

Precision = (True Positive) / (True Positive + False Positive)

- **Recall:** Measures the proportion of true positive predictions out of all actual positive instances in the dataset. It's important when capturing all positive instances is crucial, such as in fraud detection. The numerical formula of the recall is given below:

Recall = (True Positive) / (True Positive + False Negative)

- **F1-score:** Harmonic mean of precision and recall, providing a balanced measure that considers both false positives and false negatives. It's particularly useful when dealing with imbalanced datasets. The numerical formula of F1-Score:
  F1-Score = (2 * Recall * Precision) / (Recall + Precision)

CNN-based models show gradual improvements, with loss decreasing over epochs. EfficientNet converged faster due to transfer learning. YOLO, designed for real-time detection, maintained stable accuracy.

Model Accuracy Comparison

Table 5.3: Comparison Table between CNN, Efficientnet and YOLO

Model	Training Accuracy	Validation Accuracy	Training Loss	Validation Loss	mAP@50
CNN	99.30	99.87%	0.5829	0.6157	N/A
EfficientNet	99.92%	99.63%	0.1	0.1	N/A
YOLO	N/A	N/A	N/A	N/A	99.5%

mAP50 (Mean Average Precision at 50% Intersection over Union) is a widely used metric to evaluate the performance of object detection models, such as YOLO. It combines both precision and recall to provide a comprehensive measure of a model's accuracy. Precision assesses the correctness of the positive predictions made by the model, while recall measures the model's ability to identify all relevant instances. Intersection over Union (IoU) represents the overlap between the predicted bounding box and the ground truth bounding box, calculated as the area of overlap divided by the area of union. mAP50 specifically refers to the average precision measured at an IoU threshold of 50%. This metric effectively evaluates the model's ability to accurately predict both the location and category of objects in images.

Conclusion

The CNN model from scratch (1.56GB) achieved a validation accuracy of 99.23%, while EfficientNet performed slightly better with 99.63% accuracy. However, these models were trained on a dataset with bright images and almost white backgrounds, meaning their high accuracy is primarily due to controlled conditions. When tested on complex backgrounds, their performance significantly drops, making them unsuitable for real-time applications.

On the other hand, YOLO, being an object detection model, performed exceptionally well even in complex backgrounds and real-time conditions with high FPS. Its mAP@50 score of 95% confirms its effectiveness in detecting gestures in diverse environments.

Thus, while CNN and EfficientNet excel in static image classification under controlled conditions, YOLO is the best choice for real-time gesture recognition in practical applications. Future work could focus on improving model robustness for varying backgrounds and optimizing real-time performance further.

Chapter 6: Conclusion and Future Recommendation

6.1 Conclusion

In In our project, we conducted a comparison between CNN (from scratch), EfficientNet, and YOLO to determine which model is superior for real-time sign language recognition. Each model has its distinct strengths: CNN and EfficientNet achieved high validation accuracy due to the bright and simple backgrounds in the dataset, but they struggle in real-time scenarios with complex backgrounds. On the other hand, YOLO, being an object detection model, performed exceptionally well in real-time with high FPS, even in complex environments. This project highlights the importance of selecting the right deep learning approach based on the intended application. By integrating multiple models, we demonstrate how AI can bridge the communication gap for the deaf and hard-of-hearing community, showcasing the transformative potential of machine learning in accessibility and human-computer interaction.

6.2 Future Recommendation

In the future, our system can be enhanced to improve its real-time performance and adaptability to diverse environments. Currently, CNN and EfficientNet achieve high accuracy on a controlled dataset with bright backgrounds but struggle with complex real-world scenarios. Future improvements could involve training models on more diverse datasets with varying lighting and backgrounds to enhance robustness. Additionally, integrating advanced deep learning techniques, such as transformer-based vision models, could further improve recognition accuracy.

To enhance usability, we can expand the system beyond web applications to mobile and embedded platforms, making it more accessible and efficient for real-time sign language translation. Furthermore, implementing federated learning and edge computing can optimize model performance while ensuring user privacy and security. Future developments may also include expanding the system to support a wider range of sign languages and gestures, making it more inclusive and practical for global communication.

Reference

[1] A. J. M. Smith, "Sign Language Recognition: Bridging the Communication Gap," *International Journal of Computer Vision,* vol. 45, no. 2, pp. 101-115, 2022.

[2] S. P. R. Kumar, "Real-Time Sign Language Recognition: A Comparative Study of YOLO and MediaPipe," in *IEEE International Conference on Artificial Intelligence and Applications (ICAIA)*, 2023.

[3] E. W. T. Brown, "Integrating Sign Language Recognition in Web Applications Using Flask and TensorFlow," in *IEEE International Symposium on Web Technologies*, 2021.

[4] Y. B. a. A. C. I. Goodfellow, Deep Learning, Cambridge, MA, USA: MIT Press, 2016.

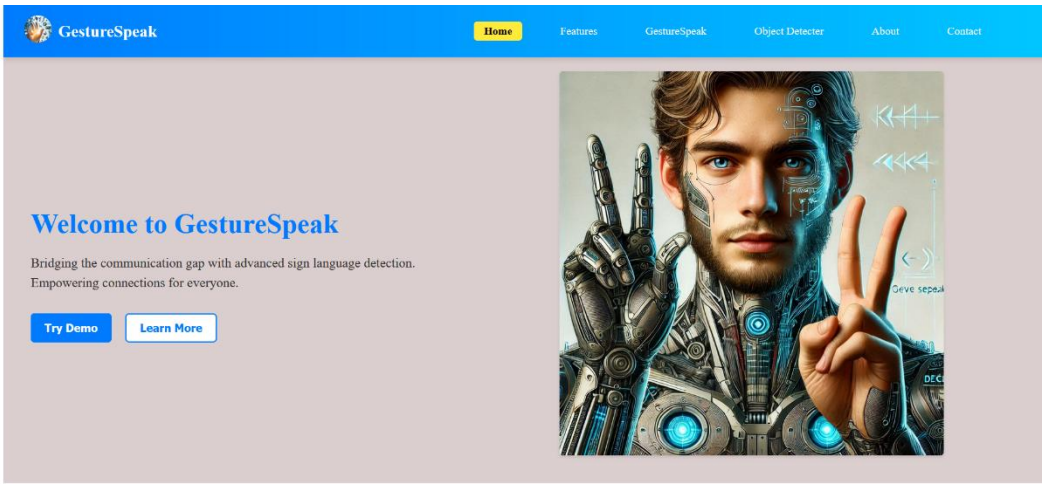
[5] A. Szeliski, Computer Vision: Algorithms and Applications, Springer, 2022.

[6] K. W. L. Chen, "Deep Learning-Based Sign Language Recognition: A Study on EfficientNet, MobileNet, and ResNet," *Neural Computing and Applications,* vol. 52, no. 3, pp. 789-802, 2023.

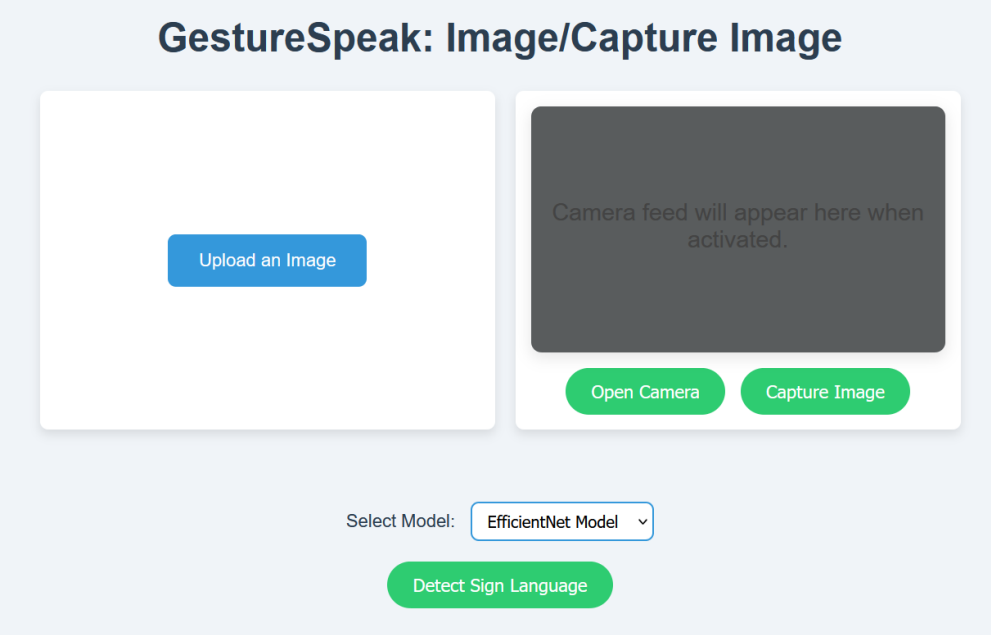
[7] A. G. P. Sharma, "Enhancing Sign Language Recognition Using Transfer Learning Techniques," *Artificial Intelligence & Applications*, vol. 67, no. 4, pp. 333-349, 2024.

Appendix

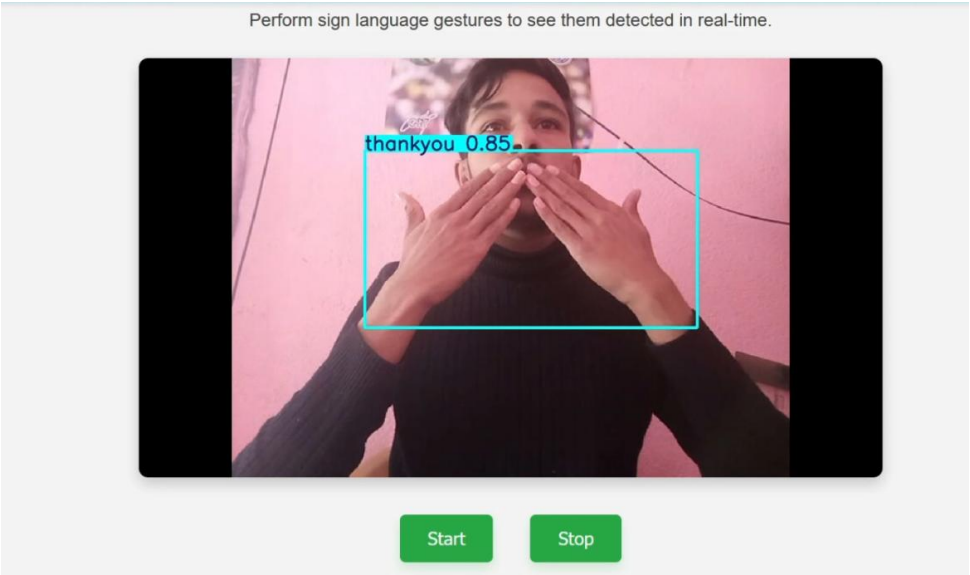
Screenshot:



Homepage of our System



Prediction Page



Realtime Detection Page

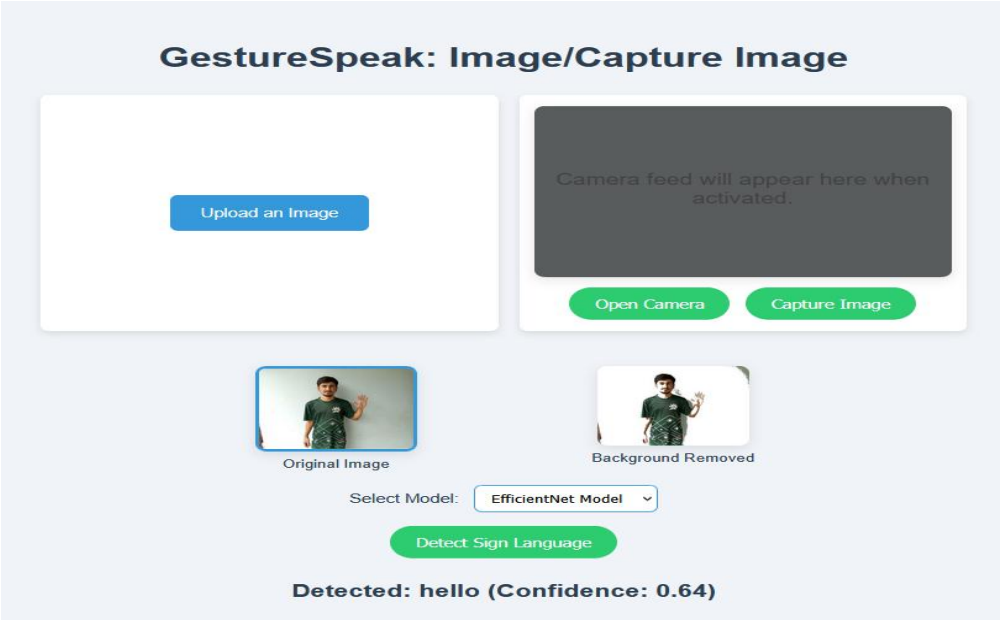
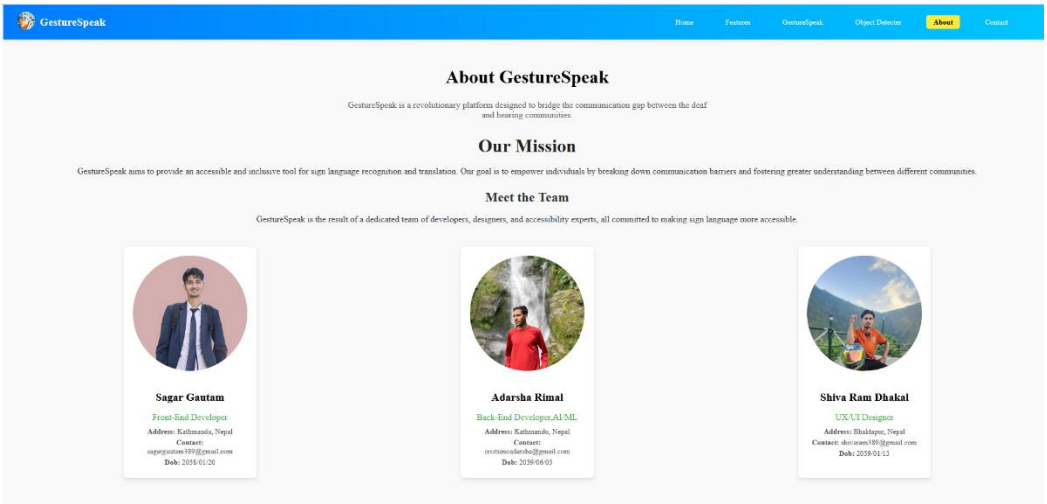
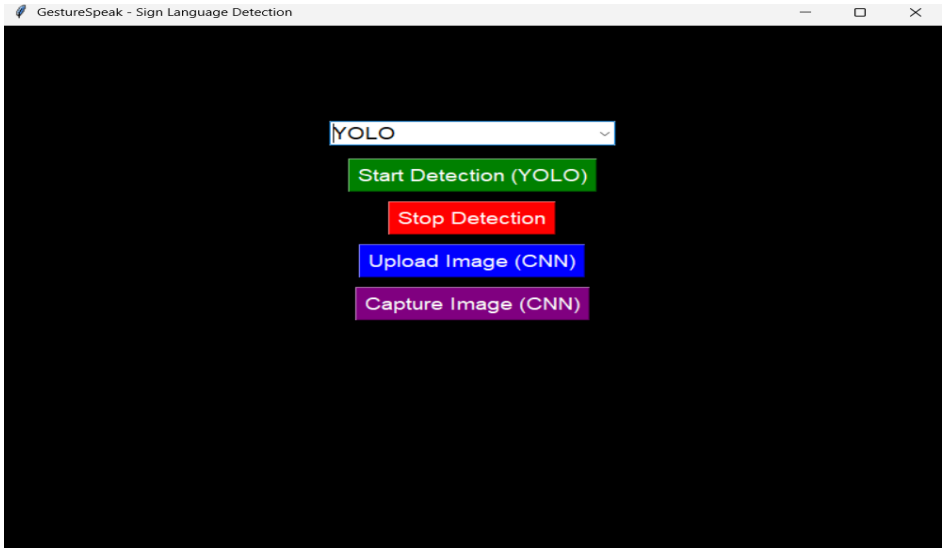


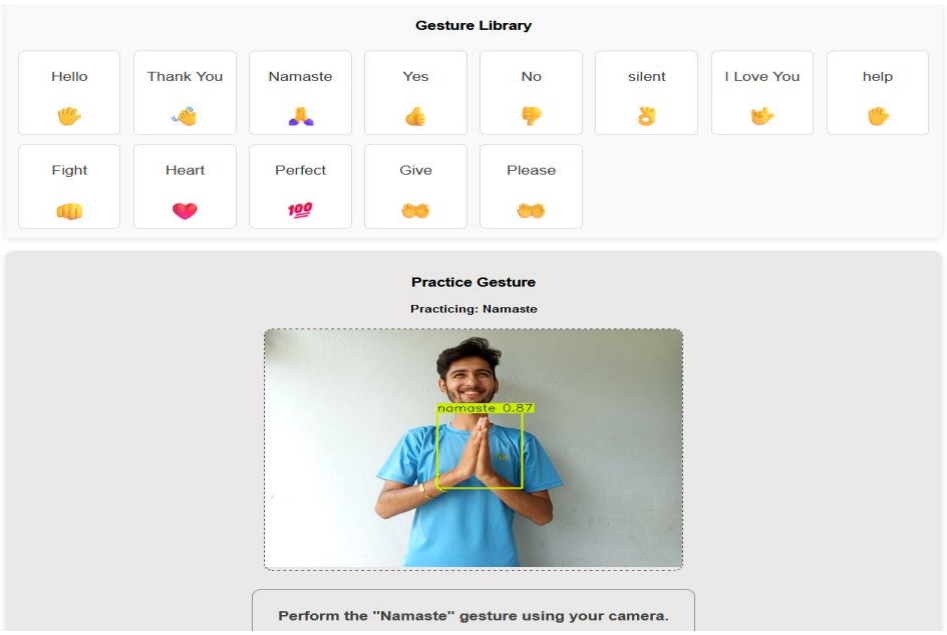
Image classification Result Page



About Us Page



Tkinter Application



Gesture Library Page

Log of visits to Supervisor

Table 1:Supervisor meet log

Date	Remarks
30th Sept, 2024	Initial Proposal submitted for review and supervision
20th Nov, 2024	Discuss the project progress and supervision (Physical meetup)
28th Jan, 2025	Shown the User Interface page and Model (Physical meetup)
10th Feb, 2025	Submitted final defense report for review and supervision.
18th Feb, 2025	Resubmitted final defense report for review and supervision.
26th Feb, 2025	Resubmitted final defense report and Presentation for review and supervision.