Innovative Project -

"American Sign Language Detection Using LSTM"

Institute of Engineering & Management (IEM).

Department of Computer Science & Engineering

Gurukul, Y-12, Block –EP, Sector-V, Salt Lake Electronics Complex
Kolkata-700091, West Bengal, India

NOVEMBER, 2023

Under the Guidance By-

Dr. Prithwineel Paul
Associate Professor,
Computer Science and Engineering,

Submitted By:-

Sumit Panja
(22021002002020)
Debopriya Dutta
(22021002002034)

ACKNOWLEDGMENT:

By receiving guidance and support from numerous well-wishers, a prolonged effort can result in success. We would like to take this opportunity to express our gratitude and appreciation to all of them. We owe a debt of gratitude to Dr. Moutushi Singh, the Head of the Computer Science and Engineering Department, for providing valuable support and guidance throughout the project implementation period. We extend our sincere gratitude to Associate Professor, Dr. Prithwineel Paul for his invaluable guidance and unwavering support throughout our journey in undertaking the project on American Sign Language Detection Using LSTM.

Our project guide, Associate Professor Dr. Prithwineel Paul, deserves our sincere thanks and appreciation for engaging in stimulating discussions, identifying problems associated with our project work, and guiding us throughout the project. The project meetings were very informative, and we are grateful for their encouragement, untiring guidance, and confidence in us. We are greatly indebted for their valuable guidance throughout our project. We also extend our thanks to all of the CSE department's staff members for their helpful advice, as well as the supporting staff for providing resources as and when needed.

This project would not have been possible without the collective dedication and collaboration of all involved. We appreciate the opportunity to contribute to the field of American Sign Language detection and are thankful for the guidance and mentorship provided by Dr. Prithwineel Paul.

TABLE OF CONTENTS

1. ABSTRACT

Sign language is a vital means of communication for individuals with hearing impairments, and automating the recognition of signs through machine learning techniques can enhance accessibility and communication for this community. This study explores the effectiveness of Long Short-Term Memory (LSTM) networks in capturing temporal dependencies inherent in sign language gestures. Developed under expert guidance, the project presents an innovative approach to American Sign Language (ASL) detection.

Addressing the communication needs of individuals with hearing impairments, this project investigates the automation of sign language recognition through machine learning, specifically employing Long Short-Term Memory (LSTM) networks.

Trained on a diverse dataset, the LSTM-based system exhibits proficiency in recognizing a broad spectrum of sign language expressions. This implementation signifies a step forward in leveraging machine learning to improve communication accessibility for the hearing-impaired community.

2. INTRODUCTION

2.1. INTRODUCTION

Machine learning and deep learning are crucial aspects of computer technology and artificial intelligence, reducing human effort in various fields such as recognition, learning, and prediction. Sign language, a visual and spatial language relying on hand movements, facial expressions, and body postures, plays a crucial role in facilitating communication for individuals with hearing impairments.

Detecting and interpreting these intricate gestures are pivotal for bridging communication gaps between those who use sign language and those who do not. This study explores the application of Long Short-Term Memory (LSTM) networks, a type of recurrent neural network (RNN), for the automatic recognition of sign language.

By investigating the efficacy of LSTM in capturing the temporal dependencies inherent in sign language expressions, this research aims to contribute to the development of a sophisticated system capable of accurately interpreting the nuanced and diverse nature of American Sign Language (ASL). Such technological advancements hold the potential to enhance inclusivity and communication accessibility for the hearing-impaired community.

2.1.1 SIGN LANGUAGE DETECTION SYSTEM:

The Sign Language Detection system utilizes Long Short-Term Memory (LSTM) networks, a type of recurrent neural network, to automatically recognize and interpret American Sign Language (ASL) gestures. This system aims to bridge communication gaps for individuals with hearing impairments by leveraging the visual and spatial language of sign, including hand movements, facial expressions, and body postures.

2.2 PROBLEM STATEMENT:

Inefficient communication accessibility for the hearing-impaired community prompts the need for an automated Sign Language Detection system. Existing systems often lack precision in interpreting the nuanced and diverse gestures of American Sign Language (ASL), necessitating the development of a robust solution for accurate and real-time recognition.

3. UML DIAGRAMS

A UML diagram is a diagram based on the UML (Unified Modelling Language) with the purpose of visually representing a system along with its main actors, roles, actions, artifacts or classes, in order to better understand, alter, maintain, or document information about the system. UML is an acronym that stands for Unified Modelling Language. Simply put, UML is a modern approach to modelling and documenting software. In fact, it's one of the most popular business process modelling techniques. It is based on diagrammatic representations of software components. As the old proverb says: "a picture is worth a thousand words". By using visual representations, we are able to better understand possible flaws or errors in software or business processes.

3.1 CLASS DIAGRAM:

Class diagram model class structure and contents using design elements such as classes, packages and objects. Class diagram describes 3 perspectives when designing a system Conceptual, Specification, Implementation. Classes are composed of three things: name, attributes and operations. Class diagrams also display relations such as containment, inheritance, associations etc. The association relationship is most common relationship in a class diagram. The association shows the relationship between instances of classes. The purpose of class diagram is to model the static view of an application. Class diagrams are the only diagrams which can be directly mapped with object-oriented languages and thus widely used at the time of construction. UML diagrams like activity diagram, sequence diagram can only give the sequence flow of the application, however class diagram is a bit different. It is the most popular UML diagram in the coder community.

The purpose of the class diagram can be summarized as: –

- Analysis and design of the static view of an application.
- Describe responsibilities of a system.
- Base for component and deployment diagrams.
- Forward and reverse engineering.

Class diagrams, a fundamental part of UML (Unified Modeling Language), offer several advantages:

Visualization: Class diagrams provide a visual representation of the structure and relationships within a system, making it easier to understand complex systems at a glance.

Abstraction: They help in abstracting complex systems, focusing on key entities (classes) and their interactions, simplifying the modeling process.

Communication: Class diagrams serve as a communication tool between stakeholders, enabling developers, designers, and non-technical stakeholders to discuss and understand system structure and behavior.

Blueprint for Code: They serve as a blueprint for code implementation, guiding developers in translating the design into executable code.

Analysis and Design: Class diagrams facilitate both system analysis and design, aiding in the identification of classes, their attributes, methods, and associations, contributing to a more systematic and organized development process.

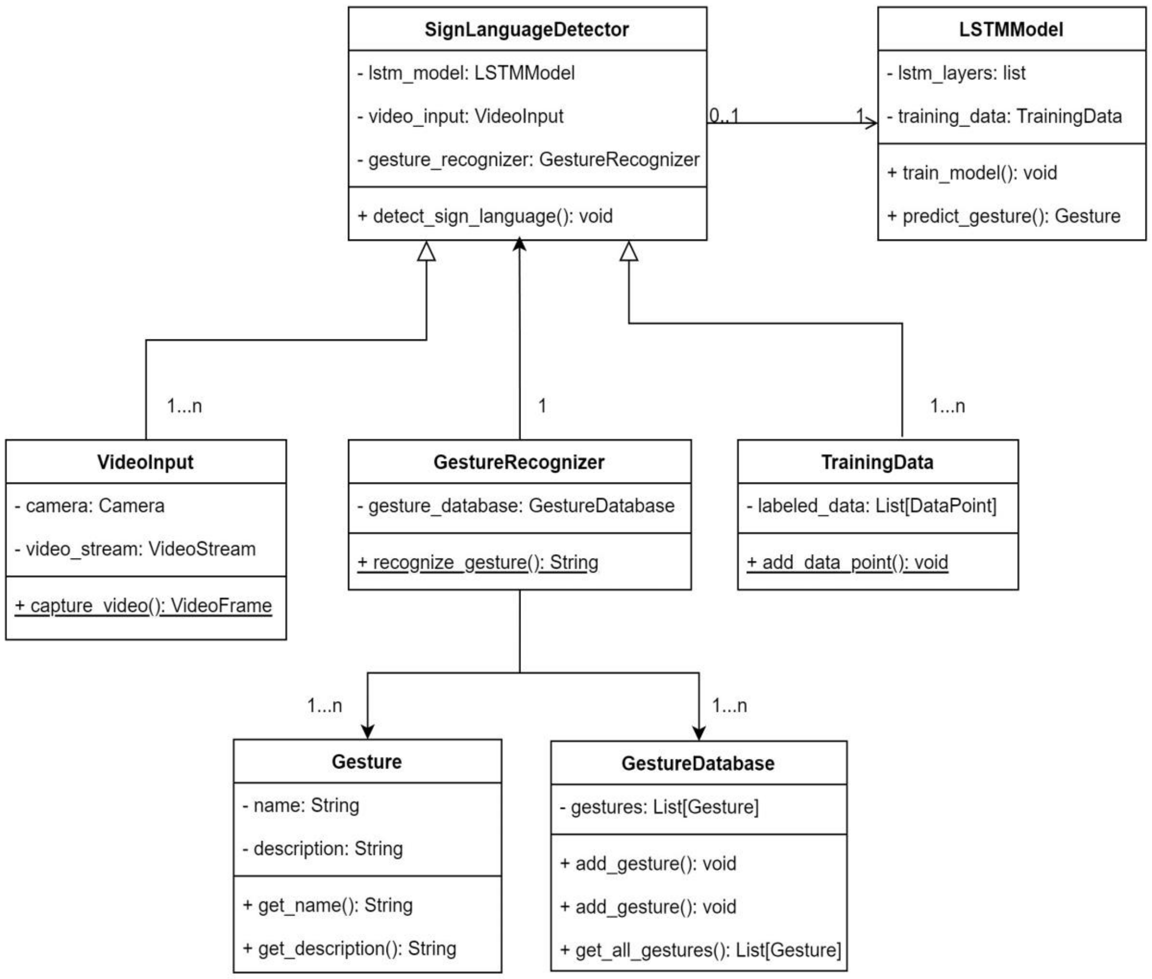

SignLanguageDetector
- lstm_model: LSTMModel
- video_input: VideoInput
- gesture_recognizer: GestureRecognizer
+ detect_sign_language(): void
LSTMModel
- lstm_layers: list
- training_data: TrainingData
+ train_model(): void
+ predict_gesture(): Gesture
0..1
1
1...n
1
1...n
VideoInput
- camera: Camera
- video_stream: VideoStream
+ capture_video(): VideoFrame
GestureRecognizer
- gesture_database: GestureDatabase
+ recognize_gesture(): String
TrainingData
- labeled_data: List[DataPoint]
+ add_data_point(): void
1...n
1...n
Gesture
- name: String
- description: String
+ get_name(): String
+ get_description(): String
GestureDatabase
- gestures: List[Gesture]
+ add_gesture(): void
+ add_gesture(): void
+ get_all_gestures(): List[Gesture]

Figure 1 Class Diagram

3.2 USE CASE DIAGRAM:

A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. Component diagrams are used to describe the components and deployment diagrams shows how they are deployed in hardware. UML is mainly designed to focus on the software artifacts of a system. However, these two diagrams are special diagrams used to focus on software and hardware components. Most of the UML diagrams are used to handle logical components of the system.
A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well. The use cases are represented by either circles or ellipses. A use-case diagram can help provide a higher-level view of the system. Use-Case provide the simplified and graphical representation of what the system must actually do.
In software and systems engineering, a use case is a list of actions or event steps typically defining the interactions between a role known in the Unified Modelling Language (UML) as an actor and a system to achieve a goal. The actor can be a human or other external system. In systems engineering, use cases are used at a higher level than within software engineering. The detailed requirements may then be captured in the Systems Modelling Language. Use case analysis is an important and valuable requirement analysis technique that has been widely used in modern software engineering.

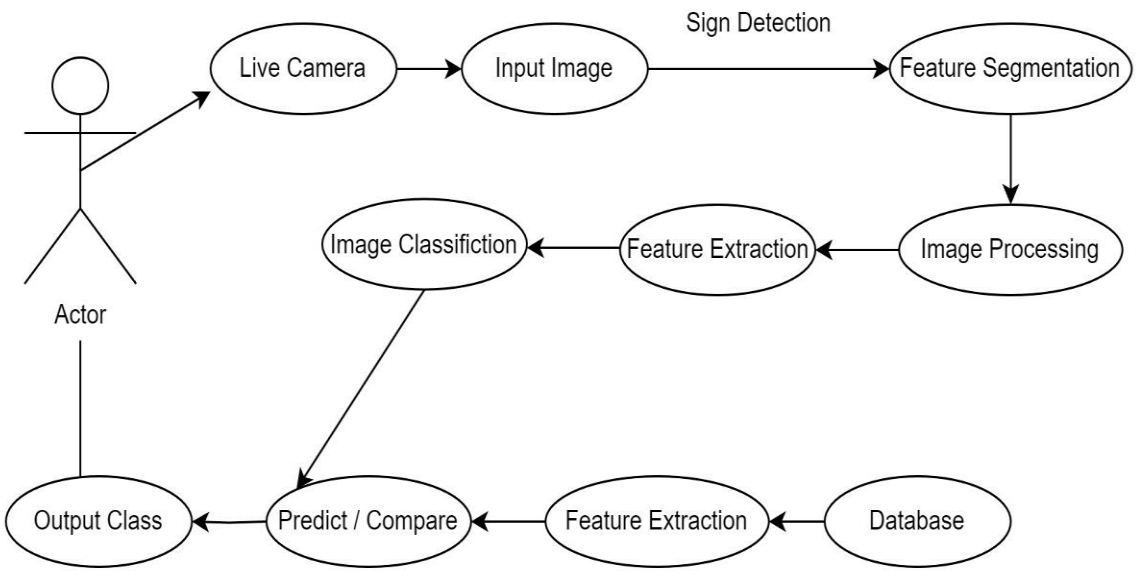


Figure 2 Use Case Diagram

4. METHODOLOGY

4.1 BASIC STEPS IN CONSTRUCTING A MACHINE LEARNING MODEL:

4.1.1 DATA COLLECTION:

- Collect a diverse dataset of American Sign Language (ASL) gestures. This involves recording video data capturing various hand movements, facial expressions, and body postures that represent the nuances of ASL communication.

4.1.2 DATA PRE-PROCESSING:

- Clean and preprocess the collected data to ensure consistency and eliminate noise. This may involve tasks such as resizing images, normalizing pixel values, and extracting relevant features from video frames.

4.1.3 MODEL DEVELOPMENT:

- Implement a Sign Language Detection model using Long Short-Term Memory (LSTM) networks. LSTM networks are particularly effective in capturing temporal dependencies inherent in sign language gestures. Train the model on the preprocessed dataset to enable it to recognize and interpret ASL gestures..

4.1.4 GESTURE RECOGNITION ALGORITHMS:

- Gesture recognition algorithms analyze preprocessed sensor data to identify and classify sign language gestures. Machine learning algorithms like Hidden Markov Models (HMMs) and Support Vector Machines (SVMs) are commonly used for gesture recognition.

4.1.5 SENSOR DATA ACQUISITION:

- Data gloves capture sensor data, typically in the form of acceleration, gyroscope, or flex sensor readings, representing hand movements and finger flexions.

4.1.6 VALIDATION AND OPTIMIZATION:

- Validate the model's performance using a separate dataset not used during training. Fine-tune the model parameters and architecture based on validation results to optimize its accuracy and generalization capabilities.

4.1.7 USER INTERFACE INTEGRATION:

- Integrate the Sign Language Detection system with a user interface that allows users to interact with the system in real-time. This interface may include features such as video input, gesture recognition, and visual feedback to enhance the user experience.

4.2 Methodologies for Sign Language Detection System:

4.2.1 Hybrid Approaches:

- Hybrid approaches combine vision-based and data glove-based techniques to leverage the strengths of both modalities. Vision-based techniques provide visual context, while data glove-based techniques offer precise hand movement data.

4.2.1.1 Hybrid Approaches:

- : Data fusion techniques combine visual features and sensor data to provide a more comprehensive representation of sign language gestures.

4.2.1.2 Context-Aware Recognition:

- Context-aware recognition algorithms incorporate contextual information, such as facial expressions and body posture, to enhance sign language detection accuracy.

4.2.2 Continuous Sign Language Recognition (CSLR):

Continuous Sign Language Recognition (CSLR) focuses on recognizing sign language gestures in real-time, enabling seamless communication between deaf individuals and hearing individuals.

4.2.2.1 Segmentation and Tracking:

- Segmentation algorithms identify and isolate individual sign language gestures in a continuous video stream. Tracking algorithms follow the hands and fingers to maintain gesture detection throughout the video.

4.2.2.2 Real-time Classification:

- Real-time classification algorithms process and classify sign language gestures with minimal latency, enabling real-time translation and communication.

4.2.2.3 Language Modelling:

- Language modelling techniques incorporate linguistic knowledge to improve the accuracy of CSLR, especially in handling ambiguous or co-articulated gestures.

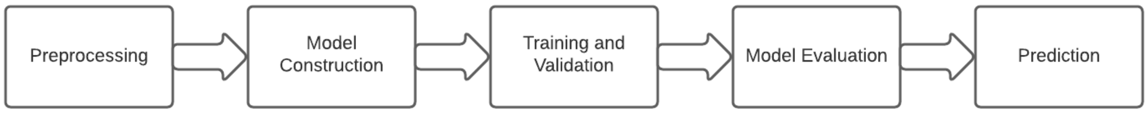


Figure 3 Steps in System development

4.2.3 IMPORT THE LIBRARIES

Libraries required are cv2, typing, numpy (np), tkinter, PIL (Image,ImageTk), mltu.inferenceModel(OnnxInferenceModel), mltu.utils.text_utils (ctc_decoder), tkinter.filedialog, mltu.configs (BaseModelConfigs).

I. **CV2 (OPENCV):**
OpenCV, imported as cv2, is a popular library used for computer vision and image processing tasks. In this code, it's employed for reading images (cv2.imread) and resizing them (cv2.resize).

II. **NUMPY (NP):**
NumPy (np) is a fundamental library for numerical operations in Python, particularly useful for handling arrays and matrices. Here, it's likely utilized for handling image data and arrays.

III. **MEDIAPIPE:**
MediaPipe, a Google-developed open-source framework, facilitates real-time multimedia tasks with modular pipelines, making it popular for applications like face detection and pose estimation..

IV. **TENSORFLOW:**
TensorFlow is an open-source machine learning framework developed by Google, widely used for building and training deep learning models. It provides a flexible platform for both beginners and experts to deploy scalable and efficient machine learning solutions.

V. **MATPLOTLIB:**
Matplotlib is a widely used Python library for creating static, animated, and interactive visualizations. It provides a flexible and customizable platform for generating plots, charts, and graphs with ease.

VI. **OS:**

The operating system (OS) is a fundamental software that manages computer hardware, providing a user interface and facilitating communication between applications and the computer's underlying hardware components. It serves as a crucial intermediary, enabling efficient resource allocation and execution of tasks.

VII. **SEQUENTIAL LIBRARY:**

The Sequential library in Python is a lightweight, intuitive tool for creating and managing sequential workflows, simplifying the development of data pipelines and task dependencies. It emphasizes simplicity and readability, making it suitable for a range of applications.

VIII. **SCIKIT-LEARN (SKLEARN):**

Scikit-learn (sklearn) is a widely-used machine learning library in Python, offering simple and efficient tools for data analysis and modeling, including various algorithms for classification, regression, clustering, and more. It provides a user-friendly interface for implementing and evaluating machine learning models.

IX. **LSTM:**

The "Keras" library in Python provides an implementation of Long Short-Term Memory (LSTM) networks, a type of recurrent neural network (RNN), for sequence modeling and time series prediction. LSTM layers can be easily incorporated into neural network architectures using the high-level API offered by Keras.

X. **DENSE :**

Dense is a Python library for deep learning, offering a high-level API for building neural networks with a focus on simplicity and flexibility. It provides an intuitive interface for creating and training various neural network architectures.

5. LSTM ARCHITECTURE

Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) architecture designed to address the vanishing gradient problem, allowing for more effective learning of long-term dependencies in sequential data. Here's a brief overview:

5.1 MEMORY CELL:

LSTMs have memory cells that store information over extended periods. These cells can retain or discard information based on the relevance of the input.

5.2 THREE GATES:

LSTMs incorporate three gates:

5.2.1 INPUT GATE: Regulates the flow of information into the memory cell.

5.2.2 FORGET GATE: Controls what information should be discarded from the memory cell.

5.2.3 OUTPUT GATE: Determines the output based on the current input and the content of the memory cell.

5.3 CELL STATE AND HIDDEN STATE:

LSTMs maintain a cell state, representing the long-term memory, and a hidden state, representing the short-term memory. These states are updated and passed to the next time step.

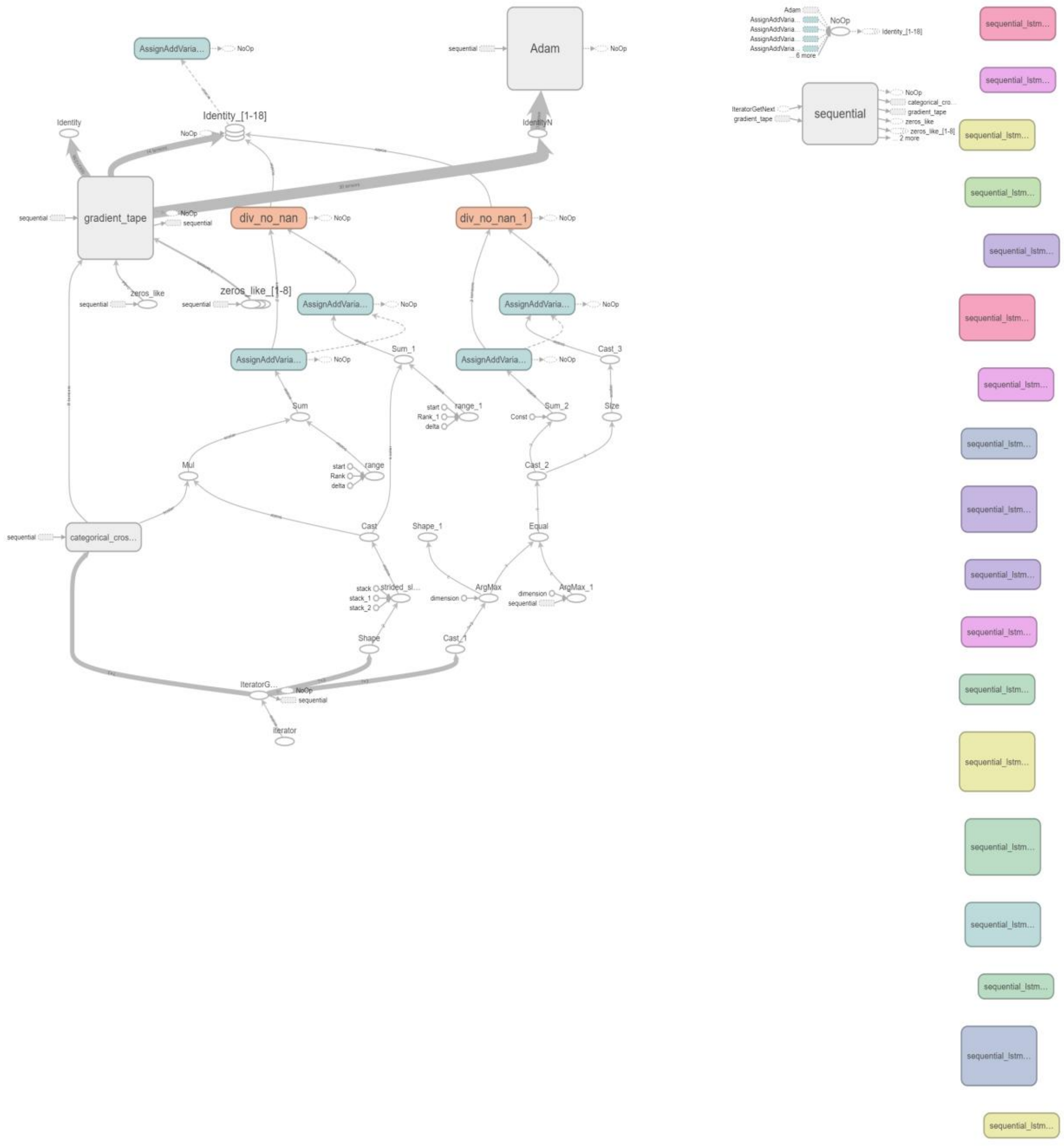
5.4 SEQUENTIAL LEARNING:

LSTMs excel at sequential learning tasks, making them suitable for applications involving time series data, natural language processing, and, as in this context, recognizing sequential patterns in sign language gestures.

5.5 APPLICATION:

LSTMs are widely used in various applications such as speech recognition, language translation, and time-series prediction, where capturing temporal dependencies is crucial.

5.6 LSTM NEURALNETWORK MODEL :



6. TRAINING PROCESS

The training process for a machine learning model, such as one using LSTM architecture for sign language detection, involves several key steps:

6.1 DATA PREPARATION:

Gather a diverse dataset of American Sign Language (ASL) gestures. This dataset should include various hand movements, facial expressions, and body postures representative of ASL communication.

6.2 DATA PREPROCESSING:

Clean and preprocess the collected data to ensure consistency and eliminate noise. This may involve resizing images, normalizing pixel values, and extracting relevant features from video frames.

6.3 MODEL ARCHITECTURE DESIGN:

Design the architecture of the LSTM-based model. Define the number of layers, nodes in each layer, and the structure of the LSTM cells. This architecture is crucial for the model's ability to capture temporal dependencies.

6.4 LOSS FUNCTION AND OPTIMIZATION:

Choose an appropriate loss function that quantifies the difference between the predicted and actual values. Select an optimization algorithm, such as stochastic gradient descent (SGD), to minimize the loss during training.

6.5 TRAINING THE MODEL:

Feed the preprocessed data into the LSTM model and iteratively update the model's parameters (weights and biases) to minimize the loss. This is typically done through backpropagation, where the gradient of the loss with respect to the parameters is calculated and used to update the model.

6.6 VALIDATION:

Assess the model's performance on a separate validation dataset not used during training. This helps ensure the model generalizes well to new, unseen data.

6.7 HYPERPARAMETER TUNING:

Fine-tune hyperparameters, such as learning rate and batch size, based on the validation results. This optimization step improves the model's accuracy and generalization capabilities.

6.8 EVALUATION:

Evaluate the trained model on a test dataset to assess its performance in real-world scenarios. This step provides insights into the model's effectiveness in recognizing ASL gestures.

6.9 MODEL INTERPRETABILITY:

Enhance the interpretability of the trained model by analyzing its decision-making process. This involves understanding which features or patterns contribute most to predictions, aiding in the transparency of the model's functionality.

6.10 AUGMENTATION TECHNIQUES:

Implement data augmentation techniques during training to artificially increase the diversity of the dataset. This can involve random rotations, flips, and zooms, contributing to the model's robustness.

6.11 EARLY STOPPING:

Incorporate early stopping mechanisms to halt the training process when the model's performance on the validation set ceases to improve, preventing overfitting.

7. EXPERIMENTAL ANALYSIS AND RESULTS

7.1 SYSTEM CONFIGURATION:

7.1.1 SOFTWARE REQUIREMENTS:

Operating system: windows 10/11.

IDE: Jupyter Notebook.

Python: Python is an interpreted, high-level, general purpose programming language created by Guido Van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant Whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects. Python is dynamically typed and garbage collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming.

Jupyter Notebook: A Jupyter Notebook is an open-source, interactive computing environment that allows users to create and share documents containing live code, equations, visualizations, and narrative text. It supports multiple programming languages, including Python, R, and Julia. Notebooks are organized into cells, each of which can contain code or markdown text. Users can execute code cells individually, see the output inline, and create a dynamic and exploratory workflow. Jupyter Notebooks are widely used for data analysis, machine learning, research, and education due to their versatility and user-friendly interface.

7.1.2 HARDWARE REQUIREMENTS:

Processor: Intel i5 9th gen or Higher.

RAM: Minimum of 8GB or higher.

HDD/SSD: 256 GB or higher.

Monitor: 15’’ or 17’’ colour monitor.

Mouse: Optical mouse.

Keyboard: Standard 110 keys keyboard.

Webcam: Accessing Camera

7.2 LEARNING CURVES:

The learning curves for the Sign Language Detection project employing LSTM for sign language interpretation and frame keypoints for action recognition provide a comprehensive depiction of the models' training and validation progress. These curves visually represent how the models evolve in terms of accuracy, precision, and recall over successive iterations, offering valuable insights into their learning dynamics. By observing these curves, one can discern patterns, identify convergence points, and assess the overall performance trajectory of the models. Additionally, learning curves aid in diagnosing potential issues such as overfitting or underfitting, facilitating informed decisions for refining the models and enhancing their effectiveness in the context of sign language detection and action recognition.

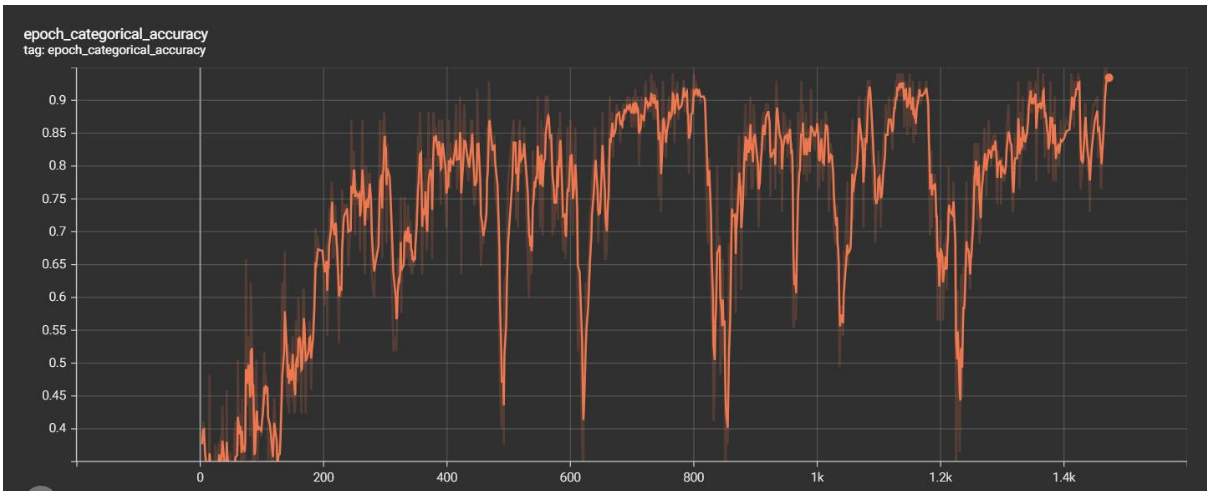
We often see these two types of learning curves appearing in charts:

- *Optimization Learning Curves*: Learning curves calculated on the metric by which the parameters of the model are being optimized, such as loss or Mean Squared Error
- *Performance Learning Curves*: Learning curves calculated on the metric by which the model will be evaluated and selected, such as accuracy, precision, recall, or F1 score.

validation has increased with increase in number of epochs and loss has been subsequently decreases during training and validation.

7.2.1 ACCURACY CURVE:

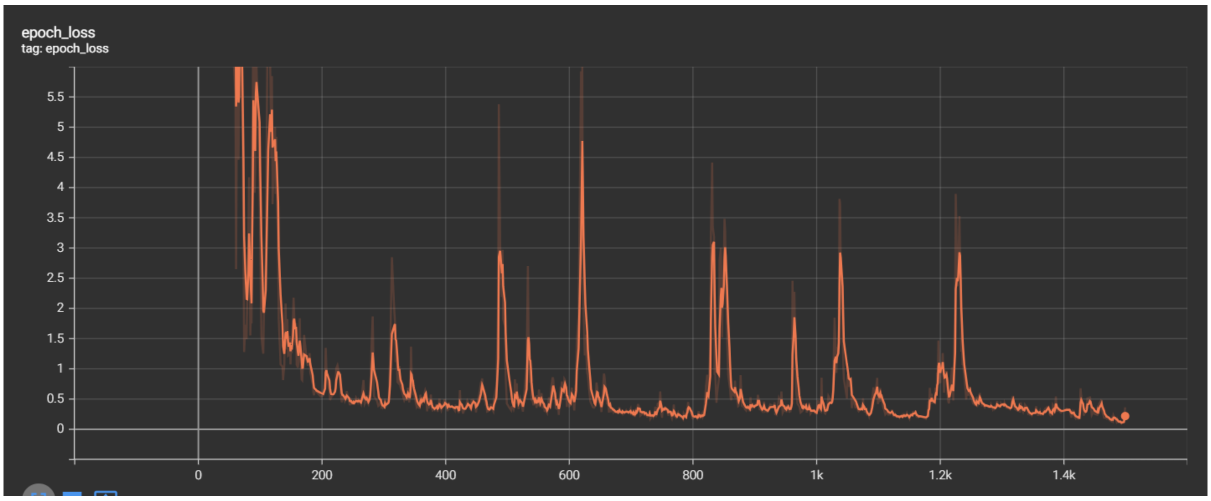
The accuracy curve in the context of sign language detection employing LSTM and action recognition based on frame keypoints serves as a metric for evaluating the classification model's performance. Expressed as a percentage, accuracy signifies the proportion of correct predictions, comparing predicted values with true values in a binary (true/false) context for individual samples. Throughout the training phase, accuracy is graphed and monitored, offering a straightforward interpretation of the model's effectiveness, with a higher percentage indicating better performance. This metric is particularly valuable in assessing the overall or final accuracy of the model, providing a user-friendly measure during the evaluation process.



7.2.2 Loss Curve:

In the context of sign language detection using LSTM and action recognition based on frame keypoints, the loss curve serves as a critical metric. Also referred to as a cost function, it accounts for prediction uncertainties and variations from true values. Unlike accuracy, which is presented as a percentage, loss represents the cumulative errors across training or validation samples. This metric is pivotal in refining model parameters, such as neural network weights, during the training process, aiming to minimize its value. Common loss functions include log loss, cross-entropy loss, mean squared error, and likelihood loss, applicable to both classification and regression problems.

The loss curve, a prominent diagnostic tool for neural networks, provides insights into the training process, showcasing the model's learning trajectory and facilitating debugging efforts.



7.3 Sample Code:

7.3.1 Setup Mediapipe Holestic Model

```
import mediapipe as mp

import tensorflow as tf

mp_holistic = mp.solutions.holistic # Holistic Model

mp_drawing = mp.solutions.drawing_utils # Drawing Utilities
```

7.3.2 Setup Mediapipe Detection Function

```
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 CONVERSION RGB 2 BGR

    return image, results
```

7.3.3 Set Landmarks

```
def draw_landmarks(image, results):

    # Draw face landmarks

    mp_drawing.draw_landmarks(image,results.face_landmarks,
mp.solutions.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.solutions.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.solutions.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.solutions.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))
```

7.3.4 Capture Video Feedback

```
# Set up mediapipe holistic model

mp_holistic = mp.solutions.holistic

holistic = mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5)

# Set up mediapipe drawing utils

mp_drawing = mp.solutions.drawing_utils

# Your video capture setup

cap = cv2.VideoCapture(1)

while cap.isOpened():

    ret, frame = cap.read()

# Make detections

    image, results = mediapipe_detection(frame, holistic)

# Draw landmarks on the frame

    draw_landmarks(image, results)

# Display the frame or perform other actions

    cv2.imshow('MediaPipe Holistic', image)
    # Check for the 'q' key to exit the loop

    if cv2.waitKey(10) & 0xFF == ord('q'):

        break

# Release the video capture object and close any open windows

cap.release()

cv2.destroyAllWindows()
```

7.3.5 EXTRACT KEYPOINTS VALUES

```
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(334)

    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(213)

    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(213)

    face = np.array([[res.x, res.y, res.z] for res in results.face_landmarks.landmark]).flatten() if results.face_landmarks else np.zeros(4683)

    return np.concatenate([pose, rh, lh, face])
```

7.3.6 SETUP FOLDERS FOR COLLECTION

```
import os

import numpy as np

DATA_PATH = os.path.join('MP_Data') # Path for exported data, numpy array

# Actions that we try to detect

actions = np.array(['hello', 'thanks', 'iloveyou'])

# Thirty Videos worth of data

no_sequences = 30

# Videos are going to be 30 frames in length

sequence_length = 30
```

```
import os
for action in actions:
    for sequence in range(no_sequences):
        try:
            os.makedirs(os.path.join(DATA_PATH, action, str(sequence)))
        except:
            pass
```

7.3.7 COLLECT KEYPOINT VALUES FOR TRAINING AND TESTING

```
# Set up mediapipe holistic model
mp_holistic = mp.solutions.holistic
holistic = mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5)
# Set up mediapipe drawing utils
mp_drawing = mp.solutions.drawing_utils
# Your video capture setup
cap = cv2.VideoCapture(1)
# Loop through actions
for action in actions:
    # Loop through sequence aka videos
    for sequence in range(no_sequences):
        # Loop through video length aka sequence length
        for frame_num in range(sequence_length):
            # Read the feed
            ret, frame = cap.read()
            # Make detections
            image, results = mediapipe_detection(frame, holistic)
            # Draw landmarks on the frame
            draw_landmarks(image, results)
            # 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)
                # Display to Screen
                cv2.imshow('MediaPipe Holistic', image)
                cv2.waitKey(2000)  # Wait for 2 seconds
            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)
                # Display the frame or perform other actions
                cv2.imshow('MediaPipe Holistic', image)
            # Export keypoints
            keypoints = extract_keypoints(results)
            npy_path = os.path.join(DATA_PATH, action, str(sequence), str(frame_num))
            np.save(npy_path, keypoints)


            # Save the frame as an image (change format as needed)
            img_path = os.path.join(DATA_PATH, action, str(sequence), f"{frame_num}.png")
            cv2.imwrite(img_path, cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
            # Check for the 'q' key to exit the loop
            if cv2.waitKey(10) & 0xFF == ord('q'):
```

```
        break
# Release the video capture object and close any open windows
cap.release()
cv2.destroyAllWindows()
```

7.3.8 Preprocess Data and Create Lables and Features

```
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
label_map = {label:num for num, label in enumerate(actions)}
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])
```

7.3.9 Build and Train LSTM Neural Network

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.callbacks import TensorBoard
import os
log_dir = os.path.join('Logs')
tb_callback = TensorBoard(log_dir=log_dir) # Monitor neuro network training
```

7.3.10 Build Neuro Network Architecture

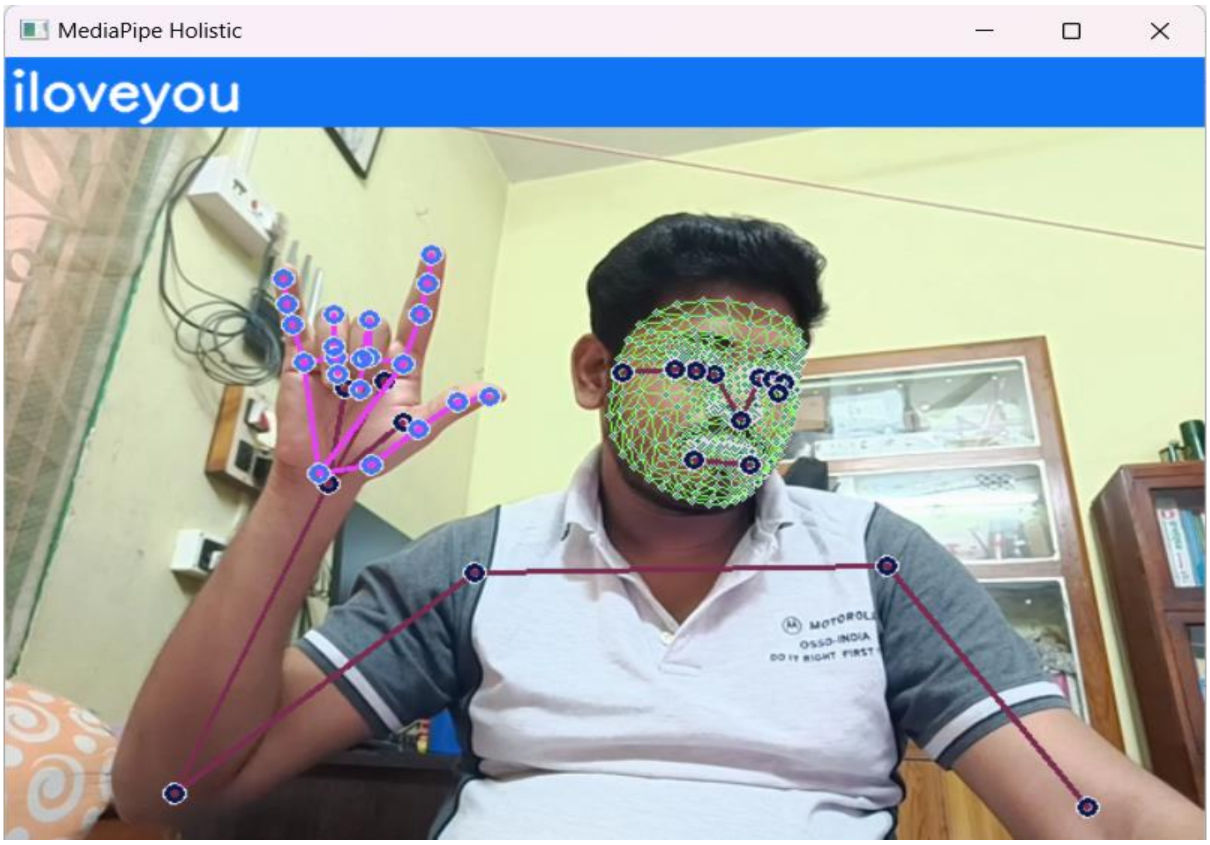
```
model = Sequential()

model.add(LSTM(64, return_sequences=True, activation='relu', input_shape=(30,1662))) # 64 LSTM units

model.add(LSTM(128, return_sequences=True, activation='relu')) # 128 LSTM units

model.add(LSTM(64, return_sequences=False, activation='relu')) # 64 LSTM units

model.add(Dense(64, activation='relu')) # 64 Dense units

model.add(Dense(32, activation='relu')) # 32 Dense units

model.add(Dense(actions.shape[0], activation='softmax'))

model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['categorical_accuracy'])

model.fit(X_train, y_train, epochs=2000, callbacks=[tb_callback])
```

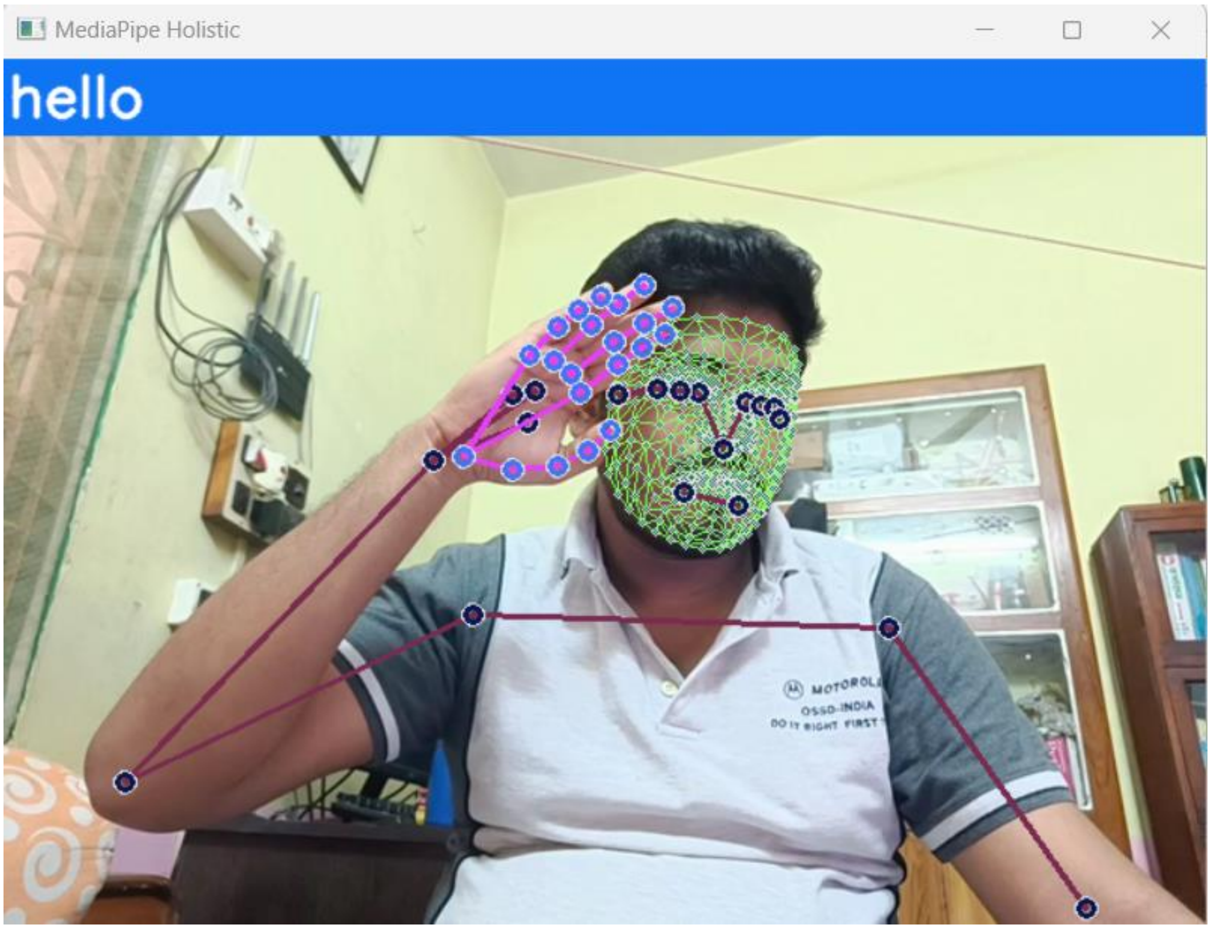
7.3.11 TEST IN REAL TIME

```
# Set up mediapipe holistic model

mp_holistic = mp.solutions.holistic

holistic = mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5)

# Set up mediapipe drawing utils

mp_drawing = mp.solutions.drawing_utils

# New detection variables

sequence = []

sentence = []

predictions = []

threshold = 0.5

# Your video capture setup

cap = cv2.VideoCapture(1)

# Initialize res outside the loop

res = None
```

```
# Initialize predicted_action before the loop
predicted_action = ""
while cap.isOpened():
    ret, frame = cap.read()  # Read feed
    # Make detections
    image, results = mediapipe_detection(frame, holistic)
    print(results)
    # Draw landmarks on the frame
    draw_landmarks(image, results)
    # Prediction Logic
    keypoints = extract_keypoints(results)
    sequence.append(keypoints)
    sequence = sequence[-30:]
    if len(sequence) == 30:
        res = model.predict(np.expand_dims(sequence, axis=0))[0]
        predicted_action = actions[np.argmax(res)]
        print(predicted_action)
        predictions.append(np.argmax(res))
        # Viz Logic
        if np.unique(predictions[-10:])[0]==np.argmax(res):
            if res is not None and res[np.argmax(res)] > threshold:
                if len(sentence) > 0:
                    if predicted_action != sentence[-1]:
                        sentence.append(predicted_action)
                else:
                    sentence.append(predicted_action)
        if len(sentence) > 5:
            sentence = sentence[-5:]
    cv2.rectangle(image, (0, 0), (640, 40), (245, 117, 16), -1)
    cv2.putText(image, predicted_action, (3, 30),
                cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)
    # Display the frame or perform other actions
    cv2.imshow('MediaPipe Holistic', image)
    # Check for the 'q' key to exit the loop
    if cv2.waitKey(10) & 0xFF == ord('q'):
        break
# Release the video capture object and close any open windows
cap.release()
cv2.destroyAllWindows()
```

7.4 OUTPUTS:





8. CONCLUSION AND FUTURE WORK

8.1 CONCLUSION:

Our project SIGN LANGUAGE DETECTION deals with identifying the sign language recognition and detection. The main goals of this project is to develop a robust American Sign Language Detection System using LSTM, aiming to enhance communication accessibility for individuals with hearing impairments by automating the recognition and interpretation of sign language gestures.

The implemented system demonstrates the effectiveness of LSTM in accurately recognizing and interpreting a diverse range of sign language gestures. Through this project, we contribute to bridging communication gaps and fostering inclusivity for the hearing-impaired community. The automated sign language detection system holds promise for real-world applications, offering a reliable and efficient means of communication through advanced machine learning techniques.

In this project, different deep learning methods, which are CNN (Convolutional Neural Networks) and RNN (Recurrent Neural Network), architectures are used to achieve high performance on the language detection and recognition problem. LSTM is a type of recurrent neural network (RNN) that is well-suited for tasks involving sequential data.

8.2 FUTURE WORK:

The proposed system already predict Real-Time Implementation input. With enhancements to the dataset and model, the system can evolve into a Sign Language Learning System. Users can then learn special sign language characters and the language for effective communication with others. Collaborate with sign language experts and the deaf community for ongoing feedback and refinement, ensuring the system aligns with the needs and preferences of its users. Later, develop a mobile application that incorporates the sign language detection system, making it easily accessible and portable for users on smartphones and tablets.

9.REFERENCES

[1] https://www.tensorflow.org/learn

[2] https://scikit-learn.org/stable/

[3] https://developers.google.com/mediapipe

[4] https://opencv.org/

[5] https://www.geeksforgeeks.org/deep-learning-introduction-to-long-short-term-memory/

[6] https://www.mdpi.com/2079-9292/11/11/1780

[7] https://ieeexplore.ieee.org/document/9823484

[8] https://www.mdpi.com/2079-9292/11/11/1780

[9] https://www.researchgate.net/publication/360853648_Sign_language_identification_and_recognition_A_comparative_study

[10] https://www.researchgate.net/publication/369857149_Sign_Language_Recognition_Application_Using_LSTM_and_GRU_RNN

[11] https://www.researchgate.net/publication/339024531_Real-Time_Translation_of_Indian_Sign_Language_using_LSTM

[12] https://ieeexplore.ieee.org/document/10080705/

[13] https://arxiv.org/abs/1505.04868

[14] https://ieeexplore.ieee.org/document/9620901/

[15] https://arxiv.org/abs/2111.08557