DEEP LEARNING TERM PROJECT

OBJECT DETECTION ANALYSIS OF BUNDESLIGA DATA SHOOTOUT

Submitted By

VISHNU ARUN
(G22015051)

BUNDESLIGA

CONTENTS

INTRODUCTION

In this project, the "DFL - Bundesliga Data Shootout," our team decided to design a computer vision model that could identify and classify specific football events, like “Playing,” “Throw-ins,” and “Challenge,” from Bundesliga match footage. Traditionally, these event data was and is to some extent collected manually, a resource-intensive method that is typically reserved for top-tier leagues.

By automating this process, we can extend sophisticated analytical capabilities to youth and semi-professional leagues, where resources are scarce but the talent pool is rich. This initiative not only aims to streamline data collection but also ensures that no potential talent goes unnoticed due to the lack of data.

The challenge involves detecting these events accurately within predefined intervals and scoring them against ground truths to evaluate the model's precision. This project could potentially revolutionize how we gather and analyze sports data, making it faster, more accurate, and widely applicable across various levels of professional sports.

PROBLEM STATEMENT

The problem statement for the "DFL - Bundesliga Data Shootout" involves developing a machine learning model that can automatically detect and classify specific football events—passes, throw-ins, crosses, and challenges—from Bundesliga match footage. The model must accurately identify these events within predefined scoring intervals and evaluate them using a metric based on average precision.

PHASE I

I began this project with the intention of implementing a machine learning model to automatically detect and classify specific football events from Bundesliga match footage, as outlined in an academic tutorial. The goal was to understand the existing Kaggle Notebook, get it up and running, and then build upon these results to enhance our model's capabilities for the class presentation. However, we encountered significant challenges from the outset. The initial day was spent delving into the dataset, coding, and addressing numerous bugs, which considerably slowed our progress. Realizing the time constraints we were under, my team and I decided to pivot our approach on the second day.

We shifted our focus to a YOLO-based object detection model, a method demonstrated in a YouTube tutorial, which seemed more feasible given our timeframe. My teammates, Shanun and Kalyani, worked on different aspects: Shanun continued working on the EfficientNET model and Kalyani & me worked on implementing the YOLO object detection model.
I also attempted to adapt the YOLO model to our needs. Despite our efforts, we faced persistent issues, primarily with the extraction of frames from the video clips, which took an exorbitant amount of time—about 20-40 minutes per clip. Additionally, repeated failures with the NVIDIA GPU on our AWS instance required us to set up new instances multiple times, each taking additional hours to configure and extract frames.

On the third Nvidia GPU failure, I decided to limit the dataset to only five clips to expedite the frame extraction process, which allowed us to complete this step in approximately one hour. However, the output from the YOLO model was not as expected and was detecting football players that were clearly people in the crowd.

On the day of our presentation, neither Shanun nor I had fully operational models. Fortunately, Kalyani's partially functional YOLO v5 model provided us with enough material to present to the class. We discussed the challenges encountered, the methodologies attempted, and the lessons learned from this rigorous exercise in applying machine learning techniques to video footage.

Code Explanation:

Python Scripts and Modules Developed:

team_assigner.py :
This module contains the `TeamAssigner` class which assigns team colors to players detected in video frames. It uses K-means clustering to determine the dominant color in the upper half of the detected player's bounding box, categorizing these into two clusters representing the two different team uniforms. After extracting individual player colors, a second round of K-means clustering assigns players to two distinct teams based on these colors.

main.py:
This script calls the overall processing flow. It reads the video input, tracks objects (players and the ball) using a pre-trained model, assigns players to teams, interpolates ball positions for smoother tracking, and finally outputs an annotated video. This script integrates object detection and tracking, color-based team classification, and ball possession dynamics into a cohesive analysis of football gameplay.

video_utils.py: This utility module provides functions for reading and saving videos. `read_video` function captures frames from a video file, and `save_video` writes frames back to a video file with annotations, facilitating both input and output video handling.

bbox_utils.py: This file includes functions to work with bounding boxes, crucial for object detection tasks. `get_center_of_bbox` calculates the center point of a given bounding box, aiding in positioning and tracking analysis. `get_bbox_width` returns the width of the bounding box, useful for size-based filtering or scaling. `measure_distance` calculates the Euclidean distance between two points, applicable in determining the spacing between players or between a player and the ball.

All the code can be found here in Phase 1 this Github Repository.

PHASE II

In the second phase of our project, we were fortunate to receive a two-day extension from Professor Amir, which allowed us to refocus our efforts on addressing the original problem statement concerning the dataset. On the first day of this extended phase, I revisited the EfficientNet implementation from a Kaggle notebook with the intention of starting fresh. However, by the evening, I was still facing persistent issues with the implementation, as it continued to return errors. Additionally, our AWS instances were not rebooting due to "insufficient capacity errors".

Given these challenges and with my teammates, who possess more expertise than myself, also still working to get their models operational.
I decided that it's not wise for me to be working on the EffectiveNet model implementation and pivoted back to a more familiar territory of object detection using YOLOv8. On this day, I switched to using Google Colab Pro. This allowed me to focus on developing the model and rather than dealing with the logistics part. My goal was to develop a functional Streamlit app by this evening, the project's deadline.

i) Code Explanation:

YOLOv8 Model:

I utilized the "football-players-detection Dataset" from Roboflow to train an object detection model. This dataset is specifically tailored for detecting football players, consisting of 663 images split into training (92%), validation (6%), and testing (2%). Such a distribution ensures that the model is trained on a robust set of data while having enough images to validate and test its performance accurately.

To enhance the model's ability to generalize across different game conditions, several augmentations to the dataset were already applied, including horizontal flips, and adjustments in saturation and brightness. These modifications help the model perform effectively under various lighting and weather conditions, and when players are in different orientations.

The dataset from Roboflow, created from an original Bundesliga dataset, has significantly contributed to the accuracy and robustness of our model. The dataset's structure and the preprocessing steps applied ensure that the model trained can handle real-world variations effectively.

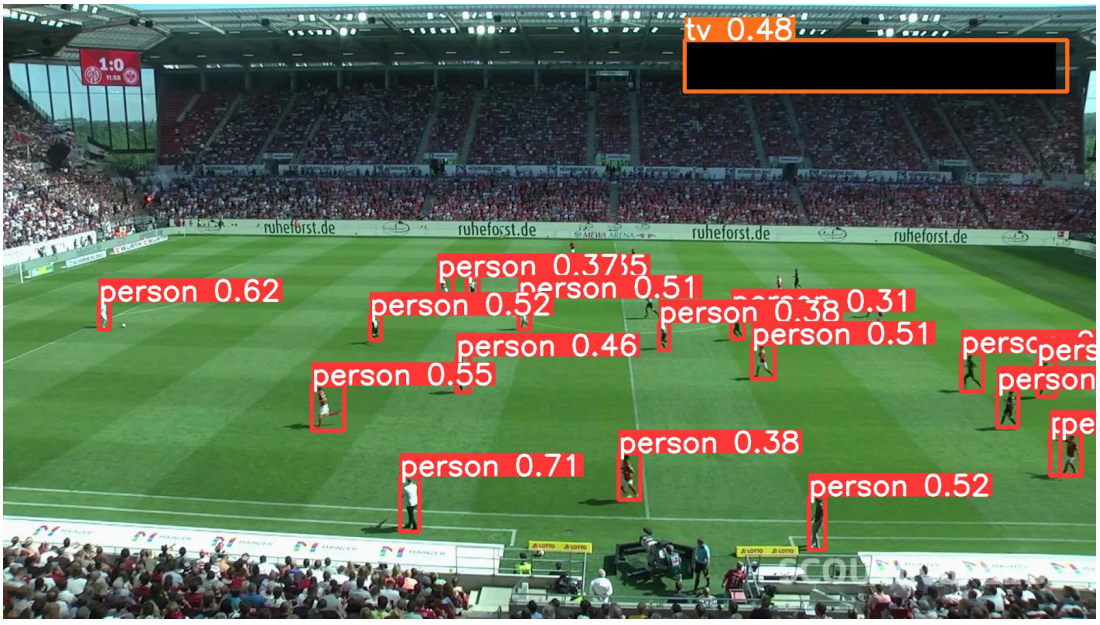
I chose this dataset over the original Bundesliga dataset, because it offered a more refined set of annotations and preprocessing tailored to the specific needs of getting a functional model that can be deployed on streamlit. Moreover, it also came in formats compatible with multiple frameworks, including YOLOv8, which I used for the model architecture.

Further details of the model architecture and its implementation are documented in google collab notebook.

Results:

Input Image:

Output Image:



Output Video:

videoclip_2.avi

ii) Code Explanation:

Custom YOLOv8 Model:

In this custom YOLOv8 model for football player detection, significant emphasis was placed on optimizing the model's weights to achieve the best possible performance. The term "best weights" refers to the set of model parameters that

yield the highest accuracy and generalization ability on the validation dataset.

Regular Evaluation: During training, the model's performance was regularly evaluated against a validation dataset not seen during the training phase. This helped in monitoring the model's ability to generalize beyond the training data.

Loss Metrics: Three primary loss metrics were tracked: box loss, class loss, and direction-focused loss (dfL). These metrics provided insights into different aspects of the learning process:

Box Loss: Measures the accuracy in predicting the bounding box coordinates for each detected object.
Class Loss: Indicates how well the model is classifying different objects in the dataset, such as players, referees, and the ball.

Direction-Focused Loss (dfL): An advanced metric used in some YOLO configurations to refine the orientation and pose estimation, enhancing the model's ability to understand object orientations in the video.

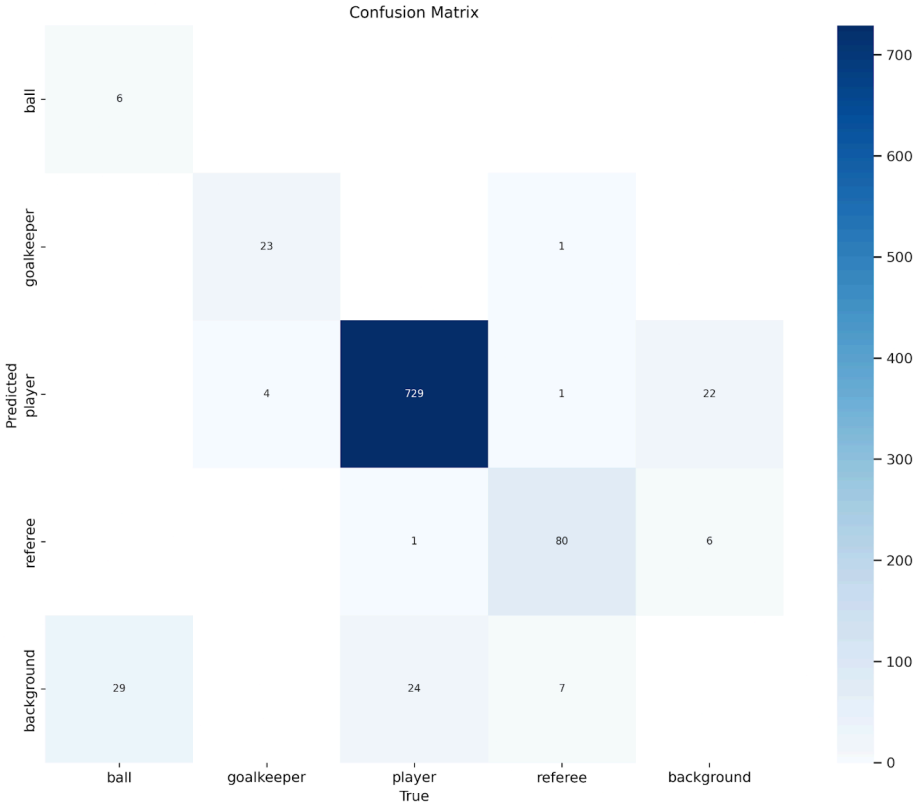
```
Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
 1/25      10.2G      1.077       1.35     0.8423        128        800: 100% 39/39 [00:43<00:00,  1.11s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:01<00:00,  1.15it/s]
             all         38        905      0.787      0.569      0.623      0.419

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
 2/25      10.1G     0.9941     0.6511     0.8147        117        800: 100% 39/39 [00:19<00:00,  2.05it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:00<00:00,  5.27it/s]
             all         38        905      0.841      0.628      0.701      0.487
```

```
Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
24/25      10.5G      0.663     0.3336     0.7889         88        800: 100% 39/39 [00:18<00:00,  2.13it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:00<00:00,  5.31it/s]
             all         38        905      0.926      0.752      0.795      0.594

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
25/25      10.5G     0.6588     0.3328     0.7852         92        800: 100% 39/39 [00:18<00:00,  2.13it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:00<00:00,  5.27it/s]
             all         38        905      0.898      0.775      0.801        0.6
```

```
Validating runs/detect/train4/weights/best.pt...
Ultralytics YOLOv8.0.196 🚀 Python-3.10.12 torch-2.2.1+cu121 CUDA:0 (NVIDIA L4, 22700MiB)
Model summary (fused): 218 layers, 25842076 parameters, 0 gradients, 78.7 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:00<00:00,  3.80it/s]
                   all         38        905      0.898      0.775      0.801      0.597
                  ball         38         35      0.833      0.229      0.268      0.125
            goalkeeper         38         27      0.926      0.925      0.979      0.765
                player         38        754      0.941      0.981      0.992      0.811
               referee         38         89      0.893      0.966      0.965      0.688
Speed: 0.2ms preprocess, 4.5ms inference, 0.0ms loss, 0.8ms postprocess per image
```

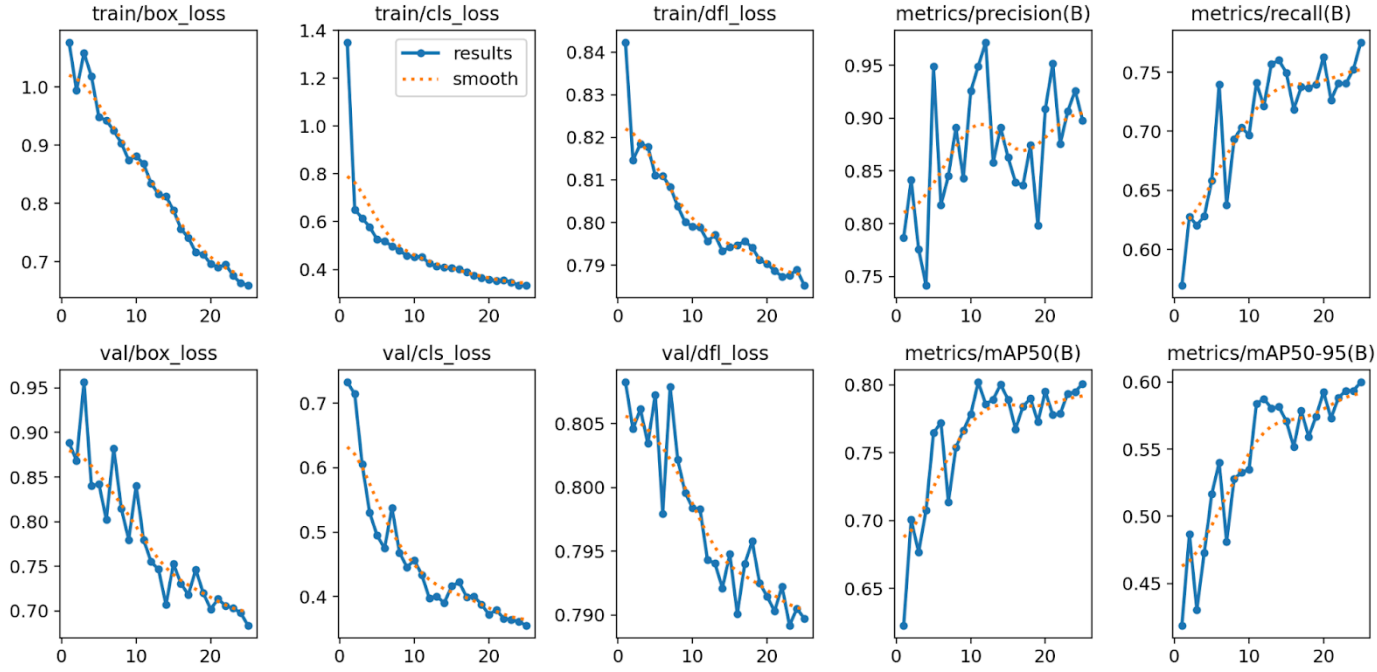


Confusion Matrix Observations:

High Accuracy for Players: The matrix indicates strong performance in identifying players, with 729 correct predictions.

Some Misclassifications: Notable misclassifications include 6 instances where the ball was incorrectly predicted as background and 23 where the goalkeeper was mislabeled.

General Performance: Overall, the confusion matrix shows a very reliable model, but highlights areas for improvement, particularly in distinguishing the ball and goalkeepers from other categories.



Loss and Performance Plot Observations:

Decreasing Loss Trends: The plots display a steady decrease in box, class, and direction-focused loss over training epochs, indicating good model learning and adaptation.

Fluctuations in Precision and Recall: Both metrics show variability but generally improve, suggesting increasing reliability in detecting objects accurately as training progresses.

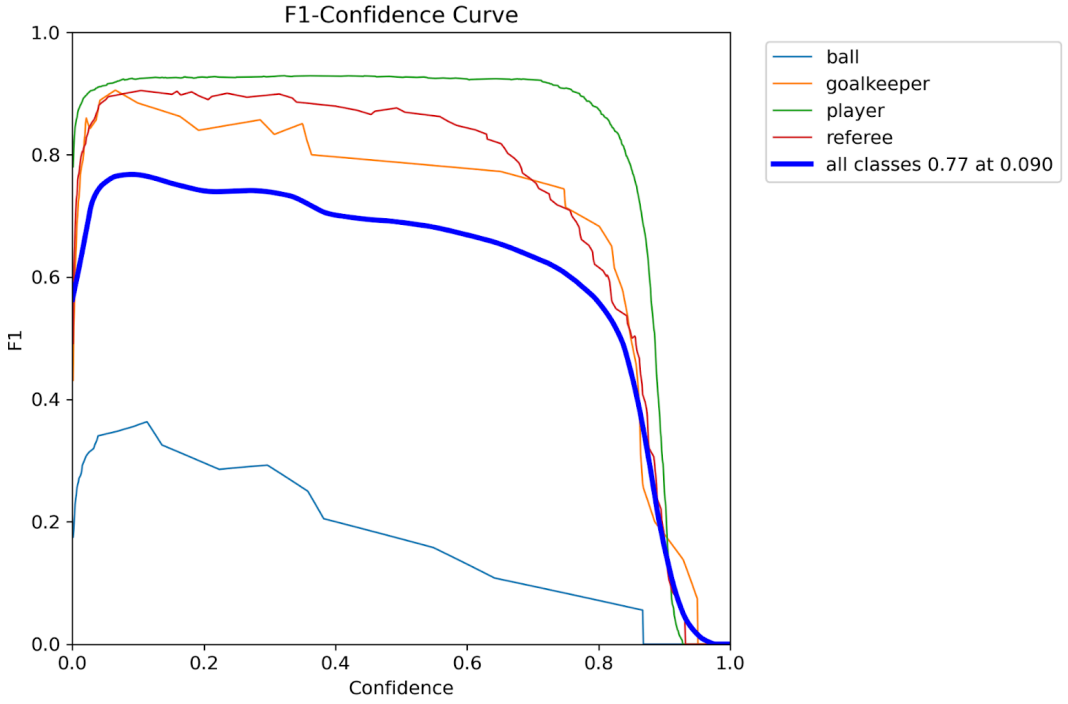
Improvement in mAP Scores: The mean Average Precision scores for both mAP@50 and mAP@50-95 increase, reflecting enhanced model accuracy and robustness across varying Intersection over Union (IoU) thresholds.

Prediction Outputs on Field Images:

Effective Detection: The model effectively identifies and locates players and referees with high confidence scores, indicating strong detection capabilities.

Detailed Annotations: Predictions include detailed bounding boxes and confidence scores for each detected object, showcasing the model's precision.

F1-Confidence Curve:

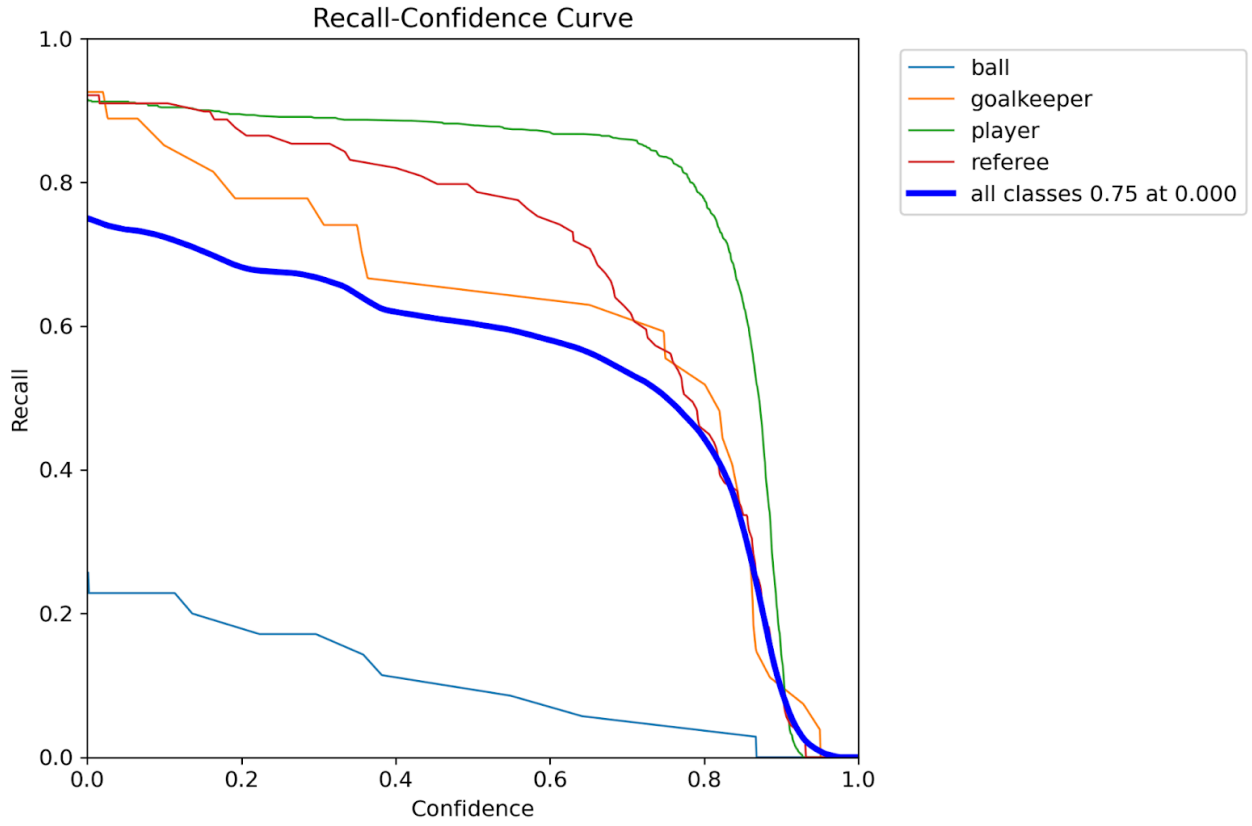


High Stability in F1 Score: The F1 score for the goalkeeper, player, and referee remains consistently high across most confidence thresholds, indicating strong precision and recall balance.

Decline in Lower Confidence: For all classes, the F1 score starts to decline significantly as the confidence threshold approaches 1.0, suggesting a trade-off at extremely high confidence levels.

Variability for the Ball: The ball shows a notably lower and more variable F1 score, indicating challenges in detecting the ball with high confidence compared to other classes.

Recall-Confidence Curve:

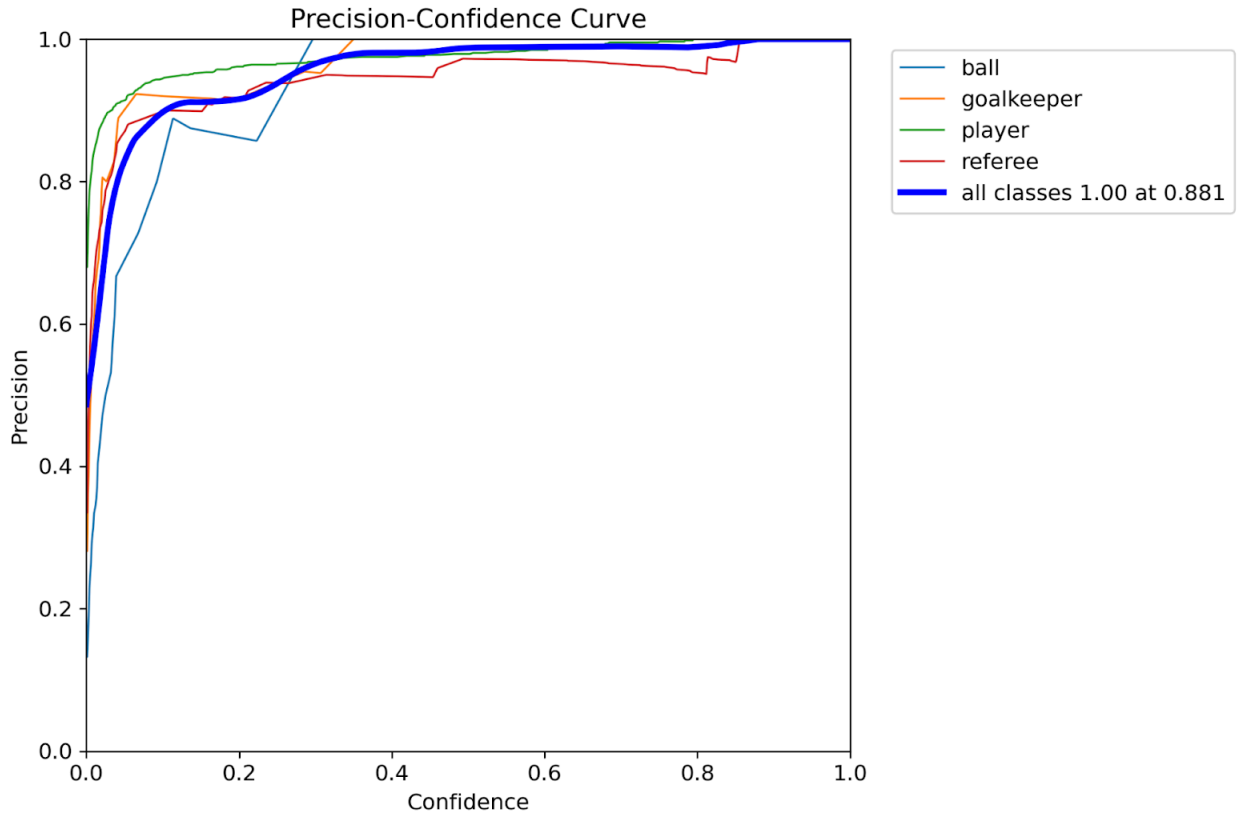


High Recall for Key Classes: Players and referees maintain high recall rates until a sharp drop at very high confidence levels, showing that the model reliably detects most true positives at moderate confidence thresholds.

Lower Recall for the Ball: The ball exhibits much lower recall across all confidence levels, reflecting difficulty in consistently identifying all true ball instances.

Overall Performance: Despite the drop at high confidence levels, the model maintains a relatively stable recall above 0.6 for key classes up to a confidence threshold of about 0.8.

Precision-Confidence Curve

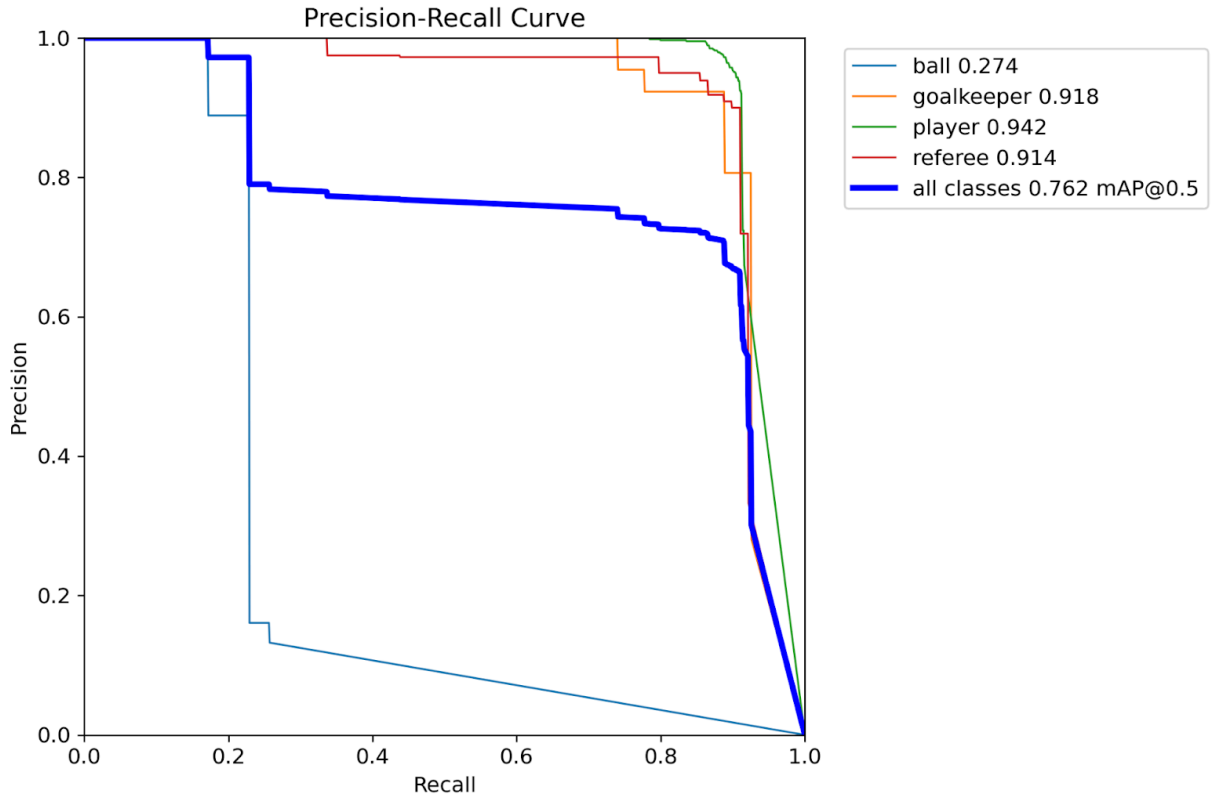


Precision Stability: Precision remains stable and high for the goalkeeper and referee across a wide range of confidence thresholds, highlighting the model's accuracy in these predictions.

Impact of Confidence on Precision: There is a noticeable increase in precision for all classes as the confidence threshold increases, peaking before slightly tapering off at the highest thresholds.

Comparison Across Classes: Players show slightly less precision compared to goalkeepers and referees, suggesting some potential overlaps or misclassifications with similar classes.

Precision-Recall Curve



High Precision with Varied Recall: The curve shows that while precision is generally high for the goalkeeper, player, and referee, the recall varies, with the player class achieving the highest recall.

Challenges with Ball Detection: The ball has significantly lower precision and recall, indicating specific challenges in detecting this class accurately.

Overall Efficiency: The model demonstrates a strong trade-off between precision and recall, with high initial precision that decreases as recall increases, which is typical in precision-recall trade-offs.

Further details of the model architecture and its implementation are documented in google collab notebook.

Results:

Output Video:

0a2d9b_5.avi

STREAMLIT APP:

In this part I built a football object detection streamlit application using a YOLOv8 model trained on a specialized dataset from Roboflow. Once the model was fully trained and had identified the best weights, I downloaded the model to my local machine. This allowed me to transition from training to deployment seamlessly.

The core functionality of the application is to perform object detection on any uploaded football video. Users can upload their videos directly to the app, and the trained YOLOv8 model processes these videos to detect and highlight football players in real-time. Here's a breakdown of how the application works:

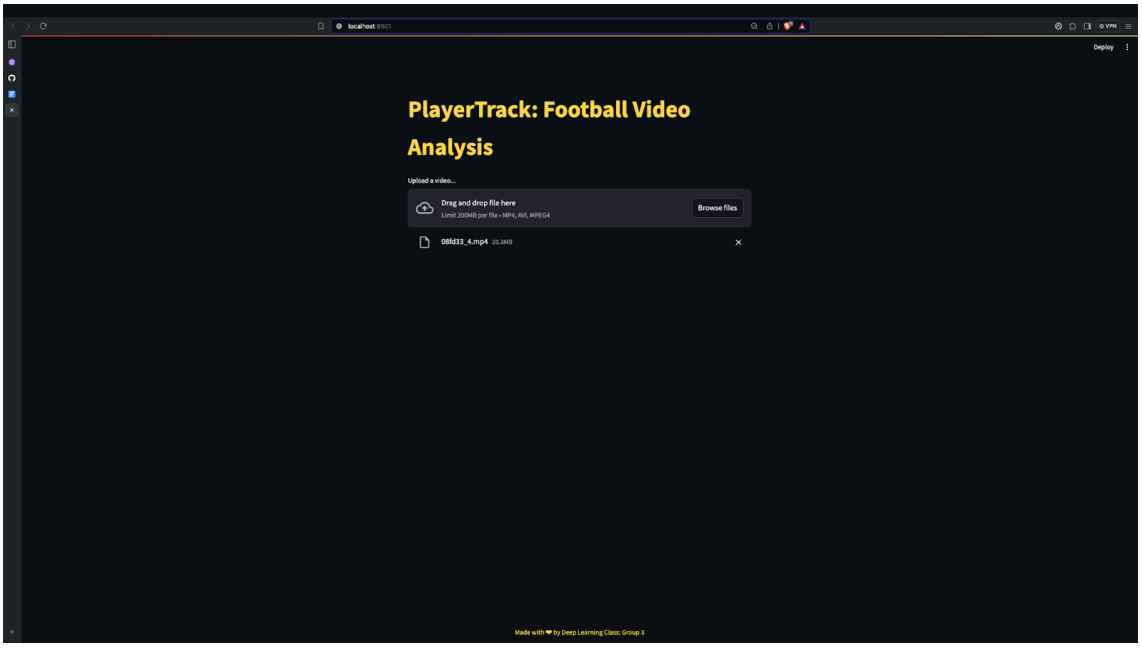
Video Upload: Users start by uploading a video file through the Streamlit interface. The app supports various video formats, making it versatile for different users.

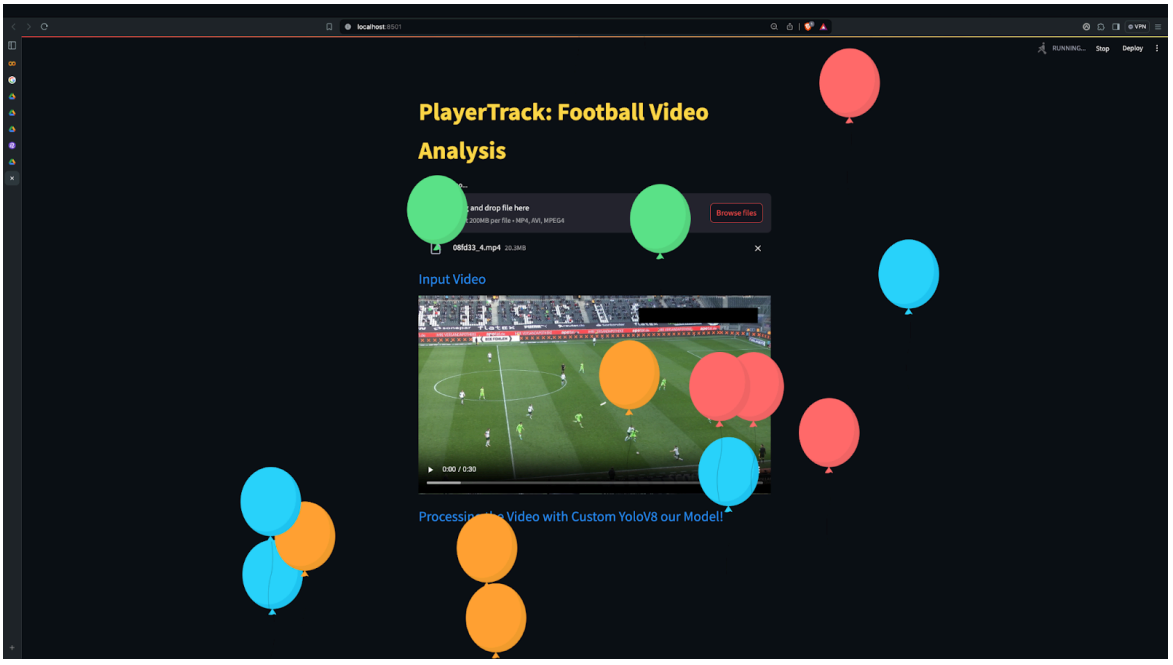
Object Detection: Once a video is uploaded, it is fed into the YOLOv8 model. The model uses the trained weights to perform object detection on each frame of the video. This process involves identifying the location of football players and bounding them with boxes.

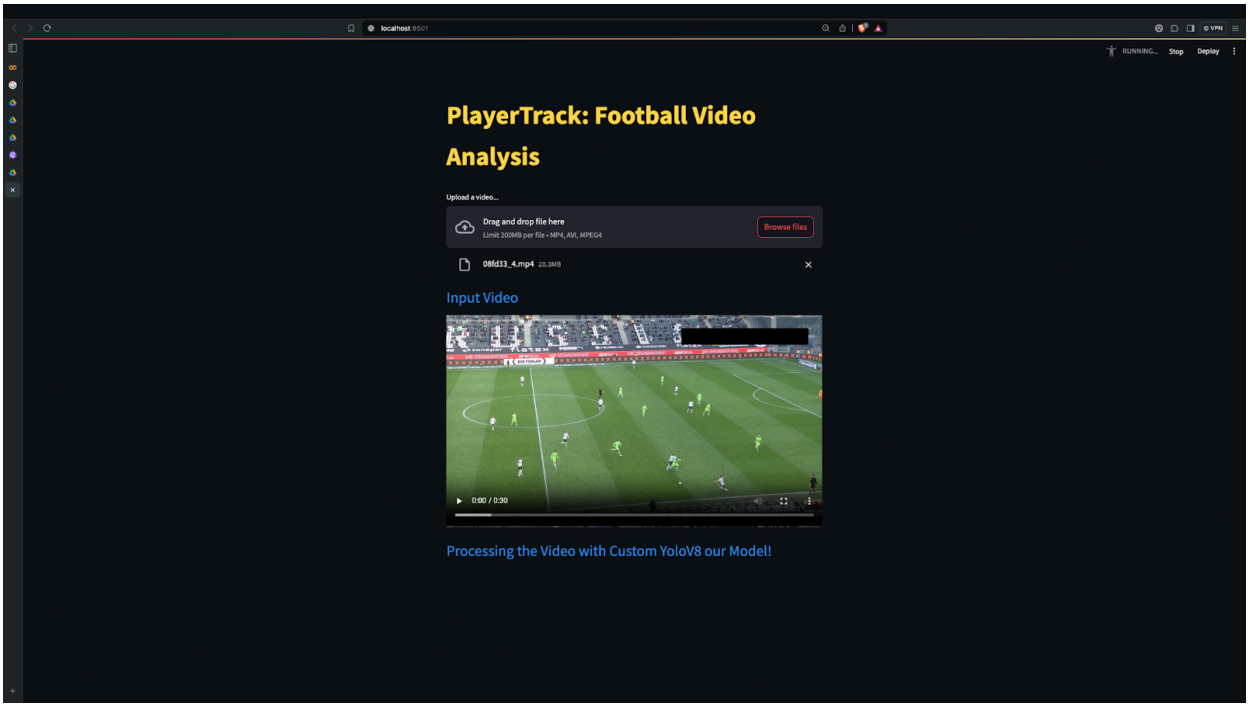
Display Results: After processing, the video with detected objects (players) is displayed on the Streamlit app interface. Users can see the processed video with bounding boxes around players, allowing them to analyze the players' positions and movements throughout the video.

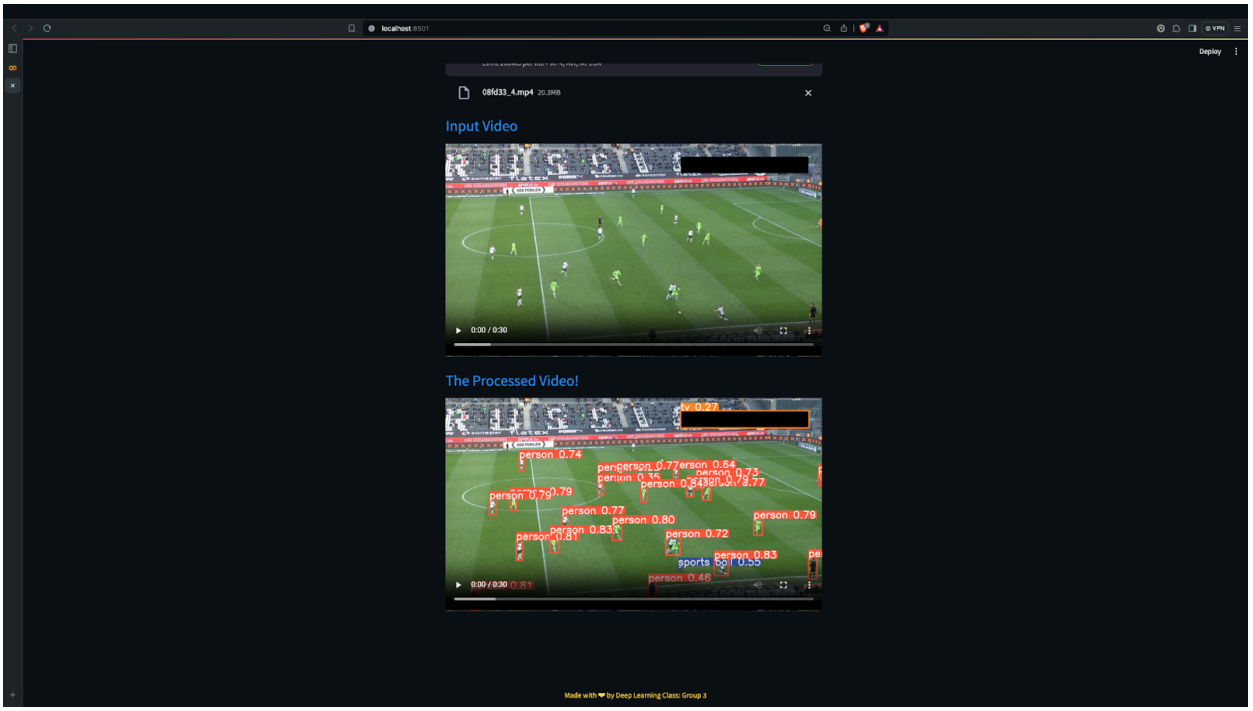
Balloon Animations: When a user uploads a video and the detection process starts, colorful balloon animations appear on the screen. This playful animation serves a dual purpose: it not only visually entertains the user but also subtly indicates that the application is actively processing the input. The balloons float across the user interface, creating a light and engaging atmosphere that reduces the perceived waiting time.

Bouncing Text: Alongside the balloon animations, I incorporated bouncing "Title Text". This text bounces around the screen, drawing the user's attention and providing real-time work-in-progress-like status in a lively and visually appealing manner. The bouncing text again helps maintain user engagement during the wait time and ensures users are aware that the model is working correctly and efficiently in the background.


localhost:8501
Deploy
PlayerTrack: Football Video Analysis
Upload a video...
Drag and drop file here
Limit 200MB per file • MP4, AVI, MPEG4
Browse files
08fd33_4.mp4 20.3MB
Made with ❤ by Deep Learning Class: Group 3


localhost:8501
RUNNING...
Stop
Deploy
PlayerTrack: Football Video Analysis
and drop file here
200MB per file • MP4, AVI, MPEG4
Browse files
08fd33_4.mp4 20.3MB
Input Video
0:00 / 0:30
Video with Custom YoloV8 our Model!


localhost:8501
RUNNING...
Stop
Deploy
PlayerTrack: Football Video Analysis
Upload a video...
Drag and drop file here
Limit 200MB per file • MP4, AVI, MPEG4
Browse files
08fd33_4.mp4 20.3MB
Input Video
0:00 / 0:30
Processing the Video with Custom YoloV8 our Model!


localhost:8501
Deploy
08fd33_4.mp4 20.3MB
Input Video
0:00 / 0:30
The Processed Video!
person 0.74
person 0.79
person 0.77
person 0.80
person 0.83
person 0.72
person 0.79
person 0.83
sports ball 0.55
person 0.46
Made with ❤ by Deep Learning Class: Group 3