AI Body Language Detector Using Mediapipe

1. Introduction

1.1 Project Overview

Body language are visual languages produced by the movement of the hands, face and body. In this project we evaluate representations based on skeleton poses, as these are explainable, person-independent, privacy-preserving, low-Dimensional representations. Basically, skeletal representations generalize over an individual's appearance and background, allowing us to focus on the recognition of motion. We present a real-time on-device body tracking pipeline that predicts hand skeleton and the whole-body notion. It is implemented via MediaPipe, a framework for building cross-platform ML solutions. We perform using pose estimation systems and analyse the applicability of the estimation systems to body language recognition by evaluating failure cases of the existing models. The proposed system and architecture demonstrate real-time inference and high prediction quality.

1.2 Purpose

Body Language Decoder helps detect and predict facial expressions, hand gestures and body pose. Facial expression recognition can help market research companies scale data and analyse quickly. The ability to perceive the shape and motion of hands can be a vital component in improving the user experience across a variety of technological domains and platforms truly. For example, it can form the basis for sign language understanding and hand posture control, and can also enable the overlay of digital content and information on top of the physical world in augmented reality.

2. Literature Survey

2.1 Existing problem

Body language is a form of nonverbal communication that consists of gestures, facial expressions, postures, and other physical behaviours. It can be used to convey a wide range of emotions, thoughts, and intentions. While body language is often used unconsciously, it can also be used deliberately to communicate with others.

Body language plays an important role in our interactions with others. It can help us to build rapport, establish trust, and express ourselves more effectively. It can also be used to deceive or manipulate others.

There are a number of existing problems related to body language. These problems can be broadly categorized into two groups:

- **Understanding body language:**

Difficulty in interpreting the meaning of body language cues: Body language cues can be ambiguous and can be interpreted in different ways. This can lead to misunderstandings and misinterpretations.
Lack of awareness of body language cues: Many people are not aware of the subtle cues that can be used to convey body language. This can lead to them missing important information or misinterpreting the intentions of others.
Cultural differences in body language: The meanings of body language cues can vary from culture to culture. This can lead to confusion and misunderstandings when people from different cultures interact.

- **Using body language effectively:**

Difficulty in controlling body language: Our body language is often a reflection of our emotions, thoughts, and intentions. This can make it difficult to control our body language, especially when we are feeling strong emotions.
Unintentional use of body language: We may sometimes use body language cues that we are not aware of. This can lead to misunderstandings and misinterpretations.
Inability to use body language to deceive or manipulate: While body language can be used to deceive or manipulate others, this can be difficult to do effectively. People are often good at detecting deception, and using body language that is not congruent with our intentions can backfire.

2.2 References

Birdwhistele, R. L. (1976). Kinesics and context: Essays on body motion communication. Philadelphia: University of Pennsylvania Press.
Ekman, P., & Friesen, W. V. (1969). Nonverbal communication. New York: Psychological Dimensions.
Mehrabian, A. (1972). Nonverbal communication. Chicago: Aldine-Atherton.
Morris, D. (1977). Manwatching: A field guide to human behavior. New York: Crown Publishers.
Argyle, M., & Cook, M. (1976). Gaze and mutual glance. Cambridge: Cambridge University Press.
Nadig, A. S., & Harper, R. M. (2009). Nonverbal cues, such as facial expressions, influence the perceived gender of a virtual agent's voice. International Journal of Human-Computer Studies, 67(6), 478-490.

In addition to the problems listed above, there are a number of other challenges that researchers face in studying body language. These challenges include:
The difficulty of measuring body language: Body language is a complex phenomenon that is difficult to measure objectively.

The need for more longitudinal studies: There is a need for more longitudinal studies that track people's body language over time.

The need for more cross-cultural studies: There is a need for more cross-cultural studies to determine how body language varies from culture to culture.

2.3 Problem Statement Definition

There are a number of ways to address the existing problems related to body language. These include:

Developing more accurate methods for interpreting body language cues: Researchers are developing new methods for interpreting body language cues, such as using machine learning to analyse facial expressions and gestures.

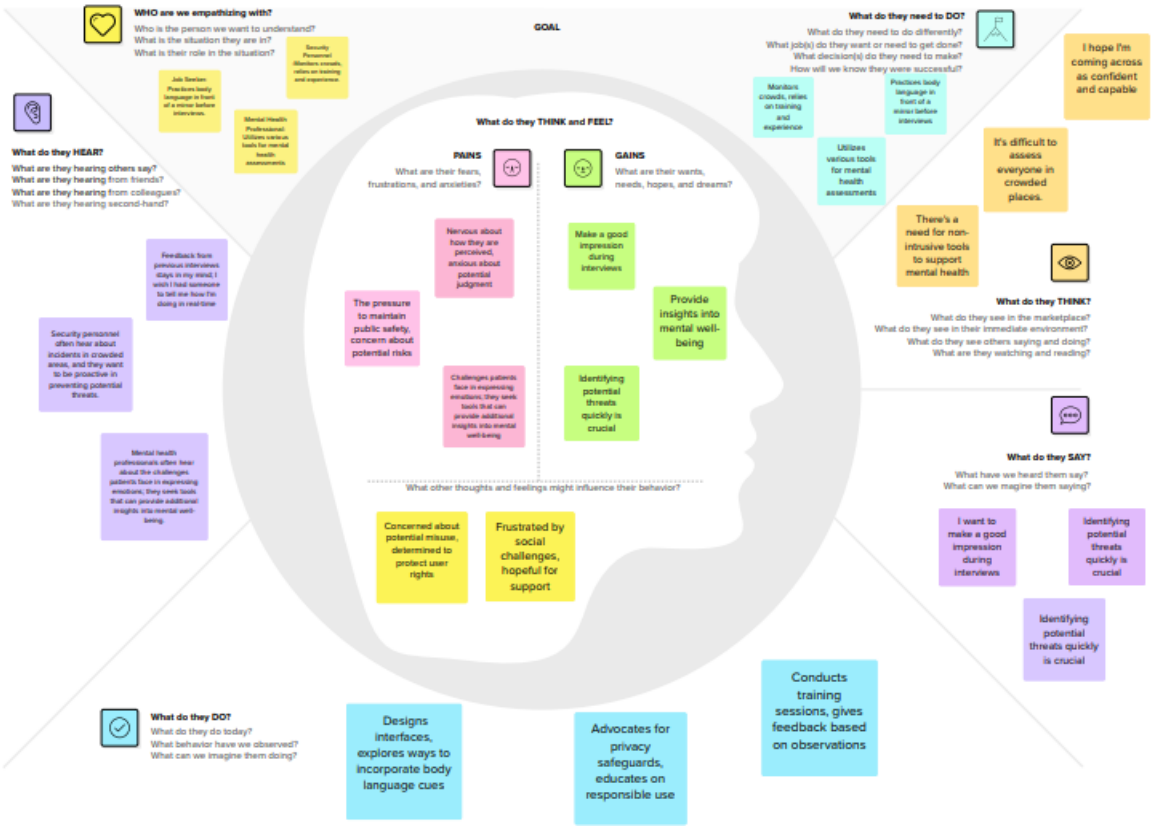
Increasing awareness of body language cues: There are a number of resources available to help people learn about body language, such as books, articles, and online courses.

Teaching people how to use body language effectively: There are a number of workshops and training programs available to teach people how to use body language effectively.

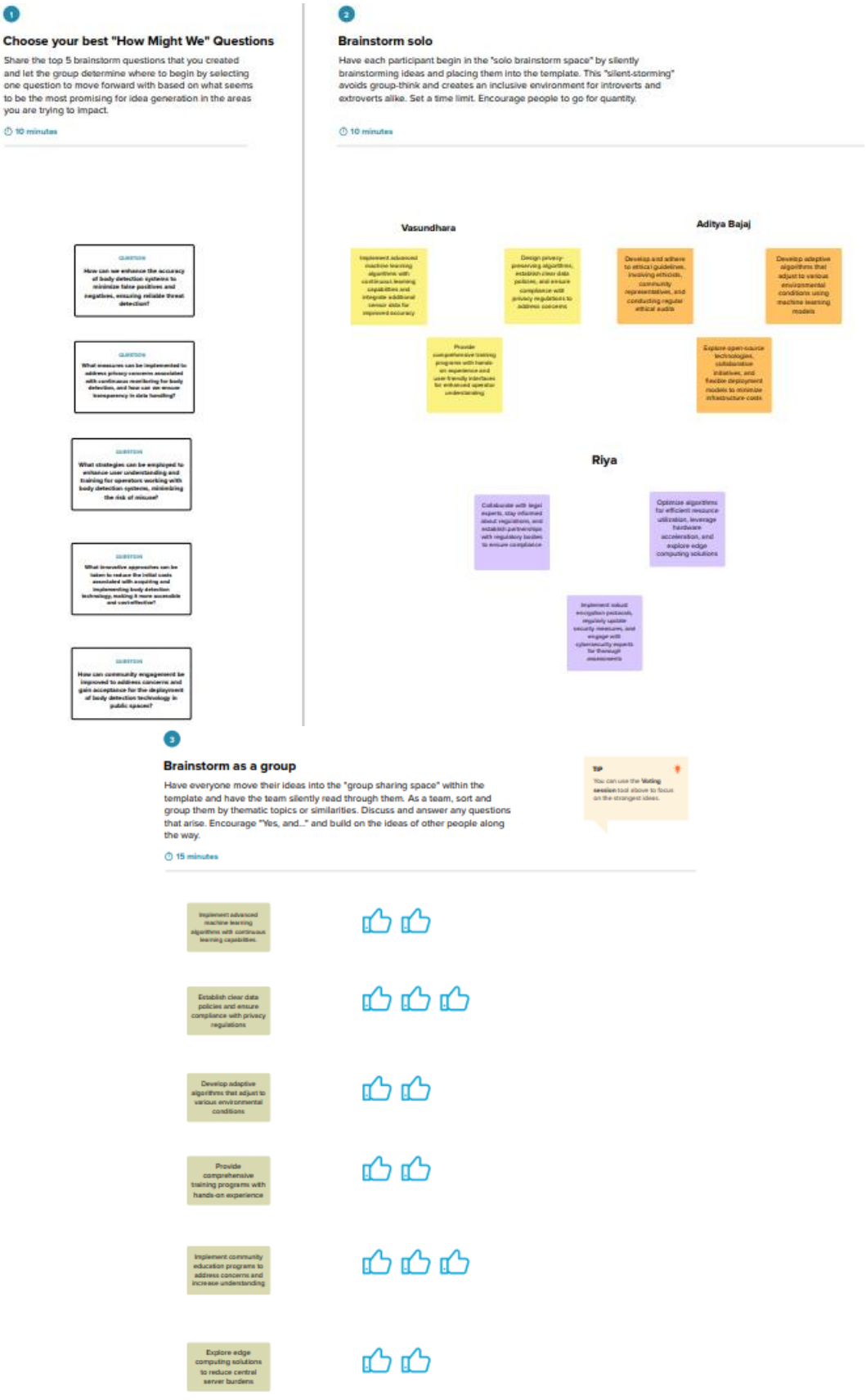
By addressing the existing problems related to body language, we can improve our communication skills and our ability to understand and interact with others.

3. Ideation & Proposed Solution

3.1 Empathy Map Canvas



3.2 Ideation & Brainstorming



4. Requirement Analysis

4.1 Functional requirement

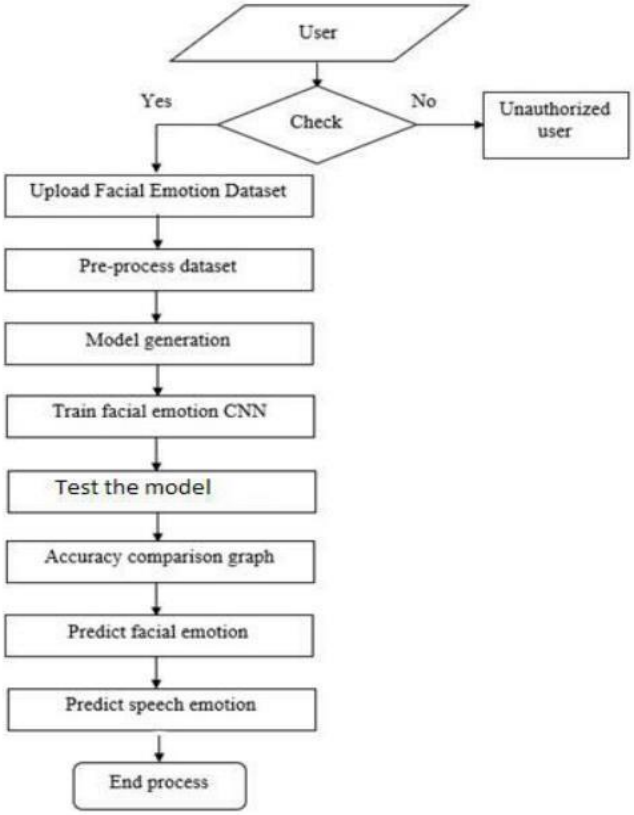
- The system should be able to accurately detect and classify a variety of body language cues, including facial expressions, gestures, and postures.
- The system should be able to provide real-time feedback on the user's body language.
- The system should be able to be customized to the specific needs of the user.
- The system should be able to be integrated with other applications, such as chatbots and virtual assistants.
- The system should be able to learn from the user's body language and adapt its responses accordingly.
- The system should be able to generate creative text formats of text content, like poems, code, scripts, musical pieces, email, letters, etc. It will try its best to fulfil all your requirements.
- The system should be able to translate between different languages.
- The system should be able to answer the user's questions in a comprehensive and informative way, even if they are open ended, challenging, or strange.

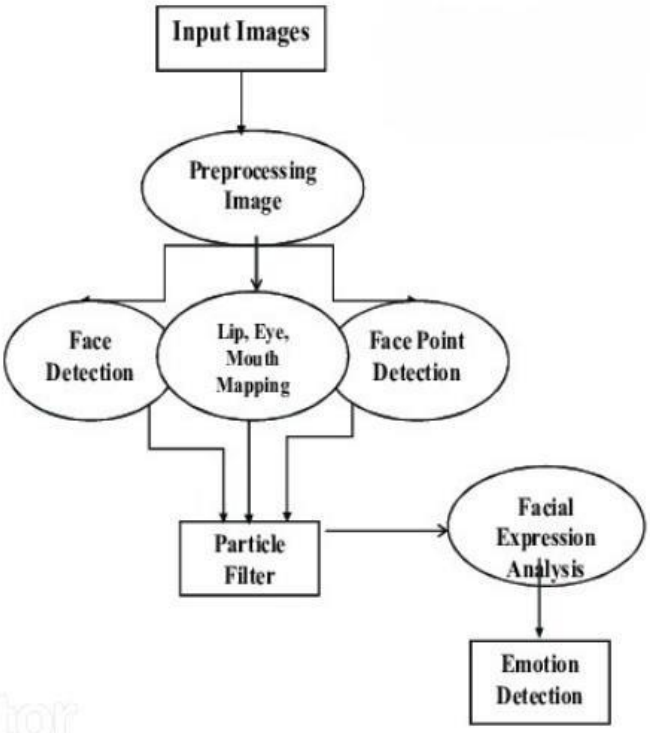
4.2 Non-Functional requirements

- The system should be easy to use and understand.
- The system should be reliable and accurate.
- The system should be efficient and scalable.
- The system should be secure and protect the user's privacy.
- The system should be maintainable and upgradable.
- In addition to the above, here are some specific functional and non-functional requirements that are relevant to body language detection using AIML.
- The system should be able to run on a variety of devices, including smartphones, tablets, and computers.
- The system should be able to handle a high volume of traffic.
- The system should be able to be deployed in a variety of environments, including production and development.
- The system should be able to be monitored and maintained remotely.
- The system should be able to be updated without disrupting the user's experience.
- These are just a few of the many functional and non-functional requirements that need to be considered when developing a body language detection system using AIML. The specific requirements will vary depending on the specific needs of the project.

5. Project Design

5.1 Data Flow Diagrams & User Stories

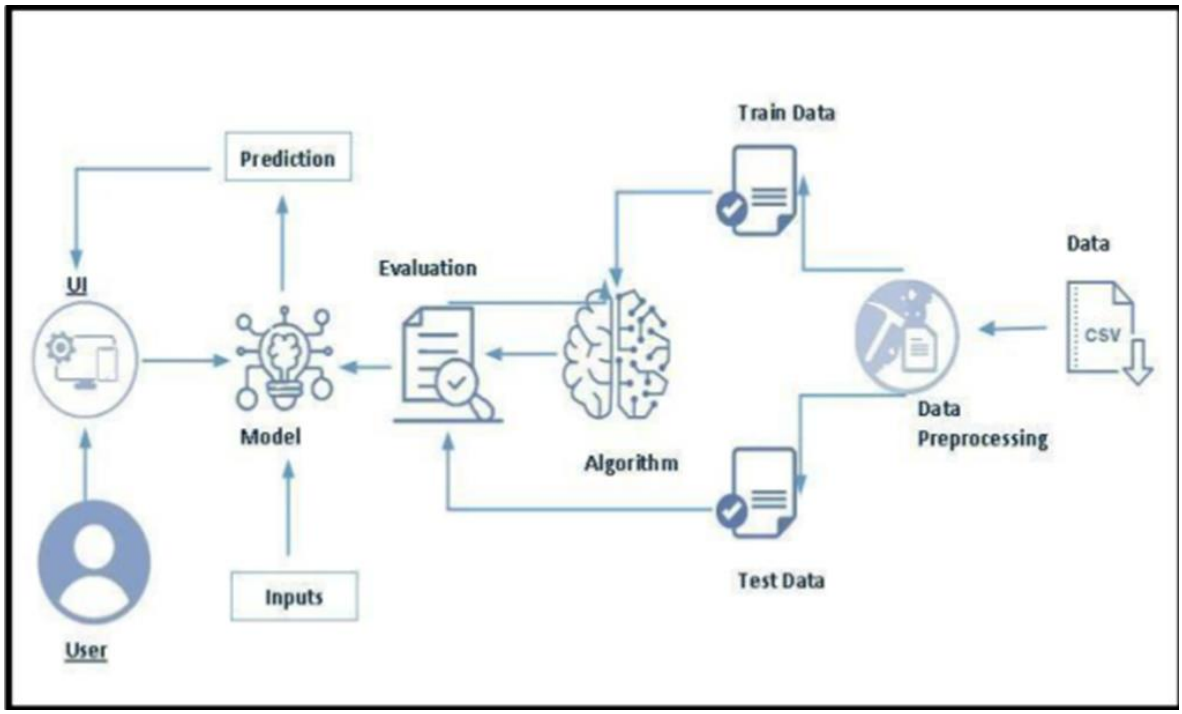




User Stories

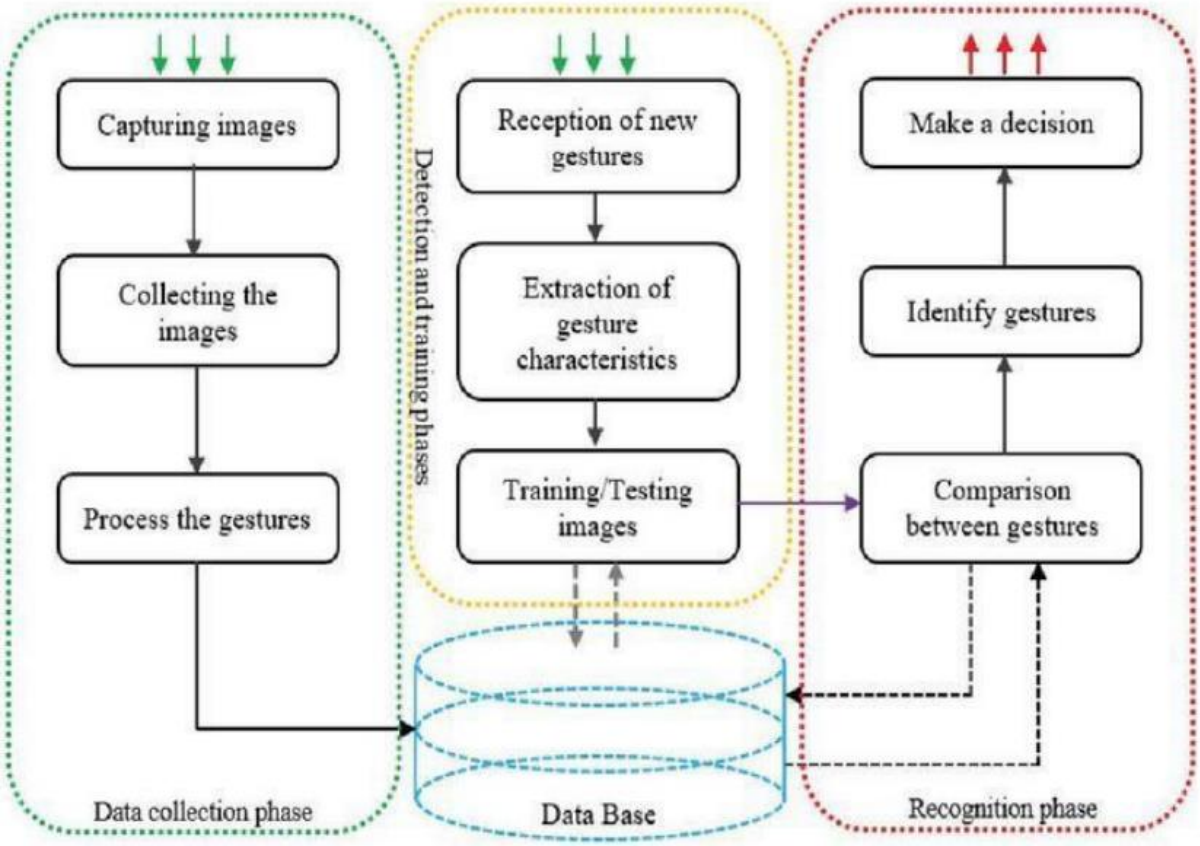
User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
AI System Integrators	Project Setup & Infrastructure	USN-1	Set up the development environment with the required tools and frameworks to start the AI body language detection project for emotion recognition (e.g., detecting sadness, anger).	The development environment is successfully configured with all necessary tools and frameworks required for AI body language detection through Mediapipe.	High	Sprint-1
Dataset Curators	Development Environment	USN-2	Gather a diverse dataset of images containing different body language expressions for training the AI body language detection model, specifically focusing on emotions like sadness and anger.	Successfully gathered a diverse dataset of images depicting various body language expressions with a focus on sadness and anger.	High	Sprint-1
Individual Contributors	Data Collection	USN-3	Preprocess the collected dataset by resizing images, normalizing pixel values, and splitting it into training and validation sets for AI body language detection.	The dataset is successfully pre-processed for AI body language detection.	High	Sprint-2
Research Scientists	Data Preprocessing	USN-4	Explore and evaluate different AI body language detection architectures within Mediapipe (e.g., pose estimation models) to select the most suitable model for emotion recognition (e.g., sadness and anger).	Explored various AI body language detection models within Mediapipe for emotion recognition.	High	Sprint-2
Non-Profit Organizations	Model Development	USN-5	Train the selected AI body language detection model using the pre-processed dataset and monitor its performance on the validation set, specifically focusing on recognizing emotions like sadness and anger.	Conducted validation of the trained AI body language detection model for recognizing emotions.	High	Sprint-3
Educational Institutions	Training	USN-6	Implement data augmentation techniques (e.g., rotation, flipping) to improve the AI body language detection model's robustness and accuracy in recognizing emotions such as sadness and anger.	Tested data augmentation techniques to enhance the model's robustness in emotion recognition.	Medium	Sprint-3
IT System Managers	Model Deployment & Integration	USN-7	Deploy the trained AI body language detection model, integrating it into an API or web service to make it accessible for emotion recognition, particularly for emotions like sadness and anger. Integrate the model's API into a user-friendly web interface for users to upload images and receive emotion classification results.	Checked the scalability of the deployed model's API for emotion recognition.	Medium	Sprint-4
Quality Assurance Analysts	Testing & Quality Assurance	USN-8	Conduct thorough testing of the AI body language detection model and web interface to identify and report any issues or bugs. Fine-tune the model hyperparameters and optimize its performance based on user feedback and testing results for emotion recognition.	Created a web application for testing purposes.	Medium	Sprint-5

5.2 Solution Architecture



6. Project Planning & Scheduling

6.1 Technical Architecture



6.2 Sprint Planning & Estimation

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Project setup & Infrastructure	USN-1	Set up the development environment with the required tools and frameworks to start the AI body language detection project for emotion recognition (e.g., detecting sadness, anger).	1	High	Riya
Sprint-1	Development Environment	USN-2	Gather a diverse dataset of images containing different body language expressions for training the AI body language detection model, specifically focusing on emotions like sadness and anger	2	High	Aditya Bajaj
Sprint-2	Data collection	USN-3	Preprocess the collected dataset by resizing images, normalizing pixel values, and splitting it into training and validation sets for AI body language detection.	2	High	Vasundhara
Sprint-2	Data preprocessing	USN-4	Explore and evaluate different AI body language detection architectures within Mediapipe (e.g., pose estimation models) to select the most suitable model for emotion recognition (e.g., sadness and anger).	3	High	Aditya Bajaj
Sprint-3	Model Development	USN-5	Train the selected AI body language detection model using the pre-processed dataset and monitor its performance on the validation set, specifically focusing on recognizing emotions like sadness and anger	4	High	Vasundhara
Sprint-3	Training	USN-6	Implement data augmentation techniques (e.g., rotation, flipping) to improve the AI body language detection model's robustness and accuracy in recognizing emotions such as sadness and anger.	6	Medium	Riya
Sprint-4	Model deployment & Integration	USN-7	Deploy the trained AI body language detection model, integrating it into an API or web service to make it accessible for emotion recognition, particularly for emotions like sadness and anger. Integrate the model's API into a user-friendly web interface for users to upload images and receive emotion classification results.	1	Medium	Aditya Bajaj
Sprint-5	Testing & quality assurance	USN-8	Conduct thorough testing of the AI body language detection model and web interface to identify and report any issues or bugs. Finetune the model hyperparameters and optimize its performance based on user feedback and testing results for emotion recognition.	1	Medium	Vasundhara

6.3 Sprint Delivery Schedule

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	3	3 Days	25 Oct 2023	28 Oct 2023	20	28 Oct 2023
Sprint-2	5	5 Days	29 Oct 2023	2 Nov 2023		
Sprint-3	10	5 Days	3 Nov 2023	7 Nov 2023		
Sprint-4	1	4 Days	8 Nov 2023	11 Nov 2023		
Sprint-5	1	4 Days	12 Nov 2023	15 Nov 2023		

7. Coding & Solutioning

7.1 Feature

Prerequisites:

To complete this project, we must require the following software's, concepts, and packages

Anaconda Navigator is a free and open-source distribution of the Python and R programming languages for data science and machine learning related applications. It can be installed on Windows, Linux, and macOS.Conda is an open-source, cross-platform, package management system. Anaconda comes with so very nice tools like JupyterLab, Jupyter Notebook, Visual Studio Code.

To build Machine learning models you must require the following packages

- **Numpy**:
  - It is an open-source numerical Python library. It contains a multidimensional array and matrix data structures and can be used to perform mathematical operations

- **Scikit-learn:**

  - It is a free machine learning library for Python. It features various algorithms like support vector machine, random forests, and k-neighbours, and it also supports Python numerical and scientific libraries like NumPy and SciPy.

  - Scikit-learn is an open-source machine learning library that provides simple and efficient tools for data analysis and modelling, including various machine learning algorithms and utilities for tasks such as classification, regression, clustering, dimensionality reduction, and more.

- **MediaPipe:**
  - MediaPipe is an open-source library developed by Google that provides tools for building applications that involve real-time perception of human body movements, facial expressions, and hand gestures, among other things. It simplifies the development of applications that use computer vision and machine learning for tasks such as pose detection, face recognition, and hand tracking.

- **TensorFlow:**
  - TensorFlow is an open-source machine learning framework developed by the Google Brain team. It is designed to facilitate the development and deployment of machine learning models, particularly deep learning models. TensorFlow is widely used in both research and industry for tasks such as image and speech recognition, natural language processing, and various other machine learning applications.
- **OpenCV:**
  - OpenCV (Open-Source Computer Vision Library) is an open-source computer vision and machine learning software library. It provides a wide range of tools and algorithms for image and video processing, computer vision tasks, and machine learning applications. OpenCV is widely used in both academia and industry for tasks such as object recognition, feature extraction, image processing, and more.

- **Python packages:**
  - Open Anaconda Prompt as administrator
  - Type “pip install NumPy” and click enter.
  - Type “pip install pandas” and click enter.
  - Type “pip install scikit-learn” and click enter.
  - Type “pip install TensorFlow” and click enter.
  - Type “pip install OpenCV” and click enter
  - Type “pip install mediapipe” and click enter.
  - Type “pip install streamlit” and click enter.

CNN: a convolutional neural network is a class of deep neural networks, most commonly applied to analysing visual imagery. CNN Basic

Project Objectives:
By the end of this project you will:

- Know fundamental concepts and techniques of Machine Learning.
- Gain a broad understanding on Mediapipe.
- Know how to pre-process/clean the data using different data preprocessing techniques.
- Know how to build a web application using the Streamlit framework.

Project Flow:

- The user interacts with the UI (User Interface) to choose the image.
- The chosen image analysed by the model which is integrated with streamlit application.
- CNN Models analyse the image, then prediction is showcased.

To accomplish this, we have to complete all the activities and tasks listed below

- Data Collection.
- Create Train and Test Folders.
- Data Preprocessing.
- Import Trainset and Test set
- Model Building
- Import the model building Libraries
- Initializing the model
  - Training and testing the model
  - Save the Model
  - Application Building
    - Create an HTML file
    - Build Python Code

Project:

Detecting body language using machine learning can offer several benefits:

1. **Enhanced Communication Understanding**: By analysing body language cues such as facial expressions, gestures, and posture, machine learning algorithms can provide insights into the underlying emotions, intentions, and attitudes of individuals. This can help in better understanding non-verbal cues and improving communication comprehension.

2. **Improved Interactions**: Machine learning models can be trained to interpret body language signals in real-time, facilitating more effective interactions. This can be particularly useful in customer service, sales, negotiations, and other domains where understanding and responding to non-verbal cues can make a significant difference.

3. **Emotional Analysis**: Body language analysis can help in gauging emotional states accurately. By recognizing facial expressions, body movements, and other physical cues, machine learning algorithms can identify emotions such as happiness, sadness, anger, or surprise. This information can be beneficial in various fields, including mental health, market research, and user experience design.

4. **Deception Detection:** Machine learning algorithms trained on body language data can assist in detecting signs of deception or dishonesty. Certain physical cues, such as avoiding eye contact, fidgeting, or inconsistent gestures, can indicate potential deception. This can be valuable in security, law enforcement, and forensic investigations.

Project Structure:

Create the Project folder which contains files as shown below

- We are building a Streamlit application which needs a python script web.py for a website.
- Model folder contains your saved models.
- The Training folder contains the code for building/training/testing the model.
- The Dataset folder contains a .csv file created by you.

Milestone 1: Define Problem / Problem Understanding

Activity 1: Specify the business problem

- Customer Assistance: Customers may need assistance, but it's not always easy for staff to identify who requires help.

- Product Interest: Understanding customer interest in specific products or sections of the store is challenging, making targeted recommendations difficult.
- Staff Optimization: Efficiently allocating staff resources to areas with high customer demand is a problem, leading to potential understaffing or overstaffing in certain sections.
- Feedback Collection: Traditional methods of collecting feedback, such as surveys, may not provide real-time insights into customer experiences within the store.

Activity 2: Business requirements

Here are some potential business requirements for an ecommerce product delivery estimation predictor using machine learning:

1. Accurate prediction: The predictor must be able to accurately predict the body language of the person. The accuracy of the prediction is crucial for the client. The client could be a doctor or a business person or anyone.

2. User-friendly interface: The predictor must have a user-friendly interface that is easy to navigate and understand. The interface should present the results of the predictor in a clear and concise manner.

3. Scalability: The predictor must be able to scale up based on the prediction from our product. The model should be able to handle any size of data without compromising on its accuracy or efficiency.

Activity 3: Social or Business Impact

1. Sales and Customer Service: Detecting and interpreting body language cues during sales interactions or customer service interactions can provide valuable insights. Sales professionals can gauge customer interest, satisfaction, or hesitation, enabling them to adjust their approach accordingly. Customer service representatives can better understand customer needs and emotions, leading to more personalized and satisfactory experiences.

2. Public Speaking and Presentations: Body language analysis can benefit individuals delivering presentations or engaging in public speaking. Machine learning can provide real-time feedback on gestures, posture, and facial expressions, helping speakers improve their delivery and connect with the audience more effectively.

Milestone 2: Data Collection & Preparation

ML depends heavily on data. It is the most crucial aspect that makes algorithm training possible.

● Importing required packages:

Importing required packages :

```
import pandas as pd
from sklearn.model_selection import train_test_split
```

Read the recorded .csv file :

Paste the path of the .csv file in the quotes.

```
df = pd.read_csv('/Users/rishi/BTECH/Programming/My_Projects/AI_Body_Language_Detector/Dataset/coords.csv')
```

After this you can just run the following commands to view the data :

- ➢ **" df.head() "**
- ➢ **" df.tail() "**

Activity 2: Data Preparation

As we have understood how the data is, let's pre-process the collected data.

Divide the data into dependent and independent variable

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

Activity 3: Splitting data into train and test and validation sets

Now let's split the Dataset into train and test sets.
The spilt will be in 8:2 ratio : train : test respectively.

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

Milestone 3: Model Building

Now it's time to build our Convolutional Neural Networking which contains an input layer along with the convolution, max-pooling, and finally an output layer.

Activity 1: Importing the Model Building Libraries

Importing the necessary libraries

```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

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

Activity 2: Designing a pipeline to train on multiple algorithms

In this step we are going to create a pipeline which will contain LogisticRegression, RidgeClassifi RandomForestClassifier and GradientBoostingClassifier algorithms.

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

Activity 3: Train the models

Once the pipeline is created we fit all the training data to every model in the pipeline by using the following code.

```
fit_models = {}
for algo, pipeline in pipelines.items():
    model = pipeline.fit(X_train, y_train)
    fit_models[algo] = model
✓ 1m 29.7s
```

Milestone 4: Model Deployment

Activity 1: Importing packages

First we will import required packages

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

Activity 2: Model Evaluation

We will just predict all the models on test data and display the accuracies for each model.

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

✓ 0.1s

```
lr 0.9924812030075187
rc 1.0
rf 0.9924812030075187
gb 0.9774436090225563
```

Model Summary

Activity 1: Importing the model

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
```

Activity 2: Initializing the model

Keras has 2 ways to define a neural network:

- Sequential
- Function API

The Sequential class is used to define linear initializations of network layers which then, collectively, constitute a model. In our example below, we will use the Sequential constructor to create a model, which will then have layers added to it using the add() method.

```
# Initializing the model
model=Sequential()
```

Activity 3: Adding CNN Layers

- We are adding a convolution layer with activation function as “relu” and with a small filter size (3,3) and the number of filters (32) followed by a max-pooling layer.
- Max pool layer is used to down sample the input.(Max pooling is a pooling operation that selects the maximum element from the region of the feature map covered by the filter) ● Flatten layer flattens the input. Does not affect the batch size.

```
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(64, 64, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Summary Model

```
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 62, 62, 32)        896

 max_pooling2d_2 (MaxPoolin  (None, 31, 31, 32)        0
 g2D)

 conv2d_3 (Conv2D)           (None, 29, 29, 64)        18496

 max_pooling2d_3 (MaxPoolin  (None, 14, 14, 64)        0
 g2D)

 flatten_1 (Flatten)         (None, 12544)             0

 dense_4 (Dense)             (None, 128)               1605760

 dense_5 (Dense)             (None, 1)                 129

=================================================================
Total params: 1625281 (6.20 MB)
Trainable params: 1625281 (6.20 MB)
Non-trainable params: 0 (0.00 Byte)
```

Activity 4: Configure The Learning Process

- The compilation is the final step in creating a model. Once the compilation is done, we can move on to the training phase. The loss function is used to find errors or deviations in the learning process. Keras requires a loss function during the model compilation process.
- Optimization is an important process that optimizes the input weights by comparing the prediction and the loss function. Here we are using Adam optimizer
- Metrics are used to evaluate the performance of your model. It is similar to the loss function, but not used in the training process

Compiling the Model

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Activity 5: Train The model

Now, let us train our model with our image dataset. The model is trained for 30 epochs and after every epoch, the current model state is saved if the model has the least loss encountered till that time. We can see that the training loss decreases in almost every epoch till 30 epochs and probably there is further scope to improve the model.

```
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.3, random_state=1234)

num_epochs = 10
fit_models = {algo: [] for algo in pipelines}

for algo, pipeline in pipelines.items():
    model = None  # Initialize model for each algorithm

    # Create a StandardScaler with feature names
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train.values)
    X_val_scaled = scaler.transform(X_val.values)

    for epoch in range(num_epochs):
        model = pipeline.fit(X_train_scaled, y_train)

        # Training accuracy
        y_train_pred = model.predict(X_train_scaled)
        train_accuracy = accuracy_score(y_train, y_train_pred)
        print(f'Epoch {epoch + 1} - Training Accuracy ({algo}): {train_accuracy}')

        # Validation accuracy
        y_val_pred = model.predict(X_val_scaled)
        validation_accuracy = accuracy_score(y_val, y_val_pred)
        print(f'Epoch {epoch + 1} - Validation Accuracy ({algo}): {validation_accuracy}')

        # Store the model for each epoch
        fit_models[algo].append(model)
```

```
Epoch 1 - Training Accuracy (lr): 1.0
Epoch 1 - Validation Accuracy (lr): 1.0
Epoch 2 - Training Accuracy (lr): 1.0
Epoch 2 - Validation Accuracy (lr): 1.0
Epoch 3 - Training Accuracy (lr): 1.0
Epoch 3 - Validation Accuracy (lr): 1.0
Epoch 4 - Training Accuracy (lr): 1.0
Epoch 4 - Validation Accuracy (lr): 1.0
Epoch 5 - Training Accuracy (lr): 1.0
Epoch 5 - Validation Accuracy (lr): 1.0
Epoch 6 - Training Accuracy (lr): 1.0
Epoch 6 - Validation Accuracy (lr): 1.0
Epoch 7 - Training Accuracy (lr): 1.0
Epoch 7 - Validation Accuracy (lr): 1.0
Epoch 8 - Training Accuracy (lr): 1.0
Epoch 8 - Validation Accuracy (lr): 1.0
Epoch 9 - Training Accuracy (lr): 1.0
Epoch 9 - Validation Accuracy (lr): 1.0
Epoch 10 - Training Accuracy (lr): 1.0
Epoch 10 - Validation Accuracy (lr): 1.0
Epoch 1 - Training Accuracy (rc): 1.0
Epoch 1 - Validation Accuracy (rc): 1.0
Epoch 2 - Training Accuracy (rc): 1.0
Epoch 2 - Validation Accuracy (rc): 1.0
Epoch 3 - Training Accuracy (rc): 1.0
Epoch 3 - Validation Accuracy (rc): 1.0
Epoch 4 - Training Accuracy (rc): 1.0
Epoch 4 - Validation Accuracy (rc): 1.0
Epoch 5 - Training Accuracy (rc): 1.0
Epoch 5 - Validation Accuracy (rc): 1.0
```

```
Epoch 6 - Training Accuracy (rc): 1.0
Epoch 6 - Validation Accuracy (rc): 1.0
Epoch 7 - Training Accuracy (rc): 1.0
Epoch 7 - Validation Accuracy (rc): 1.0
Epoch 8 - Training Accuracy (rc): 1.0
Epoch 8 - Validation Accuracy (rc): 1.0
Epoch 9 - Training Accuracy (rc): 1.0
Epoch 9 - Validation Accuracy (rc): 1.0
Epoch 10 - Training Accuracy (rc): 1.0
Epoch 10 - Validation Accuracy (rc): 1.0
Epoch 1 - Training Accuracy (rf): 1.0
Epoch 1 - Validation Accuracy (rf): 1.0
Epoch 2 - Training Accuracy (rf): 1.0
Epoch 2 - Validation Accuracy (rf): 1.0
Epoch 3 - Training Accuracy (rf): 1.0
Epoch 3 - Validation Accuracy (rf): 1.0
Epoch 4 - Training Accuracy (rf): 1.0
Epoch 4 - Validation Accuracy (rf): 1.0
Epoch 5 - Training Accuracy (rf): 1.0
Epoch 5 - Validation Accuracy (rf): 1.0
Epoch 6 - Training Accuracy (rf): 1.0
Epoch 6 - Validation Accuracy (rf): 1.0
Epoch 7 - Training Accuracy (rf): 1.0
Epoch 7 - Validation Accuracy (rf): 1.0
Epoch 8 - Training Accuracy (rf): 1.0
Epoch 8 - Validation Accuracy (rf): 1.0
Epoch 9 - Training Accuracy (rf): 1.0
Epoch 9 - Validation Accuracy (rf): 1.0
Epoch 10 - Training Accuracy (rf): 1.0
Epoch 10 - Validation Accuracy (rf): 1.0
```

Milestone 5: Application Building

Now that we have trained our model, let us build our flask application which will be running in our local browser with a user interface.

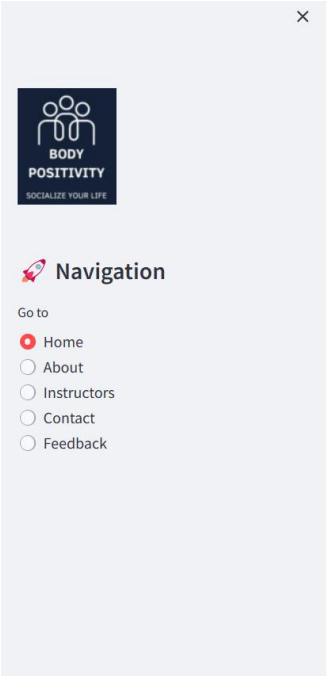
In the flask application, the input parameters are taken from the HTML page These factors are then given to the model to know to predict the type of Garbage and showcased on the HTML page to notify the user. Whenever the user interacts with the UI and selects the “Image” button, the next page is opened where the user chooses the image and predicts the output.

Activity 1: Create HTML Pages

- We use HTML to create the front-end part of the web page.
- Here, we have created 3 HTML pages- home.html, intro.html, and upload.html
- home.html displays the home page.
- Intro.html displays an introduction about the project
- upload.html gives the emergency alert

  For more information regarding HTML. We also use CSS-main.css to enhance our functionality and view of HTML pages.

Home Section: -



BODY POSITIVITY
SOCIALIZE YOUR LIFE

Navigation

Go to

- Home
- About
- Instructors
- Contact
- Feedback

📘 Know About Emotion Detection

International Journal of Advance Research, Ideas and Innovations in Technology

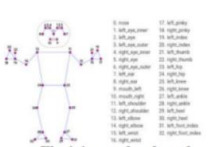


Fig 1.1 pose landmark

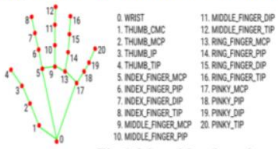


Fig 1.2 hand landmark

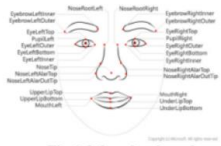


Fig 1.3 face landmark

Sentiment analysis is the process of detecting positive or negative sentiment in a sentence. It's often used by businesses to detect sentiment in social data, gauge brand reputation, and understand customers. Sentiment Analysis is already widely used by different companies to use towards their product or brand in the digital world. However, in the offline world users are also interacting with the brands and products in retail stores, showrooms, social media etc. and solutions to measure user's reaction/expression automatically under such settings has remained a challenging task. Emotion Detection from facial expressions or a text using AI can be a viable alternative to automatically measure consumer's engagement with their content and brands. On the other hand detecting body language, which is considered as one of the most effective communication method, can help interviewers analyze the candidate during the process.

This can be achieved by creating an excel sheet of coordinate points of landmarks of desired poses and training the model on that. This trained model can then be stored and used later for predicting user's gestures.

Dep

🌐 Identifying Body Language

--> Just click on 'Run' button to let us explore your moods and emotions.

📚 Instructions

-> Click on the Run button and wait for 20 seconds for execution of the program.

-> Do not move frequently as it may confuse the AI trained model.

-> Look at all the gestures to test all the emotions and to get the required result.

Run

BODY POSITIVITY
SOCIALIZE YOUR LIFE

Navigation

Go to

- Home
- About
- Instructors
- Contact
- Feedback

About Section: -

BODY POSITIVITY
SOCIALIZE YOUR LIFE

Navigation

Go to

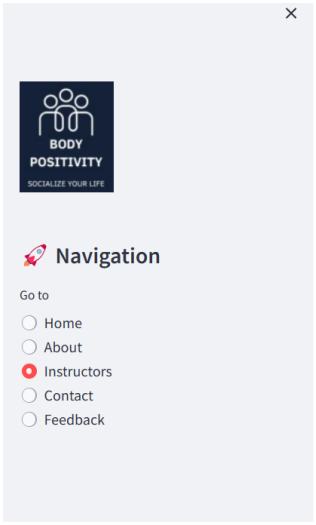
- Home
- About
- Instructors
- Contact
- Feedback

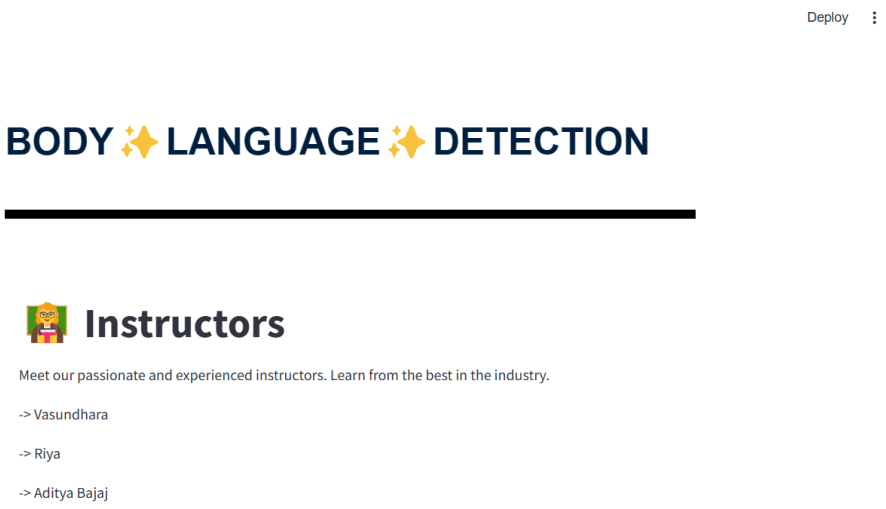
BODY ✨ LANGUAGE ✨ DETECTION

📘 About Us

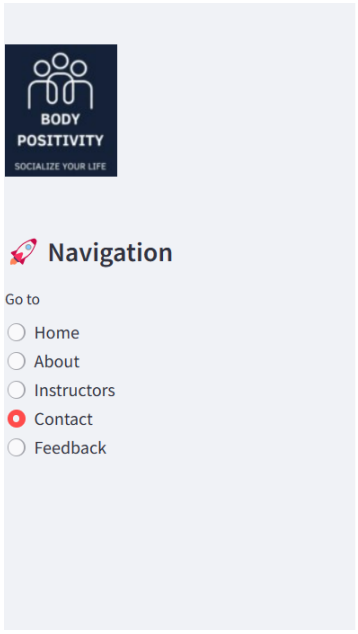
Discover the story behind body language detection. Your journey into AI begins here!

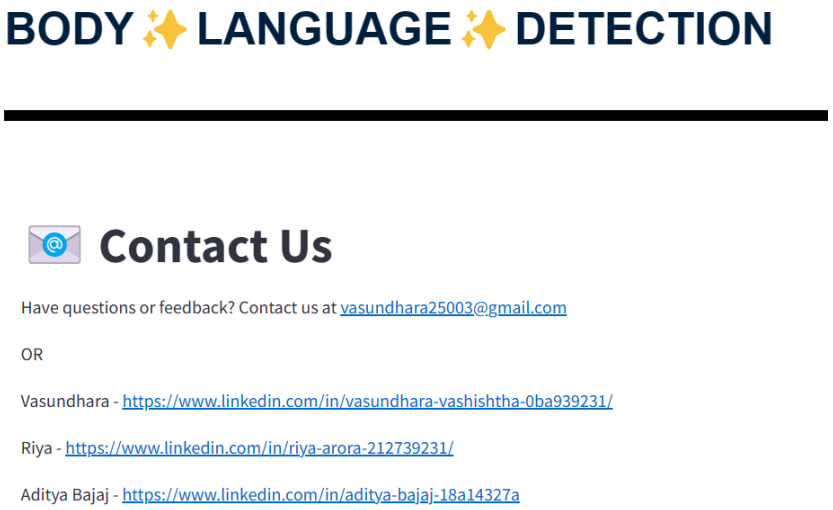
Instructors Section: -



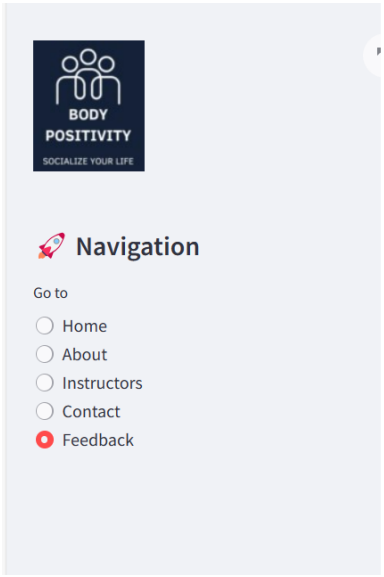


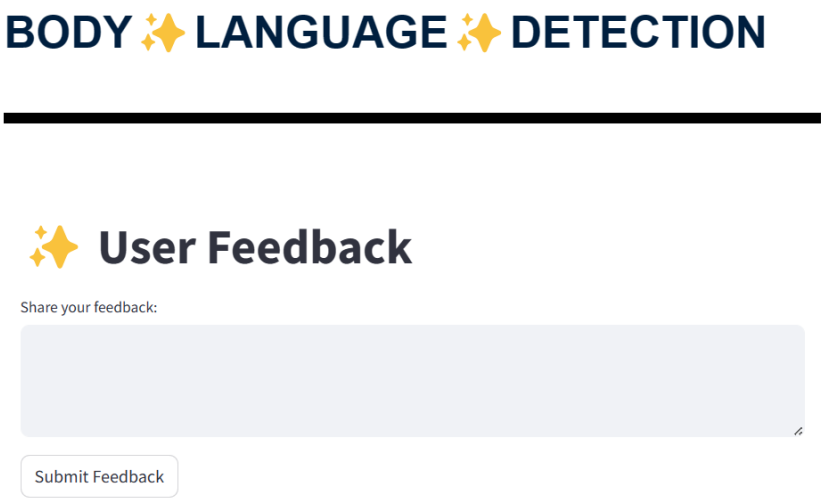
Contact Us: -





Feedback





Activity 2: Build Python code

```python
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 time

# Create a dictionary to store the state
state = {'run': False, 'end_button_clicked': False}

# Load the pre-trained model
with open('body_language.pkl', 'rb') as f:
    model = pickle.load(f)

# Setup MediaPipe
mp_drawing = mp.solutions.drawing_utils
mp_holistic = mp.solutions.holistic

def run_detection():
    # Display the webcam feed and body language detection
    camera = cv2.VideoCapture(0)
    FRAME_WINDOW = st.image([])
    end_button_placeholder = st.empty()
    end_button_displayed = False

    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 = False

            image = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
            image[:, :, 1] = image[:, :, 1] * 1.0  # Increase saturation
```

```
        image = cv2.cvtColor(image, cv2.COLOR_HSV2RGB)
        # Adjust brightness and contrast
        alpha = 1.5  # Contrast control (1.0 means no change)
        beta = 25  # Brightness control (0 means no change)

        adjusted_image = cv2.convertScaleAbs(image, alpha=alpha, beta=beta)

        # Make Detections
        results = holistic.process(image)

        # face_landmarks, pose_landmarks, left_hand_landmarks, right_hand_land

        # Recolor image back to BGR for rendering
        image.flags.writeable = True

        # 1. Draw face landmarks
        # 1. Draw face landmarks without grid
        # 1. Draw face landmarks without dots
        if results.face_landmarks is not None:
            pass  # Do not draw anything

        # 2. Right hand without dots
        if results.right_hand_landmarks is not None:
            pass  # Do not draw anything

        # 3. Left hand without dots
        if results.left_hand_landmarks is not None:
            pass  # Do not draw anything

        # 4. Pose Detections without dots
        if results.pose_landmarks is not None:
            pass  # Do not draw anything

        # 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
                                      pose]).flatten())

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

            # Concatenate 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),
                          (255, 255, 255),  # White box
                          -1)

            cv2.putText(image, body_language_class, coords,
                        cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 0), 2, cv2.LINE_AA)

            # Get status box
            cv2.rectangle(image, (0, 0), (250, 60), (255, 255, 255), -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, (0, 0, 0),
                        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, (0, 0, 0), 2, cv2.LINE_AA)
        except:
            pass

        FRAME_WINDOW.image(image, channels="RGB", output_format="JPEG")

        if state['run'] and not state['end_button_clicked'] and not end_button_displayed:
            end_button_clicked = end_button_placeholder.button("End", key="end_button")
            state['end_button_clicked'] = end_button_clicked
            end_button_displayed = True

        if state['end_button_clicked']:
            break

    camera.release()
    cv2.destroyAllWindows()

def homepage():
    st.image("1.jpg", width=700)
    st.markdown("<hr style='border: 5px solid #000000; width: 100%;'>", unsafe_allow_html=True)
    st.title(" Know About Emotion Detection ")
    st.image("2.png",width=900)
    st.markdown("<hr style='border: 5px solid #000000; width: 100%;'>", unsafe_allow_html=True)
    st.title(" Identifying Body Language ")
    st.write("--> Just click on 'Run' button to let us explore your moods and emotions.")
```

```
        st.markdown("<hr style='border: 5px solid #000000; width: 100%;'>", unsafe_allow_html=True)
        st.title("📚 Instructions")
        st.write("-> Click on the Run button and wait for 20 seconds for execution of the program.")
        st.write("-> Do not move frequently as it may confuse the AI trained model.")
        st.write("-> Look at all the gestures to test all the emotions and to get the required result.")
        # Add more content as needed
        if st.button("Run"):
            st.write("Running the program... Please wait.")
            state['run'] = True
            run_detection()
            st.success("Program executed successfully!")

    def about():
        st.title("📘 About Us")
        st.write("Discover the story behind body language detection. Your journey into AI begins here!")
        # Add more content as needed

    def instructors():
        st.title("👩‍🏫 Instructors")
        st.write("Meet our passionate and experienced instructors. Learn from the best in the industry.")
        st.write("-> Vasundhara")
        st.write("-> Riya")
        st.write("-> Aditya Bajaj")
        # Add more content as needed

    def contact():
        st.title("📧 Contact Us")
        st.write("Have questions or feedback? Contact us at vasundhara25003@gmail.com")
        st.write("OR")
        # Add more content as needed
        st.write("Vasundhara - https://www.linkedin.com/in/vasundhara-vashishtha-0ba939231/")
        st.write("Riya - https://www.linkedin.com/in/riya-arora-212739231/")
        st.write("Aditya Bajaj - https://www.linkedin.com/in/aditya-bajaj-18a14327a")

    def user_feedback():
        st.title("✨ User Feedback")
```

```
    feedback = st.text_area("Share your feedback:")
    if st.button("Submit Feedback"):
        st.success("Feedback submitted successfully! Thank you for your input.")
        # Add logic to handle the feedback (store it, send notifications, etc.)

def main():
    # Add a header space with enhanced text style, "Hey Hi" text, logo on the left, body emoticon on the left, and centered text on the right
    st.markdown(
        """
        <div style='display: flex; justify-content: center; align-items: center; padding: 1em;'>
            <div>
                <p style='color: #ffffff; font-size: 1.2em; font-family: "Roboto", sans-serif;'> </p>
            </div>
            <div>
                <h1 style='color: #001F3F; font-size: 2.5em; font-family: "Roboto", sans-serif;'>BODY✨LANGUAGE✨DETECTION</h1>
                <hr style='border: 5px solid #000000; width: 100%;'>
            </div>
        </div>
        """,
        unsafe_allow_html=True
    )
```

```
# Add logo in the navigation panel at the top
    logo_image = st.sidebar.image("3.jpg", width=100)

# Set a margin to push the logo to the top
    st.sidebar.markdown("<style>div.Widget.row-widget.stRadio > div{flex-direction: column;}</style>", unsafe_allow_html=Tr
    st.sidebar.title("🚀 Navigation")
    pages = ["Home", "About", "Instructors", "Contact", "Feedback"]
    selection = st.sidebar.radio("Go to", pages)

    st.markdown(
        """
        <style>
            body {
                background-color: #001f3f !important;  /* Dark Blue */
                color: #001f3f;
                font-family: "Roboto", sans-serif;
            }
            .sidebar .css-1d01bu2 {
                background-color: #343a40;
                padding: 1em;
                border-radius: 10px;
                color: #ffffff;
                text-align: center;  /* Center align text in the navigation bar */
            }
            .sidebar .css-17vfa1t {
                font-size: 2.5em;
                font-family: "Montserrat", sans-serif;
                margin-top: 1em;
                padding: 0.5em 1em;
                border-radius: 5px;
                text-align: center;  /* Center align text in the navigation bar */
            }
            .sidebar .css-17vfa1t:hover {
                background-color: #007bff;
                color: #ffffff;
                transition: background-color 0.3s, color 0.3s;
```

```
236                 .sidebar .css-17vfa1t:hover {
237                     background-color: #007bff;
238                     color: #ffffff;
239                     transition: background-color 0.3s, color 0.3s;
240                 }
241                 .widget-title {
242                     font-size: 1.5em;
243                     font-family: "Open Sans", sans-serif;
244                     margin-top: 1.5em;
245                 }
246             </style>
247             """,
248             unsafe_allow_html=True,
249         )
250
251         if selection == "Home":
252             homepage()
253         elif selection == "About":
254             about()
255         elif selection == "Instructors":
256             instructors()
257         elif selection == "Contact":
258             contact()
259         elif selection == "Feedback":
260             user_feedback()
261
262     # Run the Streamlit app
263     if __name__ == "__main__":
264         main()
```

Finally, Run the application

This is used to run the application in a local host.

```
262  # Run the Streamlit app
263  if __name__ == "__main__":
264      main()
```

Run the application

- Open the anaconda prompt from the start menu.
- Navigate to the folder where your app.py resides.
- Now type "**streamlit run code.py**" command.
- It will show the local host where your app is running.
- Copy that local host URL and open that URL in the browser. It does navigate me to where you can view your web page.
- Enter the values, click on the predict button and see the result/prediction on the web page.

8. Performance Testing

8.1 Performance Metrics

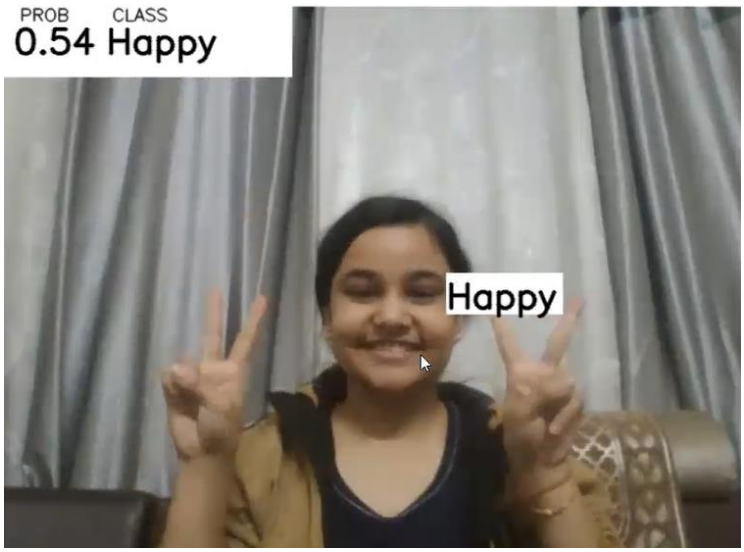
Model Performance Testing:

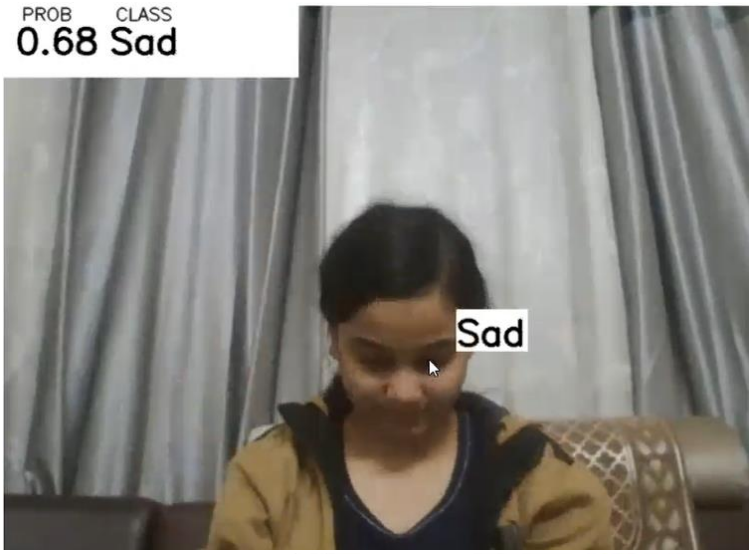
S.No	Parameter	Values	Screenshot
1.	Model Summary	**Total params:1625281 (6.20 MB)** **Trainable params:1625281(6.20 MB)** **Non-trainable params: 0**	Layer (type) Output Shape Param # conv2d_2 (Conv2D) (None, 62, 62, 32) 896 max_pooling2d_2 (MaxPooling2D) (None, 31, 31, 32) 0 conv2d_3 (Conv2D) (None, 29, 29, 64) 18496 max_pooling2d_3 (MaxPooling2D) (None, 14, 14, 64) 0 flatten_1 (Flatten) (None, 12544) 0 dense_4 (Dense) (None, 128) 1605760 dense_5 (Dense) (None, 1) 129 Total params: 1625281 (6.20 MB) Trainable params: 1625281 (6.20 MB) Non-trainable params: 0 (0.00 Byte)

2.	Accuracy	**Training Accuracy -100** **Validation Accuracy - 100**	Epoch 1 - Training Accuracy (lr): 1.0 Epoch 1 - Validation Accuracy (lr): 1.0 Epoch 2 - Training Accuracy (lr): 1.0 Epoch 2 - Validation Accuracy (lr): 1.0 Epoch 3 - Training Accuracy (lr): 1.0 Epoch 3 - Validation Accuracy (lr): 1.0 Epoch 4 - Training Accuracy (lr): 1.0 Epoch 4 - Validation Accuracy (lr): 1.0 Epoch 5 - Training Accuracy (lr): 1.0 Epoch 5 - Validation Accuracy (lr): 1.0 Epoch 6 - Training Accuracy (lr): 1.0 Epoch 6 - Validation Accuracy (lr): 1.0 Epoch 7 - Training Accuracy (lr): 1.0 Epoch 7 - Validation Accuracy (lr): 1.0 Epoch 8 - Training Accuracy (lr): 1.0 Epoch 8 - Validation Accuracy (lr): 1.0 Epoch 9 - Training Accuracy (lr): 1.0 Epoch 9 - Validation Accuracy (lr): 1.0 Epoch 10 - Training Accuracy (lr): 1.0 Epoch 10 - Validation Accuracy (lr): 1.0 Epoch 1 - Training Accuracy (rc): 1.0 Epoch 1 - Validation Accuracy (rc): 1.0 Epoch 2 - Training Accuracy (rc): 1.0 Epoch 2 - Validation Accuracy (rc): 1.0 Epoch 3 - Training Accuracy (rc): 1.0 Epoch 3 - Validation Accuracy (rc): 1.0 Epoch 4 - Training Accuracy (rc): 1.0 Epoch 4 - Validation Accuracy (rc): 1.0 Epoch 5 - Training Accuracy (rc): 1.0
3.	Confidence Score (Only Yolo Projects)	Class Detected - NA Confidence Score - NA	Not Applicable

9. Results

9.1 Output Screenshots




PROB
CLASS
0.68 Sad
Sad


PROB
CLASS
0.41 Victory
Victory


PROB
CLASS
0.48 Angry
Angry

10. Advantages & Disadvantages

10.1 Advantages

Enhanced Customer Interaction:
The system can improve customer-staff interaction by providing real-time insights into customer needs and enabling proactive assistance.

Optimized Staff Allocation:
By analysing customer density in different sections of a store, the system can help optimize staff allocation, ensuring that more staff is available in areas with high customer demand.

Targeted Marketing:
Analysing body language for signs of interest in specific products or sections allows for more targeted marketing and personalized recommendations.

Customer Feedback:
The system can provide valuable insights into customer experiences through implicit feedback, helping businesses make data-driven improvements.

Training Tool:
The system can be used as a training tool for employees, providing examples of positive and negative customer interactions based on body language analysis.

Real-time Insights:
The real-time nature of the system allows for immediate responses to customer needs and enables businesses to adapt quickly to changing situations.

10.2 Disadvantages

1. **Privacy Concerns:**
    - Analysing body language involves capturing and processing visual data, raising privacy concerns. Ensuring compliance with privacy regulations is crucial.
2. **Accuracy Challenges:**
    - The accuracy of body language detection algorithms may be affected by factors such as lighting conditions, camera quality, and the diversity of body types and clothing.
3. **Cultural Sensitivity:**
    - Body language can vary across cultures, and the system may not accurately interpret gestures or expressions in a culturally sensitive manner.
4. **Intrusiveness:**
    - Customers may find the presence of a body language detection system intrusive, leading to concerns about surveillance and a potential negative impact on the shopping experience.

5. **Cost and Implementation Complexity:**
    - Implementing and maintaining a real-time body language detection system requires investment in technology, training, and infrastructure. It may be complex to integrate with existing systems.
6. **Limited Context Understanding:**
    - The system may have limitations in understanding the context of certain body language cues, leading to misinterpretations of customer behaviour.
7. **Ethical Considerations:**
    - There are ethical considerations regarding the use of AI in analysing human behaviour. Ensuring transparent and ethical practices in deploying such systems is crucial.

11. Conclusion

The body language detection project you've implemented using AIML and computer vision offers a promising exploration into the realm of human-computer interaction and emotional analysis. Here's a summary of the potential conclusions and takeaways from the project:

1. Human-Computer Interaction Enhancement:

- The project demonstrates the potential of leveraging body language detection to enhance human-computer interaction. By enabling computers to interpret and respond to users' non-verbal cues, the project opens up possibilities for more intuitive and natural interfaces.

2. Emotion Detection in Real-Time:

- The real-time nature of the body language detection, coupled with the application of machine learning, allows for on-the-fly analysis of users' emotional states. This can have applications in various domains, including virtual communication, entertainment, and user experience design.

3. User-Friendly and Accessible Applications:

- The project contributes to the development of user-friendly and accessible applications by incorporating a visual representation of detected body language. This is particularly relevant in scenarios where traditional input methods may be limiting or impractical.

4. Potential in Education and Training:

- The technology showcased in the project has potential applications in education and training. For example, it could be used to provide feedback on public speaking skills, communication effectiveness, or interpersonal skills, offering users a tool for self-improvement.

5. Cultural Considerations in Body Language:
- The ability to detect body language and emotions may contribute to cross-cultural communication by providing insights into non-verbal cues. Considerations for cultural differences in body language interpretation can enhance the adaptability and effectiveness of the technology.

6. Future Development Opportunities:
- The project identifies several areas for future development, such as improving model accuracy, exploring multi-modal fusion, addressing privacy concerns through edge computing, and refining the application for specific use cases.

7. Educational and Skill-building Tool:
- The project can serve as an educational and skill-building tool for individuals interested in AI, computer vision, and human-computer interaction. The codebase and implementation provide a practical example for learning these concepts.

8. Responsibility and Ethical Use:
- The project emphasizes the importance of considering ethical implications, such as user privacy and potential biases in the detection model. This underscores the responsibility associated with deploying AI technologies and the need for ethical considerations in development.

9. Collaborative Learning Environment:
- The collaborative nature of the project, evident in the team members' LinkedIn profiles and the division of responsibilities, reflects the potential for collaborative learning environments in AI development. This collaborative approach fosters diversity in skills and perspectives.

10. User Feedback and Continuous Improvement:
- The inclusion of a feedback mechanism in the form of a user feedback section demonstrates a commitment to continuous improvement. User feedback can be valuable for refining the application, addressing user concerns, and adapting to evolving user needs.

In conclusion, the body language detection project provides a valuable exploration into the intersection of AIML and human communication. It showcases the potential for creating more intuitive and responsive technology, while also highlighting the importance of ethical considerations and ongoing refinement for real-world applicability.

12. Future Scope

The future scope of body language detection using Artificial Intelligence and Machine Learning (AIML) is vast and holds potential for various applications. Here are some potential avenues for future development and improvement:

1. Enhanced Accuracy and Robustness:

- Continued research and development can focus on improving the accuracy and robustness of body language detection models. This includes refining algorithms, incorporating larger and more diverse datasets, and exploring advanced neural network architectures.

2. Real-time and Low-Latency Detection:

- Advancements in model optimization and hardware acceleration can lead to real-time and low-latency body language detection. This is crucial for applications such as live video conferencing, virtual reality, and human-computer interaction.

3. Multi-Modal Fusion:

- Integrating information from multiple modalities, such as combining body language with facial expressions, voice tone, and text analysis, can provide a more comprehensive understanding of communication. Multi-modal fusion can enhance the overall accuracy and reliability of emotion and intent recognition.

4. Customizable and Personalized Models:

- Developing models that can be fine-tuned or personalized based on individual user characteristics and cultural differences can lead to more accurate and user-specific body language interpretation.

5. Edge Computing for Privacy:

- Implementing body language detection on edge devices can address privacy concerns by processing data locally rather than relying on cloud-based solutions. Edge computing also reduces latency and can be advantageous in real-time applications.

6. Human-Robot Interaction:

- Integrating body language detection into robotics can enhance human-robot interaction. Robots equipped with the ability to understand and respond to human body language can be more intuitive and user-friendly in various settings, including healthcare, customer service, and education.

7. Emotion-aware User Interfaces:

- Applying body language detection in user interfaces to create emotion-aware applications can lead to more engaging and responsive user experiences. This can be particularly useful in the design of virtual assistants, entertainment systems, and educational platforms.

8. Cross-Cultural Adaptability:

- Addressing the challenges associated with cultural variations in body language can improve the adaptability of models across different regions and user demographics. This involves training models on diverse datasets that encompass a wide range of cultural nuances.

9. Long-Term Behaviour Analysis:

- Extending the scope of body language detection to analyse long-term behavioural patterns can have applications in mental health monitoring, stress detection, and personalized well-being assessments.

10. Ethical Considerations and Bias Mitigation:

- Continued emphasis on addressing ethical considerations, ensuring fairness, and mitigating biases in body language detection models is crucial. Research should focus on reducing algorithmic biases and ensuring that the technology is deployed responsibly.

11. Collaboration with Other AI Domains:

- Collaborating with other AI domains, such as natural language processing, computer vision, and sentiment analysis, can lead to more holistic and integrated solutions for understanding human communication.

As technology evolves, the intersection of AIML and body language detection holds significant promise in improving human-computer interaction, augmenting virtual communication, and enhancing various applications across different industries.

13. Appendix

13.1 Source Code

Jupyter Code:

1. Import Dependencies

In [1]:
```
import mediapipe as mp # Import mediapipe
import cv2 # Import opencv
```

In [2]:
```
mp_drawing = mp.solutions.drawing_utils # Drawing helpers
mp_holistic = mp.solutions.holistic # Mediapipe Solutions
```

2. Make Some Detections

In [3]:
```
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_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 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()
```

In [4]:
```
results.face_landmarks.landmark[0].visibility
```

3. Capture Landmarks & Export to CSV

In [5]:
```
import csv
import os
import numpy as np
```

In [6]:
```
num_coords = len(results.pose_landmarks.landmark)+len(results.face_landmarks.landmark)
num_coords
```

Out[6]:
```
501
```

In [7]:
```
landmarks = ['class']
for val in range(1, num_coords+1):
    landmarks += ['x{}'.format(val), 'y{}'.format(val), 'z{}'.format(val), 'v{}'.format(val)]
```

In [8]:
```
landmarks
```

Out[8]:
```
['class',
 'x1',
 'y1',
```

In [9]:
```
with open('coords.csv', mode='w', newline='') as f:
    csv_writer = csv.writer(f, delimiter=',', quotechar='"', quoting=csv.QUOTE_MINIMAL)
    csv_writer.writerow(landmarks)
```

In [12]:
```
class_name = "Sad"
```

In [13]:
```
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_CONTOURS,
```

```
    # 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 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
```

```
        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()
```

4.Train Custom Model Using Scikit Learn

4.1 Read in Collected Data and Process

In [14]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
```

In [15]:
```
df = pd.read_csv('coords.csv')
```

In [16]:
```
df.head()
```

Out[16]:

	class	x1	y1	z1	v1	x2	y2	z2	v2	x3	...	z499	v499	
0	Happy	0.497000	0.699944	-0.637015	0.999998	0.520749	0.646792	-0.589081	0.999997	0.536742	...	0.003716	0.0	0
1	Happy	0.496714	0.698728	-0.858924	0.999995	0.520536	0.643332	-0.791215	0.999995	0.536448	...	0.004037	0.0	0
2	Happy	0.496537	0.707593	-0.867257	0.999974	0.520551	0.653116	-0.794210	0.999981	0.536445	...	0.004144	0.0	0
3	Happy	0.496718	0.706713	-0.862160	0.999974	0.520537	0.651828	-0.789188	0.999981	0.536278	...	0.004834	0.0	0
4	Happy	0.498438	0.706539	-0.822684	0.999975	0.520959	0.651697	-0.777824	0.999981	0.536469	...	0.006039	0.0	0

5 rows × 2005 columns

In [17]:
```
df.tail()
```

In [18]:
```
df[df['class']=='Sad']
```

Out[18]:

	class	x1	y1	z1	v1	x2	y2	z2	v2	x3	...	z499
17	Sad	0.513574	0.704613	-0.834415	0.999648	0.534595	0.643326	-0.788578	0.999146	0.550543	...	-0.005062
18	Sad	0.514869	0.702935	-0.949090	0.999667	0.535440	0.637503	-0.889859	0.999193	0.550787	...	-0.005845
19	Sad	0.516208	0.699086	-1.010334	0.999678	0.536373	0.629654	-0.947225	0.999220	0.551298	...	-0.005784
20	Sad	0.518131	0.699184	-0.992126	0.999700	0.538022	0.630038	-0.928450	0.999270	0.552951	...	-0.004765
21	Sad	0.519139	0.697457	-0.972304	0.999722	0.538567	0.627680	-0.910417	0.999323	0.553294	...	-0.005892
22	Sad	0.519124	0.695891	-0.774944	0.999739	0.538602	0.625739	-0.724947	0.999361	0.553273	...	-0.005685
23	Sad	0.519112	0.695486	-0.951461	0.999755	0.538621	0.625241	-0.888135	0.999400	0.553254	...	-0.005568
24	Sad	0.519355	0.695339	-1.044269	0.999766	0.538836	0.624522	-0.980892	0.999427	0.553387	...	-0.005773
25	Sad	0.519633	0.694649	-1.029050	0.999782	0.538922	0.623808	-0.966856	0.999469	0.553408	...	-0.005933

In [19]:
```
X = df.drop('class', axis=1) # features
y = df['class'] # target value
```

In [20]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1234)
```

In [21]:
```
y_test
```

Out[21]:
```
8     Happy
36      Sad
14    Happy
1     Happy
4     Happy
29      Sad
20      Sad
17      Sad
13    Happy
21      Sad
0     Happy
7     Happy
Name: class, dtype: object
```

4.2 Train Machine Learning Classification Model

In [22]:
```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

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

In [23]:
```
pipelines = {
    'lr': make_pipeline(StandardScaler(), LogisticRegression(max_iter=10000)),
    'rc': make_pipeline(StandardScaler(), RidgeClassifier()),
    'rf': make_pipeline(StandardScaler(), RandomForestClassifier()),
    'gb': make_pipeline(StandardScaler(), GradientBoostingClassifier()),
}
```

In [24]:
```
fit_models = {}
for algo, pipeline in pipelines.items():
    model = pipeline.fit(X_train.values, y_train)
    fit_models[algo] = model
```

In [25]:
```
fit_models
```

Out[25]:
```
{'lr': Pipeline(steps=[('standardscaler', StandardScaler()),
                       ('logisticregression', LogisticRegression(max_iter=10000))]),
 'rc': Pipeline(steps=[('standardscaler', StandardScaler()),
                       ('ridgeclassifier', RidgeClassifier())]),
 'rf': Pipeline(steps=[('standardscaler', StandardScaler()),
                       ('randomforestclassifier', RandomForestClassifier())]),
```

In [26]:
```
fit_models['rc'].predict(X_test.values)
```

Out[26]:
```
array(['Happy', 'Sad', 'Happy', 'Happy', 'Happy', 'Sad', 'Sad', 'Sad',
       'Happy', 'Sad', 'Happy', 'Happy'], dtype='<U5')
```

4.3 Evaluate and Serialize Model

In [27]:
```
from sklearn.metrics import accuracy_score # Accuracy metrics
import pickle
```

In [28]:
```
for algo, model in fit_models.items():
    yhat = model.predict(X_test.values)
    print(algo, accuracy_score(y_test, yhat))
```

```
lr 1.0
rc 1.0
rf 1.0
gb 1.0
```

In [29]:
```
fit_models['rf'].predict(X_test.values)
```

Out[29]:
```
array(['Happy', 'Sad', 'Happy', 'Happy', 'Happy', 'Sad', 'Sad', 'Sad',
       'Happy', 'Sad', 'Happy', 'Happy'], dtype=object)
```

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

Out[30]:
```
8     Happy
```

4.4 Training and Validation Accuracy

In [31]:
```
pipelines = {
    'lr': make_pipeline(StandardScaler(with_mean=False), LogisticRegression(max_iter=1000)),
    'rc': make_pipeline(StandardScaler(with_mean=False), RidgeClassifier()),
    'rf': make_pipeline(StandardScaler(with_mean=False), RandomForestClassifier()),
    'gb': make_pipeline(StandardScaler(with_mean=False), GradientBoostingClassifier()),
}
```

In [32]:
```
from sklearn.metrics import accuracy_score

# Split the data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

num_epochs = 10
fit_models = {algo: [] for algo in pipelines}

for algo, pipeline in pipelines.items():
    model = None  # Initialize model for each algorithm
```

```
for epoch in range(num_epochs):
    model = pipeline.fit(X_train_scaled, y_train)

    # Training accuracy
    y_train_pred = model.predict(X_train_scaled)
    train_accuracy = accuracy_score(y_train, y_train_pred)
    print(f'Epoch {epoch + 1} - Training Accuracy ({algo}): {train_accuracy}')

    # Validation accuracy
    y_val_pred = model.predict(X_val_scaled)
    validation_accuracy = accuracy_score(y_val, y_val_pred)
    print(f'Epoch {epoch + 1} - Validation Accuracy ({algo}): {validation_accuracy}')

    # Store the model for each epoch
    fit_models[algo].append(model)
```

5. Make Detections with Model

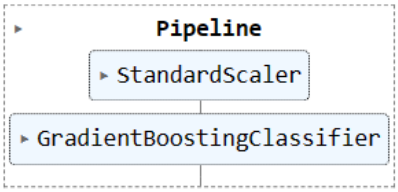
In [33]:
```
with open('body_language.pkl', 'wb') as f:
    pickle.dump(fit_models['rf'], f)
```

In [34]:
```
desired_epoch = 0  # Change this to the epoch you want to load
with open('body_language.pkl', 'rb') as f:
    loaded_models = pickle.load(f)

# Access the desired model from the list
loaded_model = loaded_models[desired_epoch]
```

In [35]:
```
model
```

Out[35]:



In [36]:
```
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_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)
                                  )
```

```
        # 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)
```

```
            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), 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()
```

```
Happy [0.7184277 0.2815723]
Happy [0.7184277 0.2815723]
Happy [0.66205232 0.33794768]
Happy [0.7184277 0.2815723]
Happy [0.66205232 0.33794768]
Happy [0.66205232 0.33794768]
Happy [0.51524393 0.48475607]
Happy [0.51524393 0.48475607]
```

In [37]:
```
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[37]: (386, 313)

6. Model summary

In [38]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
```

In [39]:
```
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(64, 64, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

In [40]:
```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

In [40]:
```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

In [41]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 62, 62, 32)        896

 max_pooling2d (MaxPooling2  (None, 31, 31, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 29, 29, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 14, 14, 64)        0
 g2D)

 flatten (Flatten)           (None, 12544)             0

 dense (Dense)               (None, 128)               1605760

 dense_1 (Dense)             (None, 1)                 129

=================================================================
Total params: 1625281 (6.20 MB)
Trainable params: 1625281 (6.20 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Python Code (Web Integration):

```
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 time

# Create a dictionary to store the state
state = {'run': False, 'end_button_clicked': False}

# Load the pre-trained model
with open('body_language.pkl', 'rb') as f:
    model = pickle.load(f)

# Setup MediaPipe
mp_drawing = mp.solutions.drawing_utils
mp_holistic = mp.solutions.holistic

def run_detection():
    # Display the webcam feed and body language detection
    camera = cv2.VideoCapture(0)
    FRAME_WINDOW = st.image([])
    end_button_placeholder = st.empty()
    end_button_displayed = False

    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 = False

            image = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
            image[:, :, 1] = image[:, :, 1] * 1.0  # Increase saturation
            image = cv2.cvtColor(image, cv2.COLOR_HSV2RGB)
            # Adjust brightness and contrast
            alpha = 1.5  # Contrast control (1.0 means no change)
            beta = 25  # Brightness control (0 means no change)
```

```
            adjusted_image = cv2.convertScaleAbs(image, alpha=alpha, beta=beta)

            # 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

            # 1. Draw face landmarks
            # 1. Draw face landmarks without grid
            # 1. Draw face landmarks without dots
            if results.face_landmarks is not None:
                pass  # Do not draw anything

            # 2. Right hand without dots
            if results.right_hand_landmarks is not None:
                pass  # Do not draw anything

            # 3. Left hand without dots
            if results.left_hand_landmarks is not None:
                pass  # Do not draw anything

            # 4. Pose Detections without dots
            if results.pose_landmarks is not None:
                pass  # Do not draw anything

            # 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())

                # Concatenate 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),
                              (255, 255, 255),  # White box
                              -1)

                cv2.putText(image, body_language_class, coords,
                            cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 0), 2,
cv2.LINE_AA)

                # Get status box
                cv2.rectangle(image, (0, 0), (250, 60), (255, 255, 255), -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, (0, 0, 0),
                            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, (0, 0, 0), 2,
cv2.LINE_AA)
            except:
                pass

            FRAME_WINDOW.image(image, channels="RGB", output_format="JPEG")

            if state['run'] and not state['end_button_clicked'] and not
end_button_displayed:
                end_button_clicked = end_button_placeholder.button("End",
key="end_button")
                state['end_button_clicked'] = end_button_clicked
                end_button_displayed = True

            if state['end_button_clicked']:
                break

    camera.release()
    cv2.destroyAllWindows()

def homepage():
    st.image("1.jpg", width=700)
    st.markdown("<hr style='border: 5px solid #000000; width: 100%;'>",
unsafe_allow_html=True)
    st.title("▧ Know About Emotion Detection ")
    st.image("2.png",width=900)
    st.markdown("<hr style='border: 5px solid #000000; width: 100%;'>",
unsafe_allow_html=True)
    st.title("🌐 Identifying Body Language ")
    st.write("--> Just click on 'Run' button to let us explore your moods and
emotions.")


    st.markdown("<hr style='border: 5px solid #000000; width: 100%;'>",
unsafe_allow_html=True)
    st.title("📄Instructions")
    st.write("-> Click on the Run button and wait for 20 seconds for execution of
the program.")
    st.write("-> Do not move frequently as it may confuse the AI trained model.")
    st.write("-> Look at all the gestures to test all the emotions and to get the
required result.")
    # Add more content as needed
    if st.button("Run"):
        st.write("Running the program... Please wait.")
```

```
        state['run'] = True
        run_detection()
        st.success("Program executed successfully!")

def about():
    st.title("▧ About Us")
    st.write("Discover the story behind body language detection. Your journey
into AI begins here!")
    # Add more content as needed

def instructors():
    st.title("🏫 Instructors")
    st.write("Meet our passionate and experienced instructors. Learn from the
best in the industry.")
    st.write("-> Vasundhara")
    st.write("-> Riya")
    st.write("-> Aditya Bajaj")
    # Add more content as needed

def contact():
    st.title("📧 Contact Us")
    st.write("Have questions or feedback? Contact us at
vasundhara25003@gmail.com")
    st.write("OR")
    # Add more content as needed
    st.write("Vasundhara - https://www.linkedin.com/in/vasundhara-vashishtha-
0ba939231/")
    st.write("Riya - https://www.linkedin.com/in/riya-arora-212739231/")
    st.write("Aditya Bajaj - https://www.linkedin.com/in/aditya-bajaj-18a14327a")

def user_feedback():
    st.title("✨ User Feedback")
    feedback = st.text_area("Share your feedback:")
    if st.button("Submit Feedback"):
        st.success("Feedback submitted successfully! Thank you for your input.")
        # Add logic to handle the feedback (store it, send notifications, etc.)

def main():
    # Add a header space with enhanced text style, "Hey Hi" text, logo on the
left, body emoticon on the left, and centered text on the right
    st.markdown(
        """
        <div style='display: flex; justify-content: center; align-items: center;
padding: 1em;'>
```

```
            <div>
                <p style='color: #ffffff; font-size: 1.2em; font-family:
"Roboto", sans-serif;'> </p>
            </div>
            <div>
                <h1 style='color: #001F3F; font-size: 2.5em; font-family:
"Roboto", sans-serif;'>BODY✨LANGUAGE✨DETECTION</h1>
                <hr style='border: 5px solid #000000; width: 100%;'>
            </div>
        </div>
        """,
        unsafe_allow_html=True
    )

# Add logo in the navigation panel at the top
    logo_image = st.sidebar.image("3.jpg", width=100)

# Set a margin to push the logo to the top
    st.sidebar.markdown("<style>div.Widget.row-widget.stRadio > div{flex-
direction: column;}</style>", unsafe_allow_html=True)
    st.sidebar.title("🚀 Navigation")
    pages = ["Home", "About", "Instructors", "Contact", "Feedback"]
    selection = st.sidebar.radio("Go to", pages)

    st.markdown(
        """
        <style>
            body {
                background-color: #001f3f !important;  /* Dark Blue */
                color: #001f3f;
                font-family: "Roboto", sans-serif;
            }
            .sidebar .css-1d01bu2 {
                background-color: #343a40;
                padding: 1em;
                border-radius: 10px;
                color: #ffffff;
                text-align: center;  /* Center align text in the navigation bar
*/
            }
            .sidebar .css-17vfa1t {
                font-size: 2.5em;
                font-family: "Montserrat", sans-serif;
                margin-top: 1em;
```

```
                padding: 0.5em 1em;
                border-radius: 5px;
                text-align: center;  /* Center align text in the navigation bar
*/
            }
            .sidebar .css-17vfa1t:hover {
                background-color: #007bff;
                color: #ffffff;
                transition: background-color 0.3s, color 0.3s;
            }
            .widget-title {
                font-size: 1.5em;
                font-family: "Open Sans", sans-serif;
                margin-top: 1.5em;
            }
        </style>
        """,
        unsafe_allow_html=True,
    )

    if selection == "Home":
        homepage()
    elif selection == "About":
        about()
    elif selection == "Instructors":
        instructors()
    elif selection == "Contact":
        contact()
    elif selection == "Feedback":
        user_feedback()

# Run the Streamlit app
if __name__ == "__main__":
    main()
```

13.2 GitHub & Project Demo Link

GitHub Link:

https://github.com/smartinternz02/SI-GuidedProject-614005-1700032595

Project Demo Link:

https://drive.google.com/file/d/1e2KM6PUB9A6ZGfAf2cXGZXj2xVrOxc0j/view?usp=drivesdk