Deep Learning Course

Instructor: Dr.Tamizharasan Periyasamy

Group Members	**ID Number**
1. Mohammad Ali Haider	2022A7PS0122U
2. Syed Afraz	2022A7PS0142U
3. Mufti Muqaram Majid Farooqi	2022A7PS0235U

A Report

On

Workout Posture Correction Using Computer Vision Techniques

Prepared
For

Deep Learning Course (CS F425)

By

Mohammad Ali, Muqaram Majid and Syed Afraz

December 2024

ABSTRACT

This project presents a novel real-time fitness tracking system designed to classify and evaluate three fundamental exercises—push-ups, squats, and bicep curls. The system begins by training a YOLO model on a custom dataset derived from 1-minute, 60-fps video recordings of each exercise. Frames extracted from these videos were annotated and augmented using Roboflow, yielding a dataset with approximately 4,000–5,000 images per class, split into 70:20:10 train, validation, and test sets. Following model training, MediaPipe Pose was integrated to overlay skeletal landmarks on detected classes, enabling the assessment of exercise form. By analyzing specific joint positions and angles, the system provides feedback on whether the exercise is performed with proper form or not. The solution is deployed via a Streamlit application offering three input options: real-time video via DroidCam USB, webcam capture, or video upload. This comprehensive approach bridges the gap between exercise classification and performance evaluation, providing users with actionable feedback to improve their workout routines.

ACKNOWLEDGEMENTS

We would like to extend our gratitude to Dr. Tamizharasan Periyasamy for his support and guidance throughout the course of this project. His role as our professor has been instrumental in providing the foundation for our research and helping us navigate through the various stages of our work.

TABLE OF CONTENTS

Abstract i

Acknowledgements ii

List of Figures iv

1. Introduction 8

    1.1 Background and Motivation 8

    1.2 Objectives of the Project 8

2. Methodology 10

    2.1. Data Collection 10

    2.2 Model Training 11

    2.3 Pose Estimation and Form Analysis 12

3. Deployment

    3.1 Deployment With Streamlit 16

4. Implementation

    4.1. Webcam, Video Upload and Droid Cam Functionality

5. Model Performance

    5.1 Model Performance and Comparsion

6. Conclusion

    6.1 Summary of Achievements

    6.2 Potential Enhancements

References 21

LIST OF FIGURES

Figure 2.1. Dataset Preview 10

Figure 2.4.1. Check Pushup Form 12

Figure 2.4.2. Pushup Phases 12

Figure 2.4.3. Bicep curl alignment 13

Figure 2.4.4. Bicep Curl Position 14

Figure 2.4.5. Squats 15

Figure 3.1.1. Streamlit code 17

Figure 3.1.2. Streamlit Interface 17

Figure 5.1.1 Metrics 19

Figure 5.1.2 Confusion Matrix 19

Figure 5.1.3 Precision-Confidence Curve 20

Figure 5.1.4 Recall-Confidence Curve 20

Figure 5.1.5 F1-Confidence Curve 21

Figure 5.1.6 Loss Graphs 21

CHAPTER 1: INTRODUCTION

1.1. BACKGROUND AND MOTIVATION

The growing emphasis on health and fitness has led to a significant increase in the adoption of exercise routines among people worldwide. However, incorrect exercise form and posture can often result in injuries, diminishing the benefits of physical activity. Despite the availability of fitness trainers, many individuals rely on home workouts or online resources, which lack personalized feedback on their performance. The motivation behind this project stems from the need for an affordable and accessible solution that can guide users to perform exercises with proper form, ensuring safety and efficiency. By leveraging computer vision and deep learning technologies, we aim to create a system that not only identifies exercises but also provides real-time feedback on form.

This project combines the power of YOLO for exercise classification and MediaPipe for skeletal landmark detection to address this need. The integration of these tools into a user-friendly Streamlit interface ensures that anyone with basic hardware can benefit from precise and actionable feedback during their workouts.

1.2. OBJECTIVES OF THE PROJECT

The primary objectives of this project are as follows:

1. **Exercise Classification**
    - Train a deep learning model capable of accurately identifying three specific exercises: push-ups, squats, and bicep curls, using video data.
2. **Form Evaluation**
    - Utilize skeletal landmark detection to analyze user posture and determine whether the exercise is being performed with proper form or not.
3. **Real-Time Feedback**
    - Provide users with immediate feedback on their exercise form, highlighting areas of improvement and ensuring safety during workouts.
4. **User-Friendly Deployment**

- Develop an intuitive interface using Streamlit that allows users to interact with the system through three modes:
  - Using their smartphone with DroidCam.
  - Using their PC webcam.
  - Uploading pre-recorded videos.

5. **Accessibility and Affordability**
   - Ensure the solution is cost-effective and accessible, requiring only basic hardware such as a smartphone or a computer with a webcam.

CHAPTER 2: METHODOLOGY

2.1. DATA COLLECTION

To ensure the model was trained on diverse and realistic data, a systematic data collection process was followed:

1. **Video Recording**
   - Videos for three exercises—push-ups, squats, and bicep curls—were recorded.
   - Each exercise was captured for 1 minute at 60 frames per second (fps) to ensure sufficient frame coverage of the movements.

2. **Frame Extraction**
   - The recorded videos were processed using a custom script to extract individual frames, resulting in thousands of images for each exercise.

3. **Data Annotation**
   - The extracted frames were uploaded to Roboflow, a data annotation and augmentation platform.
   - Using Roboflow's annotation tools, the images were labeled into three distinct classes:
     - Push-up
     - Squat
     - Bicep Curl

4. **Augmentation**
   - To increase dataset diversity and improve model generalization, data augmentations were applied through Roboflow.
   - These augmentations included variations in brightness, rotation, cropping, and flipping, simulating different real-world conditions.

5. **Dataset Splitting**
   - The final dataset, containing approximately 4,000–5,000 images per class, was split into training, validation, and test sets with the following proportions:
     - Training Set: 70% of the data
     - Validation Set: 20% of the data
     - Test Set: 10% of the data

This systematic approach to data collection ensured that the dataset was both robust and representative, allowing the model to effectively learn and distinguish between the three exercises.

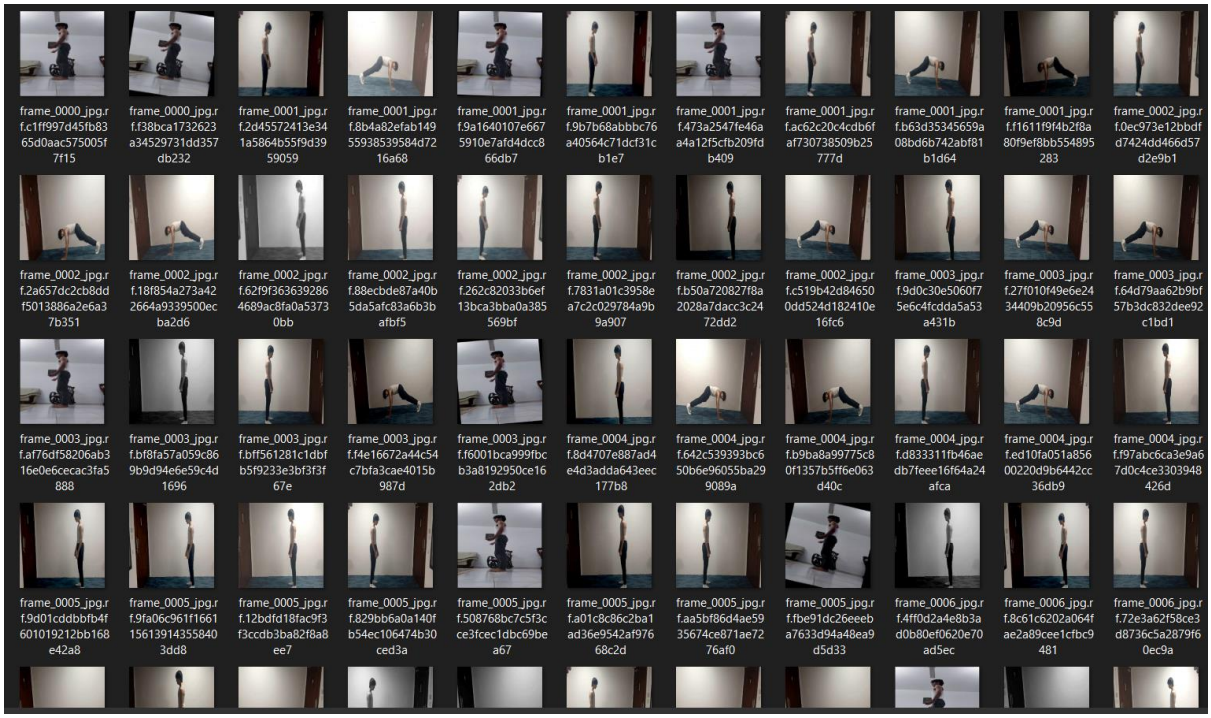

Figure 2.1 Dataset Preview

2.2. MODEL TRAINING

For the training of the exercise detection model, the YOLOv8 architecture from the Ultralytics YOLO framework was used due to its effectiveness in object detection tasks. This choice was made to ensure both speed and accuracy in real-time exercise detection.

1. Training Setup
   The model was trained on a GPU to optimize performance and reduce training time. The input images were resized to 640x640 pixels to maintain a balance between computational efficiency and detection accuracy.

2. Hyperparameters
   The model was trained for a total of 50 epochs. A batch size of 16 was used to ensure stable convergence, and the learning rate was set according to the default configuration of the YOLOv8 framework.

3. Training Process
   During training, the model learned to detect and classify the three exercises—push-ups, squats, and bicep curls—based on the labeled dataset. The training process included monitoring the loss values and other performance metrics to track the progress.

4. Model Output
   After the training was completed, the model weights were saved for subsequent evaluation and deployment, ensuring the model could be used for real-time exercise classification in future tasks.

2.3. POSE ESTIMATION AND FORM ANALYSIS

In this part of the project, pose estimation was used to assess the form of the exercises performed by the user. We used MediaPipe Pose for tracking key body landmarks and then applied specific methods for form analysis based on the detected body movements. The focus was on detecting whether the user performed each exercise with correct posture, based on predefined criteria for each exercise type. Here's how we handled form analysis for different exercises:

Push-ups

For push-ups, we divided the movement into two phases: the up phase (when the body is pushing up) and the downward phase (when the body is lowering down). We then analyzed the movement by tracking the hip's Y-coordinate to determine which phase the user was in.

- Up Phase Detection:
  In the upward phase, the angle is calculated between the shoulder, hips and knees. If the angle exceeds 200 degrees, it means the back is not straight and the user is prompted to lower their hips and straighten their back.

- Downward Phase Detection:
  In the downward phase, the hips should remain lower than the midpoint between the toes and the ankles. If the hips are lower than this midpoint but the knees are not elevated, the system suggests the user to lift their knees to avoid sagging.

By focusing on the hip's Y-coordinate, we could effectively track and differentiate between the up and downward phases of the push-up to provide real-time feedback

```python
def check_pushup_form(landmarks, frame):
    feedback = "Good Form"
    global prev_hip_y

    # Extract key landmarks
    knee = landmarks[mp_pose.PoseLandmark.LEFT_KNEE.value]
    ankle = landmarks[mp_pose.PoseLandmark.LEFT_ANKLE.value]
    toe = landmarks[mp_pose.PoseLandmark.LEFT_FOOT_INDEX.value]
    hip = landmarks[mp_pose.PoseLandmark.LEFT_HIP.value]
    shoulder = landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value]
    head = landmarks[mp_pose.PoseLandmark.NOSE.value]

    # Push-up phase detection logic
    phase = "Up Phase"
    if prev_hip_y is not None:
        if hip[1] > prev_hip_y:  # Hip moving down
            phase = "Down Phase"
        else:  # Hip moving up
            phase = "Up Phase"
    prev_hip_y = hip[1]

    # Midpoint of ankle and toe
    midpoint_y = (toe[1] + ankle[1]) / 2

    # Check for back straightness (Shoulder-Hip-Knee angle ~ 180°)
    back_angle = calculate_angle(
        a: [shoulder[0], shoulder[1]],  # Shoulder
        b: [hip[0], hip[1]],            # Hip
        c: [knee[0], knee[1]]           # Knee
    )
```

Figure 2.4.1 Check Pushup Form

```python
    # Check the conditions for bad form
    if phase == "Up Phase":
        if knee[1] >= midpoint_y:  # Knees too low
            feedback = "Bad Form: Lift your knees"
        elif back_angle > 20 or back_angle < 0:  # Relaxed threshold for back straightness
            feedback = "Bad Form: Straighten your back"

    elif phase == "Down Phase":
        if knee[1] >= midpoint_y:  # Knees too low
            feedback = "Bad Form: Lift your knees"
        elif back_angle > 20 or back_angle < 0:  # Relaxed threshold for back straightness
            feedback = "Bad Form: Straighten your back"
```

Figure 2.4.2 Pushup phases

Bicep Curls

For bicep curls, we tracked the alignment and movement of key body parts such as the shoulders, elbows, and hips. Specific checks included:

Back Alignment:
We checked whether the shoulder and hip were aligned. If the horizontal distance between the shoulder and hip was more than 20 pixels in either the X or Y coordinates, the system flagged the form as incorrect.

Elbow Distance from Hips:
We measured the horizontal distance between the elbow and the hip. If this distance exceeded 100 pixels, it indicated improper form, and feedback was given to ensure the elbow remained close to the body

Elbow Movement:
The system checked for elbow stability by comparing its position in the current frame with the previous one. If the elbow moved too much (defined as a distance greater than 15 pixels between frames), feedback was provided to ensure the elbow remained stationary.

Elbow, Shoulder, and Wrist Alignment:
We calculated the angle between the shoulder, elbow, and wrist and checked whether it was within the range of 10° to 170°. If the angle was outside this range, feedback was given to adjust the positioning and maintain proper form.

```python
# Function to check bicep curl form
def check_bicep_curl_form(landmarks, frame):
    feedback = "Good Form"
    left_shoulder = landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value]
    left_elbow = landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value]
    left_wrist = landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value]
    left_elbow_angle = calculate_angle(
        a: [left_shoulder[0], left_shoulder[1]],
        b: [left_elbow[0], left_elbow[1]],
        c: [left_wrist[0], left_wrist[1]]
    )
    shoulder_elbow_alignment = abs(left_shoulder[1] - left_elbow[1])
    if left_elbow_angle < 10 or left_elbow_angle > 170:
        feedback = "Bad Form: Incorrect Elbow Angle"
    if shoulder_elbow_alignment > 120:
        feedback = "Bad Form: Shoulder-Elbow Alignment Off"
    cv2.putText(frame, text: f"Elbow Angle: {int(left_elbow_angle)}", org: (10, 100), cv2.FONT_HERSHEY_SIMPLEX, fontScale: 0.5, color: (255, 255, 255), thickness: 1)
    cv2.putText(frame, text: f"Shoulder-Elbow Alignment: {shoulder_elbow_alignment:.2f}", org: (10, 120), cv2.FONT_HERSHEY_SIMPLEX, fontScale: 0.5, color: (255, 255, 255),
    return feedback


# Function to check push-up form
```

Figure 2.4.3 Bicep Curl Alignment

```python
# Get key landmarks
        shoulder = landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value]
        elbow = landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value]
        wrist = landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value]
        hip = landmarks[mp_pose.PoseLandmark.LEFT_HIP.value]
        head = landmarks[mp_pose.PoseLandmark.NOSE.value]  # Nose as head reference

        # Add current elbow position to the buffer
        elbow_position_buffer.append((elbow[0], elbow[1]))

        # Check if the back is straight (i.e., shoulder, hip, and head are in alignment)
        if abs(shoulder[0] - hip[0]) > 20 or abs(shoulder[1] - hip[1]) > 20:
            feedback = "Bad Form: Keep your back straight"

        # Check if the elbow is too far apart from the hips (horizontal distance between elbow and hip)
        if abs(elbow[0] - hip[0]) > 100:  # Tune this threshold based on testing
            feedback = "Bad Form: Keep your elbows closer to your body"

        # Check if the elbow is stationary (compare with previous frame)
        if len(elbow_position_buffer) > 1:
            prev_elbow = elbow_position_buffer[-2]
            # Calculate the distance moved between frames
            distance_moved = np.linalg.norm(np.array([elbow[0] - prev_elbow[0], elbow[1] - prev_elbow[1]]))
            if distance_moved > 15:  # Tune this threshold for allowed movement
                feedback = "Bad Form: Elbow is moving too much"
            else:
                feedback = "Good Form: Elbow is stationary"

        # Calculate the angle at the elbow
        elbow_angle = calculate_angle(shoulder, elbow, wrist)
```

Figure 2.4.4 Bicep Curl Position

Squats

For squats, we tracked the position of the shoulder and knee along the X-axis to ensure proper form during the downward motion:

Knee Alignment
We checked whether the shoulder was moving forward in relation to the knee during the squat. If the X-coordinate of the shoulder moved significantly forward beyond the knee, it indicated improper form, and the system suggested correcting the posture to avoid excess forward shoulder movement. Also, if the knee's X-coordinate moved ahead of the toes the user is prompted accordingly. Another point to mention is that the back angle is calculated to ensure the user does not round their back.

By analyzing these specific body movements and their alignment, we could provide actionable feedback to the user, ensuring they performed the exercises with proper form. This feedback helped users to not only perform exercises effectively but also to minimize the risk of injury.

```python
def check_squat_form(landmarks, frame):
    knee_toe_distance = knee_x - toe_x

    # Standing position
    if back_angle < 50 and hip_knee_ankle_angle < 30:
        feedback = "Good Form: Standing Position"

    # Shoulders ahead of knees
    elif shoulder_x > knee_x + 20:  # Correct logic here
        feedback = "Bad Form: Shoulders Ahead of Knees"

    # Squatting position logic
    elif 90 <= hip_knee_ankle_angle <= 160:
        if back_angle < 60:
            feedback = "Bad Form: Keep Back Straight"
        elif knee_toe_distance < -20:
            feedback = "Bad Form: Knees Past Toes"
        else:
            feedback = "Good Form: Squatting"

    # Deep squat logic
    elif hip_knee_ankle_angle > 160:
        if back_angle < 60:
            feedback = "Bad Form: Keep Back Straight"
        elif knee_toe_distance < -20:
            feedback = "Bad Form: Knees Past Toes"
        else:
            feedback = "Good Form: Deep Squat"
```

Figure 2.4.5 Squats

CHAPTER 3: DEPLOYMENT

3.1. DEPLOYMENT WITH STREAMLIT

To make the exercise form analysis accessible and user-friendly, we deployed the solution locally using Streamlit, a framework for creating interactive web applications with minimal effort. Streamlit allowed us to create an intuitive interface that could run the pose estimation model in real-time and provide immediate feedback to users on their form.

Steps Taken for Deployment

1. Setting Up the Environment:

   We installed the necessary libraries, including Streamlit, MediaPipe, and OpenCV, to handle pose estimation and video streaming. The Streamlit application was designed to take webcam input, process it frame-by-frame for pose estimation, and then display feedback on the user's form.

2. Webcam Integration:

   We used OpenCV to capture real-time video from the user's webcam. The frames from the video were processed through the pose estimation pipeline to detect key body landmarks, which were then used for form analysis.

3. Pose Estimation and Feedback Display:

   Once the body landmarks were detected using MediaPipe Pose, we performed the form analysis as described in the previous section. Based on the analysis, feedback was displayed directly on the Streamlit interface, indicating whether the user was performing the exercise correctly or needed to adjust their form.

4. User Interface Design:

   The user interface was designed to be simple and intuitive, with clear instructions and feedback. Streamlit's layout features such as text boxes, buttons, and image display were used to present real-time analysis and suggestions to the user. The real-time feedback appeared as text overlays on the webcam stream, alerting the user if their form was incorrect or if they needed to make adjustments.

5. Real-time Feedback:

   Streamlit allowed us to provide immediate feedback as users performed exercises. The system continuously analyzed the user’s form during the exercise and provided corrective suggestions, ensuring that the feedback loop was quick and responsive.

6. Local Deployment:

After developing the application, we deployed it locally. Users could run the application on their own machines by starting the Streamlit app from the command line, opening the interface directly in their browser. This allowed users to access and interact with the tool without needing an internet connection.

```python
# Streamlit App
st.title("Real-Time Exercise Feedback")
st.sidebar.title("Settings")
video_source = st.sidebar.selectbox("Select Video Source", ("DroidCam USB", "Webcam", "Upload Video"))
if video_source == "DroidCam USB":
    st.sidebar.write("Ensure DroidCam is running and connected via USB.")
    cap = cv2.VideoCapture(1)
elif video_source == "Webcam":
    cap = cv2.VideoCapture(0)
elif video_source == "Upload Video":
    uploaded_file = st.sidebar.file_uploader("Upload a Video", type=["mp4", "avi"])
    if uploaded_file:
        temp_file = "temp_video.mp4"
        with open(temp_file, "wb") as f:
            f.write(uploaded_file.read())
        cap = cv2.VideoCapture(temp_file)

if st.sidebar.button("Start"):
    if 'cap' in locals() and cap and cap.isOpened():
        st_frame = st.empty()
        while cap.isOpened():
            ret, frame = cap.read()
            if not ret:
                break
            results = model(frame)
            if len(results[0].boxes):
                sorted_boxes = sorted(results[0].boxes, key=lambda x: x.conf, reverse=True)
                class_id = int(sorted_boxes[0].cls)
                exercise_type = {0: 'bicep curl', 1: 'push-up', 2: 'squat'}.get(class_id, 'Unknown')
            else:
```

Figure 3.1.1 Streamlit Code

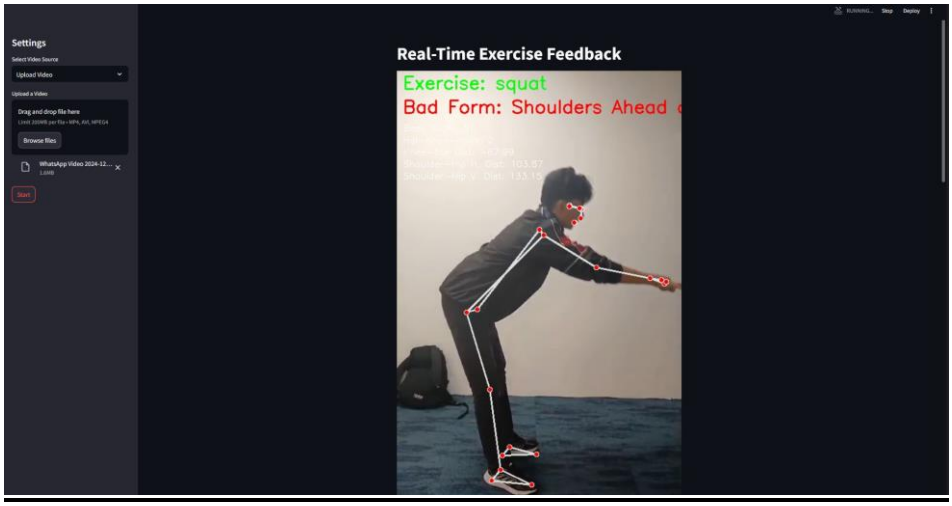

Figure 3.1.2 Streamlit Interface

CHAPTER 4: IMPLEMENTATION

The Streamlit-based Real-Time Exercise Feedback System features a clean and interactive User Interface (UI) designed for ease of use. The UI consists of a sidebar panel and a main display area. In the sidebar, users can configure their video input source, choosing between options like "DroidCam USB", "Webcam", or "Upload Video", with clear instructions for each. Users can upload pre-recorded video files directly through a file uploader for analysis. A prominent "Start" button triggers the real-time exercise feedback process.

The main display area serves as the video output screen where the user's live or uploaded video is displayed, overlaid with pose landmarks and visual feedback annotations. Key metrics such as joint angles (e.g., elbow, back), phase detection (e.g., "Up Phase" or "Down Phase"), and specific form corrections (e.g., "Bad Form: Straighten your back") are dynamically displayed in real-time using text overlays. Positive feedback like "Good Form" appears in green, while corrections appear in red for quick recognition.

4.1 WEBCAM, VIDEO UPLOAD AND DROID CAM FUNCTIONALITY

The DroidCam USB integration turns your phone into a webcam by connecting it to your computer via a USB cable. With the DroidCam app on your phone and the DroidCam client on your PC, the phone streams its camera feed directly as an external webcam. In the app, cv2.VideoCapture(1) tells OpenCV to use the DroidCam feed (registered as Camera Index 1) instead of the default webcam. This USB connection ensures a stable, lag-free stream, which is processed in real-time for pose detection and exercise feedback. It's a simple, reliable way to use your phone as a high-quality webcam.

The Webcam and Video Upload functionality provides two flexible ways for users to interact with the system:

1. Webcam Integration:
   By selecting the "Webcam" option, the app uses your computer's built-in or external webcam for real-time video streaming. OpenCV's cv2.VideoCapture(0) grabs the default webcam feed, which is processed frame-by-frame to analyze exercises and provide live feedback. This ensures an interactive, real-time experience for immediate posture correction.

2. Video Upload:
   For users who prefer offline analysis, the app allows video uploads in formats like MP4 or AVI. When a video is uploaded, it's temporarily saved and loaded using OpenCV for frame-by-frame processing. This approach is ideal for reviewing recorded workouts, enabling users to get feedback at their own pace without requiring a live camera

CHAPTER 5: RESULTS AND COMPARISON

5.1. MODEL PERFORMANCE AND COMPARISON

epoch	time	train/box_loss	train/cls_loss	train/dfl_loss	metrics/precision(B)	metrics/recall(B)	metrics/mAP50(B	metrics/mAP50-9	val/box_loss	val/cls_loss	val/dfl_loss	lr/pg0	lr/pg1	lr/pg2
50	23936.7	0.1169	0.08678	0.85346	0.99183	0.8969	0.93373	0.92731	0.13304	0.10667	0.77475	0.000298	0.000298	0.000298

Figure 5.1.1 Metrics

Precision and Recall

The 99% precision reflects that the model has an exceptional ability to correctly classify positive predictions, meaning most of the predicted exercises (bicep curls, push-ups, squats) are accurate. However, the 89% recall indicates there is a slight drop in correctly identifying all true instances. This imbalance suggests that while false positives are rare, the model sometimes misses true positives. For example, certain classes like "standing" may be harder to detect, as seen in the confusion matrix where the "standing" category has fewer correct predictions and some misclassifications.

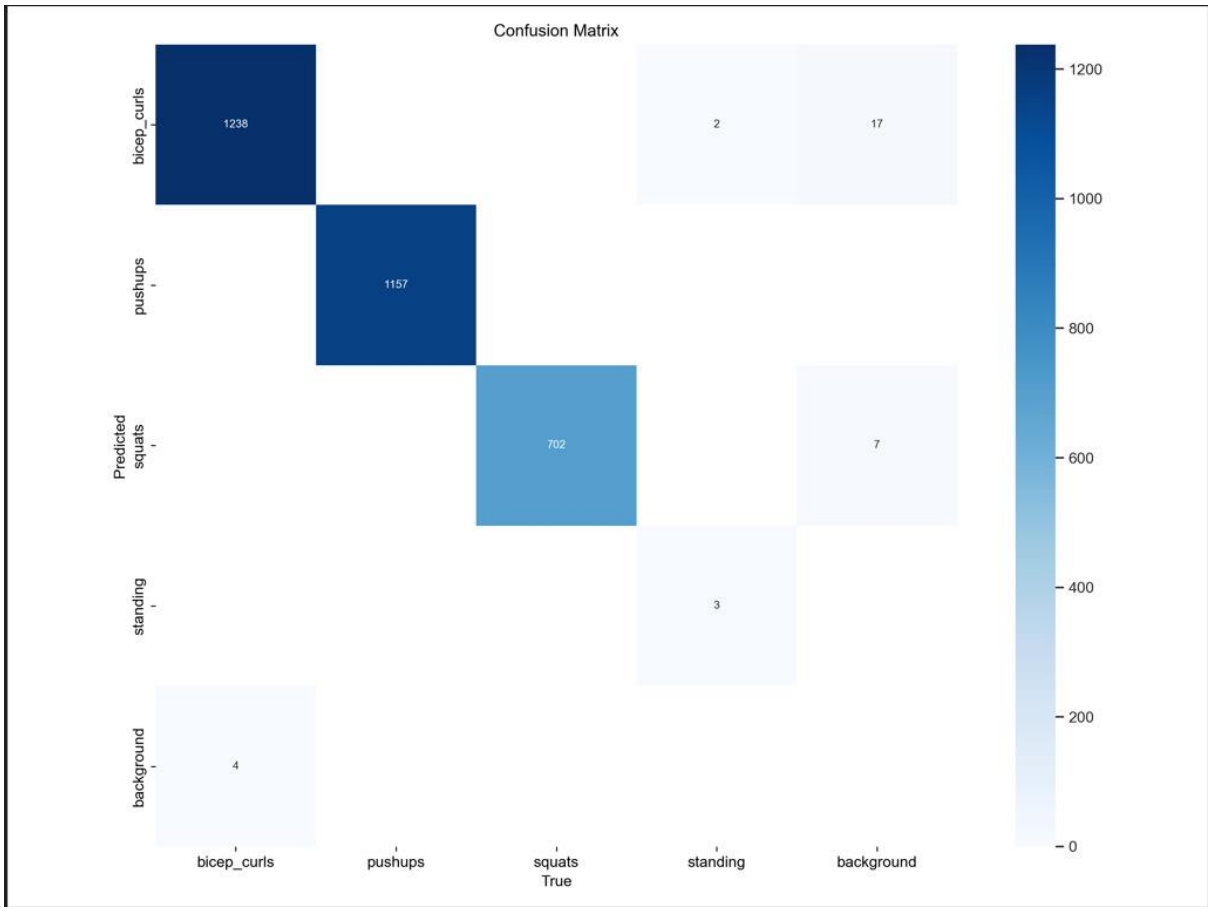

Figure 5.1.2 Confusion Matrix

The matrix highlights strong predictions for bicep curls (1238 correct) and push-ups (1157 correct) but shows confusion with "standing" and background categories, which have lower recall.

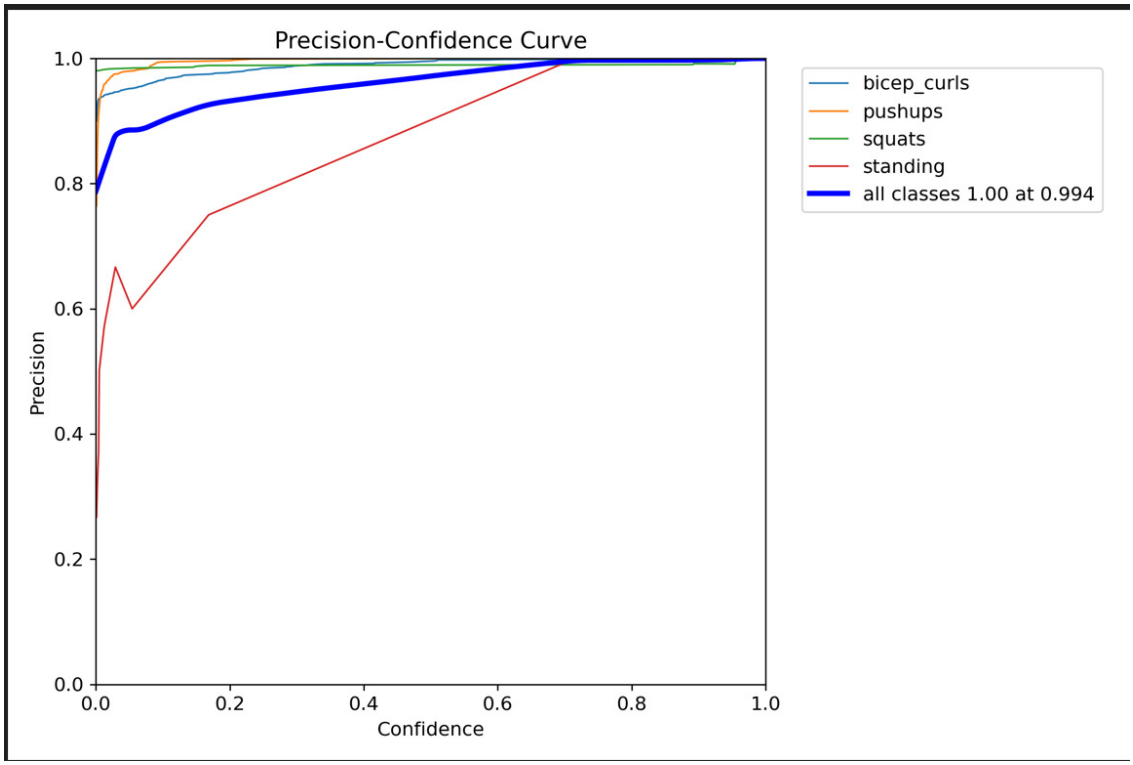

Figure 5.1.3 Precision-Confidence Curve

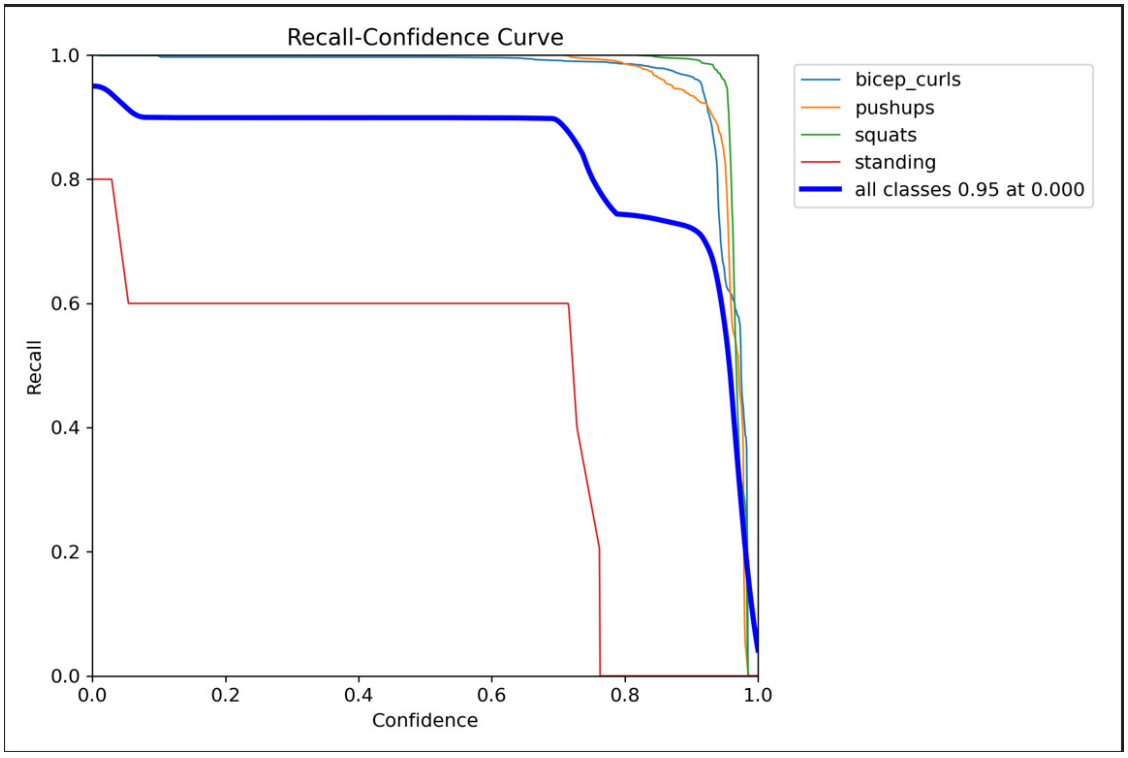

Figure 5.1.4 Recall-Confidence Curve

Precision is consistently high across confidence levels for most classes, except for "standing," which shows instability at lower confidence thresholds. The exercise classes achieve near-perfect recall at higher confidence thresholds.

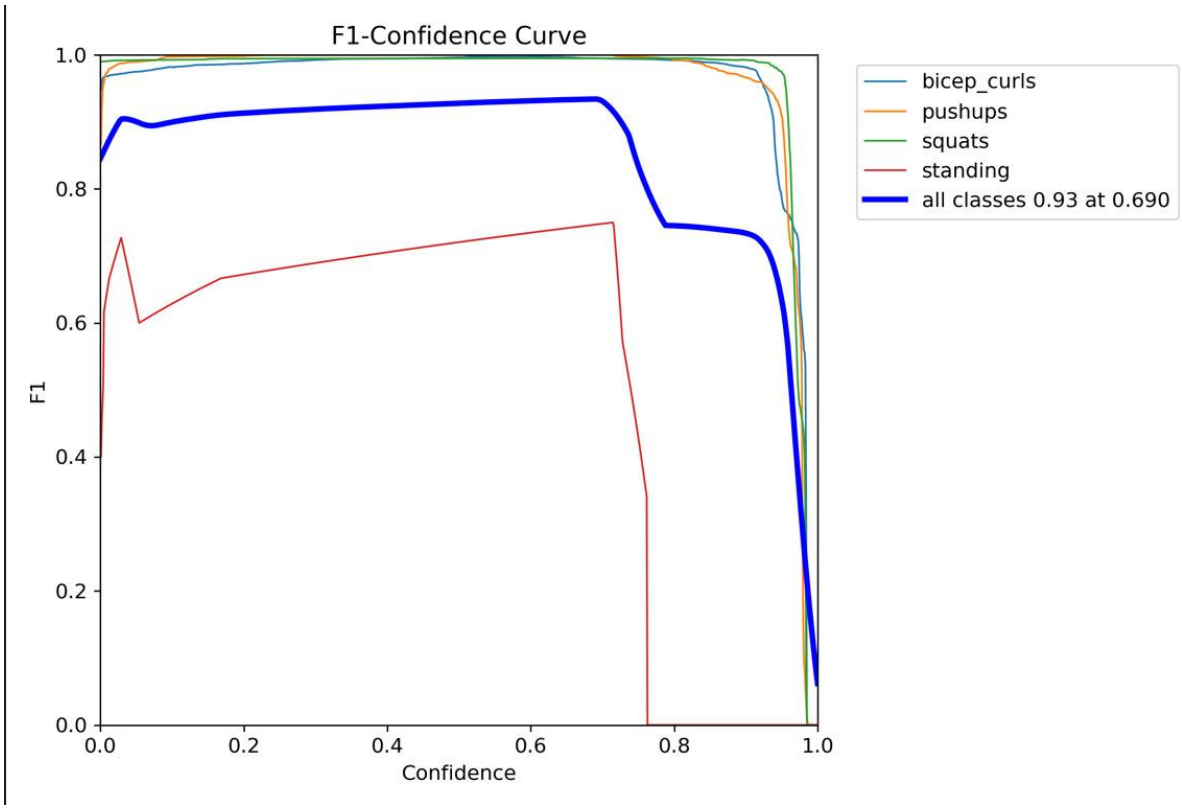

Figure 5.1.5 F1-Confidence Curve

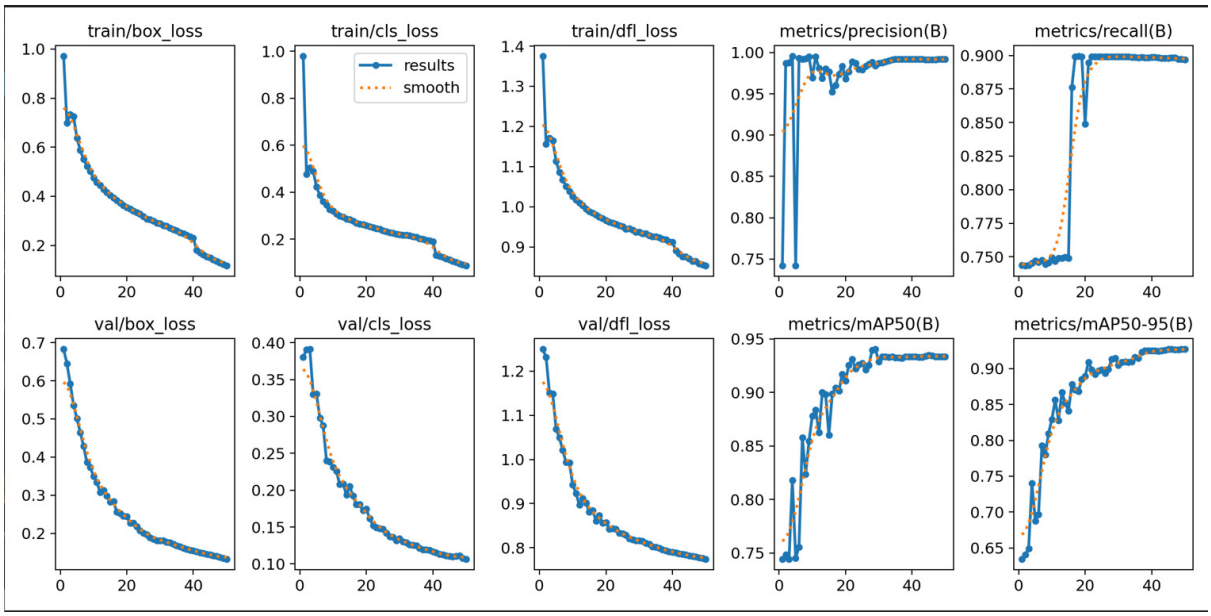

Figure 5.1.6 Loss Graphs

The F1 score remains strong for bicep curls, push-ups, and squats, highlighting excellent performance. Training and validation losses decrease steadily, indicating good convergence without overfitting. The mAP50 (93.4%) and mAP50-95 (92.7%) confirm that the model generalizes well.

Overall, the model achieves high precision and recall for all classes, with minor weaknesses in differentiating "standing" and background classes. This highlights excellent overall performance.

"Human pose estimation and action recognition for fitness movements" by Huichen Fu et al. (2023) achieved a mean average precision (mAP) of 91.7% on the authors' custom fitness action dataset. While the improved YOLOv7-Pose in the paper achieves a mAP of 95.9% for fitness actions after certain enhancements, our model boasts a 99% precision and 89% recall. These metrics indicate that our model is highly effective at correctly identifying correct forms and classifying exercises(precision) while maintaining a competitive ability to recognize true positives (recall), even in a real-time, noisy environment.

Our method classifies the exercise instead of making the user input it. This saves time and allows the user to focus on seeing if the exercise they are performing is right or wrong. We also made sure to work on at least 3 exercises, making our model a better option than others that only work on one or two exercises. We have also enabled support for phone users who wish to use their phone cameras instead.

CHAPTER 6: CONCLUSION

6.1. SUMMARY OF ACHIEVEMENTS

In conclusion, this project successfully delivers a real-time exercise feedback system that integrates YOLO object detection, MediaPipe Pose estimation, and a user-friendly Streamlit interface. With support for webcam streaming, DroidCam USB integration, and video uploads, the system provides flexibility for users to analyze their workouts both live and through recorded sessions. By accurately identifying exercises like push-ups, squats, and bicep curls, and offering instant form corrections, the application promotes proper posture and reduces the risk of injury. The visual feedback, combined with key pose angles and motion tracking, enhances user engagement and ensures clear, actionable insights for improving exercise form. This system represents a significant step towards leveraging AI for fitness monitoring, bridging the gap between technology and personal fitness coaching.

6.2. POTENTIAL ENHANCEMENTS

Firstly, there are the several system features which further can be developed to incorporate other order workouts like lunges, planks, deadlifts. Real-time voice feedback can also be implemented using text-to-speech providing immediate verbal corrections, so users don't have to focus on the screen. In order to track the progress of the user, features such as automated repetition counters, performance analysis and progress reports can help the user to see the improvements which have been made. In addition, wearable device integration can help in recording heart rate, calorie burnt and activity time to give the user a comprehensive picture of his/her fitness.

Deploying the system as a **mobile app** for iOS and Android would increase accessibility and convenience, enabling users to exercise on the go. The performance data could be stored in the cloud and the results could be exported to spreadsheet programs to track results over the months. Also, providing options for the customization of the thresholds for form analysis would allow the more experienced users or the trainers to set up certain standards for certain exercises. With these enhancements the project can become an intelligent human fitness companion and a more complex application.

REFERENCES

[1] Chen, Steven & Yang, Richard. (2018). Pose Trainer: Correcting Exercise Posture using Pose Estimation. 10.48550/arXiv.2006.11718.

[2] Z. Cao, T. Simon, S. Wei, and Y. Sheikh. Realtime multiperson 2d pose estimation using part affinity fields. In CVPR, 2017.

[3] Kotte, Hitesh & Kravcik, Milos & Duong-Trung, Nghia. (2023). Real-Time Posture Correction in Gym Exercises: A Computer Vision-Based Approach for Performance Analysis, Error Classification and Feedback.