LAU
School of Arts and Sciences

Department of Computer Science & Mathematics

CSC599H
Capstone Project

AI Coach

Gym Tracker and Exercise Form Corrector using Machine Learning and Artificial Intelligence

written by:
Edward Prescott-Decie - 202101920
Tamer Saleh - 202102710

Introduction:

Given the importance of exercise in the modern day, as well as considerable advances in accessibility with regards to artificial intelligence, this project produced a system studying fitness in order to provide an optimal and educational user experience. It aimed to utilize AI in studying exercises, determining critical movements and angles within that are necessary for ideal gain. This was accomplished through using artificial intelligence and machine learning to observe footage and form models of humans performing relevant exercises along with applying researched knowledge to provide recommendations. Furthermore, this system used AI to study a human's exercise patterns, recommending sets of daily workouts tailored to their needs and schedule, once again through the use of the above systems. These features were then formed into an app, being housed there to showcase the practicality and usability of what we have developed throughout our time working on it.

Background and Preliminaries:

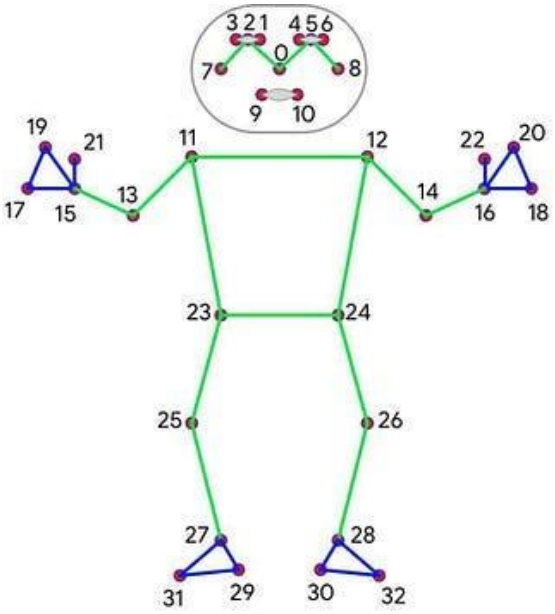
MediaPipe:

MediaPipe is an open-source framework developed by Google that offers ready-to-use Machine Learning solutions for various perceptual tasks, primarily focused on real-time multimedia processing. It provides a set of pre-trained models and a unified pipeline to process multimedia inputs such as images, video streams, and audio.

This project uses MediaPipe's **Holistic** Model, which is designed for holistic human pose estimation. It aims to detect and track multiple key points on a person's body, providing a comprehensive understanding of their pose in real-time. The MediaPipe Holistic model consists of several key components:

- Pose Detection: This component detects the 33 key points representing the human body's pose, including key points on the face, upper body, and lower body.
- Face Detection: It detects key points on the face, including landmarks for the eyes, nose, mouth, and facial contours.
- Hand Detection: This component identifies key points on each hand, including landmarks for the fingers and palm.

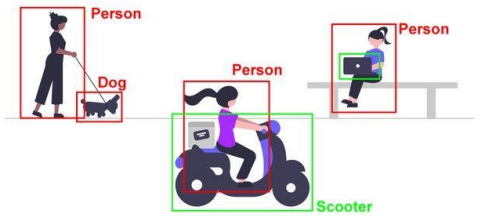
We will be using the pose detection componenets, with the 33 key points, or landmarks, listed below:



0. nose
1. right eye inner
2. right eye
3. right eye outer
4. left eye inner
5. left eye
6. left eye outer
7. right ear
8. left ear
9. mouth right
10. mouth left
11. right shoulder
12. left shoulder
13. right elbow
14. left elbow
15. right wrist
16. left wrist
17. right pinky knuckle #1
18. left pinky knuckle #1
19. right index knuclke #1
20. left index knuckle #1
21. right thumb knuckle #2
22. left thumb knuckle #2
23. right hip
24. left hip
25. right knee
26. left knee
27. right ankle
28. left ankle
29. right heel
30. left heel
31. right foot index
32. left foot index

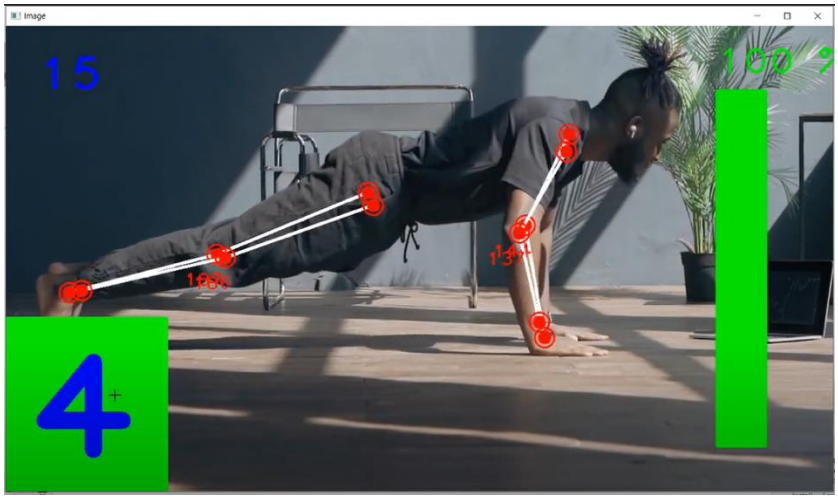
OpenCV:

OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library. It provides a wide range of functionalities for image and video processing, including image manipulation, feature detection, object recognition, and more.



YoungWonks

The reason we are using OpenCV in this project along with MediaPipe is because it can be used to capture video streams from cameras or read video files, as well as pre-processing the video frames before feeding them into the MediaPipe pipeline. This may include operations such as resizing, cropping, and color space conversion. Additionally, OpenCV can be used to overlay annotations or visualizations of the detected behavior on the video frames, providing real-time feedback to users. Hence, OpenCV complements MediaPipe by providing essential image and video processing functionalities, enabling the creation of a complete pipeline for detecting exercises in video streams or recorded videos. The combined use of OpenCV and MediaPipe offers a powerful solution for real-time computer vision-based exercise analysis and fitness tracking applications.



Exercises:

Plank

Data Collection:

Setting up the environment:

For this project, we are using libraries such as MediaPipe and OpenCV, which are useful tools for pose detection and drawing landmarks on the videos. For this, we need to use

pip install mediapipe opencv-python pandas

in an Anaconda based kernel.

```
import mediapipe as mp
import cv2
import numpy as np
import pandas as pd
import csv
```

We then import all these modules to be used in our code. We also import the **csv** module since we will be saving the data we collect in a csv file to use later on in the model.

Collecting raw data:

The way the data will be collecting is by capturing a recorded video of us actually doing the exercise in the correct way. We set an export path which is a csv file.

```
mp_drawing = mp.solutions.drawing_utils
mp_holistic = mp.solutions.holistic
mp_pose = mp.solutions.pose
cap = cv2.VideoCapture("plank.mov")
EXPORT_PATH = "plank.csv"
MODEL_PATH = "plank.pkl"
```

In the while loop of the holistic model, we are manually labelling each frame for the plank. It is either “high”, “low”, or “normal” depending on the position of the torso.

```
#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()

        #recoloring image since frame is in BGR
        image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        #make detections
        results = holistic.process(image)
        image_height, image_width, _ = image.shape
        #convert image back to BGR to process it
        image = cv2.cvtColor(image,cv2.COLOR_RGB2BGR)

        try:
            landmark_list = results.pose_landmarks.landmark

            key = cv2.waitKey(1)
            if (key == ord('1')):
                writeToCSV(results,"low")
            elif (key == ord('2')):
                writeToCSV(results,"high")
            elif(key == ord('0')):
                writeToCSV(results,"normal")

        except:
            pass
```

Bicep Curl

Data Collection:

Setting up the environment:

For this project, we are using libraries such as MediaPipe and OpenCV, which are useful tools for pose detection and drawing landmarks on the videos. For this, we need to use

pip install mediapipe opencv-python pandas

in an Anaconda based kernel.

```
import mediapipe as mp
import cv2
import numpy as np
import pandas as pd
import csv
```

We then import all these modules to be used in our code. We also import the **csv** module since we will be saving the data we collect in a csv file to use later on in the model.

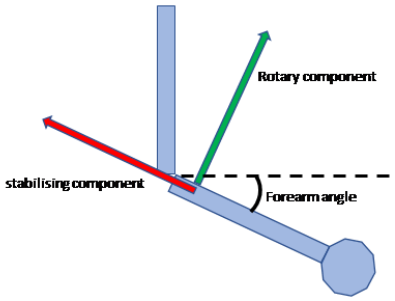
Collecting raw data:

The way the data will be collecting is by capturing a recorded video of us actually doing the exercise in the correct way. We set an export path which is a csv file.

```
cap = cv2.VideoCapture("vid.mov") #currently recording from video
EXPORT_PATH = "bicep_curl_dataset.csv"
```

Within a MediaPipe Holistic model (which is the actual pose detecting model we imported), we set a while loop which is continously reading the frames of the video. For each frame, we are calculating the angle by taking in the angle between the wrist, elbow, and shoulder for both arms. This angle is calculated by retrieving the x and y coordinate values of each landmark and applying a mathematical formula between these 3 points.

```
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
```




```python
#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()

        #recoloring image since frame is in BGR
        image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        #make detections
        results = holistic.process(image)
        image_height, image_width, _ = image.shape
        #convert image back to BGR to process it
        image = cv2.cvtColor(image,cv2.COLOR_RGB2BGR)

        try:
            landmark_list = results.pose_landmarks.landmark
            shoulderLeft = [landmark_list[mp_pose.PoseLandmark.LEFT_SHOULDER.value].x,landmark_list[mp_pose.PoseLandmark.LEFT_SHOULDER.value].y]
            elbowLeft = [landmark_list[mp_pose.PoseLandmark.LEFT_ELBOW.value].x,landmark_list[mp_pose.PoseLandmark.LEFT_ELBOW.value].y]
            wristLeft = [landmark_list[mp_pose.PoseLandmark.LEFT_WRIST.value].x,landmark_list[mp_pose.PoseLandmark.LEFT_WRIST.value].y]

            shoulderRight = [landmark_list[mp_pose.PoseLandmark.RIGHT_SHOULDER.value].x,landmark_list[mp_pose.PoseLandmark.RIGHT_SHOULDER.value].y]
            elbowRight = [landmark_list[mp_pose.PoseLandmark.RIGHT_ELBOW.value].x,landmark_list[mp_pose.PoseLandmark.RIGHT_ELBOW.value].y]
            wristRight = [landmark_list[mp_pose.PoseLandmark.RIGHT_WRIST.value].x,landmark_list[mp_pose.PoseLandmark.RIGHT_WRIST.value].y]

            # Calculate angle
            angleLeft = int(calculate_angle(shoulderLeft, elbowLeft, wristLeft))

            angleRight = int(calculate_angle(shoulderRight, elbowRight, wristRight))

            #manually write to CSV file using keys
            key = cv2.waitKey(1)
            if (key == 119):
                writeToCSV(results,"up",angleLeft)
            elif (key == 115):
                writeToCSV(results,"down",angleLeft)
            elif (key == 117):
                writeToCSV(results,"up",angleRight)
            elif (key == 100):
                writeToCSV(results,"down",angleRight)
```

In this loop, we are manually writing the coordinates and angles that correspond to each class with key stroke values. For the left arm, I am using the keys **w** and **s** to record whether the arm is up or down respectively. For the right arm, I am using **u** and **d** to classify the up and down positions. An example of how the data collection looks is provided below, using a video of Majd doing bicep curls.

Deadlift

Data Collection:

Collecting raw data:

For the deadlift exercise, we record the data by performing three different versions of the exercise (3 different videos: deadlift_arms, deadlift_back, and deadlift_stance). Instead of working with angles in the deadlift exercise, we are working with position landmarks and machine learning only. 3 different data collection processes with 3 different videos, working with 3 different models each. One for the arms, one for the legs stance, and one for the back.

```
mp_drawing = mp.solutions.drawing_utils
mp_holistic = mp.solutions.holistic
mp_pose = mp.solutions.pose
cap = cv2.VideoCapture("deadlift_arms.mov")
EXPORT_PATH = "deadliftArms.csv"
MODEL_PATH = "deadliftArms.pkl"
```

Within a MediaPipe Holistic model (which is the actual pose detecting model we imported), we set a while loop which is continously reading the frames of the video. For each frame, we have a manual data collection system set up, where landmark coordinates for each frame are classified with key strokes. For example, pressing 1 on the deadlift back data collection will label the coordinates as “bent” (this is when the back of the deadlifter is too bent downwards).

```
#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()

        #recoloring image since frame is in BGR
        image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        #make detections
        results = holistic.process(image)
        image_height, image_width, _ = image.shape
        #convert image back to BGR to process it
        image = cv2.cvtColor(image,cv2.COLOR_RGB2BGR)

        try:
            landmark_list = results.pose_landmarks.landmark

            key = cv2.waitKey(1)
            if (key == ord('1')):
                writeToCSV(results,"bent")
            elif (key == ord('2')):
                writeToCSV(results,"normal")
            elif (key == ord('0')):
                writeToCSV(results,"standing")


        except:
            pass
```

```python
#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()

        #recoloring image since frame is in BGR
        image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        #make detections
        results = holistic.process(image)
        image_height, image_width, _ = image.shape
        #convert image back to BGR to process it
        image = cv2.cvtColor(image,cv2.COLOR_RGB2BGR)

        try:
            landmark_list = results.pose_landmarks.landmark

            key = cv2.waitKey(1)
            if (key == ord('1')):
                writeToCSV(results,"narrow")
            elif (key == ord('2')):
                writeToCSV(results,"wide")
            elif(key == ord('3')):
                writeToCSV(results,"normal")


        except:
            pass
```

```python
#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()

        #recoloring image since frame is in BGR
        image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        #make detections
        results = holistic.process(image)
        image_height, image_width, _ = image.shape
        #convert image back to BGR to process it
        image = cv2.cvtColor(image,cv2.COLOR_RGB2BGR)

        try:
            landmark_list = results.pose_landmarks.landmark

            key = cv2.waitKey(1)
            if (key == ord('1')):
                writeToCSV(results,"wide")
            elif (key == ord('2')):
                writeToCSV(results,"narrow")
            elif(key == ord('0')):
                writeToCSV(results,"normal")

        except:
            pass
```

Squat

Collecting raw data:

The way the data will be collecting is by capturing a recorded video of us doing squats repeatedly. The video includes squats with good form as well as squats with bad form.

```
mp_drawing = mp.solutions.drawing_utils
mp_holistic = mp.solutions.holistic
mp_pose = mp.solutions.pose
cap = cv2.VideoCapture("squat_vid.mov")
EXPORT_PATH = "squat.csv"
MODEL_PATH = "squat.pkl"
```

We calculate angles between specific body joints like the knee, hip, and ankle using our calculate_angle function. These angles help us evaluate the correctness of exercises like squats or deadlifts.

As we iterate over frames from the video feed, we extract pose landmarks and calculate angles for the left leg. We keep track of the minimum angle of the left knee (min_knee_angle). When we find a new minimum knee angle, we check if it satisfies certain conditions related to correct squat form. If it does, we label the knee, hip, and heel angles and write them along with the angles to a CSV file.

So essentially, we're analyzing squat form based on the angles between key body joints and recording this information for further analysis.

```
try:
    landmark_list = results.pose_landmarks.landmark

    left_hip = [landmark_list[mp_pose.PoseLandmark.LEFT_HIP.value].x, landmark_list[mp_pose.PoseLandmark.LEFT_HIP.value].y]
    left_knee = [landmark_list[mp_pose.PoseLandmark.LEFT_KNEE.value].x, landmark_list[mp_pose.PoseLandmark.LEFT_KNEE.value].y]
    left_ankle = [landmark_list[mp_pose.PoseLandmark.LEFT_ANKLE.value].x, landmark_list[mp_pose.PoseLandmark.LEFT_ANKLE.value].y]
    left_shoulder = [landmark_list[mp_pose.PoseLandmark.LEFT_SHOULDER.value].x, landmark_list[mp_pose.PoseLandmark.LEFT_SHOULDER.value].y]
    left_heel = [landmark_list[mp_pose.PoseLandmark.LEFT_HEEL.value].x, landmark_list[mp_pose.PoseLandmark.LEFT_HEEL.value].y]
    left_foot = [landmark_list[mp_pose.PoseLandmark.LEFT_FOOT_INDEX.value].x, landmark_list[mp_pose.PoseLandmark.LEFT_FOOT_INDEX.value].y]

    #Calculate angles
    angle_left_knee = int(calculate_angle(left_hip, left_knee, left_ankle))
    angle_left_hip = int(calculate_angle(left_shoulder, left_hip, left_knee))
    angle_left_heel = int(calculate_angle(left_knee,left_heel,left_foot))

    knee_label = 0
    hip_label = 0
    heel_label = 0

    if (angle_left_knee <min_knee_angle):
        min_knee_angle = angle_left_knee
        if(min_knee_angle < 110):
            if(min_knee_angle < 50):
                knee_label = 1

            if(angle_left_hip < 55):
                hip_label = 1

            if(angle_left_heel < 70):
                heel_label = 1

            writeToCSV(results,knee_label,hip_label,heel_label,angle_left_knee,angle_left_hip,angle_left_heel)
            min_knee_angle = 10000
```

Data Preparation:

To prepare our data for the model, we imported the necessary libraries such as pandas, pickle, and scikit-learn modules. We load the data from the CSV file into a **pandas DataFrame**. Next, we prepare the data by separating the features (X) and the target variable (y), where X consists of all the columns except the first. This first column is the label, and is different depending on the exercise.

Data Splitting:

Furthermore, we split the dataset into training and testing sets using the `train_test_split` function from scikit-learn, with 70% of the data allocated for training and 30% for testing. This division ensures that our model can be trained on a portion of the data and evaluated on a separate portion to assess its performance.

```python
import pickle
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LogisticRegression, RidgeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score

MODEL_PATH = "bicep.pkl"

#load the data from the csv file
try:
    df = pd.read_csv('bicep_curl_dataset.csv')
except Exception as e:
    print(e)

#prepare the data by setting the X and y values
X = df.iloc[:, 1:]  #Attributes (all columns except the first)
y = df.iloc[:, 0]   #Target variable (first column)

#Split the Data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1234)
```

Model Selection:

```
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('classifier', RandomForestClassifier())
])
```

For each exercise, we created a pipeline containing preprocessing step using `StandardScaler()` followed by the Random Forest Classifier. The pipeline is first scaled using `StandardScaler()` before being passed to the classifier. This ensures that the features are on the same scale, which can improve the performance of many machine learning algorithms.

Normalization/Standardization:

The StandardScaler is a preprocessing technique to standardize the features by removing the mean and scaling to unit variance. It transforms the data so that its distribution will have a mean of 0 and a standard deviation of 1. This is achieved by subtracting the mean of each feature and dividing by its standard deviation. Standardizing the features ensures that they all have the same scale, which can help improve the performance of the model. By using StandardScaler as part of the pipeline, we ensure that our input data is properly scaled before being fed into the classifier.

Training the Model:

In this part of the project, we simply train the model to the X_train and y_train records that were generated in the data preparation stage. We use the **fit** function to fit the pipeline classifier to the data.

```
model = pipeline.fit(X_train,y_train)
```

Making Predictions and Application:

Plank

In the plank section, we are turning the coordinates into a row input, sending it to the model, and the model is predicting the class of the plank depending on the coordinates for each frame. If the probability of each stage is above a certain threhold we label it as either “high” or “low”, otherwise the plank is “normal”. Additionally, we implemented a timer for the plank screen so that users can see how long they hold the plank.

```
elif(name == "Plank"):

    # Calculate elapsed time
    if start_time is not None:
        elapsed_time = time.time() - start_time
        cv2.putText(image, f"Elapsed Time: {elapsed_time:.1f} s", (50, 700), cv2.FONT_HERSHEY_SIMPLEX, 2, (255, 255, 255), 2, cv2.LINE_AA)

    row = np.array([[res.x,res.y,res.z,res.visibility] for res in results.pose_landmarks.landmark]).flatten().tolist()

    X = pd.DataFrame([row],columns = plank_landmarks[1:])

    #predict plank position
    predicted = modelPlank.predict(X)[0]
    prob = modelPlank.predict_proba(X)[0]

    print(predicted,prob)

    if predicted == "high" and prob.max() >= 0.5:
        stage = "high"
    elif predicted == "low" and prob.max() >= 0.5:
        stage = "low"
    else:
        stage = "normal"

    #Arms Stage Display
    cv2.putText(image, 'STAGE', (150,50), cv2.FONT_HERSHEY_SIMPLEX, 1.5, (0,0,0), 2, cv2.LINE_AA)
    cv2.putText(image, stage, (150,120), cv2.FONT_HERSHEY_SIMPLEX, 1, (255,255,255), 2, cv2.LINE_AA)

    #Arms Probability Display
    cv2.putText(image, 'PROB', (350,50), cv2.FONT_HERSHEY_SIMPLEX, 1.5, (0,0,0), 2, cv2.LINE_AA)
    cv2.putText(image,str(prob[np.argmax(prob)]), (350,120), cv2.FONT_HERSHEY_SIMPLEX, 1, (255,255,255), 2, cv2.LINE_AA)
```

Deadlift

The deadlift application works by applying all three models at the same time for each frame. The coordinates are prepared and put in a row, and the each model works on the same coordinates. There is a predicted label and probability for each type of aspect of the deadlift. After the models predict the labels, we add conditional statements to display the label (depending on whether it meets a certain probability threshold).

```
elif(name == "Deadlift"):

    row = np.array([[res.x,res.y,res.z,res.visibility] for res in landmark_list]).flatten().tolist()

    X = pd.DataFrame([row],columns = deadlift_landmarks[1:])
    X.head()

    # #predict arms position
    predicted_Arms = modelArms.predict(X)[0]
    probArms = modelArms.predict_proba(X)[0]

    # #predict back position
    predicted_Back = modelBack.predict(X)[0]
    probBack = modelBack.predict_proba(X)[0]

    #predict legs stance position
    predicted_Stance = modelStance.predict(X)[0]
    probStance = modelStance.predict_proba(X)[0]

    print(predicted_Stance,probStance)

    print(predicted_Back,probBack)
    if predicted_Arms == "wide" and probArms.max() >= 0.6:
        arms_stage = "wide"
    elif predicted_Arms == "narrow" and probArms.max() >= 0.6:
        arms_stage = "narrow"
    else:
        arms_stage = "normal"

    if predicted_Back == "normal" and probBack.max() >= 0.6:
        back_stage = "normal"
    elif predicted_Back == "bent" and probBack.max() >= 0.6:
        back_stage = "bent"
    else:
        back_stage = "standing"

    if predicted_Stance == "wide" and probStance.max() >= 0.6:
        stance_stage = "wide"
    elif predicted_Stance == "narrow" and probStance.max() >= 0.6:
        stance_stage = "narrow"
    else:
        stance_stage = "normal"
```

Bicep Curl

In this section, we are applying the model into the real time tracker. Similar to the data collection stage, we are initiating the holistic model and openCV as usual. However, this time, we are taking the current angle as well as the current landmarks and predicting their class with the model. We are also calculating the probability of the model being correct with the **.prob** method.

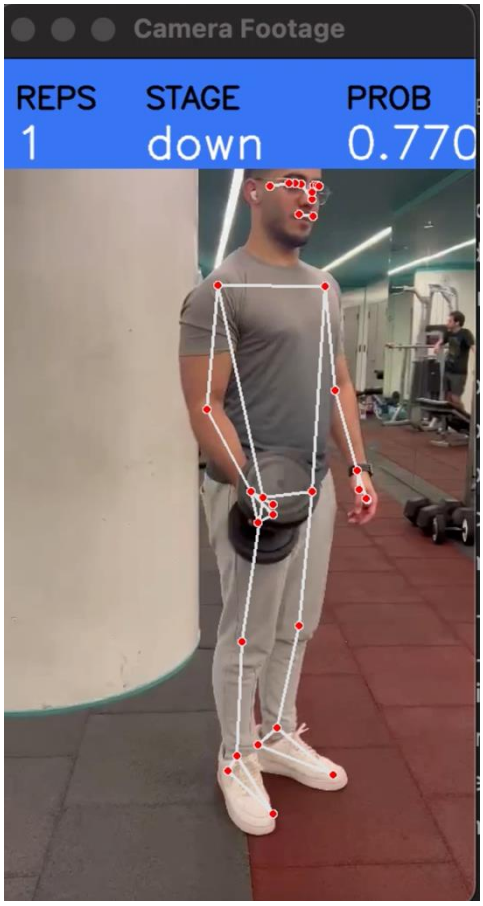
```
MODEL_PATH = "bicep.pkl"
counter = 0
stage = None
prob = np.array([0,0])

#Retrieve model from pickle file
with open(MODEL_PATH,'rb') as file:
    model = pickle.load(file)

#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()

        #recoloring image since frame is in BGR
        image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        #make detections
        results = holistic.process(image)
        image_height, image_width, _ = image.shape
        #convert image back to BGR to process it
        image = cv2.cvtColor(image,cv2.COLOR_RGB2BGR)

        try:

            landmark_list = results.pose_landmarks.landmark
            shoulder = [landmark_list[mp_pose.PoseLandmark.LEFT_SHOULDER.value].x,landmark_list[mp_pose.PoseLandmark.LEFT_SHOULDER.value].y]
            elbow = [landmark_list[mp_pose.PoseLandmark.LEFT_ELBOW.value].x,landmark_list[mp_pose.PoseLandmark.LEFT_ELBOW.value].y]
            wrist = [landmark_list[mp_pose.PoseLandmark.LEFT_WRIST.value].x,landmark_list[mp_pose.PoseLandmark.LEFT_WRIST.value].y]

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

            row = np.array([[res.x,res.y,res.z,res.visibility] for res in results.pose_landmarks.landmark]).flatten().tolist()
            row = np.insert(row,0,angle)
            X = pd.DataFrame([row],columns = landmarks[1:])

            predicted_stage = model.predict(X)[0]
            prob = model.predict_proba(X)[0]
            # print(predicted_stage)
            # print(prob)
            if predicted_stage == "down" and prob.max() >= 0.7:
                stage = "down"
            elif stage == "down" and predicted_stage == "up" and prob.max() >= 0.7:
                stage = "up"
                counter += 1
```

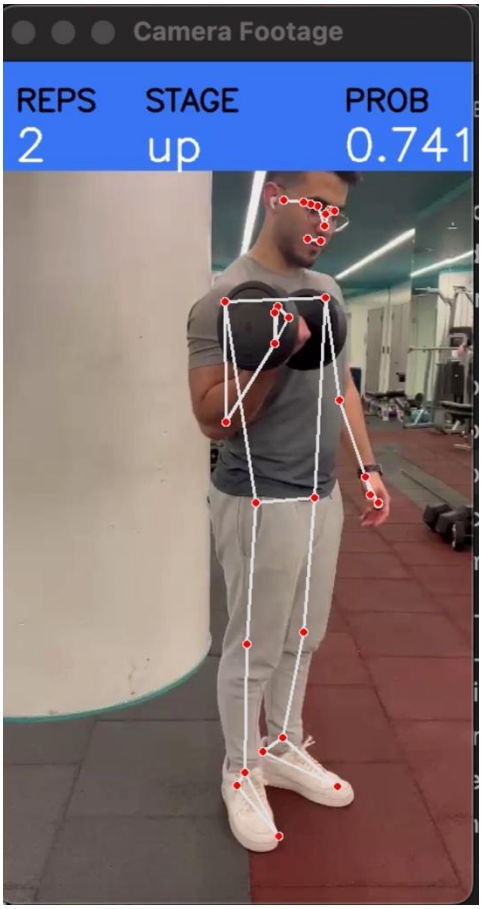
The main part of this loop where the model is working is here, where the landmarks are saved, the angle is calculated, and then the data is prepared for the model. We use **np.array** and **.flatten()** to turn all the coordinates of each landmark into one array. We insert the angle into the row at index 0, and then set the X input to be a pandas dataframe.

```
            landmark_list = results.pose_landmarks.landmark
            shoulder = [landmark_list[mp_pose.PoseLandmark.LEFT_SHOULDER.value].x,landmark_list[mp_pose.PoseLandmark.LEFT_SHOULDER.value].y]
            elbow = [landmark_list[mp_pose.PoseLandmark.LEFT_ELBOW.value].x,landmark_list[mp_pose.PoseLandmark.LEFT_ELBOW.value].y]
            wrist = [landmark_list[mp_pose.PoseLandmark.LEFT_WRIST.value].x,landmark_list[mp_pose.PoseLandmark.LEFT_WRIST.value].y]

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

            row = np.array([[res.x,res.y,res.z,res.visibility] for res in results.pose_landmarks.landmark]).flatten().tolist()
            row = np.insert(row,0,angle)
            X = pd.DataFrame([row],columns = landmarks[1:])

            predicted_stage = model.predict(X)[0]
            prob = model.predict_proba(X)[0]
```

To find the predicted stage, we simply call **model.predict(X)[0]** and find the prediction probability as well. We use the following logic to display the stage to the user and increment the counter on every rep. We only change the stage if the probability of the predicted stage is greater than 70%. The following pictures below were taken from the demo video, showcasing how the model is being applied on a new video.

```
    # print(prob)
    if predicted_stage == "down" and prob.max() >= 0.7:
        stage = "down"
    elif stage == "down" and predicted_stage == "up" and prob.max() >= 0.7:
        stage = "up"
        counter += 1
```





Squat

For the squat portion of this project, a different approach was taken. We wanted to investigate the possibility of applying neural networking techniques to body analysis and form recommendation. In this portion, we tackled the problem a little differently.

The data generated looked as follows:

	knee_angle	hip_angle	heel_angle	valid1	valid2	valid3	x1	y1	z1	v1	...	z31	v31	x32	y32	z
0	109	96	75	0	0	0	0.422298	0.313926	-0.198739	0.999966	...	0.161433	0.775709	0.447284	0.818140	-0.1610
1	102	88	75	0	0	0	0.421823	0.334982	-0.182559	0.999968	...	0.167418	0.770504	0.447909	0.818270	-0.2293
2	98	83	75	0	0	0	0.421213	0.343640	-0.196267	0.999971	...	0.130517	0.766818	0.450655	0.817757	-0.2013
3	92	77	69	0	0	1	0.421092	0.359942	-0.177569	0.999974	...	0.122099	0.761735	0.450428	0.816853	-0.1887
4	83	68	69	0	0	1	0.420553	0.377307	-0.165750	0.999975	...	0.137702	0.741701	0.450379	0.816874	-0.1697

The above figure showcases the angles generated along side with the landmark positions to the right. There are also three label fields: valid1, valid2 and valid3. These fields take on binary digits. A 1 signifies a valid state that takes into account one of the angles whereas a 0 signifies an invalid state. It is important that there are three labels as we are looking to make recommendations for three body functions being the angle at the knee and whether the squat is too low, the angle at the heel and whether the whole body is too forward, and the angle and the hip signifying how bent the back is.

We then apply a split on the data in a similar fashion to the machine learning model split:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x,y, test_size=0.2, random_state=42)
```

The data is normalized and scaled:

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

And next, the model is built:

```
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])
```

We note that the model for the time being is a simple feed forward network. The input layer consists of 64 neurons with the relu activation function. There is a hidden layer with half the number of neurons. Finally, the output layer consists of a single neuron as we are making binary decisions. As such, we are using the sigmoid activation function to form predictions.

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=100, batch_size=32, verbose=2)
model.save("knee1.keras")
```

We compile the model using adaptive moment optimization along with the binary crossentropy function to study loss. We then run the model over 100 epochs with batch sizes of 32. Finally, the model is saved in order to be loaded later into our app. It is evidently important to note that this process must be repeated three times taking into

account a separate label each time. This is done in order to have three models to generate the respective recommendations.

The accuracies generated by the model are the following:

```
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=0)
print("Test Accuracy:", test_acc)
```

```
Test Accuracy: 0.8965517282485962
```

```
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=0)
print("Test Accuracy:", test_acc)
```

```
Test Accuracy: 1.0
```

```
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=0)
print("Test Accuracy:", test_acc)
```

```
Test Accuracy: 0.9137930870056152
```

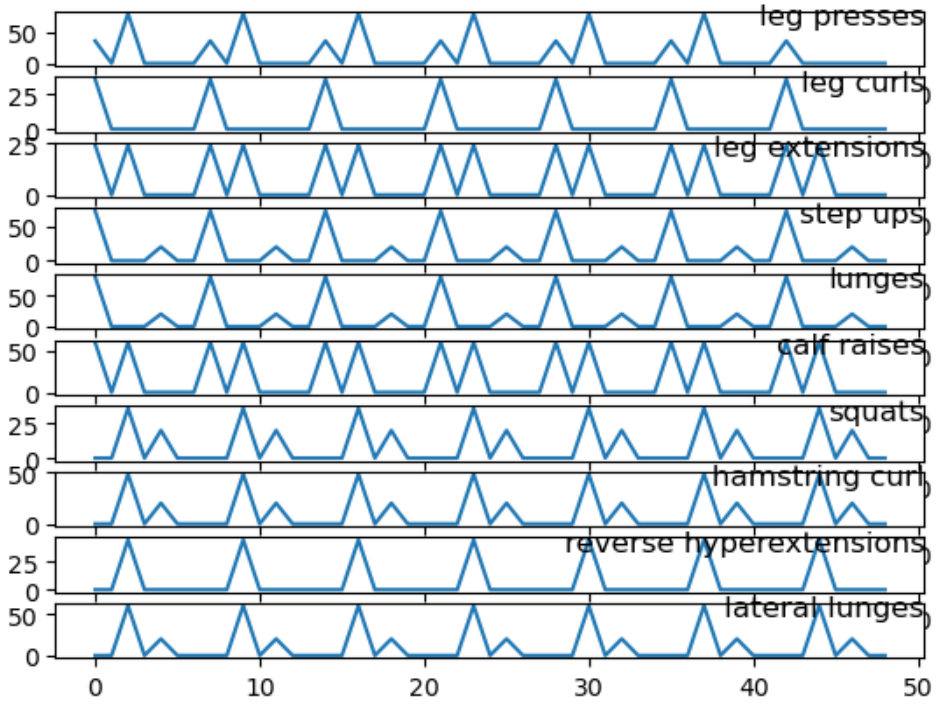
The accuracy of each model is relatively quite high. It should be noted that the relationship between what a good angle is and what a bad one is should not be very hard to gauge. This means that our model should be able to generate a generally good guess.

Recurrent Neural Network Exercise Scheduling:

In order to extend the usage of neural network techniques in this project, we decided to delve into the use of an RNN to study a sequence of data consisting of the number of reps that a person does of a certain exercise per day. In order to start, we begin with the data used to train the model. The following is the workout schedule of a friend:

year	month	day	leg presses	leg curls	leg extensi	step ups	lunges	calf raises	squats	hamstring	reverse hy	lateral lunges
2024	4	1	36	36	24	72	80	60	0	0	0	0
2024	4	2	0	0	0	0	0	0	0	0	0	0
2024	4	3	80	0	24	0	0	60	36	48	45	60
2024	4	4	0	0	0	0	0	0	0	0	0	0
2024	4	5	0	0	0	20	20	0	20	20	0	20
2024	4	6	0	0	0	0	0	0	0	0	0	0
2024	4	7	0	0	0	0	0	0	0	0	0	0
2024	4	8	36	36	24	72	80	60	0	0	0	0
2024	4	9	0	0	0	0	0	0	0	0	0	0
2024	4	10	80	0	24	0	0	60	36	48	45	60
2024	4	11	0	0	0	0	0	0	0	0	0	0
2024	4	12	0	0	0	20	20	0	20	20	0	20
2024	4	13	0	0	0	0	0	0	0	0	0	0
2024	4	14	0	0	0	0	0	0	0	0	0	0
2024	4	15	36	36	24	72	80	60	0	0	0	0

The following is the data in graph form:



We see that the data is quite regular and follows some very notable weekly patterns. This gives us confidence that a deep recurrent neural network will be able to pick up on these patterns and make appropriate predictions on what exercises a user might be interested in working out on a new date.

The next steps consist of dealing with the data and forming the model itself.

We begin by formatting the date and neatly setting it into a single field in our dataframe to be able to use it as an index more easily in the future:

```
data['date'] = pd.to_datetime(data[['year', 'month', 'day']])
data.drop(['year', 'month', 'day'], axis=1, inplace=True)
data = data.set_index('date')
```

We proceed by normalizing our data:

```
scaler = MinMaxScaler()
data_scaled = scaler.fit_transform(data)
```

And next we continue by creating our sequences. In creating the sequences, we had to decide on a specific length. We initially guessed that since the data followed weekly patterns, it would be appropriate to take sequence lengths of seven. However, after testing, we found a sweet spot to be five days. In taking sequences of length five, the most accurate results were obtained as will be shown shortly.

```
def create_sequences(data, n_steps):
    X, y = [], []
    for i in range(len(data)):
        end_ix = i + n_steps
        if end_ix > len(data)-1:
            break
        seq_x, seq_y = data[i:end_ix], data[end_ix]
        X.append(seq_x)
        y.append(seq_y)
    return np.array(X), np.array(y)

# Number of past days to consider for predicting the next day
n_steps = 5
X, y = create_sequences(data_scaled, n_steps)
```

Next we form a simple long-short term memory model. We thought it important to use an LSTM as we wanted to deal with long sequences of data that spanned over multiple days and was likely to experience small changes as the user may differ slightly per day in the choice of exercise or the choice of repetitions of that exercise. The user may also work out on an irregular basis and the model must be fit to determine and pick up on patterns that may exist within the irregularities. For this reason, We determine the LSTM to be the most appropriate for dealing with our sequential data.

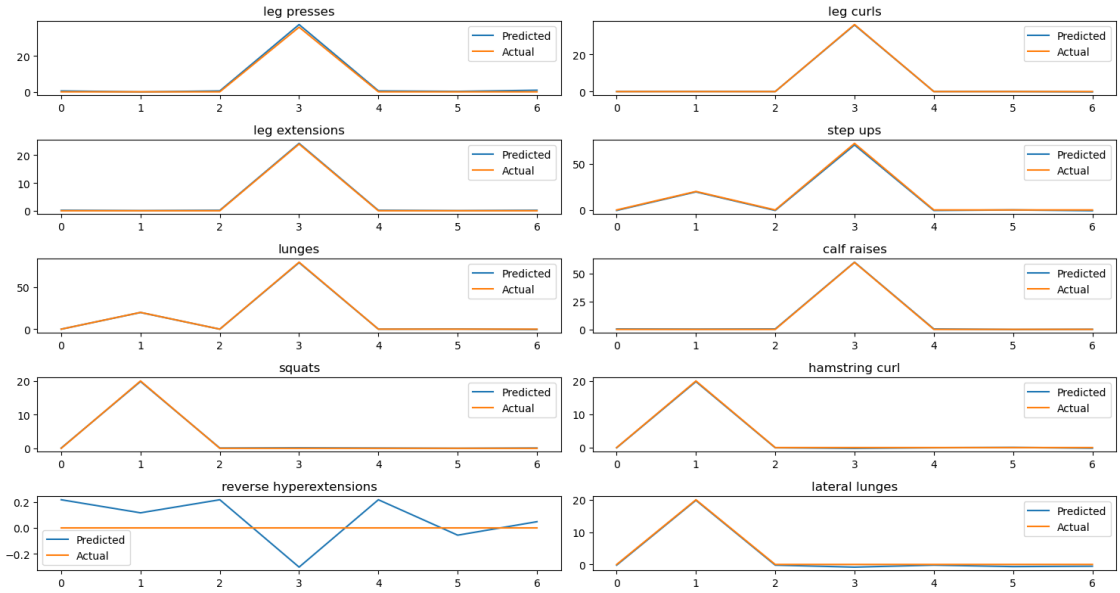
```
model = Sequential([
    LSTM(100, activation='relu', input_shape=(n_steps, X.shape[2])),
    Dense(y.shape[1])
])
model.compile(optimizer='adam', loss='mse')
model.fit(X_train, y_train, epochs=200, verbose=1, validation_split=0.2)
model.save("schedule.keras")
```

The above image showcases the use of the LSTM. We begin with an input LSTM layer with 100 neurons all of which have the relu function as an activation function.

We also found it unnecessary to have a more complicated feed forward network following the LSTM as the accuracy for our predictions already seemed to be quite high. For this

reason, we jumped straight to an output layer with 1 neuron. The model was compiled using adaptive moment optimization alongside mean squared error as the loss function. The model was then run over 200 epochs leaving a small split to be used for validation. And once again, the model was saved in order to be used in the app.

The following image showcases the accuracy of the model by plotting the predictions against the actual last week of exercises. Essentially, the model was tricked by pretending like the last week of data is unknown and trying to guess the values. As can be seen, the accuracy is quite high:



All of the predictions experience hardly any deviance at all. However, it may seem as though the model did not make a strong prediction for the “reverse hyperextension” exercise. However, we must note the scale of the y-axis being a range of 0.2 around 0. This makes sense once we realize that the exercise was not done over the span of the last week. In this regard, our model has performed exceptionally well in gauging the pattern of the user’s workout. For reference the formatting of the data in order to predict the last week is as follows:

```
predictions = model.predict(X_test)
predictions_rescaled = scaler.inverse_transform(predictions)
actual_rescaled = scaler.inverse_transform(y_test)
last_week_predictions = predictions_rescaled[-7:]
last_week_actual = actual_rescaled[-7:]

# Plotting the results
plt.figure(figsize=(15, 8))
for i in range(last_week_predictions.shape[1]):
    plt.subplot(5, 2, i+1)
    plt.plot(last_week_predictions[:, i], label='Predicted')
    plt.plot(last_week_actual[:, i], label='Actual')
    plt.title(data.columns[i])
    plt.legend()
plt.tight_layout()
plt.show()
```

Finally, we must test the model on the daily basis and see the numerical values that are generated when requiring a prediction for the exercises to be performed in a day. The following image showcases the programming required to generate the exercise values for the new day:

```
new_data = data.tail(n_steps)
new_data_scaled = scaler.transform(new_data)
new_sequence = new_data_scaled.reshape((1, n_steps, new_data_scaled.shape[1]))
new_prediction = model.predict(new_sequence)
new_prediction_rescaled = scaler.inverse_transform(new_prediction)

for i in range(len(new_prediction_rescaled[0])):
    if new_prediction_rescaled[0][i] < 5:
        new_prediction_rescaled[0][i] = 0
```

The following is the output:

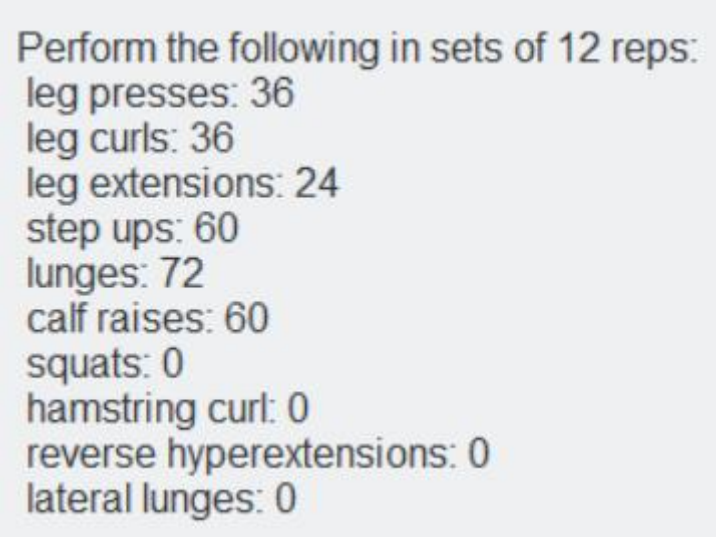
```
Prediction for the new day: [[37. 36. 24. 70. 79. 60.  0.  0.  0.  0.]]
```

We note that these values are quite accurate and almost exactly represent the actual schedule of the user.

Scheduling Exercise Recommendations Within the App:

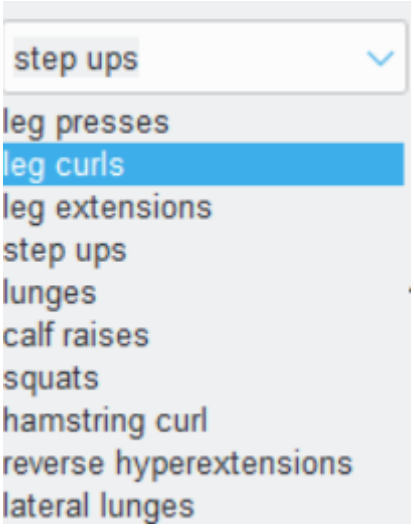
It must first be noted that the predicted data is further formatted in order to be in multiples of twelve. This is done because the user is required to perform exercises in specific sets of 12 repetitions. In this specific use case, the number of repetitions is 12, however, depending on the users preference, the repetitions may be set to any number of repetitions.

The model is loaded and the current date is inputted. If the date already has a line in the file storing all the exercises, then we must not generate new predictions. Otherwise, we create a new line and output our predictions onto them. The outputted data looks as follows:

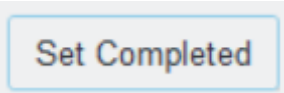


Another functionality of the app is such that any exercise that is recorded on inputted is subtracted from the recommendations. The user can manually input the number of exercises in the following manner.

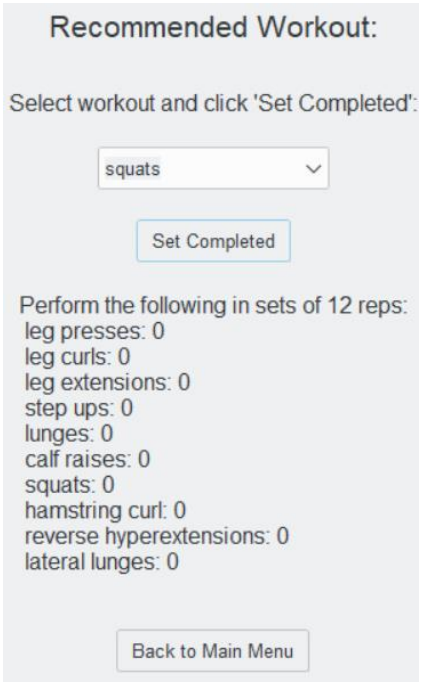
An exercise can be selected from a dropdown field:



A button exists underneath this field that when pressed, subtracts a given amount of repetitions from the total:



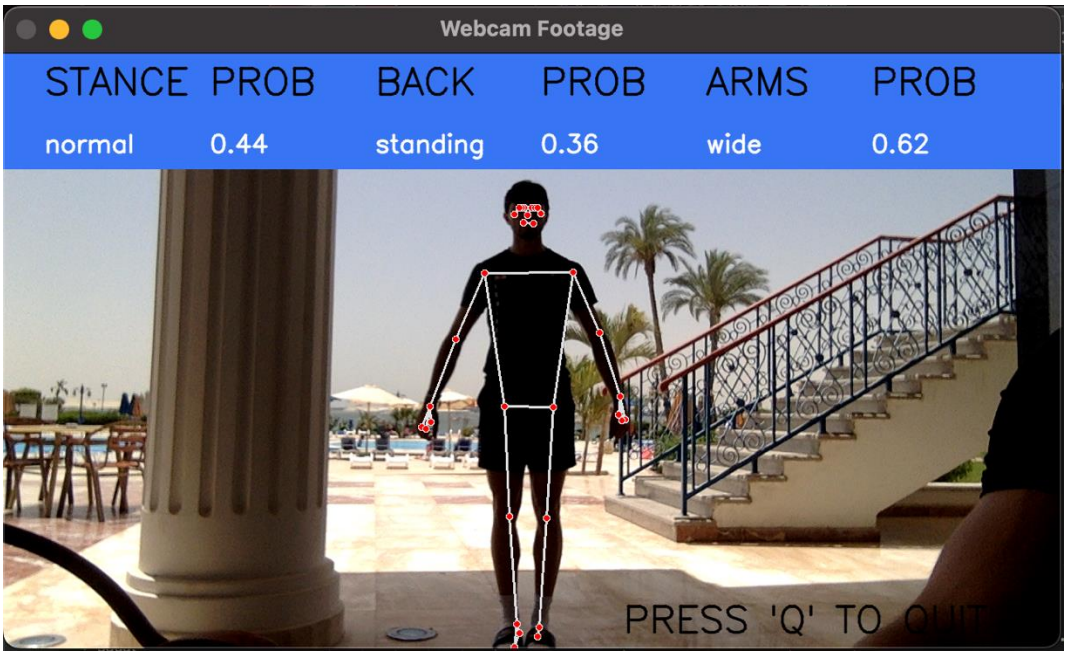
Pressing this button will subtract the values as stated above with the condition that no value go beneath 0. A complete set of exercises for the day will look as follows:



Application Design:

For the GUI, we decided to go for a simple Tkinter Python App since it would be usable with the python machine learning models. OpenCV and MediaPipe are python based, so we wanted to keep all the application within the same language for feasibility purposes. The Home Screen has buttons for all the different exercises, and within each exercise page we have a “Open Camera” button which activates the OpenCV camera. The current setup of the app uses the webcam, but OpenCV can take camera footage from any working camera on the laptop. If needed, an extra camera can be connected to the computer and be used for the live footage.

In the Camera Screen, a window is displayed for the current camera being used (either webcam or an external connected camera). Several data items are displayed on the screen as well depending on the exercise. For example, for the Deadlift exercise, we have “Stance”, “Arms”, and “Back” to display the feedback for the different aspects of the deadlift.



The labels change as the movement of the user changes (depending on the landmark positions). The user can quit the camera screen by clicking on “q”, taking you back to the workout screen of that exercise.

Recommendation System:

The application will have several different recommendations depending on the exercise being performed. Additionally, there will be a rep counter/timer for each exercise.

Exercise: Squat

- Angle 1: Knee Angle (hip – knee – foot)
  - If too low, then squat is too deep (butt is too low to the ground)
    - Feedback:
  - If too high, then squat is not low enough
  - Should be in the range of 80-100 degrees
- Angle 2: Hip Angle (shoulder – hip – knee)
  - If too low: Back too bent forward
    - Feedback: **Straighten your back**
- Angle 3: Heel Angle (knee – ankle – toe)
  - If too low: Knees are too forward
  - Feedback: **Put the weight on your heels, not on your knees**

Exercise: Deadlift

- Leg Stance:
  - If legs too far apart
    - Feedback: **Bring your legs closer together**
  - If legs too close to each other
    - Feedback: **Separate your legs until shoulder width**
- Back:
  - If back is too low while doing the deadlift
    - Feedback: **Straighten your back, keep it straight up**
- Arms:
  - If arms too far apart
    - Feedback: **Bring your arms closer together**
  - If arms too close to each other
    - Feedback: **Separate your arms until shoulder width**

Exercise: Plank

- If torso is too low
  - Feedback: **Bring your butt up so that it is in a straight line with your back and legs**
- If butt too high
  - Feedback: **Lower your butt and engage your core**

Exercise: Bicep Curl

- Feedback includes:
    - Rep Counter: Reps are only counted if user performs exercise correctly
    - Form Analysis: For each rep, there will be feedback given to the user on their form and what they were doing wrong
    - Example for deadlift exercise:
        - Rep X – Did not go low enough
        - Rep Y – Need to increase leg width
        - Etc… (different feedback for different exercises based on the angles)

Conclusion:

The purpose of this project is to expand upon an up-and-coming field. The usage of artificial intelligence and machine learning is proving more and more to be useful in the studying of human anatomy and the way that we perform exercises. There are many possible uses for this form of technology. We showcase a couple in this project being the studying of form and the ways in which AI can optimize the way that we perform exercises. Additionally, we develop a system which is able study a user's workout schedule and recommend exercises based on it. The neural network used here was a recurrent one as the data is sequential in nature. Both of these areas of the project proved to be highly successful and we were able to showcase that this form of software may be highly useful to anybody that is interested in physical activity ranging anywhere from working out to running, or swimming or even team sports like football. The purpose of this project was not necessarily to put all of this into a usable application with frontend features but rather to showcase the possibility of its use and develop a backend which can deal with this form of data and perform accurate predictions

Further Extensions:

Due to the nature of the project, there are in fact a number of areas that can be expanded upon:

- The number of exercises can be increased. For the scope of this project, the number of exercises chosen is enough as every exercise requires a lot of studying in order to determine the critical angles and areas of the body which are important. It was already a lot of effort gathering this data for some of the exercises chosen and would be even more complicated for more complicated exercises.
- The models chosen can be experimented upon further and different types of models can be used to form the predictions
- The creation of a more user-friendly application can be applied to the project in order to test the feasibility of use of the features presented

Limitations:

There are a couple areas which are limited in this project such as:

- The models chosen could have been hypertuned to a further degree to see if greater accuracy could have been obtained. The parameters could have also been tested for computationally in order to ensure the best results possible.
- There was no expert involved in the decision making behind optimal angles and form. This means that the artificial intelligence was trained on data which was researched on our end. The results would of course be more relevant to real life if a sports expert were involved

References:

Dhanke, J. A., Maurya, R. K., Navaneethan, S., Mavaluru, D., Nuhmani, S., Mishra, N., & Venugopal, E. (2022). Recurrent Neural Model to Analyze the Effect of Physical Training and Treatment in Relation to Sports Injuries. *Computational intelligence and neuroscience*, *2022*, 1359714. https://doi.org/10.1155/2022/1359714

Namatevs, Ivars & Aleksejeva, Ludmila & Polaka, Inese. (2016). Neural Network Modelling for Sports Performance Classification as a Complex Socio-Technical System. *Information Technology and Management Science*. 19. 10.1515/itms-2016-0010.

Kasim Serbest. (2022). A Biomechanical Analysis of Dumbbell Curl and Investigation of the Effects of Increasing Loads on Biceps Brachii Using A Finite Element Model/ *Research Square*. https://doi.org/10.21203/rs.3.rs-1263844/v1

Youssef, Fatma & Bayomy, Ahmed & Gomaa, Walid. (2022). Analysis of the Squat Exercise from Visual Data. 79-88. 10.5220/0011347900003271.

Github:

https://github.com/edwardpresdec/capstone.git