SIGN LANGUAGE TRANSLATION TO AUDIO AND TEXT USING DENSE AND LSTM LAYERS

A PROJECT REPORT BY

Submitted by

SIDDHARTH GANESH
SHUIB AKHTHAR.S
VIKRAM RAMANATHAN

in partial fulfilment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

EASWARI ENGINEERING COLLEGE (AUTONOMOUS), CHENNAI
ANNA UNIVERSITY :: CHENNAI 600 025
JUNE 2022

EASWARI ENGINEERING COLLEGE, CHENNAI

(AUTONOMOUS INSTITUTION)

AFFILIATED TO ANNA UNIVERSITY, CHENNAI 600025

BONAFIDE CERTIFICATE

Certified that this project report **"SIGN LANGUAGE TRANSLATION TO AUDIO AND TEXT USING DENSE AND LSTM LAYERS"** is the bonafide work of "SIDDHARTH GANESH (310618104095), SHUIB AKHTHAR (310618104094), VIKRAM RAMANATHAN (310618104116)" who carried out the project work under my supervision.

CERTIFICATE OF EVALUATION

College Name : Easwari Engineering College (Autonomous)

Branch & Semester : Computer Science and Engineering & VIII

S. No	Name of the Students	Title of the Project	Name of the Supervisor with Designation
1.	SHUIB AKHTHAR.S	SIGN LANGUAGE TRANSLATION TO AUDIO AND TEXT USING DENSE AND LSTM LAYERS	Mr.P.Hari Kumar, M.E,(Ph.D). Associate Professor
2.	SIDDHARTH GANESH		
3.	VIKRAM RAMANATHAN		

The report of project work submitted by the above students in partial fulfilment for the award of Bachelor of Engineering Degree in Computer Science and Engineering of Anna University were evaluated and confirmed to be a report of the work done by the above students.

The viva voce examination of the project work was held on 22/06/2022.

Internal Examiner **External Examiner**

ACKNOWLEDGEMENT

We hereby place our deep sense of gratitude to our beloved Founder Chairman of the institution, **Dr.T.R.Pachamuthu, B.Sc., M.I.E**., for providing us with the requisite infrastructure throughout the course. We would also like to express our gratitude towards our Chairman **Dr.R.Shivakumar, M.D., Ph.D.** for giving the necessary facilities.

We convey our sincere thanks to **Dr.R.S.Kumar, M.Tech., Ph.D.** Principal Easwari Engineering College, for his encouragement and support. We extend our hearty thanks to **Dr.S.Nagarajan,M.E.,Ph.D.**, Vice Principal (Admin), for their constant encouragement.

We take the privilege to extend our hearty thanks to **Dr.G.S.Anandha Mala,M.S.,M.E.,Ph.D.**, Head of the Department, Computer Science and Engineering, Easwari Engineering College for her suggestions, support and encouragement towards the completion of the project with perfection.

We would like to express our gratitude to our Project Coordinator, **Dr. Ahmed Ali,M.E.,Ph.D.**, Assistant Professor, Department of Computer Science and Engineering, Easwari Engineering College, for his constant support and encouragement

We would also like to express our gratitude to our guide **Mr. P. Harikumar,M.E,(Ph.D)**, Associate Professor, Department of Computer Science and Engineering, Easwari Engineering College, for his constant support and encouragement.

Finally, we wholeheartedly thank all the faculty members of the Department of Computer Science and Engineering for warm cooperation and encouragement.

TABLE OF CONTENTS

ABSTRACT

Deaf-Mute people face many hurdles in accomplishing menial tasks in their everyday life. One of those many hurdles is the ability to communicate with fellow human beings with sign language. Unfortunately, only 0.2% of the population use and practice ASL (American Sign Language). This brings about a gap between Deaf-Mute people and people wanting to communicate with them.

The proposed system “Sign Language Translation to audio and text” aims to bridge this gap and hopes it will help Deaf-Mute people to engage in conversations with the general populace. The system can be used as a tool for non native speakers to get familiar with the language. Using Dense and LSTM layers the system detects hand motion by plotting keypoints on the hand with the help of mediapipe holistics. The sign is subsequently converted to audio and text.

List of Figures

List of Abbreviations

Serial No.	Abbreviation	Expansion
1	ASL	American Sign Language
2	LSTM	Long Short Term Memory
3	RNN	Recurrent Neural Network
4	RF	Radio Frequency
5	VPPN	view-based paired pooling network
6	ISL	Indian sign Language
7	SSD	Single Shot Detector
8	2DCNN	Two Dimensional Convolutional Neural Network
9	3DCNN	Three Dimensional Convolutional Neural Network
10	RGB	Red,Green,Blue
11	ESHR	Extra Spatial Hand Relation
12	ESHR	Extra Spatial Hand Relation
13	HP	Hand Pose
14	ROI	Region of Interest
15	SVM	Support Vector Machine
16	MLP	Multilayer Perceptron
17	ReLU	Rectified linear activation unit

CHAPTER 1

Introduction

1.1 General

Sign Languages were languages developed for deaf and/or mute people and it uses the visual-manual modality for communication. Most sign languages have the same linguistics as other spoken languages but the grammar differs from English. Sign languages are full fledged languages with their own grammar and lexicon. Today, there are a wide variety of sign languages from ASL (American Sign Language) to ISL (International Sign Language), however there is no universal sign language.

ASL is a language completely separate and distinct from English. It contains all the fundamental features of language, with its own rules for pronunciation, word formation, and word order. While every language has ways of signalling different functions, such as asking a question rather than making a statement, languages differ in how this is done. Just as with other languages, specific ways of expressing ideas in ASL vary as much as ASL users themselves. In addition to individual differences in expression, ASL has regional accents and dialects; just as certain English words are spoken differently in different parts of the country, ASL has regional variations in the rhythm of signing, pronunciation, slang, and signs used. Other sociological factors, including age and gender, can affect ASL usage and contribute to its variety, just as with spoken languages. Fingerspelling is part of ASL and is used to spell out English words. In the fingerspelled alphabet, each letter corresponds to a distinct handshape. Fingerspelling is often used for proper names or to indicate the English word for sign.

In this system, we will implement the translation of ASL to coherent audio and text, both in simple English. We will emphasise various parameters such as reliability, efficiency and ease of use. With a user-friendly interface, robust algorithms running on the backend, and a plethora of signs stored in the database, we aim to establish a complete package for the simple use of translating ASL.

1.2 Objective

To translate American Sign Language into readable english text and coherent speech using Long Short Term Memory (LSTM) algorithm that is trained along with keras Dense Neural Network.

1.3 Scope of the Project

This project will have a reach towards the deaf and mute community of society. The project will contain knowledge of ASL grammar and lexicon, in hopes for people with the disability to use the system in everyday life for ease of communication and people who are interested in learning sign language.

CHAPTER 2

Literature Survey

2.1 Introduction

A literature survey is a documentation of a comprehensive review of the published and unpublished work from secondary sources of data in the areas of specific interest to the researcher.

2.2 Related works

Sevgi Z. GurbuSevgi et al.[1]

Using an RF sensor (Xbox Kinect), the system uses minimum Redundancy Maximum Relevance (mRMR) algorithm for sign language recognition. The sensor uses RF signals to capture hand signs made by the subject.

Suneetha M et al.[2]

Suneetha M et al. used a multi-view motion based sign language detection system along with Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN). The signs are learned with the help of a motion attention network, which focuses on the components that are important in constructing a view-based paired pooling network (VPPN). For improved sign recognition, the VPPN pairs views to build a maximum distributed discriminating feature from all views. The average recognition for 32 signs with only front view testing was evaluated around 82.82%. The average recognition rate for four views, each with a different subject, was found to be 93.89 percent.

Ankita Wadhawan et al.[3]

Ankita Wadhawan et al formed a system for the recognition of the sign language that detects Indian Sign Language (ISL) Using Convolutional Neural Networks built on deep learning to recognize important differentiable hand signs (CNN). 100 static signs were collected from users with accuracy 98.7%. Precision, recall, and F-score have all been used to evaluate the proposed system's performance. A total of 50 convolutional neural network models were used to evaluate the created sign language recognition system. The network is trained for a maximum of 100 epochs using algorithms with multiple optimizers and a categorical cross-entropy loss function.

Razieh Rastgoo et al.[4]

Using Single Shot Detector (SSD), Two Dimensional Convolutional Neural Network (2DCNN), Three Dimensional Convolutional Neural Network (3DCNN), and Long Short-Term Memory (LSTM) from RGB input films, For effective automatic hand sign language detection, The researchers suggest a new pipeline design based on deep learning. The Three Dimensional hand keypoints are estimated from Two Dimensional input frames using a CNN-based model. Then, using the midpoint approach, we join the calculated key points to create the hand skeleton. The accuracy was evaluated to be 91.12% for 100 Persian signs.

Kourosh Kiani et al.[5]

A model using spatiotemporal hand based information was created by deep learning approaches for the recognition of sign language, from films, comprising SSD, CNN, and LSTM. Hand detection and sign recognition are the two primary components of our simple but effective and accurate model. Hand features, Extra Spatial Hand Relation (ESHR) features, and Hand Pose (HP) features are three types of spatial characteristics that had been joined into the model to input to LSTM for temporal feature extraction. They used movies from five online sign dictionaries to train the SSD model for hand detection. Their model is tested using the dataset they proposed which includes 10,000 sign films for 100 Persian signs.

Taniya Sahana et al. [6]

The authors make use of gestures to interact with computers and a detection approach based on multiscale density features is proposed. This research uses depth pictures of American Sign Language numerals yielding an identification rate of 92.8 % for ASL numbers (0-9), considering ROI of the test subjects hand (Region Of Interest). The data was tested against Random forest, SVM (Support Vector Machine), MLP (Multilayer Perceptron), each yielding 97.3%, 97.8%, 98.2% accuray.

Adithya V et al. [7]

The paper puts forward a mechanism for identification of the gestures of the hand, these are the key part of the vocabulary of sign language, based on a deep CNN architecture that is efficient. This article proposes to recognize static hand motions using a deep learning architecture derived from convolutional neural networks. The classic pattern recognition approach's lengthy and computationally demanding feature extraction phase is avoided with this model.

A five-fold cross validation is used to assess the classification's performance. There are 40 sample photographs of each gesture type in each of the five subsets of the dataset. The classifier gets trained on some of the 4 subgroups, with the subset which is left being utilised for the process of testing. This process was duplicated 5 times in the same way till every subgroup has been used for testing and development. Five-fold cross validation is used to assess the classification's performance. There are 40 sample photographs of each gesture type in each of the five subsets of the dataset.

The accuracy of the CNN Model on the NUS Hand Posture Dataset using Statistical Measures was 94.7 percent.

Sunitha Ravi et al[8].

For sign language recognition based on RGB-D, In this study, they use a 4 stream convolutional neural network (CNN) to implement a multi-modal feature sharing technique. They put forward a feature which shared multi stream CNN for sign language on multi-nodal data, Due to scale differences, it differs from multi-stream CNNs in that prediction of output class is focused on independently running two or three modal streams. The algorithm correctly detects 15 Malaysian sign languages with an accuracy of 80.54 percent. The system was put to the test on 50 varying signs, with an overall accuracy of 89.69 percent. Despite the fact that the Leap motion sensor reliably monitors both hands, it does not capture non-manual functions.

Katerina Papadimitrieu et al[9].

The system is divided into two algorithmic stages. The first stage, in particular, builds on previous work on the detection of hand and separation in the form of videos with the face of signers, enabling detection of face to deliver hand segmentation which is based on the skin tone and locate the hands by motion, and then broadening by a maximum detection module which generates regions of proposal which would harbour the signs of interest. Obtained areas are further identified using the convolutional neural network form which extends normal convolutions into quadratic functions on the inputs, which is the first time we've seen such architecture employed for this purpose. In the case of ASL signs which are static, alphabet signs are supplied to a modified CNN classifier. Both system stages outperform a variety of competing techniques on the fingerspelling corpora which are known to most, surpassing them significantly in the two signer independent and multi signer settings. In the multi-signer situation, the process of the LeakyReLU layer had shown the accuracy varying from 99.14 percent to 99.64 percent for the set of three, and from 73.92 percent to 79.23 percent in the case of signer independence.

2.3 Issues in existing system

Existing systems use external devices such as gloves, exoskeleton or sensors in input for the system. These devices are intrusive and cause inconvenience while using them.

2.4 Proposed system

The proposed system is a software which recognises the American sign language in real time with the help of image processing technique. Machine learning algorithms such as LSTM and Dense algorithms will be used to train the model with the help of our own dataset which consist of videos of different words. Input is obtained by the video camera of the User's system and the output is predicted with the help of the trained model and displayed as text.

2.5 Summary

This chapter covers the existing systems, their limitations and their applications that are used as references throughout the proposed system.

CHAPTER 3

System Design

3.1 Introduction

The design of the software depends on the meticulous planning and execution of the architectures involved in the functions along with the modular design used to run the complete system. The architecture diagram will show a bigger picture about the proposed system and will ensure proper step by step development takes place.

3.2 Functional Architecture

Functional Architecture is an architectural model that identifies system functions and their interactions. It defines how the functions will operate together to perform the system mission.

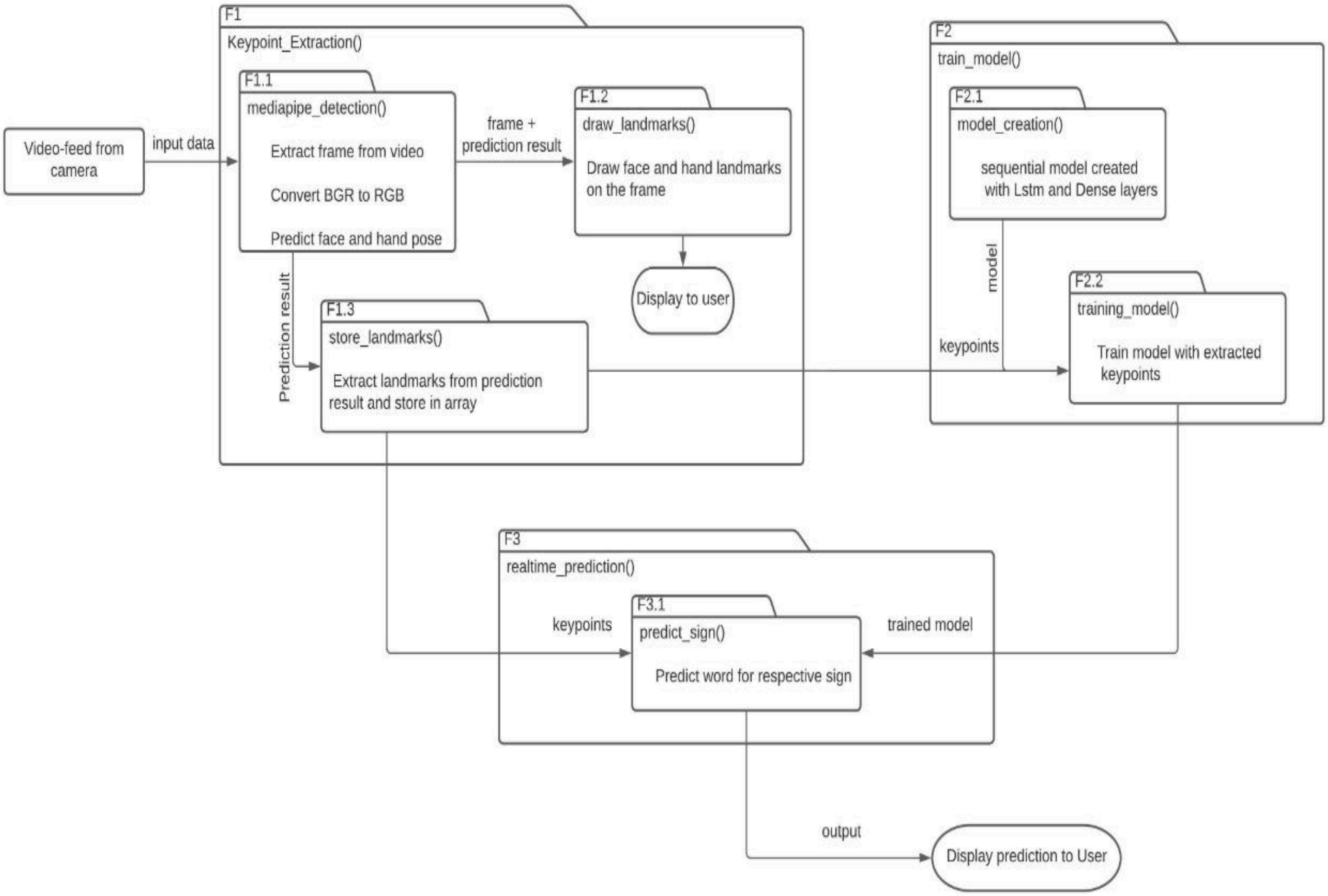

Fig 1: Functional architecture

The system takes in video-feed from the laptop camera and the input is used to extract keypoints. The funcion mediapipe_detection() extracts the frames from the video feed, converts the frame from BGR to RGB and then proceeds to predict face and hand pose. The frame along with the prediction result is sent over to draw landmarks on the frame. On the other hand, the prediction result is used to extract landmarks and store it in an array. These keypoints are then fed to create and train the model. Model is created with 3 LSTM and Dense layers and then the model is consequently trained. Finally, the model is tested for realtime prediction of ASL signs and the achieved accuracy is recorded.

3.3 Modular Design

3.3.1 Keypoint Extraction

Mediapipe uses a tracking approach similar to that used for standalone face and hand pipelines to simplify the identification of ROIs for face and hands. It assumes that the object does not move considerably between frames and uses the previous frame's assessment as a reference to the current frame's object region.

```
Function mediapipe_detection():
        Declare frame
        Input frame from video
        Convert frame from BGR to RGB
        Predict face_hand_pose

Function draw_landmarks():
        Get frame from input
        Get face_hand_pose from mediapipe_detection()
        Draw face_hand_pose into frame
        Display frame to User

Function store_landmark():
        Declare landmarks[]
        Intialize landmarks with face_hand_pose
        Store landmarks
```

Fig 2: Pseudocode for Keypoint Extraction

3.3.2 Train Model

The result of the prediction from Keypoint Extraction is used as the input for this module. This begins with the creation of a sequential model by adding 3 LSTM and 3 Dense layers. The model is then compiled with the extracted keypoints and the resulting accuracy is checked. If the accuracy is satisfactory, the training model is obtained, otherwise the process is repeated.

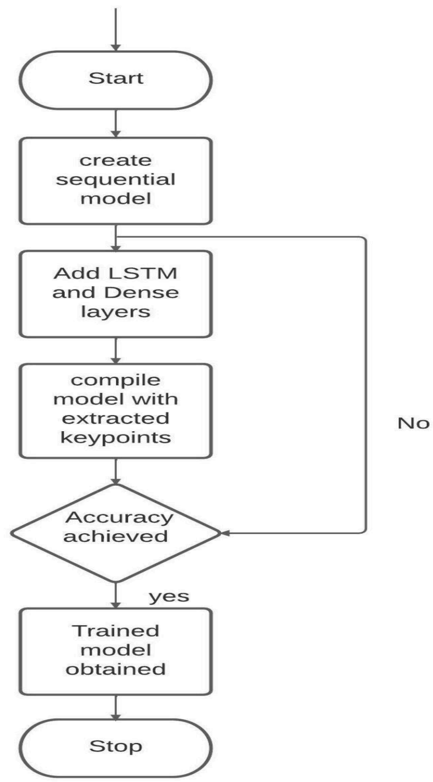

Fig 3: Flowchart for Train Model

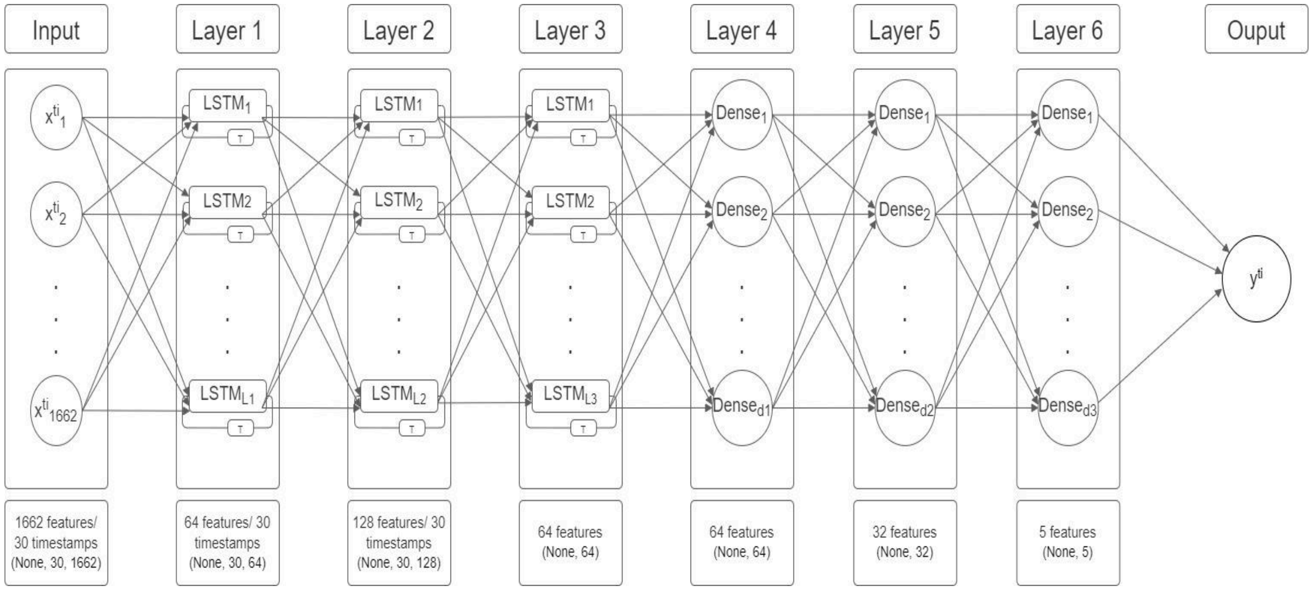

Fig 4: LSTM Architecture

3.3.3 Real Time Prediction

The module is responsible for getting realtime prediction of the ASL hand signs shown by the user. The video is taken in from the laptop webcam and the keypoints are plotted. Using the trained model from the previous module predictions of the hand signs are made. The prediction is then converted to audio and text for output.

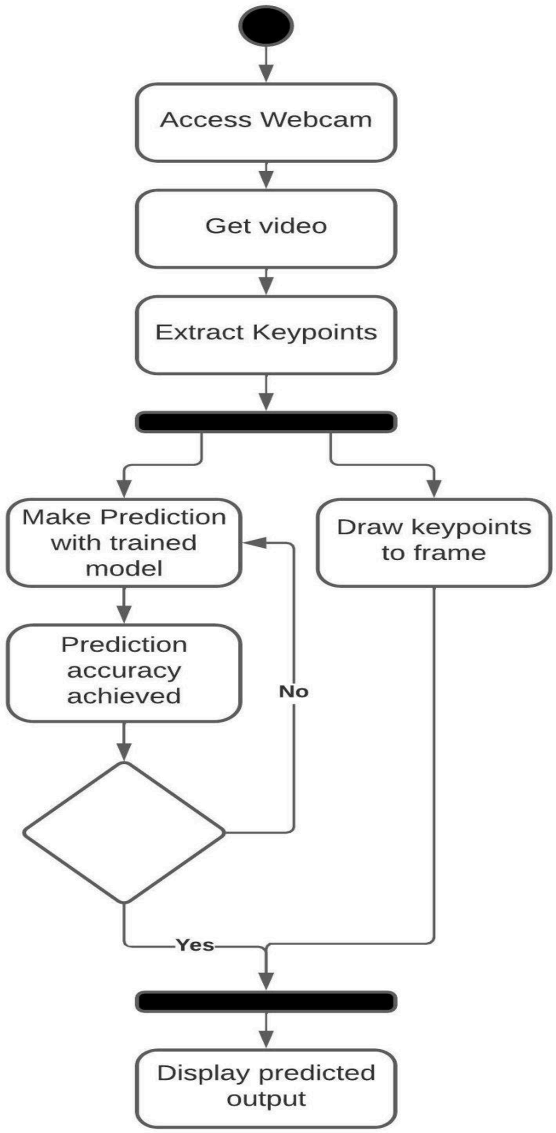

Fig 5: Activity diagram for Real Time Predictions

3.4 System Requirements

OS Platform: Windows 7 or higher

Development Tools: Visual Studio Code, Jupyter Notebook

Language Platform: Python 3.9 interpreter

Hardware Requirements: Laptop with functional camera

CHAPTER 4

System Implementation

4.1 Overview of the Platform

The system is built primarily on Python. We installed the required packages and libraries to execute the program, namely:

- Tensorflow
- OpenCV
- Mediapipe
- matplotlib

4.2 Module Implementation

4.2.1 Keypoint Extraction

FACE LANDMARKS

A list of 468 face landmarks. Each landmark consists of x, y and z. x and y are normalised to [0.0, 1.0] by the image width and height respectively. z represents the landmark depth with the depth at centre of the head being the origin, and the smaller the value the closer the landmark is to the camera. The magnitude of z uses roughly the same scale as x.

LEFT HAND LANDMARKS

A list of 21 hand landmarks on the left hand. Each landmark consists of x, y and z. x and y are normalised to [0.0, 1.0] by the image width and height respectively. z represents the landmark depth with the depth at the wrist being the origin, and the smaller the value the closer the landmark is to the camera. The magnitude of z uses roughly the same scale as x.

RIGHT HAND LANDMARKS

A list of 21 hand landmarks on the right hand, in the same representation as left hand landmarks.

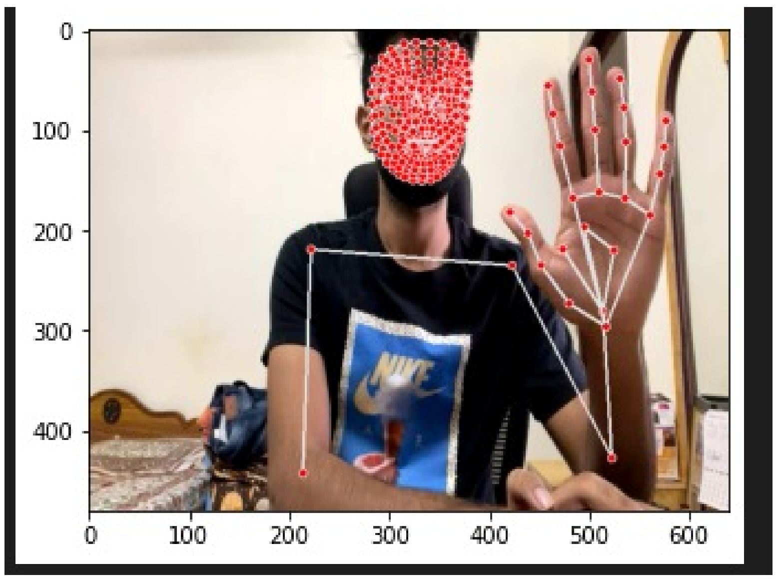

Fig 6: Output for Extracting Keypoints

4.2.2 Train Model

Using the prediction obtained from the previous module, a sequential model is created using 3 LSTM and 3 Dense layers. The created model is then tested for accuracy, and it resulted in 100% accuracy for test data.

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_6 (LSTM)                (None, 30, 64)            442112
_________________________________________________________________
activation_14 (Activation)   (None, 30, 64)            0
_________________________________________________________________
lstm_7 (LSTM)                (None, 30, 128)           98816
_________________________________________________________________
activation_15 (Activation)   (None, 30, 128)           0
_________________________________________________________________
lstm_8 (LSTM)                (None, 64)                49408
_________________________________________________________________
activation_16 (Activation)   (None, 64)                0
_________________________________________________________________
dense_10 (Dense)             (None, 64)                4160
_________________________________________________________________
activation_17 (Activation)   (None, 64)                0
_________________________________________________________________
dense_11 (Dense)             (None, 64)                4160
_________________________________________________________________
activation_18 (Activation)   (None, 64)                0
_________________________________________________________________
dense_12 (Dense)             (None, 32)                2080
_________________________________________________________________
activation_19 (Activation)   (None, 32)                0
_________________________________________________________________
dense_13 (Dense)             (None, 32)                1056
_________________________________________________________________
activation_20 (Activation)   (None, 32)                0
_________________________________________________________________
dense_14 (Dense)             (None, 5)                 165
=================================================================
Total params: 601,957
Trainable params: 601,957
Non-trainable params: 0
_________________________________________________________________
```

Fig 7: Output for Train Model

4.2.3 Real Time Prediction

After obtaining the model from the previous model, it is then tested against ASL signs. The input is taken in through the webcam and the predicted output is displayed on the screen as text along with the audio.

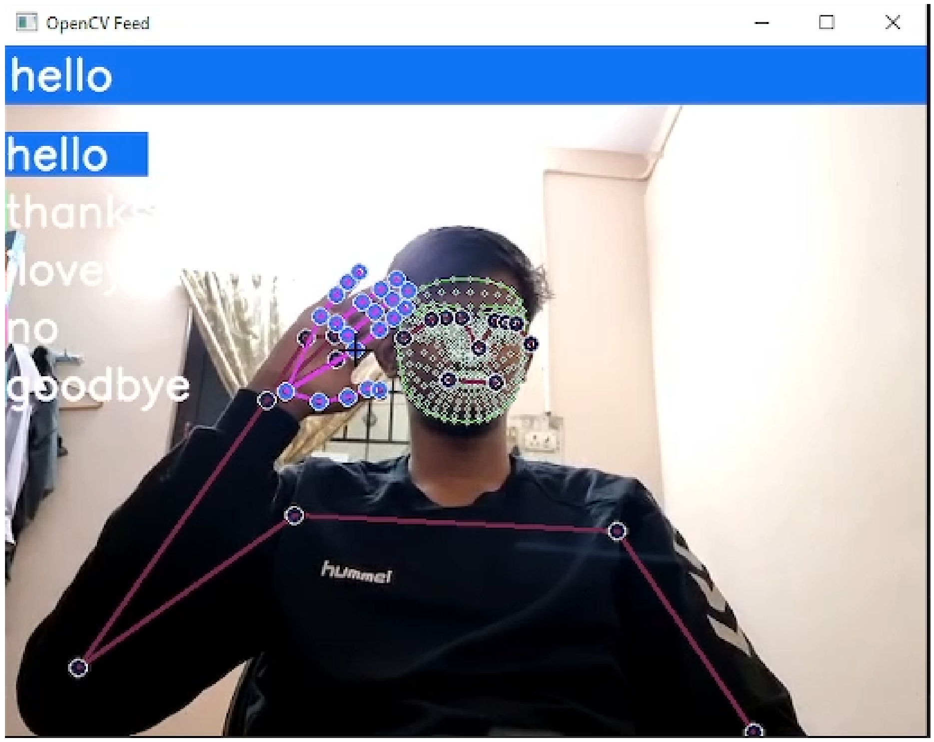

Fig 8 : Output for Hello

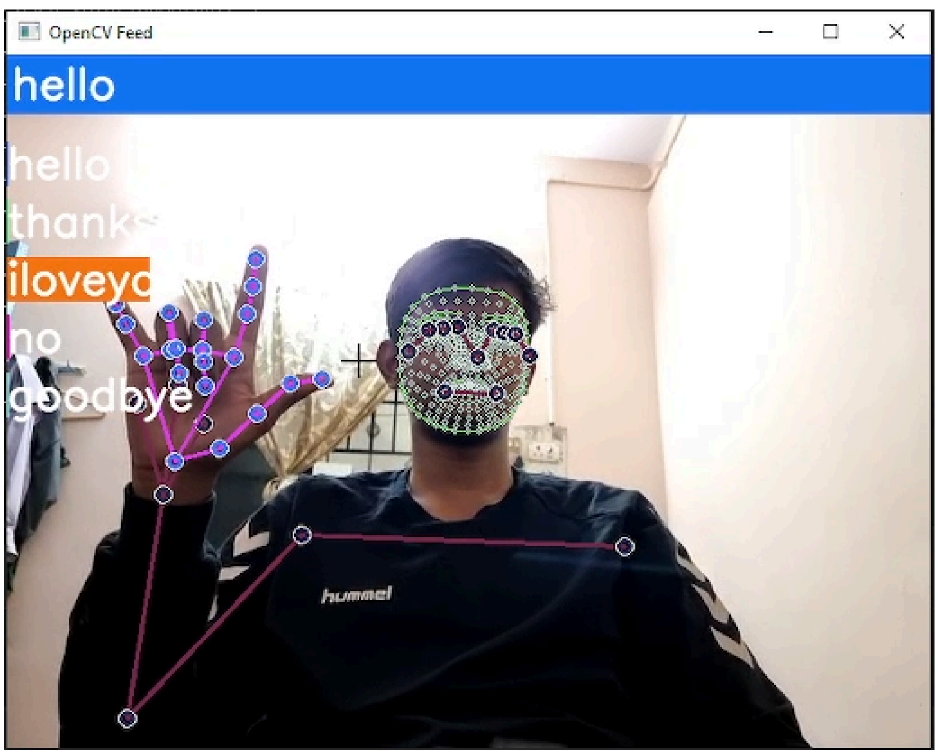

Fig 9 : Output for I Love You

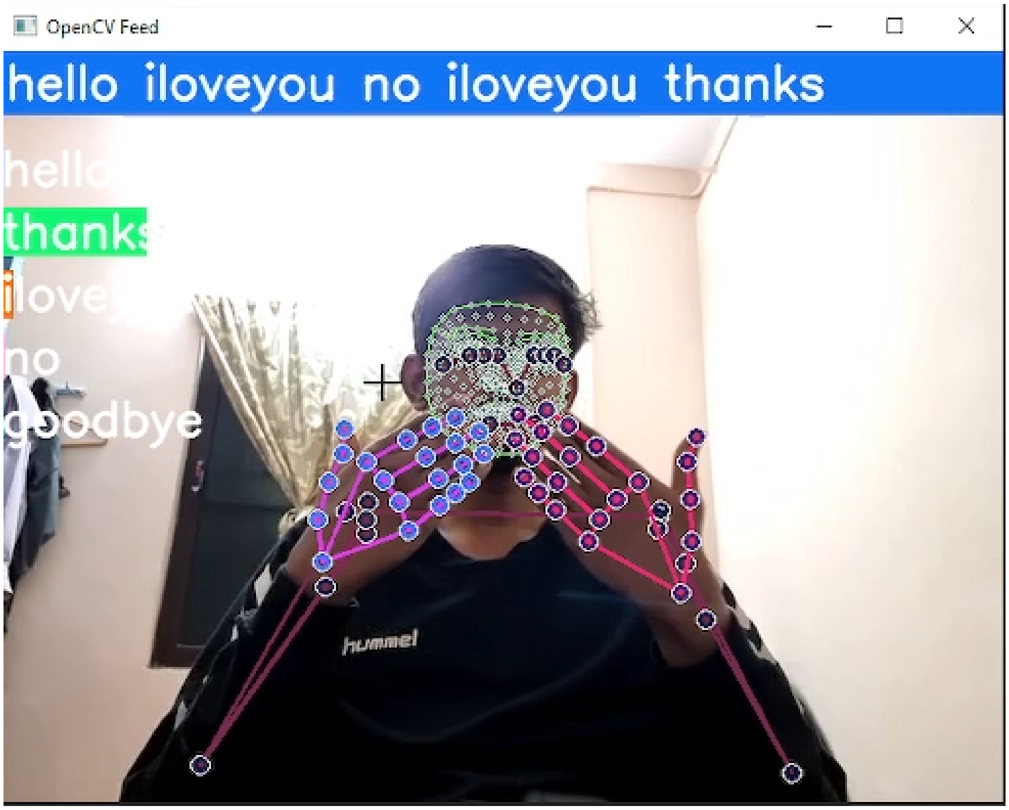

Fig 10: Output for Thanks

CHAPTER 5

System Performance

Accuracy

The detection model's accuracy indicates its ability to assess the positive as positive and the Accuracy: The detection model's accuracy indicates its ability to assess the +ve as +ve and the -ve as -ve over the whole test set; it can judge the positive as positive and the negative as negative.

Ref: TP - True Positive FN- False Negative

Accuracy = TP+FN/ TP+TN+FP+FN

Precision

The fraction of true pos eg in the pos eg determined by the detection model is referred to as precision.

Precision = TP/TP+FP

Recall

In the total positive cases, the expected positive cases proportion is referred to as recall.

Recall = TP/TP+FN

F1-Score

Harmonic mean of the accuracy and recall rates. This measures the model's discriminant ability for each category.

F1 Score = 2TP/2TP+FP+FN

Support

Support is the number of actual occurrences of the class in the specified dataset. Imbalanced support in the training data may indicate structural weaknesses in the reported scores of the classifier and could indicate the need for stratified sampling or rebalancing.

Macro average

It computes f1 for each label, and returns the average without considering the proportion for each label in the dataset.

Weighted average

It computes f1 for each label, and returns the average considering the proportion for each bel in the dataset.

5.1 Performance Measures

The model has a training accuracy of 99.73% and testing accuracy of 95.17% for 5 words: ‘Hello’, ‘I Love You’, ‘No’, ‘Thank You’, ‘Good Bye’. Given below are two graphs where the categorical accuracy and loss have been represented on y axes over the number of epochs on the x axes.

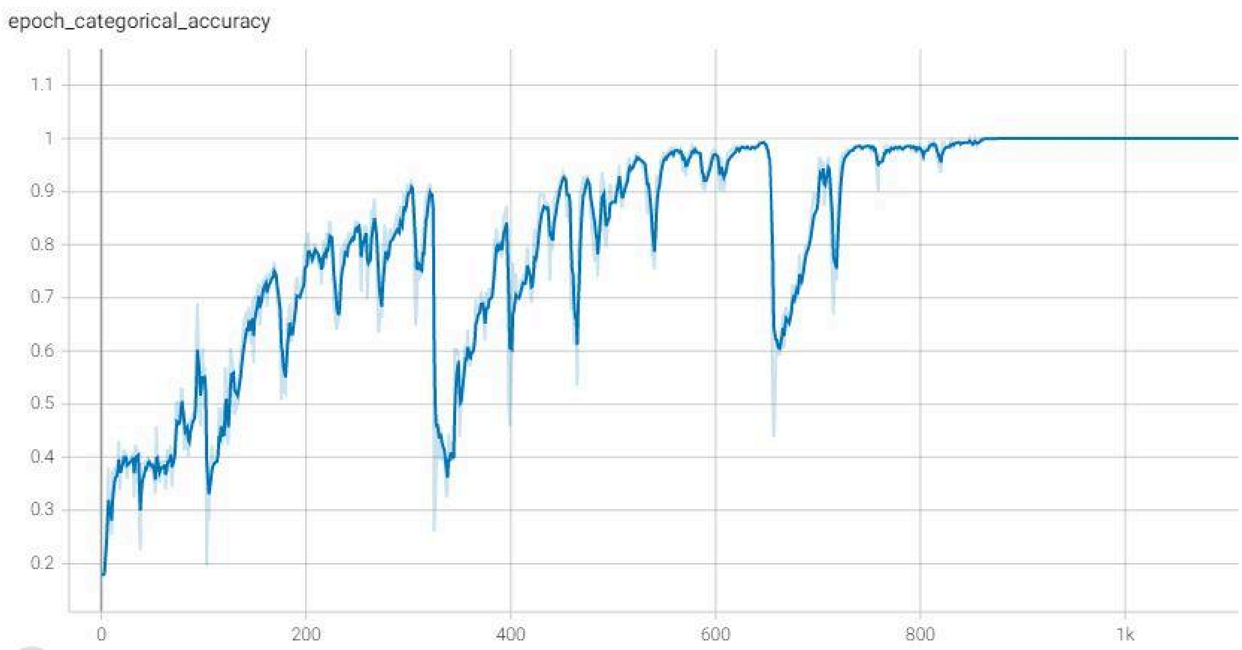

Fig 11: Graph for Categorical Accuracy

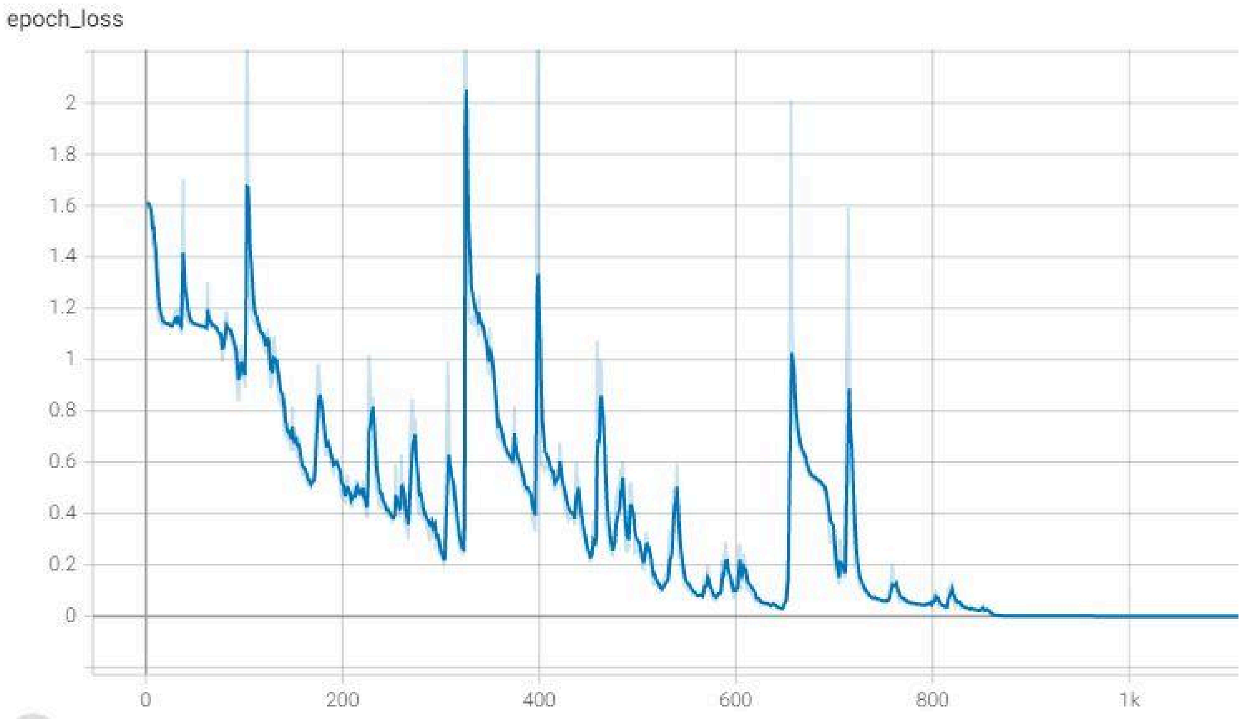

Fig 12: Graph for Loss

Parameters	**Value**
No of Dense layers	5
No of LSTM layers	3
Layer for classification	convolutional neural network
Epochs	1000
Loss function	Categorical Cross Entropy
Layer dimension	(150, 30, 1662)

Parameters Table

CHAPTER 6

Conclusion and Future Works

6.1 Conclusion

The proposed approach is a tool to bridge the communication gap between non native and native ASL signers. We were able to convert ASL signs into readable text and coherent audio. The system has 92.17% accuracy for 5 signs using 3 dense and LSTM layers. The solution offered is feasible and efficient and with certain enhancements, it can become the standard for sign language recognition softwares.

6.2 Future Works

The system can be converted into a bi-directional software wherein it is possible to implement converting speech into its corresponding ASL sign. The frontend of the software can be built using html/css/js and we can connect it to the backend using python eel library. The output of speech to sign can either be a picture, animation, or a video of the corresponding ASL sign.

Appendices - Source Code

```
# 1. Install and Import
!pip install tensorflow==2.5.0 tensorflow-gpu==2.5.0
opencv-python mediapipe sklearn matplotlib pyttsx3
import cv2
import numpy as np
import os
from matplotlib import pyplot as plt
import time
import mediapipe as mp
import pyttsx3
# 2. Keypoints using MP Holistic
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_CONTOURS)
# 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_CONTOURS,

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(1)
# 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', frame)

        # Break gracefully
        if cv2.waitKey(10) & 0xFF == ord('q'):
            break
    cap.release()
    cv2.destroyAllWindows()
len(results.left_hand_landmarks.landmark)
results
draw_landmarks(frame, results)
plt.imshow(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
# 3. Extract Keypoint Values
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_test
np.save('0', result_test)
np.load('0.npy')
# 4. Setup Folders for Collection
# Path for exported data, numpy arrays
DATA_PATH = os.path.join('prev_data_and_model/MP_Data')

# Actions that we try to detect
actions = np.array(['hello', 'thanks',
'iloveyou','no','goodbye'])
# actions =
np.array(['a','b','c','d','e','f','g','h','i','j','k','l','
m','n','o','p','q','r','s','t','u','v','w','x','y','z'])

# Thirty videos worth of data
no_sequences = 30

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

```
for action in actions:
    for sequence in range(no_sequences):
        try:
            os.makedirs(os.path.join(DATA_PATH, action,
str(sequence)))
        except:
            pass
# 5. Collect Keypoint Values for Training and Testing
cap = cv2.VideoCapture(1)
# 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(1000)
                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
        input("Press Enter to continue")
    cap.release()
    cv2.destroyAllWindows()
```

```
cap.release()
cv2.destroyAllWindows()
# 6. Preprocess Data and Create Labels 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)}
label_map
sequences, labels = [], []
for action in actions:
    for sequence in
np.array(os.listdir(os.path.join(DATA_PATH,
action))).astype(int):
        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])
np.array(sequences).shape
np.array(labels).shape
X = np.array(sequences)
X.shape
y = to_categorical(labels).astype(int)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.05)
y_test.shape
# 7. Build and Train LSTM Neural Network
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Activation
from tensorflow.keras.callbacks import TensorBoard
from tensorflow.keras.callbacks import ReduceLROnPlateau
```

```
log_dir = os.path.join('Logs5')
tb_callback = TensorBoard(log_dir=log_dir)
# reduce_lr =
ReduceLROnPlateau(monitor='categorical_accuracy',
factor=0.2,patience=5, min_lr=0.001)
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(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(actions.shape[0], activation='softmax'))
# my own model
model = Sequential()
model.add(LSTM(64, return_sequences=True,
input_shape=(30,1662)))
model.add(Activation('relu'))
model.add(LSTM(128, return_sequences=True))
model.add(Activation('relu'))
model.add(LSTM(64, return_sequences=False))
model.add(Activation('relu'))
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dense(32))
model.add(Activation('relu'))
model.add(Dense(32))
```

```
model.add(Activation('relu'))
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])
# model.fit(X_train, y_train, epochs=2000,
callbacks=[reduce_lr])
model.summary()
# 8. Make Predictions
res = model.predict(X_test)
actions[np.argmax(res[2])]
actions[np.argmax(y_test[2])]
# 9. Save Weights
model.save('alphabet.h5')
# del model
# model.load_weights('data5acchigh.h5')
model.load_weights('prev_data_and_model/data5100acc.h5')

# 10. Evaluation using Confusion Matrix and Accuracy
from sklearn.metrics import multilabel_confusion_matrix,
accuracy_score
yhat = model.predict(X_test)
ytrue = np.argmax(y_test, axis=1).tolist()
yhat = np.argmax(yhat, axis=1).tolist()
multilabel_confusion_matrix(ytrue, yhat)
accuracy_score(ytrue, yhat)
# 11. Test in Real Time
from scipy import stats
colors = [(245,117,16), (117,245,16), (16,117,245), (204,
51, 255), (255, 255, 153)]
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
plt.figure(figsize=(18,18))
plt.imshow(prob_viz(res, actions, image, colors))
def run_audio(word):
    engine = pyttsx3.init()
    engine.say(word)
    engine.runAndWait()
    engine.stop()
# 1. New detection variables
sequence = []
sentence = []
predictions = []
threshold = 0.525
lastWord = "ASL"


cap = cv2.VideoCapture(1)
# 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.append(keypoints)
        sequence = sequence[-30:]

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


        #3. Viz logic
            if 
np.unique(predictions[-10:])[0]==np.argmax(res): 
                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)
        if(len(sentence) > 0):
            print(sentence[-1],lastWord)
            if(lastWord != sentence[-1]):
                lastWord = sentence[-1]
                run_audio(sentence[-1])
#                 engine = pyttsx3.init()
#                 engine.say(lastWord)
#                 engine.runAndWait()
#                 engine.stop()
        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()
```

References

1. Sevgi Z. Gurbu, Ali Cafer Gurbuz, Evie A. Malaia, Darrin J. Griffin, Chris S. Crawford, Mohammad Mahbubur Rahman, Emre Kurtoglu, Ridvan Aksu, Trevor Macks, Robiulhossain Mdrafi, 2020, “American Sign Language Recognition Using RF Sensing”, IEEE Sensors Journal, Volume: 21, Issue: 3, pg 3763 - 3775.

2. Suneetha, M., M. V. D. Prasad, and P. V. V. Kishore. "Multi-view motion modelled deep attention networks (M2DA-Net) for video based sign language recognition." Journal of Visual Communication and Image Representation 78 (2021): 103161.

3. Wadhawan, Ankita, and Parteek Kumar. "Deep learning-based sign language recognition system for static signs." Neural computing and applications 32.12 (2020): 7957-7968.

4. Rastgoo, Razieh, Kourosh Kiani, and Sergio Escalera. "Hand sign language recognition using multi-view hand skeleton." Expert Systems with Applications 150 (2020): 113336.

5. Rastgoo, Razieh, Kourosh Kiani, and Sergio Escalera. "Video-based isolated hand sign language recognition using a deep cascaded model." Multimedia Tools and Applications 79.31 (2020): 22965-22987.

6. Sahana, Taniya, et al. "Hand sign recognition from depth images with multi-scale density features for deaf mute persons." Procedia Computer Science 167 (2020): 2043-2050.

7. Adithya, V., and Reghunadhan Rajesh. "A deep convolutional neural network approach for static hand gesture recognition." Procedia Computer Science 171 (2020): 2353-2361.

8. Ravi, Sunitha, et al. "Multi modal spatio temporal co-trained CNNs with single modal testing on RGB–D based sign language gesture recognition." Journal of Computer Languages 52 (2019): 88-102.

9. Papadimitriou, Katerina, and Gerasimos Potamianos. "Fingerspelled alphabet sign recognition in upper-body videos." 2019 27th European Signal Processing Conference (EUSIPCO). IEEE, 2019.

```
multilabel_confusion_matrix(ytrue, yhat)
```

✓ 0.5s

```
array([[[7, 0],
        [0, 1]],

       [[4, 0],
        [0, 4]],

       [[6, 0],
        [0, 2]],

       [[7, 0],
        [0, 1]]], dtype=int64)
```