UNIVERSITY of INFORMATION TECHNOLOGY

CST-8316
Play Vision
Football Player Tracking

Team Members

Hein Htet Kyaw	(4SE-1101)
Si Thu Aung	(4SE-1114)
Phyo Than Thar Kyaw	(4SE-1016)
Thiri Shwe Sin	(4SE-1145)
Su Htet Thinzar	(4SE-1094)
Khin Bhone Pyae	(4SE-1451)

(Software Engineering Group 2)

September 10, 2024

Abstract of the Project

Football analysis has traditionally relied on manual methods that are labor-intensive and often prone to missing key details. As the demand for real-time, data-driven insights grows, there is an increasing need for automated systems that can analyze matches with greater accuracy and efficiency. This project applies computer vision and machine learning techniques to detect football players, track their movements, and provide real-time analysis of game metrics such as ball possession, player speed, and team dynamics. By utilizing a custom-trained YOLO model for player detection and advanced data analysis for performance evaluation, the system delivers actionable insights to coaches, analysts, and fans. The project aims to enhance understanding of game strategies, improve player performance, and make tactical decision-making more informed, with future potential for real-time implementation.

Table of Contents

1. Introduction

Football is one of the most popular sports globally, with massive viewership and increasing interest in analytics for performance improvement. Traditional football analysis is labor-intensive and often misses details that AI systems can detect. The demand for real-time insights to help teams strategize is growing. The goal is to apply computer vision and machine learning to provide detailed, real-time analysis of football matches, enhancing player and team performance metrics.

1.1. Project Objectives

Purpose: To develop an automated system that detects football players, tracks their movements, and analyzes key game metrics.

Main Objectives:

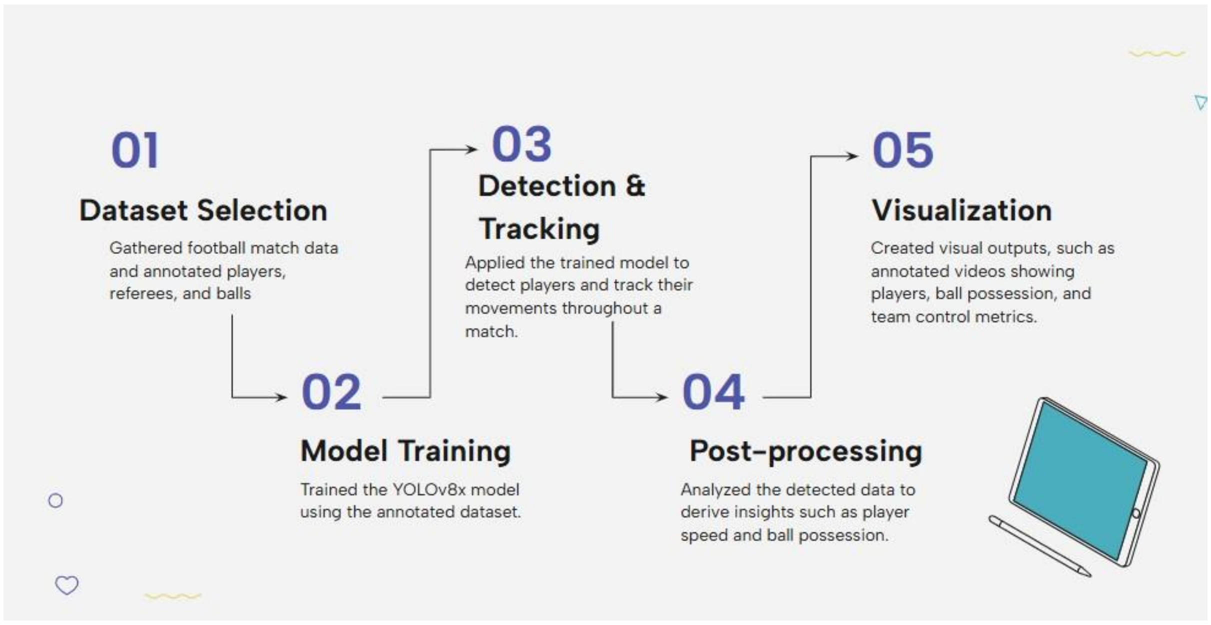
- **Player Detection**: Automatically detect all players and categorize them based on team affiliation.
- **Tracking**: Monitor and track player positions throughout the game to assess movement patterns.
- **Analysis**: Calculate key metrics such as ball possession, speed, and player performance during the game.
- **Visualization**: Display these metrics in an easy-to-understand format that coaches, analysts, and fans can use to improve game understanding.

1.2.Project Scope

This project aims to bridge computer vision with data analysis to provide a detailed study of football match dynamics. By detecting and tracking player movements and ball trajectories, the project delves into understanding team strategies and individual player performances. The scope covers the entire process from data collection to visualization, focusing on generating meaningful insights that could help in match analysis, tactical decision-making, and performance evaluation.

- **Core Focus**: The project focuses on tracking football players and the ball during matches. Key areas of analysis include player speed, distance covered, ball possession, and team dynamics.
- **Technologies Used**: Computer vision techniques, specifically a custom-trained YOLO model, are utilized for player and ball detection. Data analysis and modeling provide insights from match data.
- **Limitations**: The analysis is based on limited match data, and the scope is restricted to post-game analysis rather than real-time processing. Tactical analysis is basic, without external factors like weather.
- **Deliverables**: The project will produce visualizations in a video format, with key metrics and annotations. Future work could extend to advanced analytics and real-time application.

2. Overview of the project


01
Dataset Selection
Gathered football match data and annotated players, referees, and balls
02
Model Training
Trained the YOLOv8x model using the annotated dataset.
03
Detection & Tracking
Applied the trained model to detect players and track their movements throughout a match.
04
Post-processing
Analyzed the detected data to derive insights such as player speed and ball possession.
05
Visualization
Created visual outputs, such as annotated videos showing players, ball possession, and team control metrics.

3. Understanding the Dataset

Source: **Roboflow**

Our dataset is focused on detection of players, goalkeepers, referees and ball. It was obtained from Roboflow, which provides annotated images and videos capturing various aspects of a football match.

Here is a brief overview of the attributes within the dataset:

- **Ball**: The spherical object used in football, which players aim to move into the opponent's goal
- **Player**: An individual participating in the match, contributing to gameplay through various roles and actions.
- **Goalkeeper**: The player positioned near the goal, responsible for preventing the ball from entering the net.
- **Referee**: The official who oversees the match, enforces the rules, and ensures fair play.

4. Data Preparation

4.1 Dataset Selection

The selection of an appropriate dataset is critical for the success of any machine learning project. For this project, we focused on datasets specifically tailored for football-related object detection tasks. The dataset selection process involved the following key considerations:

Relevance to the Project Goals: The chosen dataset needed to align with the specific objectives of our project, including detecting football players, referees, the ball, and goalkeepers. We prioritized datasets that contained these elements across various scenarios and conditions to ensure comprehensive training and evaluation.

Diversity and Completeness: To build a robust model, it was essential to select a dataset with a diverse range of images and videos. This diversity includes different lighting conditions, camera angles, and game scenarios, ensuring that the model could generalize well across various real-world situations. The dataset should also be complete, with adequate annotations for each class to provide a strong foundation for training and evaluation.

Quality of Annotations: High-quality, accurate annotations are crucial for effective model training. The selected dataset included precise bounding box annotations and class labels for the ball, goalkeepers, players, and referees. Ensuring the quality of these annotations was a key factor in achieving reliable and accurate detection results.

Dataset Size: The size of the dataset impacts the model's performance and ability to generalize. A sufficiently large dataset was selected to provide enough samples for training, validation, and testing phases. This large dataset helps in mitigating overfitting and ensures that the model performs well on unseen data.

4.2 Dataset Split

The dataset was divided into three subsets to facilitate effective model training, validation, and testing:

- Training Set (80%): Consists of 298 images, used to train the model and learn object features.
- Validation Set (13%): Comprises 49 images, used to tune hyperparameters and assess model performance during training.
- Test Set (7%): Includes 25 images, used to evaluate the final model’s performance and ensure its generalization.

5. Building the model Using YOLO

5.1 Selecting the appropriate model

To achieve accurate object detection for football scenes, **the YOLO** (You Only Look Once) algorithm was utilized. YOLO is known for its efficiency and effectiveness in real-time object detection. There are 5 variants of the YOLO model-YOLOv8n, YOLOv8s, YOLOv8m, YOLOv8l, YOLOv8x. We choose YOLOv8x for the following reasons.

Model Selection**: YOLOv8x**

- Enhanced Accuracy: YOLOv8x offers superior detection precision, ideal for identifying small or partially occluded objects like players and the ball.
- Advanced Architecture: Its state-of-the-art neural network design improves feature extraction and detection performance.
- Real-Time Performance: Despite its complexity, YOLOv8x maintains high inference speed, suitable for processing live football footage.
- Community Support: Extensive research backing and community resources provide valuable support for implementation and fine-tuning.

5.2 Training the Model

To train the YOLOv8x model with the dataset, the following steps were followed:

1. **Dataset Preparation:**

Download Dataset: The dataset was obtained from Roboflow using their API. The dataset includes labeled images for training, validation, and testing. Here's the code used to download the dataset:

```
from roboflow import Roboflow
rf = Roboflow(api_key="1VNDCZygnqXz1qy3AJKw")
project = rf.workspace("roboflow-jvuqo").project("football-players-detection-
3zvbc")
version = project.version(12)
dataset = version.download("yolov8")
```

2. Setting Up Google Colab:

Environment Preparation: Google Colab was used for training due to its support for GPU acceleration. The Colab environment was set up with the necessary libraries and tools.

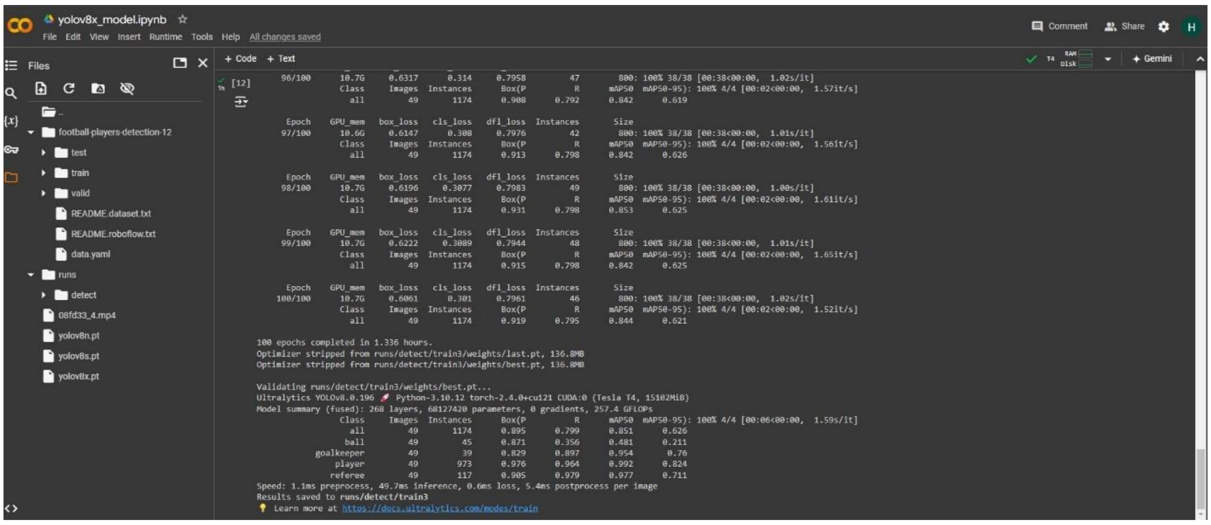
3. Training the Model:

Load YOLOv8x Model: The YOLOv8x model was configured and initialized. This model is known for its enhanced detection capabilities and was chosen for its superior performance in object detection tasks.

Run Training: The model was trained using the following command:

```
!yolo task=detect mode=train model=yolov8x.pt data={dataset.location}/data.yaml epochs=100
imgsz=640
```

4. Training Progress

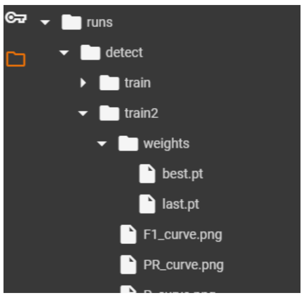


```
     96/100      10.7G     0.6317      0.314     0.7958         47        800: 100% 38/38 [00:38<00:00,  1.02s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 4/4 [00:02<00:00,  1.57it/s]
                   all         49       1174      0.908      0.792      0.842      0.619

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     97/100      10.6G     0.6147      0.308     0.7976         42        800: 100% 38/38 [00:38<00:00,  1.01s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 4/4 [00:02<00:00,  1.56it/s]
                   all         49       1174      0.913      0.798      0.842      0.626

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     98/100      10.7G     0.6196     0.3077     0.7983         49        800: 100% 38/38 [00:38<00:00,  1.00s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 4/4 [00:02<00:00,  1.61it/s]
                   all         49       1174      0.931      0.798      0.853      0.625

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     99/100      10.7G     0.6222     0.3089     0.7944         48        800: 100% 38/38 [00:38<00:00,  1.01s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 4/4 [00:02<00:00,  1.65it/s]
                   all         49       1174      0.915      0.798      0.842      0.625

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    100/100      10.7G     0.6061      0.301     0.7961         46        800: 100% 38/38 [00:38<00:00,  1.02s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 4/4 [00:02<00:00,  1.52it/s]
                   all         49       1174      0.919      0.795      0.844      0.621

100 epochs completed in 1.336 hours.
Optimizer stripped from runs/detect/train3/weights/last.pt, 136.8MB
Optimizer stripped from runs/detect/train3/weights/best.pt, 136.8MB

Validating runs/detect/train3/weights/best.pt...
Ultralytics YOLOv8.0.196 🚀 Python-3.10.12 torch-2.4.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 268 layers, 68127420 parameters, 0 gradients, 257.4 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 4/4 [00:06<00:00,  1.59s/it]
                   all         49       1174      0.895      0.799      0.851      0.626
                  ball         49         45      0.871      0.356      0.481      0.211
            goalkeeper         49         39      0.829      0.897      0.954       0.76
                player         49        973      0.976      0.964      0.992      0.824
               referee         49        117      0.905      0.979      0.977      0.711
Speed: 1.1ms preprocess, 49.7ms inference, 0.6ms loss, 5.4ms postprocess per image
Results saved to runs/detect/train3
💡 Learn more at https://docs.ultralytics.com/modes/train
```

Figure 1: Model Training with Google Colab

5. Model Saved

After training, the model was saved and the trained model file was ready for use. The final saved model can be seen in the following screenshot



```
▾ runs
  ▾ detect
    ▸ train
    ▾ train2
      ▾ weights
          best.pt
          last.pt
        F1_curve.png
        PR_curve.png
```

Figure 2: After training the model with dataset, the model "best.pt" saved

6. Results of the model

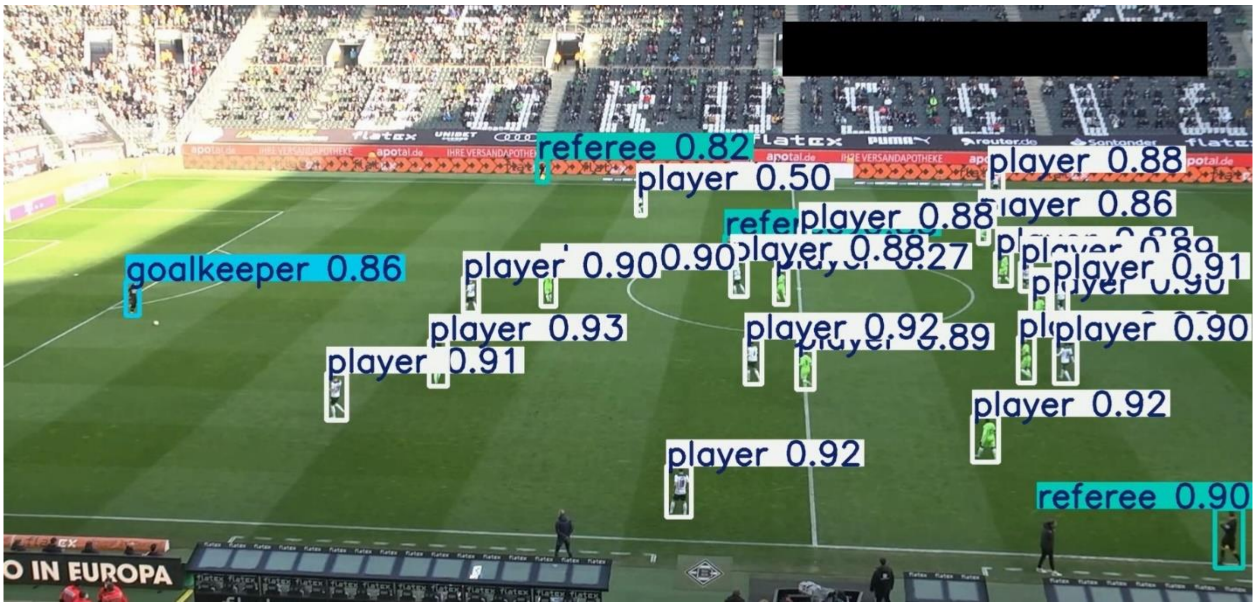

[redacted]

Figure 3: Results of the trained model

6. Post-processing and Analysis

6.1 Team Assignment

Objective: The objective of team assignment is to accurately identify and assign players to their respective teams based on visual features extracted from their images. This is crucial for subsequent analysis, such as tracking player performance and calculating team ball possession.

1. **Player Detection and Segmentation**

   Model Used: The YOLOv8x object detection model is employed to detect players within each frame of the video. The model identifies players by providing bounding boxes around them.

   Bounding Box Extraction: For each detected player, the model outputs a bounding box that encloses the player in the image. This bounding box is used to extract the player's image from the video frame.

2. **Color Clustering for Team Assignment**

   Region of Interest (ROI): To determine the team color, focus is placed on the top half of the player's bounding box. This region generally contains the most prominent color of the jersey.

   K-Means Clustering

   Procedure: Apply K-means clustering to the color data extracted from the top half of the bounding box. K-means clustering groups pixel colors into clusters, helping to identify the dominant color in the jersey area.

   Number of Clusters: Typically, K-means is configured to find 1 or 2 clusters (depending on whether the jersey has multiple colors) to identify the most prevalent color.

   Color Assignment

   Color Extraction: Extract the color that forms the largest cluster from the K-means result. This color is used to represent the player's team.

   Mapping to Teams: Map the extracted color to predefined team colors. This mapping is based on known team colors for the match.

3. **Code Implementation**

The following code demonstrates how to extract the top half of a player's bounding box and apply K-means clustering to identify the dominant color.

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import KMeans

image_path = "../output_videos/cropped_image.jpg"
image = cv2.imread(image_path)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

plt.imshow(image)
plt.show()
```

Here is the cropped image:

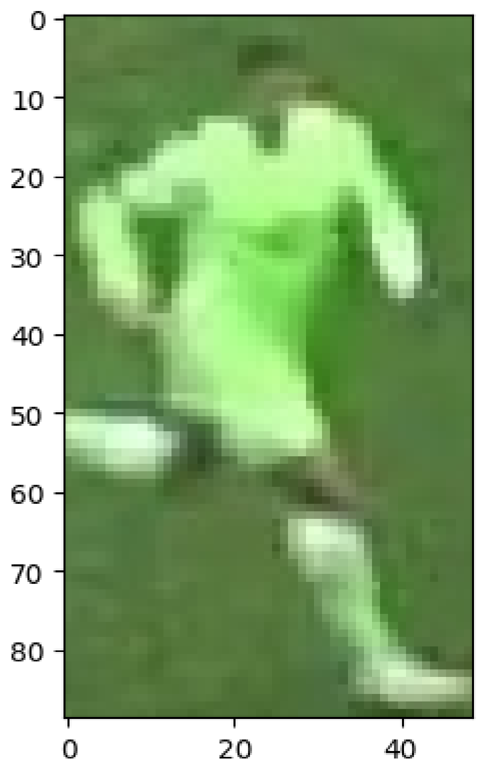


Figure 4: Cropped Image

Take the top half of the image:

```
top_half_image=  image[0: int(image.shape[0]/2), :]
plt.imshow(top_half_image)
plt.show()
```

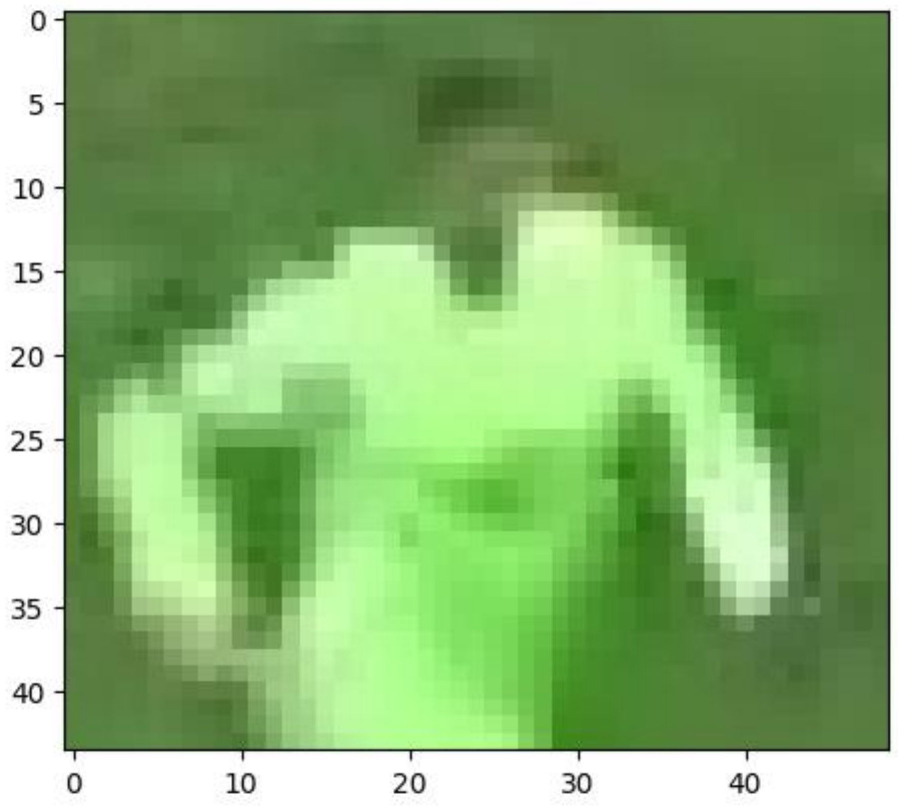


Figure 5: Top Half of the player image

Cluster the image into two clusters

```
# Reshape the image into 2d array
image_2d = top_half_image.reshape(-1, 3)

# perform k-means clustering with 2 clusters
kmeans = KMeans(n_clusters=2, random_state=0)
kmeans.fit(image_2d)

# get the cluster labels
labels = kmeans.labels_

# reshape the labels into the orginal image shape
clustered_image = labels.reshape(top_half_image.shape[0],
top_half_image.shape[1])


# Display the clustered image
plt.imshow(clustered_image)
plt.show()
```

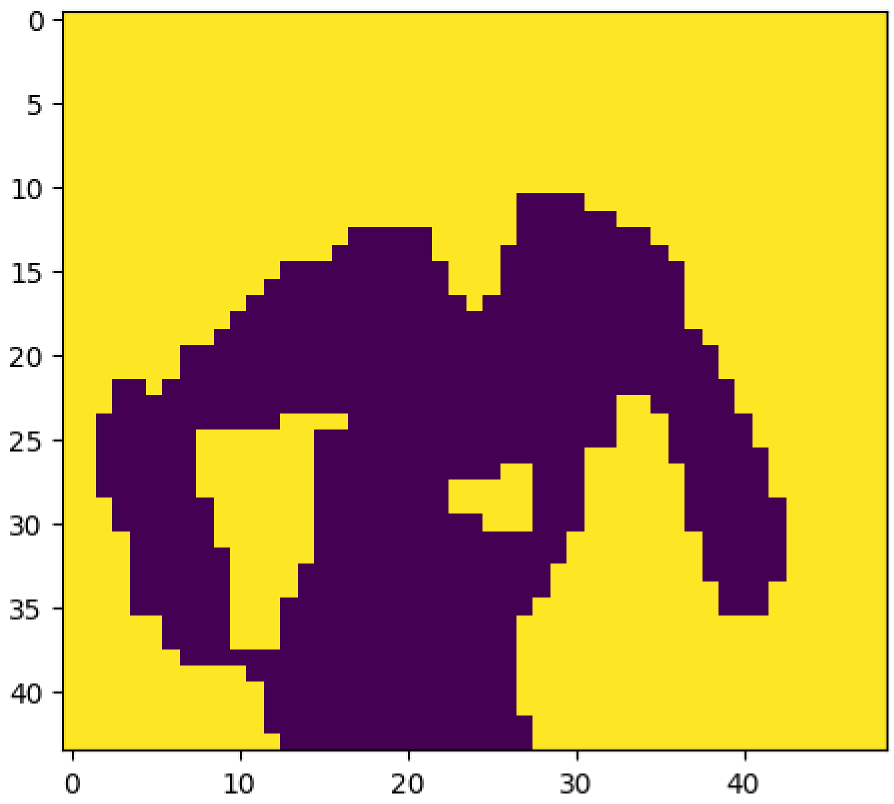


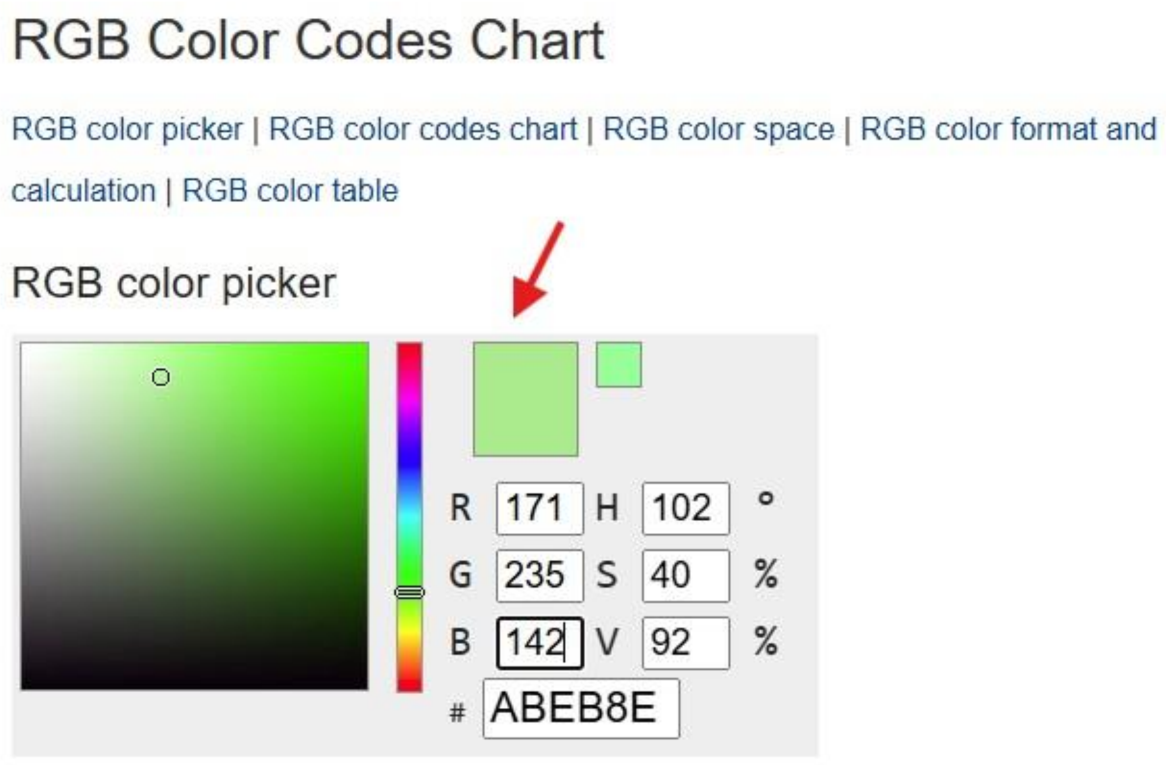
Figure 6: Clustered Image

```
corner_clusters = [clustered_image[0, 0], clustered_image[0, -1],
clustered_image[-1, 0], clustered_image[-1, -1]]

non_player_cluster = max(set(corner_clusters), key=corner_clusters.count)

# Assuming 2 clusters, player cluster is the other one
player_cluster = 1 - non_player_cluster


# Retrieve the dominant color for the player
player_color = kmeans.cluster_centers_[player_cluster]

return player_color
```

This is the result from the code above:

array([171.38378378, 235.65405405, 142.8472973])

To check for player color, we entered the results into the RGB color picker, here is the result.


RGB Color Codes Chart

RGB color picker | RGB color codes chart | RGB color space | RGB color format and calculation | RGB color table

RGB color picker

R 171 H 102 °
G 235 S 40 %
B 142 V 92 %
ABEB8E

Figure 7: Inserting results into RGB color picker and check player's color

6.2 Player Analysis

Player analysis is a critical component of football analytics, providing insights into individual player performance, physical exertion, and tactical movements. By tracking player movements, we can calculate essential metrics like speed and distance covered during a match. This data can help teams optimize their strategies, monitor player fitness, and make informed decisions on substitutions and training regimens.

The player analysis in this project focuses on **two main aspects**:

Player Tracking: This involves detecting and identifying players in each frame of the video, maintaining their identity across multiple frames, and determining their precise location on the pitch. This process allows us to monitor how players move and interact during the game.

Speed and Distance Calculation: Once players are tracked, we can calculate their speed by measuring the distance they cover between frames and dividing this by the time elapsed. The total distance covered by each player throughout the match is also calculated, which can be used to assess physical performance.

Player Tracking

Player tracking was implemented using the YOLOv8x model for object detection, which identified players on the pitch. The detections were tracked across video frames using a custom tracking algorithm. This algorithm maintained the identity of each player across frames, ensuring that we could monitor individual player movements throughout the game.

The tracking algorithm works by assigning a unique ID to each detected player and then matching this ID with the same player in subsequent frames based on position and bounding box similarity. This process allows us to create a continuous trajectory for each player, which is necessary for calculating their speed and distance covered.

Speed and Distance Calculation

Once the players are tracked, we calculate their speed and distance covered. Speed is calculated by measuring the distance a player moves between two frames and dividing it by the time interval between the frames. This gives us the player's speed in meters per second, which is then converted to kilometers per hour.

The total distance covered by each player is calculated by summing the distances between all frames in which the player appears. This gives us a measure of how much ground a player has covered during the match.

```
def add_speed_and_distance_to_tracks(self, tracks):
    total_distance = {}
    #------codes
   start_position = object_tracks[frame_num][track_id]['position_transformed']
    end_position = object_tracks[last_frame][track_id]['position_transformed']

     if start_position is None or end_position is None:
     continue
     distance_covered = measure_distance(start_position, end_position)
     time_elapsed = (last_frame - frame_num) / self.frame_rate
     speed_meteres_per_second = distance_covered / time_elapsed
     speed_km_per_hour = speed_meteres_per_second * 3.6
    #--------codes

     for frame_num_batch in range(frame_num, last_frame):
     if track_id not in tracks[object][frame_num_batch]:
     continue
     tracks[object][frame_num_batch][track_id]['speed'] = speed_km_per_hour
     tracks[object][frame_num_batch][track_id]['distance'] =
total_distance[object][track_id]
```

The function **add_speed_and_distance_to_tracks** calculates the distance a player covers between two frames and converts it into speed. The total distance covered is accumulated across the entire match.

Visualization of Player Metrics

The calculated speed and distance values are overlaid on the video frames during playback, providing real-time feedback on player performance. This visualization allows us to see not only where a player is on the pitch but also how fast they are moving and how much distance they have covered.

The following code handles the display of this information:

```
def draw_speed_and_distance(self, frames, tracks):
    output_frames = []
    for frame_num, frame in enumerate(frames):
        for object, object_tracks in tracks.items():
            if object == "ball" or object == "referees":
                continue
            for _, track_info in object_tracks[frame_num].items():
                if "speed" in track_info:
                    speed = track_info.get('speed', None)
                    distance = track_info.get('distance', None)
                    if speed is None or distance is None:
                        continue
                    bbox = track_info['bbox']
                    position = get_foot_position(bbox)
                    position = list(position)
                    position[1] += 40
                    position = tuple(map(int, position))
                    cv2.putText(frame, f"{speed:.2f} km/h", position,
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 2)
                    cv2.putText(frame, f"{distance:.2f} m", (position[0],
position[1] + 20), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 2)
        output_frames.append(frame)
    return output_frames
```

In this function, the speed and distance values are displayed near the player's feet on each frame. The text displays the player's speed in kilometers per hour and the total distance covered in meters.

Visual Examples

Below are screenshots from the video showing speed and distance annotations for different players. These visualizations help in assessing individual player performance over the course of the match.

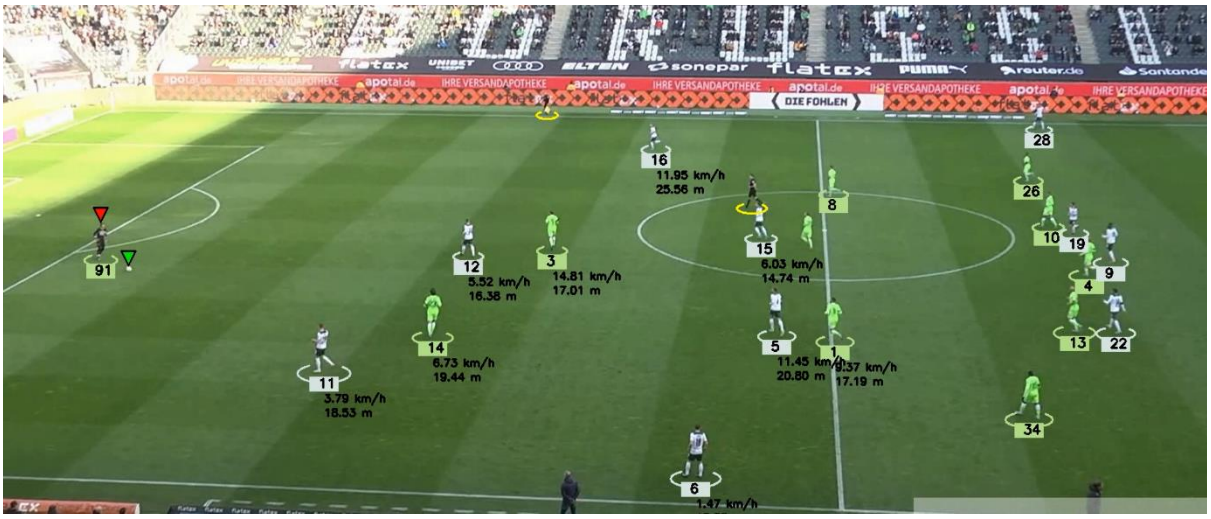


Figure 8- screenshot from the video showing speed and distance annotations for different players

6.3 Ball Possession Calculation

Ball possession calculation is essential for analyzing how well each team controls the ball during a football match. This involves tracking the ball's position and determining which team has control at different points in time.

1. **Ball Detection and Tracking**

   To track ball possession, we first need to identify the ball's location in each frame. Our model, trained with YOLOv8x, detects the ball in video frames. We use this detection to update the ball's position.

2. Tracking Ball Control

We track ball control by maintaining a record of which team has the ball in each frame. The following code snippet is used to update and store the team controlling the ball:

```
team_ball_control = []
if assigned_player != -1:
    tracks['players'][frame_num][assigned_player]['has_ball'] = True
    team_ball_control.append(tracks['players'][frame_num][assigned_player]['te
am'])
else:
    team_ball_control.append(team_ball_control[-1])
team_ball_control = np.array(team_ball_control)
```

In this code:

- **assigned_player** checks if a player is holding the ball.
- If a player is detected, their team is recorded in **team_ball_control**.
- If no player is detected, the last recorded team is used.

3. Visualization of Ball Control

To visualize ball possession, we use a function that draws the possession statistics on the video frame. The following function updates the video frame with a semi-transparent overlay showing the ball control percentages for each team:

```
def draw_team_ball_control(self, frame, frame_num, team_ball_control):
    # Draw a semi-transparent rectangle
    overlay = frame.copy()
    cv2.rectangle(overlay, (1350, 850), (1900, 970), (255, 255, 255), -1)
    alpha = 0.4
    cv2.addWeighted(overlay, alpha, frame, 1 - alpha, 0, frame)
    team_ball_control_till_frame = team_ball_control[:frame_num + 1]
    # Get the number of frames each team had ball control
  team_1_num_frames =
team_ball_control_till_frame[team_ball_control_till_frame == 1].shape[0]
    team_2_num_frames =
team_ball_control_till_frame[team_ball_control_till_frame == 2].shape[0]
    team_1 = team_1_num_frames / (team_1_num_frames + team_2_num_frames)
    team_2 = team_2_num_frames / (team_1_num_frames + team_2_num_frames)

    cv2.putText(frame, f"Team 1 Ball Control: {team_1 * 100:.2f}%", (1400,
900), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 0), 3)
    cv2.putText(frame, f"Team 2 Ball Control: {team_2 * 100:.2f}%", (1400,
950), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 0), 3)
  return frame
```

In this function:

- An overlay is drawn on the frame to display the possession stats.
- team_ball_control is sliced up to the current frame to calculate the possession percentage.
- The percentages are computed and displayed on the frame.

4. Visual Examples

Below are screenshots illustrating the ball possession calculations and the overlay results, showing how the possession data is represented visually on the video frames.

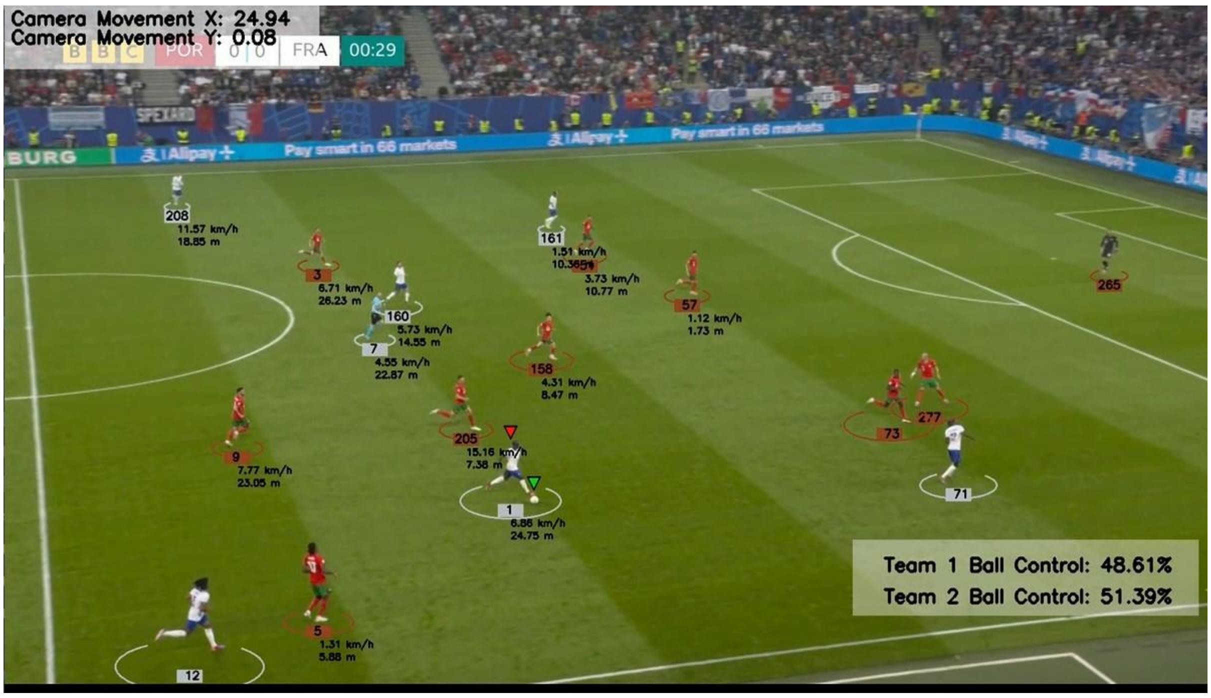


Figure 9- how the possession data is represented visually on the video frames

7. Results and Findings

The project's output focuses on analyzing player performance and team ball possession through a detailed video analysis. This section presents evaluating performance of the custom-trained model, the key results and findings from our football analysis, as illustrated in the final output video.

7.1 Evaluating Performance Metrics of the custom-trained model

The performance of the custom-trained YOLOv8x model was evaluated using several key metrics, including Mean Average Precision (mAP), Precision and Recall. These metrics provide a comprehensive assessment of the model's ability to detect and localize football players, the ball, referees, and goalkeepers accurately.

1. Mean Average Precision (mAP)

- **mAP@0.5**: This measures the average precision at an Intersection over Union (IoU) threshold of 0.5, which indicates how well the model identifies the object classes in the dataset. The custom YOLOv8x model achieved a mAP@0.5 of XX%, reflecting its strong detection performance across different objects.
- **mAP@0.5:0.95**: This measures the average precision across multiple IoU thresholds ranging from 0.5 to 0.95. The model achieved YY%, indicating consistent performance across tighter bounding box requirements.

2. Precision

Precision refers to the proportion of correct positive predictions among all positive predictions made by the model. A high precision score indicates that the model generates fewer false positives. The custom-trained model achieved a precision score of ZZ%, which demonstrates that it can effectively identify and classify objects in the football dataset.

3. Recall

Recall measures the proportion of actual positives that the model correctly identified. A high recall score suggests that the model successfully detects most of the relevant objects in the frames. The recall score for the model was AA%, highlighting its effectiveness in capturing the key objects in the video frames.

Here is my test result.

The performance of the custom-trained YOLOv8x model was evaluated using a validation set consisting of **49 images** and **1,174 object instances**. The metrics considered include Precision (**P**), Recall (**R**), **mAP@0.5**, and **mAP@0.5:0.95**. These metrics were computed across all object classes: **ball, goalkeeper, player, and referee**.

The table below provides a summary of the results:

CLASS	IMAGES	INSTANCES	PRECISION (P)	RECALL (R)	MAP@0.5	MAP@0.5:0.95
ALL	49	1,174	0.901	0.779	0.846	0.601
BALL	49	45	0.889	0.354	0.495	0.205
GOALKEEPER	49	39	0.901	0.897	0.938	0.700
PLAYER	49	973	0.956	0.948	0.988	0.817
REFEREE	49	117	0.856	0.915	0.965	0.683

1. Overall Performance

Precision (P): The overall precision of the model across all classes is **90.1%**, indicating that the model made correct predictions in the majority of cases, with relatively few false positives.

Recall (R): The overall recall is **77.9%**, showing that the model was able to identify a significant proportion of the actual objects, though some instances were missed.

mAP@0.5: The mean Average Precision at IoU threshold 0.5 is **84.6%**, reflecting the model's strong localization and classification ability.

mAP@0.5:0.95: The mean Average Precision across different IoU thresholds (from 0.5 to 0.95) is **60.1%**, which is a good indication of the model's performance across tighter bounding box requirements.

2. Class-Specific Performance

Ball: The ball class has a precision of 88.9% and a recall of 35.4%, with a relatively lower mAP@0.5 of 49.5% and mAP@0.5:0.95 of 20.5%. This shows that while the model correctly identifies balls in many cases, there are challenges in consistently detecting them.

Goalkeeper: The goalkeeper class exhibits strong performance with a precision of 90.1%, recall of 89.7%, and an mAP@0.5 of 93.8%. The mAP@0.5:0.95 is 70.0%, indicating good localization across different IoU thresholds.

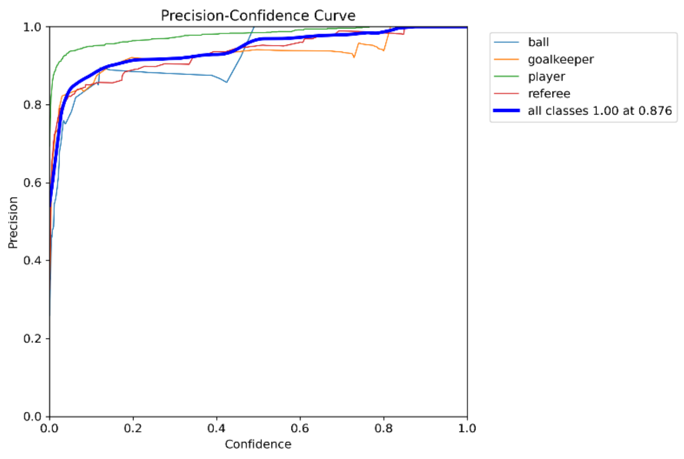
Player: The player class, which has the largest number of instances (973), demonstrates excellent performance with a precision of 95.6%, recall of 94.8%, and mAP@0.5 of 98.8%. The mAP@0.5:0.95 is also high at 81.7%, showing robust detection and classification capabilities.

Referee: The referee class shows a precision of 85.6% and recall of 91.5%, with an mAP@0.5 of 96.5% and mAP@0.5:0.95 of 68.3%, indicating that the model performs well in detecting referees in the dataset.

7.2 Visualizations of the Model Performance

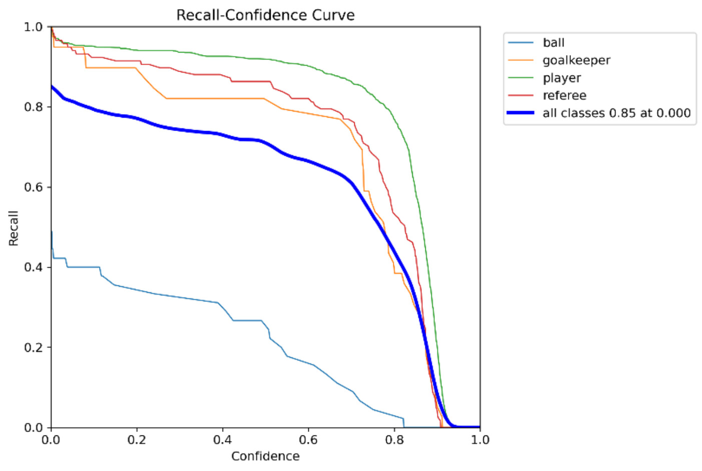
The following figures provide a graphical representation of the model's performance over the training period:

Precision Curve (P Curve)



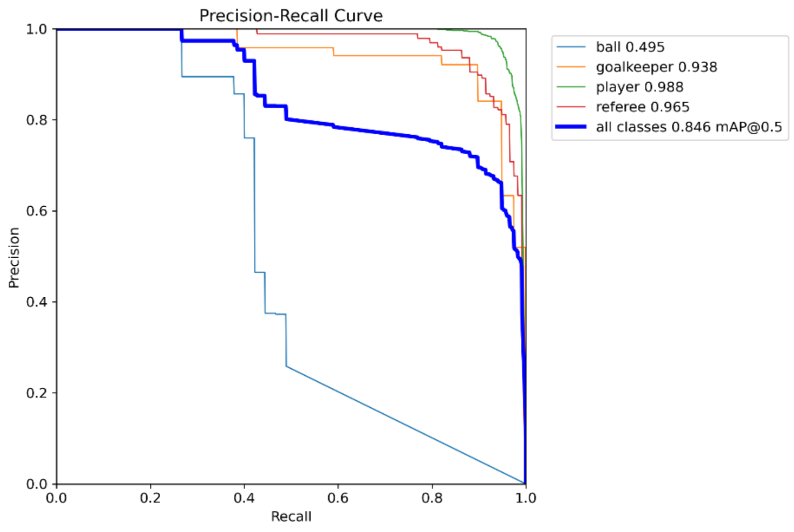
This plot will show the precision values on the y-axis and the confidence thresholds on the x-axis.

Recall Curve (R Curve)



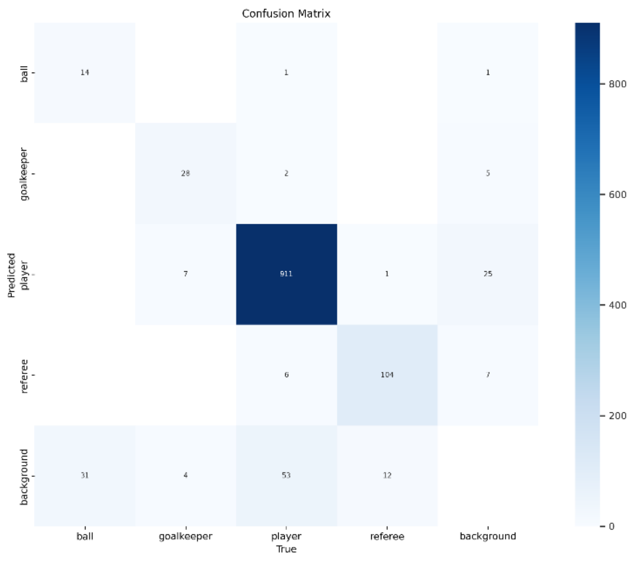
This plot will display recall values on the y-axis and confidence thresholds on the x-axis.

Precision-Recall Curve (PR Curve)



This plot provides an overview of the precision-recall tradeoff at various thresholds, showing how one metric is impacted when optimizing for the other.

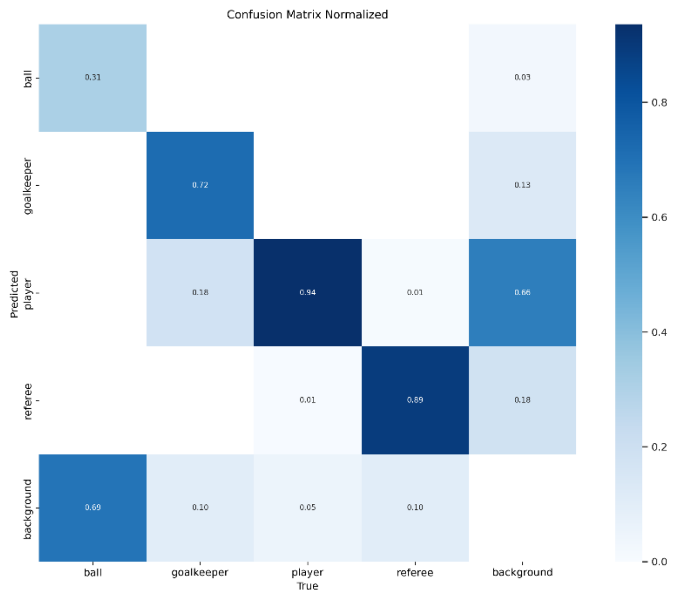
Confusion Matrix



Predicted \ True	ball	goalkeeper	player	referee	background
ball	14		1		1
goalkeeper		28	2		5
player		7	911	1	25
referee			6	104	7
background	31	4	53	12	

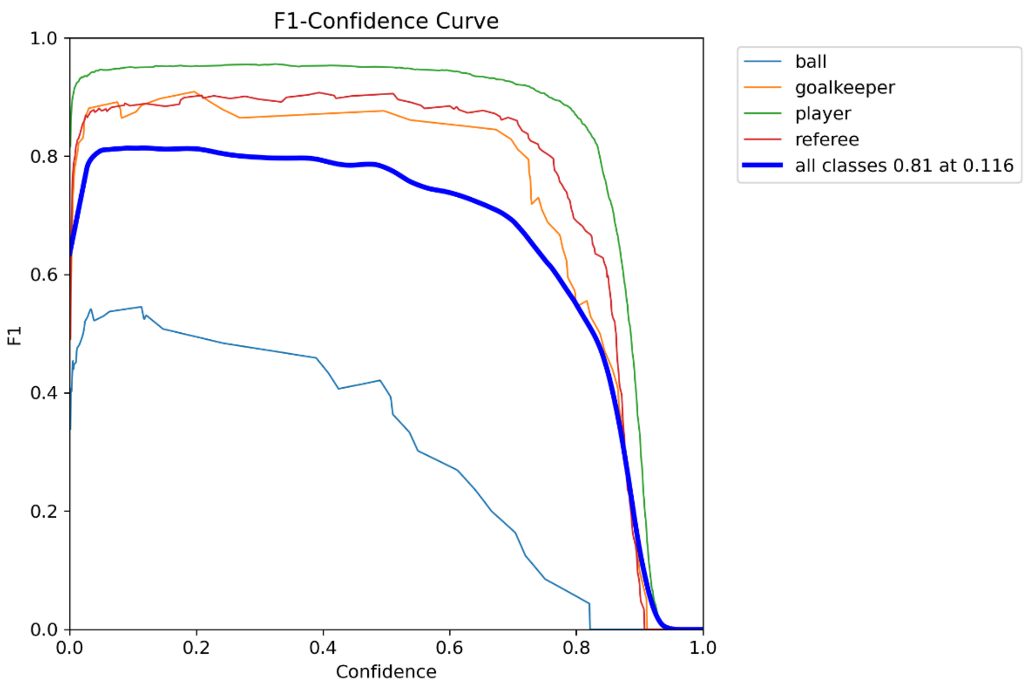
 A confusion matrix summarizes the model's classification performance. It shows how many times the model correctly classified an object versus how many times it made a mistake.

Normalized Confusion Matrix



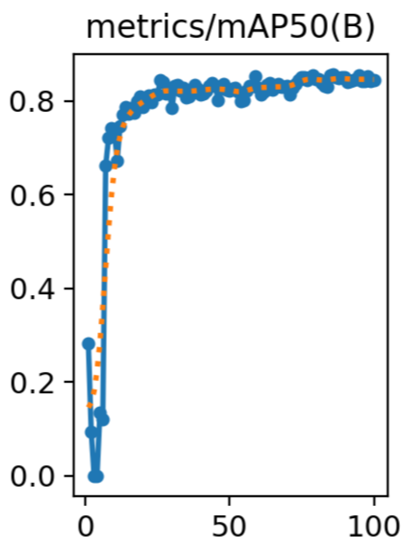
This matrix provides a normalized view, helping to understand classification errors relative to the total number of instances per class.

F1-Confidence Curve



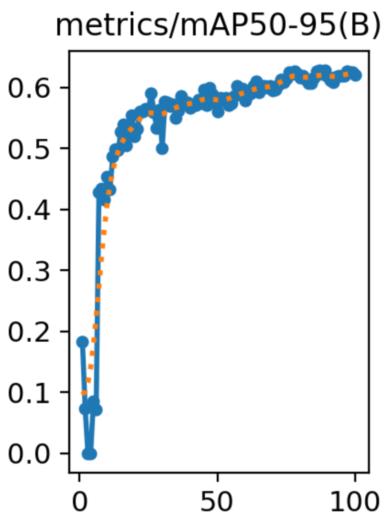
This curve illustrates the relationship between the F1-score and the confidence threshold for different classes. A higher F1-score indicates better model performance in correctly classifying objects.

Mean Average Precision at IoU=0.5 (metrics/mAP50(B)):



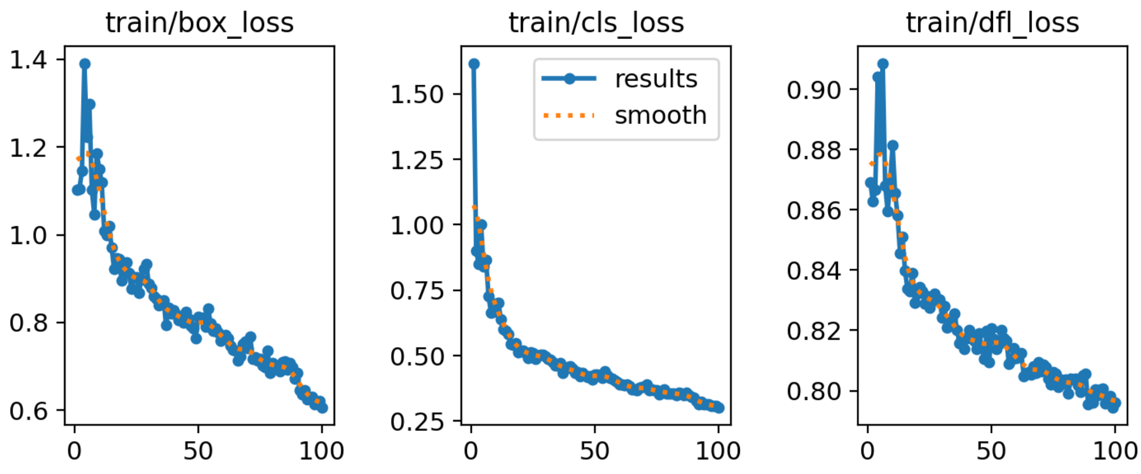
This plot shows the mAP@0.5, which is a standard object detection metric that evaluates the model's precision and recall at a specific Intersection over Union (IoU) threshold of 0.5. A higher mAP@0.5 indicates better overall performance in detecting objects with good precision and recall.

Mean Average Precision at IoU=0.5:0.95 (metrics/mAP50-95(B)):



This curve shows the mean Average Precision across multiple IoU thresholds ranging from 0.5 to 0.95. This metric is stricter than mAP@0.5 and provides a more comprehensive evaluation of the model's performance, considering both easy and difficult detection cases. A higher mAP@0.5:0.95 indicates a well-performing model across different levels of object localization precision.

Training Loss Curves (train/box_loss, train/cls_loss, train/dfl_loss):

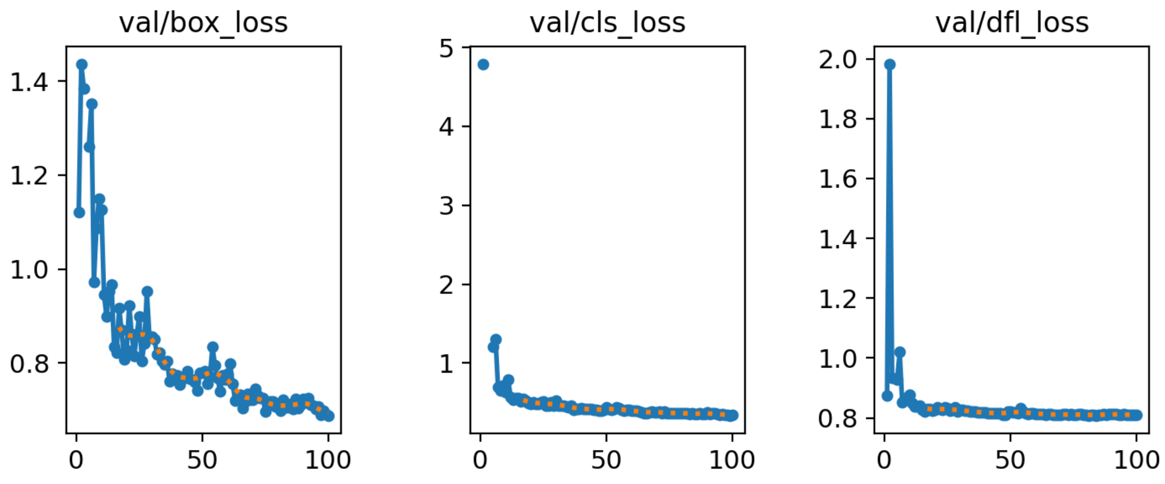


Box Loss: This curve shows how the model's error in predicting the bounding boxes for objects decreases over the training epochs. A decreasing curve indicates that the model is getting better at predicting accurate bounding box locations.

Class Loss: This curve represents the model's error in classifying the objects within the bounding boxes. A decrease in this curve implies that the model is improving in distinguishing between different classes.

DFL Loss (Distribution Focal Loss): This loss is related to the fine-grained localization and precision in object detection. A decreasing curve indicates an improvement in localization accuracy.

Validation Loss Curves (val/box_loss, val/cls_loss, val/dfl_loss):



Box Loss: This curve shows the model's error in predicting the bounding boxes for objects in the validation set. A decreasing curve indicates that the model is improving in accurately predicting the bounding box locations on unseen data.

Class Loss: This curve represents the model's error in classifying the objects within the bounding boxes in the validation set. A decrease in this curve implies that the model is getting better at distinguishing between different object classes on data it hasn't seen before.

DFL Loss (Distribution Focal Loss): This curve tracks the model's precision in localizing objects in the validation set. A decreasing DFL loss indicates that the model is becoming more accurate in refining the bounding box locations for unseen objects, improving its fine-grained localization capabilities.

7.3 Video Output Overview

The final result is a video that incorporates various analysis metrics, including:

Player Speed: This metric displays the speed of each player in real-time, calculated based on their movement across consecutive frames. The speed is visually represented on the video, providing insights into how quickly players move during the game.

Distance Traveled: The distance each player has covered throughout the match is calculated and displayed. This metric helps in understanding player activity levels and their contribution to the game.

Ball Possession: The video includes annotations showing which team has control of the ball at different points in time. This is presented as a percentage of ball control for each team, giving a clear view of the match dynamics.

7.4 Visualization Details

1. Player Speed and Distance:

Player speed is annotated directly on the video frames, with each player's speed shown in kilometers per hour (km/h).

Distance traveled is shown as an accumulative measure, indicating how far each player has moved during the match.

2. Ball Possession Calculation:

A semi-transparent overlay on the video frame provides real-time updates on the percentage of ball possession for each team.

The possession data is calculated based on the ball's control history and displayed in percentage terms.

3. Example Screenshots

The following screenshots illustrate the key elements of the video output:

Player Speed and Distance Traveled Visualization:

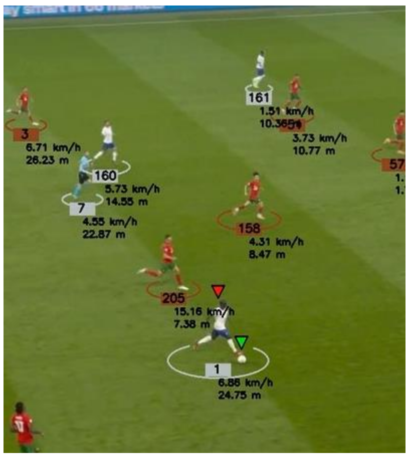


Figure 10: A screenshot showing the player speed and distance annotations on the video frame.

Ball Possession Visualization:

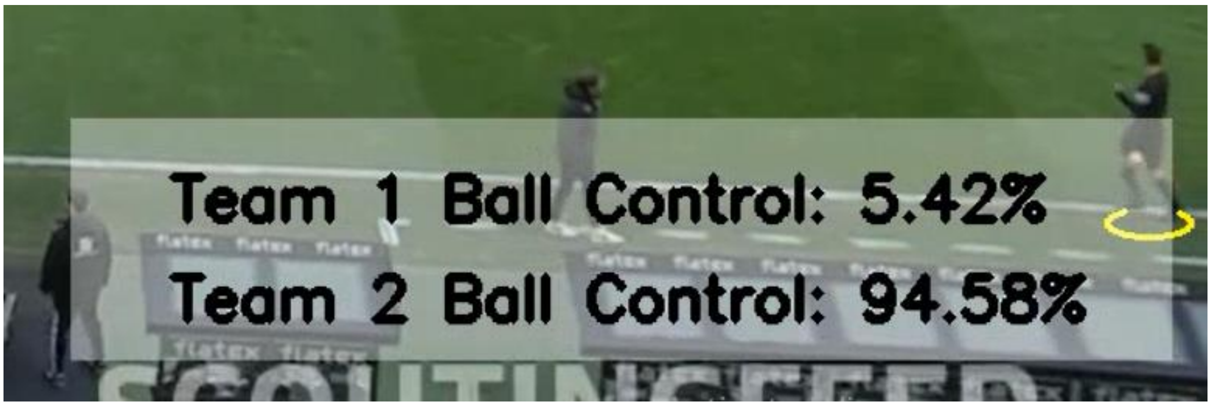

Team 1 Ball Control: 5.42%
Team 2 Ball Control: 94.58%

Figure 11: A screenshot illustrating the ball possession percentages for each team.

7.5 Key Findings

The custom-trained YOLO model demonstrates strong performance with a mAP (mean Average Precision) of 60.1% across all classes, highlighting its effective detection capabilities. The model performs particularly well in detecting players and goalkeepers, with high precision and recall values. For specific metrics:

- **Player Detection**: mAP50 = 98.8%
- **Goalkeeper Detectio**n: mAP50 = 93.8%
- **Ball Detection**: mAP50 = 49.5%
- **Referee Detection**: mAP50=96.5%

These results indicate that the model **excels at detecting players** but **struggles slightly with detecting the ball accurately.**

The analysis provided a clear visualization of player speeds and distances, allowing for performance assessment.

- Ball possession data highlighted periods of dominance by each team, which can be correlated with match events and strategies.
- The video output effectively communicated the results of the analysis, providing actionable insights into team and player performance.

- The video output effectively communicated the results of the analysis, providing actionable insights into team and player performance.

8. Future Work

The current project has successfully demonstrated the capabilities of computer vision and data analysis in football match analysis. However, there are several areas where the project can be expanded or enhanced to provide even more valuable insights and functionalities. Future work could focus on the following aspects.

8.1 Content Recommendation

To improve the relevance of video content for users, the system could integrate a content recommendation engine. This could leverage player and team performance data to suggest relevant highlights, match analyses, or similar games based on user preferences and viewing history. Such enhancements could provide a more personalized experience for coaches, analysts, and fans.

8.2 Advertising and Marketing

Utilizing player and team performance metrics for targeted advertising and marketing strategies could be beneficial. For example, insights gained from player analysis could be used to create personalized ads and promotions that align with players' performance and achievements. This approach could drive engagement and increase the effectiveness of marketing campaigns related to football.

8.3 Enhanced Player Tracking and Analysis

Future work could focus on enhancing player tracking capabilities to include more advanced metrics, such as heatmaps of player movements, detailed player interactions, and advanced tactical analysis. Improvements in tracking accuracy and detail could provide deeper insights into player performance and team strategies

8.4 Real-Time Analysis

Developing real-time analysis capabilities could significantly enhance the usefulness of the system during live matches. This would involve optimizing algorithms and computational resources to process video feeds in real-time, allowing for immediate feedback and decision-making during matches.

8.5 Integration with Other Data Sources

Integrating data from additional sources, such as GPS trackers, biometric sensors, and external databases, could provide a more holistic view of player and team performance. This could enable more accurate assessments and predictions based on a richer set of data.

9. Conclusion

The football analysis project successfully applied computer vision and data analysis techniques to evaluate and visualize player and team performance. By leveraging YOLO-based object detection, the project provided detailed insights into player movements, ball possession, and team dynamics, significantly enhancing the understanding of match events.

Key achievements include:

- Accurate Player and Ball Detection: The YOLOv8x model demonstrated high precision in detecting players and the ball, contributing to effective tracking and analysis.
- Detailed Player Analysis: The system effectively calculated player speeds and distances traveled, providing valuable metrics for performance assessment.
- Ball Possession Tracking: The ball possession calculation allowed for accurate measurement of team control throughout the match, offering insights into team strategies and performance.

The results are presented through a comprehensive output video that showcases these analyses, effectively communicating the project's findings and demonstrating its practical applications in sports analytics.

Future work could build on these achievements by incorporating additional features such as real-time analysis, enhanced player tracking, and integration with other data sources. These advancements could further enrich the analysis and expand the project’s impact on football match analysis.

Overall, this project highlights the potential of combining computer vision with data analysis to gain deeper insights into football games, paving the way for future developments and applications in sports analytics.

10. References

- Dataset - https://universe.roboflow.com/roboflow-jvuqo/football-players-detection-3zvbc/dataset/1
- Video- DFL- Bundesliga Data Shootout | Kaggle
- Google Collab -https://colab.google.com
- Pandas - https://pandas.pydata.org
- Sklearn- https://scikit-learn.org
- Numpy - https://www.geeksforgeeks.org/python-numpy
- Matplotlib- https://matplotlib.org
- K-Means Clustering - https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a
- External Help- Chatgpt (ChatGPT)