

End-to-end machine learning project to detect body language.

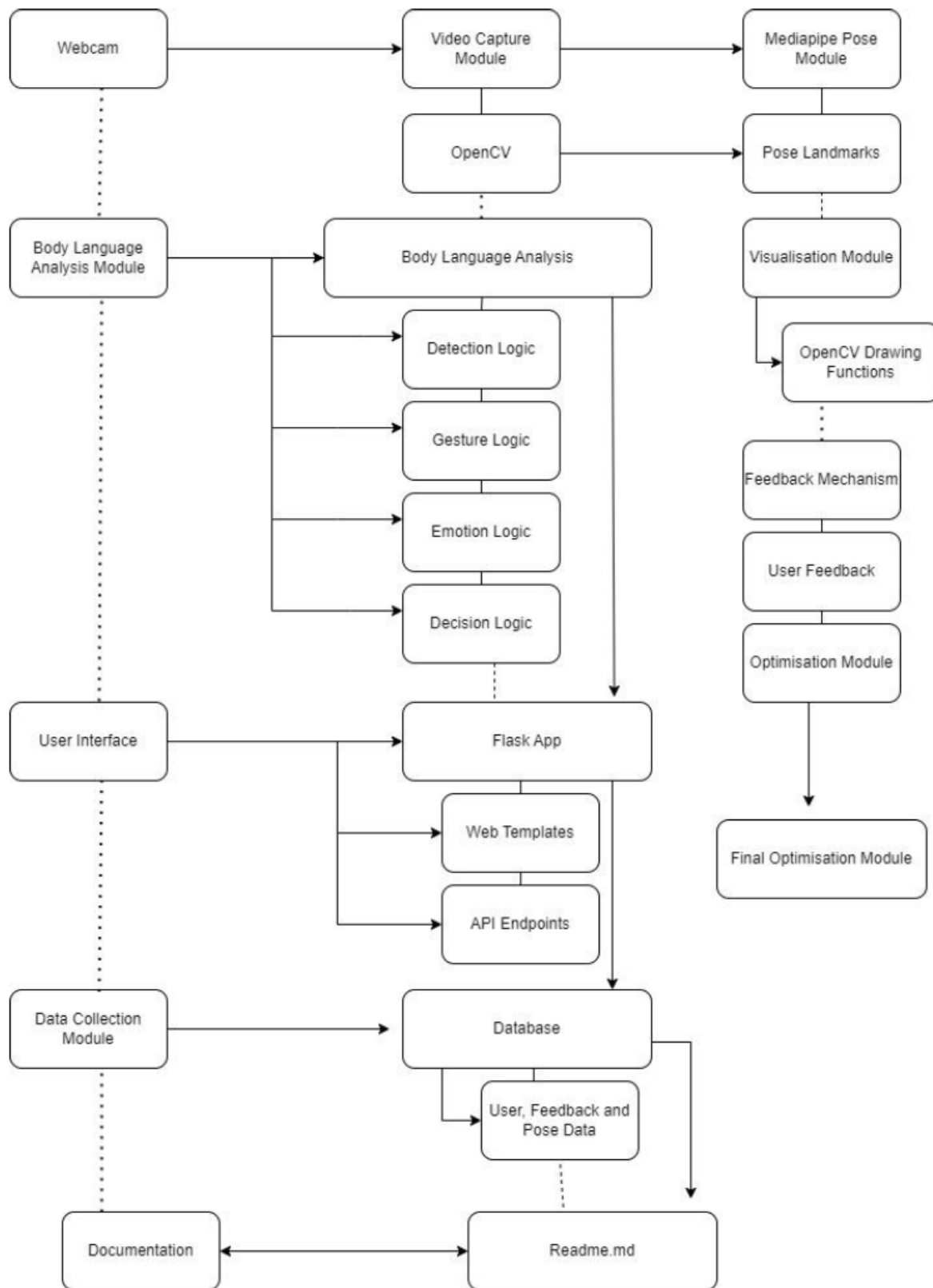
The objective of this project is to develop an end-to-end machine learning solution to detect body language and classify into four categories: happy, sad, victorious and fight. The proposed solution involves the use of machine learning algorithms like Logistic Regression, Ridge Classifier to predict the accurate output.

Detecting body language using machine learning can offer several benefits:

1. **Enhanced Communication Understanding:** By analyzing 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.

Let us look at the Technical Architecture of the project.

Technical Architecture:



Project Flow:

- The user interacts with the UI to upload the file.
- Uploaded input is analyzed by the model which is integrated/developed by you.
- Once the model analyzes the input, the prediction is showcased on the UI. To accomplish this, we have to complete all the activities listed below,
 - Define Problem / Problem Understanding
 - Specify the business problem
 - Business requirements
 - Literature Survey.
 - Social or Business Impact.
 - Data Collection & Preparation
 - Collect the dataset
 - Data Preparation
 - Exploratory Data Analysis
 - Descriptive statistical
 - Visual Analysis
 - Model Building
 - Building a model.
 - Training the model.
 - Testing the model
 - Model Deployment
 - Save the best model
 - Integrate with Web Framework
 - Project Demonstration & Documentation
 - Record explanation Video for project end to end solution
 - Project Documentation-Step by step project development procedure

Prior Knowledge:

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

- VS Code :
 - Refer to the link below to download VS Code.
 - Link : <https://code.visualstudio.com/download>
- Create Project:
 - Open VS Code.
 - Create a Folder and name it "AI_Body_Language_Detector" .
- Machine Learning Concepts
 - Machine Learning:
<https://towardsdatascience.com/machine-learning-an-introduction-23b84d51e6d0>
 - ML Models :
<https://towardsdatascience.com/all-machine-learning-models-explained-in-6-minutes-9fe30ff6776a>
- Web concepts:
 - Get the gist on streamlit :
<https://www.geeksforgeeks.org/a-beginners-guide-to-streamlit/>
- Mediapipe:
 - About Mediapipe :
<https://www.geeksforgeeks.org/python-facial-and-hand-recognition-using-mediapipe-holistic/>

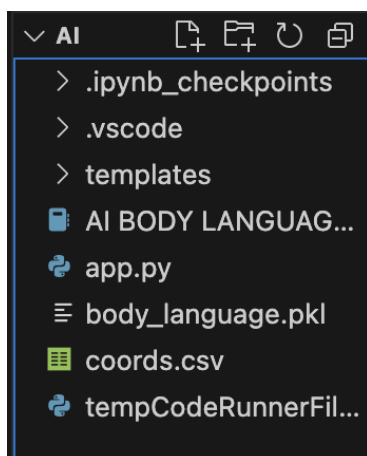
Project Objectives:

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 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

Refer Project Description

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: Literature Survey (Student Will Write)

A literature survey would involve researching and reviewing existing studies, articles, and other publications on the topic of project. The survey would aim to gather information on current systems, their strengths and weaknesses, and any gaps in knowledge that the project could address. The literature survey would also look at the methods and techniques used in previous projects, and any relevant data or findings that could inform the design and implementation of the current project.

Activity 4: 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. So, this section allows you to create the required dataset.

Activity 1: Make some detections.

First we will import some required packages

```
import mediapipe as mp
```

```
import cv2
```

The MediaPipe Holistic Landmarker task lets you combine components of the pose, face, and hand landmarks to create a complete landmarker for the human body.

Let us initialize the holistics

```
mp_drawing = mp.solutions.drawing_utils #Drawing Helpers  
mp_holistic = mp.solutions.holistic #Mediapipe Solutions
```

Now we'll make some detections of face, hands and pose.

The below code will detect and draw lines on you hands, face and your pose.

```

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.FACE_CONNECTIONS,
                                  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()

```

Once you run this code, your webcam will get activated and you can see the detections of your face, hands and pose.

Activity 1.1: Collecting data for Happy mood

Initialize a variable "class_name" equals "Happy".

```
class_name = "Happy"
```

Now let us write a code which allows us to detect our body landmarks and save them into a .csv file.

For now we'll collect the data for happy mode. So, once you run the code your webcam will get activated and you've to record your happy face for 10-15 seconds and close the window.


```

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.FACE_CONNECTIONS,
                                  mp_drawing.DrawingSpec(color=(80,110,10), thickness=1, circle_radius=1),
                                  mp_drawing.DrawingSpec(color=(80,256,121), thickness=1, circle_radius=1)
                                  )

        # 2. Right hand
        mp_drawing.draw_landmarks(image, results.right_hand_landmarks, mp_holistic.HAND_CONNECTIONS,
                                  mp_drawing.DrawingSpec(color=(80,22,10), thickness=2, circle_radius=4),
                                  mp_drawing.DrawingSpec(color=(80,44,121), thickness=2, circle_radius=2)
                                  )

        # 3. Left Hand
        mp_drawing.draw_landmarks(image, results.left_hand_landmarks, mp_holistic.HAND_CONNECTIONS,
                                  mp_drawing.DrawingSpec(color=(121,22,76), thickness=2, circle_radius=4),
                                  mp_drawing.DrawingSpec(color=(121,44,250), thickness=2, circle_radius=2)
                                  )

        # 4. Pose Detections
        mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_holistic.POSE_CONNECTIONS,
                                  mp_drawing.DrawingSpec(color=(245,117,66), thickness=2, circle_radius=4),
                                  mp_drawing.DrawingSpec(color=(245,66,230), thickness=2, circle_radius=2)
                                  )

        # 5. Export Co-ordinates

        try:
            # Extract Pose landmarks
            pose = results.pose_landmarks.landmark
            pose_row = list(np.array([[landmark.x, landmark.y, landmark.z, landmark.visibility] for landmark in pose]).flatten())

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

            # Concat rows
            row = pose_row+face_row

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

            # Export to CSV
            with open('/Users/rishi/BTECH/Programming/My_Projects/AI_Body_Language_Detector/Dataset/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()

```

Activity 1.2: Collecting data for Sad mood.

Initialize a variable "class_name" equals "Sad".

```
class_name = "Sad"
```

Now let us write a code which allows us to detect our body landmarks and save them into a .csv file.

For now we'll collect the data for sad mode. So, once you run the code your webcam will get activated and you've to record your sad face for 10-15 seconds and close the window.

```

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.FACE_CONNECTIONS,
                                  mp_drawing.DrawingSpec(color=(80,110,10), thickness=1, circle_radius=1),
                                  mp_drawing.DrawingSpec(color=(80,256,121), thickness=1, circle_radius=1)
                                  )

        # 2. Right hand
        mp_drawing.draw_landmarks(image, results.right_hand_landmarks, mp_holistic.HAND_CONNECTIONS,
                                  mp_drawing.DrawingSpec(color=(80,22,10), thickness=2, circle_radius=4),
                                  mp_drawing.DrawingSpec(color=(80,44,121), thickness=2, circle_radius=2)
                                  )

        # 3. Left Hand
        mp_drawing.draw_landmarks(image, results.left_hand_landmarks, mp_holistic.HAND_CONNECTIONS,
                                  mp_drawing.DrawingSpec(color=(121,22,76), thickness=2, circle_radius=4),
                                  mp_drawing.DrawingSpec(color=(121,44,250), thickness=2, circle_radius=2)
                                  )

        # 4. Pose Detections
        mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_holistic.POSE_CONNECTIONS,
                                  mp_drawing.DrawingSpec(color=(245,117,66), thickness=2, circle_radius=4),
                                  mp_drawing.DrawingSpec(color=(245,66,230), thickness=2, circle_radius=2)
                                  )

        # 5. Export Co-ordinates

        try:
            # Extract Pose landmarks
            pose = results.pose_landmarks.landmark
            pose_row = list(np.array([[landmark.x, landmark.y, landmark.z, landmark.visibility] for landmark in pose]).flatten())

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

            # Concat rows
            row = pose_row+face_row

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

            # Export to CSV
            with open('/Users/rishi/BTECH/Programming/My_Projects/AI_Body_Language_Detector/Dataset/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)

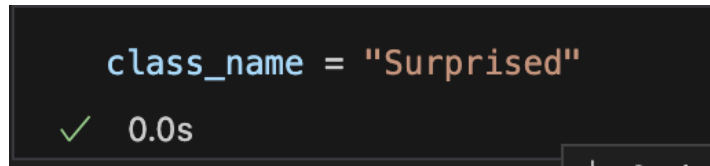
        if cv2.waitKey(10) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()

```

Activity 1.3: Collecting data for Surprised mood.

Initialize a variable "class_name" equals "Surprised".

A screenshot of a code editor with a dark background. The code `class_name = "Surprised"` is written in a light blue font. Below the code, there is a green checkmark icon followed by the text "0.0s" in a light blue font, indicating a successful execution or timing measurement.

Now let us write a code which allows us to detect our body landmarks and save them into a .csv file.

For now we'll collect the data for victorious mode. So, once you run the code your webcam will get activated and you've to record your victorious body language for 10-15 seconds and close the window.

Victorious Body Language : Put your two hands up and hold your fists. Just how Tim Cook is posing.

```

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.FACE_CONNECTIONS,
                                  mp_drawing.DrawingSpec(color=(80,110,10), thickness=1, circle_radius=1),
                                  mp_drawing.DrawingSpec(color=(80,256,121), thickness=1, circle_radius=1)
                                  )

        # 2. Right hand
        mp_drawing.draw_landmarks(image, results.right_hand_landmarks, mp_holistic.HAND_CONNECTIONS,
                                  mp_drawing.DrawingSpec(color=(80,22,10), thickness=2, circle_radius=4),
                                  mp_drawing.DrawingSpec(color=(80,44,121), thickness=2, circle_radius=2)
                                  )

        # 3. Left Hand
        mp_drawing.draw_landmarks(image, results.left_hand_landmarks, mp_holistic.HAND_CONNECTIONS,
                                  mp_drawing.DrawingSpec(color=(121,22,76), thickness=2, circle_radius=4),
                                  mp_drawing.DrawingSpec(color=(121,44,250), thickness=2, circle_radius=2)
                                  )

        # 4. Pose Detections
        mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_holistic.POSE_CONNECTIONS,
                                  mp_drawing.DrawingSpec(color=(245,117,66), thickness=2, circle_radius=4),
                                  mp_drawing.DrawingSpec(color=(245,66,230), thickness=2, circle_radius=2)
                                  )

        # 5. Export Co-ordinates

        try:
            # Extract Pose landmarks
            pose = results.pose_landmarks.landmark
            pose_row = list(np.array([[landmark.x, landmark.y, landmark.z, landmark.visibility] for landmark in pose]).flatten())

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

            # Concat rows
            row = pose_row+face_row

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

            # Export to CSV
            with open('/Users/rishi/BTECH/Programming/My_Projects/AI_Body_Language_Detector/Dataset/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()

```

1.2: Read the Collected Data

- **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('coords.csv')
```

✓ 0.1s

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.iloc[:, 1:]
y = df.iloc[:, 0]
```

✓ 0.0s

Milestone 3: Exploratory Data Analysis

Activity 1: Descriptive statistical

Descriptive analysis is to study the basic features of data with the statistical process. With this describe function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features. Which are not suitable for our dataset.

As our data consists of only coordinates, There is no unnecessary data which can be eliminated.

Activity 2: Splitting data into train and test and validation sets

Now let's split the Dataset into train and test sets.
The split 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=0)
✓ 0.0s
```

Milestone 4: Model Building

Activity 1: Importing necessary libraries

We'll import some necessary libraries using the following code

```
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, RidgeClassifier, 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 5: 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
```

We will take the best fit model which can be 'lr', 'rf' or 'gb'
'rc' could be the case of over-fitting.

Activity 3: Save the model

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

✓ 0.0s

This will allow us to save the .pkl file of the best model.

Activity 4 : Testing the model

First we have to load the model using pickle

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

✓ 0.0s

The following code is final output which allows us to predict the user's body language.

Once you run the code your webcam will get activated, the predictions and the confidence of prediction is shown on the screen.

```

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.FACE_CONNECTIONS,
                                  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:
            pose = results.pose_landmarks.landmark # Extract Pose landmarks
            pose_row = list(np.array([[landmark.x, landmark.y, landmark.z, landmark.visibility] for landmark in pose]).flatten())
            face = results.face_landmarks.landmark # Extract Face landmarks
            face_row = list(np.array([[landmark.x, landmark.y, landmark.z, landmark.visibility] for landmark in face]).flatten())
            # Concat rows
            row = pose_row+face_row
            # Make Detections
            X = pd.DataFrame([row])
            body_language_class = model.predict(X)[0]
            body_language_prob = model.predict_proba(X)[0]
            print(body_language_class, body_language_prob)
            # Grab ear coords
            coords = tuple(np.multiply(
                np.array(
                    (results.pose_landmarks.landmark[mp_holistic.PoseLandmark.LEFT_EAR].x,
                     results.pose_landmarks.landmark[mp_holistic.PoseLandmark.LEFT_EAR].y))
                , [640,480]).astype(int))
            cv2.rectangle(image,
                          (coords[0], coords[1]+5),
                          (coords[0]+len(body_language_class)*20, coords[1]-30),
                          (245, 117, 16), -1)
            cv2.putText(image, body_language_class, coords,
                        cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)
            # Get status box
            cv2.rectangle(image, (0,0), (250, 60), (245, 117, 16), -1)
            # Display Class
            cv2.putText(image, 'CLASS'
                        , (95,12), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 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()

```

Activity 4: Integrate with Web Framework

In this section, we will be building a web application that is integrated to the model we built.

We will be using the streamlit package for our website development.

Streamlit is a free and open-source framework to rapidly build and share beautiful machine learning and data science web apps.

Activity 4.1: Create a web.py file and import necessary packages:

NOTE : If you get any error like “streamlit not found”. You have to make sure streamlit is installed on your system.

- Running the command “pip install streamlit” would do that job for you.

Activity 4.2: Loading the model and creating a frame window to use OpenCV:

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

```
st.cache(allow_output_mutation=True)

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

st.write("""# Body Language Detection""")
mp_drawing = mp.solutions.drawing_utils #Drawing Helpers
mp_holistic = mp.solutions.holistic #Mediapipe Solutions
run = st.checkbox('Run')
FRAME_WINDOW = st.image([])
camera = cv2.VideoCapture(0)
```

Activity 4.3: Accepting the input from user and prediction

```

while run:
    with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:
        while camera.isOpened():
            ret, frame = camera.read()
            # Recolor Feed
            image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            image.flags.writeable = False
            # Make Detections
            results = holistic.process(image)
            # face_landmarks, pose_landmarks, left_hand_landmarks, right_hand_landmarks
            # Recolor image back to BGR for rendering
            image.flags.writeable = True
            image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
            # 1. Draw face landmarks
            mp_drawing.draw_landmarks(image, results.face_landmarks, mp_holistic.FACEMESH_CONTOURS,
                                      mp_drawing.DrawingSpec(color=(80,110,10), thickness=1, circle_radius=1),
                                      mp_drawing.DrawingSpec(color=(80,256,121), thickness=1, circle_radius=1)
                                      )
            # 2. Right hand
            mp_drawing.draw_landmarks(image, results.right_hand_landmarks, mp_holistic.HAND_CONNECTIONS,
                                      mp_drawing.DrawingSpec(color=(80,22,10), thickness=2, circle_radius=4),
                                      mp_drawing.DrawingSpec(color=(80,44,121), thickness=2, circle_radius=2)
                                      )
            # 3. Left Hand
            mp_drawing.draw_landmarks(image, results.left_hand_landmarks, mp_holistic.HAND_CONNECTIONS,
                                      mp_drawing.DrawingSpec(color=(121,22,76), thickness=2, circle_radius=4),
                                      mp_drawing.DrawingSpec(color=(121,44,250), thickness=2, circle_radius=2)
                                      )
            # 4. Pose 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:
                pose = results.pose_landmarks.landmark # Extract Pose landmarks
                pose_row = list(np.array([[landmark.x, landmark.y, landmark.z, landmark.visibility] for landmark in pose]).flatten())
                face = results.face_landmarks.landmark # Extract Face landmarks
                face_row = list(np.array([[landmark.x, landmark.y, landmark.z, landmark.visibility] for landmark in face]).flatten())
                # Concat rows
                row = pose_row+face_row
                # Make Detections
                X = pd.DataFrame([row])
                body_language_class = model.predict(X)[0]
                body_language_prob = model.predict_proba(X)[0]
                print(body_language_class, body_language_prob)
                # Grab ear coords
                coords = tuple(np.multiply(
                    np.array(
                        (results.pose_landmarks.landmark[mp_holistic.PoseLandmark.LEFT_EAR].x,
                         results.pose_landmarks.landmark[mp_holistic.PoseLandmark.LEFT_EAR].y))
                    , [640,480]).astype(int))

                cv2.rectangle(image,
                              (coords[0], coords[1]+5),
                              (coords[0]+len(body_language_class)*20, coords[1]-30),
                              (245, 117, 16), -1)
                cv2.putText(image, body_language_class, coords,
                            cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)
                # Get status box
                cv2.rectangle(image, (0,0), (250, 60), (245, 117, 16), -1)
                # Display Class
                cv2.putText(image, 'CLASS'
                              , (95,12), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 1, cv2.LINE_AA)
                cv2.putText(image, body_language_class.split(' ')[0]
                              , (90,40), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)
                # Display Probability
                cv2.putText(image, 'PROB'
                              , (15,12), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 1, cv2.LINE_AA)
                cv2.putText(image, str(round(body_language_prob[np.argmax(body_language_prob)],2))
                              , (10,40), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)
            except:
                pass
            FRAME_WINDOW.image(image)
            if cv2.waitKey(10) & 0xFF == ord('q'):
                break
        camera.release()
        cv2.destroyAllWindows()
    else:
        st.write('Stopped')

```


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

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

- “streamlit run web.py”

You can now view your Streamlit app in your browser.

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

Network URL: `http://192.168.1.11:8501`

For better performance, install the Watchdog module:

```
$ xcode-select --install
```

```
$ pip install watchdog
```

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

HOW DOES YOUR WEBSITE LOOK LIKE ?

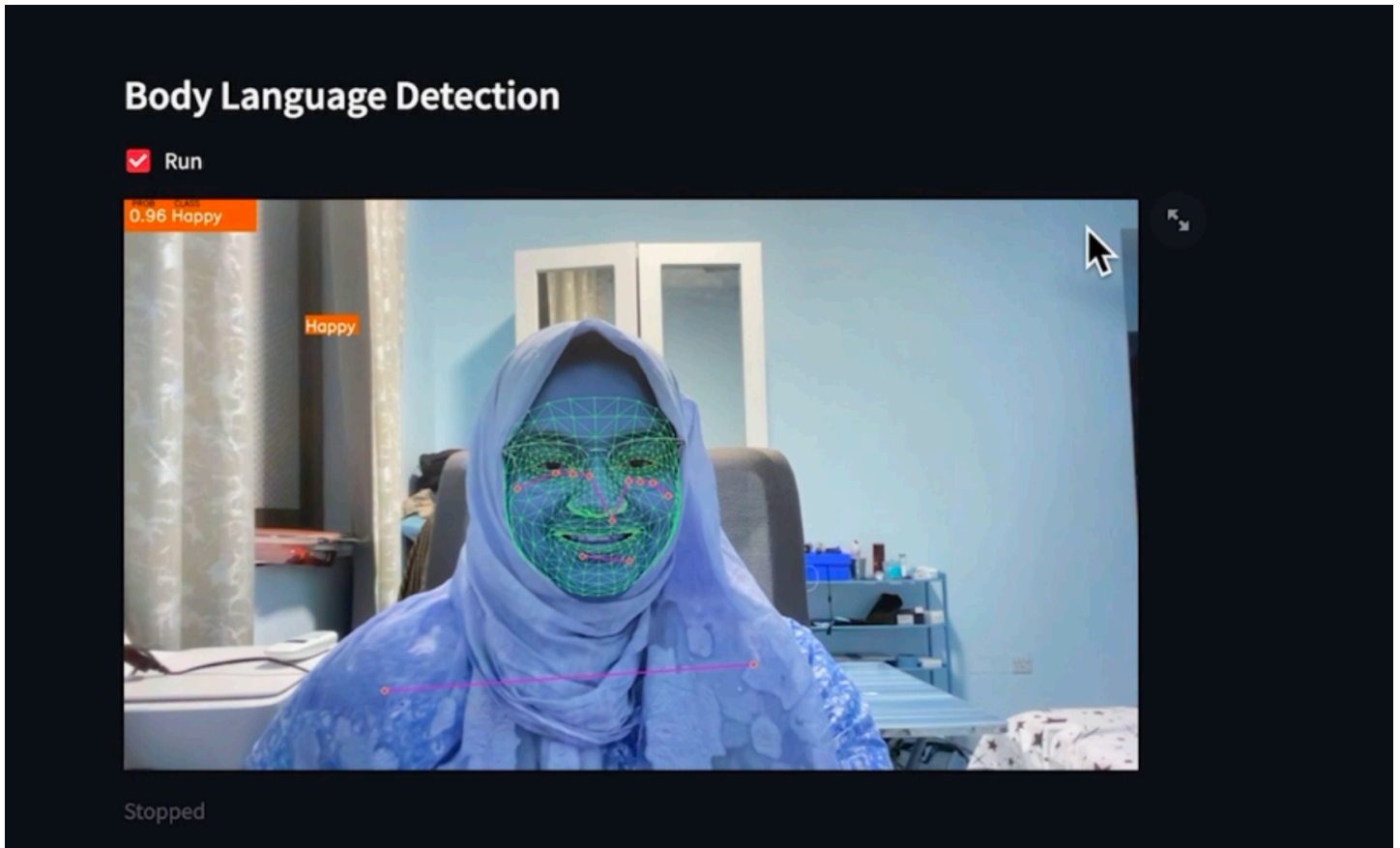
- Before clicking on Run checkbox :

Body Language Detection

☐ Run

Stopped

- After clicking on Run :



DEMO VID:

Google Drive: https://drive.google.com/file/d/19thzsgRY5RyZJ3qkIF6tf_W2jIHQkZjU/view?usp=sharing

Youtube: <https://youtu.be/rSOWZq5gsmw>