TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

LALITPUR ENGINEERING COLLEGE

SOCCER ANALYSIS

BY

ANISH MANDAL [LEC078BCT002]

KRISTAL SHRESTHA[LEC078BCT014]

PRADIP POKHREL MAGAR [LEC078BCT023]

RADHESH RAM GAMAL [LEC078BCT027]

A PROJECT REPORT

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING

IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR

THE DEGREE OF BACHELOR OF ENGINEERING IN COMPUTER ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

LALITPUR, NEPAL

2025

TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

LALITPUR ENGINEERING COLLEGE

A Project Report

on

Soccer Analysis

Submitted By

Anish Mandal [LEC078BCT002]

Kristal Shrestha[LEC078BCT014]

Pradip Pokhrel Magar [LEC078BCT023]

Radhesh Ram Gamal [LEC078BCT027]

Submitted To:

Department of Computer Engineering

Lalitpur Engineering College

Lalitpur, Nepal

In partial fulfillment for the award of Bachelor of Engineering in Computer Engineering

Under of the supervision of

Er. Praches Acharya

2025

ACKNOWLEDGEMENT

We are deeply grateful to all those who have contributed to the successful completion of our proposal report.

First and foremost, we would like to express our sincere thanks to our supervisor, **Er. Praches Acharya**, of **Thapathali Engineering Campus**, for providing invaluable guidance, encouragement, and constructive feedback throughout the planning and research phases.

We are also deeply thankful to **Er.Sandesh Sharan Poudel** , the **Project Coordinator**, for his continuous support, guidance, and valuable suggestions, which have greatly contributed to the clarity and direction of our proposal.

Furthermore, we extend our gratitude to our **HOD, Er. Bibat Thokar**, whose valuable suggestions helped in the finalization of our proposal.

We are also grateful to our classmates and friends for offering us advice and moral support. To our families, thank you for encouraging us in all our pursuits and inspiring us to follow our dreams. We are especially grateful to our parents, who have supported us emotionally, believed in us, and always wanted the best for us.

Additionally, we appreciate the support and collaboration of our peers and team members, whose dedication and creative inputs have enriched our project proposal.

ABSTRACT

Football match analysis has become much better with advances in computer vision and machine learning, enabling players, the ball, and key game events to be automatically and accurately tracked. Manual observation or expensive external tracking systems were previously used in traditional analysis, which were not scalable and time-consuming. YOLOv5, a robust object detection model, is here used to detect and track numerous objects from actual football match videos with good and accurate results. The model is trained on an annotated dataset to detect players, referees, and the ball. ByteTrack is employed for the enhancement of player tracking, retaining identities even when the players are overlapping or temporarily occluded. In addition, perspective transformation techniques are used to correct camera distortion from non-standard angles of view, to get an even better mapping of player movement onto a standard field of vision. The system is run in a web interface using Flask, as a move towards facilitating offline usage and automatic computation of performance metrics like player speed, distance covered, and ball possession. These figures are valuable information on team and player performance that can be used to guide coaches, analysts, and commentators in their decision-making. The process is cost-effective, scalable, and accurate substitute for traditional methods, revolutionizing football analytics through the application of cutting-edge object detection, calculation of the performance metrics, and interactive visualisation. Overall, the project shows the potential of machine learning to deliver actionable insight, improve strategic planning, and aid match evaluation.

Keywords: *ByteTrack, football analysis, K-means clustering, object detection, perspective transformation, player tracking, speed calculation, sports analytics, team assignment, YOLO*

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

API	Application Programming Interface
BiFPN	Bi-directional Feature Pyramid Network
CNN	Convolutional Neural Network
COCO	Common Objects In Context
CSP	Cross Stage Partial
CSS	Cascading Style Sheets
CV	Computer Vision
CUDA	Compute Unified Device Architecture
DNN	Deep Neural Network
FPN	Feature Pyramid Network
HOM	Homography Matrix
HTML	Hypertext Markup Language
IoU	Intersection over Union
JS	Javascript
JSON	JavaScript Object Notation
LKF	Lucas-Kanade Optical Flow
mAP	Mean Average Precision
NMS	Non-Maximum Suppression
Numpy	Numerical Python
PAN	Path Aggregation Network
Pkl	Pickle File Format
ReLU	Rectified Linear Unit
REST	Representational State Transfer
SGD	Stochastic Gradient Descent
SiLU	Sigmoid Linear Unit
SMM	Simple Multiclass Model
SPP	Spatial Pyramid Pooling
SPPF	Spatial Pyramid Pooling - Fast
SVM	Support Vector Machine
YOLO	You Only Look Once

1 INTRODUCTION

1.1 Background

Football is a highly popular sport globally, with millions of coaches, commentators, and fans constantly seeking better ways of analyzing players and games. Manual observation is traditional football analysis, which is inaccurate, subjective, and time-consuming. Artificial intelligence and computer vision have enabled football match analysis to be automated, providing useful information on player movement, ball possession, and other game aspects. Object detection is the most critical technology in such automation, supporting ball, player, and referee tracking in football videos. While Faster R-CNN, SSD, and RetinaNet models have been discovered to provide effective object detection in football videos, they have been discovered to lag behind in speed and accuracy. YOLO (You Only Look Once) is a speedy and high-performing object detection model that is speed-efficient as well as precision-efficient, and the best one to utilize in order to analyze football matches. YOLO is an immensely helpful tool to utilize in regards to automating the analysis of football matches as it performs better than other methods.

1.2 Motivation

Growing interest in data-driven sports analytics leads to a request to efficiently perform tasks like tracking the performance of individual players, estimating the players' speed, or ball movement tracking without access to expensive GPS trackers or with less manual annotations. Automatic players, referees, and ball detections are capable of giving much insights on the speed or distance that every moving component makes, allowing teams to elevate the level of their tactical and overall game analytical performances.While several studies have been conducted regarding football player detection using deep learning models, most approaches developed so far face various challenges such as occlusion handling, and object tracking. This project shall deal with the challenges mentioned above by adopting YOLO for object detection and incorporating appropriate tracking algorithms that can estimate player speed and distance covered.

1.3 Problem Statement

Traditional football match analysis is often inefficient, time-consuming, and subjective, as it relies on manual observation or expensive external tracking systems. While

some studies have explored deep learning-based object detection models such as Faster R-CNN and SSD for detecting players, the ball, and referees, these approaches often struggle with occlusion handling, tracking accuracy, and processing fast-moving objects. Moreover, real-time processing remains a challenge due to computational constraints and model limitations.

This report presents the development of an offline football video analysis system capable of accurately detecting and tracking players, referees, and the ball while computing key performance metrics such as speed and distance covered. By leveraging YOLO for object detection and ByteTrack for robust tracking, the proposed system aims to provide a more efficient and cost-effective alternative to traditional methods, enhancing match evaluation and performance analysis for sports analysts, coaches, and broadcasters.

1.4 Project Objectives

The objective of our project is

- To develop a object detection and tracking system for football videos.
- To analyze player movement by calculating key performance metrics such as speed and distance covered.

1.5 Scope and Application

The aim of this project is to design an end-to-end football match analysis system utilizing AI and machine learning to interpret and analyze football matches from video recordings. The system utilizes the YOLOv5 model, a state-of-the-art object detection model, to detect and track players, the ball, and other objects from football match videos. The analysis will be done on pre-recorded match videos and will provide substantial insights into the movement of the players, possession of the ball, and other important metrics.

The project will involve training the YOLOv5 model with a dataset of football match images to detect and recognize players and objects on the field. The trained model will be utilized in a Flask web interface for users to upload videos of matches to analyze. Analysis will include player tracking, ball possession calculations, and other perfor-

mance metrics, which can be shown interactively. This data will assist coaches, analysts, and commentators in making informed decisions and comprehending the game better.The system will give comprehensive game statistics like distance traveled, speed, giving detailed views of player performance and team performance.

While the system here will concentrate on analyzing captured video footage, there are future possibilities where real-time is added in for live video casts.

The applications of our project are as stated below:

- **Player Performance Analysis:**
  The system provides in-depth analysis of player movement, ball control, speed, distance covered, and other performance indicators. The management, analysts, and coaches can use them to analyze players' capability, fitness, and game performance.

- **Tactical Analysis:**
  The ability to track player location and movement during the game enables the monitoring of tactics closely. The data can be utilized by analysts and coaches to analyze team shape, passing patterns, and strategy, with a sense of how successful each team's strategy is.

- **Player Scouting:**
  The system is utilized by player scouts and clubs to monitor players' performance in games. With the detailed statistics, the scouts make objective player purchase and investment decisions based on measurable facts rather than personal opinions.

- **Sports Broadcasting Potential:**
  Although the system is not real-time yet, the integration of player tracking and performance data can transform sports broadcasting. With further optimizations, the system can potentially provide real-time visualization of data, dynamic player tracking, and end-to-end statistical analysis in real-time for live streams, making viewing more interactive and informative.

1.6 Originality of project

Our project takes a fresh approach to soccer match analysis by combining powerful computer vision techniques into a simple, web-based tool. Unlike existing systems that rely on manual tracking or expensive multi-camera setups, our solution works with just a single camera. It uses YOLO for object detection, ByteTrack to track players, and optical flow to estimate camera movement,all seamlessly integrated into a Flask-based web application.To enhance analysis, we also use KMeans clustering to automatically separate teams based on jersey colors, apply perspective transformation to correct field distortion, and estimate player speed and distance covered. While top-tier soccer analytics platforms leverage massive datasets, multi-camera tracking, and deep learning for predictive insights, our goal is different.We want to make advanced match analysis more accessible, affordable, and open-source.With our user-friendly interface, anyone can upload gameplay footage and receive insights like player tracking, ball possession stats, and movement analysis—all without needing specialized hardware. By balancing automation, affordability, and ease of use, our project makes high-quality soccer analytics available to a much wider audience.

1.7 Organization of Project Proposal

Chapter 1 :**Introduction:** It deals with the introduction of the project along with the background, motivation, problem statement, project objectives, and scope of the project.

Chapter 2: Literature Review: It deals with all the literature articles and papers taken in account for the ideas and motivation to associate this project.

Chapter 3: Requirement Analysis: It deals with requirements analysis such as hardware and software requirements.

Chapter 4: System Architecture and Methodology: It consists of all system architecture, algorithm,, expression and software languages used for the development of project.

Chapter 5: Implementation Details: It explains the working principle of the model and the dataset to be used in this project.

Chapter 6: Result and analysis: It deals with the results part of the project . All the output of various scenarios (best case and worst case analysis). Figures, table and graph the defines the data set are shown. Tabulated and plot performance metrics that are used to verify the output are shown.

It deals with all results and discussion part required to enumerate the project with all the charts, snapshots and other aspects.

Chapter 7: Conclusion And Future Recommendation: It contains summary of our project and its future recommendation/enhancements.

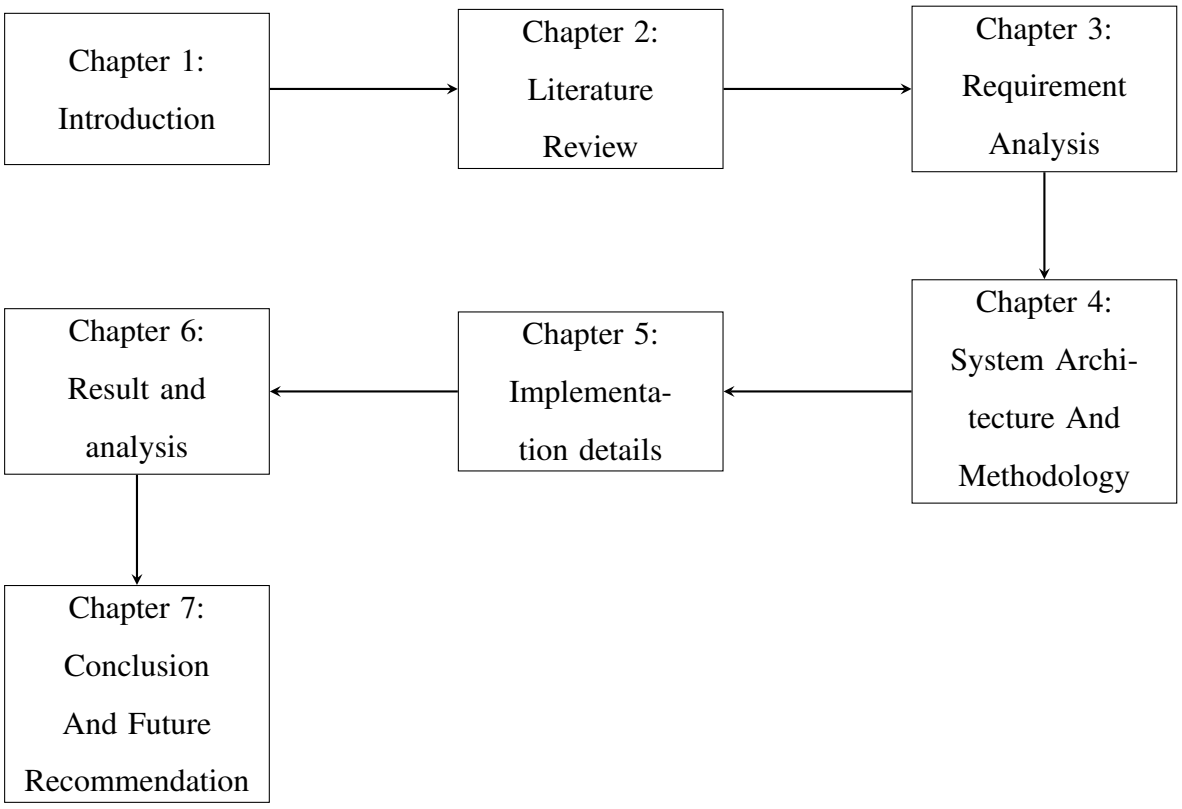

Figure 1.1: Organisation of Project Report

2 LITERATURE REVIEW

The use of artificial intelligence (AI) and machine learning (ML) in sports analytics has evolved significantly over the years. Early sports video analysis primarily relied on manual observation and rule-based approaches, which were limited in scalability and complexity. However, advancements in ML techniques have enabled AI to automatically track and analyze players and objects in football videos, offering new possibilities for performance evaluation.

In the early 2000s, computer vision techniques such as background subtraction and motion detection were used to identify players and track their movements. While these methods were effective in controlled environments, they struggled in dynamic sports settings where occlusions and high-speed movements were common. To address these limitations, researchers explored more advanced tracking algorithms, including the Kalman filter and particle filter, which improved player location predictions and identity maintenance across frames [1]. However, these approaches still faced difficulties in handling complex interactions between players and camera motion.

A major breakthrough came with the development of deep learning-based object detection models, particularly YOLO (You Only Look Once). YOLO revolutionized object detection by providing fast processing and high accuracy, enabling efficient detection of players and the ball in each frame. Before YOLO, object detection relied on slower, computationally expensive techniques such as sliding windows and region proposal networks, which were impractical for fast-paced sports like football [2]. As object detection improved, research shifted towards more sophisticated tracking methods. Early tracking approaches used basic motion models or heuristic rules to estimate player trajectories, but they often are slow and fails in frequent occlusions. The introduction of tracking-by-detection algorithms such as DeepSORT significantly improved tracking accuracy by integrating deep learning-based detection with identity tracking across frames [3].

Despite improvements in detection and tracking, camera motion remains a major challenge in football analysis. Matches are recorded from various angles, and dynamic camera movements such as panning and zooming complicate object tracking. Early ap-

proaches like fixed background subtraction proved ineffective in such scenarios. More recent studies have incorporated optical flow techniques to estimate pixel movement between frames, helping compensate for camera motion and improve tracking accuracy [4]. Another key challenge is perspective distortion, caused by varying camera angles. Perspective transformation methods, such as homography, help project the playing field onto a bird's-eye view, allowing for more precise measurement of player movements. While early methods required manual calibration, modern techniques employ automated perspective adjustments for improved accuracy [5].

As AI-driven sports analytics evolved, performance metrics became more sophisticated. Initially, research focused on basic statistics such as player positioning and ball possession. However, improvements in tracking technology enabled the computation of key metrics such as player speed, distance covered, and acceleration. These insights provide valuable information on player performance, fitness, and tactical strategies. While some applications require real-time metric calculations, many scenarios such as post-match analysis and tactical reviews,benefits from offline processing, where accuracy is prioritized over real-time decision-making [6].

The integration of tracking data with interactive visualizations has further enhanced sports analytics. Early systems relied on static visualizations, whereas modern platforms offer dynamic, interactive displays that allow analysts, coaches, and broadcasters to explore match data in detail [7]. These advancements have significantly improved tactical analysis and team strategy development.

In conclusion, football match analysis has transitioned from manual observation and rule-based techniques to advanced AI-driven methods that leverage deep learning for detection, tracking, and performance evaluation. While real-time analysis has gained traction, offline processing remains crucial for in-depth post-match analysis. This research builds on recent advancements by developing an offline football video analysis system that integrates object detection, tracking, and performance metric computation. By leveraging models such as YOLO and ByteTrack, the proposed system aims to offer an efficient and scalable solution for football analytics, contributing to the broader field of sports performance evaluation.

3 REQUIREMENT ANALYSIS

3.1 Functional Requirements

Functional requirements specify what the system should do. For our Soccer Analysis System, the following functional requirements can be identified:

- **Object Detection:**This feature will help in detecting players and referees on the field as well as provide localization of the football inside the video footage using a YOLO model. The system will be implemented to present bounding boxes around the detected objects along with class labels and confidence scores.
- **Player Tracking:** The system must track players and the football across frames in the video. Each detected player should be assigned a unique ID for tracking purposes.
- **Team Assignment:**The system must assign players to teams based on their jersey colors using color segmentation and clustering techniques (e.g., K-means).
- **Ball Acquisition Measurement:**The system should calculate and display the ball acquisition percentage for each team during the match.
- **Camera Movement Analysis:**The system must analyze and estimate camera movement using optical flow techniques. The system should adjust player positions based on camera movement to provide accurate metrics.
- **Perspective Transformation:**The system must transform the camera's distorted viewpoint into a normalized perspective to measure player movement in meters.
- **Speed and Distance Calculation:** The system should calculate and display the speed of players in kilometers per hour and the distance covered in meters.
- **User Interface:**The system should provide a visual output (video) with annotations showing detected objects, player IDs, team colors, speed, and distance metrics.
- **Data Storage:**The system stores processed data, which includes detection results, tracking information, and calculated metrics, for further analysis in a pickle file.

3.2 Non-Functional Requirements

Non-functional requirements define how the system performs its functions. Key non-functional requirements for our project include:

- **Performance:** The system processes video frames from recorded footage to provide detailed post-match analysis, enabling accurate detection and tracking of players, referees, and the ball in match videos.The object detection and tracking algorithms works even for high frame rate (e.g., 24 FPS standard).
- **Usability:** The user interface must be intuitive and easy to use for users with different levels of technical experience.The system must give clear visual feedback and annotations on the video output.
- **Security:** The system must guarantee that any stored information is safe and unreachable to unauthorized parties.
- **Scalability:**The system should be able to handle videos of varying lengths and resolutions without significant degradation in performance.
- **Accuracy:** The system must have a high rate of accuracy in detecting and tracking the football and players with a confidence level higher than a given threshold (e.g., 0.5).
- **Maintainability:** The codebase must be properly documented and modular for easy updates and maintenance in the future. The system must enable easy integration of new features or enhancements.

3.3 Instrumentation Requirements

Hardware Tools:

A computer that has a high-speed CPU, high-performance GPU, along with a minimum of 16 GB of RAM shall be used for developing and training the system while medium-speed CPU and medium-performance GPU can run the system, or hosting the website (future enhancement).

3.3.1 Software Tools

- **Python:** implements backend logic, trains the model, and handles data analysis and video processing.
- **Flask:** Builds the web application for user interaction.
- **Ultralytics:** A library used for training object detection model.
- **Matplotlib:** Library used for data visualization and plotting.
- **Pandas:**Library used for data manipulation and analysis.
- **Visual Studio Code:**Integrated development environment for writing and debugging code.
- **OpenCV:** Library used for image and video processing including object detection,tracking and optical flow analysis.
- **NumPy:** Performs numerical operations and handles array.
- **HTML/CSS/JavaScript:** Used for designing and implementing the web interface.

3.4 Feasibility Study

3.4.1 Technical Feasibility

The project is technically feasible because:

- **Libraries for Object Detection:**
  Libraries like Ultralytics, YOLO, and OpenCV are among the most developed and well-supported object detection libraries and are therefore greatly well-placed to solve football video analysis market problems. They enable efficient object detection and tracking of significant objects such as players, referees, and the ball in football videos. Ultralytics is a performance-optimized YOLO implementation with the capability to provide fast and accurate detection performance. YOLO (You Only Look Once) is widely recognized for faster processing and very accurate results, while OpenCV is a computer vision library featuring general-purpose computer vision capabilities such as image processing, object detection,

and tracking, which can be custom-fit to individual specifications. Together, the libraries are essential in the automatic analysis of football matches by facilitating accurate and efficient object detection to improve the overall comprehension of game dynamics.

- **Frontend Technologies:**
  The frontend technologies like HTML,CSS,JavaScript are mature enough to make interactive, responsive interfaces for the presentation of football match analysis, hence granting a good user experience no matter from which device.

- **Data Analysis Libraries:**
  Libraries like Pandas and Matplotlib would be good for processing players' statistics and movement visualization in order to handle large datasets efficiently with clean visual reports.

3.4.2 Operational Feasibility

- **User-Friendly Interface:** We aimed at developing a system that was easy to use, comprehend, and clean, with interactive interfaces that work in football match analysis, right from the movement of players to even speed and distance based data.

- **Accessibility:** The system is easily accessed through any device connected to the internet, facilitating easy upload of football videos and viewing the analysis results. They can be done on the website through a web browser. This enables the system to be accessible so that coaches, analysts, and broadcasters can use it without any problems.

- **Post-Match Analysis:** The system is capable of processing recorded football videos to provide detailed performance feedback on players and match dynamics, ensuring accurate analysis without much significant delays.

3.4.3 Economical Feasibility

- **Development Cost:** Development costs are kept quite down with all the tools needed for work, like Ultralytics for object detection YOLO, Flask for backend

development, and OpenCV for video processing, open-sourced and freely available. Moreover, we can also use free libraries like Pandas and Matplotlib, for data analysis and visualization.

- **Maintenance Cost:**We haven't hosted our soccer analysis yet but if we did host then its maintenance cost will be minimum if hosted on cloud platforms such as AWS or Google Cloud, leveraging pay-as-you-go plans or a private server. Because the technologies are open-source, there is no licensing fee, this would significantly reduce the running costs.

4 SYSTEM ARCHITECTURE AND METHODOLOGY

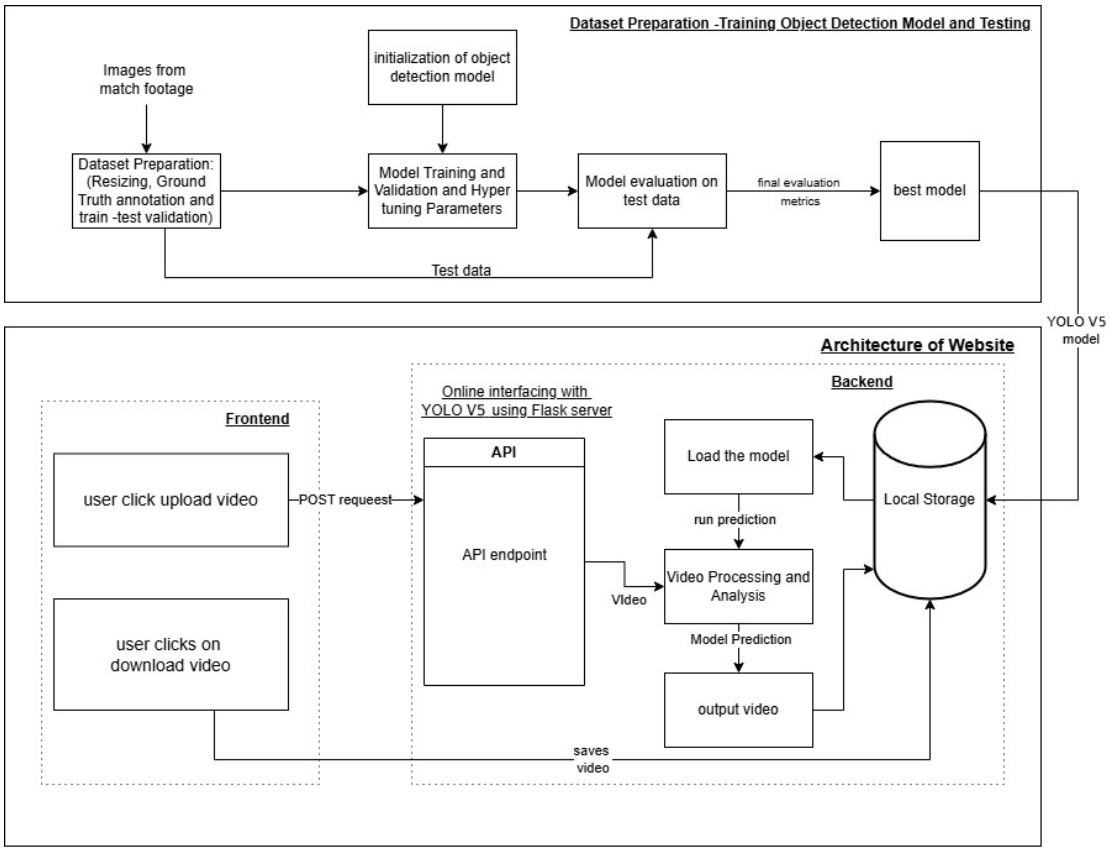

Figure 4.1: System Block Diagram

The two primary tasks in the automatic object identification system of sports based on single-shot detectors are dataset preparation and training of the object detection model. The process begins with the collection of images from match videos, which are preprocessed including resizing, ground truth annotation, and train-test validation splitting. The preprocessed data is used to train the object detection model YOLO v5 with validation and hyperparameter optimization to achieve maximum performance. The models are then tested on test data to get final evaluation metrics. The best-performing model is preserved for our use.

The second task is to develop a website that enables inferencing on the trained YOLO v5 object detection model. The application is a client-server application with frontend and backend functionalities being distinct. The frontend provides a user interface via which users can upload videos to be analyzed and download the analysis results. When a user uploads a video, the frontend makes a POST request to the backend API, which is done through a Flask server. The backend handles the requests by first loading the

trained model from local storage, then video processing and analysis on the uploaded content. In the process, the YOLO v5 model identifies and tracks objects in each frame of the video.

The website architecture allows users to comfortably interact with sophisticated object detection technology through a simple-to-use interface. After completing object detection, the system outputs an overlaid detection result video and saves it to local storage. Users can download the detected video through the frontend interface. This system design effectively bridges the gap between advanced computer vision capabilities and practical applications in the real world, opening up powerful object detection technology to sports video analysis.

4.1 Dataset Preparation

4.1.1 Dataset collection

To train the object detection model for football analysis, a dataset was created by manually extracting frames from football match videos. The frames were carefully selected to capture various match scenarios, ensuring diversity in player positions, ball movement, and referee appearances.Each extracted frame was then annotated using Roboflow, where players, referees, and the ball were labeled accurately. The labeled dataset was further divided into training, validation, and test sets to facilitate effective model training and evaluation.

4.1.2 Image Resizing

The image data that was acquired by extracting frames from videos or by scraping web site was then manually filtered. Then, the filtered images were resized to match the input image size for Object Detection models. Bilinear interpolation technique was used to resize the images.

4.1.3 Ground Truth Annotation

An object detection model must be capable of classifying and localizing every object within an image. For this, alongside the input image the model must be provided with the bounding box coordinates of each class object in the image. A bounding box also referred as ground truth bounding box is nothing but the extreme coordinates (i.e., (Xmin,Ymin) and (Xmax,Ymax) coordinates) of a rectangle that outlines region of

interest by enclosing a class object within an image through the Roboflow website's tools. Using those tools, every image data was labelled. For each image , XML files were utilized to create YOLO Darknet TXT file for YOLOv5 model. These files stored information about every class object and bounding box coordinate corresponding to each class object.

4.2 Data Augmentation

Data augmentation is the process of artificially synthesizing new samples of image data using existing data. Data augmentation increases the size of dataset, helps reducing overfitting, increases generalization and improves performance of a model. Augmentation can be challenging in case of annotated image data because the coordinates of the ground truth boxes must be changed according to the augmentation technique used. Augmentation techniques used for creating the image dataset of monuments from collected images are mentioned below:

4.2.1 Photometric Augmentation:

Photometric augmentation techniques are applied to modify the appearance of an image like changing brightness, contrast, hue, saturation or adding noise. Applying photometric augmentation improves the robustness of the model to different exposures and lighting conditions.

Brightness and Contrast Adjustment

Brightness refers to overall luminosity or darkness of an image. Contrast is a measure of the difference in brightness between the lightest and darkest parts of an image. A high difference between lightest and darkest pixels signifies higher contrast. Both, brightness and the contrast of an image can be changed to augment the image data.

The general procedure for adjusting the brightness of an image involves multiplying each pixel value by a constant factor, which can be greater than 1 to brighten the image or less than 1 to darken it.

The general procedure for adjusting the contrast of an image involves scaling the difference between the pixel values and a mean or median value by a constant factor.

Saturation Adjustment

Hue and saturation, like brightness, are two other aspects of color in the RGB (Red, Green, and Blue) color scheme. Hue refers to the actual color, such as red, green, or blue, while saturation implies the purity of those colors. Brightness indicates how much white (or black) is mixed in the color, while saturation indicates the amount of gray in a color. By manipulating these three aspects of color, an image can be augmented.

To adjust the saturation of an image using an algorithm, the general procedure involves first converting the image from the RGB color space to the HSV color space. Then, the values in the HSV channels can be modified to alter the saturation of the image. Finally, the image is converted back to the RGB color space to view the updated result.

Gaussian Noise Addition

Adding Gaussian noise is a common data augmentation technique used in object detection. This technique involves randomly introducing normally distributed noise to the pixels of an image, which can help the model become more resilient to small changes in the input data. This, in turn, leads to better generalization and improved performance. The noise can be added in a controlled manner, such as by specifying the mean and standard deviation of the Gaussian distribution used to generate the noise.The probability distribution of Gussian distribution is given as:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \tag{4.1}$$

The parameter μ is the mean or expectation of the distribution, while the parameter σ is its standard deviation.

Gaussian Blur Addition

Gaussian blur is a common data augmentation technique used in object detection, which involves applying a blur filter to an image. The blur filter is based on a Gaussian distribution, which smooths the image by convolving it with a kernel that corresponds to the Gaussian function. The Gaussian distribution used for the blur filter is defined by its mean and standard deviation parameters in equation (4.1).

The blur filter works by replacing each pixel in the image with a weighted average of

the pixels in its surrounding area, where the weights are determined by the Gaussian function. The Gaussian function gives higher weights to pixels that are closer to the center pixel, and lower weights to pixels that are farther away. This results in a smoother image that reduces noise and sharp edges.

4.2.2 Geometric Augmentation

Geometric augmentations are applied to change the geometry of an image such as translating, rotating or scaling images. Applying geometric augmentations improves the robustness of the model to the change in object orientation, size and position.

Translation

Translating an image involves shifting its position in either the horizontal or vertical direction, which involves moving all the pixels in the image by a fixed amount in either the x or y direction. This operation can be used to augment data for object detection, thereby increasing the variety of the training data and enhancing the model's robustness to small changes in object position. The general procedure to translate an image involves shifting the position of the pixels in the image by a specified number of pixels along the x and y axes. Then filling the remaining pixels with default values such as white or black.

Rotation

Rotating an image involves turning it around a specific point by a certain angle. This process involves transforming the position of each pixel in the image to a new position based on the rotation angle and the center of rotation. Rotating an image can be a useful tool for data augmentation in object detection, as it can help to expand the variety of training data and enhance the model's resilience to slight changes in object orientation.

The general procedure to rotate an image involves transforming the position of each pixel in the image using a rotation matrix, and then resampling the image to fill in the gaps left by the rotation using the nearest pixel to it. Rotation matrix for rotating pixel(x, y) to (x', y') by an angle about point (m, n) in anticlockwise direction is given by:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & m(1-\cos\theta)+n\sin\theta \\ \sin\theta & \cos\theta & n(1-\cos\theta)+m\sin\theta \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{4.2}$$

4.3 Model Architecture of YOLOv5 Object Detector

4.3.1 High-level YOLO architecture

Object detection models can typically be categorized as either two-stage or single-stage object detectors. YOLO object detectors belong to the single-stage category and are made up of three components - a backbone network, a model neck, and a detection head that make dense predictions, as shown in the diagram below: The input image is first

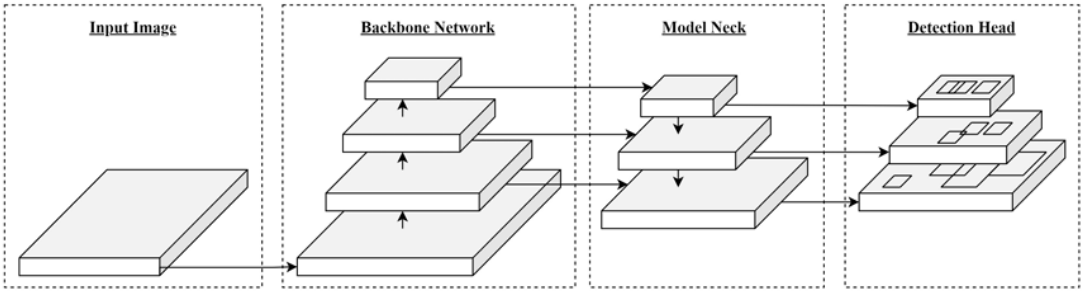

Figure 4.2: High-level architecture of YOLO family

processed by a backbone network such as VGG16, ResNet-50, or CSPDarknet-53, to extract multiple feature maps from an intermediate layer. Our system uses CSPDarknet-53 as backbone network. Additional high-level feature maps are generated by convolving the base feature maps with different filters of various sizes and strides, which allows for the extraction of more complex and detailed features from the input image. The Model Neck integrates these image features by merging a sequence of layers before passing them forward for prediction. It includes supplementary components such as SPP (Spatial Pyramid Pooling), ASPP (Atrous), and Path-aggregation blocks like FPN (Feature Pyramid Network), PAN (Path Aggregation Network), and BiFPN. Finally, the detection head applies anchor boxes on the feature maps and renders the final output, including classes, objectness scores, and bounding boxes.

4.3.2 YOLOv5(You Only Look Once vesion 5)

YOLOv5 is a state-of-the-art object detection algorithm released in 2020 by Ultralytics, a computer vision company. YOLOv5 comes in four distinct versions: small (s), medium (m), large (l), and extra-large (x). These versions are classified according to the memory storage size, but the principle is the same. Although there is no official documentation on the implementation details of YOLOv5, the architecture still follows the backbone network, neck model, and dense prediction head of the previous versions of YOLO. All versions of YOLOv5 are composed of three components, that include CSP-Darknet53 serving as the backbone, SPP and PANet integrated into the model neck, and the same detection head utilized in YOLOv3. YOLOv5 has introduced several improvements over its earlier versions, such as the implementation of the Focus layer and utilization of CSP (Cross-Stage Partial Connections) in the bottleneck layers. The primary objective of the Focus layer is to reduce the number of layers, parameters, FLOPS, and CUDA memory usage while improving the speed of both forward and backward operations, without significantly affecting mAP. This was achieved by consolidating the first three layers of YOLOv3 into a single layer, constructed using convolutional layers. For example, a Focus layer with a kernel size of 3 can be represented as a convolutional layer with a kernel size of 6 and a stride of 2. The expanded architectural block diagram for the YOLOv5 6.0 is shown below:

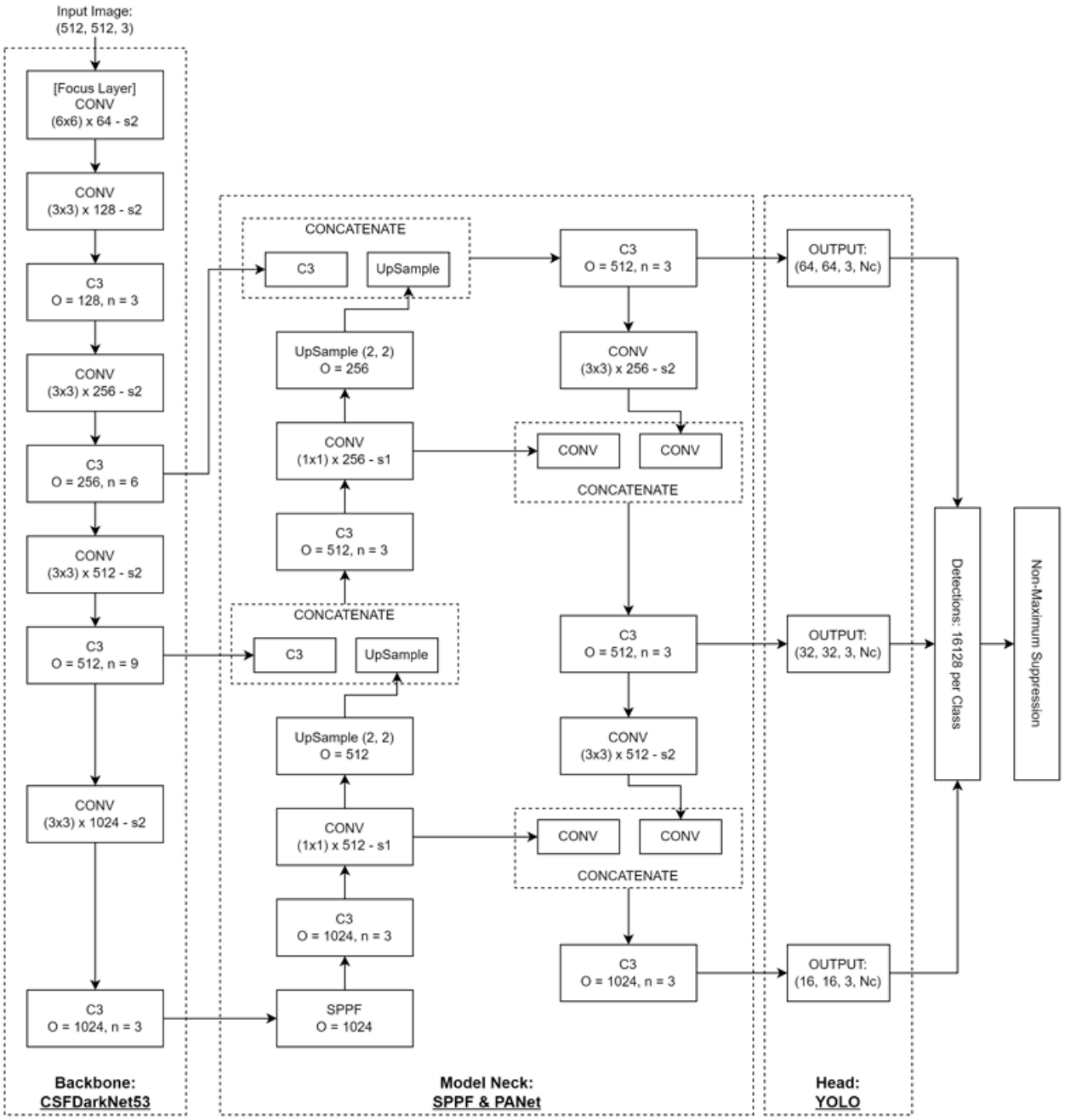

Figure 4.3: Architecture block diagram of YOLO v5 model

(a) **CONV Layer**

In the YOLOv5 architecture, the fundamental building block of a convolutional layer comprises of a conventional Conv2d layer, succeeded by a Batch Normalization layer. The activation function of choice in this architecture is SiLU (Sigmoid Linear Unit).

(b) **CSPBottleneck (C3) Layer**

The CSPBottleneck layer is a key component of the YOLOv5 object detection architecture. It stands for "Cross-Stage Partial Bottleneck" layer and is designed to improve the accuracy and speed of the model. The CSPBottleneck layer consists of two main parts: a bottleneck block and a cross-stage hierarchy connection. The bottleneck block consists of a CONV-BN-SiLU (convolutional layer, batch normalization layer and SiLU activation function) sub-blocks, with a number of residual bottleneck layers. The cross-stage connection, a technique introduced by CSPNet, is a new strategy that truncates the gradient flow to decrease the excessive amount of redundant gradient information. YOLOv5 employs CSPNet strategy to partition the feature map of the base layer into two parts: one part goes through the convolution block followed by a number of bottleneck layers with or without skip connection, while other part directly concatenates to the output from the first part. As a result of this entire process, the spatial dimension is halved.

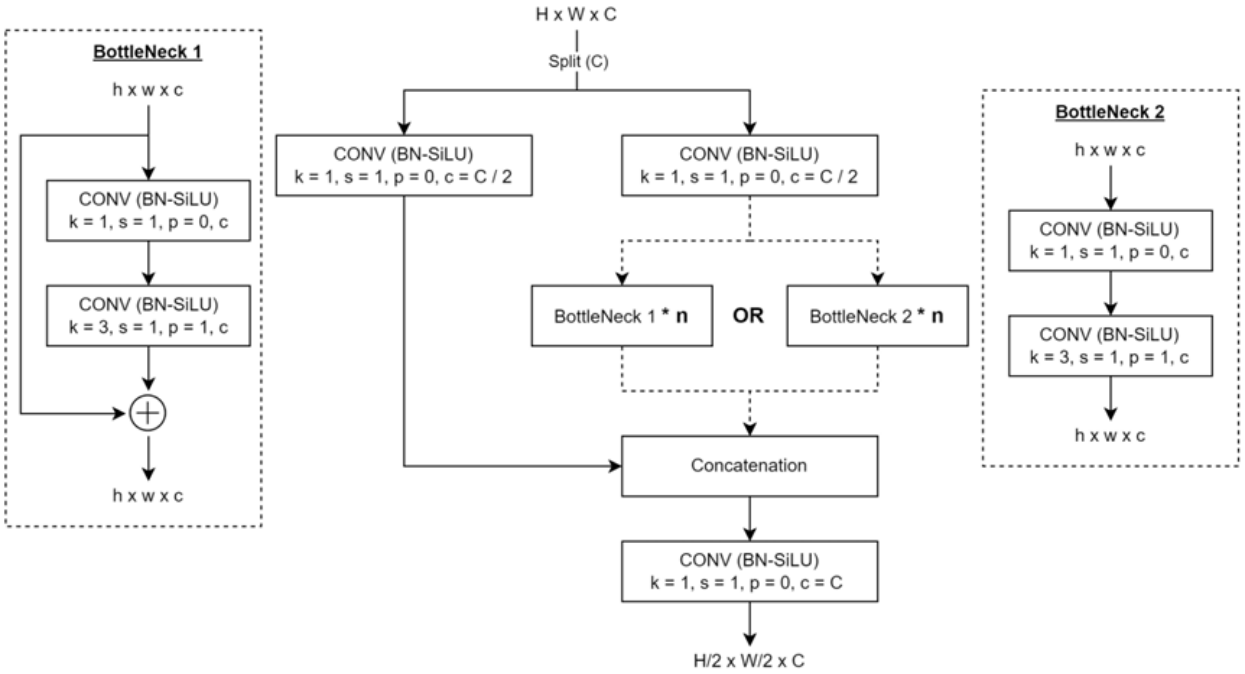

Figure 4.4: CSPBottleneck Layer Architecture

(c) **Spatial Pyramid Pooling – Fast (SPPF) Layer**

The Spatial Pyramid Pooling (SPP) layer is a type of pooling layer that enables convolutional neural networks (CNNs) to process images of varying sizes. Un-

like other pooling techniques that require fixed input sizes, the SPP layer allows the network to handle inputs of different dimensions. In YOLOv3, the SPP layer is positioned after the final convolutional layer in the Darknet-53 backbone network, and it plays a critical role in extracting important spatial information from the feature map. Furthermore, the SPP layer incorporates a stitching process that connects the features together, enhancing their quality and improving the network's performance. This is particularly useful for blurred images with densely distributed objects where the SPP layer can aid in identifying and detecting objects with greater accuracy. SPP-Fast is a variation of SPP that has been modified to include a fast spatial correlation layer, which enhances the network's ability to perform spatial pooling more efficiently by taking advantage of the correlation between neighboring feature maps. In SPPF, the input undergoes a convolutional operation that reduces the channel depth by half. The resulting output is then processed by multiple MaxPool2d layers, which perform spatial correlation by transitioning outputs from one layer to the next. Finally, all the outputs are concatenated with an additional convolutional layer to maintain the original channel depth of the input. This entire process is illustrated in the figure below:

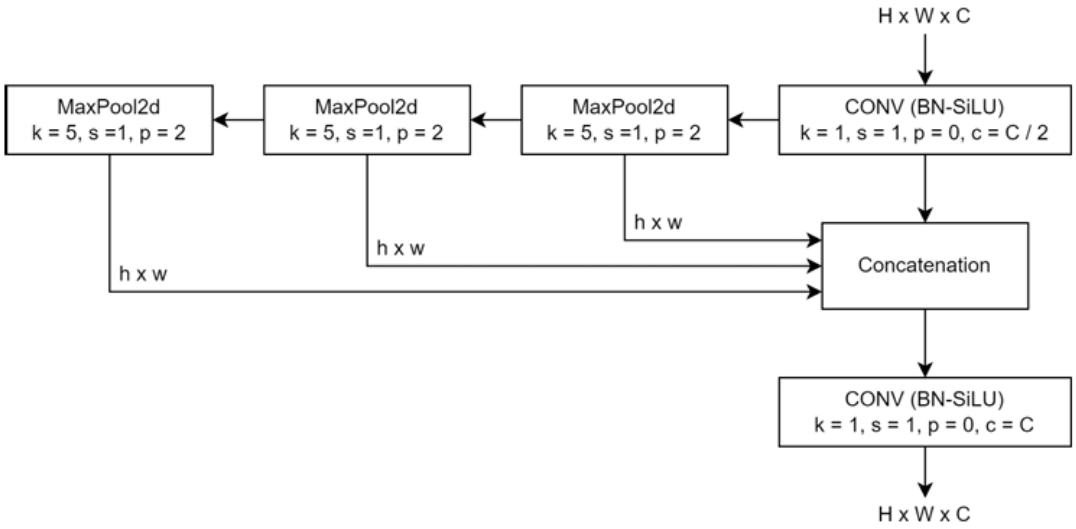

Figure 4.5: Spatial Pyramid Pooling - Fast (SPPF) Layer Architecture

(d) **Path Aggregation Network (PANet)**

PANet, which stands for Path Aggregation Network, is a feature fusion module that is commonly used in convolutional neural networks. Its main purpose is

to merge feature maps from various levels of the network, enabling the network to capture both lowlevel and high-level features, leading to better performance on complex tasks. In YOLOv5, a modified version of PANet is utilized, where some of the regular convolutional layers are replaced with CSPBottleneck (C3) layers, as illustrated in Figure 4.4. The features obtained from different levels of the CSPDarknet-53 backbone, as well as the output of the SPPF, are combined through a bottom-up path, and then undergo top down path aggregation. This results in outputs from three different sections that are fed into the detection head module.

(e) **YOLO Head**

The YOLOv5 architecture utilizes the same head as YOLOv3 and YOLOv4, which consists of three convolutional layers. These layers are responsible for predicting the location of the bounding boxes, as well as the objectness score and classes of objects detected in the image. The YOLO head produces three outputs with dimensions of (H/8, W/8, 3, Nc), (H/16, W/16, 3, Nc), and (H/32, W/32, 3, Nc), where H and W represent the height and width of the input image, respectively, and Nc represents the number of object classes. When the input image size is 512 x 512, the YOLO head generates a total of 16128 detections for each class. These detections are then passed through the Non-Maximum Suppression module, which selects the most confident bounding boxes and eliminates redundant detections to produce the final predictions.

4.4 Initialization of object detection model

4.4.1 Initislizing YOLOv5 model

The YOLOv5s object detection model, developed by Ultralytics, is first instantiated by acquiring the pre-trained weights of the model. The model is pre-trained on the COCO (Common Objects in Context) dataset, which includes more than 1.5 million labeled images spanning 80 different object categories. These categories include common objects like people, animals, vehicles, and household items, making it a diverse and comprehensive dataset. To initialize the YOLOv5s model, the pre-trained weights are loaded into the feature extraction network of the model. This network is responsible for extracting meaningful features from the input images, which are then used by the

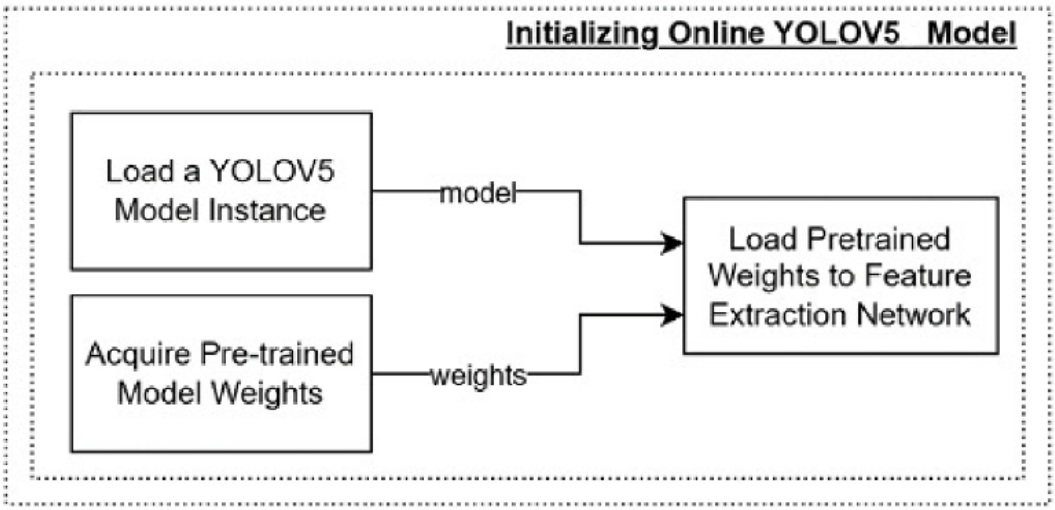

Figure 4.6: Initializing YOLOv5 model

object detection layers to identify and localize objects within the image. The pre-trained weights provide a solid foundation of learned features that can be fine-tuned to better suit the specific task of player,ball,referee detection.

4.5 Training, Validation and Hyperparameter Tuning, and Testing

4.5.1 Training

During training, batches of labeled frames were fed into the YOLO model, which were annotated with player, referee, and ball labels. The model's weights were updated through backpropagation, where gradients of the parameters were computed based on the loss function. This process minimized the difference between predicted and ground-truth labels, enabling the model to improve its ability to accurately detect and localize objects in football match frames.

4.5.2 Validation and Hyperparameter Tuning

Validation is the process used for evaluating the model during training. During training, the model is evaluated on validation data set, and the performance is monitored to determine if the model is overfitting or underfitting.

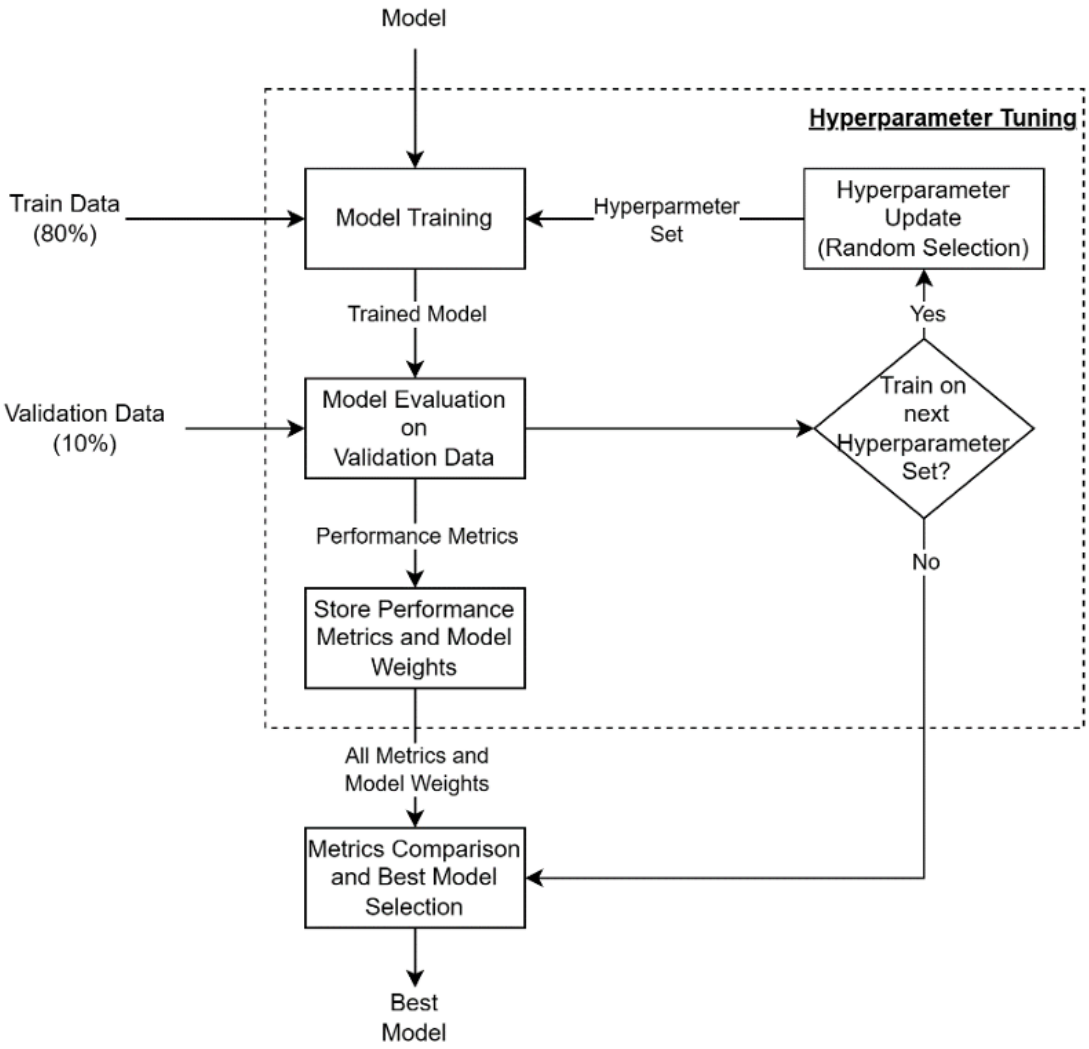

Figure 4.7: Validation and Hyperparameter Tuning using Random Selection

Hyperparameter tuning is the process of selecting the best set of hyperparameters. Selecting the best hyperparameter set can significantly impart the performance of the model on a validation set and ultimately its ability to generalize to new, unseen data. It is essential to tune the hyperparameters to get the best possible model.

Different hyperparameters that are tuned in context of the object detection models include:

a. **Number of Epochs**

   This controls the number of times the model will iterate over the training data. More epochs can improve the accuracy of the model, but may also increase the risk of overfitting.

b. **Batch Size**

This determines the number of samples used in each iteration of the training process. Larger batch sizes can speed up training, but may also require more memory.

c. **Learning Rate**

This controls the step size at which the model updates its weights during training. Setting a higher learning rate can speed up training, but may cause the model to converge to a sub-optimal solution.

d. **Box Regularization Factor**

This parameter controls the weight given to the bounding box coordinates and helps prevent the model from predicting bounding boxes that are too small or too large.

e. **Label Regularization Factor**

The label regularization factor controls the weight given to the objectness score, which determines whether a particular region of the image contains an object or not. The range of the label regularization factor is selected from 1e-3 to 1e-4.

4.5.3 Testing

Testing is a critical step in evaluating the trained object detection model. The model was tested on a separate dataset of labeled frames that were not used during training or validation. This test set included new images from football match videos to assess the model's accuracy and generalization.Evaluation metrics such as mean average precision (mAP), precision, and recall were calculated to measure the model's performance. The results from testing helped determine whether the model was ready for deployment or required further optimization.

4.6 Model fundamentals

Beside the description of different types of layers in base network and interconnection of feature maps with detection heads, other fundamental building blocks are necessary to understand which have been realized in building the model. These fundamentals include the type of activation functions employed, custom loss functions for localization and confidence losses, optimizers and metrics for evaluating the performance of the model on test set.

4.6.1 Activation Functions

Activation functions are a fundamental part of neural networks as they determine whether a neuron should be activated based on the weighted input it receives. The activation function transforms the weighted input into an output value, which is then passed on to the next layer in the network. By introducing non-linearity, activation functions enable the model to learn complex patterns that linear models cannot capture. This non-linearity is critical for tasks like object detection, where the relationships between input features and output predictions are not straightforward. In the context of our object detection project using YOLO, the following activation functions are used:

a. **ReLU activation function**

The rectified linear unit, or ReLU, is a commonly used activation function in neural networks that sets all negative inputs to zero and leaves positive inputs unchanged. ReLU6 is a variation of ReLU that adds a cap of 6 to the maximum activation value, meaning that any input greater than 6 will be set to 6. ReLU6 have shown to empirically perform better under low-precision conditions by encouraging the model to learn sparse features earlier.

$$ReLU(z) = \min(\max(0, z), 6) \tag{4.3}$$

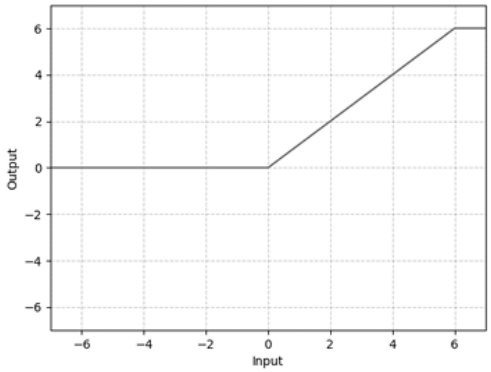

Figure 4.8: ReLU6 activation function

b. **SoftMax Activation Function**

The SoftMax activation function is applied to transform the raw output scores (logits) from the model into a set of probabilities, ranging from 0 to 1, where the sum of all probabilities equals 1. This is particularly useful for multi-class

classification tasks, such as identifying players, referees, and the ball in the football match analysis.In this project, after object detection using YOLO, SoftMax is used to normalize the class scores for each detected object. These class scores, which are the raw output logits from the detection layer, are converted into probabilities. The class with the highest probability is then assigned to the detected object.

Mathematically, for a vector of raw output scores z, the SoftMax function is defined as:

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{4.4}$$

where, z is the vector of raw outputs from the neural network and e = 2.718

The ith entry in the SoftMax output vector SoftMax (z) can be thought of as the predicted probability of the test input belonging to class i.

For example, if the raw scores for a detected object are [8, 5, 0], applying SoftMax converts these values into probabilities. After applying the SoftMax formula, the resulting probabilities might be [0.9523, 0.0474, 0.0003], which sum to 1. This ensures that the model can confidently classify objects into one of the categories (player, referee, or ball).Thus, the SoftMax activation function enables accurate classification of the detected objects in the football video frames.

c. **SiLU Activation Function**

SiLU (Sigmoid Linear Unit), also known as SiLU, is another commonly used activation function in neural networks. SiLU applies the sigmoid function to the input, scaled by the input itself. It can be expressed mathematically as:

$$SiLU(z) = z * sigmoid(z) \tag{4.5}$$

Here, the input value z is multiplied by the output of the sigmoid function, which returns a value between 0 and 1.

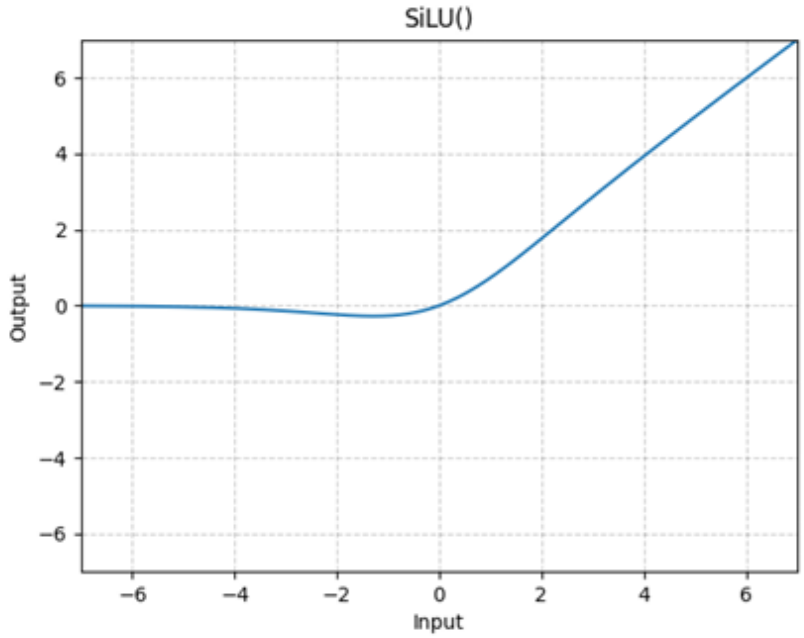

Figure 4.9: SiLU Activation Function

4.6.2 Loss Function

A loss function is a metric that measures the degree of deviation between the predicted values of a machine learning model and the actual (ground truth) values. The loss function takes two inputs: the predicted values from the model and the corresponding actual values. The output of the loss function is a single scalar value that indicates how well the model is performing in terms of its predictions. Essentially, the loss function serves as a guide for the model to adjust its parameters during training to minimize the error between the predicted and actual values.

Generally, in object detection models, two losses are required. The loss for the localization error of bounding box coordinates and the confidence score loss for correct predicting ground truth label. A custom loss function as a combination of two loss is used in the model. Huber loss function is used for bounding box localization loss while Categorical Cross Entropy is used for confidence score.

a. Huber Loss

   Huber loss is a loss function used in robust regression, which is less sensitive to outliers in data than the squared error loss. Huber loss is a loss function that strikes a balance between the advantages of mean-squared error (MSE) and

mean-absolute error (MAE). The Mean-Squared Error (MSE) is a widely used loss function in machine learning that measures the average squared difference between the predicted values and the actual values across the dataset. The MSE is formally defined by the following equation:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{4.6}$$

Where, N is the number of samples being tested against.

While MSE is sensitive to outliers and magnifies the error if the model makes a single very bad prediction due to the squaring part of the function.

MAE is robust to outliers but doesn't punish errors of larger magnitude. Unlike the Mean Squared Error (MSE), which squares the differences between the predictions and ground truth, the MAE calculates the absolute differences and averages them across the entire dataset. The MAE is formally defined by the following equation:

$$\text{MAE} = \frac{1}{n}\sum_{i,j=1}^{n}|y_i - \hat{y}_j| \tag{4.7}$$

Where, n is the number of samples being tested against. Huber loss is designed to be less sensitive to outliers than MSE, while still taking the magnitude of the error into account. It uses a hyperparameter, delta, to determine the point at which to switch from quadratic to linear loss. It can be defined using the following piecewise function:

$$\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2, & \text{if } |y - f(x)| < \delta \\ \delta|y - f(x)| - \frac{1}{2}\delta^2, & \text{otherwise} \end{cases} \tag{4.8}$$

Where, δ is a hyperparameter that controls the split between the two sub-function intervals.

Huber loss for the bounding box loss is calculated as:

$$pred_delta = [delta_y, delta_x, , delta_w, delta_h] \tag{4.9}$$

$$actual_delta = [delta_y, delta_x, delta_w, delta_h] \tag{4.10}$$

$$L_\delta(actual_del, pred_del) = \begin{cases} \frac{1}{2}\sum_{i=0}^{4}(\delta_i - \hat{\delta}_i)^2 & , |\delta_i - \hat{\delta}_i| < \delta \\ \sum_{i=0}^{4}\left(\delta \cdot |\delta_i - \hat{\delta}_i| - \frac{1}{2}\delta^2\right)^2 & , \text{otherwise} \end{cases} \tag{4.11}$$

b. **Categorical Cross Entropy Loss Function** Cross entropy loss function is an optimization function which is used in case of training a classification model which classifies the data by predicting the probability of whether the data belongs to one class or other class. Cross-entropy is given by:

$$CE = -\sum_{i=1}^{n} t_i \log(p_i), \quad \text{for all } n \text{ classes} \tag{4.12}$$

Where, t_i is the truth label and p_i is the softmax probability for the ith class. Both Categorical Cross Entropy and Sparse Categorical Cross Entropy are used when there are two or more label classes. However, for one-hot representation of true labels Categorical Cross entropy loss is preferred.

4.7 Adam Optimizer

Adaptive Moment Estimation (Adam) is a technique for optimizing the gradient descent algorithm that relies on adaptive estimation of first and second-order moments. Unlike traditional stochastic gradient descent, which maintains a constant learning rate for all weight updates, Adam calculates unique and adaptive learning rates for various parameters. This is achieved by estimating the first and second moments of the gradient. Adam combines the features of the gradient descent with momentum algorithm and the root mean square propagation (RMSP) algorithm, resulting in a more efficient and robust optimization technique.

4.7.1 Momentum

Momentum is an optimization technique used to speed up the gradient descent algorithm by incorporating an ”exponentially weighted average” of the gradients. This ap-

proach helps the algorithm converge towards the minimum more quickly by taking into account the past gradients, rather than just the current one.

$$w_{t+1} = w_t - \alpha \cdot m_t \tag{4.13}$$

$$m_t = \beta \cdot m_{t-1} + (1-\beta) \cdot \left[\frac{\delta L}{dw_t}\right] \tag{4.14}$$

where,

- m_t is the aggregate of gradients at time t (current), initially, $m_t = 0$.
- w_t = weights at time t
- α_t = learning rate at time t
- δL = derivative of the loss function
- w_t = derivative of weights at time t
- β = moving average parameter (constant, 0.9)

4.7.2 Root Mean Square Propagation (RMSP)

RMSprop is an optimization algorithm that aims to improve AdaGrad by using exponential moving averages instead of cumulative sum of squared gradients. In this algorithm, the weight updates are scaled by the moving average of the past gradients, giving more weightage to recent gradients than the past ones. This ensures that the learning rate is adaptive and the algorithm can converge to the optimal solution faster.

$$w_{t+1} = w_t - \left(\frac{\alpha_t}{(v_t + \varepsilon)^{\frac{1}{2}}}\right) \cdot \left[\frac{\delta L}{\delta w_t}\right] \tag{4.15}$$

$$v_t = \beta \cdot v_{t-1} + (1-\beta) \cdot \left[\frac{\delta L}{\delta w_t}\right]^2 \tag{4.16}$$

where,

- w_t = weights at time t
- w_{t+1} = weights at time $t+1$
- δ L = derivative of the loss function
- δ w_t = derivative of weights at time t
- v_t = sum of squares of past gradients, i.e., $v_t = \sum \left(\frac{\delta L}{\delta w_{t-1}} \right)^2$ (initially, $v_t = 0$)
- α_t = learning rate at time t
- β = moving average parameter (constant, 0.9)
- ε = a small positive constant (e.g., 10^{-8})

From equation (4.13) and (4.14), m_t and v_t are both initialized as 0, i.e., they are both biased towards 0. The Adam optimizer resolves this issue by calculating bias-corrected $\hat{m}_t$ and $\hat{v}_t$ as:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{4.17}$$

Adam Optimizer general equation:

$$w_{t+1} = w_t - \hat{m}_t \frac{\alpha}{\sqrt{\hat{v}_t} + \varepsilon} \tag{4.18}$$

4.8 Non-Maximum Suppression Algorithm

Non-Maximum Suppression (NMS) is an algorithm used to reduce the number of detected objects in an image by suppressing those with lower confidence scores. It works by comparing the confidence scores of adjacent objects and suppressing the one with the lower score. This helps to reduce the number of false positives and improve the accuracy of the detection. NMS is an essential part of many computer vision applications, such as object detection and facial recognition.

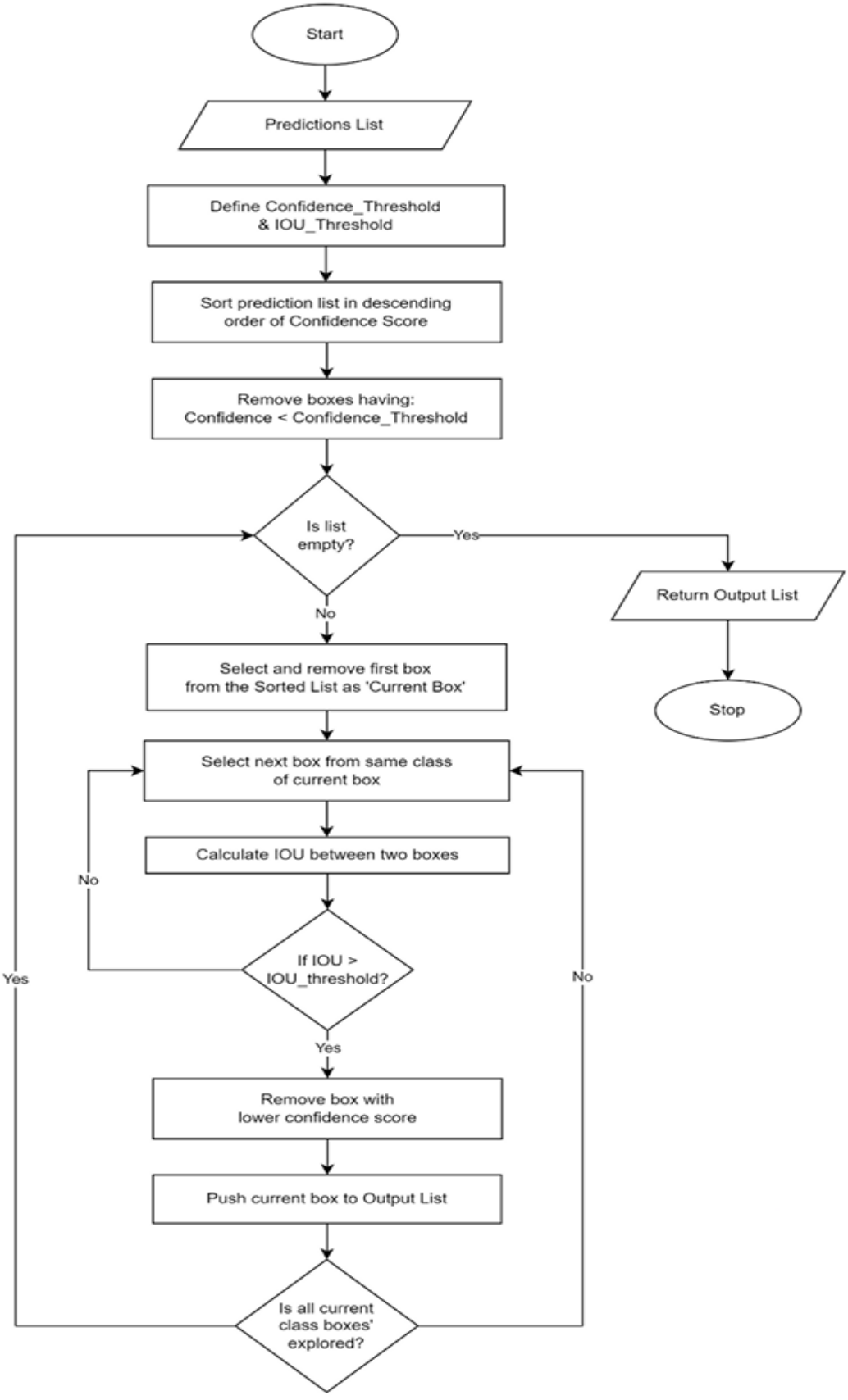

Figure 4.10: Flowchart for Non-Maximum Suppression (NMS) Algorithm

4.9 Model architecture of Bytetrack

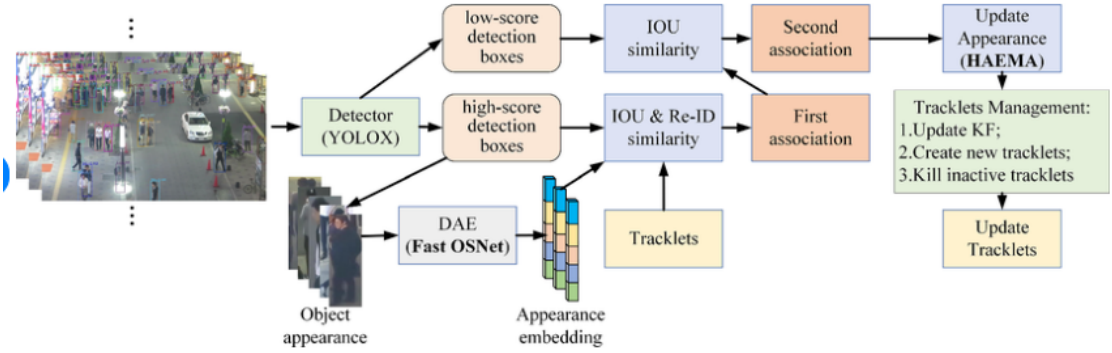

Figure 4.11: Pipeline of Bytetrack

ByteTrack is a multi-object tracking algorithm that enhances tracking performance by associating every detection box, including those with low confidence, with existing trajectories. This approach minimizes the loss of true objects and reduces trajectory fragmentation. By integrating seamlessly with various object detectors, ByteTrack processes detection boxes from each frame, associating them across frames using motion models and assignment algorithms. This comprehensive association strategy ensures stable and accurate tracking, even in complex scenarios. Its ability to track objects with high accuracy and efficiency makes it suitable for applications such as autonomous driving, video surveillance, and augmented reality.

4.10 initializing Bytetracking

Once ByteTracker is selected, it is initialized to track detected objects. The tracker is provided with the initial object detections, including bounding boxes and class labels (players, referees, and the ball) from the YOLO detection stage. Each object is assigned a unique tracking ID by ByteTracker (e.g., player 1, player 2, referee 1, ball). As the tracker processes each frame, it associates the current detections with the correct objects from previous frames, ensuring consistent tracking across the video.

4.11 Kalman Filter for Object Tracking

The Kalman Filter is a recursive algorithm used for estimating the state of a dynamic system from noisy observations. In object tracking, it helps predict the next position of an object based on previous states, effectively smoothing detections and handling occlusions. It operates in two main steps:

4.11.1 Prediction Step

The Prediction Step is the phase where the Kalman Filter predicts the future state of an object based on its previous state and a motion model. This step uses a dynamic model to estimate the object's position and velocity at the next time step.

a. **Predicted State Estimate:**

The state vector at time k, $\hat{X}_k$, is predicted from the previous state X_{k-1}, using the state transition matrix A. The state vector contains the position (x, y) and velocity (v_x, v_y) of the object.

$$\hat{X}_k = AX_{k-1} + BU_k \tag{4.19}$$

where:

- A is the state transition matrix that describes how the state evolves over time.
- X_{k-1} is the previous state (previous position and velocity).
- B is the control input matrix, and U_k is any external control input (e.g., acceleration or external forces), though it might be zero if no control is applied.

b. **Predicted Covariance Estimate:**

The covariance matrix P_k is updated to reflect the uncertainty in the prediction. It is influenced by the previous covariance P_{k-1}, the state transition matrix A, and the process noise Q, which accounts for errors or unmodeled system dynamics.

$$P_k = AP_{k-1}A^T + Q \tag{4.20}$$

where:

- P_k is the predicted covariance matrix, representing the uncertainty in the prediction.
- Q is the process noise covariance matrix, which accounts for errors or unknown disturbances in the motion model.

4.11.2 Update Step

In the Update Step, the Kalman Filter refines the prediction based on the actual measurements (e.g., the detected position of an object). It adjusts the predicted state and covariance by comparing the prediction with the new measurement.

a. **Kalman Gain:**

The Kalman Gain K_k is calculated to determine how much weight should be given to the prediction versus the new measurement. It is based on the predicted uncertainty and the measurement uncertainty.

$$K_k = P_k H^T \left(HP_k H^T + R\right)^{-1} \tag{4.21}$$

where:

- H is the observation matrix, which maps the predicted state space to the measurement space.
- R is the measurement noise covariance matrix, representing the uncertainty in the measurement.
- K_k is the Kalman Gain, which balances the prediction and the measurement.

b. **Updated State Estimate:**

The Kalman Gain is used to update the state estimate. The difference between the actual measurement Z_k and the predicted measurement $H\hat{X}_k$ (residual) is multiplied by K_k to correct the prediction.

$$X_k = \hat{X}_k + K_k \left(Z_k - H\hat{X}_k\right) \tag{4.22}$$

where:

- X_k is the updated state estimate (the corrected position and velocity of the object).
- Z_k is the new measurement (e.g., the new position of the object detected in the current frame).

- $H\hat{X}_k$ is the predicted measurement based on the predicted state.

c. **Updated Covariance Estimate:**

Finally, the covariance matrix P_k is updated to reflect the reduced uncertainty after the correction.

$$P_k = (I - K_k H) P_k \tag{4.23}$$

where:

- I is the identity matrix.
- $K_k H$ is the portion of the Kalman gain that adjusts the covariance based on the measurement correction.

4.12 Edge and Corner Detection:

Computer vision feature detection is the identification of salient points or regions in an image that hold significant information. The most common features are edges, corners, and blobs. These features form the foundation for more complicated tasks such as object recognition, image matching, and object tracking.

An edge is a region in an image where the pixel intensity changes significantly. Edges typically represent boundaries between different objects or surfaces in an image.Edges can be caused by changes in color or texture,changes in surface orientation,variations in scene depth,changes in material properties and Lighting conditions as shadows, highlights.

4.12.1 Harris Corner Detection:

The Harris Corner Detection algorithm, proposed by Chris Harris and Mike Stephens in 1988, stems from the observation that at a corner, intensity in the image changes significantly in most directions.The basic principle of Harris Corner Detection is the study of how the intensity of a small window (patch) of pixels changes as it is translated slightly in different directions:

- Flat Region: The intensity remains constant when moved in any direction.

- Edge: When displaced along the edge, there is minimal change, when displaced across the edge, there is significant change.
- Corner: When moved in any direction, there is great change.

For a small window shift (u, v), the intensity change is given by:

$$E(u,v) = \sum_{x,y} w(x,y)\,[I(x+u,y+v) - I(x,y)]^2 \tag{4.24}$$

Where:

- $I(x,y)$ is the image intensity at position (x,y)
- $w(x,y)$ is a weighting function (typically a Gaussian window and its usually 1)
- The sum is over all pixels in the window

Using Taylor series expansion, we can approximate:

$$I(x+u,y+v) \approx I(x,y) + u\frac{\partial I}{\partial x} + v\frac{\partial I}{\partial y} \tag{4.25}$$

Which leads to:

$$E(u,v) \approx \sum_{x,y} w(x,y)\,[uI_x + vI_y]^2 \tag{4.26}$$

Where I_x and I_y are the partial derivatives of the image.

This can be rewritten in matrix form:

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix} \tag{4.27}$$

Where M is the structure tensor (also called the Harris matrix):

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \tag{4.28}$$

The eigenvalues (λ_1 and λ_2) and eigenvectors ($\mathbf{v}_1$ and $\mathbf{v}_2$) of matrix M provide crucial information about the local structure.The eigenvectors ($\mathbf{v}_1$ and $\mathbf{v}_2$) represent the directions in the image where the intensity changes are most significant in two perpendicular directions respectively while Eigenvalues (λ_1 and λ_2) represents the magnitude of the intensity change in those directions respectively. In the context of corner detection, the behavior of the eigenvalues (λ_1 and λ_2) provides insight into the local image structure. For a flat region, both eigenvalues are small, indicating little to no intensity change ($\lambda_1 \approx 0$ and $\lambda_2 \approx 0$). In the case of an edge, one eigenvalue is much larger than the other, reflecting significant intensity change in one direction and minimal change in the perpendicular direction ($\lambda_1 \gg \lambda_2$ or $\lambda_2 \gg \lambda_1$). Lastly, for a corner, both eigenvalues are large, suggesting that intensity changes are significant in both directions, resulting in a point where two edges meet (λ_1 and λ_2 are both large).

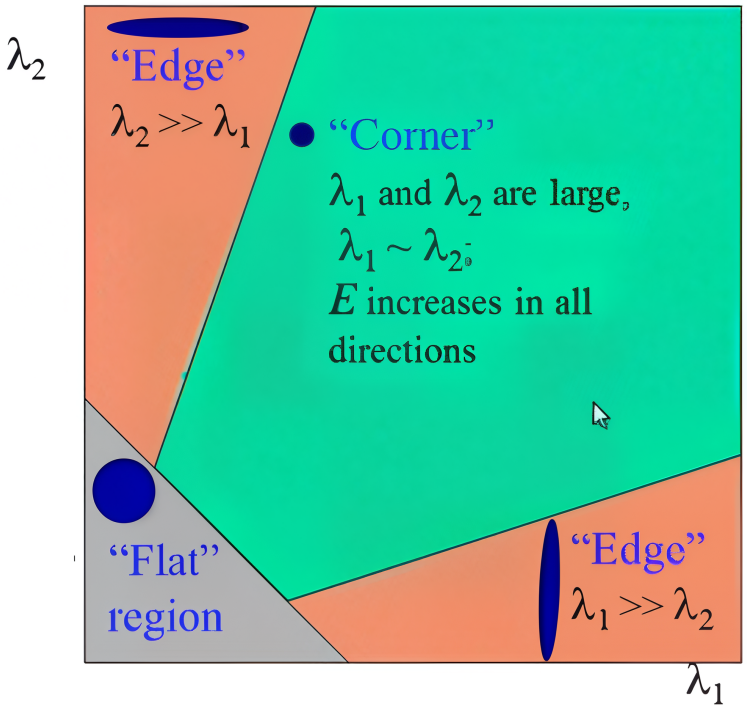

Figure 4.12: Harris Space for Corner Detection

For a corner, intensity changes significantly in both directions, resulting in two large eigenvalues.Computing eigenvalues is computationally expensive, so Harris proposed a

corner response function:

$$R = \det(M) - k \cdot \text{trace}(M)^2 \tag{4.29}$$

Where:

- $\det(M) = \lambda_1 \lambda_2$
- $\text{trace}(M) = \lambda_1 + \lambda_2$
- k is an empirical constant (typically 0.04-0.06)

The classification based on R is:

- $R > 0$: If R is positive, it typically indicates a corner (both eigenvalues are large).When both eigenvalues (λ_1 and λ_2) are large (indicating significant intensity changes in both directions), R will be large, suggesting a corner.
- $R \approx 0$: If R is near zero, it suggests a flat region (both eigenvalues are small, indicating little to no intensity change).
- $R < 0$: If R is negative, it indicates an edge (one eigenvalue is much larger than the other, indicating intensity change in only one direction).

4.12.2 Shi-Thomasi Corner Detection Method:

The Shi-Tomasi corner detection algorithm is a feature detection method used in computer vision to identify distinctive points (corners) in images. Developed by Jianbo Shi and Carlo Tomasi in their 1994 paper *Good Features to Track*, this algorithm improves upon the Harris corner detector by modifying the corner response function to achieve more reliable results.

Unlike the Harris detector which uses a composite score function, Shi-Tomasi uses the minimum eigenvalue of the structure tensor:

$$R = \min(\lambda_1, \lambda_2) \tag{4.30}$$

where λ_1 and λ_2 are the eigenvalues of matrix M.

A pixel point is considered a corner if:

$$R > \text{threshold} \tag{4.31}$$

This approach is based on the observation that a corner is characterized by significant intensity changes in multiple directions. When both eigenvalues are large, the point is likely a corner.

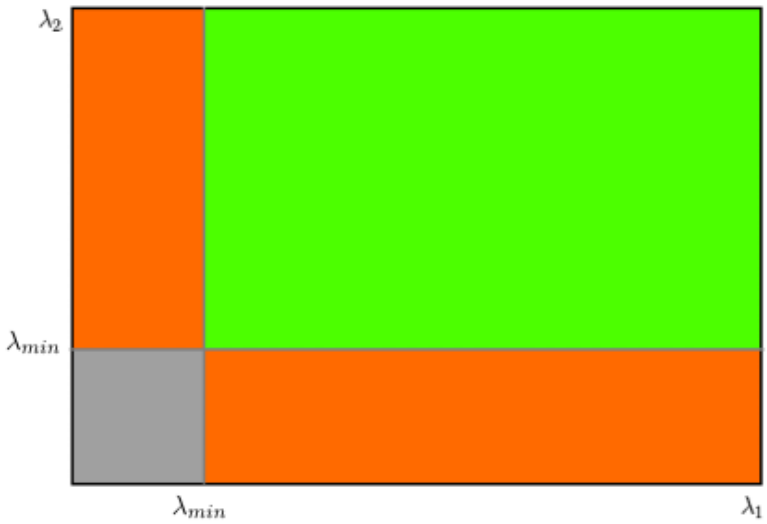

Figure 4.13: Shi-Thomasi space

From the figure, we can see that only when λ_1 and λ_2 are above λ_min a minimum value,it is considered as a corner(green region).

4.13 Camera movement Estimation:

In our project,the frames are not static ,the movement of camera makes extra changes in the movement prediction of objects.So we need to estimate the movement of camera in both x and y directions and subtract it from our predicted movement.The corners which are features collected from Shi-Thomasi detection method ,we will be using that corners and use lucas kanade optical flow algorithm to calculate the camera movement. Camera movement estimation using background reference relies on the principle that a stationary background should remain unchanged unless the camera itself moves. This method is based on feature tracking, where a key point in the background is identified

and monitored across frames. If the camera moves, the background features appear to shift due to the change in perspective. By measuring the displacement of these points, the direction and magnitude of camera movement can be determined. This approach assumes that the background is relatively stable, minimizing interference from moving objects. OpenCV (cv2) was used to detect and track these key points.

4.13.1 Lucas Kanade Optical Flow:

Lucas-Kanade is a foundation algorithm in computer vision that has been used to estimate optical flow: the movement of objects, surfaces, and lines between two images of a sequence of a video. Developed by Bruce D. Lucas and Takeo Kanade in 1981, the technique provides a way through which pixels in an image move from one image to another. Optical flow is the apparent motion of objects, surfaces, and edges in an image caused by relative motion between the scene and observer. The Lucas-Kanade algorithm approximates this flow by locally solving the fundamental optical flow equations in a least-squares sense around each pixel.

The Lucas-Kanade method relies on several important assumptions:

1. **Brightness Constancy**: The intensity or brightness of a particular point in the image remains constant between frames despite its position changing.

$$I(x,y,t) = I(x+u, y+v, t+\Delta t) \tag{4.32}$$

2. **Small Motion**: The displacement of image content between consecutive frames is relatively small.

3. **Spatial Coherence**: All pixels within a small neighborhood move together with approximately the same velocity.

4. **Temporal Persistence**: The motion of a surface patch changes slowly over time.

The brightness constancy assumption leads to the optical flow constraint equation. Using the Taylor series expansion and neglecting higher-order terms:

$$I(x+u, y+v, t+\Delta t) \approx I(x,y,t) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t}\Delta t \tag{4.33}$$

From the brightness constancy assumption and setting $\Delta t = 1$:

$$\frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t} = 0 \tag{4.34}$$

more concisely:

$$I_x u + I_y v + I_t = 0 \tag{4.35}$$

Where:

- $I_x = \frac{\partial I}{\partial x}$ (spatial derivative in the x-direction)
- $I_y = \frac{\partial I}{\partial y}$ (spatial derivative in the y-direction)
- $I_t = \frac{\partial I}{\partial t}$ (temporal derivative)
- u (horizontal optical flow component)
- v (vertical optical flow component)

For a single pixel, we have one equation with two unknowns (u and v), creating what's known as the aperture problem. This makes it impossible to determine a unique solution for the flow at a single point. This is why the Lucas-Kanade method uses a small neighborhood of pixels.

For a window of pixels, we obtain a system of equations:

$$A \cdot \begin{bmatrix} u \\ v \end{bmatrix} = b \tag{4.36}$$

Where:

- A is an $N \times 2$ matrix containing the spatial derivatives for each pixel in the window:

$$A = \begin{bmatrix} I_x(x_1, y_1) & I_y(x_1, y_1) \\ I_x(x_2, y_2) & I_y(x_2, y_2) \\ \vdots & \vdots \\ I_x(x_N, y_N) & I_y(x_N, y_N) \end{bmatrix} \tag{4.37}$$

- b is an $N \times 1$ vector containing the negated temporal derivatives:

$$b = - \begin{bmatrix} I_t(x_1, y_1) \\ I_t(x_2, y_2) \\ \vdots \\ I_t(x_N, y_N) \end{bmatrix} \tag{4.38}$$

The least-squares solution is:

$$\begin{bmatrix} u \\ v \end{bmatrix} = (A^T A)^{-1} A^T b \tag{4.39}$$

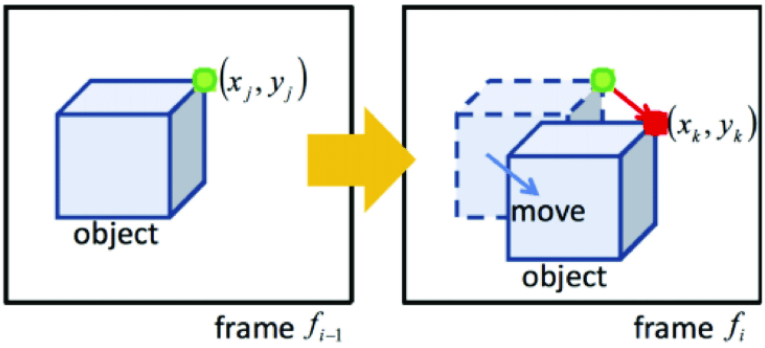

Figure 4.14: Movement estimation

Now after finding u and v i.e the movement of pixels of that neighborhood in x and y directions, we can subtract it to get the actual movement of objects located in those neighborhood. The final camera movement is represented as cameraMovementX and cameraMovementY, which indicate the horizontal and vertical displacement of the camera, respectively. These values are derived by tracking a reference point in the background across frames.

If a key point (x_0, y_0) is detected in the initial frame and its new position in a later frame

is (x_t, y_t), the camera movement in the X and Y directions is calculated as:

$$\text{cameraMovementX} = x_t - x_0 \tag{4.40}$$

$$\text{cameraMovementY} = y_t - y_0 \tag{4.41}$$

4.14 Perspective Transformation

For calculation of speed and distance,the pixel coordinate system is not accurate, we need to convert pixel coordinate system to real world coordinate system so that the distance and speed can be accurately measured.A rectangle in real world might seem as trapezoid shape due to issue in different orientation of view plane of the camera.In our project, perspective transformation was used to map the football field from the camera's view (which could be distorted due to camera angle) to a top-down view. This transformation ensures that the positions of players and the ball are represented accurately on a uniform plane. The most common technique used for this transformation is homography.

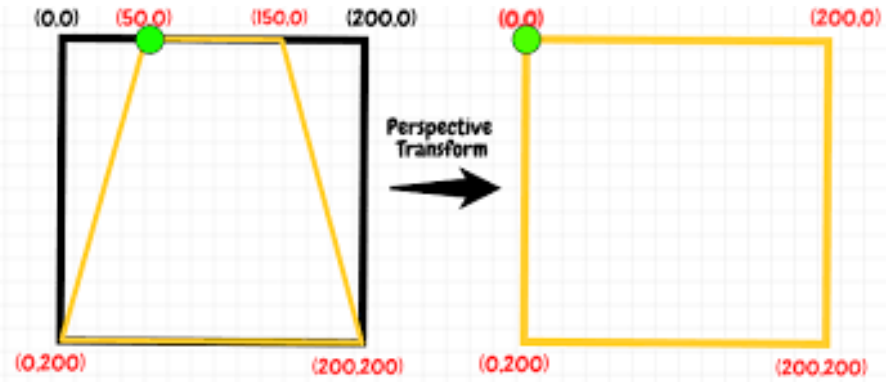

Figure 4.15: Perspective Transformation

4.14.1 Homography

Homography is a mathematical transformation that maps points from one plane to another. It is commonly used in computer vision tasks when we need to transform an image from one perspective to another, like turning an angled view into a top-down view.

The homography transformation can be represented using the following matrix equa-

tion:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \mathbf{H} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{4.42}$$

Where:

- (x, y) are the coordinates of a point in the original image (perspective view),
- (x', y') are the coordinates of the transformed point in the top-down view,
- H is the 3×3 homography matrix that encodes the perspective transformation.

4.15 Ball tracking and Active Player Identification

The ball is tracked separately to ensure consistent identification across frames. Since the ball is small relatively to othe players and also it moves rapidly and frequently changing its direction, tracking it accurately is very challenging. We utilize object detection and tracking techniques to locate the ball in each frame and interpolate missing positions if necessary to maintain smooth tracking.To determine the active player, the system calculates the shortest distance between the ball and all detected player's left and right foot. The player closest to the ball is identified as the active participant, which helps analyze player involvement in different phases of the game.Also we have created a threshold distance to handle long pass cases ,where calculating shortest distance may not be accurate.So in such case, the last player having the ball will be assigned as the active player. This approach ensures accurate tracking of player-ball interactions throughout the match.

4.15.1 Linear interpolation

Interpolation is a mathematical method used to estimate unknown values between two known values. In the context of ball tracking, interpolation helps estimate the position of the ball when it is not detected in certain frames, such as when it is occluded, blurred, or moving too quickly. This ensures a consistent and smooth tracking of the ball across frames, even when the ball is temporarily invisible to the tracking system.By using interpolation, the algorithm can predict the ball's location based on its position in previous and future frames. This way, the tracking system maintains continuity in the

ball's movement, reducing gaps and inaccuracies in the trajectory.In our project, linear interpolation is specifically used to estimate the missing positions.We may visualize it as by drawing a straight line between the last detected position and the next detected position of the ball, assuming constant motion between these frames.

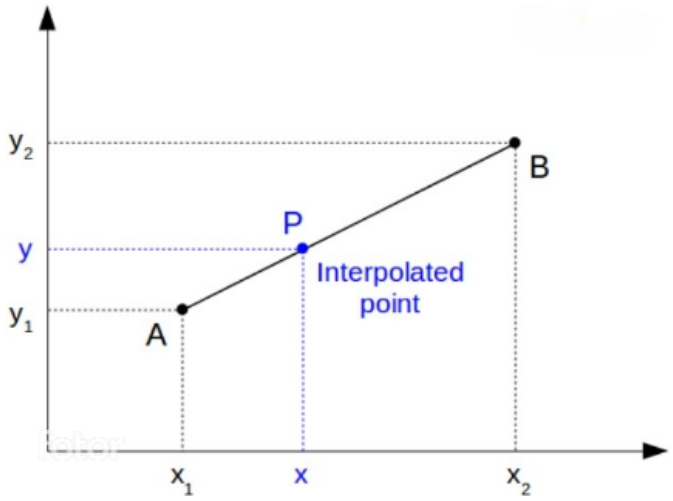

Figure 4.16: Linear interpolation

Formula for interpolation:

$$P_t = P_{t-1} + \frac{(P_{t+1} - P_{t-1})}{P_{t+1} - P_{t-1}} \times (P_t - P_{t-1}) \tag{4.43}$$

Where:

- P_t is the estimated position of the ball at time t,
- P_{t-1} and P_{t+1} are the known positions of the ball in the previous and next detected frames,
- t represents the frame index.

4.16 Speed and Distance Covered Calculation

The speed and distance covered by players are computed based on their movement across consecutive frames in the video. This process involves tracking each player's position in every frame, identifying their displacement over time, and using timestamps

to determine their speed. To achieve this, a combination of object detection, tracking algorithms, and mathematical calculations is applied to ensure accurate measurements.

The tracking system continuously updates the positions of detected players using the Byte Track tracking algorithm.Also the positions are adjusted from camera movement and view transformation,so that calculations is reliable. By analyzing the movement of each player across frames, the displacement is calculated using the Euclidean distance formula, which determines the straight-line distance between two points in a 2D plane.

4.16.1 Euclidean distance

Euclidean Distance is a measure of the straight-line distance between two points in a space, commonly used in many fields like machine learning, physics, and geometry. It is based on the Pythagorean theorem and calculates the shortest distance between two points.

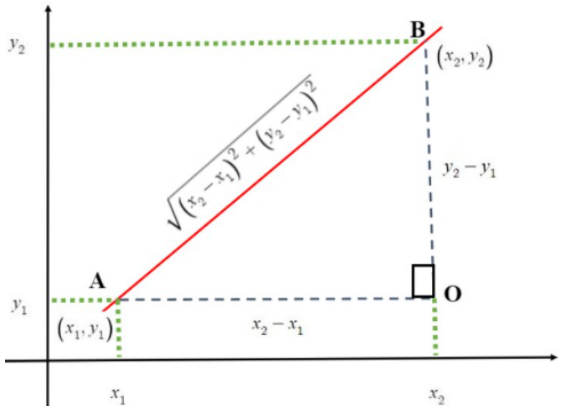

Figure 4.17: Euclidean distance

the Euclidean distance formula for shortest distance calculation:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{4.44}$$

Where:

- (x_1, y_1) are the coordinates of the object,
- (x_2, y_2) are the coordinates of another object.

4.16.2 Distance-Time Ratio

Players are detected and tracked across frames using the Byte Track tracking algorithm. The tracker assigns a unique ID to each player and updates their position in every frame. The movement of each player is determined by calculating the displacement between consecutive frames.

- The time elapsed between two frames is determined using the frame rate (FPS) of the video.Formula for time calculation is:

$$time = \frac{1}{FPS} \tag{4.45}$$

  where FPS stands for Frame Per Second

  for eg, If the video runs at 30 FPS, the time per frame is:

$$\text{Time Elapse} = \frac{1}{30}\,\text{sec} \tag{4.46}$$

  This allows precise measurement of how long it takes for a player to move between frames.

- The distance is calculated by using Euclidean distance formula between frames of the videos.

so the formula for the speed is given by:

$$speed = \frac{distance}{\text{time elapse}} \tag{4.47}$$

4.17 Team assignment

In this project, Team Assignment refers to the process of grouping players detected during a football match into two distinct teams. This classification is necessary for subsequent analysis, such as team ball posession, active players and understanding team dynamics. After detecting the players' positions through YOLO, K-Means Clustering is used to assign each player to one of the two teams based on their spatial distribution on the field.

4.17.1 K-mean clustering

K-Means clustering is an unsupervised machine learning algorithm used to group similar data points into clusters based on their features. The algorithm iteratively assigns each data point to the nearest cluster centroid and updates these centroids until convergence. This process continues until there is minimal movement of centroids, ensuring stable and meaningful clusters. The primary goal of K-Means is to minimize intra-cluster variance while maximizing inter-cluster differences, leading to effective classification of data points into distinct groups.

In our project, K-Means clustering is used in two key areas: distinguishing players from the background and assigning players to their respective teams. The first step in analyzing a soccer match is to isolate players from the field, ensuring that only meaningful information is processed. To achieve this, we extract the top half of each player's bounding box, as this region primarily contains the player's jersey, which is essential for team classification. The extracted pixel data are then reshaped into a 2D array of RGB values, where each row represents the color information of a pixel. Applying K-Means clustering with K=2, we separate pixels into two distinct groups: one corresponding to the player's jersey and the other representing the background. To determine which cluster represents the background, we analyze the corner pixels of the bounding box, as these are more likely to belong to the field or surrounding environment. The remaining cluster is then identified as the player's jersey, which is later used for team classification.

After successfully distinguishing the player from the background, we proceed with the team classification. Since soccer players wear distinct uniforms, we again use K-Means clustering (K=2) to differentiate between two teams based on the color of their jerseys. By extracting the dominant color from each player's detected cluster, we compare it with the centroids of the two team clusters obtained from the initial training phase. Each player is then assigned to a team based on the closest match between their jersey color and the pre-determined team color centroids. This approach ensures robust and automated team assignment, improving the accuracy of our match analysis.

By integrating K-Means clustering into our project pipeline, we enhance the reliability

of player tracking, ball possession analysis, and movement visualization. The ability to dynamically separate players from the background and classify teams with minimal supervision significantly improves the efficiency of automated sports analytics.

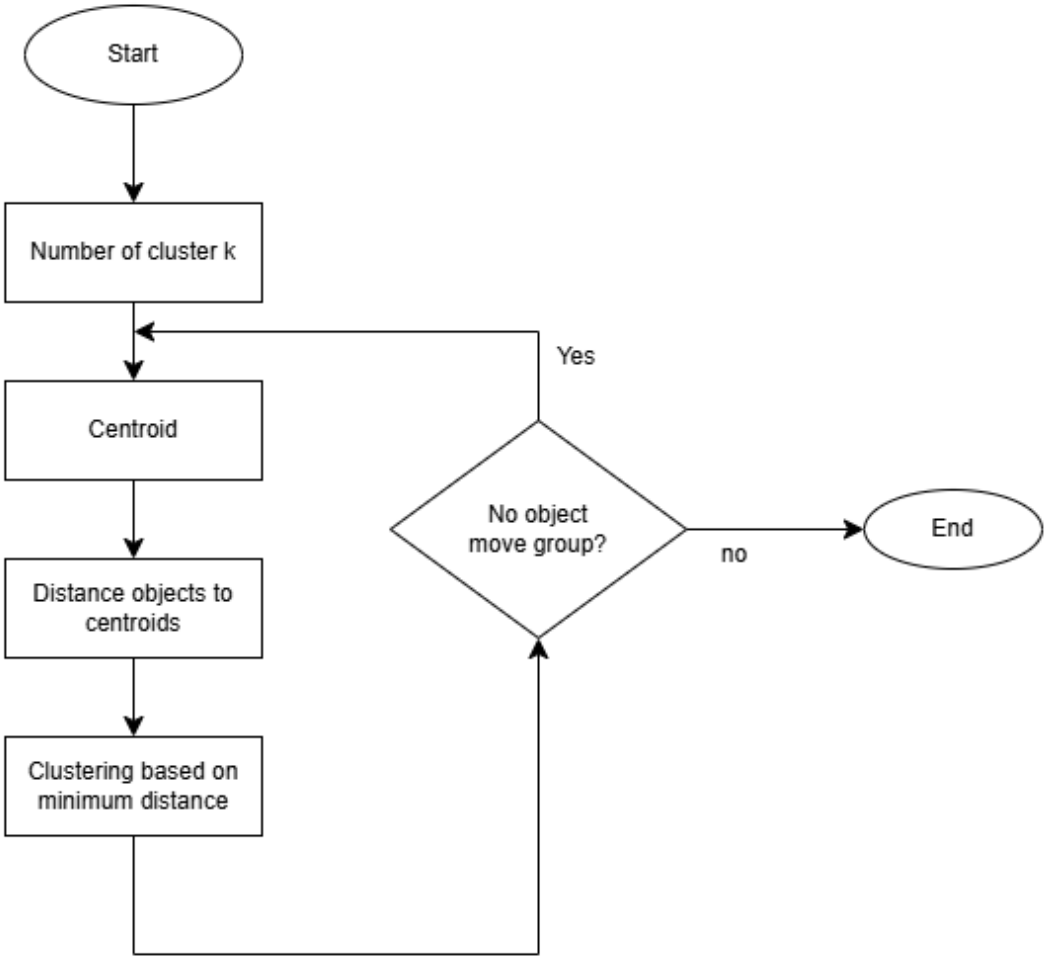

Figure 4.18: k-mean Clustering flowchart

formula of k-mean clustering is:

$$J = \sum_{i=1}^{k} \sum_{j=1}^{n} \left| x_i - \mu_j \right|^2 \tag{4.48}$$

where,

- J is the cost function,
- k is the number of clusters,

- n is the number of data points,
- $x_j(i)$ is a data point in cluster i,
- μ_i is the centroid of cluster i.

4.18 Performance Metrics

4.18.1 Mean Average Precision

The performance of the object detection and localization algorithm is evaluated using a metric called Average Precision (AP) and mean Average Precision (mAP). Average Precision is not the average of precision across different classes. Average precision is calculated from Precision-Recall Curve using technique such as 11 Point Interpolation. Precision-Recall Curve requires calculation of precision and recall which takes True Positive, False Positive and False Negative. Furthermore, the TP or FP for any class is calculated using IoU calculation. To understand Average Precision, IoU, TP, FP, FN, PR-Curve needs to be known . The block diagram for the calculation of mAP score is presented below:

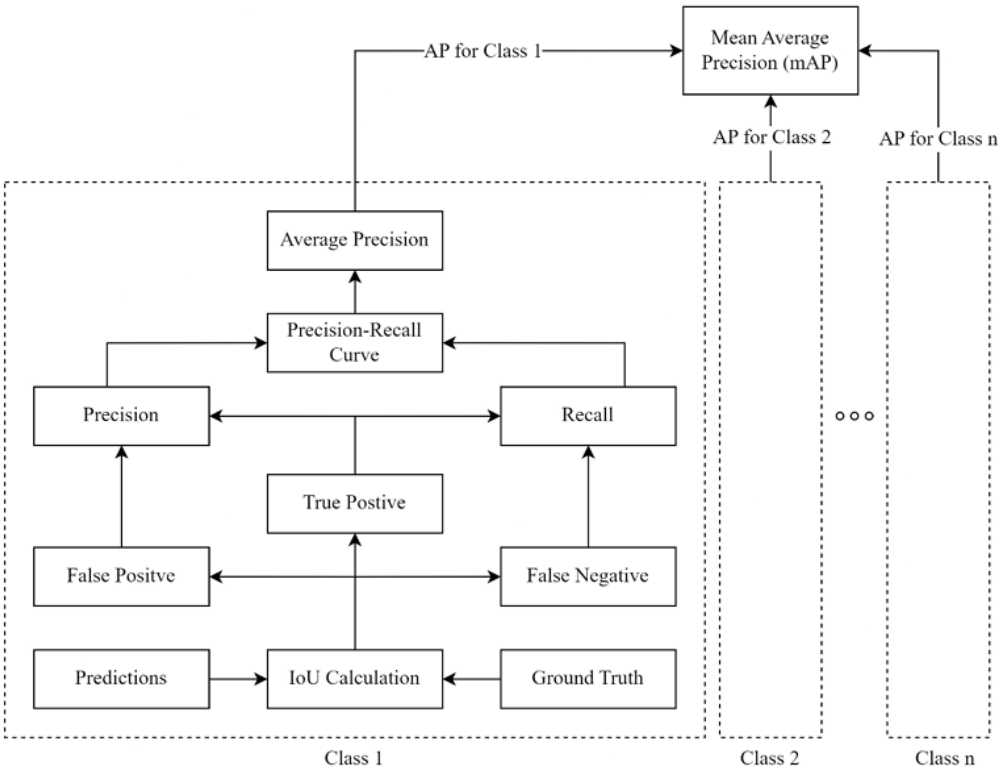

Figure 4.19: Block diagram for Mean Average Precision (mAP) calculation

a. **Intersection over Union (IoU)**

Intersection over Union (IoU) is a metric used to evaluate the accuracy of object detection models by measuring the overlap between two bounding boxes: the predicted bounding box and the ground truth bounding box.

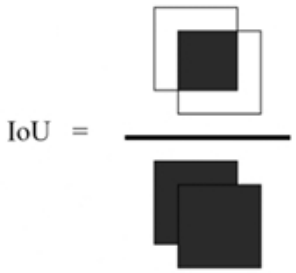

Figure 4.20: IoU Calculation Illustration

Let (x_1, y_1) be the upper-left and (a_1, b_1) be the bottom-right coordinates of the first bounding box. Similarly, let (x_2, y_2) be the upper-left and (a_2, b_2) be the bottom-right coordinates of the second bounding box. The coordinates of the overlapping box are denoted by (xx, yy) as the upper-left and (aa, bb) as the bottom-right.

$$\text{Area of Box 1 } (ab_1) = (a_1 - x_1) \times (b_1 - y_1)$$

$$\text{Area of Box 2 } (ab_2) = (a_2 - x_2) \times (b_2 - y_2)$$

$$\text{Overlap Box Coordinates: } (xx, yy) \text{ and } (aa, bb)$$

$$xx = \max(x_1, x_2), \quad yy = \max(y_1, y_2)$$

$$aa = \min(a_1, a_2), \quad bb = \min(b_1, b_2)$$

$$\text{Intersection Area} = \max(0, aa - xx) \times \max(0, bb - yy)$$

$$\text{Union Area} = ab_1 + ab_2 - \text{Intersection Area}$$

$$\text{IoU} = \frac{\text{Area of Intersection}}{\text{Area of Union}} \tag{4.49}$$

The flowchart of IoU calculation is shown below:

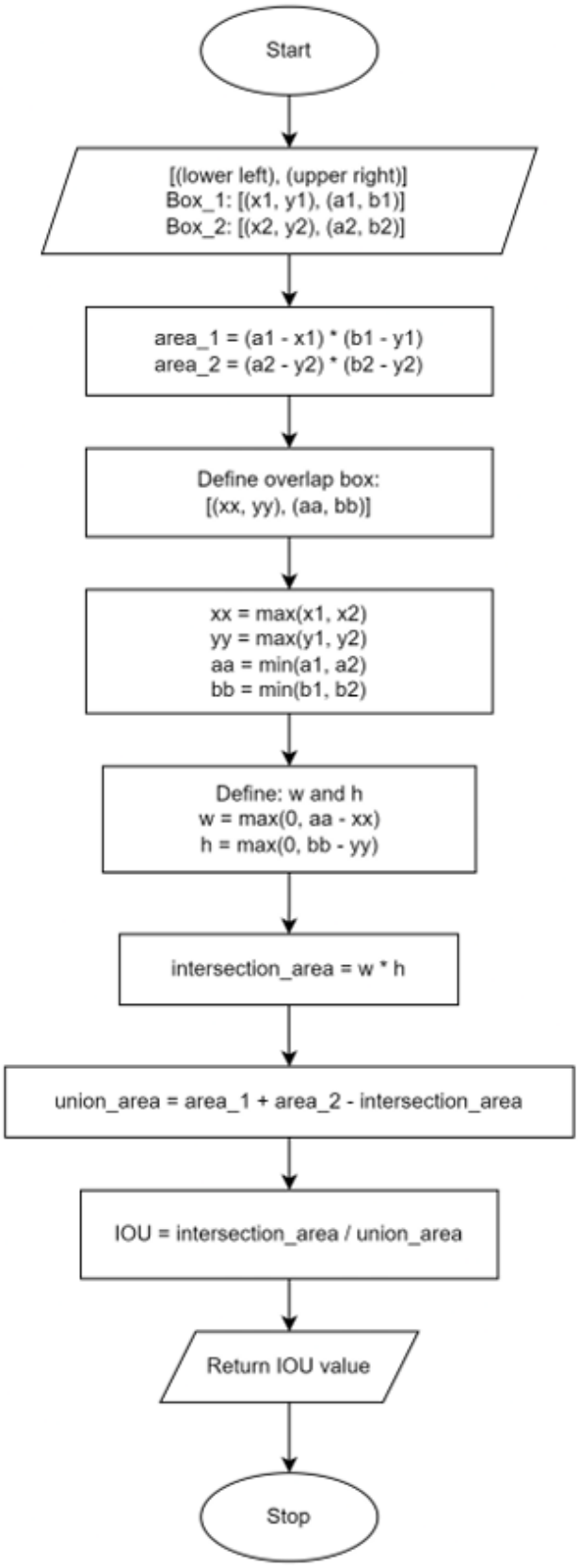

Figure 4.21: Flowchart of IoU calculation

b. **True Positive, False Positive, False Negative**

For a prediction to be considered correct, the class label of both the predicted bounding box and the ground truth bounding box must be identical, and their Intersection over Union (IoU) should exceed a threshold value. The following three metrics are computed based on the IoU threshold and the class labels of the ground truth and predicted bounding boxes:

- **True Positive (TP)**: The model predicted that a bounding box exists at a certain position (positive) and it was correct (true).
- **False Positive (FP)**: The model predicted that a bounding box exists at a particular position (positive) but it was wrong (false).
- **False Negative (FN)**: The model didn't predict a bounding box at a certain position (negative) and it was wrong (false), i.e., a ground truth bounding box existed at that position.
- **True Negative (TN)**: The model didn't predict a bounding box (negative) and it was correct (true). This corresponds to the background, the area without bounding boxes, and is not used to calculate the final metrics.

Precision: Precision is the fraction of correct predictions out of all the predictions the model has made for a given class.

$$\text{Precision} = \frac{TP}{TP+FP} \tag{4.50}$$

Recall: Recall is the fraction of examples of a given class in the data that the model has found.

$$\text{Recall} = \frac{TP}{TP+FN} \tag{4.51}$$

The goal is to attain high precision and recall simultaneously. The precision and recall values are determined by the number of true positives detected by the model. Assigning bounding boxes as TP, FP, and FN is contingent on two factors: the predicted label compared to the ground truth label and the IoU between

the two boxes. To plot precision-recall curve, detection-wise Precision and Recall values needs to be calculated. First, the table of TP, FP, and FN is sorted in descending order of confidence score. The cumulative TP and cumulated FP i.e. keeping on adding the current value with the previous row, is calculated in the table. Then row-wise Precision and Recall is calculated.

$$\text{precision} = \frac{\text{Cumulative } TP}{\text{Cumulative } TP + \text{Cumulative } FP} \tag{4.52}$$

$$\text{recall} = \frac{\text{Cumulative } TP}{\text{Cumulative } TP + \text{Total Ground Truths}} \tag{4.53}$$

Now, the plot of Recall vs Precision gives the PR-Curve.

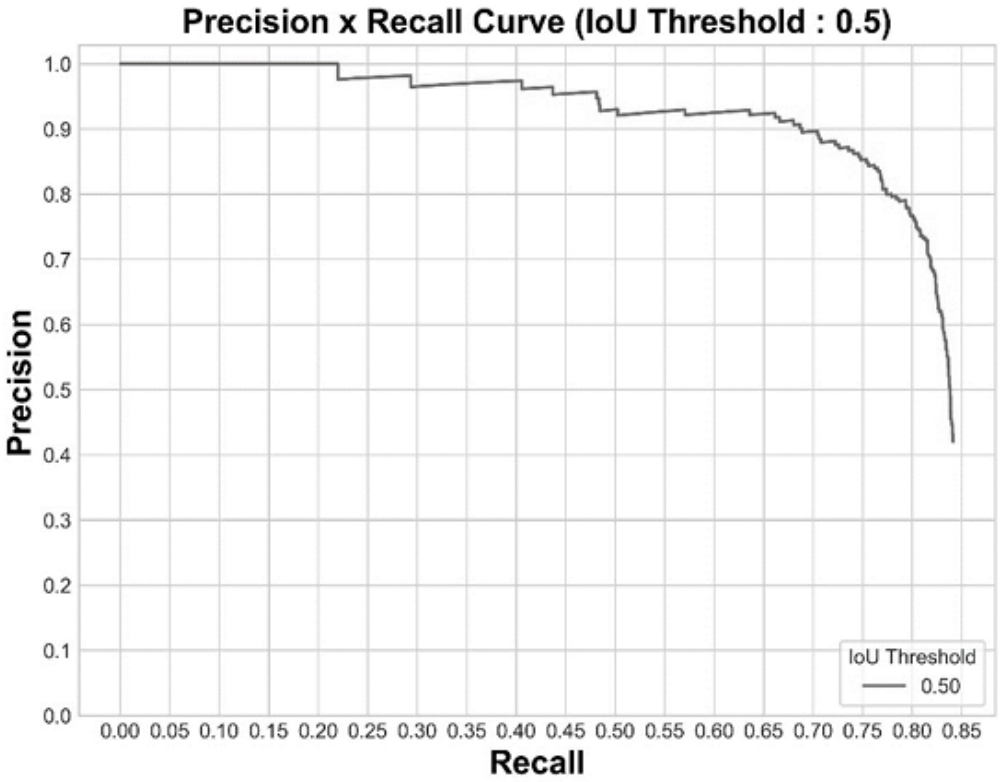

Figure 4.22: Sample of PR curve

c. **Average Precision**

To calculate Average Precision (AP) from the PR-Curve, the 11-point interpolation method, introduced in the 2007 PASCAL VOC challenge, is used. The

Precision values are recorded across 11 equally spaced Recall values, i.e., 0.0, 0.1, 0.2, 0.3... 1.0. Average Precision is defined as:

$$AP = \frac{1}{11} \sum_{i=0.0}^{1.0} \text{Precision at Recall}_i \tag{4.54}$$

The Average Precision is calculated for all classes individually. At last, the average of the AP for all classes gives the Mean Average Precision or mAP:

$$mAP = \frac{1}{n} \sum_{i=0}^{n} AP_i, \quad n \text{ is the total number of classes} \tag{4.55}$$

4.18.2 F1 Score

In the context of multiclass object detection, the F1 score is a popular metric that combines both precision and recall to provide a single measure of the model's accuracy. The F1 score is calculated separately for each class and then averaged across all classes to give an overall score for the model.

The F1 score is calculated as follows:

$$F1 = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4.56}$$

The F1 score for each class provides a single measure of the model's accuracy in detecting objects of that class, by balancing the trade-off between precision and recall. A high F1 score indicates that the model is achieving both high precision and high recall for that class, while a low F1 score indicates that the model is not accurately detecting objects of that class.

4.18.3 Confusion Matrix

A confusion matrix is a tool used to evaluate the performance of a machine learning classification model. It is a table that contains information about the predicted and actual class labels of the data. The confusion matrix is typically used when the output of a model can be one of two or more classes. It a table with 4 different combinations of predicted and actual values.

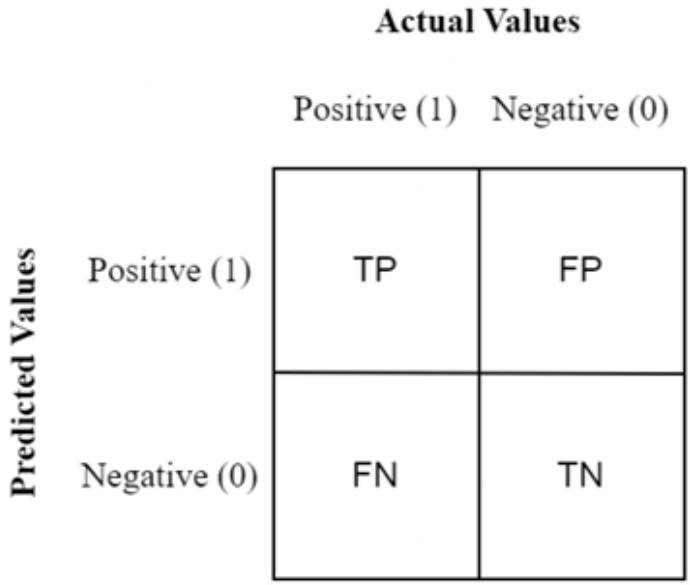

Figure 4.23: Confusion Matrix for Binary Classification

Confusion matrix for multi-class classification is constructed taking all the classes in the model into a 2D matrix, initially containing 0 for all cells. Each row of the matrix represents prediction classes while each column account for the actual classes. For each predicted class or label, the corresponding class occurrence is increased only if the actual class coincide with the predicted class, else the corresponding wrong class is increased. Confusion matrix gives a measure of how well the classes are being correctly classified and for which classes the model is being confused with other classes. The confusion matrix is then normalized with the highest number of truly predicted class count. The visual 2D diagram for confusion matrix is drawn using seaborn heatmap module.

5 IMPLEMENTATION DETAILS

5.1 Dataset Preparation:

5.1.1 Dataset Collection:

The dataset was collected from Roboflow, we also added our own custom data. This way, our data contains a variety of football-related images that were well labeled with annotations for objects of interest (players, referees, ball). We had a total of 620 well-labeled images for training the model.

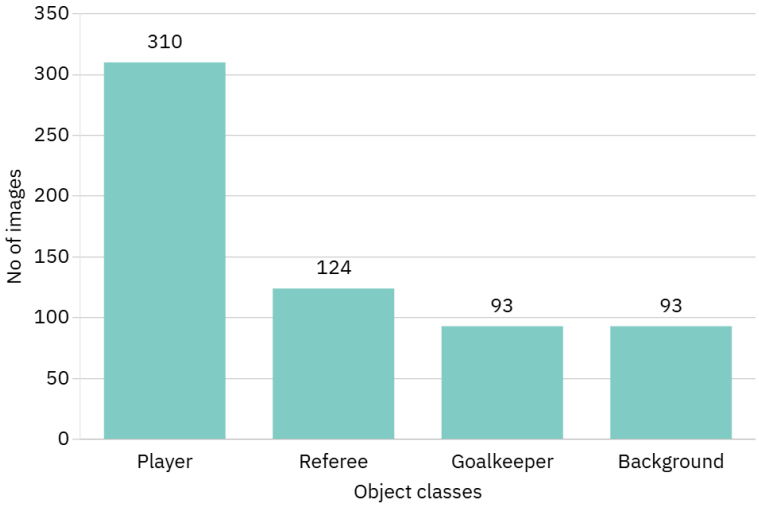

Figure 5.1: Bar Plot of our Dataset

5.1.2 Data Pre-processing:

The images and video frames were resized and standardized to a uniform resolution to 416x416 pixel to ensure consistency during training.

5.1.3 Data Labelling:

These images or frames were labeled with bounding boxes around the players, referees, and the ball to help train our YOLO model.We used Roboflow website feature to label ground truth boxes of every class object.

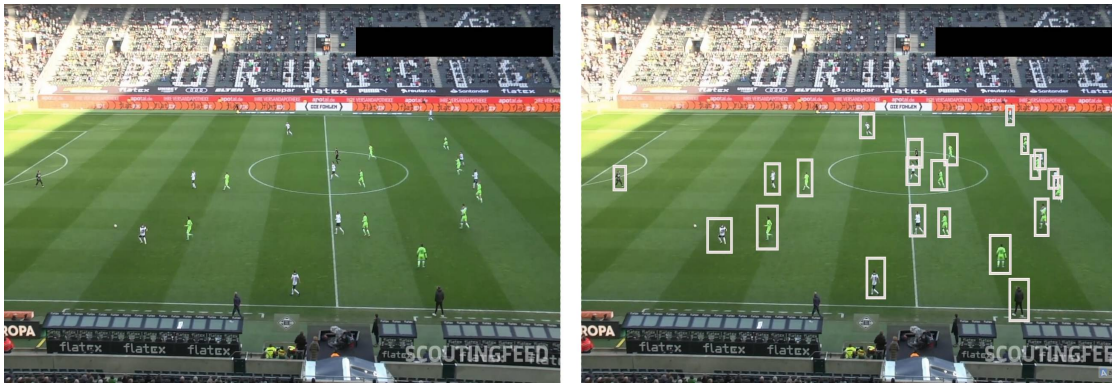

Figure 5.2: Original image(left) Annotated image (right)

5.1.4 Data Augmentation:

We performed augmentation by applying image transformations like rotations, flipping, cropping, and adjusting brightness to enhance our dataset so that the detections accuracy increase for all possible real life cases. The augmented images were processed and ready for model training to improve generalization and reduce overfitting.

Random Brightness and Contrast Adjustment

The brightness of images was randomly adjusted using tf.image.adjustBrightness(image, brightnessValue) with factors from {-0.2, -0.1, 0.0, 0.2, 0.3, 0.4}. Contrast was modified using *tf.image.adjustContrast(image, contrastValue)* with factors between 0.85 and 1.35 of the mean value. Since these are photometric augmentations, bounding box coordinates remained unchanged.

Figure 5.3: Original image (left) and Augmented image (right)

Random Brightness and Contrast Adjustment

The image's pixel values were randomly adjusted: hue was modified using tf.image.randomHue (image, 0.4) with a factor between -0.4 and +0.4, and saturation was adjusted using *tf.image.randomSaturation(image, 0.3, 0.6)* with a factor between 0.3 and 0.6. As photometric augmentation techniques, no changes to bounding box coordinates were needed.

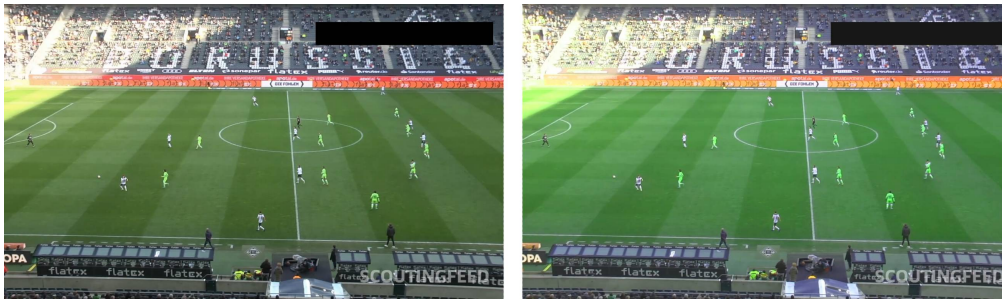

Figure 5.4: Original image (left) and Augmented image (right)

Gaussian Noise Addition

Gaussian noise with a random standard deviation between 0.06 and 0.09 was added to the images. The noise was generated using *tf.random.normal(shape=tf.shape(image), mean=0.0, stddev=18/255, dtype=tf.float32)* and then added to the image. Since adding noise is a photometric augmentation technique, no changes to bounding box coordinates were needed.

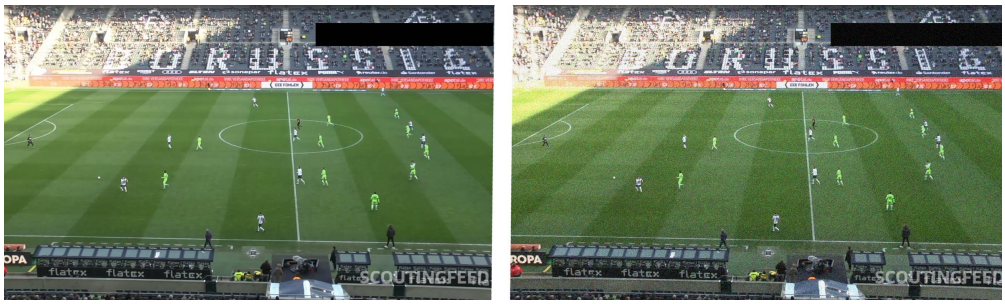

Figure 5.5: Original image (left) and Augmented image (right)

Gaussian Blur Addition

Gaussian blur was added to the images using the *cv2.GaussianBlur(image, kernel, cv2.BORDERDEFAULT)* function from the OpenCV library. A random kernel size was selected from the values: [(3, 3), (5, 5), (1, 5), (5, 1)]. Since adding blur is a photometric

augmentation technique, no changes to bounding box coordinates were needed.

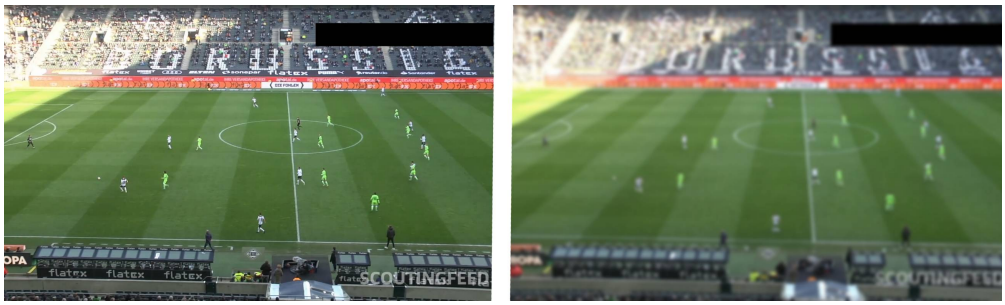

Figure 5.6: Original image (left) and Augmented image (right)

Rotation and translation

Random translation of 40px was applied by selecting a translation vector from values like [[40, 0], [0, 40], [-40, 0], [0, -40], [40, 40], [-40, -40], [-40, 40], [40, -40]] using tfa.image.translate(image, dxDy, ”bilinear”, ”constant”). The remaining pixels were filled with black. Since it’s geometric translation, bounding box coordinates were also adjusted. Additionally, images were randomly rotated anti-clockwise between -8 to -16 and 8 to 16 degrees using tfa.image.rotate(image, angle, ”bilinear”, ”nearest”), and bounding box coordinates were adjusted accordingly using geometric rotation.

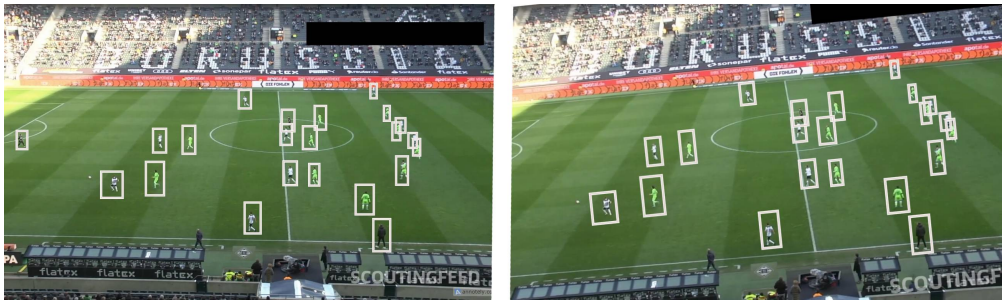

Figure 5.7: Original image (left) and Augmented image (right)

5.2 YOLO Model Training

Model Selection: YOLO (You Only Look Once) v5 model was chosen due to its performance and accuracy in detecting objects.

Data Preparation: The labeled dataset from Roboflow was split into training and validation sets, ensuring that the model could be tested on unseen data.

Output: The trained YOLO model outputs bounding boxes for detected objects (players, referees, ball) in each frame of a video.

5.3 Object Detection and Bounding Box Conversion

Now that we have a working model for the prediction , we use this model to predict the objects in the frame. The input video is broken into frames , and each individual frame is fed to YOLO model. 20 batches of images is fed to YOLO model at a time to ensure no memory overflow issue occurs.

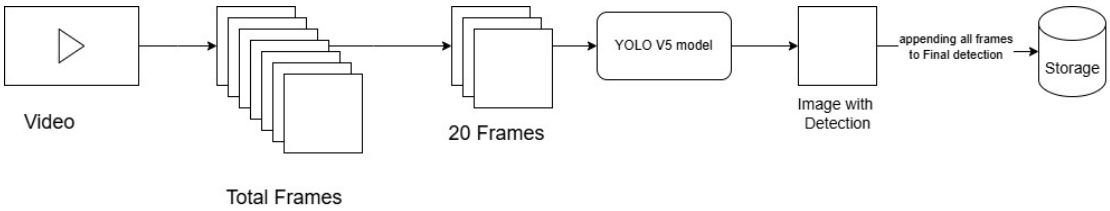

Figure 5.8: Video to Detection

Now that we have detected all the objects in the video and stored the detection information in a dictionary, the next step is to perform tracking. ByteTrack can only track objects in detections that are in the Supervision format. This format includes essential details such as the bounding box coordinates in the xyxy format, class ID, confidence, and the detection data. Therefore, we convert the detections from the previous step into the Supervision format and then use ByteTrack to perform the tracking. Once the tracking is complete, we obtain the tracker ID (highlighted in red), as shown in the illustration below.

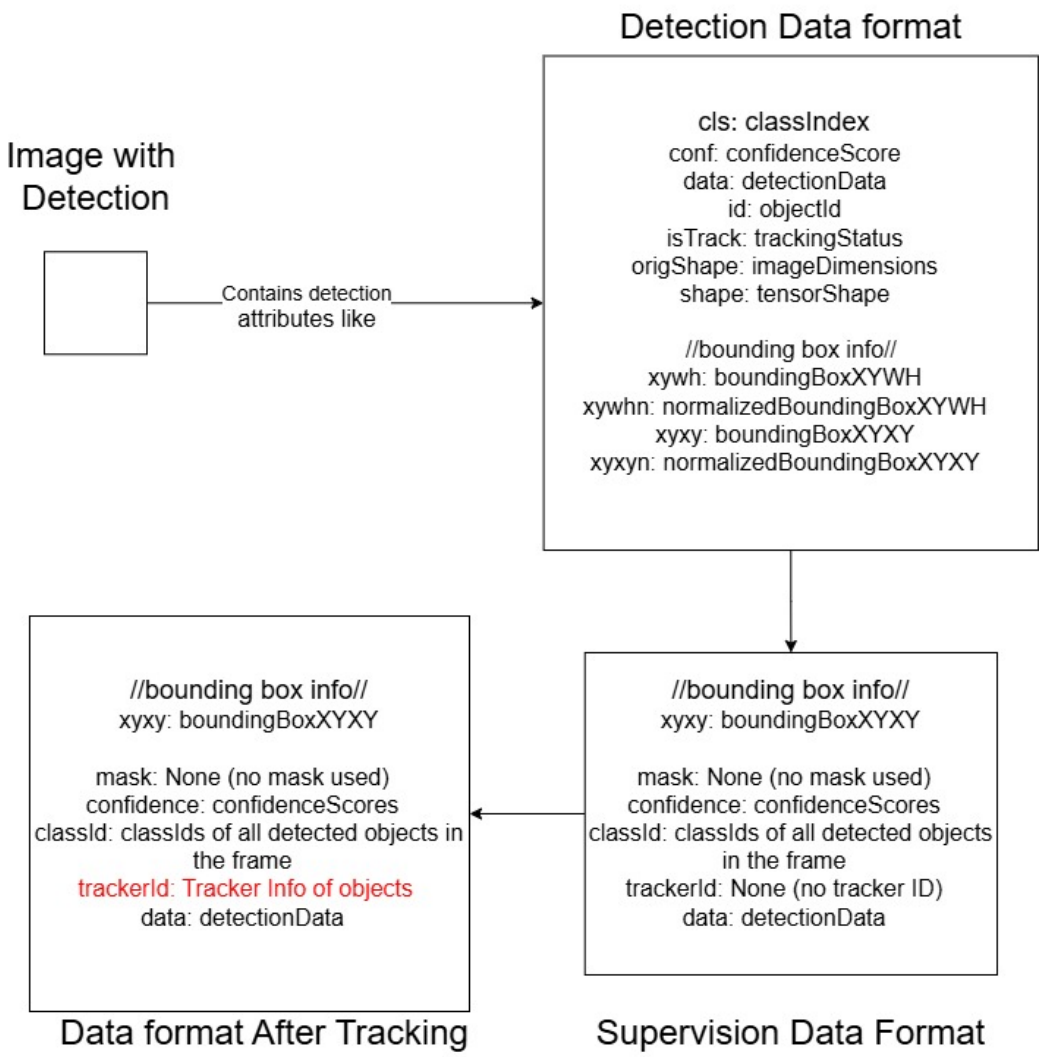

Figure 5.9: data conversion

Now that we have the detections and tracker IDs, we saved this information in a pickle file. The pickle file stores the detection data, including the bounding boxes, confidence scores, class IDs, and the corresponding tracker IDs, which are crucial for tracking object movement across frames. This saved data is used later to perform annotations, allowing us to easily retrieve and visualize the object trajectories or match detections with the tracked objects during subsequent processing. By saving the tracking and detection information in a pickle file, we ensured that we can efficiently access and manipulate the data for future analysis, such as generating annotated videos or reviewing object tracking performance.

5.4 Object Tracking with ByteTrack: Maintaining Identity Across Frames

In the object tracking process, the system uses ByteTrack, a powerful tracking algorithm that combines motion prediction and object association techniques to maintain the identity of detected objects across frames. ByteTrack is initialized as *sv.Bytetrack()*.It receiving the bounding boxes of detected objects, assigning each object a unique tracking ID. This ensures that every object in the scene can be consistently identified throughout the video. As the video progresses, the tracker updates the position of each object by associating detections in subsequent frames, based on the consistency of their movement patterns. Essentially, the system predicts the future positions of objects using their current motion and associates them with the correct identity, ensuring that each object is tracked properly.

ByteTrack is also designed to handle occlusion situations, where objects may temporarily disappear from the view due to overlapping with other objects or moving out of the camera's frame. The tracker uses sophisticated algorithms to re-identify objects when they reappear in the scene, ensuring continuous and accurate tracking. This capability is crucial for scenarios involving multiple objects or complex scenes with frequent overlaps, as it helps maintain the integrity of the tracking process. As a result, ByteTrack is highly effective in keeping track of the objects, even in dynamic and challenging environments, ensuring that the identity and movement of each object are consistently monitored throughout the video.

5.5 Camera Movement Estimation

Since the cameras move (by panning, tilting, or zooming), tracking objects based only on pixel positions would be wrong. This project solves the issue by adjusting the positions of players and the ball according to the camera's movement.

First, we find stable feature points in the first frame using *cv2.goodfeaturesToTrack()*, such as boundary lines or stationary background details, which don't move. We apply a mask to focus on the edges of the frame, avoiding moving players as reference points. These stable points are then tracked in later frames using *cv2.calcOpticalFlowPyrLK()*, which calculates how much they have moved. The camera's movement is estimated by picking the point with the greatest movement as the main indicator of the camera's

motion.

5.6 Camera Movement Analysis and Adjustment in Player Tracking

In our project, analysis of camera movements plays a crucial role in accurately tracking players and the ball, especially when there are dynamic changes in the camera view. The optical flow technique is employed to calculate the pixel shifts between two consecutive frames, allowing the system to understand how the camera has moved. This movement can include actions like panning (horizontal movement), zooming (increasing or decreasing focal length), and tilting (vertical movement). By calculating the optical flow, the system detects the changes in camera position, which is essential for fine-tuning the player and ball tracking to ensure that their positions are adjusted correctly in response to these camera movements.

The adjustment of camera shifts is crucial for maintaining the accuracy of player tracking. When the camera shifts, it could cause the tracked players or ball to appear in slightly different positions on the screen, which would affect the precision of the analysis. By accounting for the optical flow and understanding the exact camera motion, the tracking system ensures that the players remain at a constant spot, even if the camera has shifted. This adjustment process enables the system to effectively handle dynamic camera movements, ensuring that the tracking data remains accurate and that players and the ball are consistently monitored throughout the video, regardless of the camera's motion.

5.7 Perspective Transformation

Perspective transformation is a critical aspect of our project, correctly projecting player and ball paths onto a normalized view of the field. Videos are recorded from various angles of the camera, so things in the world may appear distorted, and straight measurement of speed, distance, and ball possession won't be correct. Our project makes up for this by using perspective transformation to project the perspective view from the camera into a top-down bird's-eye view of the field. The algorithm begins with the creation of a set of four reference points in the input video that delineate a trapezoidal region which is the soccer pitch as seen by the camera viewpoint. The points are manually picked to correspond to the significant positions on the pitch, e.g., the corners of the

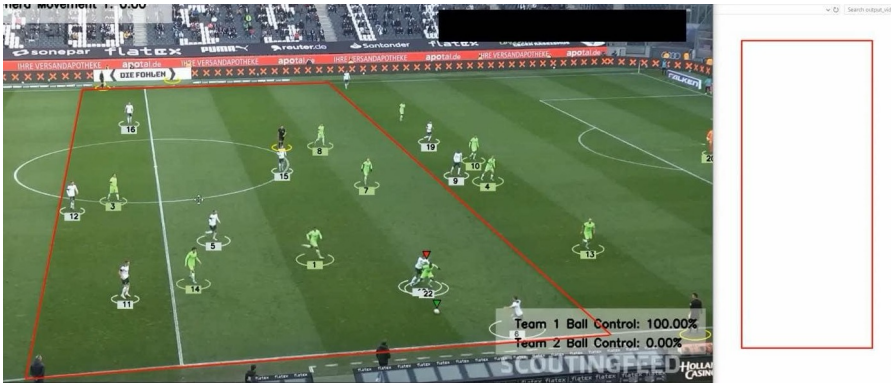

Figure 5.10: Perspective Transformation

penalty area or center circle. These pixel locations are then mapped to a corresponding set of four target points that represent the field as a standard rectangular region of known real-world dimensions (e.g., 68m x 23.32m).

By invoking *cv2.getPerspectiveTransform()*, a transformation matrix is determined by which the location of any point in the original frame can be transformed to its corresponding position in the normalized field view. Anytime a location of an object is detected in a frame, *cv2.perspectiveTransform()* is invoked to transform the detected coordinates into the new view. This ensures tracking data is consistent even in terms of camera angle or movement.

5.8 Ball Position Interpolation

Ball tracking is one of the hardest things to do in our project due to rapid motion, occlusions, and short-term out-of-view losses. To give smooth and continuous tracking, Our project employs interpolation techniques to make predictions of the ball's position in frames in which it does not appear. This guarantees that good data is preserved for possession analysis, speed estimation, and trajectory prediction.

Ball locations are stored in a dictionary data type where the ball's detected location within every frame is stored using its bounding box coordinates. Some frames may not have a detected ball bounding box in certain situations, however, due to the impacts of motion blur or players blocking. Our project compensates for this using interpolation techniques with pandas' interpolate() function in order to insert missing locations according to the surrounding frames.

The process starts by acquiring the bounding box coordinates of the ball for each frame into a structured format using pandas DataFrame. The missing values (frames where the ball is not detected) are imputed using linear interpolation, which estimates positions by assuming uniform motion pattern from previous and next detected positions. If there are missing values at the beginning of the video (i.e., before any detection has been made), the backward fill (bfill()) is employed to give the first detected position to those frames so that there are no undefined values.

After interpolation, the new positions are converted back to dictionary format, maintaining the format for future processing. This way, tracking of the ball is continuous even when YOLO loses track of the ball in certain frames. With interpolation, our project improves movement tracking accuracy, possession calculation, and trajectory estimation, thereby making the system more reliable and robust.

5.9 Player Movement Analysis and Active Player

Player Movement Analysis Player movement plays a key role in football analysis to examine performance metrics, positional tactics, and game strategy as a whole. Our project implements object tracking by ByteTrack that assigns track IDs to players and follows them across frames. Movement of individual players is thus captured end-to-end, even as players move along the field or briefly overlap others.

One of the key aspects of our project is determining the active player, i.e., the player that is in contact with the ball. This is achieved by attributing ownership of the ball to the nearest player such that actions such as passing, dribbling, or shooting can be examined.

Towards this end, the system:

- Computes the position of the ball for every frame.
- Finds the nearest player by calculating distances of all observed players from the ball.
- Uses the distance to attribute possession (i.e., where the ball is 70 pixels away from the feet of a player).

- Takes the attribution of the closest player as possessing the ball and labels them the active player.
- Tracking this active player is extremely helpful for possession statistics, tactics, and player performance analysis. It can be analyzed further, like seeing which player is making the most passes, shots, or losing possession frequently.

With the addition of player movement tracking, ball possession analysis, and active player tracking, our project provides a full soccer analysis system that reveals more about the game.

5.10 Speed and Distance Analysis

Player distance traveled and player speed are significant drivers in soccer analysis used to provide figures on players' performance, endurance, and position based on tactics. In the project, speed and distance are derived from tracking up-the players to get movement captured over time accurately. The calculation is triggered by retrieving each player's foot position from the bounding box as it is the constant point for motion observation. Because camera motion may influence the accuracy of the positions, positions obtained are calibrated by camera motion estimation for consistency across frames.

Perspective deformation in the video is eliminated using a homography transformation with *cv2.getPerspectiveTransform* from OpenCV, transforming the detected positions onto a standardized soccer pitch. This is done to have the movement calculations done in real distances and not displacements in pixels. Once they get transformed positions, the overall distance covered by an individual player is determined by summing their frame-to-frame displacements. In case a player is occluded from view in the frame, interpolation techniques are used to estimate missing positions so that tracking becomes smooth.

Speed is determined using the displacement of players' time. Since the video is recorded in 24 frames per second (FPS), it is assumed that the difference between frames is 1/24 seconds. Player speed is calculated by dividing distance covered between the frames by time taken. An average sliding window is applied to reduce the sudden change that is caused due to minor tracking errors, where velocity is averaged over a series of subse-

quent frames. Calculated speed and overall displacement are then assigned to respective players in Annotation Process in Soccer Analysis Project.

5.11 Team Assignment Using K-Means Clustering

In the team assignment process, feature extraction is initially carried out by analyzing the visual contents within the bounding boxes surrounding each player. The primary focus is on the color of their jerseys and their positions on the field, as these visual features play a key role in distinguishing between the two teams. The jersey color is a particularly effective feature, as it is usually the most prominent visual difference between opposing teams, while player positions provide additional context for identifying the team's area on the field.

Once these features are extracted, the system applies K-Means clustering, a machine learning technique that groups players based on the similarity of their jersey colors. K-Means clustering works by partitioning the players into clusters, where each cluster contains players whose jersey colors are most similar. This process allows the system to identify players from the same team, as those with similar jersey colors are grouped together. After the clustering process, each player is assigned to a specific team based on the cluster they belong to, ensuring that all players with matching jersey colors are correctly identified as part of the same team. This technique is an efficient way to automatically categorize players, making the team assignment process quick and accurate.

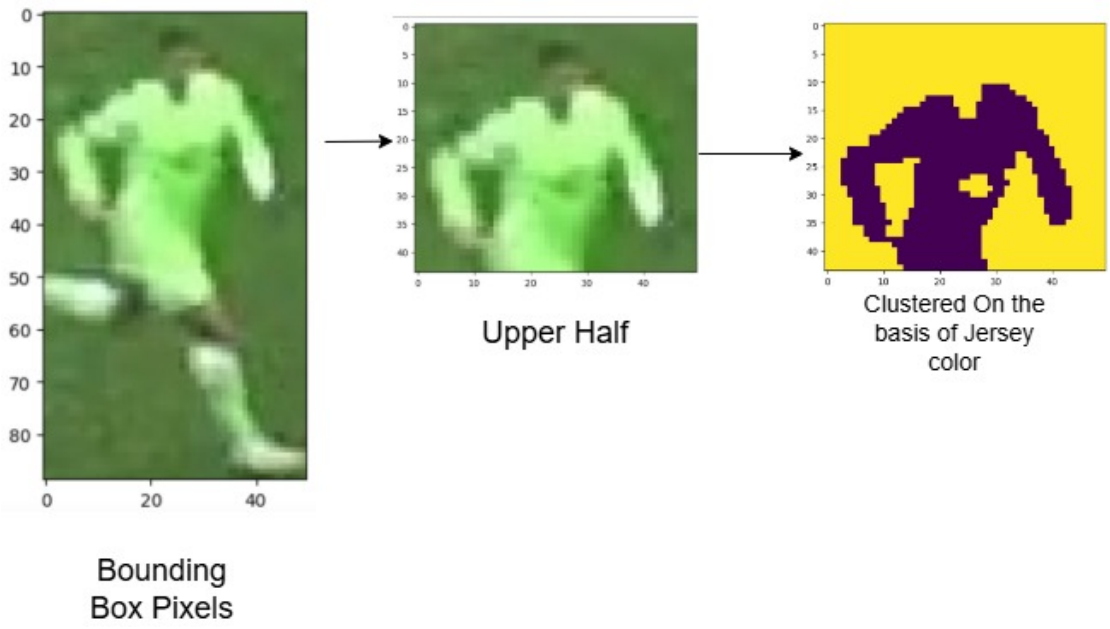

Figure 5.11: k-mean Clustering

5.12 Storing Processed Video Data Using Pickle

After each video is processed, the system captures a variety of crucial data points such as player locations, ball movement, and information about active players at each frame of the video. This data is stored using Python's Pickle module, which is widely used for serializing and deserializing Python objects. Serialization is the process of converting the data structures into a format that can be easily saved into a file and later restored or used in further computations. In this case, the Pickle module is used to store the complex data generated during the video processing, ensuring that it includes all the relevant information, like player positions, ball tracking, and actions of the active players across every frame. The storage of this data provides a valuable resource for future analysis, retrieval, or even reprocessing if needed.

The tracking information saved in the Pickle file is essential for multiple purposes. It can be used for future computational processes, allowing for analysis of the same video at a later time or with a different focus. For example, the same data could be analyzed from various angles, such as to study the patterns of player movement, to evaluate the speed and trajectory of the ball, or to perform statistical analysis on player performance. Additionally, since the Pickle file is a serialized format, it is relatively easy to load and manipulate for visualization or for use in machine learning models. The stored tracking

data can be fed into new computational processes or systems to enhance the functionality of the web application, offering opportunities for further analysis, visualization, or development of insights about the players and the game.

Moreover, the Pickle file is designed to be flexible and convenient for the user, allowing for seamless access to the stored information. It enables you to avoid the need to reprocess the video or run the AI/ML model from scratch each time the data is needed. Instead, by storing the processed tracking information, the system can quickly retrieve the relevant data and use it for further computations or visualizations without having to go through the entire processing pipeline again. This greatly improves efficiency, making it possible to focus on detailed analysis or to generate multiple views of the video data without adding unnecessary computational overhead. Overall, using Pickle for data storage ensures that the tracking information is preserved in a format that can be easily accessed and utilized in future processes, contributing to the overall functionality and performance of the system.

5.13 Annotation

The annotation process begins after all the tracking calculations, such as object detection, team classification, ball possession, speed, and distance calculations, are completed. Using OpenCV drawing functions, different types of annotations are created on the frames before the processed final video is saved. Every one of the annotation components is crafted with care to be legible but without overly cluttering the video. The annotation process in our project is a crucial step in creating a visual representation of the information monitored on the video output. This is achieved through overlaying various graphical aspects such as bounding boxes, player tags, team colors, possession of the ball, speed, distance, and camera movement adjustments to enhance the interpretability of the analysis. The annotations provide real-time visualization of ball and player tracking, thus simplifying analysis of player behavior, team formation, and match dynamics.

The annotation process begins after all the tracking calculations such as object detection, team identification, ball possession, speed, and distance estimation have been performed. Multiple annotations are subsequently applied to the frames via OpenCV

drawing tools prior to saving the resultant processed video. Each annotation part is specifically designed to be comprehensible but not excessively cluttered in the video.

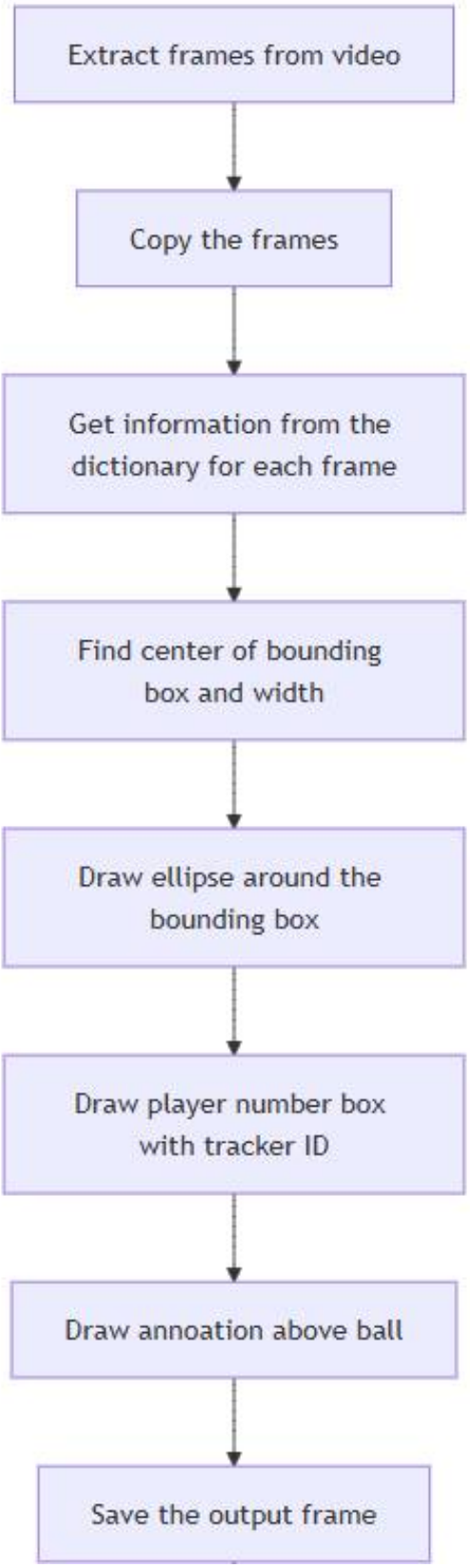

Figure 5.12: Annotation Process

- **Player and Referee Annotations:** Players are labeled with an ellipse marker, drawn at their foot positions. The color of the ellipse is determined based on the team color of the player, which is obtained from the team clustering algorithm. Referees are labeled with a yellow ellipse so that they can be distinguished as individual entities from players. A track ID is also assigned to each player, which is written above their marker using *cv2.putText()*.

- **Ball Annotation and Active Player Indicator:** The ball is traced and marked with a green triangle so that it can be readily recognized from players. Also, if a player possesses the ball (by proximity-based assignment), a red triangle is drawn over that player's marker, signifying that he is the present player at that moment.

- **Speed and Distance Display:** Each player's calculated speed (in km/h) and travelled distance (in meters) are marked on the video by *cv2.putText()*. The text is left marginally below the player marker to not obscure other markings.

- **Ball Possession and Team Control Display:** A transparent overlay at the top or in the corner of the frame shows the percentage ball possession for each team. It is updated frame by frame based on the team of the current ball possessor.

- **Camera Movement Adjustment:** Since camera movements tend to deceive tracking, a secondary annotation box shows live camera movement data so viewers can see how the system adjusts for shifting views.

- **Final Video Output:** After all the annotations have been added, the modified frames are saved and combined into a processed video file using *cv2.VideoWriter()*.

5.14 Web Integration

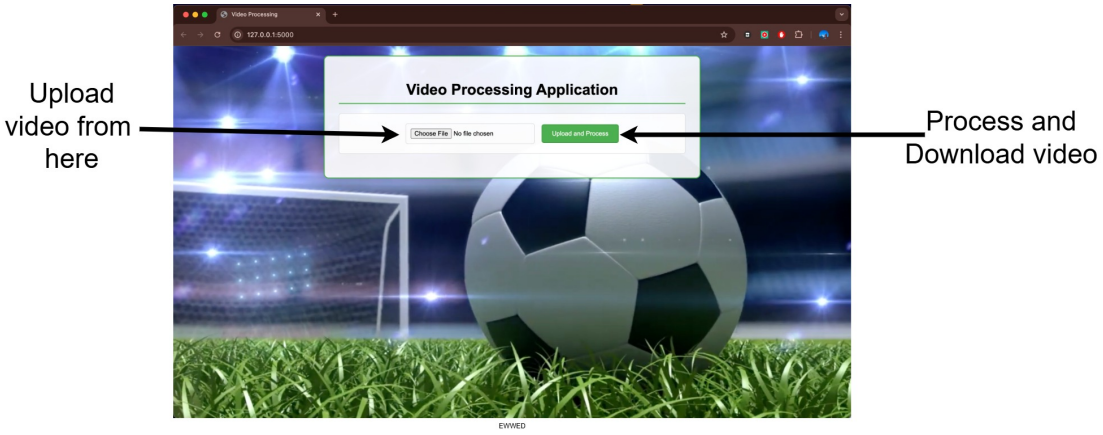

Figure 5.13: Website

In the web integration of our project, the front-end development is designed to provide a simple user interface (UI) using HTML, CSS, and JavaScript. The UI enables users to upload a video file that they want to process. This video upload is essential as it serves as the input for the back-end system, where the core processing takes place.

On the back-end, Flask is utilized for video processing. Once a video is uploaded from the front-end, it is sent to the backend, where the AI/ML model is triggered to process the video . The processing involves several steps, including object tracking, identifying the ball's position, and highlighting active players. These activities are carried out by the AI/ML model as it analyzes the video frame by frame.

Once the video has been processed, the resulting video file, complete with tracking and highlights, is temporarily stored on the server. This temporary storage allows the system to manage the file while it prepares to send the processed video back to the client. After processing, the video is streamed back to the user's browser, where it can be played directly, allowing users to view the tracked video with the enhanced features, such as active player highlights and ball positioning.

This seamless flow between front-end and back-end ensures that users can interact with the system easily, upload videos, and receive video processing results for analysis or review.

5.15 Diagrams

5.15.1 Usecase Diagram

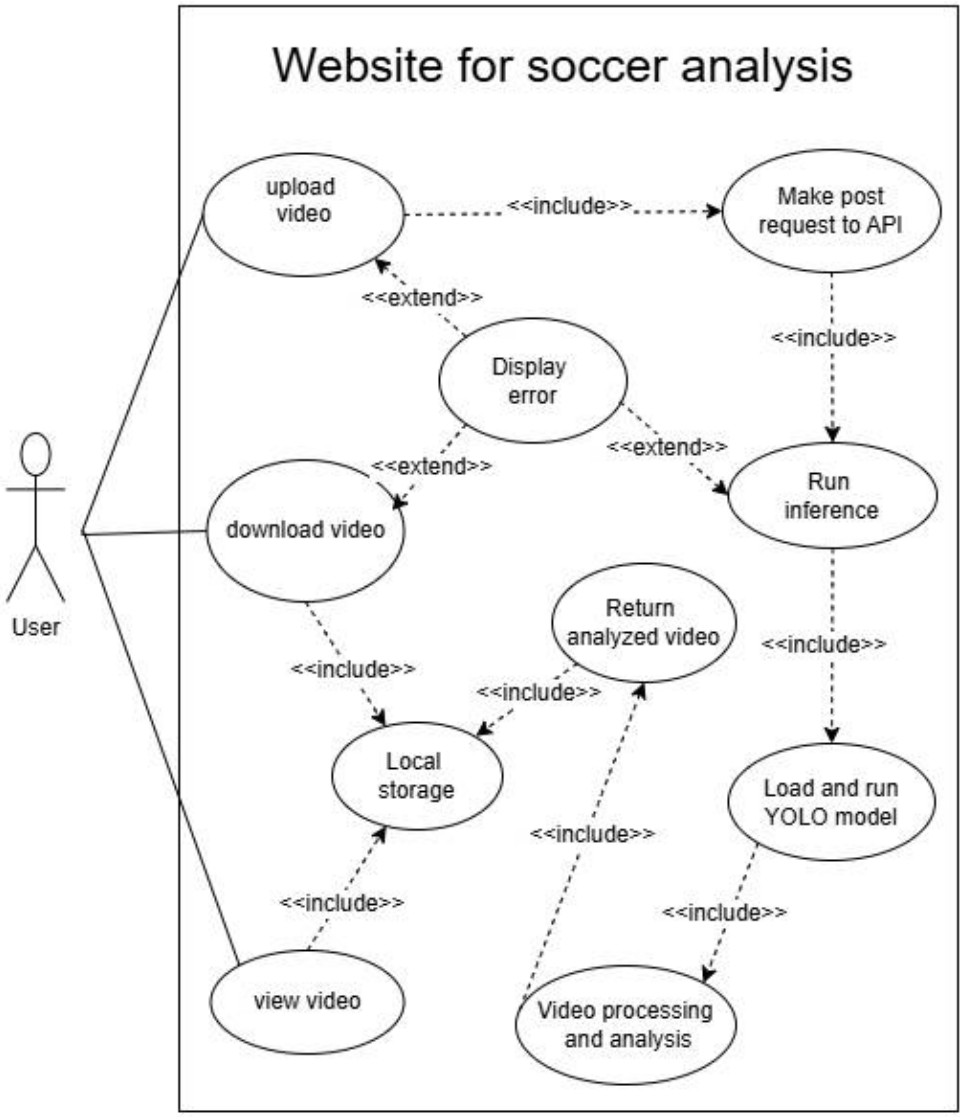

Figure 5.14: Use Case Diagram of the System

5.15.2 Data Flow Diagram

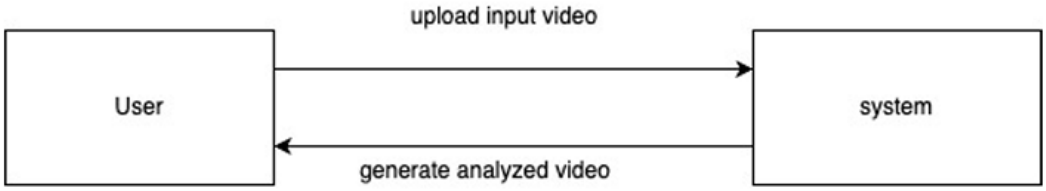

Figure 5.15: Level 0 DFD of the System

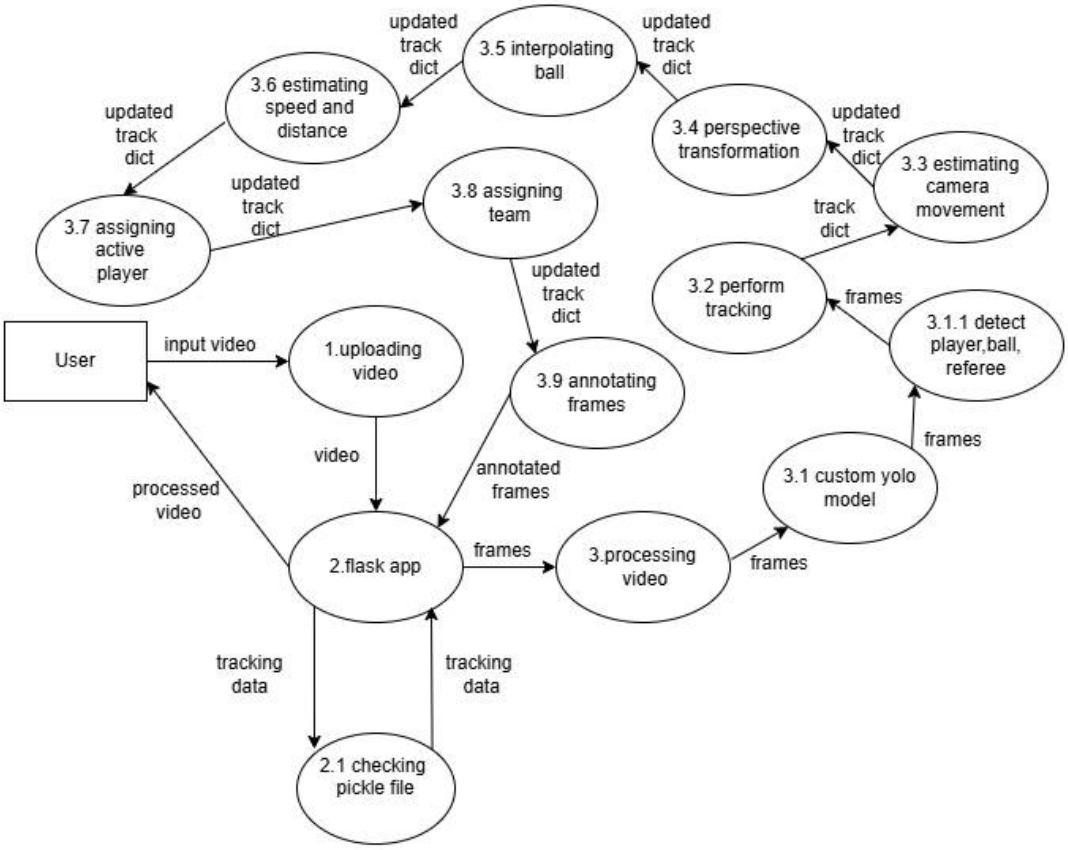

Figure 5.16: Level 1 DFD of the System

5.15.3 Sequence Diagram

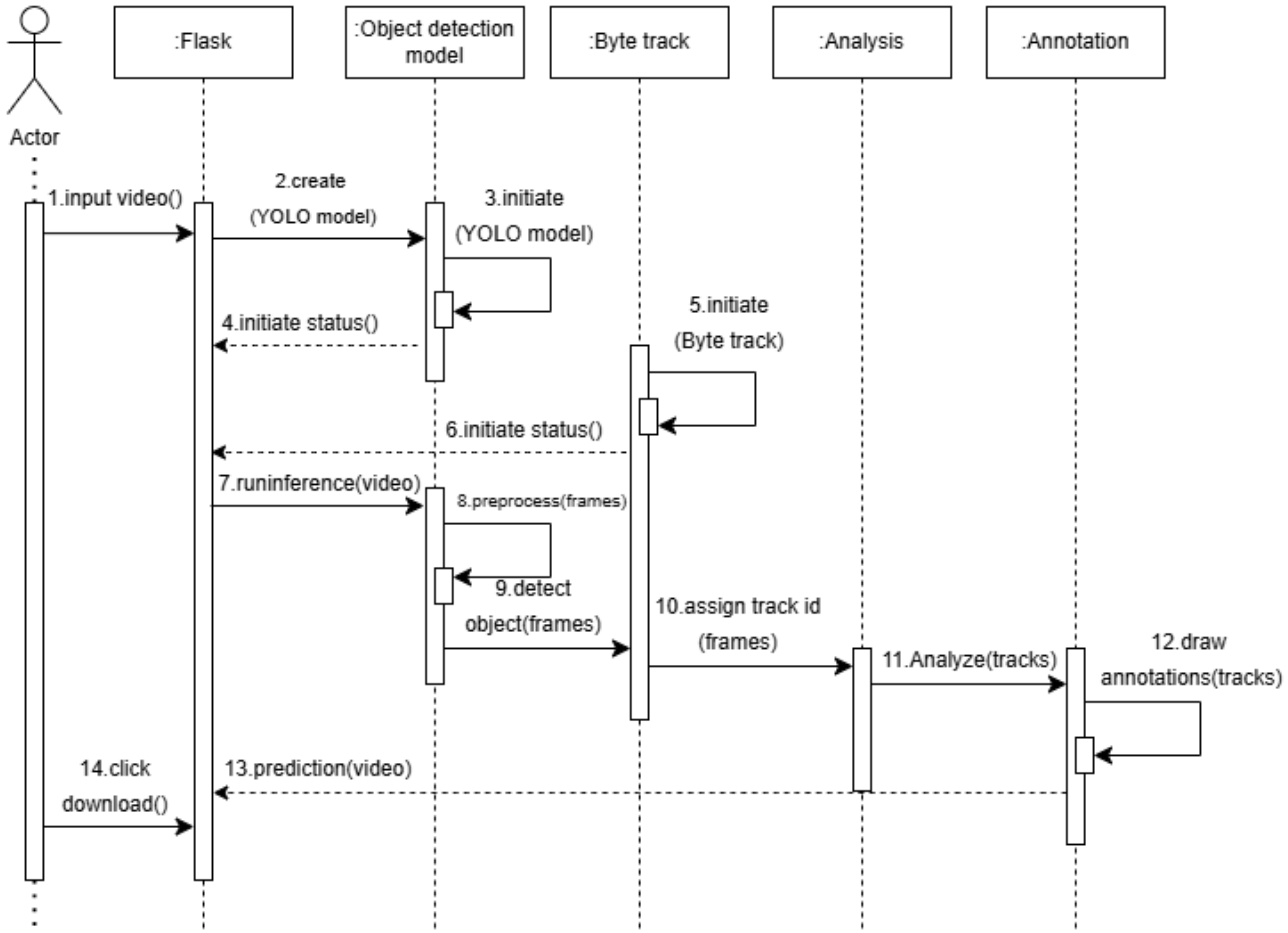

Figure 5.17: Sequence diagram of the System

5.15.4 Collaboration Diagram

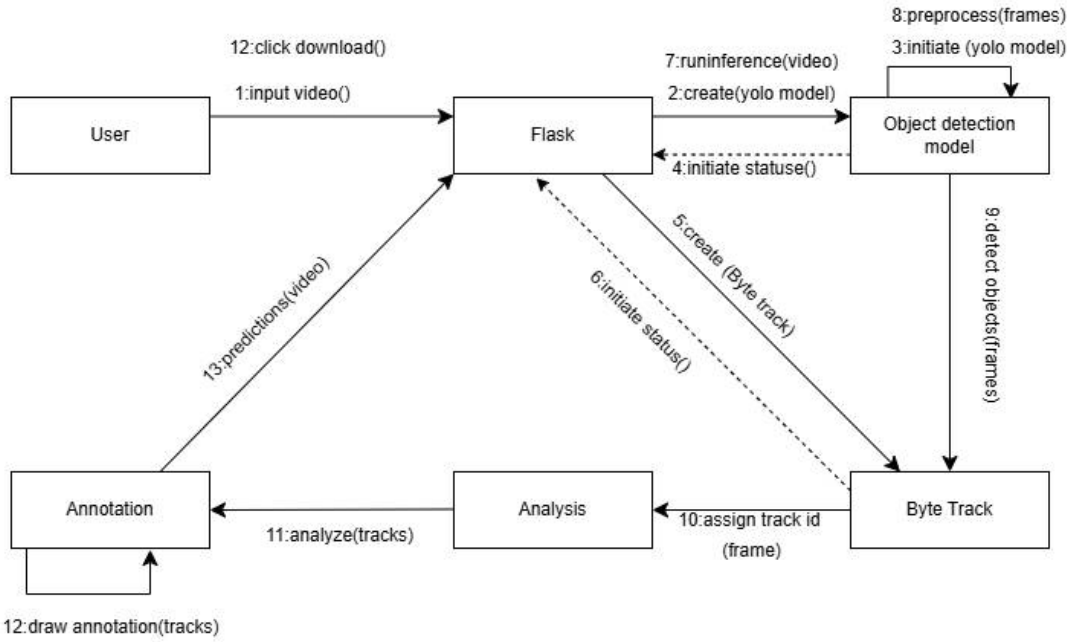

Figure 5.18: Collaboration diagram of the System

5.15.5 Activity Diagram

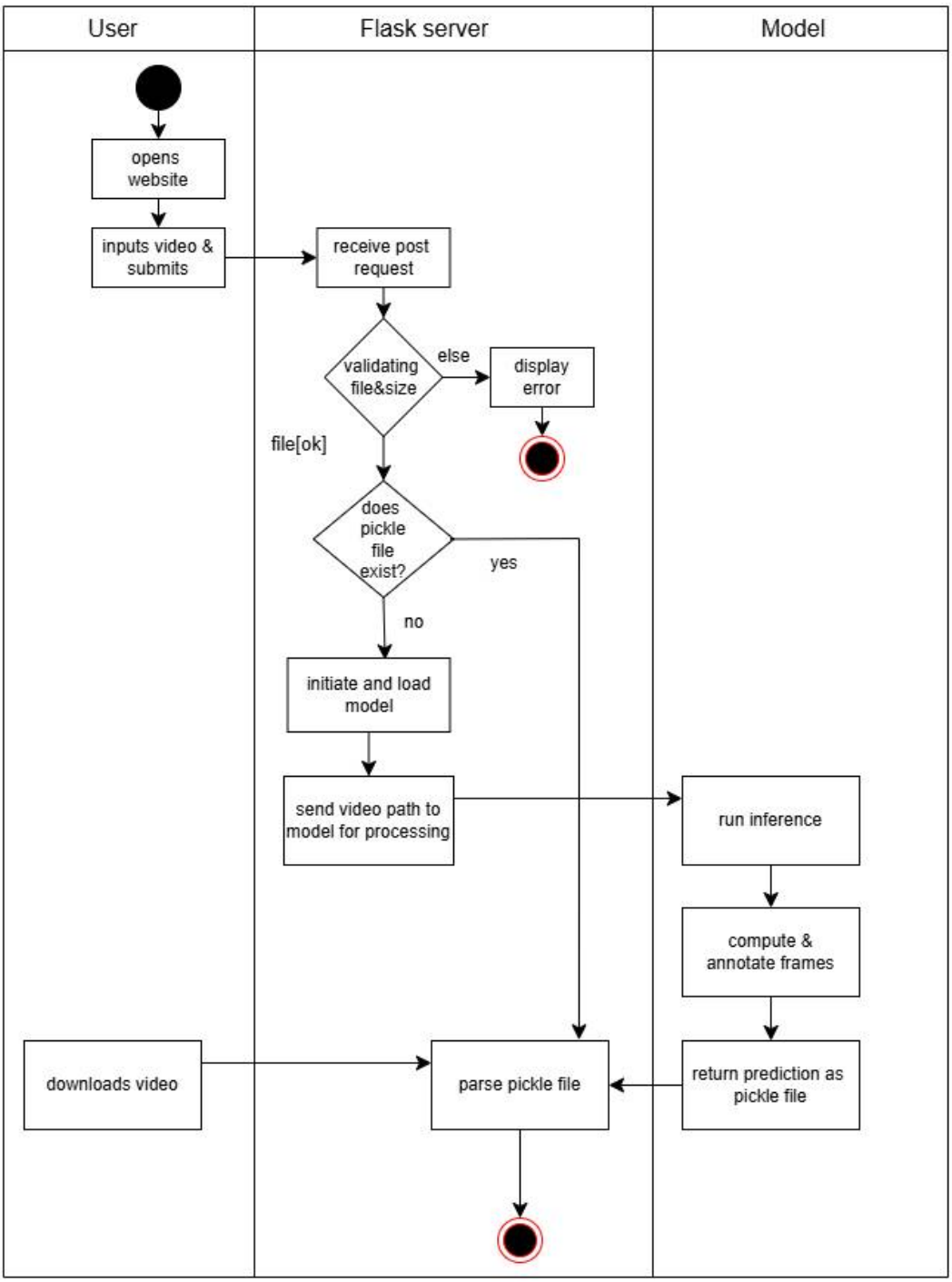

Figure 5.19: Activity diagram of the system

5.15.6 Class diagram

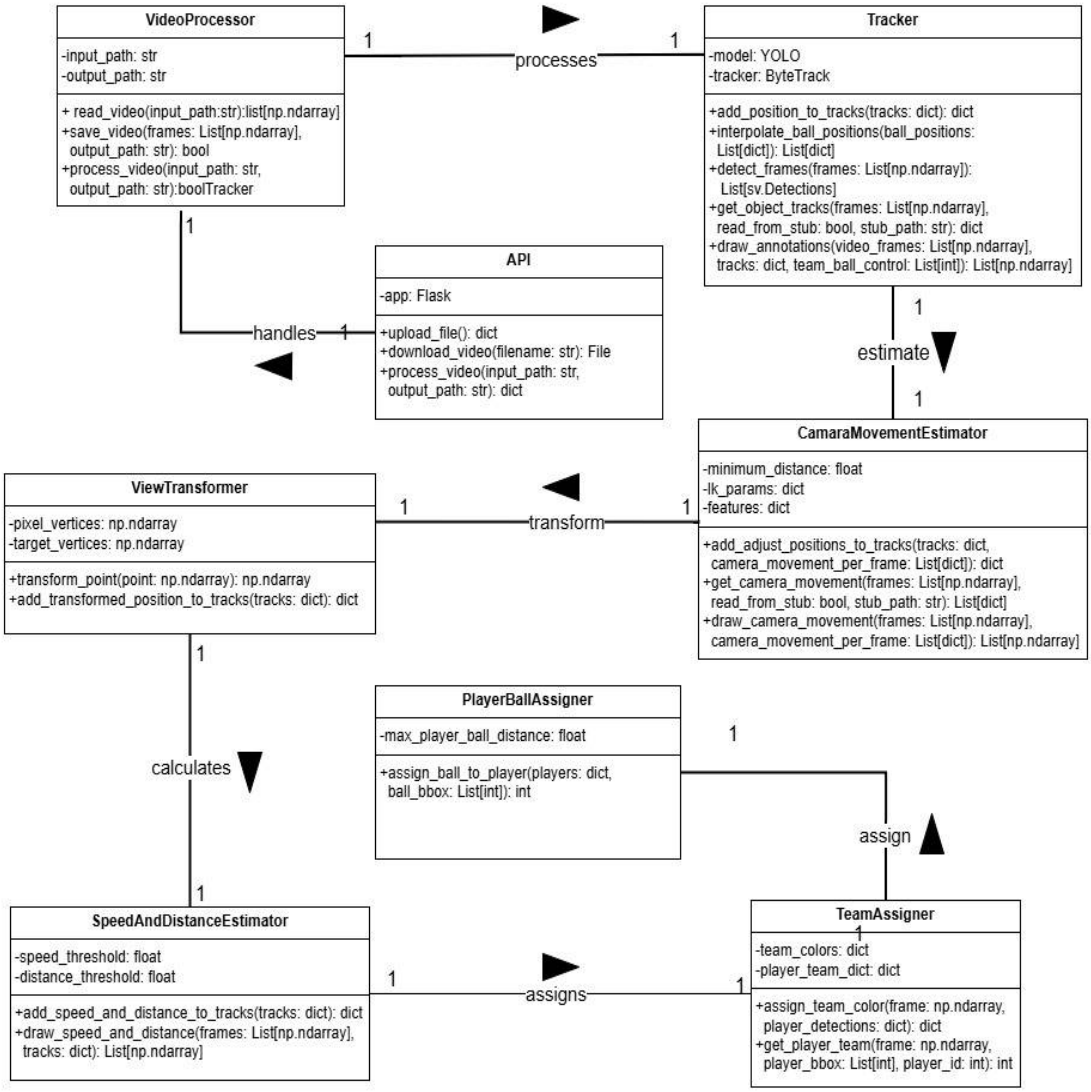

Figure 5.20: Class diagram of the system

5.15.7 Domain diagram

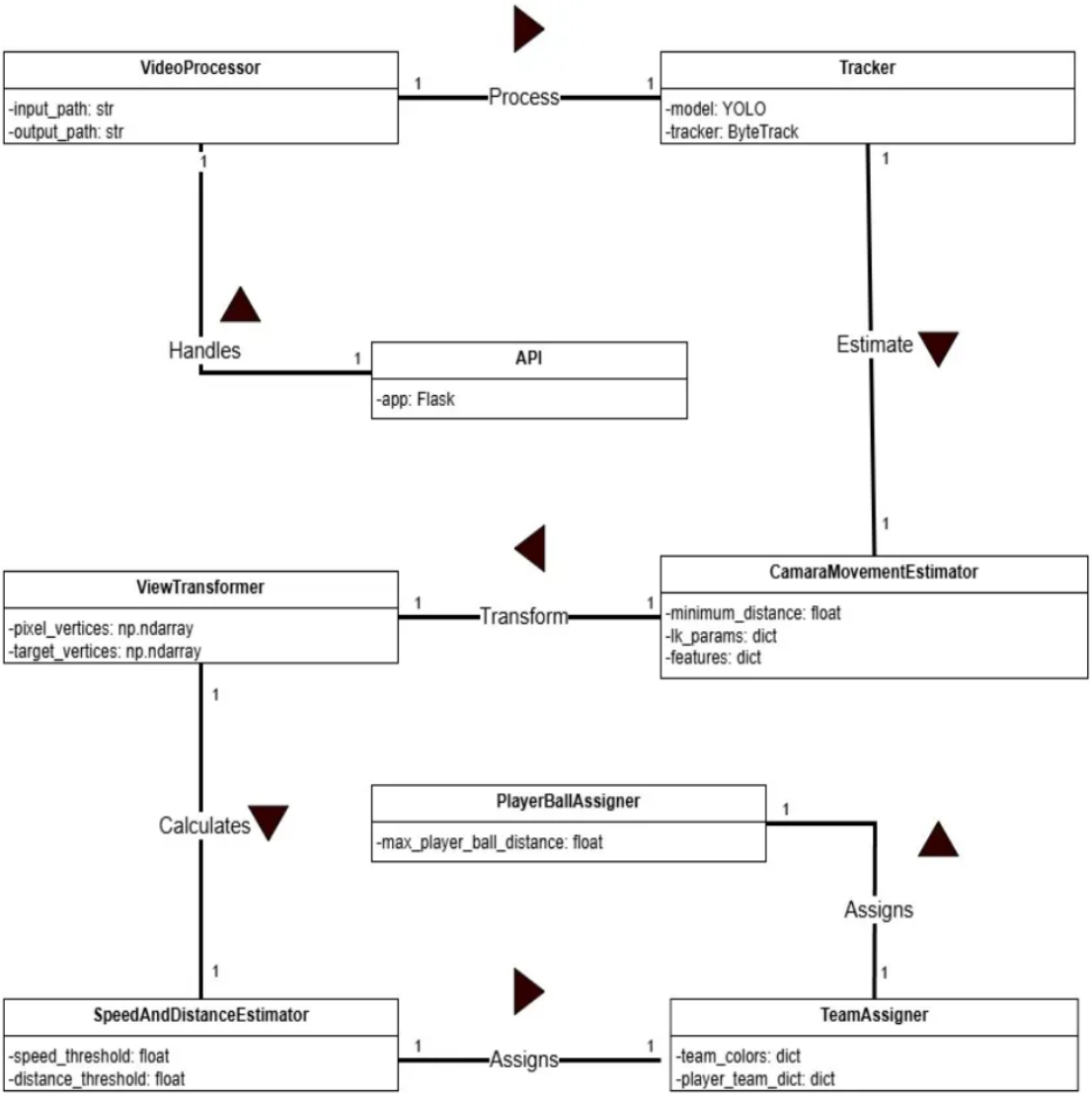

Figure 5.21: Domain diagram of the system

6 RESULTS

6.1 YOLOv5 Model Results

6.1.1 Traning Vs Validation -Loss Curve

The training vs. validation loss curve shows how well the model is learning. Training loss decreases as the model gets better at fitting the training data. Validation loss shows how well the model performs on new, unseen data. At first, both losses go down, but if the model overfits, the validation loss will stop improving and might even go up, while the training loss keeps dropping. A good model has a small gap between the two, showing it's learning well without overfitting. Following are the figures for train/validation loss for box loss, class loss, and DFL loss.

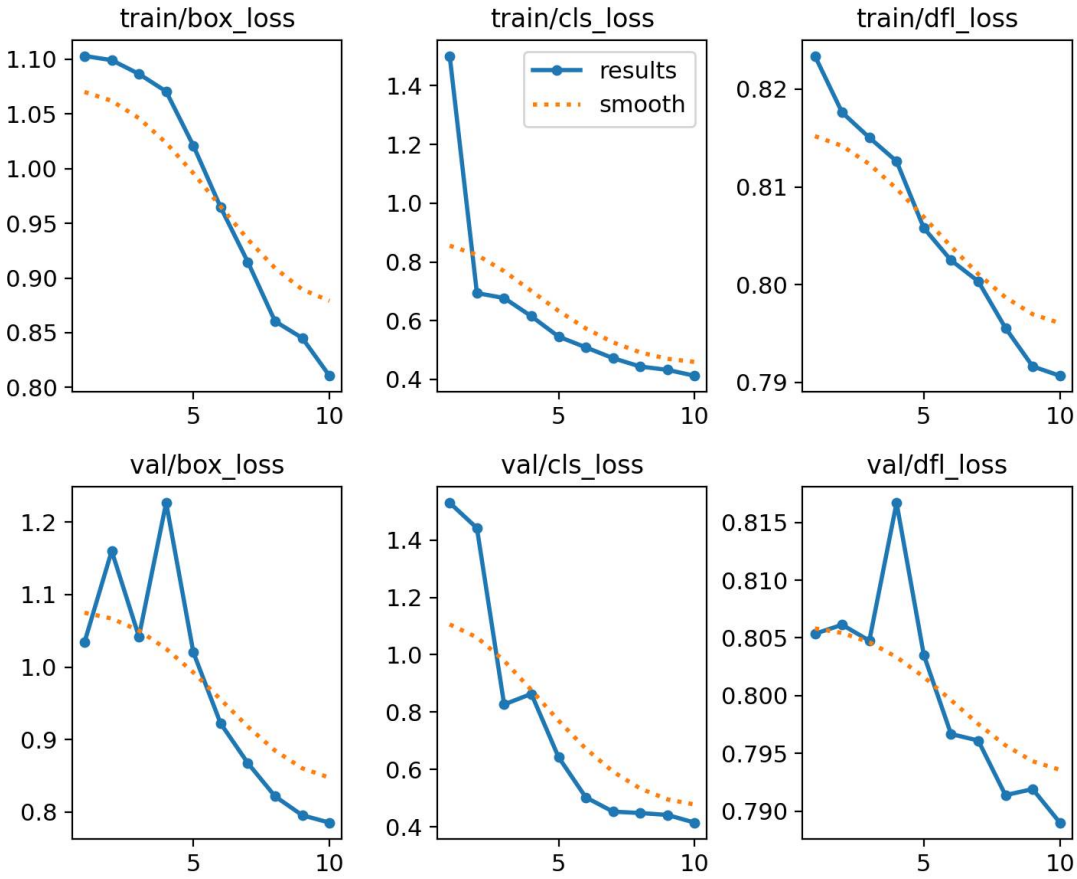

Figure 6.1: Train/Validation loss

6.1.2 Mean Average Precision (mAP)

Mean Average Precision (mAP) is a metric used to measure how well an object detection model is performing. It averages the precision (accuracy) of the model's predictions at different levels of recall (how many true objects it finds) for each class. Then, it calculates the overall average across all classes. A higher mAP means the model is better at finding and correctly identifying objects.

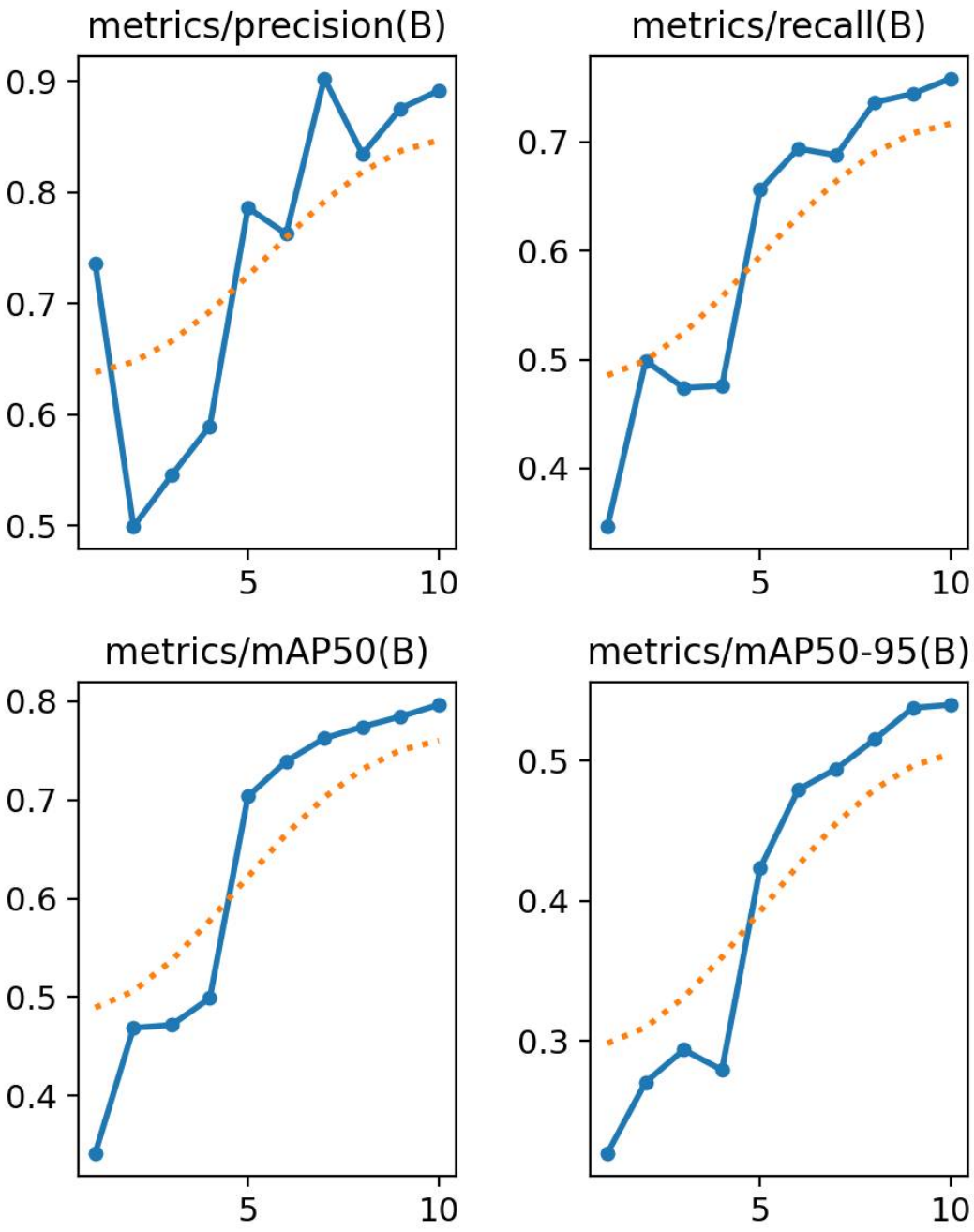

Figure 6.2: mAP metrics

6.1.3 Precision x Recall Curve

The Precision x Recall curve shows how precision and recall change as the model's decision threshold is adjusted. Precision is about how many predicted positives are correct, and recall is about how many actual positives the model finds. The curve helps find a balance between the two, with the goal of having both precision and recall as high as possible.

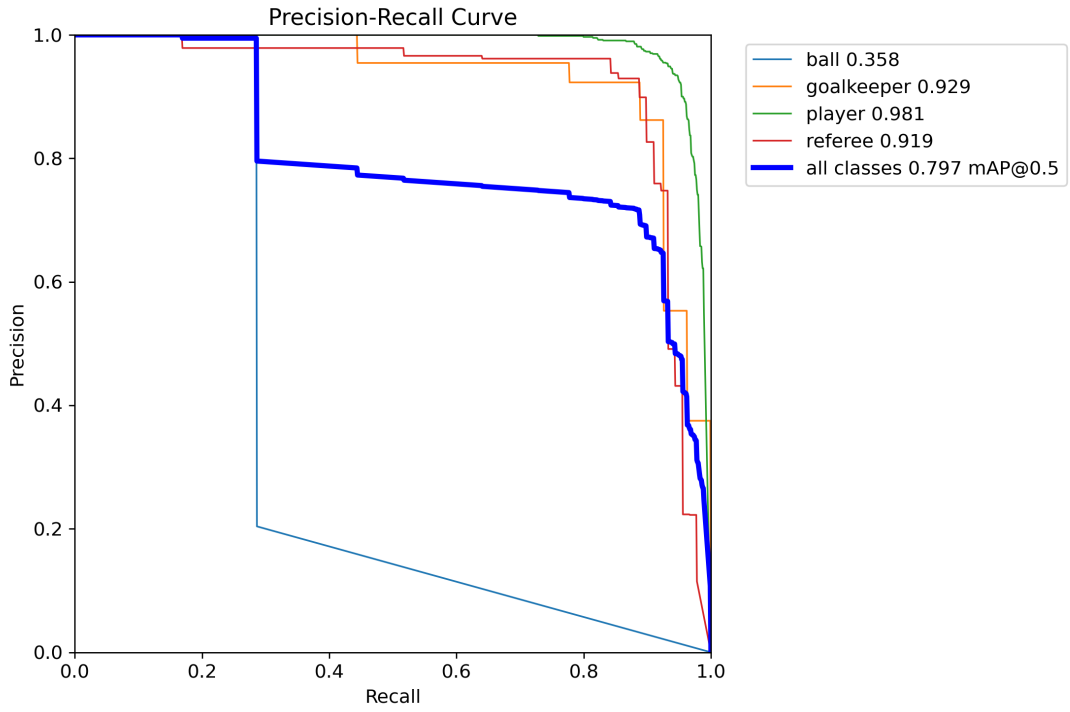

Figure 6.3: Precision x Recall Curve

6.1.4 Confusion Matrix

A confusion matrix compares the predicted labels to the actual labels, showing True Positives, False Positives, True Negatives, and False Negatives. It helps calculate performance metrics like accuracy, precision, and recall.

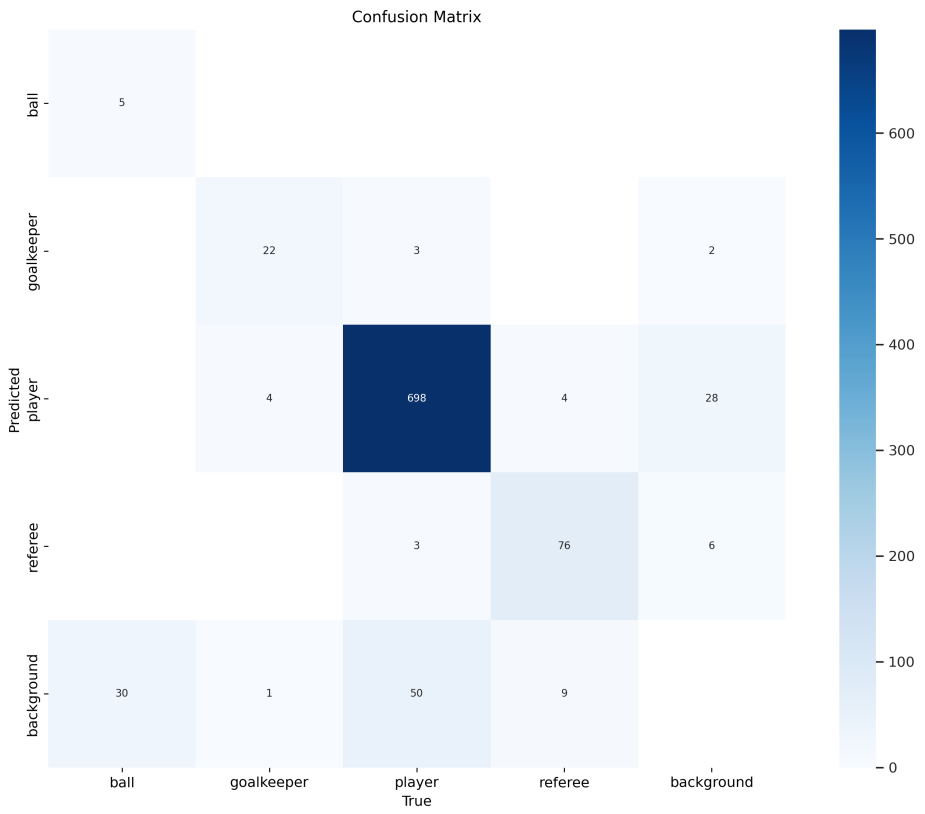

Figure 6.4: Confusion Matrix

6.1.5 Confusion Matrix Normalized

A normalized confusion matrix scales the values in the confusion matrix to show the proportion of predictions rather than raw counts. It helps visualize the model's performance in terms of percentage, making it easier to compare across different datasets or models.

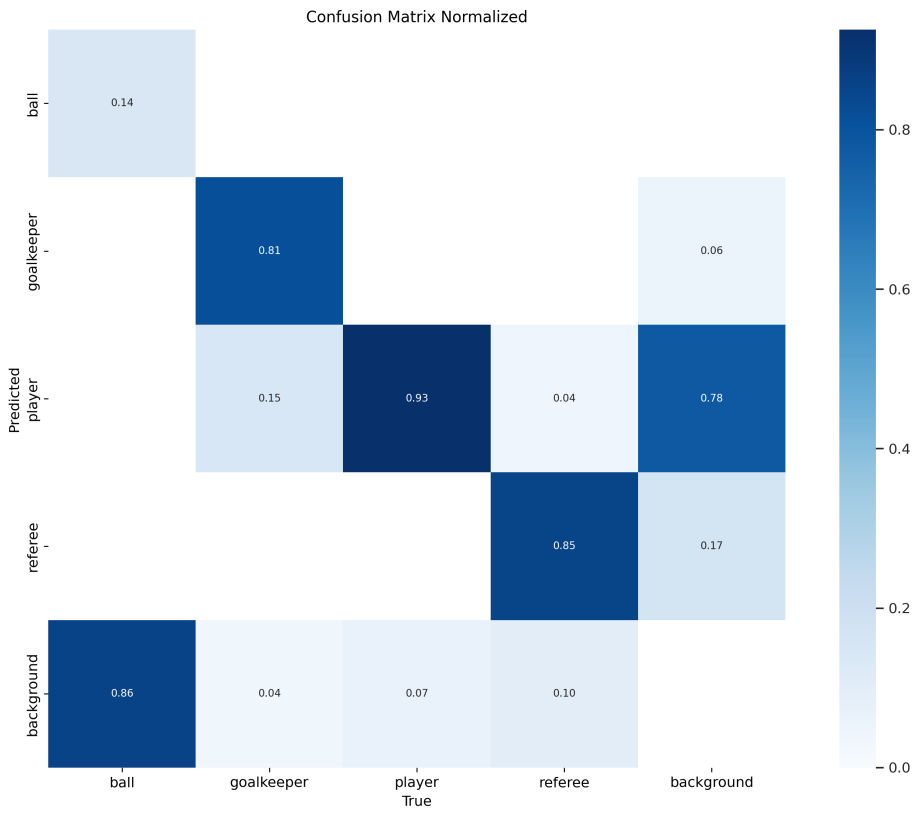

Figure 6.5: Confusion Matrix Normalized

6.1.6 Labels

Labels are the categories or classes assigned to data points in classification tasks.

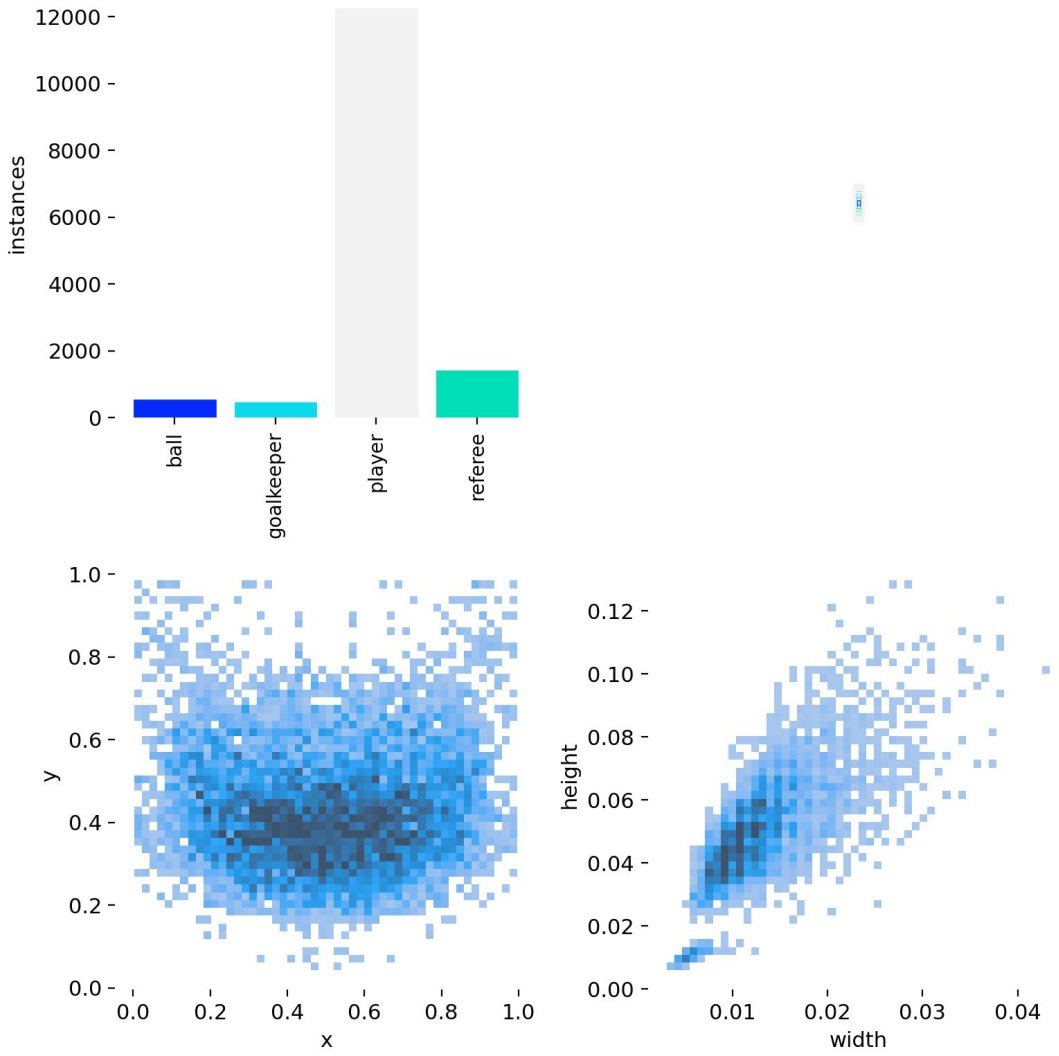

Figure 6.6: Labels

6.1.7 Labels Correlogram

A labels correlogram is a chart that shows how different labels in a dataset are related to each other. It helps to see patterns or connections between the labels.

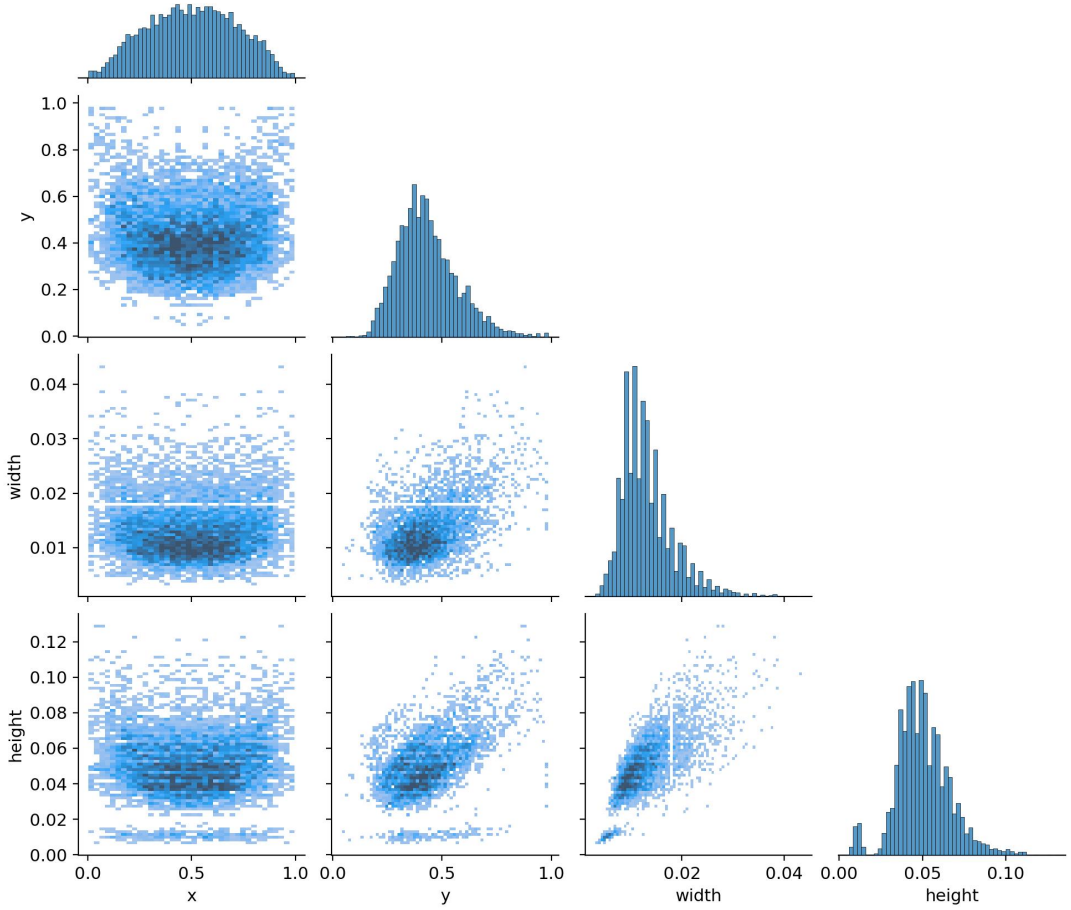

Figure 6.7: Labels Correlogram

6.1.8 F1 curve

The F1 curve shows how the F1 score (a balance of precision and recall) changes at different thresholds. It helps evaluate a model's performance by showing how well it balances correctly identifying positive instances while minimizing false positives and false negatives. A higher F1 score indicates better model performance.

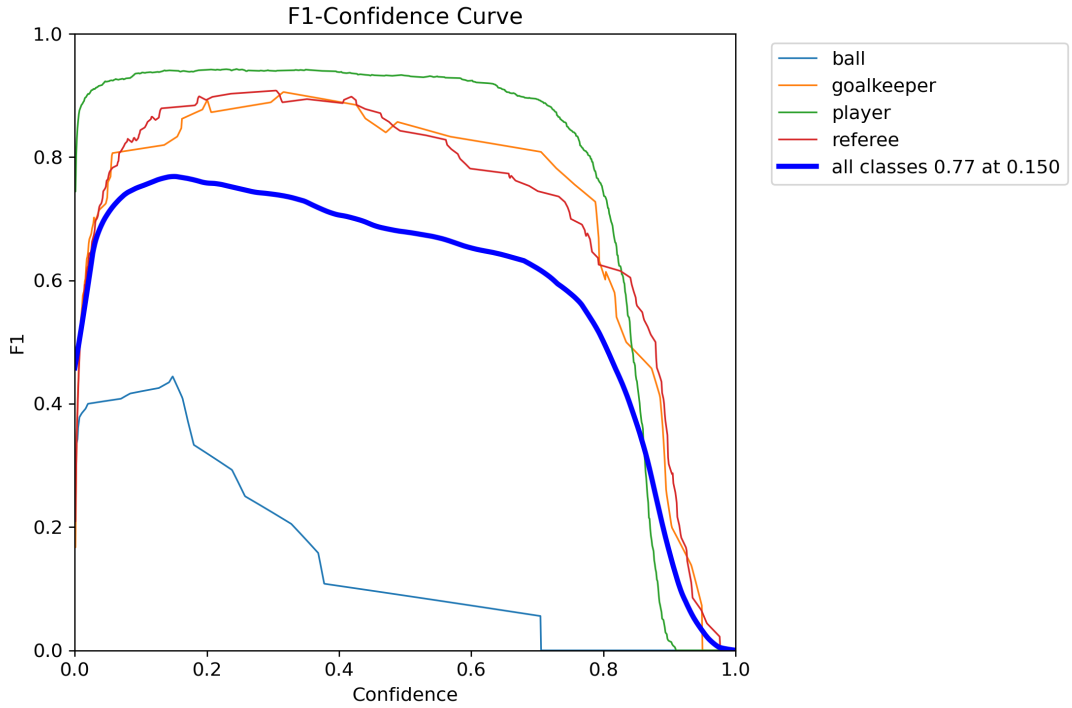

Figure 6.8: F1 curve

6.1.9 Results from different Stages of Annotation

Figure 6.9: detection with Confidence score

This figure shows object detections with bounding boxes around the detected objects, each associated with a confidence score that indicates the model's certainty in its prediction. The class names displayed are "player," "ball," "referee," and "goalkeeper,"

identifying the types of objects detected in the image. These class names, along with the bounding boxes and confidence scores, help visualize the model's ability to detect and classify each object accurately in the scene.

Figure 6.10: Ellipse Annotation with Player Number

In this updated detection, the bounding boxes are replaced with ellipses for a smoother look. Each player is given a unique tracker number, shown next to their class name. A green triangle is added on top of the ball to make it easier to see. This method makes it clearer to track and highlight objects like the ball, improving the overall visibility and understanding of the detections.

Figure 6.11: current ball holder

In this updated detection, a red triangle annotation is added to the current ball holder (the player holding the ball). This helps make it easier to identify which player has possession of the ball, further improving the clarity and visibility of key objects in the scene. The ellipses, tracker numbers, and the green triangle on the ball remain, enhancing overall tracking and object recognition.

Figure 6.12: Possession Displayed for Teams

In this updated detection, a ball possession counter is added, showing how long the cur-

rent ball holder has had possession of the ball. This counter, along with the red triangle on the ball holder, helps track the player's control of the ball over time. The ellipses, tracker numbers, and the green triangle on the ball continue to assist in visualizing and identifying key objects, improving both tracking and game analysis.

Figure 6.13: Camera-moment and speed Displayed

In this updated detection, annotations for camera movement and speed are added. The camera movement is indicated to show how the camera is following the action, and the speed of the current ball holder or player is displayed to track their movement. This, along with the ball possession counter, red triangle on the ball holder, ellipses, tracker numbers, and the green triangle on the ball, provides a complete and detailed view of the game's dynamics, improving both visualization and analysis.

6.1.10 Final website Result

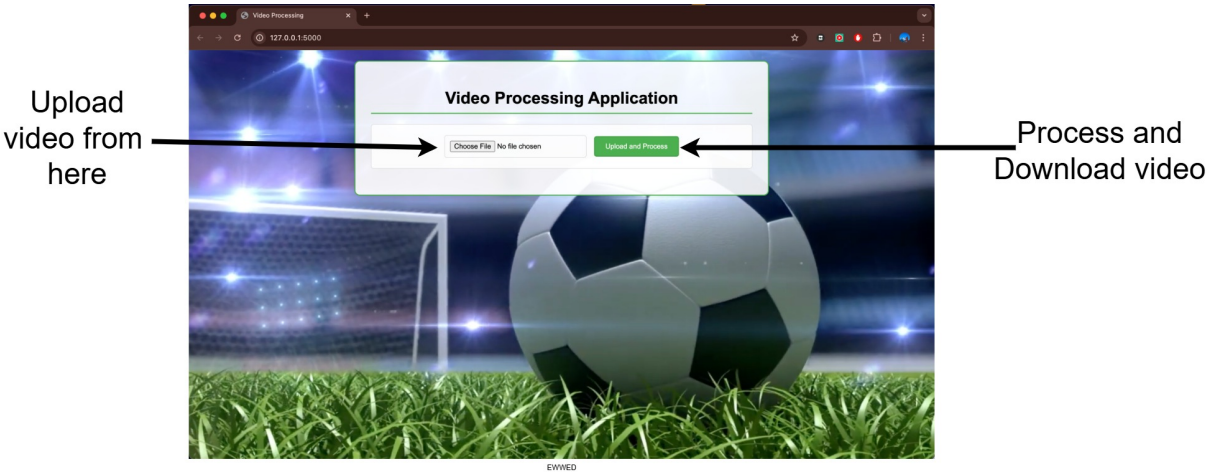

Figure 6.14: Website

This website is our final product: a video processing platform that allows users to upload a video file, process it, and then download the processed result. It features a clean and simple interface with an intuitive user experience. A user simply needs to click the ”Choose File” button to select their video, then click ”Upload and Process”. Once the processing is complete, a download link will be provided to obtain the processed video.

6.2 Analysis

This project focuses on leveraging machine learning and computer vision techniques to analyze soccer games, with an emphasis on player tracking, ball tracking, team assignment, and active player detection. By combining these methodologies, the system provides a detailed analysis of player movements and ball positions during the match. Each component of the system plays a crucial role in ensuring accurate and efficient analysis of the game.

The integration of a web interface using Flask enables users to upload video files for processing. Once uploaded,first our custom trained yolo model is loaded with that video which gives bounding boxes and confidence scores of every objects in each frames then the system processes the video by applying object tracking algorithms, such as ByteTrack, to analyze player and ball positions. The processed video, including the tracking data of players and ball movement, is then sent back to the front-end for visualization. This integration allows users to interact with the system and visualize the game dy-

namics, providing a straightforward way to track player movements and ball activity throughout the match.

ByteTrack is central to the project, ensuring that the identity of players and the ball is maintained across video frames. Through motion prediction and object association, ByteTrack effectively tracks objects, even when they temporarily disappear due to occlusion. This is crucial in soccer, where players may overlap or move rapidly. ByteTrack's ability to handle occlusion and re-associate objects as they reappear ensures that tracking remains uninterrupted, providing accurate and consistent data for analysis.

Team assignment is another important feature of the system, using K-Means clustering to group players based on jersey color and position. By clustering players with similar visual features, the system can automatically identify players belonging to the same team. However, challenges can arise if teams wear similar-colored jerseys or if the camera view changes significantly, which could affect the clustering algorithm's accuracy. Refining the feature extraction and clustering process could improve the system's ability to handle such scenarios.

Active player detection enhances the analysis by identifying and highlighting the player closest to the ball. By calculating the Euclidean distance between each player and the ball in each frame, the system can determine the player who is actively involved with the ball. The active player is visually marked with a red triangle, making it easy for users to identify them. This feature allows for a clearer understanding of player involvement during the match, providing a visual focus on the player closest to the ball.

The annotation process in the system adds clarity by visualizing the tracked objects. Instead of using traditional bounding boxes, the system uses ellipses, which offer a smoother and more natural representation of the players and ball. Players are annotated with tracker IDs, and the ball is highlighted with a green triangle, allowing for easy identification and tracking throughout the video. This visualization helps users follow the movement of players and the ball and provides a clearer representation of the game's progression.

Speed and distance calculation adds valuable performance metrics to the analysis. By

calculating the Euclidean distance between a player's position in two consecutive frames, the system measures how much distance the player has covered. The speed is calculated by dividing the distance by the time difference between frames. This provides insight into a player's physical performance and activity, offering quantitative data on how fast they are moving and how much ground they have covered during the match.

While the system performs well, there are several challenges that need to be addressed. One of the main challenges is accurately assigning teams when teams wear similar colors or when the camera perspective changes significantly. In such cases, the clustering algorithm may have difficulty distinguishing players. Further refinement in feature extraction and the clustering algorithm could improve team assignment accuracy. Additionally, handling occlusions and overlaps in crowded scenes could be improved by incorporating more advanced object detection models to enhance tracking robustness.

Although the system processes videos effectively, there is room for improvement in processing efficiency. Reducing latency and optimizing the processing pipeline would enhance the system's scalability and usability, especially for more complex video data. Improving processing speed would make the system more efficient, allowing for quicker analysis of videos.

In conclusion, this project successfully integrates various computer vision and machine learning techniques to analyze soccer games. The system's ability to track players and the ball, calculate movement metrics, identify active players, and assign teams based on visual features demonstrates the potential of automated sports analysis. While there are areas for improvement, such as handling dynamic camera views, occlusions, and optimizing processing efficiency, the system offers valuable insights into player performance and game analysis. As the project continues to evolve, addressing these challenges will further enhance its functionality and make it more applicable to practical soccer analysis.

7 CONCLUSION AND FUTURE ENHANCEMENT

7.1 Conclusion

In conclusion, the project's two goals have been effectively met. In football match videos, precise player, referee, and ball detection and tracking were made possible by the use of YOLOv5 and ByteTrack. These models were included into an online tool that allows users to submit video from games and examine team assignments, player movements, and ball tracking. Additionally, the system deployed perspective transformation techniques to change the camera view into a top-down perspective for more accurate analysis, and it used K-Means clustering for automated team assignment. Throughout the project many problems were encountered like,Occlusions during ball tracking, different camera angles that affected detection accuracy, and player misclassification because of overlapping bounding boxes were some of the difficulties that arose during the project. Nevertheless, these difficulties were overcome by improving tracking algorithms to increase overall accuracy, employing perspective transformation for field mapping, and interpolating for missing ball positions. Future improvements have also been suggested, such as automatic event detection to identify crucial events like goals, fouls, and passes; enhanced ball possession analysis; and player activity recognition by pose estimation. These upgrades will improve the system's usability and offer more in-depth match analysis information. The project shows a strong grasp of football analytics and can be expanded for more extensive sports analysis and real-time applications.

7.2 Future recommendation

7.2.1 Enhancing the accuracy of system

Expanding the dataset by adding more frames from a greater range of recorded football match footage could be a potential improvement to increase the system's accuracy.Currently, our YOLO model is only trained on 620 images, which might limit detection precision under different match conditions. Making the dataset larger will increase detection robustness. Development in the future will be to make the dataset larger by collecting more soccer images labeled from other leagues, lighting conditions, and camera views. In addition, data augmentation techniques such as rotation, scaling, and brightness will be employed to increase flexibility in the model. Fine-tuning the model with larger datasets such as SoccerNet or COCO will enhance detection performance.

7.2.2 Real-Time Processing in Live Match

The system is now working offline on video processing, limiting its use in live match analysis. To enable real-time insights to be extracted, there will be various optimizations. First of all, the YOLO inference speed will be accelerated with the help of software like TensorRT or ONNX Runtime to facilitate rapid model execution. Apart from that, GPU acceleration and parallel processing techniques will be employed in order to accelerate frames. Integration of WebRTC-based live streaming capabilities will provide real-time match analysis and visualization of insights in real-time. Reducing latency in processing will allow for virtually instant feedback to the audience and match analysts.

7.2.3 Advanced Ball Tracking and Possession Analysis

Ball tracking in the system faces challenges such as occlusions, where the ball is temporarily hidden behind players, and rapid movements that make continuous tracking difficult. To enhance trajectory prediction, integrating Kalman Filters with DeepSORT can improve tracking accuracy. The Kalman Filter estimates the ball's future position based on previous states, allowing tracking to continue even when the ball is momentarily lost. DeepSORT, on the other hand, utilizes deep learning to re-identify objects across frames, ensuring the ball remains correctly tracked despite multiple moving objects. By combining these techniques, the system enhances tracking reliability and reduces misidentifications. Additionally, analyzing ball possession is crucial for tactical evaluation, which can be achieved by calculating the Euclidean distance between the ball and each player, assigning possession to the closest player. Tracking ball movement patterns further provides insights into passing accuracy, team coordination, and overall possession control. This enables the system to generate valuable statistics such as possession percentage, pass sequences, and player influence, offering deeper analytical insights into team strategies and individual performances.

7.2.4 Player Action Recognition

By utilizing object tracking and posture estimation techniques, an Action Classification Model may be implemented to recognize and classify different player activities, including tackling, dribbling, shooting, and passing. While pose estimation uses body posture and limb location to identify individual actions, object tracking makes sure that player

movements are continuously monitored between frames. The model can effectively recognize motions and identify patterns because it is trained on a labeled dataset that includes various football actions. By recognizing movement patterns and evaluating their efficacy, this model's integration with the system enables comprehensive statistics on player performance, strategy execution, and team dynamics. It also supports automated match analysis, coaching insights, and player skill appraisal. Using pose estimation frameworks such as OpenPose or MediaPipe to improve action classification increases the system's capacity to

7.2.5 Automatic Event Detection

Developing an AI-based Event Recognition System enables the automatic detection of crucial match events such as goals, fouls, substitutions, and offsides by utilizing a combination of rule-based logic and machine learning models. This system aims to minimize human intervention by automating event detection, making football analysis more efficient and accurate.

- **Rule-Based Logic:** Some events, like offsides and substitutions, follow clear, predefined rules. For example, offsides can be detected by analyzing player positions relative to the last defender and the ball when a pass is made. Similarly, substitutions can be identified by tracking the entry and exit of players on the field.
- **Machine Learning Models:** More complex events, such as fouls and goals, require pattern recognition based on visual and contextual cues. Machine learning models can be trained to analyze player interactions, body movements, and ball dynamics to determine if a foul or a goal has occurred. This system can be used to assist referees, broadcasters, and analysts by providing instant event detection and reducing the reliance on manual annotation. By integrating this with a football analysis platform, match statistics and highlights can be generated automatically, improving the overall viewing experience and tactical breakdowns.

APPENDIX A

A.1 Project Schedule

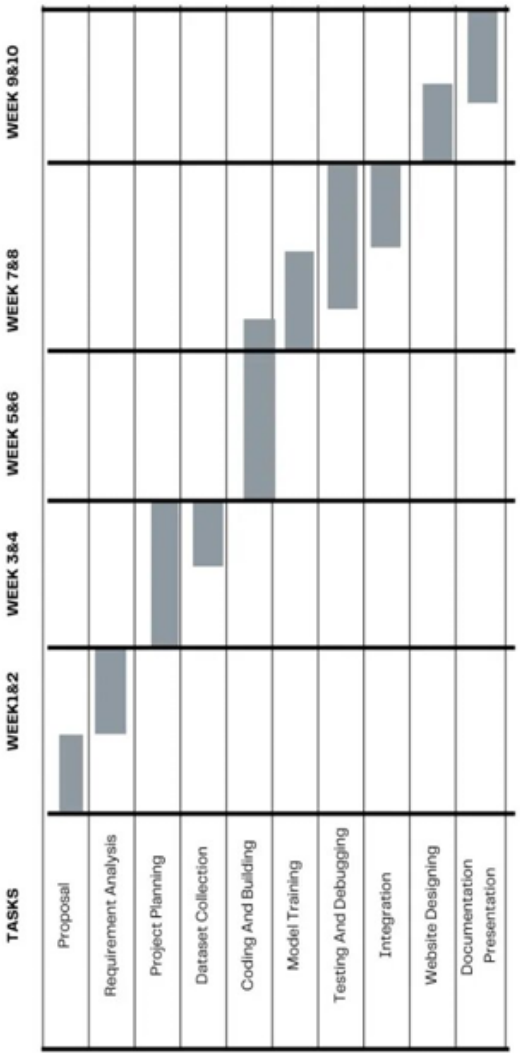

Figure A.1: Gantt Chart

REFERENCES

[1] P. Perez et al. Kalman and particle filtering approaches in player tracking. *Journal of Sports Analytics*, 23(4):123–145, 2002.

[2] J. Redmon et al. Yolo: You only look once. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 779–788, 2016.

[3] Y. Zhang et al. Deepsort: Deep learning-based object tracking. *IEEE Transactions on Image Processing*, 28(10):4234–4245, 2019.

[4] J.L. Barron et al. Optical flow estimation and its application to object tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(5):493–505, 1994.

[5] N. Pappas et al. Homography-based perspective transformation for sports video analysis. *Computer Vision and Sports Analytics*, 5:90–103, 2018.

[6] Z. Li et al. Real-time performance metrics calculation in sports analytics. *Journal of Machine Learning in Sports*, 11:234–245, 2019.

[7] C. Krauss et al. Interactive visualizations for football analytics: Advancements and applications. *International Journal of Sports Data Science*, 2(1):32–45, 2020.