REAL-TIME COMMUNICATION SYSTEM POWERED BY AI FOR SPECIALLY ABLED

(SIGN LANGUAGE DETECTION USING MEDIA PIPE HOLISTICS)

TEAM ID: PNT2022TMID04444

NOOR ARFIN A (737819CSR129)

MANOJ KUMAR P (737819CSR103)

RANJITH C (737819CSR152)

NANDA KISHORE M A (737819CSR116)

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 PROJECT OVERVIEW

Sign language is a mode of communication that uses visual ways like expressions, hand gestures, and body movements to convey meaning. This is extremely helpful for people who face difficulty with hearing or speaking.

Sign language recognition refers to the conversion of these gestures into words or alphabets for deaf people of existing formally spoken languages or gestures into audio for blind people. Thus, the conversion of sign language into words or audio by an algorithm or a model can help bridge the gap between people with hearing or speaking impairment and the rest of the world.

We start by collecting key points from media pipe holistic and collect a bunch of data from key points. We then build an LSTM model and train with our stored data which helps us to detect action with a number of frames. Once training is done, we can use this model for real-time hand gesture detection and simultaneously convert the gesture to speech using OpenCV.

1.2 PURPOSE

Sign language is manual communication commonly used by people who are deaf. Sign language is not universal. People who are deaf from different countries speak different sign languages. The gestures or symbols in sign language are organized in a linguistic way.

The purpose of sign language recognition (SLR) systems is to provide an efficient and accurate way to convert sign language into text or voice aids for the hearing impaired for example or enable very young children to interact with computers (recognizing sign language), among others.

Signs give extra visual information about the words used in the message making it easier to understand. The extra visual cues given in signs supports the learning of new words and helps to model how and when to use them.

CHAPTER 2

LITERATURE SURVEY

2.1 EXISTING PROBLEM

Communication between specially-abled and ordinary people has always been a challenging task. Ordinary persons cannot learn the way to communicate with specially-abled persons easily.

As communication needs to be faster in order to obtain the exact meaning, the system which is to be developed needs to be faster and more accurate. This system should also convey the message at the time of emergency. The message should be transferred from one person to another without any change in the meaning.

2.2 REFERENCES

Liang and Ouhyoung proposed a sign language recognition system using Hidden Markov Model and an integrated statistical approach used in computational linguistics. Real-time continuous gesture recognition system for sign language by Rung-Huei Liang, Ming Ouhyoung intended to recognize a large set of vocabularies in sign language by recognizing constructive postures and context information. The system uses position, orientation, and motion models, in addition to the posture model, which is implemented to enhance the performance of the system.

Starner and Pentland's American sign language system could recognize short sentences of American Sign Language (ASL) with 40 vocabularies, each being attached to its part of speech, which greatly reduced the computational complexity. The feature vector was fed to a Hidden Markov Model (HMM) for recognition of the signed words. This system gracefully integrated a useful concept in computational linguistics into gesture recognition. Furthermore, Nam's system tried to recognize hand movement patterns. An HMM-based method for recognizing the space-time hand movement pattern was proposed, and 10 kinds of movement primes could be

recognized successfully.

Fel's Glove Talk focused on a gesture-to-speech interface. Moreover, a multilayer perceptron model was used in Beale and Edward's posture recognizer to classify sensed data into five postures in ASL. To help people with disabilities, Newby worked on the recognition of the letters and numbers of the ASL manual alphabet based on statistical similarity. A simplified method, using an appropriate spline, was proposed by Watson. Gestures are represented by a sequence of critical points (Local minima and maxima) of the motion of the hand and wrist. This approach is more flexible in matching a gesture both spatially and temporally and thus reduces the computational requirement.

2.3 PROBLEM STATEMENT DEFINITION

Deaf and mute people face difficulty to communicate with normal people through sign language. Deaf and mute people just share information through sign language and these gestures are made using hands, fingers, arms, head, and also facial expressions. Communication between specially-abled and ordinary people has always been a challenging task. The relatives or family members of deaf and mute people also face difficulties to express their opinion and communicating with them. They feel being left out of social activities, not being understood, and being left out of important discussions. They want to express their emotions are frustration, anger, being left out, lonely, fear, and neglect. Sign language recognition is the task of recognizing sign language glosses from video streams and the glosses are converted into audio. It can bridge the communication gap between deaf and mute people, facilitating the social inclusion of hearing-impaired people.

CHAPTER 3

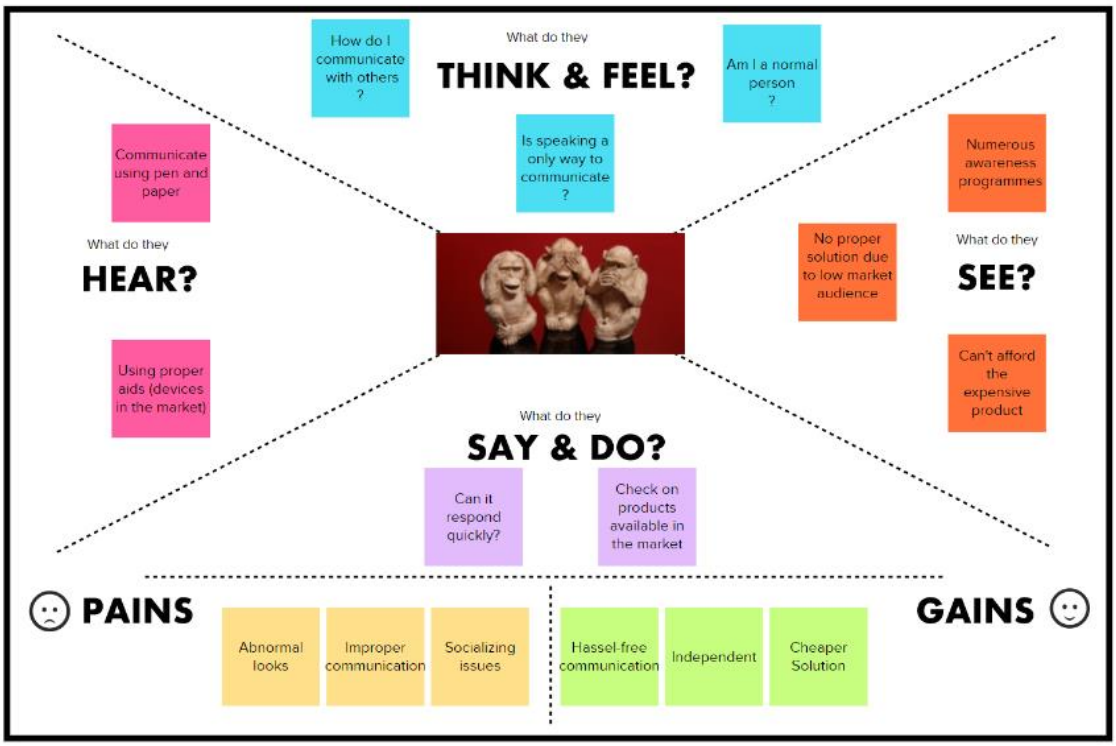
IDEATION & PROPOSED SOLUTION

3.1 EMPATHY MAP CANVAS

An empathy map is a collaborative tool team can use to gain a deeper insight into their customers. This tool helps to understand the reason behind some actions a user takes deeply. This tool helps build empathy towards users and helps design teams shift focus from the product to the users who are going to use the product.

Deaf and mute people think how do I communicate with others? and is speaking the only way to communicate? They feel Am I a normal person. Deaf and mute people communicate using pen and paper and using proper aids (devices in the market).

Pains are Abnormal looks, Improper communication, and Socializing issues. The gains are Hassle-free communication, independent, and a Cheaper solution.



3.2 IDEATION & BRAINSTORMING

There are various ideas to implement sign language recognition.

- A study on-manual sign involves the facial region, including the movement of the lead, eye blinking, eyebrow movement, and mouth shape. This can be traced and interpreted to show communication.
- The recognition of signs with facial expressions, hand gestures, and body movement simultaneously with better recognition accuracy in real-time with improved performance helps in better communication.
- Blind people can use smart sticks to enable visually impaired people to find difficulties in detecting obstacles and dangers in front of them during walking and to identify the world around them and it acts like an artificial vision and alarm unit.

- The keyboard for the deaf feature can support the sign language images and symbols in the keyboard as a different feature to convert between the normal person language and the deaf language.

- The deaf person faces a very difficult problem to understand or identify the medicine's instructions. Idea is to prepare a sign language video have all the instructions on the medicine and what is the quantity of the medicine that should be taken by the deaf person.

- Object detection models can be used in order to specify the objects in front of the people with the positions of the objects which can be said in text/audio as per the need.

After brainstorming, selecting the best idea to propose the sign language recognition. Choosing recognition of signs with facial expression, hand gestures, and body movement simultaneously with better accuracy in real-time with improved performance. Using LSTM Model to implement this solution by using media-pipe holistic for taking inputs from the user.

3.3 PROPOSED SOLUTION

To model a system for aiding deaf and dumb people and help them to communicate in real time. We start by collecting key points from mediapipe holistic and collect a bunch of data from key points, then build an LSTM model and train with our stored data which helps us to detect action with a number of frames. Once training is done, we can use this model for real-time hand gesture detection and simultaneously convert the gesture to speech using OpenCV.

We will be using the latest and trending wearable technology which makes it possible to access (Web applications) easily anywhere and everywhere by disabled persons which make communication possible by both specially abled and normal people. We will be using the most recent convolution neural network architecture to improve the efficiency of the trained model.

3.4 PROBLEM-SOLUTION FIT

Communication between specially-abled and ordinary people has always been a challenging task. We take this problem and give a solution by recognizing words or sentences using sign language. This solution is extremely helpful for people who face difficulty with hearing or speaking. Hearing disabilities and speaking problems are becoming common among kids.

The recognition of signs with facial expressions, hand gestures, and body movement simultaneously with better recognition accuracy in real-time with improved performance helps in better communication. Deaf and mute people face difficulties in communicating with normal people, not being understood, and being left out of important discussions. Sign language recognition is the task of recognizing sign language glosses from video streams and the glosses are converted into audio. It can bridge the communication gap between deaf and mute people, facilitating the social inclusion of hearing-impaired people.

CHAPTER 4

REQUIREMENT ANALYSIS

4.1 FUNCTIONAL REQUIREMENT

A functional requirement defines a system or its component. It may involve calculations, technical details, data manipulation and processing, and other specific functionality that define what a system is supposed to accomplish.

User communication – The user must know the sign language because if the user may be deaf and mute means they have to share their message into sign language. Different countries have different sign languages, so know their native sign language for collecting information from that.

User communication – The user has to communicate in front of the camera. The camera covers all the body parts because it takes input from facial expressions, hand gestures, eyebrows, and body movements.

4.2 NON-FUNCTIONAL REQUIREMENT

Non-functional requirements are a set of specifications that describe the system's operational capabilities and constraints and attempt to improve its usability, performance, scalability, and reliability.

Usability – The camera captures all expressions including facial expressions and hand gestures which can be easily used by all age groups.

Reliability – The system is very liable; it can last for long amounts of time if well maintained.

Performance – The cost-effective nature of the system makes it extremely liable and thus, efficient.

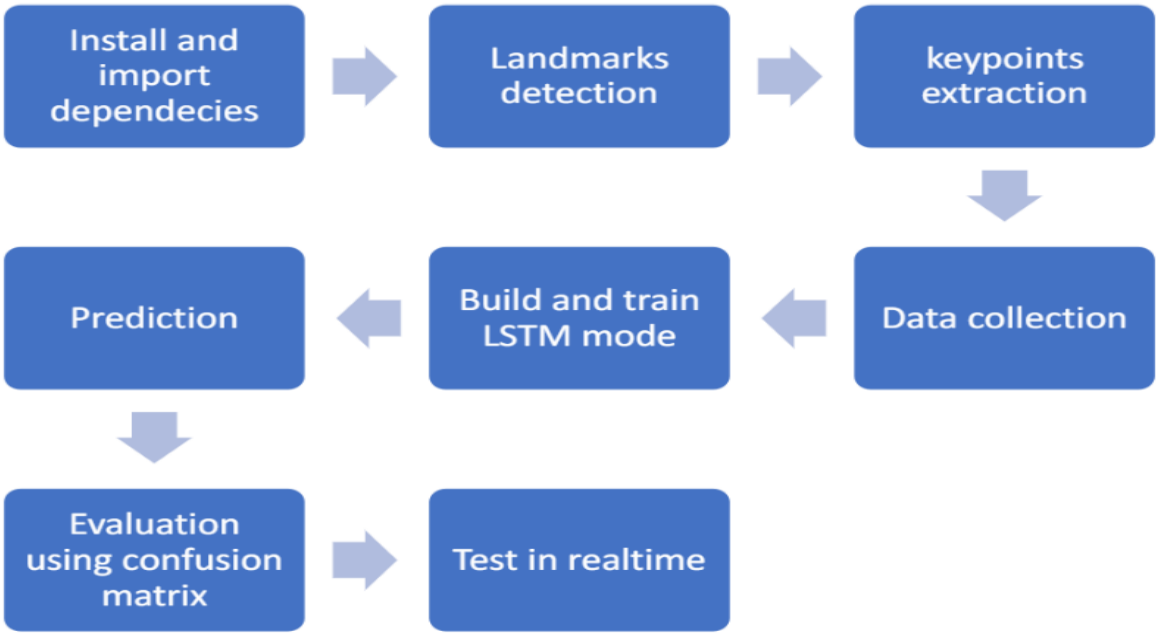
Availability – The solution fits all the sign languages when we train the model for all the sign languages, so it is used by all the countries with different languages.

Scalability – The system gives output rapidly. It also predicts quickly when it gets so many inputs at a time. It predicts different types of sign language at a time.

CHAPTER 5

PROJECT DESIGN

5.1 DATA FLOW DIAGRAM

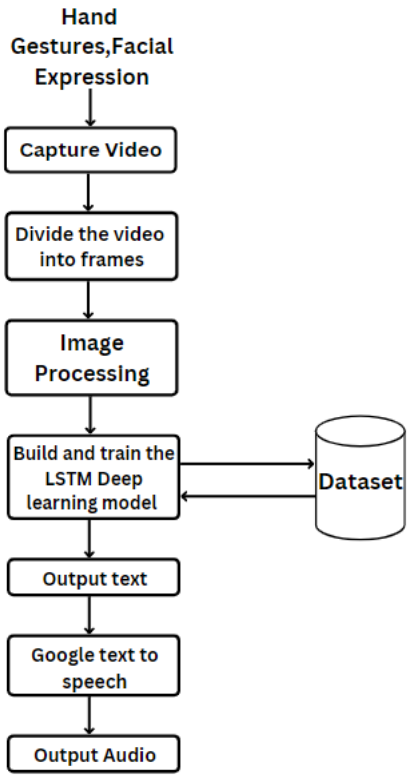


5.2 SOLUTION & TECHNICAL ARCHITECTURE

Solution architecture is a complex process – with many sub-processes – that bridges the gap between business problems and technology solutions.

- Technology used are python, Jupyter, Numpy, OpenCV, Keras, TensorFlow, and GTTS.
- Methodology – We start by collecting key points from media pipe holistic and collect a bunch of data from key points i.e., our hands, on our body and on our face and save data in the form of Numpy arrays. We can vary the number of sequences according to our need but each sequence will have 30 frames.

- We then build an LSTM model and train with our stored data which helps us to detect action with a number of frames.
- The number of epochs for the model is determined by us, if we increase the number of epochs the accuracy increases but the time taken to run the model also increases and overfitting of a model can happen, for gesture recognition.
- Once training is done, we can use this model for real-time gesture detection and simultaneously convert the gesture to speech or text using OpenCV.



5.3 USER STORIES

User Type – Developer

Data collection – Collect dataset for sign language recognition. Collecting key points using media pipe holistic by dataset videos.

Model building – Model initialization with required layers and train the model using LSTM from key points collected.

Testing – Testing the model's performance properly with different types of testing and converting final output text to speech using google API.

User Type – Customer (Web user)

Communication – communicating in front of camera using sign language and also speech and text are delivered by web interface.

CHAPTER 6

PROJECT PLANNING & SCHEDULING

6.1 SPRINT PLANNING & ESTIMATION

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Data Collection	USN-1	Collect Dataset	5	High	Noor Arfin, Manoj Kumar
Sprint-1		USN-2	Collect Key points using Media Pipe Holistic	5	High	Ranjith
Sprint-2	Model Building	USN-3	Model initialisation with required layers	5	High	Nanda Kishore
Sprint-2		USN-4	Training model using LSTM from key points collected	5	Medium	Noor Arfin
Sprint-3	Testing	USN-5	Testing the model's performance	10	High	Manoj Kumar
Sprint-4	Speech feature implementation	USN-6	Converting text to speech using google API	10	Medium	Ranjith

6.2 SPRINT DELIVERY SCHEDULE

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	10	6 Days	24 Oct 2022	29 Oct 2022	10	29 Oct 2022
Sprint-2	10	6 Days	31 Oct 2022	05 Nov 2022	5	05 Nov 2022
Sprint-3	10	6 Days	07 Nov 2022	12 Nov 2022	6	12 Nov 2022
Sprint-4	10	6 Days	14 Nov 2022	19 Nov 2022	6	19 Nov 2022

CHAPTER 7

CODING & SOLUTIONING

7.1 FEATURE 1

We used flask for Web UI as a user interface. We use python language for processing all the data. We start by collecting key points from mediapipe holistic and collect a bunch of data from key points. We then build an LSTM model and train with our stored data which helps us to detect action with a number of frames. Using this model sign language is converted into text. By using google speech API is used to convert text into speech. A machine learning model is used to predict and recognize sign language.

7.2 FEATURE 2

Python and Jupyter are the technology used for creating an application. These technologies are open-source frameworks. Using HTML, CSS, and Python are used to build scalable architecture. The client layer is used for the web interface. The data layer is used for dataset storage and processing. The application layer is used to model building and training. IBM cloud used for high availability. High performance for accurate prediction of signs and less prediction time.

CHAPTER 8

RESULTS

8.1 PERFORMANCE METRICS

When deaf-mute people need to share information, communication between normal people is established through the sign language recognition technique. Creating a flask app to share deaf-mute people's emotions and their opinions with others. Finding appropriate meanings for every sign done by the customers who use our flask app. Our application can respond to normal people through on-screen translation for every sign language done by our customers. This recognition system may be used by deaf-mute people to continue to sharing the information to others. It is detailed how the sign language recognition application will change in the future.

CHAPTER 9

ADVANTAGES & DISADVANTAGES

9.1 ADVANTAGES

9.1.1 STRONGER BOND BETWEEN PARENTS AND INFANTS

Young children's inability to communicate about what they are feeling can often be a source of anger and frustration, both for the parents and the children. When a child is distraught, the ability to sign can prevent tantrums and frustration by allowing them to communicate their basic needs, whether they're hungry or in pain.

Sign language can also increase opportunities for parents and children to bond, because of the eye-to-eye and tactile contact it requires.

9.1.2 ENHANCED ABILITY TO INTERPRET BODY LANGUAGE

It involves facial expressions and body language as well as hand gestures, learning sign language could also enhance your ability to recognize and interpret body language. Body language includes a range of nonverbal signals that people use to communicate their feelings and intentions, such as posture and facial expressions.

9.1.3 EASY UNDERSTANDABLE

Nowadays normal person doesn't know the meaning of sign language, but deaf-mute people are needed to share their opinions and emotions with others or parents.

9.2 DISADVANTAGES

9.2.1 HANDS-ON SPEECH

Sign language requires the use of hands to make gestures. This can be a problem for people who do not have full use of their hands. Even seemingly manageable disabilities such as Parkinson's or arthritis can be a major problem for people who must communicate using sign language. Having a broken arm or carrying a bag of groceries can, for a deaf person, limit communication. The amount of light in a room also affects the ability to communicate using sign language.

9.2.2 DEVICE MANAGEMENT

Deaf-mute people need one device with a camera option all the time to use a sign language recognition application to share their information with others. Because that application only translating the sign language into text or audio to others who doesn’t understand the sign language. The devices like mobile phones, system, tablet etc.

CHAPTER 10

CONCLUSION

We provide a flask application for recognizing sign language into text or audio through on-screen mode. Deaf-mute people doing sign language in front of the camera, we start by collecting key points from mediapipe holistic by facial expressions, body movement, and hand gestures. Collect a bunch of data from key points. We build an LSTM model and train the model with stored data which helps to detect action. Use this model to recognize the sign language for hand gesture detection and convert the gesture into text and text are also converted into speech using OpenCV. The text should be shown in the on-screen through flask application.

CHAPTER 11

FUTURE SCOPE

Deaf-mute people use their hands and body movements to share their information. But we have some disadvantages to that type of communication between a deaf-mute person and an ordinary person. Gestures are difficult in understanding, informal etiquette, information might get distorted, etc. It is not precise and sometimes it is vague and painful. One cannot make long explanations or conversations through gestures. In the future, also have to develop a model with lip reading recognition. It is used by both ordinary people and deaf-mute people. It is simpler than hand gestures because it only depends upon mouth movements not hand gestures, body movements, and facial expressions.

CHAPTER 12

APPENDIX

12.1 FOLDER STRUCTURE

```
import os
folder = "E:/dataset_creation/videos"
contents = os.listdir(folder)
for content in contents:
    path = folder+'/'+content
    files = os.listdir(path)
    for filename in files:
        dst = f"{filename}"
        res = ''.join([i for i in dst if not i.isdigit() if not i == '.' if not i == ' ']).lower()
        src =f"{folder}/{content}/{filename}"  # foldername/filename, if .py file is outside
folder
        dst =f"{folder}/{content}/{res}"
        # rename() function will
        # rename all the files
        os.rename(src, dst)
```

12.2 IMG_NPARRAY

```
import os
import cv2
```

```
import numpy as np
import time
import mediapipe as mp
from matplotlib import pyplot as plt
videos = 'E:/IBM/Code/ActionDetectionforSignLanguage/Test/videos/'
dataset = 'E:/IBM/Code/ActionDetectionforSignLanguage/Test/dataset/'
mp_holistic = mp.solutions.holistic # Holistic model
mp_drawing = mp.solutions.drawing_utils # Drawing utilities
mp_pose = mp.solutions.pose
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))
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])
contents = os.listdir(videos)
sequence_length = 60
for content in contents:
    path = videos+content+'/'
    vids = os.listdir(path)
    cmd = 'mkdir E:\IBM\Code\ActionDetectionforSignLanguage\Test\dataset'+'\\'+content
    os.system(cmd)
```

```
    indx = 0
    for vid in vids:
        cmd = 'mkdir
E:\IBM\Code\ActionDetectionforSignLanguage\Test\dataset'+'\\'+content+'\\'+str(indx)
        os.system(cmd)
        cap = cv2.VideoCapture(videos+content+'/'+vid)
        # Set mediapipe model
        with mp_holistic.Holistic(min_detection_confidence=0.5,
min_tracking_confidence=0.5) as holistic:
            for frame_num in range(sequence_length):
                # Read feed
                ret, frame = cap.read()
                if not ret:
                    break
                # Make detections
                image, results = mediapipe_detection(frame, holistic)
                # 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 {}'.format(content, vid),
(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(500)
            else:
                cv2.putText(image, 'Collecting frames for {} Video {}'.format(content, vid),
(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 = dataset+content+'/'+str(indx)+'/'+str(frame_num)
            np.save(npy_path, keypoints)
            # Break gracefully
            if cv2.waitKey(10) & 0xFF == ord('q'):
                break
        cap.release()
        cv2.destroyAllWindows()
    indx = indx+1
```

12.3 LIVE DETECTOR

```
#Load the Trained Model
import os
import cv2
import numpy as np
import time
import mediapipe as mp
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.callbacks import TensorBoard

log_dir = os.path.join('Logs')
tb_callback = TensorBoard(log_dir=log_dir)

model = Sequential()
model.add(LSTM(64, return_sequences=True, activation='relu', input_shape=(60,1662)))
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'))
```

```
model.compile(optimizer='Adam', loss='categorical_crossentropy',
metrics=['categorical_accuracy'])
model.load_weights('model.h5')

#Live Detection

mp_holistic = mp.solutions.holistic # Holistic model
mp_drawing = mp.solutions.drawing_utils # Drawing utilities
mp_pose = mp.solutions.pose

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)
                              )

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])

# 1. New detection variables
sequence = []
sentence = []
predictions = []
threshold = 0.5

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.append(keypoints)
sequence = sequence[-60:]

if len(sequence) == 60:
    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)
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)
        if cv2.waitKey(10) & 0xFF == ord('q'):
            break
    cap.release()
    cv2.destroyAllWindows()
```

12.4 TRAINING_TEST

```
#Train_Test_data
import os
import cv2
import numpy as np
import time
import mediapipe as mp
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical

DATA_PATH = 'E:/dataset_creation/dataset/season'
actions = np.array(['exmonsoon','fall','season','spring','summer','winter'])

#actions = []
# files = os.listdir('E:/dataset_creation/dataset/season')
# for file in files:
#     actions.append(file)
```

```
# actions = np.array([actions])


label_map = {label:num for num, label in enumerate(actions)}
sequences, labels = [], []
for action in actions:
    for sequence in np.array(os.listdir(os.path.join(DATA_PATH, action))).astype(int):
        window = []
        count = 0
        loc = DATA_PATH+'/'+action+'/'+str(sequence)+'/'
        for path in os.scandir(loc):
            if path.is_file():
                count += 1
        sequence_length = count
        if sequence_length == 60:
            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])


X = np.array(sequences)
y = to_categorical(labels).astype(int)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.05)
```

#Model Building

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.callbacks import TensorBoard

log_dir = os.path.join('Logs')
tb_callback = TensorBoard(log_dir=log_dir)

model = Sequential()
model.add(LSTM(64, return_sequences=True, activation='relu', input_shape=(60,1662)))
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'))

model.compile(optimizer='Adam', loss='categorical_crossentropy',
metrics=['categorical_accuracy'])
model.fit(X_train, y_train, epochs=2000, callbacks=[tb_callback])
model.summary()

res = model.predict(X_test)
model.save('model.h5')
```

12.5 TRAINING_VALIDATION

```
#pip install split-folders


import os
import splitfolders
dataset = 'E:/IBM/Code/ActionDetectionforSignLanguage/Test/dataset/'
model = 'E:/IBM/Code/ActionDetectionforSignLanguage/Test/training/'
root = 'E:/IBM/Code/ActionDetectionforSignLanguage/Test/'
def main():
    files = os.listdir(dataset)
    for fname in files:
        splitfolders.ratio(dataset+fname+'/',output=root,seed=1337, ratio=(.8, .2),
group_prefix=None, move=False)
        cmd = 'mkdir
E:\IBM\Code\ActionDetectionforSignLanguage\Test\training\train\\'+fname
        os.system(cmd)
        path = root+'train/'
        train_files = os.listdir(path)
        for train_file in train_files:
            shutil.move(root+'train/',model+'train/fname/')
        os.rmdir(root+'train')
        cmd = 'mkdir E:\IBM\Code\ActionDetectionforSignLanguage\Test\training\val\\'+fname
        os.system(cmd)
```

```
        path = root+'val/'
        val_files = os.listdir(path)
        for val_file in val_files:
            shutil.move(root+'val/',model+'val/fname/')
        os.rmdir(root+'val')
if __name__ == "__main__":
    main()
```

12.6 LIVE SPEECH DETECTOR

```
import pyttsx3
import os
import cv2
import numpy as np
import time
import mediapipe as mp
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical

mp_holistic = mp.solutions.holistic # Holistic model
mp_drawing = mp.solutions.drawing_utils # Drawing utilities
mp_pose = mp.solutions.pose
```

```
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)
                        )

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])

# 1. New detection variables
sequence = []
sentence = []
predictions = []
threshold = 0.5
```

```
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.append(keypoints)
        sequence = sequence[-60:]
        if len(sequence) == 60:
            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)
        engine = pyttsx3.init()
        engine.say(sentence)
        engine.runAndWait()
        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)

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

12.7 HOME.HTML

```
<html lang="en">
<head>
    <title>Conversation Engine</title>
    <link href="https://cdn.bootcss.com/bootstrap/4.0.0/css/bootstrap.min.css"
rel="stylesheet">
<style>
.header {         position: relative;
                                top:0;
                                margin:0px;
                                z-index: 1;
                                left: 0px;
                                right: 0px;
                                position: fixed;
                                background-color: #F36262 ;
                                color: white;
                                box-shadow: 0px 8px 4px grey;
                                overflow: hidden;
                                padding-left:20px;
                                font-family: 'Josefin Sans';
                                font-size: 2vw;
                                width: 100%;
                                height:8%;
                                text-align: center;
```

```
                    }
                    .topnav {
  overflow: hidden;
  background-color: #FCAD98;
}
.topnav-right a {
  float: left;
  color: black;
  text-align: center;
  padding: 14px 16px;
  text-decoration: none;
  font-size: 18px;
}
.topnav-right a:hover {
  background-color: #FCAD98;
  color: black;
}
.topnav-right a.active {
  background-color: #FCAD98;
  color: white;
}
.topnav-right {
  float: right;
  padding-right:100px;
}
```

```
body {
  background-color: ;
  background-repeat: no-repeat;
  background-size:cover;
  background-image:
url("https://i.pinimg.com/originals/b2/1d/c6/b21dc69346915015bc4e19bd502f401b.gif");
    background-size: cover;
  background-position: 0px 0px;
  }
  .button {
  background-color: #091425;
  border: none;
  color: white;
  padding: 15px 32px;
  text-align: center;
  text-decoration: none;
  display: inline-block;
  font-size: 12px;
  border-radius: 16px;
}
.button:hover {
  box-shadow: 0 12px 16px 0 rgba(0,0,0,0.24), 0 17px 50px 0 rgba(0,0,0,0.19);
}
form {border: 3px solid #f1f1f1; margin-left:400px;margin-right:400px;}
```

```
input[type=text], input[type=password] {
  width: 100%;
  padding: 12px 20px;
  display: inline-block;
  margin-bottom:18px;
  border: 1px solid #ccc;
  box-sizing: border-box;
}
button {
  background-color: #091425;
  color: white;
  padding: 14px 20px;
  margin-bottom:10px;
  border: none;
  cursor: pointer;
  width: 17%;
  border-radius:4px;
  font-family:Montserrat;
}
button:hover {
  opacity: 0.8;
}
.cancelbtn {
  width: auto;
```

```
  padding: 10px 18px;
  background-color: #f44336;
}
.imgcontainer {
  text-align: center;
  margin: 24px 0 12px 0;
}
img.avatar {
  width: 30%;
  border-radius: 50%;
}
.container {
  padding: 16px;
}
span.psw {
  float: right;
  padding-top: 16px;
}
/* Change styles for span and cancel button on extra small screens */
@media screen and (max-width: 300px) {
  span.psw {
     display: block;
     float: none;
  }
  .cancelbtn {
```

```
    width: 100%;
  }
}
.home{
        margin:80px;
  width: 84%;
  height: 500px;
  padding-top:10px;
  padding-left: 30px;
}
.login{
        margin:80px;
        box-sizing: content-box;
  width: 84%;
  height: 420px;
  padding: 30px;
  border: 10px solid blue;
}
.left,.right{
 box-sizing: content-box;
 height: 400px;
 margin:20px;
 border: 10px solid blue;
}
.mySlides {display: none;}
```

```
img {vertical-align: middle;}
/* Slideshow container */
.slideshow-container {
  max-width: 1000px;
  position: relative;
  margin: auto;
}
/* Caption text */
.text {
  color: #f2f2f2;
  font-size: 15px;
  padding: 8px 12px;
  position: absolute;
  bottom: 8px;
  width: 100%;
  text-align: center;
}
/* The dots/bullets/indicators */
.dot {
  height: 15px;
  width: 15px;
  margin: 0 2px;
  background-color: #bbb;
  border-radius: 50%;
  display: inline-block;
```

```
  transition: background-color 0.6s ease;
}
.active {
  background-color: #FCAD98;
}
/* Fading animation */
.fade {
  -webkit-animation-name: fade;
  -webkit-animation-duration: 1.5s;
  animation-name: fade;
  animation-duration: 1.5s;
}
@-webkit-keyframes fade {
  from {opacity: .4}
  to {opacity: 1}
}
@keyframes fade {
  from {opacity: .4}
  to {opacity: 1}
}
/* On smaller screens, decrease text size */
@media only screen and (max-width: 300px) {
  .text {font-size: 11px}
}
```

```
.bar{
margin: 0px;
padding:20px;
background-color:white;
opacity:0.6;
color:black;
font-family:'Roboto',sans-serif;
font-style: italic;
border-radius:20px;
font-size:25px;
}
a{
color:grey;
float:right;
text-decoration:none;
font-style:normal;
padding-right:20px;
}
a:hover{
background-color:black;
color:white;
border-radius:15px;
font-size:30px;
padding-left:10px;
}
```

```
p{
color:black;
font-style:italic;
font-size:30px;
}
</style>
</head>
<body style="background-
image:url({{url_for('static',filename='images/bck3.png')}});background-position:
center;background-repeat: no-repeat;
        background-size: cover;">
<div class="header">
<div style="width:50%;float:left;font-size:2vw;text-align:left;color:black; padding-
top:1%;padding-left:5%;">Real Time Communication System for Deaf & Dumb</div>
  <div class="topnav-right"style="padding-top:0.5%;">
    <a  href="/home">Home</a>
    <a class="active" href="/upload">Open Web Cam</a>
  </div></div>
 <div>
  <div class="text">
    <img src="{{ url_for('video') }}" width="50%"/>
  </div>
</div>
</body>
</html>
```

12.8 FLASK.PY

```
from flask import Flask,render_template,Response
import pyttsx3
import os
import cv2
import numpy as np
import time
import mediapipe as mp
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
app=Flask(__name__)
mp_holistic = mp.solutions.holistic # Holistic model
mp_drawing = mp.solutions.drawing_utils # Drawing utilities
mp_pose = mp.solutions.pose
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)
                    )
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])
actions = np.array(['exmonsoon','fall','season','spring','summer','winter'])
# 1. New detection variables
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.callbacks import TensorBoard
model = Sequential()
model.add(LSTM(64, return_sequences=True, activation='relu', input_shape=(60,1662)))
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'))
model.compile(optimizer='Adam', loss='categorical_crossentropy',
metrics=['categorical_accuracy'])
model.load_weights('model.h5')
sequence = []
sentence = []
predictions = []
```

```
threshold = 0.5
cap=cv2.VideoCapture(0)
def generate_frames():
    with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5)
as holistic:
        global sequence
        global sentence
        global predictions
        global threshold
        while True:
        # Read feed
            success, frame = cap.read()
            if not success:
                break
            else:
            # 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[-60:]
                if len(sequence) == 60:
```

```
            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)
            engine = pyttsx3.init()
            engine.say(sentence)
            engine.runAndWait()
            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)
        if cv2.waitKey(10) & 0xFF == ord('q'):
```

```
            break
        # cap.release()
        # cv2.destroyAllWindows()
        ret,buffer=cv2.imencode('.jpg',image)
        s=buffer.tobytes()
        yield(b'--frame\r\n'
        b'Content-Type: image/jpeg\r\n\r\n' + s + b'\r\n')
@app.route('/')
def index():
    return render_template('home.html')
@app.route('/upload')
def upload():
    return render_template('upload.html')
@app.route('/video')
def video():
    return Response(generate_frames(),mimetype='multipart/x-mixed-replace;
boundary=frame')
if __name__=="__main__":
    app.run(debug=True)
```