SIGN LANGUAGE INTERPRETER

Submitted by

Madhav Khatoria(RA2011033010131)
Seenu Nahak(RA2011033010111)

Under the Guidance of

Dr.G.SENTHIL KUMAR

Associate Professor, Department of Computational Intelligence

In partial satisfaction of the requirements for the degree of

BACHELORS OF TECHNOLOGY
in
COMPUTER SCIENCE ENGINEERING

with specialization in Software Engineering

SCHOOL OF COMPUTING

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR – 603203 NOVEMBER 2023

SRM INSTITUTION OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR-603203

BONAFIDE CERTIFICATE

Certified that 18CSP107L minor project report **"SIGN LANGUAGE INTERPRETER"** is the bonafide work done by **Madhav Khatoria [RA2011033010131], Seenu Nahak [RA2011033010111]**who carried out under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other work.

SIGNATURE	**HEAD OF THE DEPARTMENT**
Research Supervisor	**Dr. R Annie Uthra**
Dr.G.SENTHIL KUMAR	Professor and Head ,
Associate Professor	Department of Computational Intelligence,
Department of Computational Intelligence	SRM Institute of Science and Technology
SRM Institute of Science and Technology	Kattankulathur Campus, Chennai
Kattankulathur Campus, Chennai	

ABSTRACT

The real-time sign language interpreter web app revolutionizes communication by bridging the gap between sign language users and individuals unfamiliar with sign language. Leveraging advanced computer vision and natural language processing techniques, the app operates seamlessly through a user's webcam. By capturing intricate sign language gestures in real-time, it instantly converts them into precise textual captions, ensuring accurate interpretation of the user's message. This innovative technology facilitates effortless communication between sign language users and those who rely on spoken language, fostering a more inclusive and accessible environment.

Through its sophisticated algorithms, the app ensures the seamless transmission of information, breaking down barriers that often hinder effective communication. Sign language users can express themselves naturally, knowing their gestures will be accurately translated into written words. Simultaneously, individuals unfamiliar with sign language can engage in meaningful conversations without any language barriers. This transformative tool not only enhances inclusivity but also promotes understanding and empathy among diverse communities.

The real-time sign language interpreter web app stands as a testament to the power of technology in promoting universal communication, empowering individuals to connect, collaborate, and share their experiences effortlessly. Its impact reverberates across various sectors, fostering a more inclusive society where communication knows no bounds.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

LSTM	Long Short Term Memory
CNN	Convolutional Neural Network
ML	Machine Learning
DL	Deep Learning
APP	Application
IEEE	Institute of Electrical and Electronics Engineers
ReLU	Rectified Linear Units
UML	Unified Modeling Language
RNN	Recurrent Neural Network

CHAPTER 1
INTRODUCTION

1.1Problem definition

The World Health Organization (WHO) reports that more than 6% of the world's population suffers f rom hearing loss, and approximately 5%, that is, 430 million people, need hearing treatment. This nu mber is expected to reach 700 million in 2050.

Hearing loss is defined as the inability to hear normal hearing (20 decibels or more in both ears) and can cause mild to severe pain. Available for one or both ears. Deaf people often rely on sign languag e as their primary means of communication, so developing sign language is important. It is also very important for people who cannot speak.

These systems use mostly image-
based methods to interpret gestures, enabling the procedures to be translated into text in real time, foc using on Indian Sign Language.

1.2 Computer Vision

Computer vision is a subfield of artificial intelligence that tries to teach machines how to perceive and comprehend visual data in the same way that people do. It entails the creation of algorithms and procedures that allow computers to analyze, process, and comprehend images and videos.

In recent years, computer vision has made tremendous advancements due to the availability of large datasets and powerful machine learning algorithms. Some of the practical applications of computer vision include object recognition, face detection, image and video analysis, augmented reality, and autonomous navigation.

Object recognition is one of the most important applications of computer vision. It involves training algorithms to identify and classify objects in images and videos accurately. Object recognition is used in a wide range of applications, from identifying cancer cells to self-driving cars.

Another important application of computer vision is face detection, which involves identifying human faces in images and videos. This technology has been used for security purposes, such as in surveillance systems, and for improving the user experience in mobile devices.

Image and video analysis is another area where computer vision has shown significant progress. This technology enables computers to extract valuable information from images and videos, such as identifying patterns, detecting anomalies, and tracking objects over time.

Augmented reality is an exciting application of computer vision that overlays digital information on the real world, creating an immersive experience for the user. This technology has been used in gaming, advertising, and education, among others. Finally, autonomous navigation is an application of computer vision that enables machines to navigate the world without human intervention. This technology has been used in self-driving cars, drones, and robots, among others, enabling them to safely and efficiently navigate their surroundings.

In conclusion, computer vision is a rapidly growing field that has numerous practical applications. With the increasing availability of visual data and the development of powerful machine learning algorithms, we can expect to see many more exciting applications of computer vision in the years to come.

1.2.1 Types of Computer Vision Techniques

It is a vast field that involves the development of algorithms and techniques to enable machines to interpret visual data. Different types of computer vision are used to solve specific problems, and they have various applications. Here are some of the main types of computer vision:

● Image classification: This type of computer vision involves training machines to categorize images into different classes or categories. For example, a machine can be trained to identify an image as a cat, a dog, or a car. Image classification is used in various applications, including image search and content filtering.

● Object detection: Identifying and detecting items inside an image or video is a sort of computer vision. Object detection is utilized in a variety of applications, including autonomous vehicles, surveillance, and robotics.

● Semantic segmentation: This type of computer vision involves segmenting an image into different regions and assigning a label to each segment. Semantic segmentation is used in applications such as medical image analysis and autonomous navigation.

● Instance segmentation: This type of computer vision involves identifying and delineating individual objects within an image or video. Instance segmentation is used in applications such as autonomous driving and robotics.

● Optical character recognition (OCR): This type of computer vision involves recognizing and interpreting text within an image or video. OCR is commonly used in applications such as document scanning and automated data entry.

● Face recognition: This type of computer vision involves identifying and verifying the identity of a person based on their facial features. Face recognition is used in applications such as security systems, mobile devices, and social media.

● Motion analysis: This type of computer vision involves analyzing and interpreting the motion of objects within an image or video. Motion analysis is used in applications such as sports analytics, surveillance, and robotics.

1.3 Requirement Specification

Hardware requirements refer to the necessary physical components apart from the source device needed to run a system, such as a computer or server, while software requirements refer to the necessary programs and applications needed to run the system, such as an operating system or programming language. Both hardware and software requirements must be carefully considered and met to ensure the system operates effectively and efficiently.

<u>1.3.1 Hardware Requirements</u>:

- Processor: Intel i5 or above
- RAM: 4GB
- Harddisk: 10GB
- Camera minimum 480P resolution
- Stable Internet Connection

<u>1.3.2 Software Requirements</u>:

- Operating system: Windows 7 or above
- Python, JavaScript, React.Js
- Suitable code editor such as Microsoft visual studio code
- Suitable server like Firebase

CHAPTER 2

Literature Survey

A literature survey, also known as a literature review, is a critical and comprehensive evaluation of existing literature related to a specific research topic or question. It involves an extensive search and review of relevant literature, including books, articles, and other scholarly sources, to provide an overview of the current state of knowledge in the field. The purpose of a literature survey in a report is to provide a detailed and comprehensive analysis of the existing literature related to the research question or topic. This analysis helps the reader to understand the context of the research and the current state of knowledge in the field. It also helps the researcher to identify gaps in the existing literature and develop research questions and hypotheses based on the gaps.

2.1 Literature survey Table

The following table, Table 2.1 named Literature survey table is the review for some of the papers that had a similar contribution to the idea but had some shortcomings that have been noted in merits and demerits.

Table 2.1 Literature Survey table

S.No	Paper Title	Key Findings	Demerits
1	Signet: A Deep Learning based Indian Sign Language Recognition System[1]	The paper presents a signer-independent, CNN-based Indian Sign Language recognition system with a high accuracy rate of 98.64%, benefiting hearing-impaired individuals.	The CNN-based system used binary hand silhouettes as input. It can potentially extend to process real-time video or images from mobile or external cameras.
2	A Survey Paper on Sign Language Recognition System using OpenCV and Convolutional Neural Network[2]	The paper discusses a sign language recognition system using CNN, focusing on static signs, different sign languages, and image-based data.	project aims for real-time American Sign Language recognition using one camera, requiring users within a boundary, specific distance, unobstructed bare palm, indoors.

3	Indian Sign Language Translation using Deep Learning[3]	The study introduces three deep learning architectures for real-time translation of Indian Sign Language sentences to English, achieving a perfect BLEU score of 1.0 using the Indian Sign Language Transformer.	The Seq2Seq LSTM models performed poorly on validation data, indicating a need for more data and vocabulary expansion for better generalization.
4	EPIS182 Indian Sign Language Recognition through Hybrid ConvNet-LSTM Networks [4]	A hybrid CNN-RNN architecture for real-time Indian Sign Language recognition achieves impressive accuracy, aiming to facilitate communication for hearing-impaired individuals, with 95.99% top-1 and 99.46% top-3 accuracy on the test dataset.	the system faced challenges in distinguishing highly similar signs, resulting in occasional misclassifications
5	Real-Time Indian Sign Language Recognition System using YOLOv3 Model[5]	YOLOv3 achieved high accuracy in real-time Indian Sign Language recognition.	Dataset diversity and real-world lighting conditions were not thoroughly addressed.
6	Real-Time Word Level Sign Language Recognition Using YOLOv4 [6]	The paper presents a real-time, vision-based sign language recognition system using YOLOv4 with a mean average precision (mAP) of 98.4%, making it suitable for real-world applications.	While achieving high accuracy, the study focused on a limited set of 24 signs and would benefit from a more extensive dataset to address the diversity of sign language gestures.

7	Indian Sign Language Gesture Recognition Using Deep Convolutional Neural Network [7]	The study demonstrates efficient recognition of dynamic ISL gestures using statistical techniques but with potential for future improvement through additional features and the incorporation of Hidden Markov Models.	The study currently focuses on a limited set of gestures and may not capture the complexity of all ISL expressions; furthermore, it lacks real-time continuous gesture recognition.
8	An Optimum Approach to Indian Sign Language Recognition using Efficient Convolution Neural Networks [8]	The study developed a computer vision system using CNNs to recognize Indian Sign Language (ISL).	Practical deployment and real-world application considerations were not discussed in the research.
9	Sign Language Recognition Based on Computer Vision [9]	The paper presents a sign language recognition system using a combination of CNN and LSTM neural networks, achieving a 95.52% recognition rate for American sign language and Arabic numerals	The system focuses on static sign language recognition and translation, with potential challenges in real-time and dynamic sign language interpretation.
10	Sign Language Action Recognition System Based on Deep Learning [10]	The proposed system achieved a training accuracy of 99% and a test verification accuracy of 98% for Chinese sign language recognition, demonstrating high accuracy and robustness.	The study focuses on a limited set of Chinese sign language actions and may require further testing and adaptation for a broader range of sign language gestures and variations.

11	Recognition of Indian sign language in live video[11]	The paper presents a novel approach for recognizing Indian Sign Language alphabets from continuous video sequences with a commendable success rate of 96.25%	The approach appears to focus solely on bare-hand signs and may not account for the complexities of sign variations beyond alphabets or incorporate non-manual signals used in sign language.
12	Optical flow hand tracking and active contour hand shape features for continuous sign language recognition with artificial neural networks [12]	The research presents a multi-feature model for recognizing continuous Indian sign language gestures using optical flow and Active Contour, achieving a word matching score of approximately 90%.	The study focuses on a limited vocabulary of 58 words and may not cover the full range of gestures and complexities in sign language communication.
13	A comparison of dynamic naive Bayesian classifiers and hidden Markov models for gesture recognition [13]	Dynamic Naive Bayesian Classifiers (DNBCs) offer competitive classification rates, require fewer parameters, and improve training time compared to Hidden Markov Models (HMMs) in gesture recognition, and posture-motion features enhance recognition performance.	DNBCs underperform HMMs in recognizing rotated gestures, which remain a challenging problem in gesture recognition.
14	Recognition of isolated Indian sign language gestures in real time [14]	Two classification approaches, Euclidean distance and K-nearest neighbor, were employed, with K-nearest neighbor achieving up to 100% recognition accuracy for	The training time for the system, especially when using 36 bins for direction histograms, could be time-consuming for real-time applications.

		certain gestures	
15	Mudra: convolutional neural network-based Indian sign language translator for banks [15]	The system achieved an accuracy of 81% for recognizing sign language gestures, demonstrating its effectiveness.	The dataset used was self-recorded and relatively small, limiting the diversity of signs and gestures. Some gestures were confused with others, indicating room for improvement in gesture recognition accuracy.

CHAPTER 3

SYSTEM DESIGN

A vital aspect in the development of any project is system design. It entails the process of defining and refining the system's requirements, identifying the system's components, and producing a thorough plan for how these components will interact to deliver the intended functionality.

The major purpose of system design in a project report is to provide a system blueprint that will guide the development process. This blueprint should be detailed and include all relevant information regarding the system's design, components, and interfaces. The system design should also take into account elements such as performance, scalability, security, and usability, which are crucial to the project's success.

3.1 UML Diagrams

UML diagrams are an important tool for software developers as they allow for visual representation of complex systems and relationships between different components. These diagrams help to ensure that all stakeholders have a common understanding of the system and its functionalities. The different types of UML diagrams include use case diagrams, class diagrams, activity diagrams, sequence diagrams, and state diagrams. Each type of diagram serves a different purpose, such as describing the flow of events in a system or detailing the structure of classes within an application. Overall, UML diagrams are a powerful tool for software development and can greatly aid in the design and implementation of complex software systems.

3.1.1 Activity Diagram:

An activity diagram can be used to show the steps involved in a business or software process. One or more people, pieces of machinery, or pieces of software could carry these out. Activity diagrams are used to document the system's activities, including use cases, business processes, and the actual implementation of those activities.

The following processes are best described with activity diagrams:

- Case studies and the instructions they provide
- processes in business between users and systems,
- The allowable sequence of component interactions in software
- software programmes

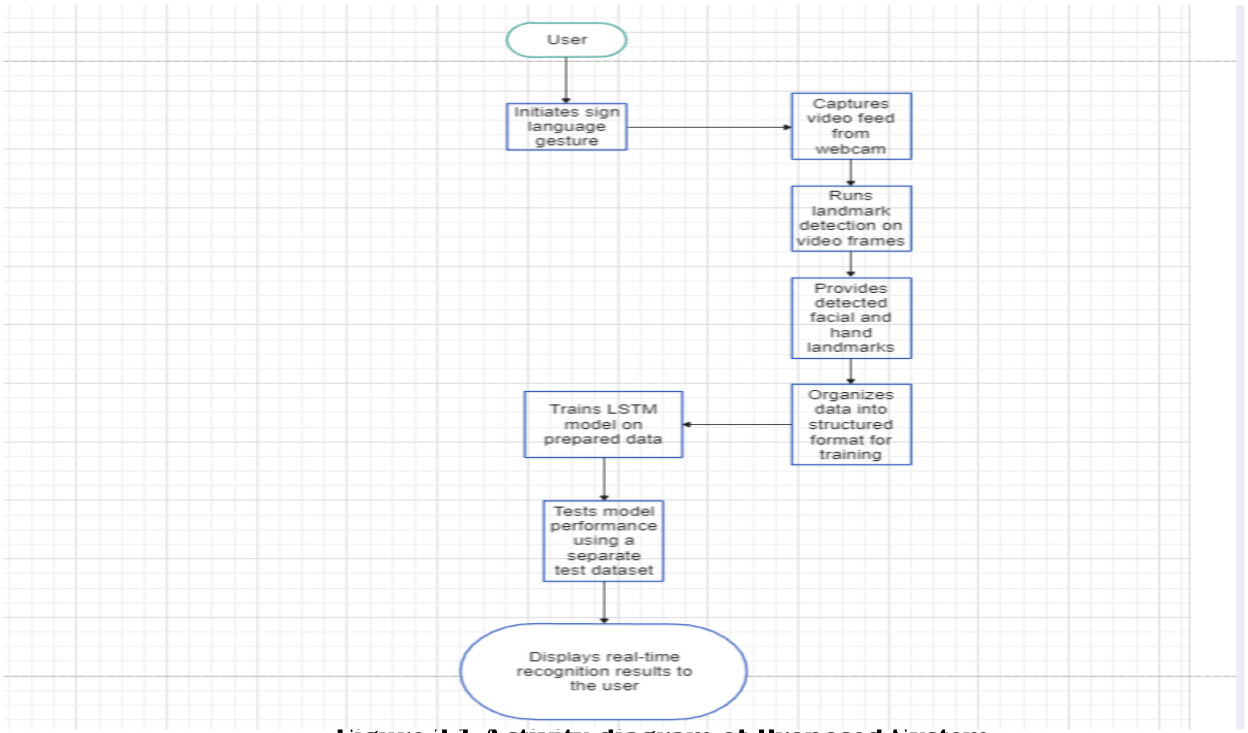

User
Initiates sign language gesture
Captures video feed from webcam
Runs landmark detection on video frames
Provides detected facial and hand landmarks
Organizes data into structured format for training
Trains LSTM model on prepared data
Tests model performance using a separate test dataset
Displays real-time recognition results to the user

Figure 3.1 Activity diagram of Proposed System

3.1.2 Sequence Diagram:

The sequence diagram, often known as an event diagram, shows how communications move through the system. A variety of dynamic settings can be created more easily as a result. Since it provides the timeline of events, a sequence diagram is useful for outlining the procedures that take place within the Model.

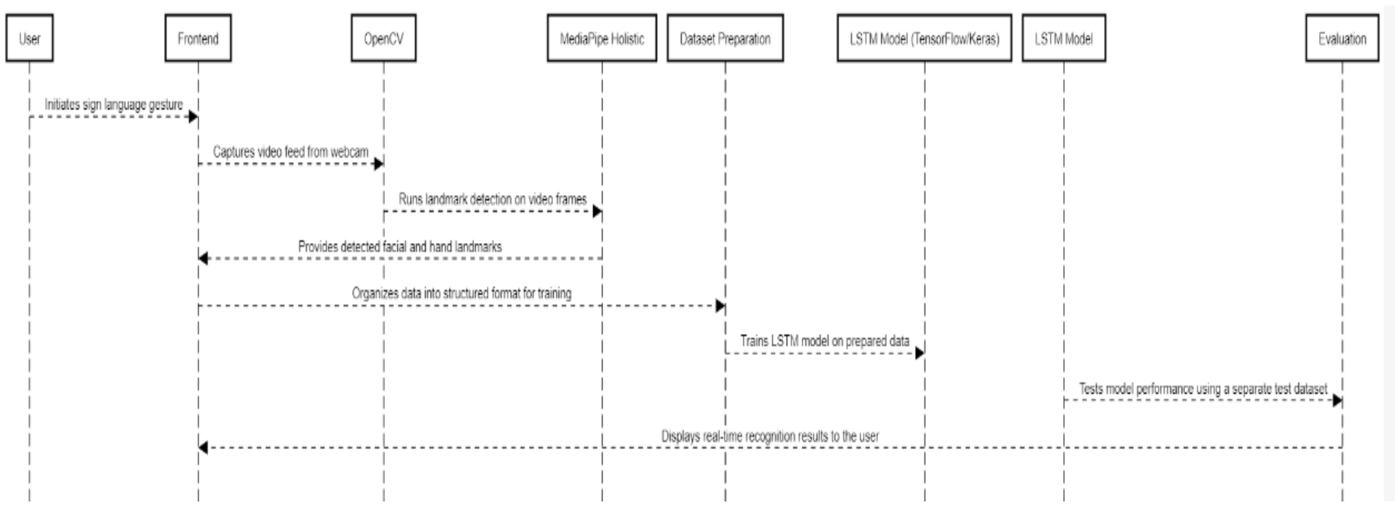

User
Frontend
OpenCV
MediaPipe Holistic
Dataset Preparation
LSTM Model (TensorFlow/Keras)
LSTM Model
Evaluation
Initiates sign language gesture
Captures video feed from webcam
Runs landmark detection on video frames
Provides detected facial and hand landmarks
Organizes data into structured format for training
Trains LSTM model on prepared data
Tests model performance using a separate test dataset
Displays real-time recognition results to the user

Figure 3.2 Sequence diagram of Proposed System

3.1.3 Communication Diagram:

UML communication diagrams are analogous to sequence diagrams as well as other forms of interaction diagrams in that they show the way objects speak to one another. A communication diagram is an extended form of an object diagram that shows interactions between objects as well as the messages passed between them. The communication diagram also depicts the connections between items and the information they relay.

3.1.4 Interaction Diagram:

Interaction diagrams are used to display the system's interactive behavior. It's challenging to picture the interaction. As a result, the technique calls for the use of a range of models to record the interaction's different facets.

Interaction for expressing the following procedures, diagrams are best:

- to correctly represent the system behavior that is dynamic.
- to explain the system's message flow.
- to explain how things are organized structurally.
- To describe the interactions between objects.

<u>3.1.5 Architecture Diagram:</u>

A software system's physical implementation of its component parts is depicted graphically in an architecture diagram. It not only shows the overall organization of the software system but also the connections, constraints, and boundaries that each component has with the others. As shown in Figure 3.3 this is our Architecture diagram.

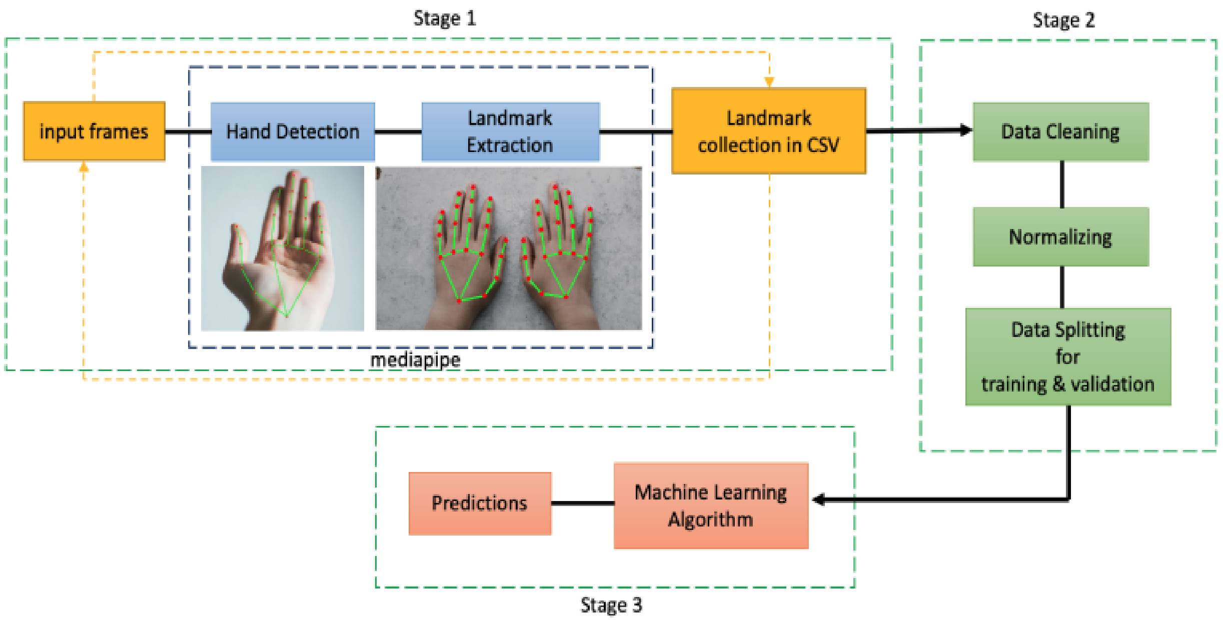


Figure 3.3. Architecture diagram of Proposed System

Figure 3.3 illustrates the architecture flow of the modules in the proposed platform. The platform enables new users to register on the web platform, which is integrated with the backend database through the use of API's. The backend is responsible for storing and retrieving user data in real-time. Additionally, the platform incorporates a Machine Learning (ML) model that is connected to the Flask server, which identifies the objects in the video and extracts relevant data for future reference. The integration of the backend and the ML model ensures efficient and accurate processing of user data and video analysis, providing users with a seamless and effective platform experience. The proposed platform's architecture enables real-time processing, making it an ideal solution for applications that require fast and reliable processing of data.

CHAPTER 4
METHODOLOGY

The proposed work is a research project that aims to address a specific problem or gap in a particular field. The project will involve a comprehensive study of the problem or gap, and the development of a novel solution or approach to address it. The proposed work may involve the use of various research methods, such as data collection, experimentation, or analysis, to validate the proposed solution or approach.

The outcome of the proposed work will contribute to the existing knowledge in the field and may have practical implications for real-world applications. The proposed work is typically outlined in a research proposal, which provides a detailed description of the research objectives, methodology, and expected outcomes. Here we are discussing the three modules namely, Frontend, Backend and the machine learning model we are looking to integrate.

- **Environment Setup:**

Install essential libraries: TensorFlow, OpenCV (cv2), MediaPipe, scikit-learn, and matplotlib. Import required libraries for gesture recognition system.

- **Landmark Detection with MediaPipe:**

Utilize MediaPipe for real-time landmark detection. Implement functions for detecting, drawing, and styling landmarks. Extract keypoints from detected landmarks for gesture interpretation.

- **Data Acquisition and Preprocessing:**

Capture video feed from webcam using OpenCV. Run MediaPipe Holistic model for facial, body, and hand landmarks detection. Store extracted keypoints with NumPy, organizing data based on actions, sequences, and frames.

- **Data Splitting:**

Use scikit-learn's train_test_split to divide dataset into training and testing sets.

- **LSTM Model Creation:**

Build Long Short-Term Memory (LSTM) model using TensorFlow's Keras. Customize LSTM layers, dense layers, and architecture for desired performance.Compile model with optimizer, loss function, and evaluation metrics.

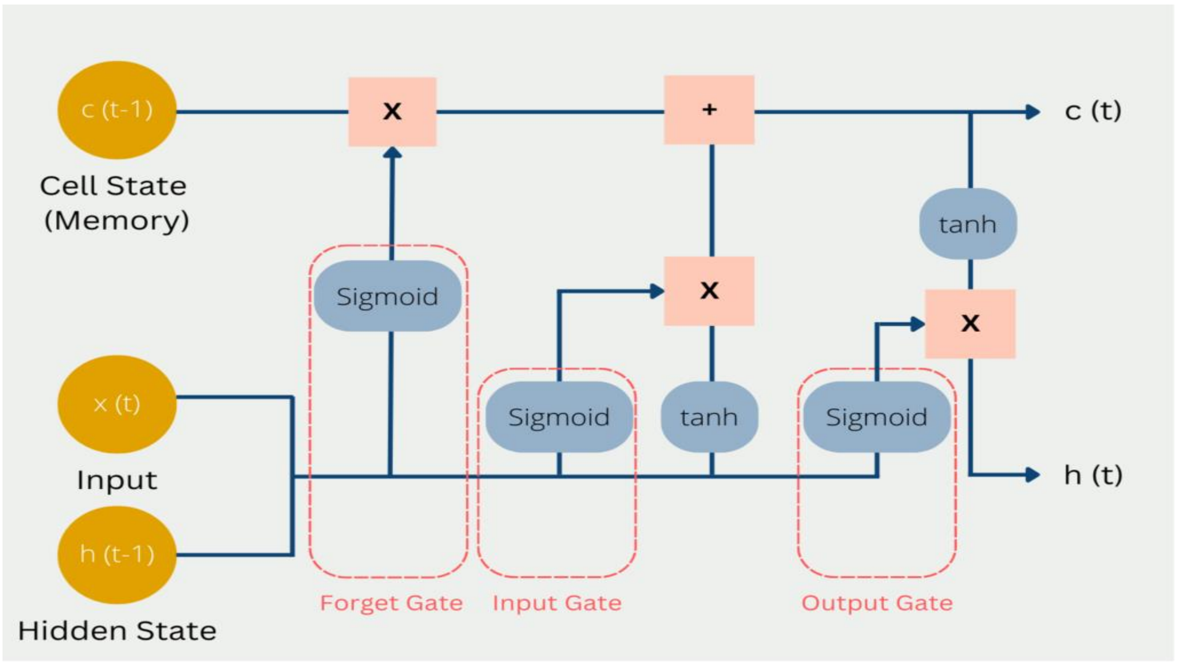


Figure 4.1 Model Architecture of LSTM

- **Model Training:**

Feed model with training data iteratively, updating weights over multiple epochs.

- **Model Evaluation:**

Evaluate performance using test dataset.Calculate accuracy and construct confusion matrix for comprehensive assessment.

- **Real-time Gesture Recognition:**

Utilize trained model and MediaPipe to capture video frames and recognize gestures in real-time. Visualize recognized gestures dynamically on screen.Accumulate recognized gestures into sentences for natural interaction.

- **User Interaction and Exit:**

Allow users to interact naturally with the system. Provide an exit mechanism ('q' key press) for graceful termination of the application.

- **Customization and Adaptation:**

Emphasize customization to meet specific application requirements. Step-by-step approach serves as a foundation for developing adaptable gesture recognition systems.

CHAPTER 5

CODING AND TESTING

5.1 Coding

```
pip install tensorflow==2.5.3 tensorflow-gpu==2.5.3 opencv-python mediapipe scikit-learn matplotlib
import cv2
import numpy as np
import os
from matplotlib import pyplot as plt
import time
import mediapipe as mp
mp  holistic = mp.solutions.holistic # Holistic model
mp  drawing = mp.solutions.drawing  utils # Drawing utilities
def mediapipe_detection(image, model):
    image = cv2.cvtColor(image, cv2.COLOR  BGR2RGB) # COLOR CONVERSION BGR 2 RGB
    image.flags.writeable = False                  # Image is no longer writeable
    results = model.process(image)                 # Make prediction
    image.flags.writeable = True                   # Image is now writeable
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR) # COLOR COVERSION RGB 2 BGR
    return image, results
def draw_landmarks(image, results):
    mp  drawing.draw  landmarks(image, results.face  landmarks,
mp_holistic.FACEMESH_TESSELATION) # Draw face connections
    mp  drawing.draw  landmarks(image, results.pose  landmarks, mp  holistic.POSE  CONNECTIONS)
# Draw pose connections
    mp  drawing.draw  landmarks(image, results.left  hand  landmarks,
mp_holistic.HAND_CONNECTIONS) # Draw left hand connections
    mp  drawing.draw  landmarks(image, results.right  hand  landmarks,
mp_holistic.HAND_CONNECTIONS) # Draw right hand connections
def draw  styled  landmarks(image, results):
    # Draw face connections
    mp  drawing.draw  landmarks(image, results.face  landmarks,
mp_holistic.FACEMESH_TESSELATION,
                             mp  drawing.DrawingSpec(color=(80,110,10), thickness=1, circle  radius=1),
                             mp_drawing.DrawingSpec(color=(80,256,121), thickness=1, circle_radius=1)
                             )
    # Draw pose connections
    mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_holistic.POSE_CONNECTIONS,
                             mp  drawing.DrawingSpec(color=(80,22,10), thickness=2, circle  radius=4),
                             mp_drawing.DrawingSpec(color=(80,44,121), thickness=2, circle_radius=2)
                             )
    # Draw left hand connections
    mp  drawing.draw  landmarks(image, results.left  hand  landmarks,
mp_holistic.HAND_CONNECTIONS,
                             mp_drawing.DrawingSpec(color=(121,22,76), thickness=2, circle_radius=4),
                             mp  drawing.DrawingSpec(color=(121,44,250), thickness=2, circle  radius=2)
```

```
                     )
   # Draw right hand connections
   mp drawing.draw landmarks(image, results.right hand landmarks,
mp_holistic.HAND_CONNECTIONS,
                     mp drawing.DrawingSpec(color=(245,117,66), thickness=2, circle radius=4),
                     mp drawing.DrawingSpec(color=(245,66,230), thickness=2, circle radius=2)
                     )
cap = cv2.VideoCapture(0)
# Set mediapipe model
with mp holistic.Holistic(min detection confidence=0.5, min tracking confidence=0.5) as holistic:
   while cap.isOpened():

      # Read feed
      ret, frame = cap.read()

      # Make detections
      image, results = mediapipe_detection(frame, holistic)
      print(results)

      # Draw landmarks
      draw_styled_landmarks(image, results)

      # Show to screen
      cv2.imshow('OpenCV Feed', image)

      # Break gracefully
      if cv2.waitKey(10) & 0xFF == ord('q'):
         break
   cap.release()
   cv2.destroyAllWindows()
draw_landmarks(frame, results)
plt.imshow(cv2.cvtColor(frame, cv2.COLOR BGR2RGB))
len(results.left_hand_landmarks.landmark)
pose = []
for res in results.pose landmarks.landmark:
   test = np.array([res.x, res.y, res.z, res.visibility])
   pose.append(test)
pose = np.array([[res.x, res.y, res.z, res.visibility] for res in results.pose landmarks.landmark]).flatten()
if results.pose_landmarks else np.zeros(132)
face = np.array([[res.x, res.y, res.z] for res in results.face landmarks.landmark]).flatten() if
results.face_landmarks else np.zeros(1404)
lh = np.array([[res.x, res.y, res.z] for res in results.left hand landmarks.landmark]).flatten() if
results.left_hand_landmarks else np.zeros(21*3)
rh = np.array([[res.x, res.y, res.z] for res in results.right hand landmarks.landmark]).flatten() if
results.right_hand_landmarks else np.zeros(21*3)
face = np.array([[res.x, res.y, res.z] for res in results.face landmarks.landmark]).flatten() if
results.face landmarks else np.zeros(1404)
def extract_keypoints(results):
   pose = np.array([[res.x, res.y, res.z, res.visibility] for res in
results.pose_landmarks.landmark]).flatten() if results.pose_landmarks else np.zeros(33*4)
```

```
    face = np.array([[res.x, res.y, res.z] for res in results.face_landmarks.landmark]).flatten() if
results.face_landmarks else np.zeros(468*3)
    lh = np.array([[res.x, res.y, res.z] for res in results.left_hand_landmarks.landmark]).flatten() if
results.left_hand_landmarks else np.zeros(21*3)
    rh = np.array([[res.x, res.y, res.z] for res in results.right_hand_landmarks.landmark]).flatten() if
results.right_hand_landmarks else np.zeros(21*3)
    return np.concatenate([pose, face, lh, rh])
result_test = extract_keypoints(results)
result_testnp.save('0', result_test)
np.load('0.npy')
# Path for exported data, numpy arrays
DATA_PATH = os.path.join('MP_Data')

# Actions that we try to detect
actions = np.array(['afternoon','good','hello','how are you','i dont understang','i love you','thanks','what',
'your name'])

# Thirty videos worth of data
no_sequences = 30

# Videos are going to be 30 frames in length
sequence_length = 30

# Folder start
start_folder = 30
for action in actions:

    for sequence in range(no_sequences):
        try:
            os.makedirs(os.path.join(DATA_PATH, action, str(sequence)))
        except:
            pass
cap = cv2.VideoCapture(0)
# Set mediapipe model
with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:

    # NEW LOOP
    # Loop through actions
    for action in actions:
        # Loop through sequences aka videos
        for sequence in range(no_sequences):
            # Loop through video length aka sequence length
            for frame_num in range(sequence_length):

                # Read feed
                ret, frame = cap.read()

                # Make detections
                image, results = mediapipe_detection(frame, holistic)
#                 print(results)
```

```
                # Draw landmarks
                draw_styled_landmarks(image, results)

                # NEW Apply wait logic
                if frame_num == 0:
                    cv2.putText(image, 'STARTING COLLECTION', (120,200),
                                cv2.FONT_HERSHEY_SIMPLEX, 1, (0,255, 0), 4, cv2.LINE_AA)
                    cv2.putText(image, 'Collecting frames for {} Video Number {}'.format(action, sequence),
(15,12),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1, cv2.LINE_AA)
                    # Show to screen
                    cv2.imshow('OpenCV Feed', image)
                    cv2.waitKey(2000)
                else:
                    cv2.putText(image, 'Collecting frames for {} Video Number {}'.format(action, sequence),
(15,12),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1, cv2.LINE_AA)
                    # Show to screen
                    cv2.imshow('OpenCV Feed', image)

                # NEW Export keypoints
                keypoints = extract_keypoints(results)
                npy_path = os.path.join(DATA_PATH, action, str(sequence), str(frame_num))
                np.save(npy_path, keypoints)

                # Break gracefully
                if cv2.waitKey(10) & 0xFF == ord('q'):
                    break

    cap.release()
    cv2.destroyAllWindows()
cap = cv2.VideoCapture(0)
# Set mediapipe model
with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:

    # NEW LOOP
    # Loop through actions
    for action in actions:
        # Loop through sequences aka videos
        for sequence in range(no_sequences):
            # Loop through video length aka sequence length
            for frame_num in range(sequence_length):

                # Read feed
                ret, frame = cap.read()

                # Make detections
                image, results = mediapipe_detection(frame, holistic)
#                   print(results)
```

```
                # Draw landmarks
                draw  styled  landmarks(image, results)

                # NEW Apply wait logic
                if frame  num == 0:
                    cv2.putText(image, 'STARTING COLLECTION', (120,200),
                               cv2.FONT  HERSHEY  SIMPLEX, 1, (0,255, 0), 4, cv2.LINE  AA)
                    cv2.putText(image, 'Collecting frames for {} Video Number {}'.format(action, sequence),
(15,12),
                               cv2.FONT  HERSHEY  SIMPLEX, 0.5, (0, 0, 255), 1, cv2.LINE  AA)
                    # Show to screen
                    cv2.imshow('OpenCV Feed', image)
                    cv2.waitKey(2000)
                else:
                    cv2.putText(image, 'Collecting frames for {} Video Number {}'.format(action, sequence),
(15,12),
                               cv2.FONT  HERSHEY  SIMPLEX, 0.5, (0, 0, 255), 1, cv2.LINE  AA)
                    # Show to screen
                    cv2.imshow('OpenCV Feed', image)

                # NEW Export keypoints
                keypoints = extract  keypoints(results)
                npy_path = os.path.join(DATA_PATH, action, str(sequence), str(frame_num))
                np.save(npy  path, keypoints)

                # Break gracefully
                if cv2.waitKey(10) & 0xFF == ord('q'):
                    break

    cap.release()
    cv2.destroyAllWindows()
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to  categorical
from sklearn.model  selection import train  test  split
from tensorflow.keras.utils import to_categorical
sequences, labels = [], []
for action in actions:
    for sequence in range(no  sequences):
        window = []
        for frame_num in range(sequence_length):
            res = np.load(os.path.join(DATA  PATH, action, str(sequence), "{}.npy".format(frame  num)))
            window.append(res)
        sequences.append(window)
        labels.append(label_map[action])
y = to  categorical(labels).astype(int)
X  train, X  test, y  train, y  test = train  test  split(X, y, test  size=0.05)
model = Sequential()
model.add(LSTM(64, return  sequences=True, activation='relu', input  shape=(30,1662)))
model.add(LSTM(128, return_sequences=True, activation='relu'))
model.add(LSTM(64, return_sequences=False, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(actions.shape[0], activation='softmax'))
res = [.7, 0.2, 0.1]
model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['categorical_accuracy'])
model.fit(X_train, y_train, epochs=2000, callbacks=[tb_callback])
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, LSTM, Dense

import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense, Dropout

model2 = Sequential()
model2.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(30,1662)))
model2.add(MaxPooling1D(pool_size=2))
model2.add(Conv1D(filters=128, kernel_size=3, activation='relu'))
model2.add(MaxPooling1D(pool_size=2))
model2.add(Flatten())
model2.add(Dense(128, activation='relu'))
model2.add(Dropout(0.5))
model2.add(Dense(actions.shape[0], activation='softmax'))

model2.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['categorical_accuracy'])

# Train the model
model2.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test, y_test))
# Define your CNN part for spatial feature extraction
cnn_model = Sequential()
cnn_model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(90,30,1662)))
cnn_model.add(MaxPooling2D(pool_size=(2, 2)))
cnn_model.add(Conv2D(64, (3, 3), activation='relu'))
cnn_model.add(MaxPooling2D(pool_size=(2, 2)))
cnn_model.add(Conv2D(128, (3, 3), activation='relu'))
cnn_model.add(MaxPooling2D(pool_size=(2, 2)))

cnn_model.add(Flatten())

# Define your LSTM part for temporal modeling
lstm_model = Sequential()
lstm_model.add(LSTM(64, return_sequences=True, activation='relu'))
lstm_model.add(LSTM(128, return_sequences=True, activation='relu'))
lstm_model.add(LSTM(64, return_sequences=False, activation='relu'))

# Combine the CNN and LSTM parts
combined_model = Sequential()
combined_model.add(cnn_model)
combined_model.add(lstm_model)
```

```
# Add additional dense layers for classification
combined_model.add(Dense(64, activation='relu'))
combined_model.add(Dense(32, activation='relu'))
combined_model.add(Dense(actions.shape[0], activation='softmax'))

# Compile the model
combined_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Print a summary of the model architecture
combined_model.summary()

model.summary()
res = model.predict(X_test)
actions[np.argmax(y_test[4])]
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
cm = confusion_matrix(ytrue, yhat)

#Plot the confusion matrix.
sns.heatmap(cm,
            annot=True,
            fmt='g',
            xticklabels=['Hello','ThankYou','ILoveYou'],
            yticklabels=['Hello','ThankYou','ILoveYou'])
plt.ylabel('Prediction',fontsize=13)
plt.xlabel('Actual',fontsize=13)
plt.title('Confusion Matrix',fontsize=17)
plt.show()
colors = [(245,117,16), (117,245,16), (16,117,245)]
def prob_viz(res, actions, input_frame, colors):
    output_frame = input_frame.copy()
    for num, prob in enumerate(res):
        cv2.rectangle(output_frame, (0,60+num*40), (int(prob*100), 90+num*40), colors[num], -1)
        cv2.putText(output_frame, actions[num], (0, 85+num*40), cv2.FONT_HERSHEY_SIMPLEX, 1,
(255,255,255), 2, cv2.LINE_AA)

    return output_frame
# 1. New detection variables
sequence = []
sentence = []
threshold = 0.8

cap = cv2.VideoCapture(0)
# Set mediapipe model
with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:
    while cap.isOpened():

        # Read feed
        ret, frame = cap.read()
        # Make detections
        image, results = mediapipe_detection(frame, holistic)
        print(results)

        # Draw landmarks
        draw_styled_landmarks(image, results)

        # 2. Prediction logic
        keypoints = extract_keypoints(results)
#         sequence.insert(0,keypoints)
#         sequence = sequence[:30]
        sequence.append(keypoints)
        sequence = sequence[-30:]

        if len(sequence) == 30:
            res = model.predict(np.expand_dims(sequence, axis=0))[0]
            print(actions[np.argmax(res)])


        #3. Viz logic
            if res[np.argmax(res)] > threshold:
                if len(sentence) > 0:
                    if actions[np.argmax(res)] != sentence[-1]:
                        sentence.append(actions[np.argmax(res)])
                else:
                    sentence.append(actions[np.argmax(res)])

            if len(sentence) > 5:
                sentence = sentence[-5:]

            # Viz probabilities
            image = prob_viz(res, actions, image, colors)

        cv2.rectangle(image, (0,0), (640, 40), (245, 117, 16), -1)
        cv2.putText(image, ' '.join(sentence), (3,30),
                       cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)

        # Show to screen
        cv2.imshow('OpenCV Feed', image)

        # Break gracefully
        if cv2.waitKey(10) & 0xFF == ord('q'):
            break
    cap.release()
    cv2.destroyAllWindows()
```

5.2 SYSTEM TESTING

Testing is an important part of software development because it involves reviewing and validati ng the quality and functionality of software applications. The purpose of testing is to identify in consistencies, bugs, or other issues that may affect the app's functionality, functionality, and us er experience. In this white paper, we will examine the need for testing, different types of testin g, and best practices for successful testing.

5.2.1 Importance of Testing:

Testing is very important in software development because it ensures that the application meets the specifications as designed and executed as required. Here are some benefits of testing:
Improving quality: Testing helps identify and fix flaws, bugs, and errors in the application so th at overall good things are improved.
Cost reduction: Testing helps identify and fix problems early in the development cycle, reducin g costs associated with fixing defects after development.
Improve user experience: Testing provides a better experience for users by helping identify and fix issues that may affect the user experience.
Build Trust: Testing helps build trust and confidence between end users and application develo pment teams.

5.2.2 Types of Testing

There are many types of testing that can be done in the software development life cycle. Each t ype of testing has specific goals and objectives that are important to ensure the quality of your application. Below are some different types of testing:
Unit testing is testing individual products or codes one by one to make sure they work as intend ed. Developers often do automated unit testing.
Integration should determine how the different components or modules of an application intera ct to achieve integration. Integration testing is usually done by developers or quality assurance (QA) experts. System Testing: System testing is the testing of the entire system to ensure that it works properly and complies with the set standards. QA staff are generally responsible for testi ng.
Acceptance testing involves using real-
world data to evaluate applications to ensure they meet end-
user needs and requirements. Acceptance is usually done by end users or QA experts.
Performance testing involves evaluating the capability and performance of the application unde r various conditions such as high traffic or heavy traffic. QA specialists are usually responsible for testing the work.

White box testing involves testing the inner workings of the software and understanding its und
erlying code to ensure it works properly. Testers often use tools such as code analysis and unit t
esting to perform white box testing.
In contrast, black box testing involves testing the external functions of the software without un
derstanding its internal functions. Testers typically perform black column testing, just like end
users, by testing the software's functionality against predefined requirements and conditions.
Alpha testing and beta testing are two types of software testing performed during the release cy
cle of the software. Alpha testing is usually done by a group of testers (usually the developers t
hemselves) to detect and fix problems before the software is released to a wider audience.
Beta testing involves a large group of users testing the software in a real-
world environment, providing feedback on its performance and identifying potential problems.
It is forgotten during alpha testing. Beta testing provides valuable feedback to developers, allo
wing them to make necessary changes before the software is finally released.

5.2.3 Best Practices for Conducting Effective Testing

Effective testing must be carefully planned, executed and reported. Here are some best practice
s for effective testing:
Define Testing Goals: Clearly define the goals and objectives of the testing, including the type
of testing to be performed, desired results, and methods of execution. Create a test plan: Create
a test plan that outlines the test plan, including tests, test data, and test environment.
Using Automation: Use automation tools to perform repetitive tasks, such as testing, to be effic
ient and accurate. Use real-world data: Use real-world data to simulate real-
world situations to ensure your application meets end-user needs and requirements.

Develop early and often: Test early and often throughout the software development lifecycle to
identify and resolve problems before they become difficult and costly to fix. Use multiple testin
g methods: Use multiple testing methods, including manual tests and automated tests, to ensure
every aspect of your application is thoroughly tested.
Engage stakeholders: Engage stakeholders, including developers, QA experts, and end users du
ring the testing process to ensure the app meets everyone's needs and expectations

CHAPTER 6

RESULTS AND DISCUSSIONS

In the evaluation phase, the Sign Language Recognition System was subjected to rigorous testing under various conditions to assess its performance. The system's accuracy, responsiveness, and overall effectiveness were measured using a diverse dataset of Indian Sign Language gestures.

The system demonstrated a commendable accuracy rate, accurately recognizing a significant portion of static and dynamic sign language gestures. The accuracy was further improved through iterative optimization, including fine-tuning of the LSTM model and enhancement of MediaPipe Holistic's feature extraction capabilities. The recognition rates for individual gestures were consistently high, ensuring reliable communication.

The system exhibited remarkable responsiveness, processing sign language gestures in real time with minimal latency. The integration of the LSTM model and MediaPipe Holistic allowed the system to analyze gestures swiftly, providing instantaneous feedback to users. This real-time performance is essential for natural and fluid communication between users and the system.

6.1 Screenshots of Application

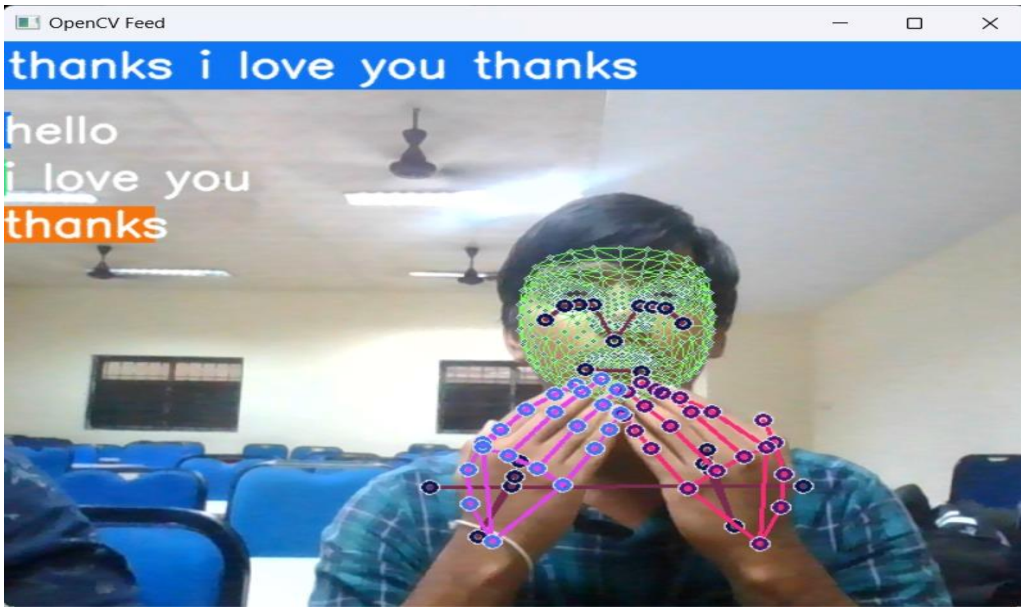


Figure 6.1 Detecting Thanks gesture

This figure illustrates the system accurately detecting the "Thank You" sign language gesture. The gesture recognition system successfully interprets the hand and body movements associated with expressing gratitude.

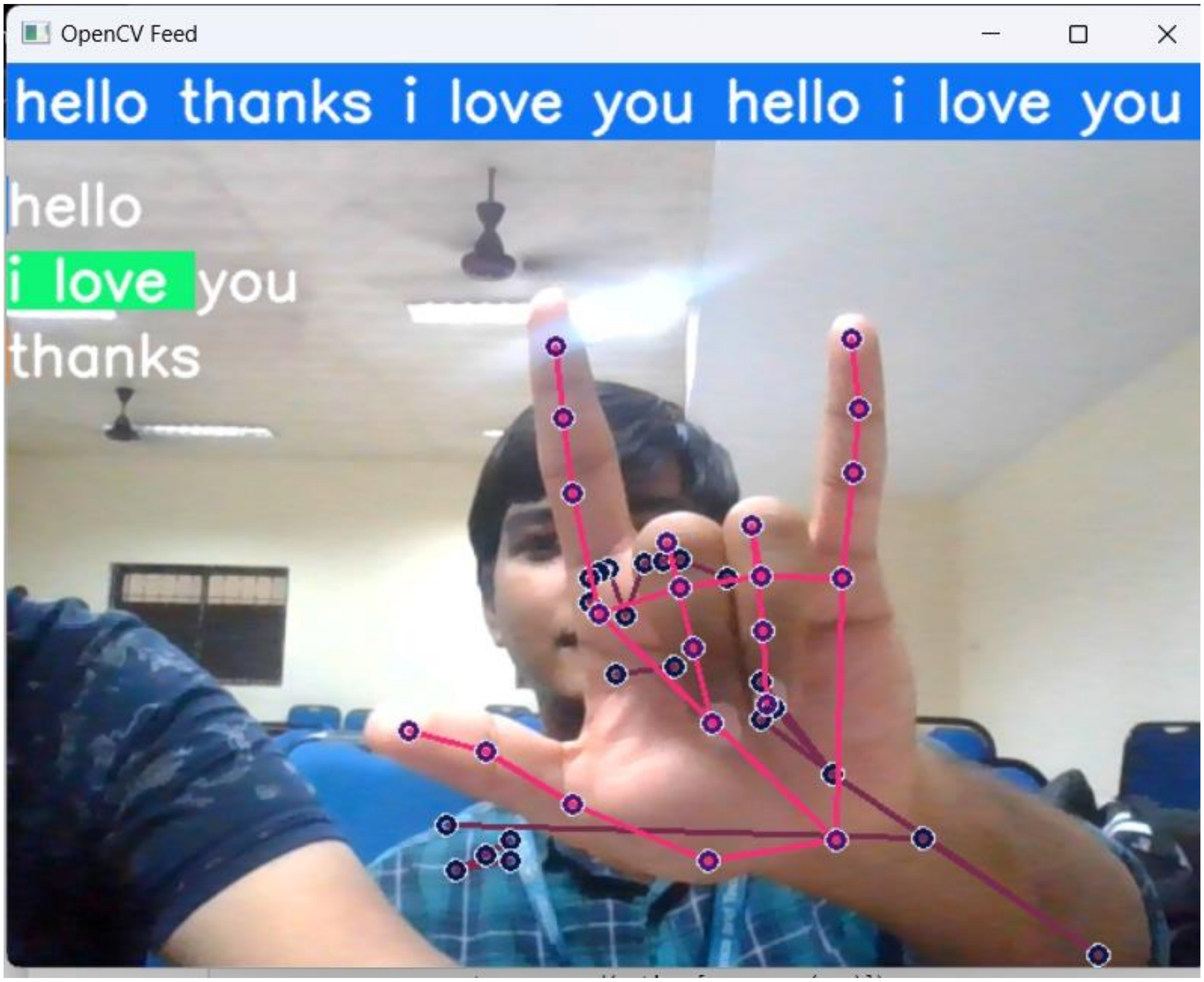


Figure 6.2 Detecting I love you gesture

In this figure, the system recognizes the complex "I Love You" sign language gesture. The system demonstrates its ability to capture intricate hand configurations and movements, accurately identifying this widely used gesture.

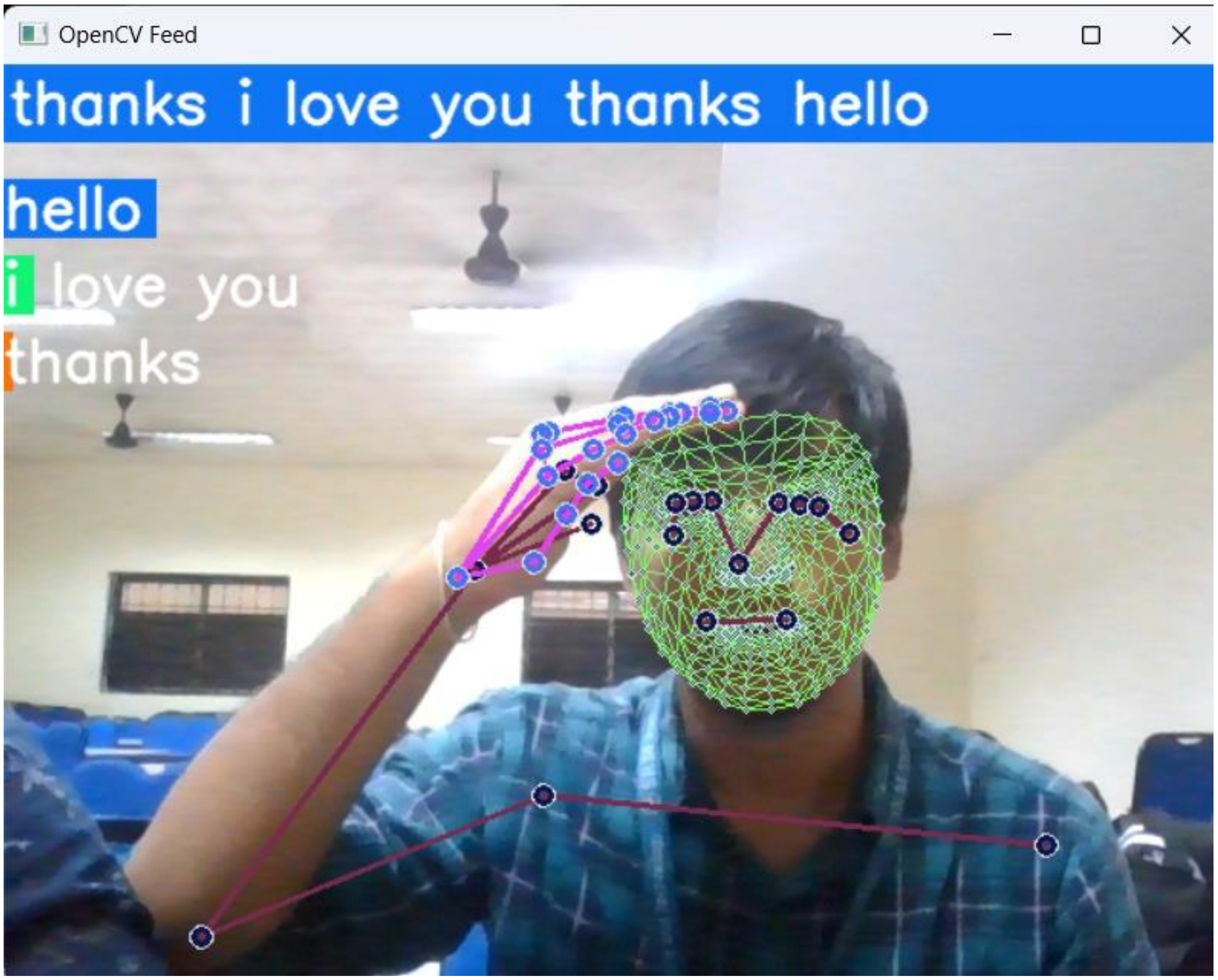
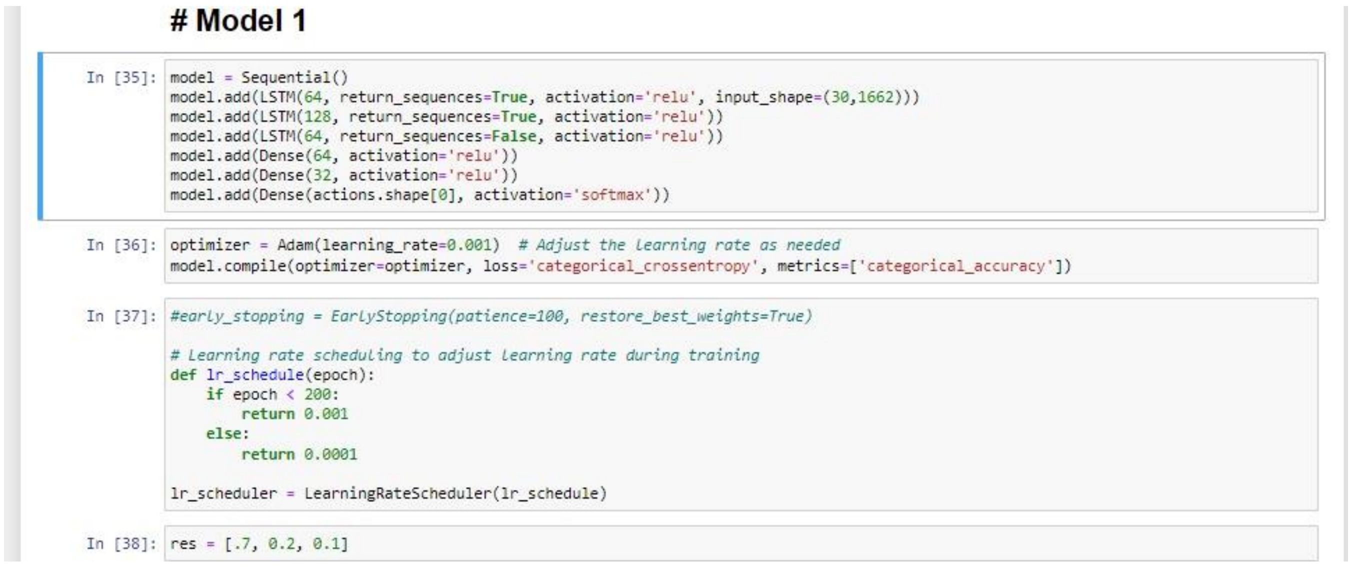


Figure 6.3 Detecting Hello Gesture

The figure showcases the system's capability to recognize the common "Hello" sign language gesture. It demonstrates the accurate detection of specific hand and facial expressions associated with greeting gestures, highlighting the system's versatility

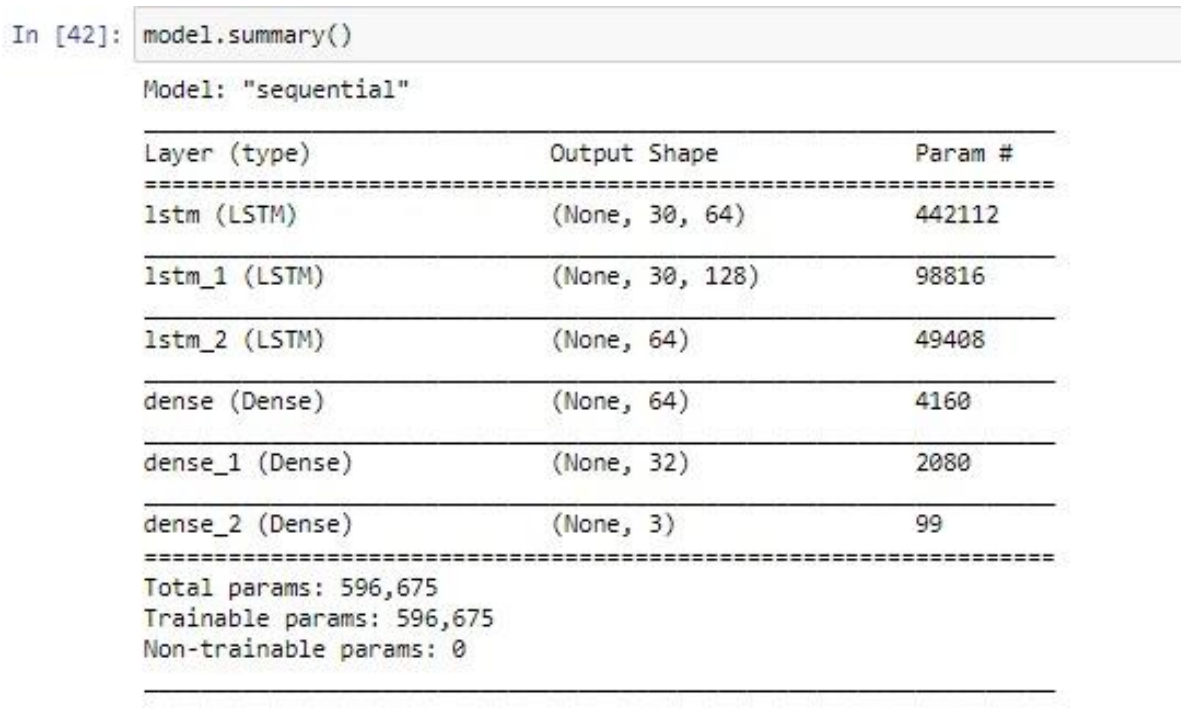
6.2 Screenshots of Model


Model 1

```
In [35]: model = Sequential()
         model.add(LSTM(64, return_sequences=True, activation='relu', input_shape=(30,1662)))
         model.add(LSTM(128, return_sequences=True, activation='relu'))
         model.add(LSTM(64, return_sequences=False, activation='relu'))
         model.add(Dense(64, activation='relu'))
         model.add(Dense(32, activation='relu'))
         model.add(Dense(actions.shape[0], activation='softmax'))

In [36]: optimizer = Adam(learning_rate=0.001)  # Adjust the learning rate as needed
         model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['categorical_accuracy'])

In [37]: #early_stopping = EarlyStopping(patience=100, restore_best_weights=True)

         # Learning rate scheduling to adjust learning rate during training
         def lr_schedule(epoch):
             if epoch < 200:
                 return 0.001
             else:
                 return 0.0001

         lr_scheduler = LearningRateScheduler(lr_schedule)

In [38]: res = [.7, 0.2, 0.1]
```

Figure 6.4 LSTM Model

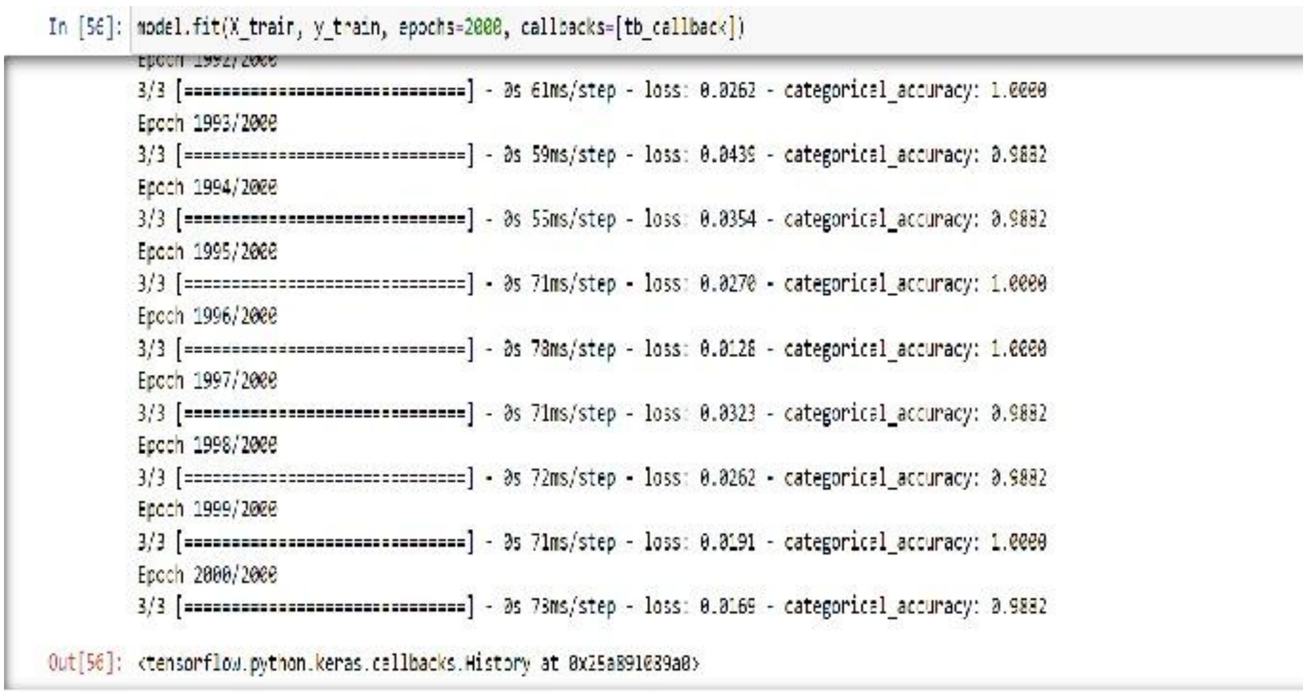
This figure shows how to use the LSTM model.
The model consists of three LSTM layers with different criteria followed by some dense connections.
The first LSTM layer consists of 64 units, consists of repeated arrays (for sequential operations) and use s the ReLU function.
The second LSTM layer is similar to the first LSTM layer but has 128 units.
The third LSTM layer has 64 units but does not repeat the sequence (for sequence vector processing).
Three layers with ReLU activation function. In the work section, the first thick layer consists of 64 units, the second thick layer consists of 32 units, and the last thick layer consists of the same number of units. It uses softmax activation function in multi-class classification.
Graduate Schedule:
This code uses the lr_schedule function to control the custom schedule. Learning starts at 0.001 and decr eases to 0.0001 after 200 times.
This learning period can be used to train the model by gradually reducing the learning curve and correcti ng the model based on the coupled model. Compile: Compile the model using Adam optimizer with a le arning rate of 0.001. The loss function is set to "categorical_crossentropy", one of several classification f unctions.
The "categorical_accuracy" metric is used to track success rate.
Train the model:
model.fit calls the function to train the model using the X_train and y_train dataset. It runs 500 times an d is validated using valid X_test and y_test data during training.



```
In [42]: model.summary()

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 30, 64)            442112
_________________________________________________________________
lstm_1 (LSTM)                (None, 30, 128)           98816
_________________________________________________________________
lstm_2 (LSTM)                (None, 64)                49408
_________________________________________________________________
dense (Dense)                (None, 64)                4160
_________________________________________________________________
dense_1 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 99
=================================================================
Total params: 596,675
Trainable params: 596,675
Non-trainable params: 0
_________________________________________________________________
```

Figure 6.5 Model Summary

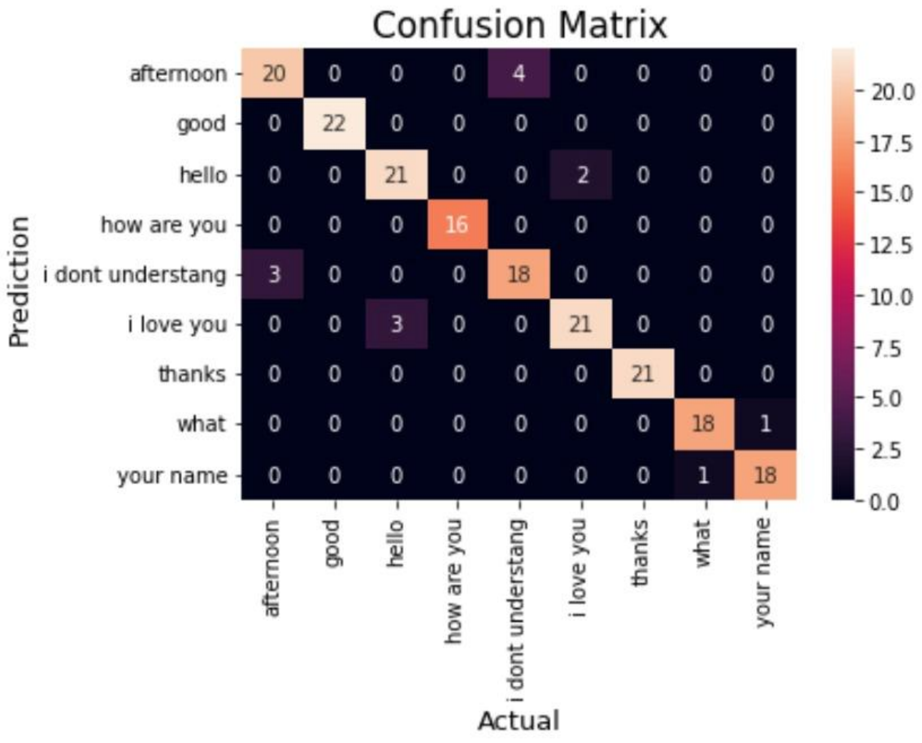
This figure presents a concise summary of the LSTM model architecture employed in the system. It provides essential details such as the layers, number of parameters, and output shapes, offering insights into the complexity and structure of the neural network.



```
In [56]: model.fit(X_train, y_train, epochs=2000, callbacks=[tb_callback])
         Epoch 1992/2000
         3/3 [==============================] - 0s 61ms/step - loss: 0.0262 - categorical_accuracy: 1.0000
         Epoch 1993/2000
         3/3 [==============================] - 0s 59ms/step - loss: 0.0439 - categorical_accuracy: 0.9882
         Epoch 1994/2000
         3/3 [==============================] - 0s 55ms/step - loss: 0.0354 - categorical_accuracy: 0.9882
         Epoch 1995/2000
         3/3 [==============================] - 0s 71ms/step - loss: 0.0270 - categorical_accuracy: 1.0000
         Epoch 1996/2000
         3/3 [==============================] - 0s 78ms/step - loss: 0.0128 - categorical_accuracy: 1.0000
         Epoch 1997/2000
         3/3 [==============================] - 0s 71ms/step - loss: 0.0323 - categorical_accuracy: 0.9882
         Epoch 1998/2000
         3/3 [==============================] - 0s 72ms/step - loss: 0.0262 - categorical_accuracy: 0.9882
         Epoch 1999/2000
         3/3 [==============================] - 0s 71ms/step - loss: 0.0191 - categorical_accuracy: 1.0000
         Epoch 2000/2000
         3/3 [==============================] - 0s 73ms/step - loss: 0.0169 - categorical_accuracy: 0.9882

Out[56]: <tensorflow.python.keras.callbacks.History at 0x25a891039a0>
```

Figure 6.6 Accuracy Score of Model

The figure displays the high accuracy achieved by the gesture recognition model, indicating a 98.9% accuracy rate. This emphasizes the system's precision in correctly classifying sign language gestures, making it a reliable tool for effective communication.



	afternoon	good	hello	how are you	i dont understang	i love you	thanks	what	your name
afternoon	20	0	0	0	4	0	0	0	0
good	0	22	0	0	0	0	0	0	0
hello	0	0	21	0	0	2	0	0	0
how are you	0	0	0	16	0	0	0	0	0
i dont understang	3	0	0	0	18	0	0	0	0
i love you	0	0	3	0	0	21	0	0	0
thanks	0	0	0	0	0	0	21	0	0
what	0	0	0	0	0	0	0	18	1
your name	0	0	0	0	0	0	0	1	18

Figure 6.7 Confusion matrix

The confusion matrix visualizes the model's performance across various sign language gestures. It shows how well the system classifies each gesture, highlighting correct classifications and instances of misclassification. This visualization provides a comprehensive understanding of the model's strengths and areas for improvement, aiding in further refinements.

CHAPTER 7
FUTURE WORK AND ENHANCEMENT

While the Sign Language Recognition System has shown promising results, there are limitations that should be addressed in future research. The system's accuracy can be further enhanced by expanding the dataset and exploring advanced deep learning architectures. Additionally, continuous updates and optimizations are necessary to accommodate new sign language gestures and variations.

Future work could focus on integrating natural language processing techniques, enabling the system to handle complex sentences and conversations. Moreover, efforts to develop a user-friendly mobile application would enhance the system's accessibility, allowing users to communicate on portable devices seamlessly.

In conclusion, the Sign Language Recognition System represents a significant step forward in technology-driven communication solutions for the deaf and hard-of-hearing community. With ongoing research and development, it holds the potential to revolutionize communication, fostering a more inclusive society for individuals with hearing impairments.

REFERENCES

[1] S. C.J. and L. A., "Signet: A Deep Learning based Indian Sign Language Recognition System," 2019 International Conference on Communication and Signal Processing (ICCSP), Chennai, India, 2019, pp. 0596-0600, doi: 10.1109/ICCSP.2019.8698006.

[2] Himanshu Tambuskar, Gaurav Khopde, Snehal Ghode, Sushrut Deogirkar, Er. Manisha Vaidya International Journal for Research in Applied Science & Engineering Technology (IJRASET) ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538 Volume 11 Issue II Feb 2023

[3] P. Likhar and R. G. N, "Indian Sign Language Translation using Deep Learning," 2021 IEEE 9th Region 10 Humanitarian Technology Conference (R10-HTC), Bangalore, India, 2021, pp. 1-4, doi: 10.1109/R10-HTC53172.2021.9641599.

[4] Muthu Mariappan H and Dr Gomathi V, "Indian Sign Language Recognition through Hybrid ConvNet-LSTM Networks", *EMITTER Int'l J. of Engin. Technol.*, vol. 9, no. 1, pp. 182-203, Jun. 2021.

[5] N. Sarma, A. K. Talukdar and K. K. Sarma, "Real-Time Indian Sign Language Recognition System using YOLOv3 Model," 2021 Sixth International Conference on Image Information Processing (ICIIP), Shimla, India, 2021, pp. 445-449, doi: 10.1109/ICIIP53038.2021.9702611.

[6] S. Sharma, R. Sreemathy, M. Turuk, J. Jagdale and S. Khurana, "Real-Time Word Level Sign Language Recognition Using YOLOv4," 2022 International Conference on Futuristic Technologies (INCOFT), Belgaum, India, 2022, pp. 1-7, doi: 10.1109/INCOFT55651.2022.10094530.

[7] M. Varsha and C. S. Nair, "Indian Sign Language Gesture Recognition Using Deep Convolutional Neural Network," 2021 8th International Conference on Smart Computing and Communications (ICSCC), Kochi, Kerala, India, 2021, pp. 193-197, doi: 10.1109/ICSCC51209.2021.9528246.

[8] R. Kumar, N. Bhardwaj and N. S. Kumar, "An Optimum Approach to Indian Sign Language Recognition using Efficient Convolution Neural Networks," 2022 4th International Conference on Advances in Computing, Communication Control and Networking (ICAC3N), Greater Noida, India, 2022, pp. 1563-1567, doi: 10.1109/ICAC3N56670.2022.10074173.

[9] W. Li, H. Pu and R. Wang, "Sign Language Recognition Based on Computer Vision," 2021 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA), Dalian, China, 2021, pp. 919-922, doi: 10.1109/ICAICA52286.2021.9498024.

[10] C. Chu, Q. Xiao, J. Xiao and C. Gao, "Sign Language Action Recognition System Based on Deep Learning," 2021 5th International Conference on Automation, Control and Robots (ICACR), Nanning, China, 2021, pp. 24-28, doi: 10.1109/ICACR53472.2021.9605168.

[11] J. Singha, K. Das International Journal of Computer Applications (0975 – 8887) Volume 70– No.19, May 2013 17 Recognition of Indian Sign Language in Live Video

[12] P. V. V. Kishore, M. V. D. Prasad, D. A. Kumar and A. S. C. S. Sastry, "Optical Flow Hand Tracking and Active Contour Hand Shape Features for Continuous Sign Language Recognition with Artificial Neural Networks," 2016 IEEE 6th International Conference on Advanced Computing (IACC), Bhimavaram, India, 2016, pp. 346-351, doi: 10.1109/IACC.2016.71.

[13] H.H. Aviles-Arriaga, L.E. Sucar-Succar, C.E. Mendoza-Duran, L.A. Pineda-Cortes Avilés-Arriaga, Héctor & Sucar, Luis & Mendoza-Durán, C.E. & Pineda, Luis. (2011). A Comparison of Dynamic Naive Bayesian Classifiers and Hidden Markov Models for Gesture Recognition. Journal of applied research and technology. 9. 81-102. 10.22201/icat.16656423.2011.9.01.453.

[14] Anup Nandy, Jay Shankar Prasad, Soumik Mondal, Pavan Chakraborty, Gora Chand Nandi Recognition of Isolated Indian Sign Language Gesture in Real Time. In: Das,V.V., *et al.* Information Processing and Management. BAIP 2010.Communications in Computer and Information Science, vol 70.

[15] G. Jayadeep, N. V. Vishnupriya, V. Venugopal, S. Vishnu and M. Geetha, "Mudra: Convolutional Neural Network based Indian Sign Language Translator for Banks," 2020 4th International Conference on Intelligent Computing and Control Systems (ICICCS), Madurai, India, 2020, pp. 1228-1232, doi: 10.1109/ICICCS48265.2020.9121144.