AI BODY LANGUAGE DECODER USING MEDIAPIPE

PROJECT REPORT

1. ABSTRACT

The communication of non-verbal cues through body language involves the expressive movements of hands, face, and the body. This project focuses on evaluating representations based on skeletal poses, prioritizing explainability, person-independence, privacy preservation, and low-dimensional features. Skeletal representations offer generalization over individual appearance and background, emphasizing the recognition of motion.

Our approach involves a real-time, on-device body tracking pipeline that predicts both hand skeleton and overall body pose. Leveraging MediaPipe, a versatile framework for developing cross-platform machine learning solutions, we integrate a system that excels in pose estimation. We assess the effectiveness of existing pose estimation systems in body language recognition, scrutinizing failure cases to refine models.

The proposed system and architecture showcase real-time inference capabilities and deliver high-quality predictions, emphasizing the practical applicability of our approach to understanding and interpreting body language in diverse scenarios.

Keywords - Body Landmarks, MediaPipe, Body Language, Prediction, Accuracy, Real-time on-device Tracking, Pose Estimation, Recognition.

2. INTRODUCTION

The Body Language Decoder serves as a valuable tool for discerning and anticipating facial expressions, hand gestures, and body posture. The ability to recognize facial expressions can significantly expedite data scaling and analysis for market research companies. The capacity to interpret hand shape and motion plays a pivotal role in enhancing user experiences across various technological domains, such as sign language understanding, hand posture control, and the augmentation of digital content onto the physical world in augmented reality.

The MediaPipe Body Landmark model, leveraging Blaze Pose, provides high-fidelity body pose tracking by deducing 33 2D landmarks on the body (or 25 upper-body landmarks) from RGB video frames. This model detects landmarks for each body pose, defaulting to full-body but configurable for upper-body tracking, predicting the first 25 landmarks in such cases.

MediaPipe Hands offers a sophisticated solution for hand and finger tracking, inferring up to 21 3D hand landmarks in a single frame. Operating as a hybrid model, it combines a palm/hand detection model for the entire image with an oriented hand bounding box and a hand landmark model for the cropped region defined by the palm detector, yielding high-fidelity 3D hand key points. This module can detect landmarks for a single hand or both hands, depending on the specified type.

Additionally, MediaPipe Face Mesh accurately estimates 468 3D face landmarks in real-time on mobile devices, utilizing deep neural networks to infer surface geometry from a single camera input without requiring a dedicated depth sensor. Its applications span driver drowsiness detection, sign language interpretation, market research data analysis, and body language assessment in interviews.

In the realm of sentiment analysis, which involves detecting positive or negative sentiment in a sentence, businesses commonly employ this process to assess sentiment in social data, manage brand reputation, and comprehend customer feedback. While sentiment analysis is widely used in the digital world, measuring user reactions and expressions in offline settings, such as retail stores, showrooms, and social media, poses a challenge. AI-powered emotion detection from facial expressions or text emerges as a viable solution to automatically gauge consumer engagement with content and brands in these scenarios.

Moreover, the detection of body language, recognized as one of the most effective communication methods, can aid interviewers in analyzing candidates during the interview process. This can be achieved by generating an Excel sheet of coordinate points for desired poses' landmarks and training a model on this data. The trained model can then be stored and utilized later for predicting user gestures.

2. LITERATURE SURVEY

2.1 Existing problem

i) Real-Time Processing:
- Challenge: Achieving real-time processing for comprehensive body language decoding can be demanding, especially when dealing with multiple input sources such as face, hands, and body simultaneously.
- Considerations: Explore optimizations and efficient algorithms to ensure the system's responsiveness in real-world applications.

ii) Integration of Multiple Modules:
- Challenge: Integrating separate models for body, face, and hand components can be complex, requiring synchronization and coordination among these modules.
- Considerations: Enhance the integration process to ensure seamless communication and synchronization between the individual components of the system.

iii) Accuracy and Precision:
- Challenge: Achieving high accuracy and precision in recognizing subtle nuances of body language, facial expressions, and hand gestures is a persistent challenge.
- Considerations: Continuously refine and train models with diverse datasets to improve accuracy, particularly in capturing fine-grained details of human expression.

iv) Adaptability to Varied Environments:
- Challenge: The system may struggle with adaptability to diverse environments, lighting conditions, and user appearances.
- Considerations: Implement techniques for robustness, such as data augmentation during training and adapting the system to handle variations in environmental factors.

v) Privacy Concerns:
- Challenge: Analysing and decoding body language raises privacy concerns, particularly in public spaces or applications that involve sensitive interactions.
- Considerations: Implement privacy-preserving measures, such as anonymizing data or providing user-controlled settings for information sharing.

vi) Limited Dataset Diversity:

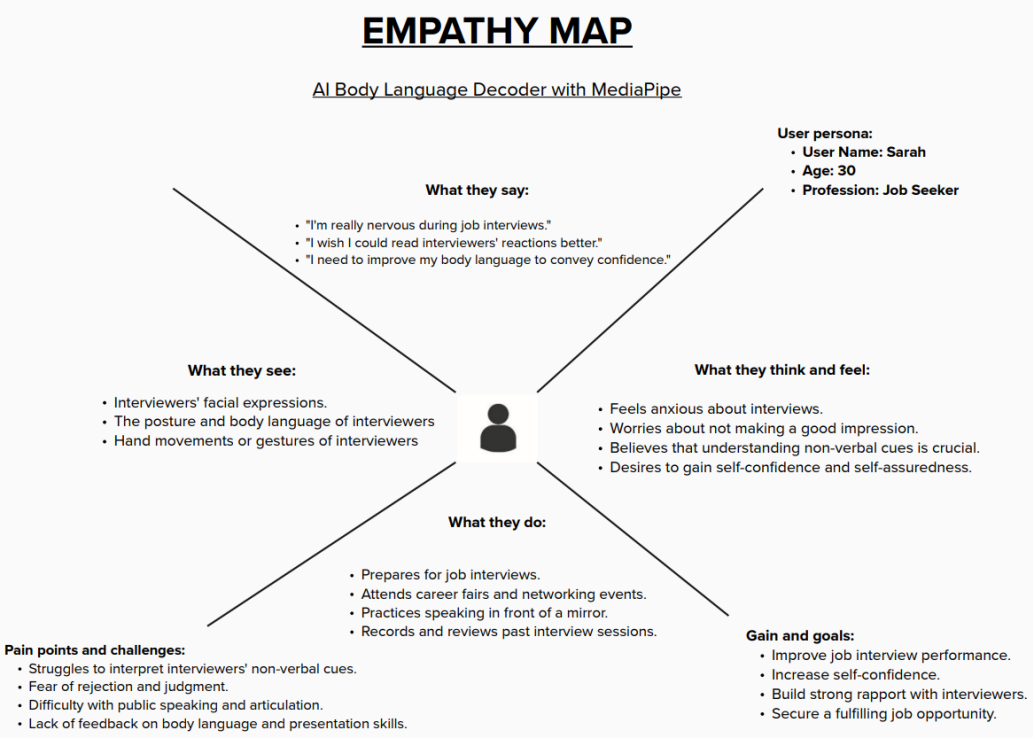
- Challenge: Limited diversity in training datasets may result in biased models that struggle to generalize well across different demographics and cultural contexts.
- Considerations: Continuously expand and diversify training datasets to ensure the model's ability to understand a wide range of body language expressions.

2.2 Problem Statement Definition

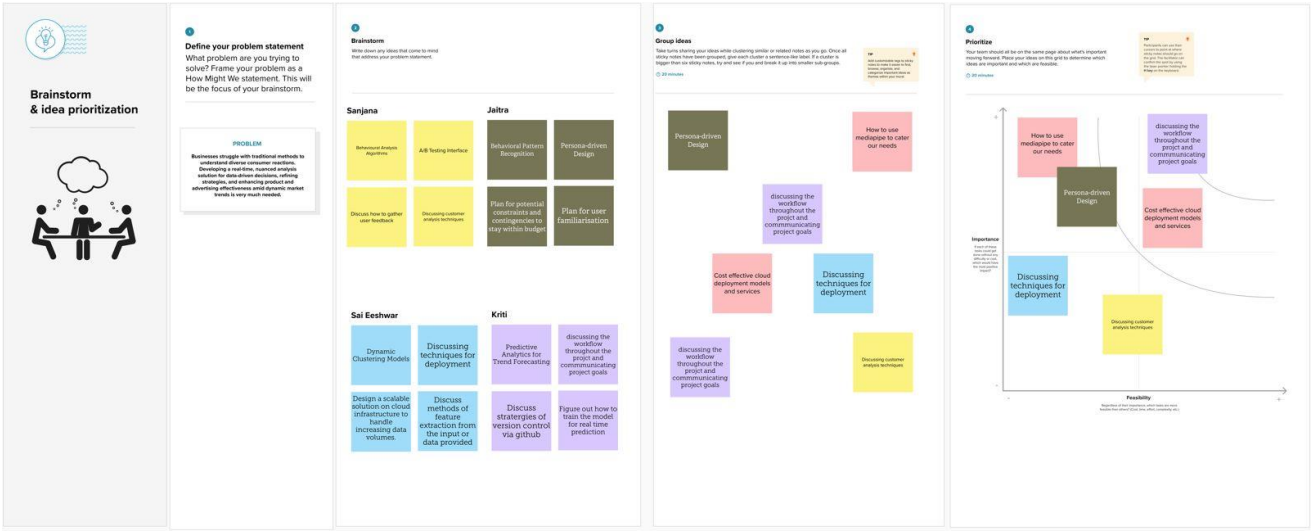
The field of human-computer interaction has seen significant advancements with the integration of artificial intelligence (AI) technologies. One intriguing avenue of exploration is the development of an AI Body Language Decoder – a system capable of real-time interpretation and analysis of human body language, encompassing facial expressions, hand gestures, and overall posture. This project aims to address the following key challenges in the creation of an effective and versatile AI Body Language Decoder.

3. IDEATION & PROPOSED SOLUTION

3.1 Empathy Map Canvas



3.2 Ideation & Brainstorming



4. REQUIREMENT ANALYSIS

4.1 Functional requirement

- Multimodal Real-Time Processing: The system must be capable of simultaneously processing and interpreting information from multiple modalities, including facial expressions, hand gestures, and overall body posture.

- Specialized Model Integration: Integration of specialized models for face, hand, and body components, ensuring cohesive functioning while maintaining real-time performance.

- Accuracy and Precision: Achieving high accuracy and precision in decoding subtle nuances of body language, considering the variability in expressions and gestures across diverse demographics.

- Adaptability to Varied Environments: The system may struggle with adaptability to diverse environments, lighting conditions, and user appearances. Changes in illumination, background noise, and variations in user characteristics can affect the system's performance.

- Privacy and Ethical Considerations: Addressing privacy concerns related to the collection and analysis of personal body language data and ensuring ethical usage to avoid biases or stereotypes.

- User Interaction Design: Designing a user interface that effectively communicates the AI's interpretation of body language, ensuring user comprehension and usability.

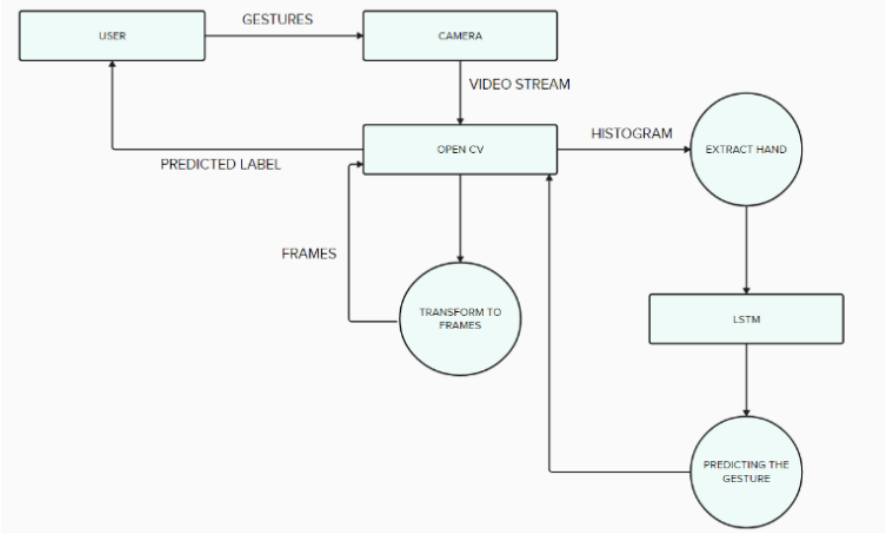
4.2 Non-Functional requirements

- Performance: The system must provide real-time processing capabilities with low latency, ensuring quick and seamless interpretation of body language cues.
- Reliability: The system must demonstrate high reliability, ensuring consistent and accurate interpretation of body language across varied conditions and user scenarios.
- Scalability: The architecture must be scalable to accommodate potential increases in user base or data volume without compromising performance.
- Security: The system must prioritize user privacy, implementing robust security measures to protect sensitive body language data.
- Usability: The user interface must be intuitive and user-friendly, ensuring that users can easily understand and interact with the system.
- Maintainability: The system must be designed for ease of maintenance and updates to accommodate evolving technologies and user needs.
- Compatibility: The system must be compatible with a variety of devices and platforms to maximize accessibility for users.
- Ethical and Legal Compliance: The system must adhere to ethical guidelines and legal regulations concerning user privacy, data handling, and the responsible use of AI technologies.

5. PROJECT DESIGN

5.1 Data Flow Diagrams & User Stories

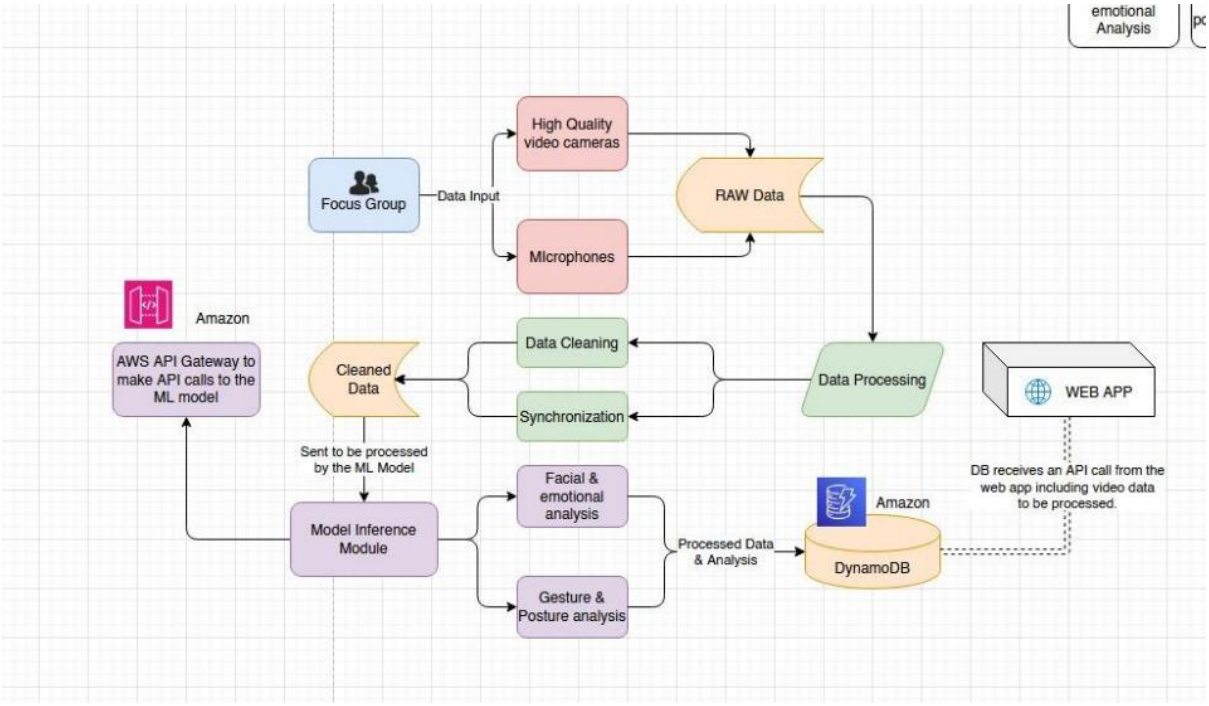
A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It shows how data enters and leaves the system, what changes the information, and where data is stored.



User Stories:

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Regular User	Live Video Capture	US01	Start the AI Body Language Detector and capture live video.	The system provides an interface to start the detector, capturing live video from the user's camera.	High	V1.0
Regular User	Gesture Recognition	US02	Detect and analyse basic body language gestures in real-time.	The detector accurately identifies and highlights key body parts involved in basic gestures, with real-time responsiveness.	High	V1.0
Regular User	Pose Recognition	US03	Recognize specific body poses (e.g., standing, sitting).	The system clearly indicates recognized body poses and accurately distinguishes between different poses.	Medium	V1.1
Regular User	Emotional Analysis	US04	Interpret and display emotions based on body language.	The system provides feedback indicating detected emotions based on body language cues.	Medium	V1.1
System Admin	Data Storage	US05	Store historical body language data for analysis and improvement.	The system has a mechanism to store historical body language data. The stored data should be accessible for further analysis.	High	V1.1

5.2 Solution Architecture:



6. PROJECT PLANNING & SCHEDULING

6.1 Technical Architecture

Fig. 6.1.1 Flowchart of working model using IDE:

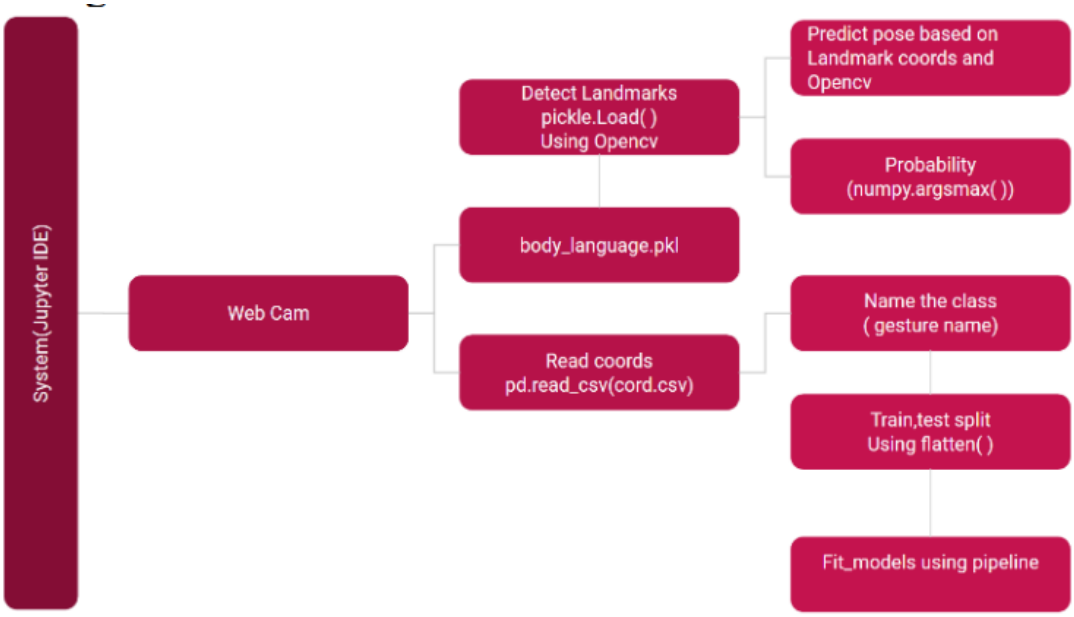
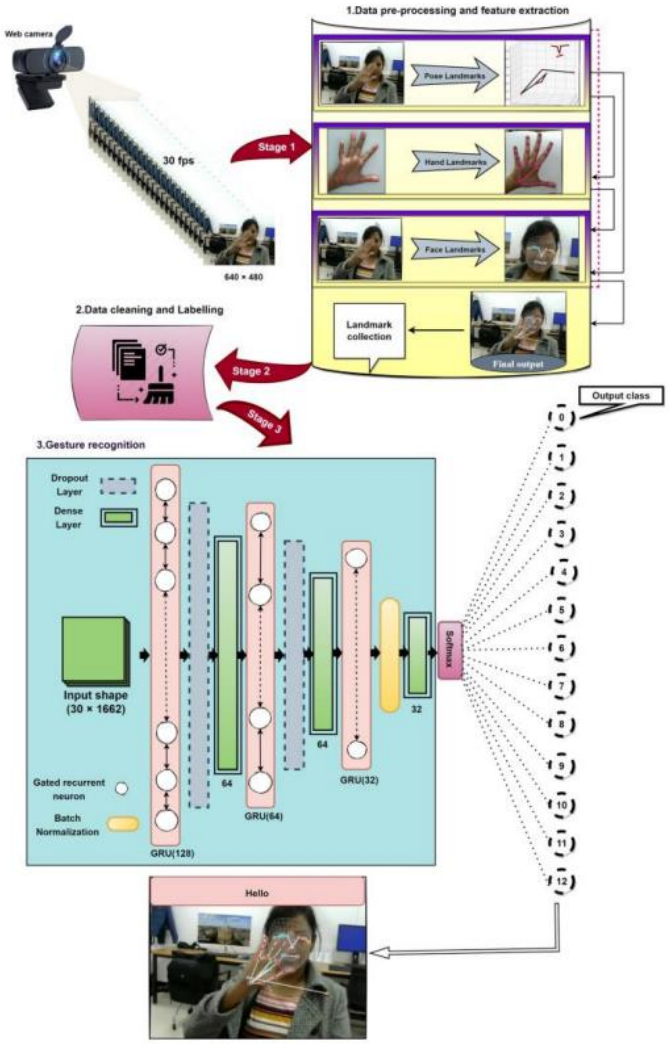


Fig 6.1.2 Overview of technical Architecture:



6.2 Sprint Planning & Estimation

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint 1	Live Video Capture	US01	Start the AI Body Language Detector and capture live video.	5	High	Developer A, Developer B
Sprint 2	Gesture Recognition	US02	Detect and analyse basic body language gestures in real-time.	8	High	Developer C, Developer D
Sprint 3	Pose Recognition	US03	Recognize specific body	5	Medium	Developer B, Developer E

			poses (e.g., standing, sitting).			
Sprint 4	Emotional Analysis	US04	Interpret and display emotions based on body language.	8	Medium	Developer A, Developer D
Sprint 5	Data Storage	US05	Store historical body language data for analysis and improvement.	5	High	Developer C, Developer E

6.3 Sprint Delivery Schedule

Sprint	Start Date	End Date	User Stories Completed	Notes
Sprint 1	11-11-2023	15-11-2023	US01, US02	Initial setup, live video capture, and basic gesture analysis.
Sprint 2	12-11-2023	16-11-2023	US03, US04	Recognition of specific body poses and emotional analysis.
Sprint 3	13-11-2023	17-11-2023	US05	Implementation of data storage for historical analysis.

7. CODING & SOLUTIONING (Explaining the features added in the project along with code)

System Modules:

(a) Install and import dependencies: In this step Mediapipe, OpenCV, pandas and scikit-learn are installed and imported.

(b) Make some detections: Here a webcam window is opened using cv2 and specifications like colour and thickness of our face, hand and pose landmarks are mentioned/modified.

(c) Capture landmarks and export to CSV: Here various coordinates of facial expressions and body gestures are captured and exported to a csv file named coords.csv.

(d) Train custom model using scikit learn: Here the collected data is read and processed to train our machine learning classification model on it. The model is then evaluated and serialized.

(e) Make detections with model: Now the code, when run, can make predictions of the user's gestures using the above trained model.

CODES FOR BUILDING THE MODEL AND WEBPAGE:

Install several Python packages using the pip command

```
!pip install mediapipe opencv-python pandas scikit-learn
```

```
Requirement already satisfied: mediapipe in c:\users\sanja\anaconda3\lib\site-packages (0.10.7)
Requirement already satisfied: opencv-python in c:\users\sanja\anaconda3\lib\site-packages (4.8.1.78)
Requirement already satisfied: pandas in c:\users\sanja\anaconda3\lib\site-packages (1.4.4)
Requirement already satisfied: scikit-learn in c:\users\sanja\anaconda3\lib\site-packages (1.0.2)
Requirement already satisfied: matplotlib in c:\users\sanja\anaconda3\lib\site-packages (from mediapipe) (3.5.2)
Requirement already satisfied: attrs>=19.1.0 in c:\users\sanja\anaconda3\lib\site-packages (from mediapipe) (21.4.0)
Requirement already satisfied: protobuf<4,>=3.11 in c:\users\sanja\anaconda3\lib\site-packages (from mediapipe) (3.20.3)
Requirement already satisfied: flatbuffers>=2.0 in c:\users\sanja\anaconda3\lib\site-packages (from mediapipe) (23.5.26)
Requirement already satisfied: numpy in c:\users\sanja\anaconda3\lib\site-packages (from mediapipe) (1.24.4)
Requirement already satisfied: absl-py in c:\users\sanja\anaconda3\lib\site-packages (from mediapipe) (2.0.0)
```

Imported the necessary libraries, Mediapipe and OpenCV-python, for the project.

```
In [16]: import mediapipe as mp # Import mediapipe
         import cv2 # Import opencv
```

Imported the drawing_utils and holistic modules from Mediapipe. These modules can be helpful for drawing landmarks and connecting lines on the image as well as for using the holistic model for body pose estimation.

```
In [17]: mp_drawing = mp.solutions.drawing_utils # Drawing helpers
         mp_holistic = mp.solutions.holistic # Mediapipe Solutions
```

Integrated the MediaPipe holistic model into your script for real-time body pose detection. The code here captures video from the webcam, processes each frame using the holistic model, and overlays the detected landmarks on the image for the face, right hand, left hand, and overall body pose.

```
In [4]: cap = cv2.VideoCapture(0)

# initialize the model
with mp_holistic.Holistic(min_detection_confidence=0.5,min_tracking_confidence=0.5) as holistic:

    while cap.isOpened():
        ret , frame = cap.read()

        image = cv2.cvtColor(frame , cv2.COLOR_BGR2RGB)

        image.flags.writeable = False

        results = holistic.process(image)

        # print(results.pose_landmarks)
        image.flags.writeable = True

        image = cv2.cvtColor(image , cv2.COLOR_RGB2BGR)

        # face
        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

        # right hand
        mp_drawing.draw_landmarks(image , results.right_hand_landmarks , mp_holistic.HAND_CONNECTIONS,
        mp_drawing.DrawingSpec(color=(80,22,10) ,thickness = 2 ,circle_radius=4 ) ,          mp_drawing.DrawingSpec(color=(80,44,

        # left hand
        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

        # pose detection
        mp_drawing.draw_landmarks(image , results.pose_landmarks , mp_holistic.POSE_CONNECTIONS,
        mp_drawing.DrawingSpec(color=(245,117,66) ,thickness = 2 ,circle_radius=4 ) ,        mp_drawing.DrawingSpec(color=(245,66
```

```
        cv2.imshow("Holistic Model Detections",image)

        if cv2.waitKey(30) & 0xFF == ord('q'):
            break

cap.release()
cv2.destroyAllWindows()
```

'results' is the variable that holds the output of the MediaPipe holistic model after processing each frame

```
In [5]: results
Out[5]: mediapipe.python.solution_base.SolutionOutputs
```

The 'results.pose_landmarks' attribute in the MediaPipe holistic model contains the 3D landmarks for the full body pose.

```
In [6]: results.pose_landmarks
Out[6]: landmark {
          x: 0.4710385203361511
          y: 0.5277755856513977
          z: -1.7152698040008545
          visibility: 0.9994410276412964
        }
        landmark {
          x: 0.5034003257751465
          y: 0.4536381661891937
          z: -1.6668291091918945
          visibility: 0.9991921186447144
        }
        landmark {
          x: 0.5298535227775574
          y: 0.4500362277030945
          z: -1.6671292781829834
          visibility: 0.9992175102233887
        }
        landmark {
```

In the MediaPipe holistic model, each landmark has a visibility attribute that represents the visibility of the landmark in the detected image.

```
In [7]: results.face_landmarks.landmark[0].visibility
Out[7]: 0.0
```

i)Capture Landmarks & Export to CSV

```
In [8]: import csv
        import os
        import numpy as np
```

Calculating the total number of coordinates by summing the number of landmarks from both the pose and face landmarks detected by the MediaPipe holistic model.

In [9]:
```
num_coords = len(results.pose_landmarks.landmark)+len(results.face_landmarks.landmark)
num_coords
```

Out[9]:
```
501
```

Creating a list called landmarks that will be used to store information about each coordinate.

In [10]:
```
landmarks = ['class']
for val in range(1, num_coords+1):
    landmarks += ['x{}'.format(val), 'y{}'.format(val), 'z{}'.format(val), 'v{}'.format(val)]
```

In [11]:
```
landmarks
```

Out[11]:
```
['class',
 'x1',
 'y1',
 'z1',
 'v1',
 'x2',
 'y2',
 'z2',
 'v2',
 'x3',
 'y3',
 'z3',
 'v3',
 'x4',
 'y4',
 'z4',
 'v4',
 'x5',
 'y5',
```

Using the csv module to write the header (column names) to a CSV file named 'coords.csv'.

In [12]:
```
with open('coords.csv', mode='w', newline='') as f:
    csv_writer = csv.writer(f, delimiter=',', quotechar='"', quoting=csv.QUOTE_MINIMAL)
    csv_writer.writerow(landmarks)
```

Initializing a variable named class_name with the value "Happy" and visualizing the landmarks, exporting the coordinates of the face and pose landmarks, along with the class name, to a CSV file ('coords.csv').

In [13]:
```
class_name = "Happy"
```

In [14]:
```
cap = cv2.VideoCapture(0)
# Initiate holistic model
with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:

    while cap.isOpened():
        ret, frame = cap.read()

        # Recolor Feed
        image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        image.flags.writeable = False

        # Make Detections
        results = holistic.process(image)
        # print(results.face_landmarks)

        # face_landmarks, pose_landmarks, left_hand_landmarks, right_hand_landmarks

        # Recolor image back to BGR for rendering
        image.flags.writeable = True
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

        # 1. Draw face landmarks
        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)
                                  )

        # 2. Right hand
        mp_drawing.draw_landmarks(image, results.right_hand_landmarks, mp_holistic.HAND_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)
                                  )
```

```
        # 3. Left Hand
        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)
                                  )

        # 4. Pose Detections
        mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_holistic.POSE_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)
                                  )
        # Export coordinates
        try:
            # Extract Pose landmarks
            pose = results.pose_landmarks.landmark
            pose_row = list(np.array([[landmark.x, landmark.y, landmark.z, landmark.visibility] for landmark in pose]).flatten())

            # Extract Face landmarks
            face = results.face_landmarks.landmark
            face_row = list(np.array([[landmark.x, landmark.y, landmark.z, landmark.visibility] for landmark in face]).flatten())

            # Concate rows
            row = pose_row+face_row

            # Append class name
            row.insert(0, class_name)

            # Export to CSV
            with open('coords.csv', mode='a', newline='') as f:
                csv_writer = csv.writer(f, delimiter=',', quotechar='"', quoting=csv.QUOTE_MINIMAL)
                csv_writer.writerow(row)

        except:
            pass

        cv2.imshow('Raw Webcam Feed', image)
```

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

cap.release()
cv2.destroyAllWindows()
```

Initializing class_name to "Sad", the script will now append data with this class label to the 'coords.csv' file during its execution.

In [15]:
```
class_name = "Sad"
```

In [16]:
```
cap = cv2.VideoCapture(0)
# Initiate holistic model
with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:

    while cap.isOpened():
        ret, frame = cap.read()

        # Recolor Feed
        image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        image.flags.writeable = False

        # Make Detections
        results = holistic.process(image)
        # print(results.face_landmarks)

        # face_landmarks, pose_landmarks, left_hand_landmarks, right_hand_landmarks

        # Recolor image back to BGR for rendering
        image.flags.writeable = True
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

        # 1. Draw face landmarks
        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)
                                  )

        # 2. Right hand
        mp_drawing.draw_landmarks(image, results.right_hand_landmarks, mp_holistic.HAND_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)
                                  )
```

```
        # 3. Left Hand
        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)
                                  )

        # 4. Pose Detections
        mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_holistic.POSE_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)
                                  )
        # Export coordinates
        try:
            # Extract Pose landmarks
            pose = results.pose_landmarks.landmark
            pose_row = list(np.array([[landmark.x, landmark.y, landmark.z, landmark.visibility] for landmark in pose]).flatten())

            # Extract Face landmarks
            face = results.face_landmarks.landmark
            face_row = list(np.array([[landmark.x, landmark.y, landmark.z, landmark.visibility] for landmark in face]).flatten())

            # Concate rows
            row = pose_row+face_row

            # Append class name
            row.insert(0, class_name)

            # Export to CSV
            with open('coords.csv', mode='a', newline='') as f:
                csv_writer = csv.writer(f, delimiter=',', quotechar='"', quoting=csv.QUOTE_MINIMAL)
                csv_writer.writerow(row)

        except:
            pass

        cv2.imshow('Raw Webcam Feed', image)
```

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

cap.release()
cv2.destroyAllWindows()
```

Initializing class_name to "Victorious", the script will now append data with this class label to the 'coords.csv' file during its execution.

In [19]:
```
class_name = "Victorious"
```

In [20]:
```
cap = cv2.VideoCapture(0)
# Initiate holistic model
with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:

    while cap.isOpened():
        ret, frame = cap.read()

        # Recolor Feed
        image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        image.flags.writeable = False

        # Make Detections
        results = holistic.process(image)
        # print(results.face_landmarks)

        # face_landmarks, pose_landmarks, left_hand_landmarks, right_hand_landmarks

        # Recolor image back to BGR for rendering
        image.flags.writeable = True
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

        # 1. Draw face landmarks
        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)
                                  )

        # 2. Right hand
        mp_drawing.draw_landmarks(image, results.right_hand_landmarks, mp_holistic.HAND_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)
                                  )
```

```
        # 3. Left Hand
        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)
                                  )

        # 4. Pose Detections
        mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_holistic.POSE_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)
                                  )
        # Export coordinates
        try:
            # Extract Pose landmarks
            pose = results.pose_landmarks.landmark
            pose_row = list(np.array([[landmark.x, landmark.y, landmark.z, landmark.visibility] for landmark in pose]).flatten())

            # Extract Face landmarks
            face = results.face_landmarks.landmark
            face_row = list(np.array([[landmark.x, landmark.y, landmark.z, landmark.visibility] for landmark in face]).flatten())

            # Concate rows
            row = pose_row+face_row

            # Append class name
            row.insert(0, class_name)

            # Export to CSV
            with open('coords.csv', mode='a', newline='') as f:
                csv_writer = csv.writer(f, delimiter=',', quotechar='"', quoting=csv.QUOTE_MINIMAL)
                csv_writer.writerow(row)
```

```
        except:
            pass

        cv2.imshow('Raw Webcam Feed', image)

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

cap.release()
cv2.destroyAllWindows()
```

Initializing class_name to "Fight", the script will now append data with this class label to the 'coords.csv' file during its execution.

In [21]:
```
class_name = "Fight"
```

In [22]:
```
cap = cv2.VideoCapture(0)
# Initiate holistic model
with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:

    while cap.isOpened():
        ret, frame = cap.read()

        # Recolor Feed
        image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        image.flags.writeable = False

        # Make Detections
        results = holistic.process(image)
        # print(results.face_landmarks)

        # face_landmarks, pose_landmarks, left_hand_landmarks, right_hand_landmarks

        # Recolor image back to BGR for rendering
        image.flags.writeable = True
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

        # 1. Draw face landmarks
        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)
                                  )

        # 2. Right hand
        mp_drawing.draw_landmarks(image, results.right_hand_landmarks, mp_holistic.HAND_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)
                                  )
```

```
        # 3. Left Hand
        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)
                                  )

        # 4. Pose Detections
        mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_holistic.POSE_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)
                                  )
        # Export coordinates
        try:
            # Extract Pose landmarks
            pose = results.pose_landmarks.landmark
            pose_row = list(np.array([[landmark.x, landmark.y, landmark.z, landmark.visibility] for landmark in pose]).flatten())

            # Extract Face landmarks
            face = results.face_landmarks.landmark
            face_row = list(np.array([[landmark.x, landmark.y, landmark.z, landmark.visibility] for landmark in face]).flatten())

            # Concate rows
            row = pose_row+face_row

            # Append class name
            row.insert(0, class_name)

            # Export to CSV
            with open('coords.csv', mode='a', newline='') as f:
                csv_writer = csv.writer(f, delimiter=',', quotechar='"', quoting=csv.QUOTE_MINIMAL)
                csv_writer.writerow(row)
```

```
        except:
            pass

        cv2.imshow('Raw Webcam Feed', image)

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

cap.release()
cv2.destroyAllWindows()
```

ii) Train Custom Model Using Scikit Learn

Read in Collected Data and Process

Imported the pandas library as pd and the train_test_split function from sklearn.model_selection.

```
In [23]: import pandas as pd
         from sklearn.model_selection import train_test_split
```

Used the pd.read_csv function from the pandas library to read data from the 'coords.csv' file and create a DataFrame named df.

```
In [24]: df = pd.read_csv('coords.csv')
```

```
In [25]: df.head()
```

Out[25]:

	class	x1	y1	z1	v1	x2	y2	z2	v2	x3	...	z499	v499	x500	y500	z500	v500	
0	Happy	0.495655	0.460593	-1.167008	0.999972	0.534861	0.393980	-1.106970	0.999941	0.557452	...	0.001500	0.0	0.586997	0.371252	0.024953	0.0	0
1	Happy	0.495817	0.456177	-1.244716	0.999967	0.534145	0.390922	-1.170501	0.999933	0.557030	...	0.000472	0.0	0.584488	0.375723	0.020989	0.0	0
2	Happy	0.497176	0.453167	-1.305596	0.999951	0.533907	0.388751	-1.232236	0.999905	0.556852	...	0.001796	0.0	0.582349	0.378712	0.023489	0.0	0
3	Happy	0.497807	0.452978	-1.336381	0.999938	0.533542	0.388488	-1.267701	0.999883	0.556555	...	0.001722	0.0	0.583654	0.379514	0.024056	0.0	0
4	Happy	0.497936	0.453001	-1.369595	0.999925	0.533267	0.388378	-1.299591	0.999862	0.556372	...	0.000236	0.0	0.584783	0.376571	0.022291	0.0	0

5 rows × 2005 columns

```
In [26]: df.tail()
```

Out[26]:

	class	x1	y1	z1	v1	x2	y2	z2	v2	x3	...	z499	v499	x500	y500	z500	v500
3722	Fight	0.405397	0.374829	-1.110246	0.999843	0.445683	0.301851	-1.098439	0.999762	0.475914	...	-0.012665	0.0	0.517114	0.295852	0.002268	0.0
3723	Fight	0.405655	0.376815	-1.082071	0.999854	0.447153	0.304375	-1.067877	0.999774	0.477437	...	-0.013168	0.0	0.521496	0.297059	0.002022	0.0
3724	Fight	0.405613	0.376172	-0.969324	0.999855	0.447231	0.303774	-0.954455	0.999770	0.477735	...	-0.011655	0.0	0.518651	0.302048	0.004402	0.0
3725	Fight	0.407560	0.393380	-1.238061	0.999861	0.448479	0.317040	-1.193756	0.999775	0.478336	...	-0.011794	0.0	0.514961	0.323503	0.009872	0.0
3726	Fight	0.402376	0.402691	-1.354719	0.999861	0.440341	0.323980	-1.311067	0.999775	0.471137	...	-0.011932	0.0	0.505385	0.322433	0.010070	0.0

5 rows × 2005 columns

Using boolean indexing to filter rows in the DataFrame where the value in the 'class' column is equal to 'Sad'.

```
In [27]: df[df['class']=='Sad']
```

Out[27]:

	class	x1	y1	z1	v1	x2	y2	z2	v2	x3	...	z499	v499	x500	y500	z500	v50
944	Sad	0.494555	0.496568	-1.304255	0.999854	0.532973	0.434768	-1.242087	0.999679	0.554859	...	-0.002931	0.0	0.579079	0.432162	0.019825	0.
945	Sad	0.496238	0.499725	-1.345671	0.999855	0.533064	0.437296	-1.277252	0.999684	0.555008	...	-0.003496	0.0	0.584158	0.430011	0.018819	0.
946	Sad	0.498341	0.502023	-1.367751	0.999859	0.533279	0.439050	-1.298056	0.999697	0.555172	...	-0.004398	0.0	0.582418	0.430202	0.016630	0.
947	Sad	0.498273	0.499288	-1.340260	0.999858	0.530767	0.435472	-1.267860	0.999698	0.552943	...	-0.001356	0.0	0.571141	0.422916	0.017539	0.
948	Sad	0.498160	0.495624	-1.333367	0.999858	0.529912	0.431722	-1.258193	0.999701	0.552190	...	-0.001452	0.0	0.567905	0.415834	0.018156	0.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
1418	Sad	0.492832	0.553343	-1.490562	0.999327	0.525294	0.482845	-1.447301	0.998745	0.547714	...	-0.023812	0.0	0.576488	0.482443	-0.009857	0.
1419	Sad	0.492719	0.549848	-1.432481	0.999372	0.525086	0.478673	-1.384969	0.998828	0.547439	...	-0.023679	0.0	0.578846	0.478692	-0.009101	0.
1420	Sad	0.492303	0.549329	-1.520523	0.999400	0.524501	0.478426	-1.478714	0.998871	0.546890	...	-0.024098	0.0	0.578340	0.483202	-0.009366	0.
1421	Sad	0.492169	0.549643	-1.493320	0.999412	0.524356	0.478866	-1.452610	0.998890	0.546764	...	-0.023773	0.0	0.578457	0.481620	-0.009471	0.
1422	Sad	0.491823	0.551777	-1.543472	0.999425	0.523651	0.480759	-1.505510	0.998902	0.546101	...	-0.024243	0.0	0.578403	0.485280	-0.009344	0.

479 rows × 2005 columns

Preparing data for a machine learning task, where X represents the features (independent variables) and y represents the target variable (dependent variable).

```
In [28]: X = df.drop('class', axis=1) # features
         y = df['class'] # target value
```

Splitting dataset into training and testing sets

```
In [29]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1234)
```

```
In [30]: y_test
```

```
Out[30]: 219          Happy
         1547    Victorious
         149          Happy
         1415           Sad
         2763    Victorious
                    ...
         3265         Fight
         2466    Victorious
         71           Happy
         62           Happy
         3646         Fight
         Name: class, Length: 1119, dtype: object
```

iii) Train Machine Learning Classification Model:

Importing modules and classes from scikit-learn to set up a pipeline and use different classifiers

```
In [31]: from sklearn.pipeline import make_pipeline
         from sklearn.preprocessing import StandardScaler

         from sklearn.linear_model import LogisticRegression, RidgeClassifier
         from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
```

Created a dictionary of pipelines, each using StandardScaler for feature scaling and a different classifier

```
In [32]: pipelines = {
             'lr':make_pipeline(StandardScaler(), LogisticRegression()),
             'rc':make_pipeline(StandardScaler(), RidgeClassifier()),
             'rf':make_pipeline(StandardScaler(), RandomForestClassifier()),
             'gb':make_pipeline(StandardScaler(), GradientBoostingClassifier()),
         }
```

To access the first pipeline in the dictionary

```
In [33]: list(pipelines.values())[0]
```

```
Out[33]: Pipeline(steps=[('standardscaler', StandardScaler()),
                         ('logisticregression', LogisticRegression())])
```

Iterating through the dictionary of pipelines ('pipelines') and fitting each pipeline on your training data.

In [32]:
```
fit_models = {}
for algo , pipeline in pipelines.items():
    model = pipeline.fit(X_train , y_train)
    fit_models[algo] = model
```

```
C:\Users\sanja\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

In [33]:
```
fit_models
```

Out[33]:
```
{'lr': Pipeline(steps=[('standardscaler', StandardScaler()),
                       ('logisticregression', LogisticRegression())]),
 'rc': Pipeline(steps=[('standardscaler', StandardScaler()),
                       ('ridgeclassifier', RidgeClassifier())]),
 'rf': Pipeline(steps=[('standardscaler', StandardScaler()),
                       ('randomforestclassifier', RandomForestClassifier())]),
 'gb': Pipeline(steps=[('standardscaler', StandardScaler()),
                       ('gradientboostingclassifier', GradientBoostingClassifier())])}
```

Using the Ridge Classifier model ('rc') from fit_models dictionary to make predictions on the testing set ('X_test')

In [34]:
```
fit_models['rc'].predict(X_test)
```

Out[34]:
```
array(['Sad', 'Fight', 'Sad', 'Fight', 'Happy', 'Happy', 'Happy',
       'Victorious', 'Fight', 'Fight', 'Happy', 'Fight', 'Happy', 'Fight',
       'Fight', 'Fight', 'Sad', 'Victorious', 'Sad', 'Fight', 'Sad',
       'Fight', 'Sad', 'Victorious', 'Sad', 'Victorious', 'Fight',
       'Victorious', 'Happy', 'Victorious', 'Fight', 'Happy',
       'Victorious', 'Fight', 'Happy', 'Sad', 'Fight', 'Fight', 'Happy',
       'Fight', 'Sad', 'Victorious', 'Happy', 'Sad', 'Happy', 'Fight',
       'Happy', 'Victorious', 'Fight', 'Victorious', 'Fight',
       'Victorious', 'Fight', 'Victorious', 'Sad', 'Fight', 'Happy',
       'Sad', 'Sad', 'Victorious', 'Sad', 'Happy', 'Victorious', 'Sad',
       'Happy', 'Fight', 'Happy', 'Happy', 'Fight', 'Sad', 'Fight', 'Sad',
       'Victorious', 'Sad', 'Happy', 'Fight', 'Happy', 'Happy', 'Happy',
       'Happy', 'Victorious', 'Sad', 'Victorious', 'Fight', 'Victorious',
       'Victorious', 'Sad', 'Sad', 'Fight', 'Happy', 'Sad', 'Sad',
       'Victorious', 'Happy', 'Happy', 'Victorious', 'Victorious',
       'Fight', 'Fight', 'Sad', 'Sad', 'Sad', 'Victorious', 'Happy',
       'Fight', 'Happy', 'Sad', 'Sad', 'Happy', 'Victorious', 'Fight',
       'Fight', 'Happy', 'Victorious', 'Sad', 'Sad', 'Happy', 'Sad',
       'Sad', 'Fight', 'Sad', 'Happy', 'Happy', 'Sad', 'Fight',
       'Victorious', 'Sad', 'Happy', 'Sad', 'Sad', 'Happy', 'Fight',
       'Fight', 'Victorious', 'Fight', 'Happy', 'Fight', 'Happy',
       'Victorious', 'Fight', 'Victorious', 'Happy', 'Sad', 'Fight',
       'Sad', 'Victorious', 'Happy', 'Happy', 'Happy', 'Happy',
       'Victorious', 'Fight', 'Happy', 'Happy', 'Victorious', 'Sad',
       'Fight', 'Happy', 'Sad', 'Fight', 'Sad', 'Sad', 'Sad', 'Fight',
       'Happy', 'Fight', 'Happy', 'Happy', 'Happy', 'Sad', 'Fight',
       'Happy', 'Victorious', 'Victorious', 'Victorious', 'Happy',
       'Happy', 'Happy', 'Fight', 'Happy', 'Happy', 'Happy', 'Victorious',
       'Victorious', 'Happy', 'Sad', 'Happy', 'Sad', 'Fight',
       'Victorious', 'Fight', 'Happy', 'Happy', 'Fight', 'Fight', 'Happy',
       'Sad', 'Victorious', 'Victorious', 'Happy', 'Fight'], dtype='<U10')
```

iv) Evaluate and Serialize Model:

```
In [35]: from sklearn.metrics import accuracy_score # Accuracy metrics
         import pickle
```

Iterating through the fitted models in ‘fit_models’ and making predictions on the testing set (‘X_test’).

```
In [36]: for algo, model in fit_models.items():
             yhat = model.predict(X_test)
             print(algo, accuracy_score(y_test, yhat))
```

```
lr 1.0
rc 1.0
rf 0.9850746268656716
gb 0.9950248756218906
```

Using the Random Forest Classifier ('rf') from ‘fit_models’ dictionary to make predictions on the testing set (‘X_test’).

```
In [37]: fit_models['rf'].predict(X_test)
```

```
Out[37]: array(['Sad', 'Fight', 'Sad', 'Victorious', 'Happy', 'Happy', 'Happy',
                'Victorious', 'Fight', 'Fight', 'Happy', 'Fight', 'Happy',
                'Victorious', 'Fight', 'Fight', 'Sad', 'Victorious', 'Sad',
                'Fight', 'Sad', 'Fight', 'Sad', 'Happy', 'Sad', 'Victorious',
                'Fight', 'Victorious', 'Happy', 'Victorious', 'Fight', 'Happy',
                'Victorious', 'Fight', 'Happy', 'Sad', 'Fight', 'Fight', 'Happy',
                'Fight', 'Sad', 'Victorious', 'Happy', 'Sad', 'Happy', 'Fight',
                'Happy', 'Victorious', 'Fight', 'Victorious', 'Fight',
                'Victorious', 'Fight', 'Victorious', 'Sad', 'Fight', 'Happy',
                'Sad', 'Sad', 'Victorious', 'Sad', 'Happy', 'Victorious', 'Sad',
                'Happy', 'Fight', 'Happy', 'Happy', 'Fight', 'Sad', 'Fight', 'Sad',
                'Victorious', 'Sad', 'Happy', 'Fight', 'Happy', 'Happy', 'Happy',
                'Happy', 'Victorious', 'Sad', 'Victorious', 'Fight', 'Victorious',
                'Victorious', 'Sad', 'Sad', 'Fight', 'Happy', 'Sad', 'Sad',
                'Victorious', 'Happy', 'Happy', 'Victorious', 'Victorious',
                'Fight', 'Fight', 'Sad', 'Sad', 'Sad', 'Victorious', 'Happy',
                'Fight', 'Happy', 'Sad', 'Sad', 'Happy', 'Victorious', 'Fight',
                'Fight', 'Happy', 'Victorious', 'Sad', 'Sad', 'Happy', 'Sad',
                'Sad', 'Fight', 'Sad', 'Happy', 'Happy', 'Sad', 'Fight',
                'Victorious', 'Sad', 'Happy', 'Sad', 'Sad', 'Happy', 'Fight',
                'Fight', 'Victorious', 'Fight', 'Happy', 'Fight', 'Happy',
                'Victorious', 'Fight', 'Victorious', 'Happy', 'Sad', 'Fight',
                'Sad', 'Victorious', 'Happy', 'Happy', 'Happy', 'Happy',
                'Victorious', 'Fight', 'Happy', 'Happy', 'Victorious', 'Sad',
                'Fight', 'Happy', 'Sad', 'Fight', 'Sad', 'Sad', 'Sad', 'Fight',
                'Happy', 'Fight', 'Happy', 'Happy', 'Happy', 'Sad', 'Fight',
                'Happy', 'Victorious', 'Victorious', 'Victorious', 'Happy',
                'Happy', 'Happy', 'Fight', 'Happy', 'Happy', 'Happy', 'Victorious',
                'Victorious', 'Happy', 'Sad', 'Happy', 'Sad', 'Fight',
                'Victorious', 'Fight', 'Happy', 'Happy', 'Fight', 'Fight', 'Happy',
                'Sad', 'Victorious', 'Victorious', 'Happy', 'Fight'], dtype=object)
```

```
In [38]: y_test
```

```
Out[38]: 240          Sad
         542        Fight
         214          Sad
         492        Fight
         190        Happy
                   ...
         309          Sad
         387    Victorious
         370    Victorious
         173        Happy
         590        Fight
         Name: class, Length: 201, dtype: object
```

Using pickle to save the Random Forest model (fit_models['rf']) to a file named 'body_language.pkl'.

In [39]:
```
with open('body_language.pkl', 'wb') as f:
    pickle.dump(fit_models['rf'], f)
```

v) Make Detections with Model:

Successfully loaded the Random Forest model from the file 'body_language.pkl'.

In [40]:
```
with open('body_language.pkl', 'rb') as f:
    model = pickle.load(f)
```

In [41]:
```
model
```

Out[41]:
```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('randomforestclassifier', RandomForestClassifier())])
```

Integrated the body language classification using the previously trained Random Forest model into the real-time Holistic model processing loop

In [42]:
```
cap = cv2.VideoCapture(0)

# initialize the model
with mp_holistic.Holistic(min_detection_confidence=0.5,min_tracking_confidence=0.5) as holistic:

    while cap.isOpened():
        ret , frame = cap.read()

        image = cv2.cvtColor(frame , cv2.COLOR_BGR2RGB)

        image.flags.writeable = False

        results = holistic.process(image)

        # print(results.pose_landmarks)
        image.flags.writeable = True

        image = cv2.cvtColor(image , cv2.COLOR_RGB2BGR)

        # face
        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

        # right hand
        mp_drawing.draw_landmarks(image , results.right_hand_landmarks , mp_holistic.HAND_CONNECTIONS,
        mp_drawing.DrawingSpec(color=(80,22,10) ,thickness = 2 ,circle_radius=4 ) ,          mp_drawing.DrawingSpec(color=(80,44,

        # left hand
        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

        # pose detection
        mp_drawing.draw_landmarks(image , results.pose_landmarks , mp_holistic.POSE_CONNECTIONS,
        mp_drawing.DrawingSpec(color=(245,117,66) ,thickness = 2 ,circle_radius=4 ) ,        mp_drawing.DrawingSpec(color=(245,66
```

```
        # export coordinates
        try:
            pose = results.pose_landmarks.landmark
            pose_row = list(np.array([[landmark.x , landmark.y , landmark.z , landmark.visibility] for landmark in pose]).flatter

            face = results.face_landmarks.landmark
            face_row = list(np.array([[landmark.x , landmark.y , landmark.z , landmark.visibility] for landmark in face]).flatter

            row = pose_row + face_row
            X = pd.DataFrame([row])
            body_language_class = model.predict(X)[0]
            body_language_prob = model.predict_proba(X)[0]
            # print(body_language_class , body_language_prob)

            # grab ear coords
            coords = tuple(
                np.multiply(
                    np.array(
                        (results.pose_landmarks.landmark[mp_holistic.PoseLandmark.LEFT_EAR].x ,
                        [640,480]).astype(int)
                        )

            cv2.rectangle(image,
            (coords[0],coords[1]+5) ,
            (coords[0]+len(body_language_class)*20,coords[1]-30), (245 , 117 , 16),-1)

            cv2.putText(image , body_language_class , coords ,
             cv2.FONT_HERSHEY_SIMPLEX , 1 , (255,255,255),2 , cv2.LINE_AA)
```

```
        except:
            pass

        cv2.imshow("Holistic Model Detections",image)

        if cv2.waitKey(30) & 0xFF == ord('q'):
            break

cap.release()
cv2.destroyAllWindows()
```

Creating a tuple containing the coordinates of the left ear landmark in pixel coordinates.

```
In [43]: tuple(np.multiply(np.array((results.pose_landmarks.landmark[mp_holistic.PoseLandmark.LEFT_EAR].x,
         results.pose_landmarks.landmark[mp_holistic.PoseLandmark.LEFT_EAR].y)), [640,480]).astype(int))

Out[43]: (401, 238)
```

Integrate with Web Framework:

In this section, we will be building a web application that is integrated to the model we built. We will be using the streamlit package for our website development. Streamlit is a free and open-source framework to rapidly build and share beautiful machine learning and data science web apps.

Create a web.py file and import necessary packages:

web.py > image

```
1  import streamlit as st
2  import pandas as pd
3  import tensorflow as tf
4  import mediapipe as mp
5  import cv2
6  import numpy as np
7  import pickle
8  import csv
9  import os
```

Loading the model and creating a frame window to use OpenCV:

```
st.cache(allow_output_mutation = True)

with open('body_language.pkl', 'rb') as f:
    model = pickle.load(f)

st.write("""# Body Language Detection""")
mp_drawing = mp.solutions.drawing_utils # Drawing helpers
mp_holistic = mp.solutions.holistic #Mediapipe Solutions
run = st.checkbox('Run')
FRAME_WINDOW = st.image([])
```

Accepting the input from user and prediction:

web.py > image

```
cap = cv2.VideoCapture(0)
while run:
    with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:
        while cap.isOpened():
            ret, frame = cap.read()

        # Recolor Feed
        image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        image.flags.writeable = False
        # Make Detections
        results = holistic.process(image)
        # print(results.face_landmarks)

        # face_landmarks, pose_landmarks, left_hand_landmarks, right_hand_landmarks

        # Recolor image back to BGR for rendering
        image.flags.writeable = True
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

        # 1. Draw face landmarks
        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)
                                  )

        # 2. Right hand
        mp_drawing.draw_landmarks(image, results.right_hand_landmarks, mp_holistic.HAND_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)
                                  )

        # 3. Left Hand
        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)
                                  )
```

```
            # 4. Pose Detections
            mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_holistic.POSE_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)
                                      )
            # Export coordinates
            try:
                # Extract Pose landmarks
                pose = results.pose_landmarks.landmark
                pose_row = list(np.array([[landmark.x, landmark.y, landmark.z, landmark.visibility] for landmark in pose]).flatten())

                # Extract Face landmarks
                face = results.face_landmarks.landmark
                face_row = list(np.array([[landmark.x, landmark.y, landmark.z, landmark.visibility] for landmark in face]).flatten())

                # Concate rows
                row = pose_row+face_row

                #Make detections
                X = pd.DataFrame([row])
                body_language_class = model.predict(X)[0]
                body_language_prob = model.predict_proba(X)[0]
                print(body_language_class, body_language_prob)

                #Grab ear coords
                coords = tuple(np.multiply(
                                np.array(
                                    (results.pose_landmarks.landmark[mp_holistic.PoseLandmark.LEFT_EAR].x,
                                    results.pose_landmarks.landmark[mp_holistic.PoseLandmark.LEFT_EAR].y))
                            ,[640,480]).astype(int))

                cv2.rectangle(image,
                            (coords[0],coords[1]+5),
                            (coords[0]+len(body_language_class)*20,coords[1]-30),
                            (245 , 117 , 16),-1)
```

```
                cv2.putText(image , body_language_class , coords ,
                                    cv2.FONT_HERSHEY_SIMPLEX , 1 , (255,255,255), 2, cv2.LINE_AA)
                #Get status box
                cv2.rectangle(image, (0,0), (250,60), (245, 117, 16), -1)
                #Display Class
                cv2.putText(image, 'CLASS'
                                    , (95,12), cv2.FONT_HERSHEY_SIMPLEX , 0.5 , (0,0,0), 2 , cv2.LINE_AA)
                cv2.putText(image , body_language_class.split(' ')[0]
                                    , (90,40), cv2.FONT_HERSHEY_SIMPLEX , 1 , (255,255,255), 2 , cv2.LINE_AA)
                #Display Probability
                cv2.putText(image , 'PROB'
                                    , (15,12), cv2.FONT_HERSHEY_SIMPLEX , 0.5 , (0,0,0), 1, cv2.LINE_AA)
                cv2.putText(image ,str(round(body_language_prob[np.argmax(body_language_prob)],2))
                                    , (10,40), cv2.FONT_HERSHEY_SIMPLEX , 1 , (255,255,255), 2, cv2.LINE_AA)

            except:
                pass
            cv2.imshow("Holistic Model Detections",image)
            FRAME_WINDOW.image(frame, channels="BGR")
            if cv2.waitKey(30) & 0xFF == ord('q'):
                break
        cap.release()
        cv2.destroyAllWindows()
    else:
        st.write('Stopped')
```

- st.write() function will write the title for your website.
- FRAME_WINDOW.image(image) will run the OpenCV on the streamlit frame.

To run your website go to your terminal with your respective directory and run the command:

"streamlit run web.py"

```
PS D:\sanja\OneDrive\Documents\SmartInternz Project> streamlit run web.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://192.168.1.50:8501
```

On successful execution you'll get to see this in the terminal.

8. PERFORMANCE TESTING

8.1 Performance Metrics

Performance testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Performance testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include, but are not limited to, the process of executing a program or application with the intent of finding software bugs (errors or other defects). Performance testing can provide objective, independent information about the quality of software and risk of its failure to users and/or sponsors.

Table 8.1 Test Cases Representation

S. No	Description	Input	Precision	Recall	F1-score	Support
1	Predicting a Happy face as happy	Happy face Through Web cam	0.97	1.00	0.99	39
2	Predicting a Sad face as sad	Sad face through web cam	0.98	0.98	0.98	55
3	Predicting a Victorious Gesture.	Victorious gestures through Web cam	1.00	0.96	0.98	69
4	Predicting Fight gesture.	Fight gesture Through web cam	0.96	1.00	0.98	54

Model Summary:

In [66]:
```
# Model Summary
rf_model = fit_models['rf']

# Access the RandomForestClassifier from the pipeline
rf_classifier = rf_model.named_steps['randomforestclassifier']

# Get the number of trees (estimators) in the forest
n_estimators = rf_classifier.n_estimators
print(f"Number of Estimators (Trees): {n_estimators}")

# Get feature importances (if applicable)
feature_importances = rf_classifier.feature_importances_
print("Feature Importances:")
for feature, importance in zip(X.columns, feature_importances):
    print(f"{feature}: {importance}")
```

```
Number of Estimators (Trees): 100
Feature Importances:
0: 0.0
1: 4.3254354878427415e-05
2: 0.0025111311299143998
3: 0.0026400775012742467
4: 0.0
5: 0.0
6: 0.0
7: 0.001036813742109247
8: 0.00022671438726009802
9: 0.0002297972998864533
10: 0.0019030727023623165
11: 0.002945873278431087
12: 0.0
13: 5.319454118895791e-05
14: 0.00015355536107018465
15: 0.006268527477386391
```

Training Accuracy:

In [67]:
```
#training accuracy
rf_model = fit_models['rf']

# Predictions on the training data
y_train_pred = rf_model.predict(X_train)

# Calculate training accuracy
training_accuracy = accuracy_score(y_train, y_train_pred)
print(f"Training Accuracy: {training_accuracy}")
```

```
Training Accuracy: 1.0
```

Validation Accuracy:

]:
```
y_val_pred = model.predict(X_test)
val_accuracy = accuracy_score(y_test, y_val_pred)
print(f"Validation Accuracy: {val_accuracy}")
```

```
Validation Accuracy: 0.9815668202764977
```

Confusion Matrix, Accuracy score, Classification Report

[78]:
```
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
from sklearn.model_selection import train_test_split
import pandas as pd

# Assuming you have a DataFrame df with the columns 'class' and other features
# Replace df with your actual DataFrame

# Read the CSV file into a DataFrame
df = pd.read_csv('coords.csv')

# Extract features (X) and target labels (y)
X = df.drop('class', axis=1)
y = df['class']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1234)

# Assuming you have a fitted Random Forest model stored in fit_models['rf']
rf_model = fit_models['rf']

# Predictions on the test data
y_pred = rf_model.predict(X_test)

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)

# Accuracy Score
acc_score = accuracy_score(y_test, y_pred)
print(f"Accuracy Score: {acc_score}")

# Classification Report
class_report = classification_report(y_test, y_pred)
print("Classification Report:")
print(class_report)
```

```
Confusion Matrix:
[[54  0  0  0]
 [ 0 39  0  0]
 [ 0  1 54  0]
 [ 2  0  1 66]]
Accuracy Score: 0.9815668202764977
Classification Report:
              precision    recall  f1-score   support

       Fight       0.96      1.00      0.98        54
       Happy       0.97      1.00      0.99        39
         Sad       0.98      0.98      0.98        55
  Victorious       1.00      0.96      0.98        69

    accuracy                           0.98       217
   macro avg       0.98      0.98      0.98       217
weighted avg       0.98      0.98      0.98       217
```

9. RESULTS

9.1 Output Screenshots

Fig 9.1 “Happy expression”

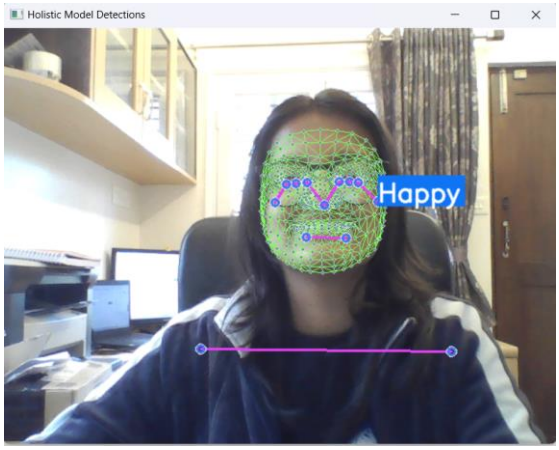


Fig 9.2 “Sad expression”

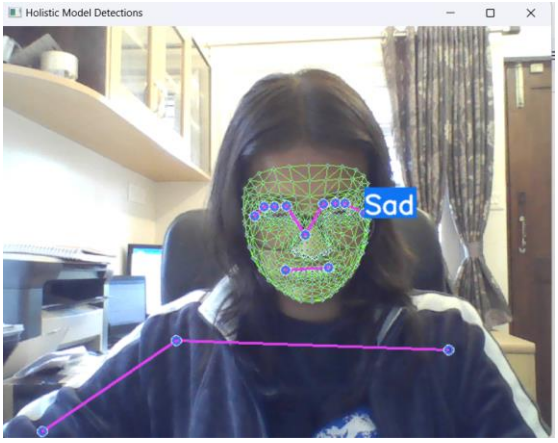


Fig 9.3 “Victorious gesture”

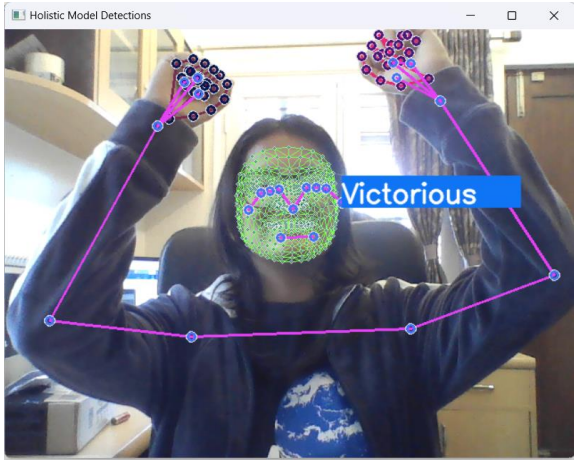


Fig 9.4 “Fight gesture”

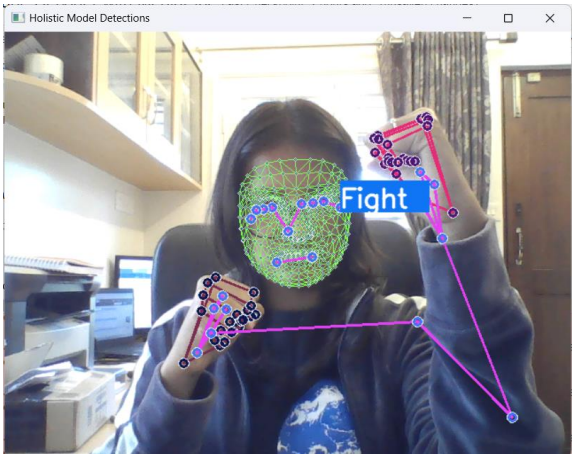
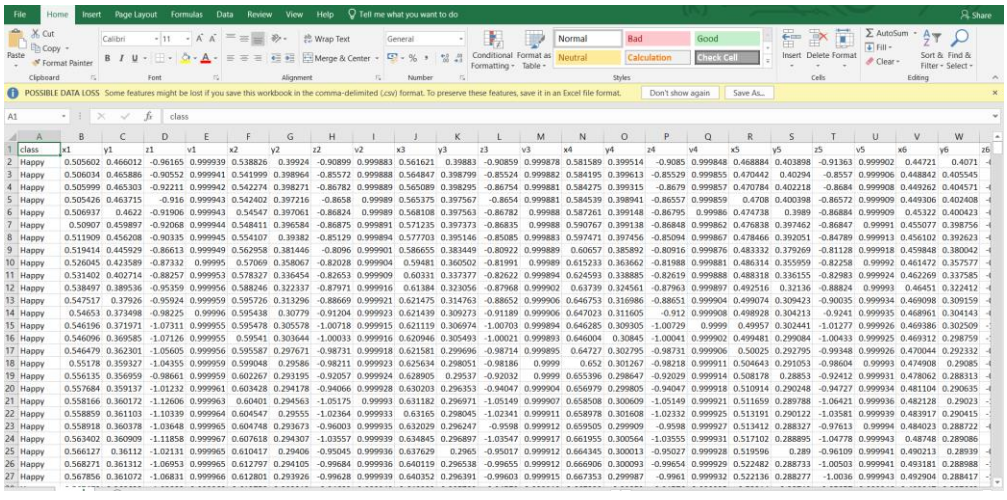


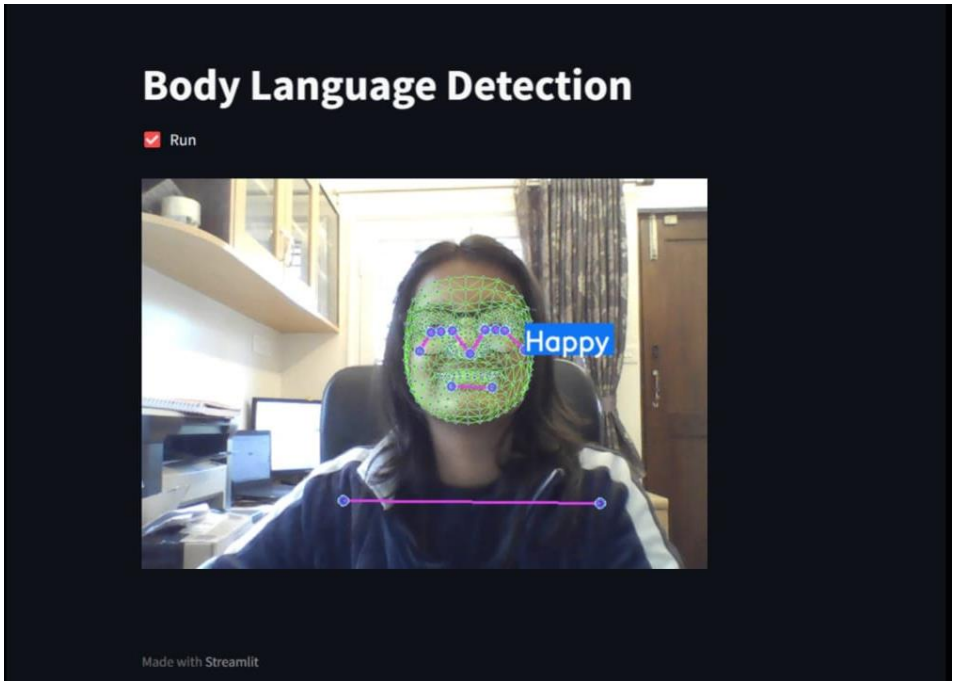
Fig 9.4 coords.csv(coordinates saved in excel sheet)



class	x1	y1	z1	v1	x2	y2	z2	v2	x3	y3	z3	v3	x4	y4	z4	v4	x5	y5	z5	v5	x6	y6	z6
Happy	0.505602	0.466012	-0.96165	0.999939	0.538826	0.39924	-0.90899	0.999883	0.561621	0.39883	-0.90859	0.999878	0.581589	0.399514	-0.9085	0.999848	0.468884	0.403898	-0.91363	0.999902	0.44721	0.4071	-0
Happy	0.506034	0.465886	-0.90552	0.999941	0.541999	0.398964	-0.85572	0.999888	0.564847	0.398799	-0.85524	0.999882	0.584195	0.399613	-0.85529	0.999855	0.470442	0.40294	-0.8557	0.999906	0.448842	0.405545	-0
Happy	0.505999	0.465303	-0.92211	0.999942	0.542274	0.398271	-0.86782	0.999889	0.565089	0.398295	-0.86754	0.999881	0.584275	0.399315	-0.8679	0.999857	0.470784	0.402218	-0.8684	0.999908	0.449262	0.404571	-0
Happy	0.505426	0.463715	-0.916	0.999943	0.542402	0.397216	-0.8658	0.99989	0.565375	0.397567	-0.8654	0.999881	0.584539	0.398941	-0.86557	0.999859	0.4708	0.400398	-0.86572	0.999909	0.449306	0.402408	-0
Happy	0.506937	0.4622	-0.91906	0.999943	0.54547	0.397061	-0.86824	0.99989	0.568108	0.397563	-0.86782	0.99988	0.587261	0.399148	-0.86795	0.99986	0.474738	0.3989	-0.86884	0.999909	0.45322	0.400423	-0
Happy	0.50907	0.459897	-0.92068	0.999944	0.548411	0.396584	-0.86875	0.999891	0.571235	0.397373	-0.86835	0.99988	0.590767	0.399138	-0.86848	0.999862	0.476838	0.397462	-0.86847	0.99991	0.455077	0.398756	-0
Happy	0.511909	0.456208	-0.90335	0.999945	0.554107	0.39382	-0.85129	0.999894	0.577703	0.395146	-0.85085	0.999883	0.597471	0.397456	-0.85094	0.999867	0.478466	0.392051	-0.84789	0.999913	0.456102	0.392623	-0
Happy	0.519414	0.445929	-0.86613	0.999949	0.562958	0.381446	-0.8096	0.999901	0.586655	0.383449	-0.80922	0.999889	0.60657	0.385892	-0.80916	0.999876	0.483332	0.379269	-0.81128	0.999918	0.459848	0.380042	-0
Happy	0.526045	0.423589	-0.87332	0.99995	0.57069	0.358067	-0.82028	0.999904	0.59481	0.360502	-0.81991	0.99989	0.615233	0.363662	-0.81988	0.999881	0.486314	0.355959	-0.82258	0.99992	0.461472	0.357577	-0
Happy	0.531402	0.402714	-0.88257	0.999953	0.578327	0.336454	-0.82653	0.999909	0.60331	0.337377	-0.82622	0.999894	0.624593	0.338885	-0.82619	0.999888	0.488318	0.336155	-0.82983	0.999924	0.462269	0.337585	-0
Happy	0.538497	0.389536	-0.95359	0.999956	0.588246	0.322337	-0.87971	0.999916	0.61384	0.323056	-0.87968	0.999902	0.63739	0.324561	-0.87963	0.999897	0.492516	0.32136	-0.88824	0.99993	0.46451	0.322412	-0
Happy	0.547517	0.37926	-0.95924	0.999959	0.595726	0.313296	-0.88669	0.999921	0.621475	0.314763	-0.88652	0.999906	0.646753	0.316986	-0.88651	0.999904	0.499074	0.309423	-0.90035	0.999934	0.469098	0.309159	-0
Happy	0.54653	0.373498	-0.98225	0.99996	0.595438	0.30779	-0.91204	0.999923	0.621439	0.309273	-0.91189	0.999906	0.647023	0.311605	-0.912	0.999908	0.498928	0.304213	-0.9241	0.999935	0.468961	0.304143	-0
Happy	0.546196	0.371971	-1.07311	0.999955	0.595478	0.305578	-1.00718	0.999915	0.621119	0.306974	-1.00703	0.999894	0.646285	0.309305	-1.00729	0.9999	0.49957	0.302441	-1.01277	0.999926	0.469386	0.302509	-0
Happy	0.546096	0.369585	-1.07126	0.999955	0.59541	0.303644	-1.00033	0.999916	0.620946	0.305493	-1.00021	0.999893	0.646004	0.30845	-1.00041	0.999902	0.499481	0.299084	-1.00433	0.999925	0.469312	0.298759	-0
Happy	0.546479	0.362301	-1.05605	0.999956	0.595587	0.297671	-0.98731	0.999918	0.621581	0.299696	-0.98714	0.999895	0.64727	0.302795	-0.98731	0.999906	0.50025	0.292795	-0.99348	0.999926	0.470044	0.292332	-0
Happy	0.55178	0.359327	-1.04355	0.999959	0.599048	0.29586	-0.98211	0.999923	0.625634	0.298051	-0.98186	0.9999	0.652	0.301267	-0.98218	0.999911	0.504643	0.291053	-0.98604	0.99993	0.474908	0.29085	-0
Happy	0.556135	0.356959	-0.98661	0.999959	0.602267	0.293195	-0.92057	0.999924	0.628905	0.29537	-0.92032	0.9999	0.655396	0.298647	-0.92029	0.999914	0.508178	0.28853	-0.92412	0.999931	0.478062	0.288313	-0
Happy	0.557684	0.359137	-1.01232	0.999961	0.603428	0.294178	-0.94066	0.999928	0.630203	0.296353	-0.94047	0.999904	0.656979	0.299805	-0.94047	0.999918	0.510914	0.290248	-0.94727	0.999934	0.481104	0.290635	-0
Happy	0.558166	0.360172	-1.12606	0.999963	0.60401	0.294563	-1.05175	0.99993	0.631182	0.296971	-1.05149	0.999907	0.658508	0.300609	-1.05149	0.999921	0.511659	0.289788	-1.06421	0.999936	0.482128	0.29023	-0
Happy	0.558859	0.361103	-1.10339	0.999964	0.604547	0.29555	-1.02364	0.999933	0.63165	0.298045	-1.02341	0.999911	0.658978	0.301608	-1.02332	0.999925	0.513191	0.290122	-1.03581	0.999939	0.483917	0.290415	-0
Happy	0.558918	0.360378	-1.03648	0.999965	0.604748	0.293673	-0.96003	0.999935	0.632029	0.296247	-0.9598	0.999912	0.659505	0.299909	-0.9598	0.999927	0.513412	0.288327	-0.97613	0.99994	0.484023	0.288722	-0
Happy	0.563402	0.360909	-1.11858	0.999967	0.607618	0.294307	-1.03557	0.999939	0.634845	0.296897	-1.03547	0.999917	0.661955	0.300564	-1.03555	0.999931	0.517102	0.288895	-1.04778	0.999943	0.48748	0.289086	-0
Happy	0.566127	0.36112	-1.02131	0.999965	0.610417	0.29406	-0.95045	0.999936	0.637629	0.2965	-0.95017	0.999912	0.664345	0.300013	-0.95027	0.999928	0.519596	0.289	-0.96109	0.999941	0.490213	0.28939	-0
Happy	0.568271	0.361312	-1.06953	0.999965	0.612797	0.294105	-0.99684	0.999936	0.640119	0.296538	-0.99655	0.999912	0.666906	0.300093	-0.99654	0.999929	0.522482	0.288733	-1.00503	0.999941	0.493181	0.288988	-0
Happy	0.567856	0.361072	-1.06831	0.999966	0.612801	0.293926	-0.99628	0.999939	0.640352	0.296391	-0.99603	0.999915	0.667353	0.299987	-0.9961	0.999932	0.522136	0.288277	-1.0036	0.999943	0.492904	0.288417	-0

How the Webpage Looks:

After clicking on Run:



10. ADVANTAGES & DISADVANTAGES

Advantages of AI Body Language Decoder with MediaPipe:

- Real-Time Processing: MediaPipe enables real-time processing, allowing for instantaneous interpretation and analysis of body language cues, which is crucial for applications requiring quick responses.

- Multimodal Integration: MediaPipe facilitates the integration of multiple models for face, hand, and body components, providing a cohesive solution for comprehensive body language decoding.
- Accuracy and Precision: The modular architecture of MediaPipe allows for the integration of specialized models, contributing to high accuracy and precision in recognizing and interpreting subtle nuances of body language.

- Flexibility and Extensibility: MediaPipe's modular design and clear APIs enhance flexibility and extensibility, enabling developers to easily integrate new features, components, or models into the system.

- Versatility Across Platforms: The AI Body Language Decoder with MediaPipe is suitable for deployment on various platforms, including mobile devices and desktop systems, providing versatility in its application.

- Integration with Existing Tools: MediaPipe can be seamlessly integrated with other machine learning frameworks and tools, enhancing its compatibility with existing development ecosystems.

Disadvantages of AI Body Language Decoder with MediaPipe:

- Complexity in Integration: Integrating and synchronizing multiple models for different body components may introduce complexity, requiring careful coordination and communication between these modules.

- Resource Intensiveness: Real-time processing of multiple body components may demand significant computational resources, potentially leading to increased power consumption and hardware requirements.

- Environmental Sensitivity: The system's accuracy may be affected by variations in environmental conditions, such as changes in lighting, background noise, or user appearance.

- Privacy Concerns: The collection and analysis of personal body language data raise privacy concerns. Proper measures must be implemented to address these concerns and ensure user consent and data protection.

- Learning Curve for Developers: Developers unfamiliar with MediaPipe may face a learning curve when implementing and customizing the system, potentially slowing down the development process.

- Dependency on Third-Party Libraries: The system's functionality relies on external libraries such as TensorFlow and OpenCV. Changes or updates to these libraries may impact the system's stability and performance.

11. CONCLUSION & FUTURE SCOPE

Detecting and analyzing body language is gaining a lot of attention lately. Being able to detect and analyze facial expressions of client/customer helps businesses and marketing teams to get honest reviews and feedbacks. But facial expression is just a small part of body language. Body language consists of others elements like hand gestures and body poses. And body language plays a very important role in communication. For example, in interviews, interviewers take candidate's body language into consideration. By enhancing this project, a tool can be provided to the interviewers which aids them in understanding how the candidate is responding when asked questions from different domains or put in different situations during HR rounds. Since this project supports real time hand landmark detection, hand sign language detection can also be implemented. Not only that, using

this project, implementation of already existing projects like drowsiness detection of drivers, action detection etc can be made easier with much better results.

12. APPENDIX

Source Code:

Machine Learning Concepts:

- Machine Learning: https://towardsdatascience.com/machine-learning-an-introduction-23b84d51e6d0
- ML Models: https://towardsdatascience.com/all-machine-learning-models-explained-in-6-minutes- 9fe30ff6776a

Web concepts:

- Get the gist on streamlit: https://www.geeksforgeeks.org/a-beginners-guide-to-streamlit/

Mediapipe:

- About Mediapipe: https://www.geeksforgeeks.org/python-facial-and-hand-recognition-using-mediapipe-holistic/

13. References

[1] https://heartbeat.fritz.ai/simultaneously-detecting-face-hand-motion-and-pose-in-real-time-on-mobile-devices-27849560fc4e

[2] https://medium.com/jstack-eu/using-machine-learning-to-analyse-body-language-and-facial-expressions-a779172cc98

[3] http://ai.googleblog.com/2019/08/on-device-real-time-hand-tracking-with.html

[4] https://www.altexsoft.com/blog/business/functional-and-non-functional-requirements-specification-and-types/

[5] https://www.guru99.com/software-testing-introduction-importance.html

[6] https://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf?source=post_page----------

[7] https://link.springer.com/chapter/10.1007/978-3-642-41190-8_50

[8] https://medium.com/jstack-eu/using-machine-learning-to-analyse-body-language-and-facial-expressions-a779172cc98

[9] https://arxiv.org/pdf/1906.08172.pdf

GitHub & Project Demo Link

GitHub - https://github.com/smartinternz02/SI-GuidedProject-604030-1699270489

Project Demo Link - https://drive.google.com/drive/folders/1k7xh0RteWv7jD2CdlgejkMoEHPjyqodq?usp=drive_link

TEAM MEMBERS:

KRITI CHINTA

JAITRA VENKITEELA

SAI EESHWAR D

SANJANA BHAT