PROJECT DOCUMENTATION

—Tannu Kumari, Welcome to my documentation which is a journey through the projects that paved my success at Dataintelliage, a comprehensive system designed. These projects serves as a guide to understanding, implementing, and extending the capabilities of my knowledge and skills in the field of Data Science & AI.
Let's go beyond this and get a quick tour of innovation,creativity and problem-solving

1. Face Recognition with Emotion Detection

1. Overview

- Implement face recognition using the face_recognition library.

- Integrate emotion detection using a pre-trained model.
- Incorporate anti-spoofing measures with a pre-trained anti-spoofing model.
- Logged data into excel .

2. Libraries Used

Face Recognition Project used these libraries.

1. cv2 is used for capturing video, image processing, and basic computer vision tasks.
2. face_recognition is specifically designed for face-related tasks, such as face detection and recognition.
3. os helps in managing files and directories, crucial for loading models and organizing data.
4. numpy is used for efficient numerical operations, particularly helpful for handling image data.
5. keras.models.model_from_json is part of Keras and is used for loading neural network models from JSON files.
6. pandas is used for creating and manipulating DataFrames, which is useful for organizing and storing data.

3. Project Creation

This project involves the Recognition of face with known folder(known_image) and unknown folders (unknown_faces) in which it store no.of images of every unknown person in unknown folders by creating a unique folder for each unknown one and by that it recognize the unknown person with no spoofing. Each recognized face is assigned a unique identifier and stored data into XSLX file.

The following project structure outline is here :

```
1. FaceRecognitionProject/
2. │
3. ├── known_image/
4. │   ├── person1/
5. │   │   ├── image1.jpg
6. │   │   ├── image2.jpg
7. │   │   ├── ...
```

```
│   ├── person2/
│   │   ├── image1.jpg
│   │   ├── image2.jpg
│   │   ├── ...
│   ├── ...
│
├── unknown_faces/
│   ├── unknown_1/
│   │   ├── unknown1.jpg
│   │   ├── unknown1.jpg
│   │   ├── ...
│   ├── unknown_2/
│   │   ├── unknown2.jpg
│   │   ├── unknown2.jpg
│   │   ├── ...
│   ├── ...
│
├── antispoofing_models/
│   ├── antispoofing_model.json
│   └── antispoofing_model.h5
│
├── models/
│   └── haarcascade_frontalface_default.xml

├── face_recog.py
├── facialemotionmodel.json
├── facialemotionmodel.h5
├── output_data.xlsx
```

4. Application

The Face Recognition project described aims to create a system capable of recognizing faces, detecting emotions, and implementing anti-spoofing measures. Here's how the application might be used:

- **Face Recognition:**
  - Utilizes the `face_recognition` library for face detection and recognition.

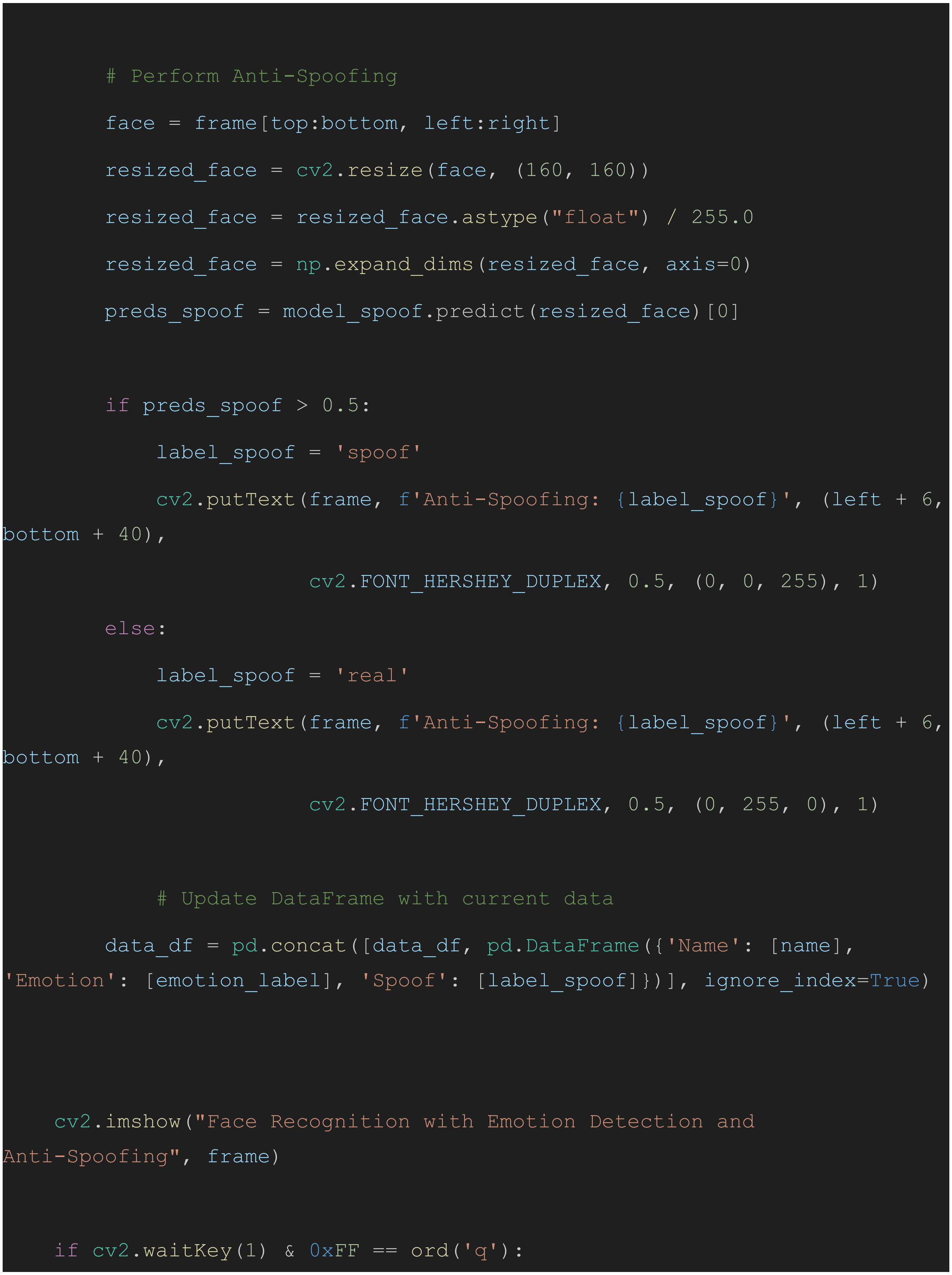
  - Compares unknown faces with known faces to identify individuals.
  - Assigns a unique identifier to each recognized face.
- **Emotion Detection:**
  - Implements emotion detection using a pre-trained model (`facialemotionmodel.json` and `facialemotionmodel.h5`).
  - Extracts face regions from recognized faces.
  - Predicts and labels emotions (angry, disgust, fear, happy, neutral, sad, surprise) in real-time.
- **Anti-Spoofing Measures**:
  - Utilizes an anti-spoofing model stored in the `antispoofing_models/` directory.
  - Captures faces in real-time using the webcam (`cv2.VideoCapture`).
  - Implements anti-spoofing measures to distinguish real faces from spoofed faces.
  - Labels each face as 'spoof' or 'real' based on the anti-spoofing model predictions.
- **Data Storage:**
  - Stores information for each recognized face into an Excel file (`output_data.xlsx`).
  - Includes the person's name (if known), detected emotion, and spoofing label ('spoof' or 'real').
- **Execution:**
  - Run the `project_code.py` script to initiate face recognition, emotion detection, and anti-spoofing processes.
  - View real-time results in the application displaying video feed with recognition outcomes.
  - Check the `output_data.xlsx` file for recorded data on recognized faces.
- **Further Improvements:**
  - Add a graphical user interface (UI) for a more user-friendly experience.
  - Optimize code for improved performance and scalability, especially with larger datasets.
  - Implement additional security measures based on specific use-case requirements.

5. Source Code

```
import cv2

import face_recognition

import os

import numpy as np

from keras.models import model_from_json

import pandas as pd



# Load emotion detection model

json_file = open("facialemotionmodel.json", "r")

model_json = json_file.read()

json_file.close()

emotion_model = model_from_json(model_json)

emotion_model.load_weights("facialemotionmodel.h5")


# Define emotion labels

emotion_labels = {0: 'angry', 1: 'disgust', 2: 'fear', 3: 'happy', 4:
'neutral', 5: 'sad', 6: 'surprise'}


# Load Anti-Spoofing Model

json_file_spoof = open('antispoofing_models/antispoofing_model.json', 'r')

loaded_model_json_spoof = json_file_spoof.read()

json_file_spoof.close()

model_spoof = model_from_json(loaded_model_json_spoof)

model_spoof.load_weights('antispoofing_models/antispoofing_model.h5')
```

```
print("Anti-Spoofing Model loaded from disk")

# Load Face Detection Model
face_cascade =
cv2.CascadeClassifier("models/haarcascade_frontalface_default.xml")


# Function to preprocess the image for emotion detection
def preprocess_image(image):
    resized_image = cv2.resize(image, (48, 48))
    gray_image = cv2.cvtColor(resized_image, cv2.COLOR_BGR2GRAY)
    normalized_image = gray_image / 255.0
    return normalized_image.reshape(1, 48, 48, 1)

faces_dir =
r"C:\Users\HP\Desktop\Face_Antispoofing_System-main\known_image"
unknown_faces_dir =
r"C:\Users\HP\Desktop\Face_Antispoofing_System-main\unknown_faces"

known_face_encodings = []
known_face_names = []

for filename in os.listdir(faces_dir):
    if filename.endswith(".npz"):
        path = os.path.join(faces_dir, filename)
        data = np.load(path, allow_pickle=True)
        known_face_encodings.append(data['encoding'])
        known_face_names.append(data['name'])
```

```python
known_face_encodings_dict = {tuple(encoding): name for encoding, name in zip(known_face_encodings, known_face_names)}

confidence_threshold = 0.6

cap = cv2.VideoCapture(0)

unknown_person_dict = {}

# Create an empty DataFrame to store data
columns = ['Name', 'Emotion', 'Spoof']
data_df = pd.DataFrame(columns=columns)

# Create a dictionary to keep track of added faces
added_faces = {}

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

    face_locations = face_recognition.face_locations(frame)
    face_encodings = face_recognition.face_encodings(frame, face_locations)

    for (top, right, bottom, left), face_encoding in zip(face_locations, face_encodings):

        cv2.rectangle(frame, (left, top), (right, bottom), (0, 255, 0), 2)
        # Extract face region for emotion detection
        face_region = frame[top:bottom, left:right]
        preprocessed_face = preprocess_image(face_region)

        # Predict emotion
        emotion_prediction = emotion_model.predict(preprocessed_face)
        emotion_label = emotion_labels[np.argmax(emotion_prediction)]

        # Display emotion label on the frame
        cv2.putText(frame, f'Emotion: {emotion_label}', (left + 6, bottom + 20), cv2.FONT_HERSHEY_DUPLEX, 0.5, (255, 255, 255), 1)

        matches = face_recognition.compare_faces(known_face_encodings, face_encoding, tolerance=confidence_threshold)

        if any(matches):
            first_match_index = matches.index(True)
            name = known_face_names[first_match_index]
            cv2.putText(frame, f"Known: {name}", (left + 6, bottom - 6), cv2.FONT_HERSHEY_DUPLEX, 0.5, (255, 255, 255), 1)
        else:
            face_encoding_tuple = tuple(face_encoding)

            found_match = False
            for known_encoding, identifier in unknown_person_dict.items():
                score = face_recognition.face_distance(np.array([known_encoding]), np.array(face_encoding_tuple))[0]
                if score < confidence_threshold:
                    identifier = unknown_person_dict[known_encoding]
                    found_match = True
                    break

            if found_match:
                name = f"Unknown_{identifier}"
            else:
                identifier = len(unknown_person_dict) + 1
                unknown_person_dict[face_encoding_tuple] = identifier
                name = f"Unknown_{identifier}"

                unknown_person_folder = os.path.join(unknown_faces_dir,
f"unknown_{identifier}")
                os.makedirs(unknown_person_folder, exist_ok=True)

            cv2.putText(frame, f"Unknown: {name}", (left + 6, bottom - 6),
cv2.FONT_HERSHEY_DUPLEX, 0.5, (255, 255, 255), 1)

            img_counter = len(os.listdir(unknown_person_folder)) + 1
            if img_counter <= 5:
                person_img = frame[top:bottom, left:right]
                person_filename =
f"unknown_{identifier}_{img_counter}.jpg"
                person_path = os.path.join(unknown_person_folder,
person_filename)

                cv2.imwrite(person_path, person_img)
```



```
        # Perform Anti-Spoofing
        face = frame[top:bottom, left:right]
        resized_face = cv2.resize(face, (160, 160))
        resized_face = resized_face.astype("float") / 255.0
        resized_face = np.expand_dims(resized_face, axis=0)
        preds_spoof = model_spoof.predict(resized_face)[0]

        if preds_spoof > 0.5:
            label_spoof = 'spoof'
            cv2.putText(frame, f'Anti-Spoofing: {label_spoof}', (left + 6,
bottom + 40),
                        cv2.FONT_HERSHEY_DUPLEX, 0.5, (0, 0, 255), 1)
        else:
            label_spoof = 'real'
            cv2.putText(frame, f'Anti-Spoofing: {label_spoof}', (left + 6,
bottom + 40),
                        cv2.FONT_HERSHEY_DUPLEX, 0.5, (0, 255, 0), 1)

            # Update DataFrame with current data
        data_df = pd.concat([data_df, pd.DataFrame({'Name': [name],
'Emotion': [emotion_label], 'Spoof': [label_spoof]})], ignore_index=True)


    cv2.imshow("Face Recognition with Emotion Detection and
Anti-Spoofing", frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        # Save the DataFrame to an Excel file before exiting
        excel_filename = 'output_data.xlsx'
        data_df.to_excel(excel_filename, index=False)
        break
```

6. References

https://github.com/tannukumari742/Facial_recognition

- **OpenCV documentation: https://docs.opencv.org/**
- **Keras documentation: https://keras.io/**
- **Haar Cascade Classifier: https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html**

2. Face Mask Detection

1. Overview

- Implement face mask detection using computer vision techniques.
- Utilize pre-trained deep learning models for accurate mask detection.
- Incorporate real-time video processing for mask detection.
- Face detection using a Haar Cascade classifier, and model prediction to label individuals as "Mask" or "No Mask."

2. Libraries Used

- cv2: For capturing video, image processing, and drawing rectangles and text on the frame.
- keras: For loading the pre-trained CNN model and image preprocessing.
- numpy: For numerical operations on the image array.

3. Project Creation

The Face Mask Detection project involves recognizing faces and determining whether individuals are wearing masks. The project structure is outlined below:

```
FaceMaskDetection/
|
|   ├── dataset/
|   |   ├──with mask/
|   |   ├── without mask/
├── dataset.h5
├── face_mask_detection.py
├── face_train.ipynb
```

4. Applications

Model Architecture

The CNN model architecture is as follows:

- Input Layer: Convolutional layer with 32 filters, 3x3 kernel size, and ReLU activation.
- MaxPooling Layer: Pooling layer with a 2x2 pool size.
- Convolutional Layer: 64 filters, 3x3 kernel size, and ReLU activation.
- MaxPooling Layer: Pooling layer with a 2x2 pool size.
- Convolutional Layer: 128 filters, 3x3 kernel size, and ReLU activation.
- MaxPooling Layer: Pooling layer with a 2x2 pool size.
- Flatten Layer: To flatten the output from convolutional layers.
- Dense Layer: Fully connected layer with 128 neurons and ReLU activation.
- Output Layer: Dense layer with a single neuron and a sigmoid activation function for binary classification.

Model Loading

The script loads the pre-trained weights of the model from a file named 'dataset.h5'.

Face Mask Detection Process

- The script captures video feed from the default camera (index 0).
- It resizes each frame to 150x150 pixels and normalizes the pixel values to the range [0, 1].
- The pre-trained model predicts whether a face in the frame is wearing a mask or not.
- The Haar Cascade classifier detects faces in the frame.
- If a face is detected, a red rectangle is drawn around it, and the predicted mask status is displayed below the rectangle.
- The application continues to process video frames until the user presses the 'q' key.

Execution

- To run the face mask detection, execute the script.
- The live video feed will be displayed, and faces will be labeled as "Mask" or "No Mask."

Further Imporvements

- Implement a more sophisticated face detection algorithm for improved accuracy.
- Add a confidence score or probability threshold for more reliable predictions.
- Integrate with a graphical user interface (UI) for better user interaction.
- Consider deploying the model on edge devices for real-world applications.

5. Source Code

A simple CNN model is built using Keras and loaded with pre-trained weights.
Video is captured using OpenCV, and each frame is processed for face detection and mask prediction.
Face detection is done using the Haar Cascade classifier.
The script continuously displays the video feed, drawing rectangles around faces and indicating mask status.

```
import cv2
#from tensorflow.keras.preprocessing import image
from keras.preprocessing import image
import numpy as np

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

def maskdetect():
    # Build a simple CNN model
    model = Sequential()
    model.add(Conv2D(32, (3, 3), input_shape=(150, 150, 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(Conv2D(128, (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'))

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

    # Load the trained model
    model.load_weights(r'C:\Users\HP\Desktop\face_mask\dataset.h5')

    # Open camera
    cap = cv2.VideoCapture(0)

    while True:
        ret, frame = cap.read()
        img = cv2.resize(frame, (150, 150))
        img = image.img_to_array(img)
        img = np.expand_dims(img, axis=0)
        img /= 255

        prediction = model.predict(img)
        mask_status = "Mask" if prediction[0][0] > 0.5 else "No Mask"

        # Detect face using a face detection algorithm (e.g., Haar
Cascade)
```

```
        # Assuming you have a trained Haar Cascade XML file for face detection named 'haarcascade_frontalface_default.xml'
        face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
        faces = face_cascade.detectMultiScale(frame, scaleFactor=1.3, minNeighbors=5)

        for (x, y, w, h) in faces:
            # Draw a red rectangle around the face
            cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 0, 255), 2)
            # Write text below the rectangle
            cv2.putText(frame, mask_status, (x, y+h+30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2, cv2.LINE_AA)

        cv2.imshow('Mask Detection', frame)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()

    #To call the function
if __name__ == "__main__":
    maskdetect()
```

6. References

GITHUB: https://github.com/tannukumari742/Facemask_detect

DATASET: https://drive.google.com/drive/folders/1z6cPVvMwrwucZw6yEEoB_MtxYuA3G_EW?usp=drive_link

3. Object Detection

1. Overview

- Implement an object detection project using the YOLO (You Only Look Once) model in Python.

2. Libraries Used

- Ultralytics: The YOLO model is loaded using the Ultralytics library.
- cv2 (OpenCV): Used for video capture, image processing, and drawing rectangles and text on the frame.
- math: Utilized for mathematical operations, specifically for calculating confidence values.

3. Dependencies

- Python 3.x
- Ultralytics library (`pip install yolov5`)
- OpenCV (`pip install opencv-python`)
- Pre-trained YOLO weights (`yolov8l.pt`)

4. Webcam Configurations

- Access the webcam using OpenCV (`cv2.VideoCapture`)
- Set webcam resolution to 640x480

4. Project Structure

The project includes the following components:

- YOLO Model: Pre-trained YOLO model weights are loaded from the 'yolo-Weights/yolov8l.pt' file.
- Webcam: The project captures video from the default camera, setting the dimensions to 640x480 pixels.
- Object Classes: A list of object classes is defined, including a variety of common objects.
- Object Detection Loop: The main loop continuously captures video frames, performs object detection, and displays the results in real-time.
- BoundingBoxes: Detected objects are enclosed with bounding boxes, and class labels and confidence scores are displayed.

4. Execution

To run the object detection project:

- Execute the script using a Python interpreter (`python script_name.py`).
- The webcam feed will display with bounding boxes around detected objects and associated class labels.
- Press 'q' to exit the application.

4. Object Detection Process

1. Video Capture: The script captures video frames from the webcam in real-time.
2. Model Loading: The pre-trained YOLO model is loaded using Ultralytics.
3. Object Detection: YOLO is applied to each frame to detect and classify objects.
4. Bounding Boxes: Detected objects are enclosed with colored bounding boxes.
5. Class Labels and Confidence Scores: The script prints class names and confidence scores for each detected object.
6. Real-time Display: The webcam feed is displayed with overlaid bounding boxes and object details.

4. Object Classes

The YOLO model can detect a wide range of objects, including but not limited to:

- Person
- Bicycle
- Car
- Traffic Light
- Banana
- Apple
- Laptop
- and many more...

4. Further Improvements

To enhance the project, consider the following improvements:

- Implement a more sophisticated user interface for better interaction.
- Fine-tune the YOLO model on a specific dataset for better accuracy.
- Integrate the project with additional features, such as object counting or tracking.

4. Source Code

```
from ultralytics import YOLO
import cv2
import math
# start webcam
cap = cv2.VideoCapture(0)
cap.set(3, 640)
cap.set(4, 480)

# model
model = YOLO("yolo-Weights/yolov8l.pt")
```

```
# object classes
classNames = ["person", "bicycle", "car", "motorbike", "aeroplane", "bus", "train", "truck",
"boat",
              "traffic light", "fire hydrant", "stop sign", "parking meter", "bench", "bird", "cat",
             "dog", "horse", "sheep", "cow", "elephant", "bear", "zebra", "giraffe", "backpack",
"umbrella",
              "handbag", "tie", "suitcase", "frisbee", "skis", "snowboard", "sports ball", "kite",
"baseball bat",
              "baseball glove", "skateboard", "surfboard", "tennis racket", "bottle", "wine glass",
"cup",
              "fork", "knife", "spoon", "bowl", "banana", "apple", "sandwich", "orange", "broccoli",
              "carrot", "hot dog", "pizza", "donut", "cake", "chair", "sofa", "pottedplant", "bed",
              "diningtable", "toilet", "tvmonitor", "laptop", "mouse", "remote", "keyboard", "cell
phone",
              "microwave", "oven", "toaster", "sink", "refrigerator", "book", "clock", "vase",
"scissors",
              "teddy bear", "hair drier", "toothbrush"
              ]
while True:
    success, img = cap.read()
    results = model(img, stream=True)

    # coordinates
    for r in results:
        boxes = r.boxes

        for box in boxes:
            # bounding box
            x1, y1, x2, y2 = box.xyxy[0]
            x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2) # convert to int values
            # put box in cam
            cv2.rectangle(img, (x1, y1), (x2, y2), (255, 0, 255), 3)

            # confidence
            confidence = math.ceil((box.conf[0]*100))/100
            print("Confidence --->",confidence)

            # class name
            cls = int(box.cls[0])
            print("Class name -->", classNames[cls])

            # object details
            org = [x1, y1]
            font = cv2.FONT_HERSHEY_SIMPLEX
            fontScale = 1
            color = (255, 0, 0)
            thickness = 2

            cv2.putText(img, classNames[cls], org, font, fontScale, color, thickness)

    cv2.imshow('Webcam', img)
    if cv2.waitKey(1) == ord('q'):
        break

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

5. References

https://github.com/tannukumari742/object_detect

4. AI Gym Trainer

1. Overview

The AI Gym Trainer is a computer vision-based project using the Mediapipe library to track and analyze human body movements during workout exercises. The system is capable of making real-time detections of body landmarks, calculating joint angles, and implementing specific workout-related features, such as counting curls.

2. Libraries Used

- cv2 (OpenCV): Used for video capture, image processing, and rendering visualizations.
- Mediapipe: Employs the Mediapipe library for pose estimation and landmark detection.
- numpy: Utilized for efficient numerical operations and handling landmark coordinates.

2. Source Code

```
!pip install mediapipe opencv-python
import cv2
import mediapipe as mp
import numpy as np
mp_drawing = mp.solutions.drawing_utils
mp_pose = mp.solutions.pose

# VIDEO FEED
```

```
cap = cv2.VideoCapture(0)
while cap.isOpened():
    ret, frame = cap.read()
    cv2.imshow('Mediapipe Feed', frame)

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

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

1. Make Detections

```
cap = cv2.VideoCapture(0)
## Setup mediapipe instance
with mp_pose.Pose(min_detection_confidence=0.5, min_tracking_confidence=0.5) as pose:
    while cap.isOpened():
        ret, frame = cap.read()

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

        # Make detection
        results = pose.process(image)

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

        # Render detections
        mp_drawing.draw_landmarks(image, results.pose_landmarks,
mp_pose.POSE_CONNECTIONS,
                                mp_drawing.DrawingSpec(color=(245,117,66), thickness=2, circle_radius=2),
                                mp_drawing.DrawingSpec(color=(245,66,230), thickness=2, circle_radius=2)
                                 )

        cv2.imshow('Mediapipe Feed', image)

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

    cap.release()
    cv2.destroyAllWindows()

mp_drawing.DrawingSpec??
```

2. Determining Joints

```
cap = cv2.VideoCapture(0)
## Setup mediapipe instance
with mp_pose.Pose(min_detection_confidence=0.5, min_tracking_confidence=0.5) as pose:
    while cap.isOpened():
        ret, frame = cap.read()

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

        # Make detection
        results = pose.process(image)

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

        # Extract landmarks
        try:
            landmarks = results.pose_landmarks.landmark
            print(landmarks)
        except:
            pass
        # Render detections
        mp_drawing.draw_landmarks(image, results.pose_landmarks,
mp_pose.POSE_CONNECTIONS,
                                mp_drawing.DrawingSpec(color=(245,117,66), thickness=2, circle_radius=2),
                                mp_drawing.DrawingSpec(color=(245,66,230), thickness=2, circle_radius=2)
                                 )
       cv2.imshow('Mediapipe Feed', image)

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

for lndmrk in mp_pose.PoseLandmark:
     print(lndmrk)
 landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].visibility

 landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value]
 landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value]
```

3. Calculate Angles

```
def calculate_angle(a,b,c):
    a = np.array(a) # First
    b = np.array(b) # Mid
    c = np.array(c) # End

    radians = np.arctan2(c[1]-b[1], c[0]-b[0]) - np.arctan2(a[1]-b[1], a[0]-b[0])
    angle = np.abs(radians*180.0/np.pi)

    if angle >180.0:
        angle = 360-angle

    return angle
```

In [14]:

```
shoulder = [landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].x,landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].y]
elbow = [landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value].x,landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value].y]
wrist = [landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value].x,landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value].y]

shoulder, elbow, wrist
calculate_angle(shoulder, elbow, wrist)
tuple(np.multiply(elbow, [640, 480]).astype(int))
```

```
cap = cv2.VideoCapture(0)
## Setup mediapipe instance
with mp_pose.Pose(min_detection_confidence=0.5, min_tracking_confidence=0.5) as pose:
    while cap.isOpened():
        ret, frame = cap.read()

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

        # Make detection
        results = pose.process(image)

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

        # Extract landmarks
        try:
            landmarks = results.pose_landmarks.landmark

            # Get coordinates
            shoulder = [landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].x,landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].y]
            elbow = [landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value].x,landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value].y]
            wrist = [landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value].x,landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value].y]

            # Calculate angle
            angle = calculate_angle(shoulder, elbow, wrist)

            # Visualize angle
            cv2.putText(image, str(angle),
                           tuple(np.multiply(elbow, [640, 480]).astype(int)),
                           cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2, cv2.LINE_AA
                                )

        except:
            pass


        # Render detections
        mp_drawing.draw_landmarks(image, results.pose_landmarks,
mp_pose.POSE_CONNECTIONS,
                                mp_drawing.DrawingSpec(color=(245,117,66), thickness=2, circle_radius=2),
                                mp_drawing.DrawingSpec(color=(245,66,230), thickness=2, circle_radius=2)
                                 )

        cv2.imshow('Mediapipe Feed', image)

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

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

4. Curl Counter

```
cap = cv2.VideoCapture(0)

# Curl counter variables
counter = 0
stage = None

## Setup mediapipe instance
with mp_pose.Pose(min_detection_confidence=0.5, min_tracking_confidence=0.5) as pose:
    while cap.isOpened():
        ret, frame = cap.read()
```

```
# Recolor image to RGB
image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
image.flags.writeable = False

# Make detection
results = pose.process(image)

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

# Extract landmarks
try:
    landmarks = results.pose_landmarks.landmark

    # Get coordinates
    shoulder = [landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].x,landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].y]
    elbow = [landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value].x,landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value].y]
    wrist = [landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value].x,landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value].y]

    # Calculate angle
    angle = calculate_angle(shoulder, elbow, wrist)

    # Visualize angle
    cv2.putText(image, str(angle),
                tuple(np.multiply(elbow, [640, 480]).astype(int)),
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2, cv2.LINE_AA
                    )
```

```
            # Curl counter logic
            if angle > 160:
                stage = "down"
            if angle < 30 and stage =='down':
                stage="up"
                counter +=1
                print(counter)

        except:
            pass

        # Render curl counter
        # Setup status box
        cv2.rectangle(image, (0,0), (225,73), (245,117,16), -1)

        # Rep data
        cv2.putText(image, 'REPS', (15,12),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,0), 1, cv2.LINE_AA)
        cv2.putText(image, str(counter),
                    (10,60),
                    cv2.FONT_HERSHEY_SIMPLEX, 2, (255,255,255), 2, cv2.LINE_AA)

        # Stage data
        cv2.putText(image, 'STAGE', (65,12),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,0), 1, cv2.LINE_AA)
        cv2.putText(image, stage,
                    (60,60),
                    cv2.FONT_HERSHEY_SIMPLEX, 2, (255,255,255), 2, cv2.LINE_AA)



        # Render detections
        mp_drawing.draw_landmarks(image, results.pose_landmarks,
mp_pose.POSE_CONNECTIONS,
                                mp_drawing.DrawingSpec(color=(245,117,66), thickness=2, circle_radius=2),
                                mp_drawing.DrawingSpec(color=(245,66,230), thickness=2, circle_radius=2)
                                 )

        cv2.imshow('Mediapipe Feed', image)

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

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

2. References

https://github.com/tannukumari742/Ai_pose_estimator

5. Hand Gesture Recognition

1. Overview

Hand Gesture Recognition is a computer vision application that involves the identification and interpretation of gestures made by the human hand using image processing techniques. This documentation covers the key components, libraries, and steps involved in implementing a simple Hand Gesture Recognition system.

2. Libraries Used

- OpenCV: Used for image and video processing, providing a foundation for capturing and manipulating frames.
- MediaPipe: Employs the MediaPipe library for hand tracking, landmark detection, and gesture recognition.
- NumPy: Utilized for efficient numerical operations and handling landmark coordinates.

2. Source Code

Mouse.py:

```
#Step - 1  -Import Libraries and capture camera

import cv2

import mediapipe as m #to detect the landmarks

import pyautogui

 #Read Camera

def mouse():

 cam=cv2.VideoCapture(0)

 cam.set(3,1280)

 cam.set(4,1080)

 drawing=m.solutions.drawing_utils
```

```
hands=m.solutions.hands
screen_width, screen_height = pyautogui.size()
x1 = y1 = x2 = y2 = 0
#How many hand u want to detect in frame
hand_obj=hands.Hands(max_num_hands =1) #1 hand in a frame
#to count the fingures using keypoints or landmarks on the hand
while True:
  #To read the frame
  _,frm = cam.read()
  #To flip the fame as it works like a window
  frm= cv2.flip(frm,1)
  frm_height, frm_width, _ = frm.shape
  res =hand_obj.process(cv2.cvtColor(frm,cv2.COLOR_BGR2RGB))
  hand_keyPoints =res.multi_hand_landmarks
  if hand_keyPoints:
       for hand in hand_keyPoints:
           drawing.draw_landmarks(frm, hand)
           one_hand_landmarks = hand.landmark
           for id, lm in enumerate(one_hand_landmarks):
               x = int(lm.x * frm_width)
               y = int(lm.y * frm_height)
               if id == 8:
                   mouse_x = int(screen_width/frm_width * x)
                   mouse_y = int(screen_height/frm_height * y)
                   cv2.circle(frm, center=(x,y), radius=10, color=(0,
255, 255))
                   pyautogui.moveTo(mouse_x,mouse_y)
                   pyautogui.click()
```

```
                    x1 = x
                    y1 = y
                if id == 4:
                    x2 = x
                    y2 = y
                    cv2.circle(frm, center=(x,y), radius=10, color=(0, 255, 255))
        dist = y2 -y1
        d= x1-x2
        print(dist,d)
        if(dist<35):
            pyautogui.click()
            print("CLICKED")
   #To show the frame
   cv2.imshow("windows",frm)
   #To destroy the window or frame on the screen
   if (cv2.waitKey(1)==27):
      cv2.destroyAllWindows()
      cam.release()
      break
```

Main.py

```
import tkinter as tk
from tkinter.ttk import *
from PIL import Image, ImageTk
#import MediaPlayer
import Mouse
```

```
#import Shopping
#import other

window = tk.Tk()
window.title("PCM ( PALM COMMUNICATION WITH MACHINE )")

#h1 = tk.Button(window, text=' P C M ', fg='black', width=10, bg="sky blue",font=("calibre",40,"bold"))
#h1.place(x=420,y=10)
#h2 = tk.Label(window, text='|< P.C.M >|', fg='pink', width=10, bg="teal",font=("calibre",70,"bold"))
#h2.place(x=180,y=15)



#MOUSE
b2=tk.Button(window,text="  MOUSE  " ,font=("Bahnschrift SemiBold SemiConden",25),bg="light yellow",fg="brown" , width=13,command=Mouse.mouse)
b2.place(x=1070,y=600)
image = Image.open(r"C:\Users\HP\Desktop\mouse\mouse.png")
# Resize the image using resize() method
resize_image = image.resize((350, 350))
img = ImageTk.PhotoImage(resize_image)
# create label and add resize image
label1 = Label(image=img)
label1.image = img
label1.place(x=1110,y=240)
```

```
#window

window.geometry("1500x1200")

window.configure(bg="teal")

window.mainloop()
```

2. References

https://github.com/tannukumari742/Ai_pose_estimator