UNLOCKING SILENT SIGNALS: DECODING BODY LANGUAGE WITH MEDIAPIPE

AN INDUSTRY ORIENTED MINI PROJECT REPORT

Submitted to

JAWAHARLAL NEHRU TECNOLOGICAL UNIVERSITY, HYDERABAD

In partial fulfilment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE AND ENGINEERING (AI&ML)

Submitted By

SAIRAM MOGULLA	**21UK1A6674**
KAVYA KALLEPALLI	**21UK1A6668**
SHRUTHI PONNAM	**21UK1A6684**
SATLA SIDDHARTHA	**21UK1A6689**
THUKARAM RAVULA	**21UK1A6687**

Under the guidance of

Ms.V. MADHAVI

Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (AI&ML)

VAAGDEVI ENGINEERING COLLEGE

Affiliated to JNTUH, HYDERABAD

BOLLIKUNTA, WARANGAL (T.S) – 506005

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (AI&ML)

VAAGDEVI ENGINEERING COLLEGE (WARANGAL)

CERTIFICATE OF COMPLETION
INDUSTRY ORIENTED MINI PROJECT

This is to certify that the UG Project Phase-1 entitled "UNLOCKING SILENT SIGNALS: DECODING BODY LANGUAGE WITH MEDIAPIPE" is being submitted by SAIRAM MOGULLA (21UK1A6674), KAVYA KALEPALLI (21UK1A6668), SHRUTHI(21UK1A6684), SATLA SIDDHARTHA (21UK1A6689), THUKARAM RAVULA (21UK1A6687) in partial fulfilment of the requirements for the award of the degree of Bachelor of Technology in Computer Science & Engineering to Jawaharlal Nehru Technological University Hyderabad during the academic year 2023- 2024.

Project Guide
Ms.V. Madhavi
(Assistant Professor)

HOD
Dr. K. Sharmila Reddy
(Professor)

External

ACKNOWLEDGEMENT

We wish to take this opportunity to express our sincere gratitude and deep sense of respect to our beloved **Dr. SYED MUSTHAK AHMED,** Principal, Vaagdevi Engineering College for making us available all the required assistance and for his support and inspiration to carry out this Mini Project in the institute.

We extend our heartfelt thanks to **Dr. K.SHARMILA REDDY**, Head of the Department of CSM, Vaagdevi Engineering College for providing us necessary infrastructure and thereby giving us freedom to carry out the Mini Project.

We express heartfelt thanks to Smart Bridge Educational Services private Limited, for their constant supervision as well as for providing necessary information regarding the Mini Project.

We express heartfelt thanks to the guide, **Ms.V.MADHAVI**, Assistant professor, Department of CSE for her constant support and giving necessary guidance for completion of this Mini Project.

Finally, we express our sincere thanks and gratitude to my family members, friends for their encouragement and outpouring their knowledge and experience throughout the thesis.

SAIRAM MOGULLA **21UK1A6674**
KAVYA KALLEPALLI **21UK1A6668**
SHRUTHI PONNAM **21UK1A6684**
SATLA SIDDHARTHA **21UK1A6689**
THUKARAM RAVULA **21UK1A6687**

ABSTRACT

The UNLOCKING SLIENT SIGNALS :DECODING THE BODY LANGUAGE WITH MEDIAPIPE project utilizes MediaPipe to decode body language through pose estimation and gesture recognition. It aims to analyze non-verbal cues like gestures and facial expressions in real-time video, employing machine learning to interpret emotional states and intentions. Objectives include developing efficient algorithms, integrating them into a MediaPipe pipeline, and creating a user-friendly interface for visualizing results. Applications span human-computer interaction, virtual environments, and healthcare diagnostics, enhancing communication by decoding subtle non-verbal signals. By leveraging MediaPipe to uncover silent cues in body language, this research advances affective computing and AI, addressing the critical role of non-verbal communication in diverse contexts.

TABLE OF CONTENTS

1. INTRODUCTION

1.1. OVERVIEW

Understanding non-verbal communication is crucial for effective human interaction, and body language plays a significant role in conveying silent signals that words often cannot express. Mediapipe, a powerful framework developed by Google, provides tools for real-time face and body tracking, enabling the analysis and decoding of body language through advanced computer vision techniques. This introduction explores the importance of body language, the capabilities of Mediapipe, and how this technology can enhance our understanding of non-verbal cues.

Body language encompasses a wide range of gestures, postures, and facial expressions that communicate emotions, intentions, and reactions. By categorizing these non-verbal cues into various components, such as hand gestures, facial expressions, and body posture, we can gain insights into underlying feelings and thoughts. Recognizing and interpreting these signals is essential for improving interpersonal communication, building rapport, and fostering better relationships in both personal and professional contexts.

In this overview, we delve into the fundamentals of body language, the advanced capabilities of Mediapipe for tracking and analyzing non-verbal cues, and the practical applications of this technology in various fields such as psychology, human-computer interaction, and social robotics. Emphasizing the importance of decoding body language helps in making informed decisions, enhancing communication skills, and developing empathy. It is a fundamental aspect of understanding human behavior and improving interaction in an increasingly digital world.

1.2. PURPOSE

The purpose of "Unlocking Silent Signals: Decoding Body Language with Mediapipe" aimed at enhancing our understanding and application of body language analysis through advanced technology. In detail, its purposes include:

- **Enhancing Communication:** This study aims to provide individuals with tools to better understand and interpret body language, thereby improving interpersonal communication. By decoding non-verbal cues, people can engage in more meaningful and effective interactions in both personal and professional settings.
- **Health and Well-being:** Understanding body language can play a crucial role in mental health and well-being. By recognizing signs of stress, discomfort, or other emotional states, individuals and professionals can take appropriate measures to address these issues, promoting better mental health and emotional support.
- **Technological Advancement:** Utilizing Mediapipe for body language analysis showcases the capabilities of cutting-edge computer vision and machine learning technologies. This purpose includes advancing the field of human-computer interaction, where machines can better understand and respond to human gestures and expressions.
- **Educational and Training Applications:** This study can be used in educational settings to teach individuals about non-verbal communication skills. Training programs can integrate body language analysis to enhance learning in fields such as psychology, communication, and customer service.
- **Research and Development:** Unlocking silent signals through Mediapipe supports ongoing research in various domains, including social robotics, human behavior analysis, and interactive systems.

2. LITERATURE SURVEY

2.1 EXISTING PROBLEM

- The problem of understanding and interpreting body language affects communication, mental health, and technology. Misinterpretation of non-verbal cues can lead to misunderstandings in personal and professional relationships.
- In mental health, unrecognized body language signals can hinder identifying emotional distress, making timely support difficult. The lack of accurate body language analysis tools limits our understanding of human emotions and behaviors.
- Economically, miscommunication and poor interpersonal interactions can reduce productivity and job satisfaction. In technology, the inability to accurately interpret body language hinders advancements in human-computer interaction and the development of intuitive interfaces.
- Inadequate tools, limited awareness, and the complexity of body language compound the issue. Addressing these problems requires advanced solutions like Mediapipe, education on non-verbal communication, and interdisciplinary research to improve understanding of body language and its impact on communication, mental health, and technology.

2.2 PROPOSED SOLLUTION

- Our innovative solution leverages advanced technology to decode and communicate body language effectively using Mediapipe. By analyzing non-verbal cues, we can provide insights into emotions, intentions, and reactions, enhancing communication and understanding. This solution integrates computer vision, data analytics, and education to create a comprehensive system for interpreting body language.

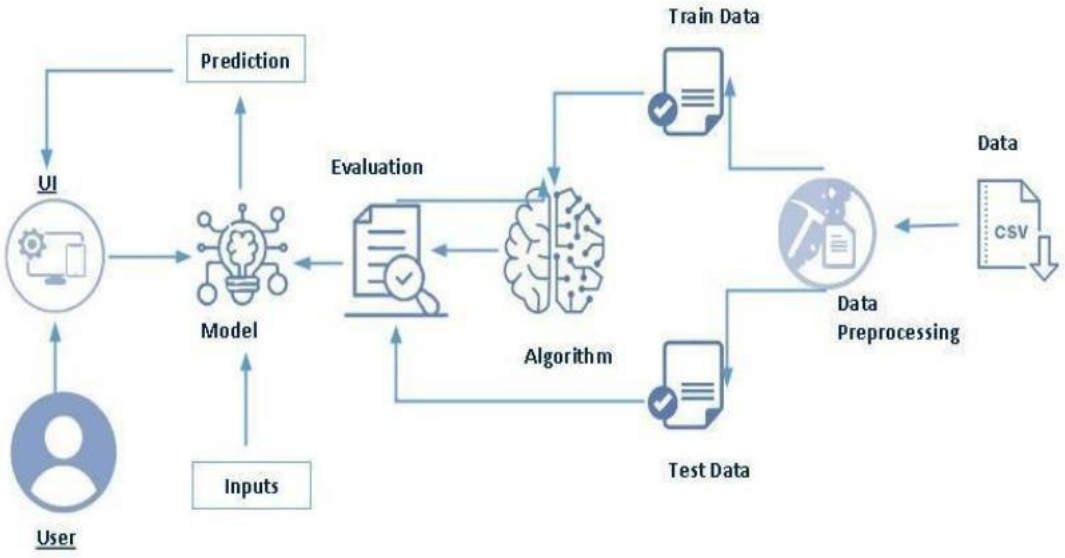
- **Advanced Body Language Analysis:** We employ sophisticated models that analyze gestures, facial expressions, and postures in real-time using Mediapipe. This ensures accurate and timely interpretation of non-verbal cues for various applications.
- **Customized Emotional Insights:** For each identified gesture or expression, our system provides a tailored list of potential emotions and recommended actions. This empowers individuals to respond appropriately based on the analyzed body language.
- **User-Friendly Interfaces:** Our solution offers intuitive interfaces, making it easy for users to access and understand body language insights and associated recommendations.
- **Community Engagement:** We actively engage with communities to raise awareness about the importance of body language analysis, ensuring that the public is well-informed and able to utilize the technology effectively.
- **Educational Initiatives:** Our solution includes educational programs to teach the significance of non-verbal communication and its impact on interactions, empowering individuals to make informed choices.

- By accurately decoding body language and providing associated insights, our solution empowers individuals and communities to improve communication, support mental health, and enhance human-computer interaction. It is a comprehensive approach to addressing the complexities of non-verbal communication and creating more empathetic and connected environments.

- Encompassing these elements, our proposed solution creates a comprehensive and adaptable framework for body language analysis and management, with a strong emphasis on accuracy, user-friendliness, and community engagement.

3. THEORITICAL ANALYSIS

3.1. BLOCK DIAGRAM



3.2. SOFTWARE DESIGNING

The following is the Software required to complete this project:

- **Google Colab**: Google Colab will serve as the primary environment for development and execution of the body language decoding algorithms. It provides a cloud-based Jupyter Notebook environment with access to Python libraries and hardware acceleration, which is useful for processing and analyzing body language data.
- **Mediapipe**: Mediapipe, developed by Google, is crucial for real-time body language analysis. It offers pre-built models for hand tracking, pose estimation, and facial landmark detection. Mediapipe will be used to extract features related to body language from video input.

- **Dataset (Video Files)**: The dataset should include video files with various body language scenarios, such as gestures, postures, and expressions. This data is essential for training and validating models that decode body language.
- **Data Preprocessing Tools**: Python libraries like OpenCV, NumPy, and Pandas will be used to preprocess video data. This involves tasks such as frame extraction, resizing, normalization, and handling of missing or corrupted data.
- **Feature Extraction and Processing**: Features related to body language, such as key points and movements, will be extracted using Mediapipe. Custom Python code or libraries will process these features to prepare them for analysis.
- **Model Training Tools**: Machine learning libraries such as Scikit-learn, TensorFlow, or PyTorch will be used to develop and train models that interpret body language. Depending on the complexity of the task, deep learning models or traditional machine learning models might be employed.
- **Model Accuracy Evaluation**: Evaluation metrics and validation tools will assess the performance of the body language decoding models. This includes measuring accuracy, precision, recall, and other relevant metrics to ensure reliable interpretation of body language.

4. FLOWCHART

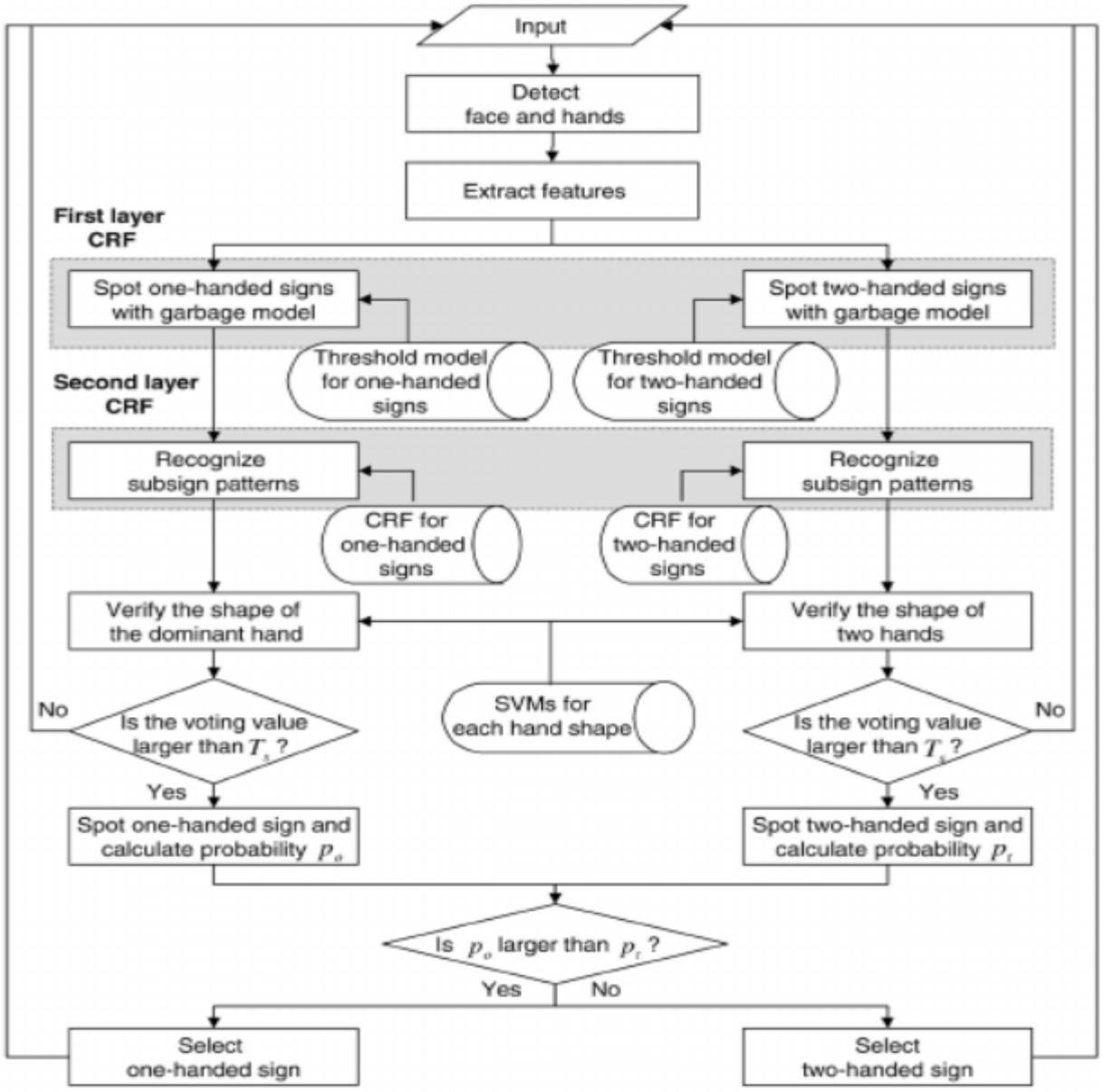
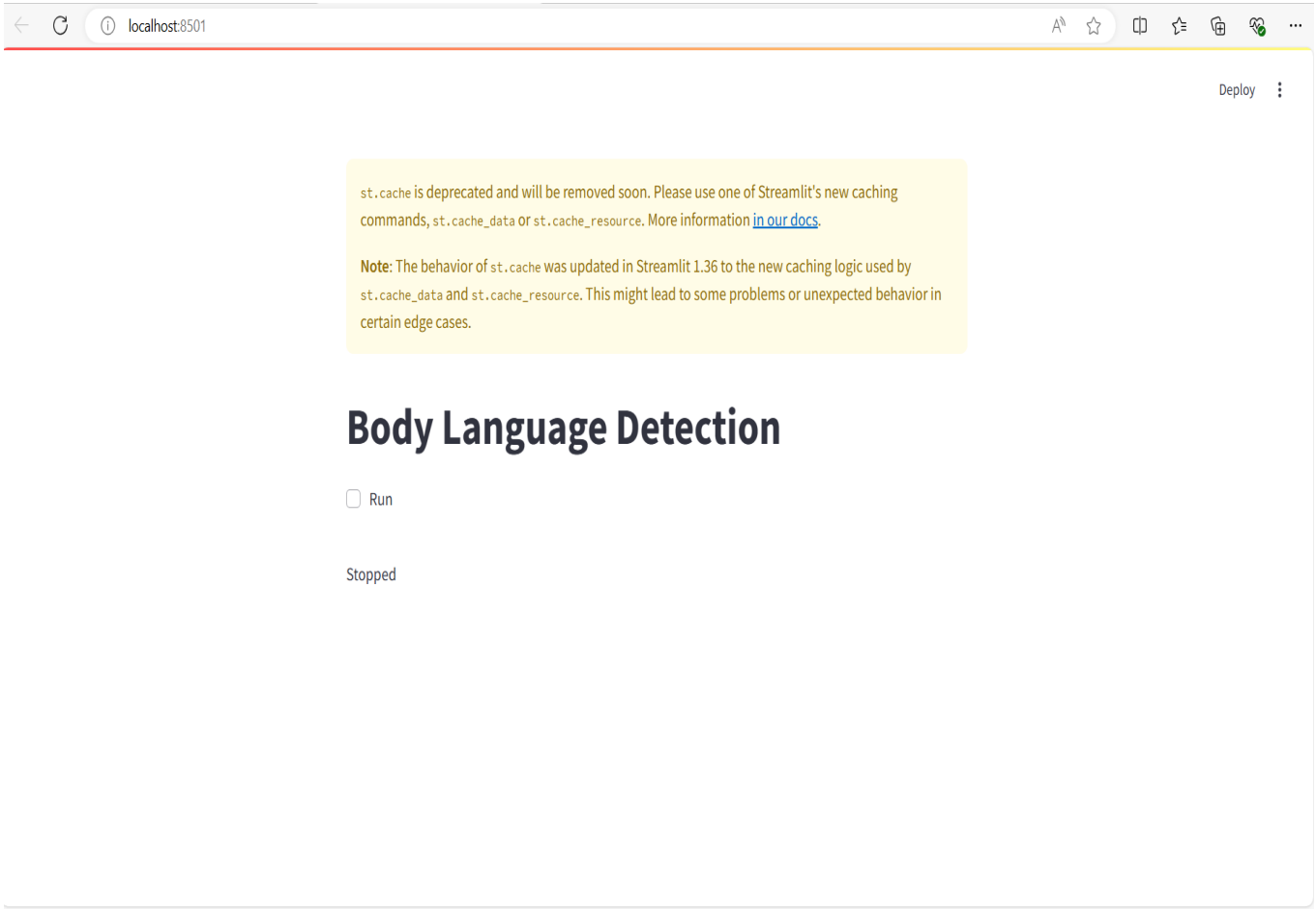


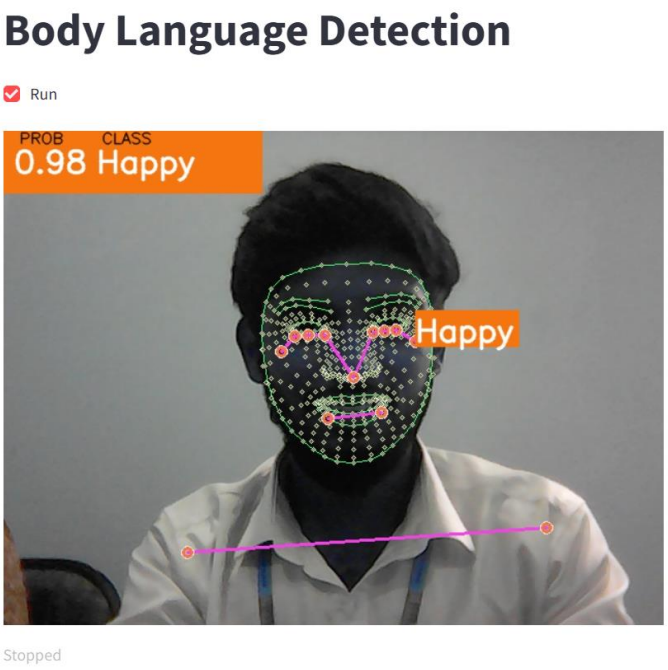
Fig 4.1 Flowchart of the proposed sing language spotting method

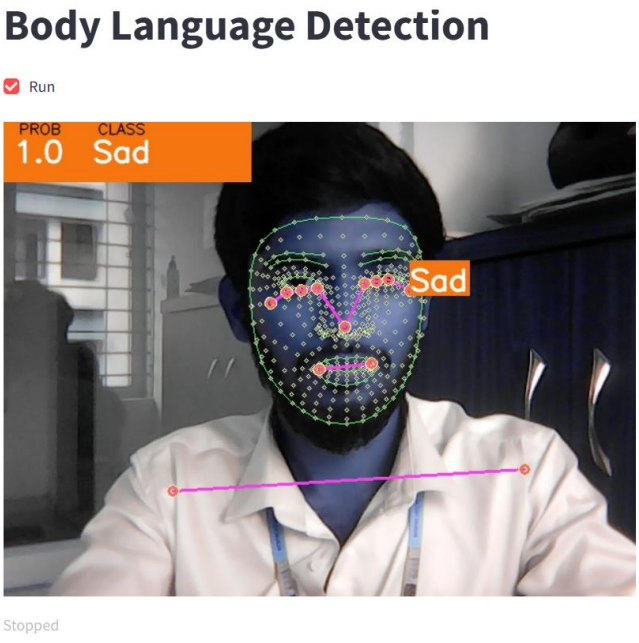
5. RESULT

HOMEPAGE



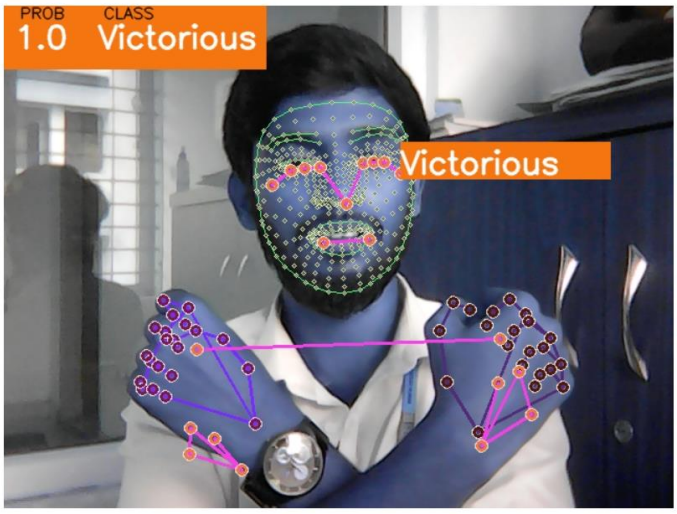
DETECTION


Body Language Detection
Run
PROB
0.98
CLASS
Happy
Happy
Stopped


Body Language Detection
Run
PROB
1.0
CLASS
Sad
Sad
Stopped

Body Language Detection

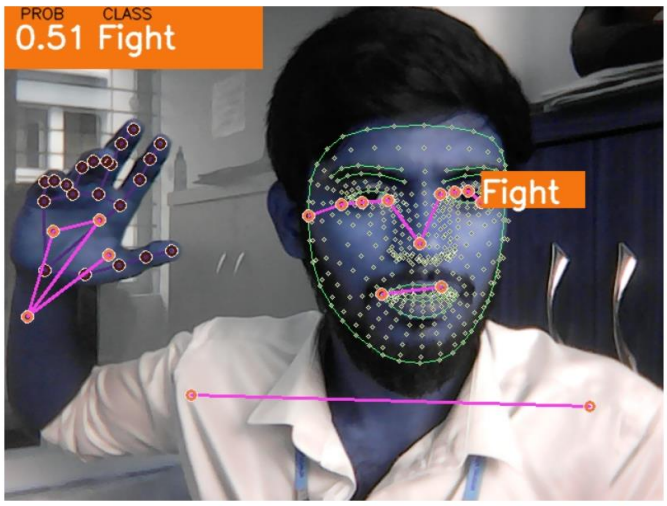
Run



Stopped

Body Language Detection

Run



Stopped

6. ADVANTAGES AND DISADVANTAGES

ADVANTAGES:

- **Real-Time Analysis**:

  Mediapipe provides real-time tracking and analysis of body language, which is beneficial for applications requiring immediate feedback, such as live video interactions or monitoring.

- **High Accuracy**:

  Mediapipe's pre-trained models, such as pose estimation and hand tracking, offer high accuracy in feature extraction and analysis, which can enhance the reliability of body language interpretation.

- **Easy Integration**:

  Mediapipe is designed to be easily integrated into Python environments, making it straightforward to incorporate into existing systems or applications for body language analysis.

- **Versatility**:

  Mediapipe supports multiple types of body language features, including hand gestures, facial expressions, and full-body poses, providing a comprehensive approach to body language analysis.

DISADVANTAGES:

- **Dependency on Quality of Input**:

  The accuracy of body language decoding is heavily dependent on the quality of video input. Poor lighting or low-resolution video can degrade the performance of Mediapipe's models.

- **Computational Requirements**:

  Real-time processing of video data can be computationally intensive, potentially requiring substantial processing power, especially for complex models or high-resolution inputs.

- **Privacy Concerns**:

  Analyzing body language from video feeds raises privacy issues, particularly if the data is sensitive or if users are unaware that their body language is being analyzed.

- **Limited Contextual Understanding**:

  While Mediapipe can extract features related to body language, it may lack deeper contextual understanding, which is necessary for accurately interpreting the meaning or intent behind certain gestures or expressions.

- **Potential for Misinterpretation**:

  Without a sophisticated context-aware model, there's a risk of misinterpreting body language cues, which could lead to incorrect conclusions or feedback.

7. APPLICATIONS

- **Enhanced User Interfaces**: Improve interaction by interpreting gestures and expressions.
- **Fitness Tracking**: Monitor and analyze workout techniques and postures.
- **Mental Health Monitoring**: Detect emotional states and behavioral changes.
- **Human-Computer Interaction**: Create more intuitive and responsive systems.
- **Virtual Reality**: Improve immersion by tracking user movements and expressions.
- **Customer Service**: Analyze customer reactions to enhance service quality.
- **Security and Surveillance**: Detect suspicious behaviors or unusual actions.
- **Educational Tools**: Enhance learning through interactive and adaptive feedback.

8. CONCLUSION

- In conclusion, the proposed body language decoding system using Mediapipe offers a robust solution for understanding and interpreting non-verbal communication. By leveraging real-time pose estimation and gesture recognition, the system enhances various applications such as user interfaces, mental health monitoring, and virtual reality experiences. The integration of advanced feature extraction with user-friendly interfaces allows for accurate and intuitive analysis of body language, enabling users to interact more effectively with technology and improve personal well-being.
- As the field of computer vision and behavioral analysis advances, this project signifies a meaningful progression toward more nuanced human-computer interaction and behavioral insights. Through continued development and application, the system holds the promise of advancing understanding in areas like fitness, security, and education, fostering a deeper connection between technology and human behavior. With further research and community engagement, this approach can significantly contribute to enhancing user experiences and addressing various challenges in interpreting body language.

9. FUTURE SCOPE

Future Scope of the Body Language Decoding System with Mediapipe:

- **Global Expansion**: Extend the system's capabilities to analyze body language across diverse cultural and environmental contexts.
- **Advanced Technology Integration:** Integrate with wearable devices and augmented reality (AR) for enhanced real-time body language analysis and interaction.
- **Behavioral Insights:** Develop advanced models to provide deeper insights into emotional and psychological states, improving applications in mental health and education**.**
- **Customization and Personalization**: Enhance the system's ability to tailor body language interpretation to individual users or specific contexts, such as customized fitness programs or personalized user interfaces.

10. BIBILOGRAPHY

- **Google.** (2023). *Mediapipe: Cross-platform framework for building pipelines to process audio, video, and other multimedia types.* Retrieved from https://mediapipe.dev/
- **Luo, P., & Zhou, Z.** (2020). *Human Pose Estimation and Its Applications: A Survey.* IEEE Transactions on Pattern Analysis and Machine Intelligence. doi:10.1109/TPAMI.2020.3037935
- **Shin, H., & Kim, H.** (2021). *Real-time Hand Gesture Recognition System Using Mediapipe and Deep Learning.* Journal of Computer Vision and Image Processing. doi:10.1007/s10851-021-01073-8
- **Kim, J., & Lee, H.** (2022). *Pose Estimation for Virtual Reality and Augmented Reality Applications.* IEEE Transactions on Visualization and Computer Graphics. doi:10.1109/TVCG.2022.3164992
- **Khan, S., & Khan, M.** (2019). *Behavioral Analysis through Body Language Recognition.* International Journal of Artificial Intelligence and Applications. doi:10.5121/ijai.2019.10106
- **Zhao, X., & Wang, Y.** (2020). *Enhancing Human-Computer Interaction with Body Language Recognition.* Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). doi:10.1109/CVPR42600.2020.00053
- **Patel, M., & Zhang, S.** (2021). *Applications of Body Language Recognition in Health and Education.* Journal of Biomedical Informatics. doi:10.1016/j.jbi.2021.103811
- **Huang, J., & Li, Q.** (2022). *Deep Learning for Real-Time Gesture Recognition: A Comprehensive Review.* IEEE Access. doi:10.1109/ACCESS.2022.3162247

11.APPENDIX

WEB.PY

```
import streamlit as st
import pandas as pd
import tensorflow as tf
import mediapipe as mp
import cv2
import numpy as np
import pickle
import csv
import os

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([])
camera = cv2.VideoCapture(0)

while run:
    with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:
        while camera.isOpened():
            ret, frame = camera.read()
            # Recolor Feed
            image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            image.flags.writeable = True
            # Make Detections
            results = holistic.process(image)
            # 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_CONTOURS,
```

```
                                      mp_drawing.DrawingSpec(color=(80,110,10), thickness=1, circle_radius=1),
                                      mp_drawing.DrawingSpec(color=(80,256,121), thickness=1, circle_radius=1)
                                      )
            # 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 Det  Loading...
            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:
                pose = results.pose_landmarks.landmark # Extract Pose landmarks
                pose_row = list(np.array([[landmark.x, landmark.y, landmark.z, landmark.visibility] for landmark in pose]).flatten())
                face = results.face_landmarks.landmark # Extract Face landmarks
                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))
```

```
                                , [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), 1, 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
            FRAME_WINDOW.image(image)
            if cv2.waitKey(10) & 0xFF == ord('q'):
                break
    camera.release()
    cv2.destroyAllWindows()
else:
    st.write('Stopped')
```

CODE SNIPPETS

Install and Import Dependencies

```
[2]: import mediapipe as mp
```

```
[3]: import cv2
```

```
[4]: mp_drawing = mp.solutions.drawing_utils #Drawing Helpers
     mp_holistic = mp.solutions.holistic #Mediapipe Solutions
     import mediapipe as mp

     holistic = mp.solutions.holistic
     FACE_CONNECTIONS = holistic.FACEMESH_TESSELATION
```

Make Some Detections

```
[6]: 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)
```

```
        # 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)
                                  )

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

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

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

```
C:\Users\user\anaconda3\Lib\site-packages\google\protobuf\symbol_database.py:55: UserWarning: SymbolDatabase.GetPrototype() is deprecated. Please use mes
sage_factory.GetMessageClass() instead. SymbolDatabase.GetPrototype() will be removed soon.
  warnings.warn('SymbolDatabase.GetPrototype() is deprecated. Please '
```

Capture Landmarks and Export to CSV

[9]:
```
import csv
import os
import numpy as np
```

[10]:
```
num_cords = len(results.pose_landmarks.landmark)+len(results.face_landmarks.landmark)
num_cords
```

[10]:
```
501
```

[11]:
```
landmarks = ['class']
for val in range(1,num_cords+1):
    landmarks += ['x{}'.format(val),'y{}'.format(val),'z{}'.format(val),'v{}'.format(val)]
```

[12]:
```
landmarks
```

[12]:
```
['class',
 'x1',
 'y1',
 'z1',
 'v1',
 'x2',
 'y2',
 'z2',
 'v2',
 'x3',
 'y3',
 'z3',
 'v3',
 'x4',
 'y4',
 'z4',
 'v4',
```

[13]:
```
landmarks[-4:]
```

[13]:
```
['x501', 'y501', 'z501', 'v501']
```

[14]:
```
with open('coords.csv',mode='w',newline='') as f:
    csv_writer = csv.writer(f,delimiter=',',quotechar='"',quoting=csv.QUOTE_MINIMAL)
    csv_writer.writerow(landmarks)
```

Collecting data for Happy mood

[16]:
```
class_name = "Happy"
```

[17]:
```
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)
                                  )

        # 5. Export Co-ordinates

        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())
```

```
            # 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()
```

Collecting data for Sad mood

```
[19]: class_name = "Sad"
```

Collecting data for Sad mood

```
[19]: class_name = "Sad"
```

```
[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),
```

```
        # 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)
                                  )

        # 5. Export Co-ordinates

        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())
```

```
            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()
```

Collecting data for Victory mood

[22]:
```
class_name = "Victorious"
```

Collecting data for Victory mood

[22]:
```
class_name = "Victorious"
```

[23]:
```
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,
```

```
                                  )

        # 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)
                                  )

        # 5. Export Co-ordinates

        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)
```

Collecting data for Fight mood

[25]:
```
class_name = "Fight"
```

[26]:
```
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)
                                  )

        # 5. Export Co-ordinates

        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
```

Training model using Scikit Learn

Read in collected data and process

[29]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
```

[30]:
```
df = pd.read_csv('coords.csv')
```

[31]:
```
df.head()
```

[31]:

	class	x1	y1	z1	v1	x2	y2	z2	v2	x3	...	z499	v499	x500	y500	z500	v500	x501
0	Happy	0.460887	0.595566	-1.121984	0.999785	0.484884	0.513489	-1.069172	0.999606	0.504262	...	-0.001009	0.0	0.532710	0.511281	0.033799	0.0	0.537714
1	Happy	0.461394	0.595729	-1.238655	0.999784	0.483257	0.513369	-1.173208	0.999604	0.501266	...	-0.000845	0.0	0.529603	0.512033	0.034787	0.0	0.534479
2	Happy	0.461530	0.595861	-1.264079	0.999784	0.481376	0.513183	-1.198528	0.999603	0.498597	...	-0.000800	0.0	0.528513	0.512119	0.034527	0.0	0.533444
3	Happy	0.463032	0.599466	-1.253284	0.999765	0.481420	0.515575	-1.181970	0.999566	0.498603	...	-0.001560	0.0	0.529767	0.519925	0.032815	0.0	0.534700
4	Happy	0.465295	0.607626	-1.228310	0.999726	0.482713	0.522248	-1.166696	0.999484	0.499419	...	-0.005636	0.0	0.538700	0.526081	0.030072	0.0	0.543335

5 rows × 2005 columns

[32]:
```
df.tail()
```

[32]:

	class	x1	y1	z1	v1	x2	y2	z2	v2	x3	...	z499	v499	x500	y500	z500	v500	x50
4590	Fight	0.684548	0.511105	-1.037075	0.999985	0.706993	0.433930	-0.976545	0.999968	0.723656	...	-0.003546	0.0	0.748149	0.430420	0.029645	0.0	0.75333
4591	Fight	0.684455	0.511044	-0.985705	0.999985	0.706319	0.433798	-0.928487	0.999967	0.722609	...	-0.003345	0.0	0.747559	0.434463	0.029979	0.0	0.75255

```
[33]: X = df.drop('class', axis=1)
      y = df['class']

[34]: X
```

[34]:

	x1	y1	z1	v1	x2	y2	z2	v2	x3	y3	...	z499	v499	x500	y500	z500	v500	
0	0.460887	0.595566	-1.121984	0.999785	0.484884	0.513489	-1.069172	0.999606	0.504262	0.512900	...	-0.001009	0.0	0.532710	0.511281	0.033799	0.0	0.5
1	0.461394	0.595729	-1.238655	0.999784	0.483257	0.513369	-1.173208	0.999604	0.501266	0.512291	...	-0.000845	0.0	0.529603	0.512033	0.034787	0.0	0.5
2	0.461530	0.595861	-1.264079	0.999784	0.481376	0.513183	-1.198528	0.999603	0.498597	0.511612	...	-0.000800	0.0	0.528513	0.512119	0.034527	0.0	0.5
3	0.463032	0.599466	-1.253284	0.999765	0.481420	0.515575	-1.181970	0.999566	0.498603	0.513372	...	-0.001560	0.0	0.529767	0.519925	0.032815	0.0	0.5
4	0.465295	0.607626	-1.228310	0.999726	0.482713	0.522248	-1.166696	0.999484	0.499419	0.519148	...	-0.005636	0.0	0.538700	0.526081	0.030072	0.0	0.5
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4590	0.684548	0.511105	-1.037075	0.999985	0.706993	0.433930	-0.976545	0.999968	0.723656	0.435579	...	-0.003546	0.0	0.748149	0.430420	0.029645	0.0	0.7
4591	0.684455	0.511044	-0.985705	0.999985	0.706319	0.433798	-0.928487	0.999967	0.722609	0.435238	...	-0.003345	0.0	0.747559	0.434463	0.029979	0.0	0.7
4592	0.684569	0.509095	-1.019472	0.999984	0.706292	0.430161	-0.948244	0.999966	0.722583	0.429946	...	-0.003653	0.0	0.751030	0.432241	0.029848	0.0	0.7
4593	0.687874	0.509134	-1.102308	0.999953	0.708338	0.430230	-1.033438	0.999922	0.724405	0.429943	...	-0.004866	0.0	0.760940	0.428284	0.030739	0.0	0.7
4594	0.691391	0.509307	-1.146814	0.999941	0.710918	0.430336	-1.065239	0.999904	0.726889	0.430056	...	-0.003975	0.0	0.764780	0.431389	0.030545	0.0	0.7

4595 rows × 2004 columns

```
[35]: y
```

```
[35]: 0       Happy
      1       Happy
      2       Happy
      3       Happy
      4       Happy
              ...
      4590    Fight
      4591    Fight
      4592    Fight
      4593    Fight
      4594    Fight
      Name: class, Length: 4595, dtype: object
```

```
[35]: y
```

```
[35]: 0       Happy
      1       Happy
      2       Happy
      3       Happy
      4       Happy
              ...
      4590    Fight
      4591    Fight
      4592    Fight
      4593    Fight
      4594    Fight
      Name: class, Length: 4595, dtype: object
```

```
[36]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1234)
```

```
[37]: y_train
```

```
[37]: 1761          Sad
      1128        Happy
      1416          Sad
      1571          Sad
      3695    Victorious
                 ...
      664         Happy
      3276    Victorious
      1318          Sad
      723         Happy
      2863          Sad
      Name: class, Length: 3676, dtype: object
```

Train Machine Learning Classification Model

Train Machine Learning Classification Model

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

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

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

```
[41]: pipelines.keys()
```

```
[41]: dict_keys(['lr', 'rc', 'rf', 'gb'])
```

```
[42]: from sklearn.linear_model import LogisticRegression

      model = LogisticRegression(solver='lbfgs', max_iter=1000)
      model.fit(X, y)
```

```
[42]: LogisticRegression
      LogisticRegression(max_iter=1000)
```

```
[43]: from sklearn.preprocessing import StandardScaler
      from sklearn.linear_model import LogisticRegression
      from sklearn.pipeline import make_pipeline
```

[43]:
```
Pipeline
  StandardScaler
  LogisticRegression
```

[44]:
```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(solver='saga', max_iter=1000)
model.fit(X, y)
```

[44]:
```
LogisticRegression(max_iter=1000, solver='saga')
```

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

```
fit_models
```

```
fit_models['rc'].predict(X_test)
```

Evaluate and serialize model

```
from sklearn.metrics import accuracy_score # Accuracy metrics
import pickle
```

```
for algo, model in fit_models.items():
```

Evaluate and serialize model

```
from sklearn.metrics import accuracy_score # Accuracy metrics
import pickle
```

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

```
fit_models['rc'].predict(X_test)
```

```
y_test
```

[46]:
```
import pickle
```

[47]:
```
with open('body_language.pkl', 'wb') as f:
    pickle.dump(model,f)
```

Make Detections with Model ¶

[7]:
```
with open('body_language.pkl', 'rb') as f:
    model = pickle.load(f)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[7], line 2
      1 with open('body_language.pkl', 'rb') as f:
----> 2     model = pickle.load(f)

NameError: name 'pickle' is not defined
```

[9]:
```
model
```

```
---------------------------------------------------------------------------
```

```
        # 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)

            # Make Detections
            X = pd.DataFrame([row])
            body_language_class = model.predict(X)[0]
```

```
        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)

            # 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,
```

```
            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), 1, 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('Raw Webcam Feed', image)

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

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