Group Project Report

Group – 08

Topic : Classifying Entities in Bundesliga Matches

Kalyani Vinayagam Sivasundari

Shanun Randev

Vishnu Arun

Deep Learning (DATS – 6303)

Dr. Amir Jafari

Introduction

In the realm of professional football, the acquisition and analysis of event data are paramount for enhancing team strategies and uncovering emerging talents. This project targets the automatic detection and classification of specific football events—namely, passes, throw-ins, crosses, and challenges—within video recordings from Bundesliga matches. The significance of this endeavor lies in its potential to extend the benefits of sophisticated event analysis to lower-tier leagues, which traditionally suffer from a lack of resources. By automating this process, we aim to democratize access to valuable performance analytics, thereby fostering the sport's growth at all levels.

This automated event detection is facilitated by advancements in computer vision and deep learning technologies. We employ the latest iteration of the YOLO (You Only Look Once) architecture, YOLOv8, known for its efficiency and accuracy in real-time object detection.

The goal of this project is twofold: to enhance the current manual method of event annotation, which is labor-intensive and error-prone, and to ensure that no potential talent goes unnoticed, regardless of the league's level of exposure or resources.

Dataset

The dataset underpinning this project comprises video recordings from nine Bundesliga football games, split into halves. This rich dataset is sourced from the DFL - Bundesliga Data Shootout on Kaggle and includes:

Training Data (train): Consists of video recordings from eight games, providing a comprehensive set of scenarios for training our model. For four of these games, both halves are included, while for the other four, only one half is provided. These videos come annotated with various football events such as challenges, plays, and throw-ins, alongside their specific occurrence times within the videos, creating a robust training environment for our models.

Test Data (test): Contains video recordings for evaluating the model's performance. It includes one full game and four half-games, with the complementary halves included in the training set. This setup is designed to rigorously test the model's ability to generalize across different game scenarios.

Additional Clips (clips): Features short clips from ten additional games. These are not annotated and are intended to help the model generalize to a variety of match environments not specifically represented in the main training or test data.

The event annotations are detailed in train.csv, providing a structured format that includes:

video_id: Identifier for the video where the event occurred.

event: Specifies the type of event, such as challenge, play, or throw-in.

event_attributes: Describes additional attributes relevant to the event.

time: Marks the exact time (in seconds) of the event within the video.

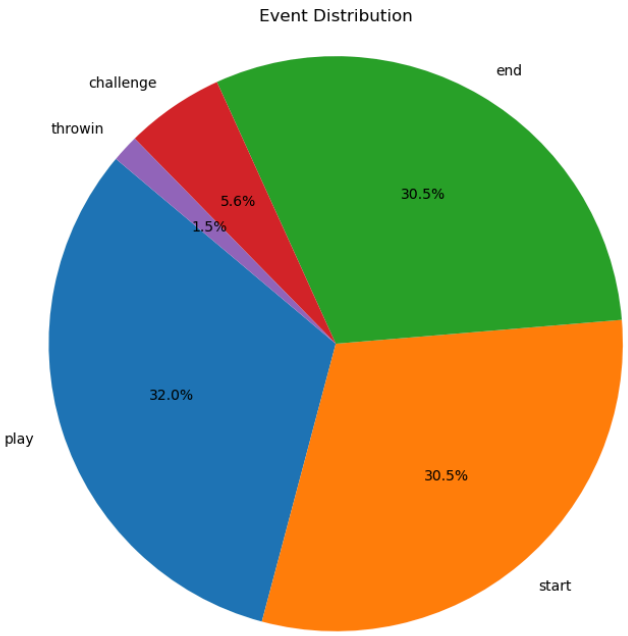
This dataset is not only comprehensive in terms of the volume and variety of the game footage but also in its detailed annotation of specific football events, making it an ideal foundation for training deep learning models tailored to event detection in football matches.

Exploratory Data Analysis

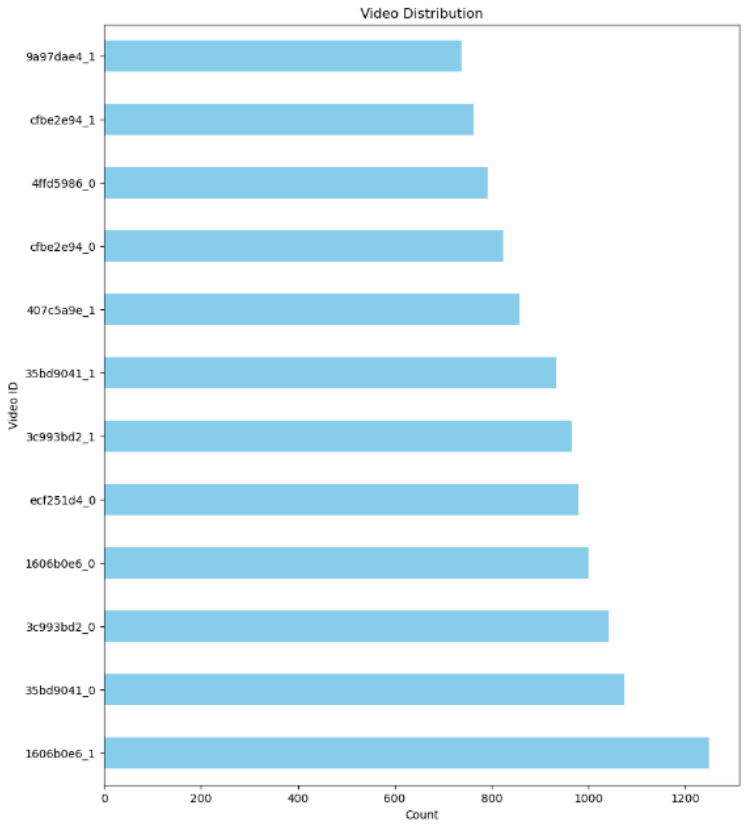
	video_id	time	event	event_attributes
0	1606b0e6_0	200.265822	start	NaN
1	1606b0e6_0	201.150000	challenge	['ball_action_forced']
2	1606b0e6_0	202.765822	end	NaN
3	1606b0e6_0	210.124111	start	NaN
4	1606b0e6_0	210.870000	challenge	['opponent_dispossessed']
...	...	...	...	...
11213	ecf251d4_0	3056.587000	challenge	['opponent_dispossessed']
11214	ecf251d4_0	3058.072895	end	NaN
11215	ecf251d4_0	3068.280519	start	NaN
11216	ecf251d4_0	3069.547000	throwin	['pass']
11217	ecf251d4_0	3070.780519	end	NaN

11218 rows × 4 columns

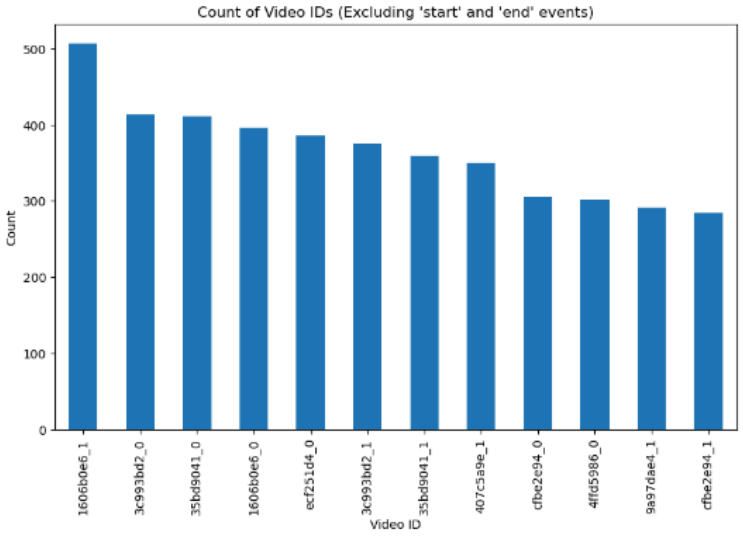
This picture shows how the train.csv file is loaded in a dataframe and it gives us a clear picture of the number of columns and rows with the column names.



This demonstrates the number of times the event occurs in total in the entire training set giving a clear scale of which event occurs the most often and the least.

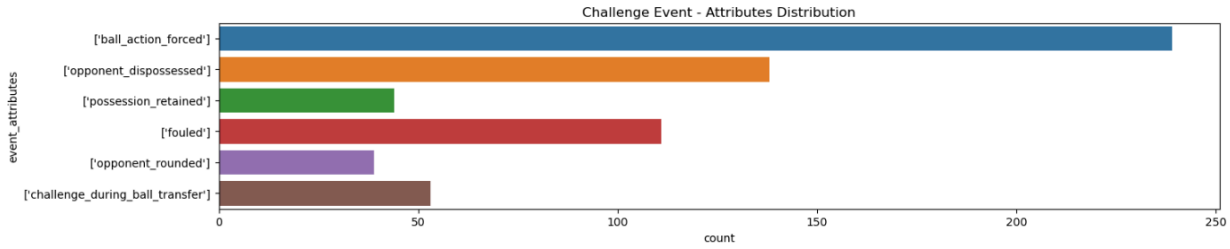


This shows the distribution of video ids and across the dataset representing no. of times the video split into frames and loaded in the training set.

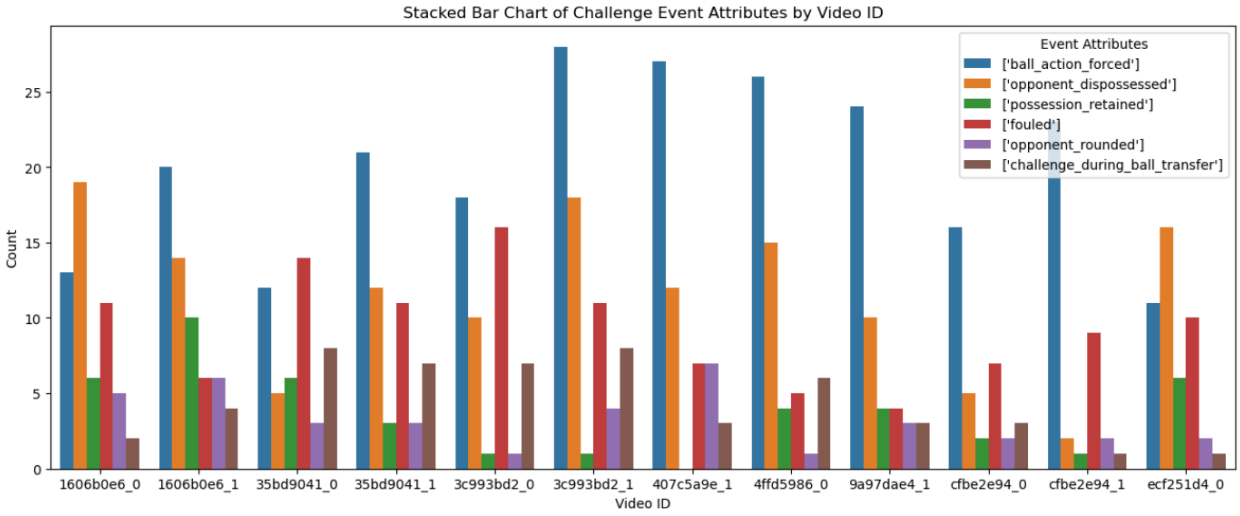


This represents video-id distribution with respect to events excluding “start” and “end” events.

Challenge Event:

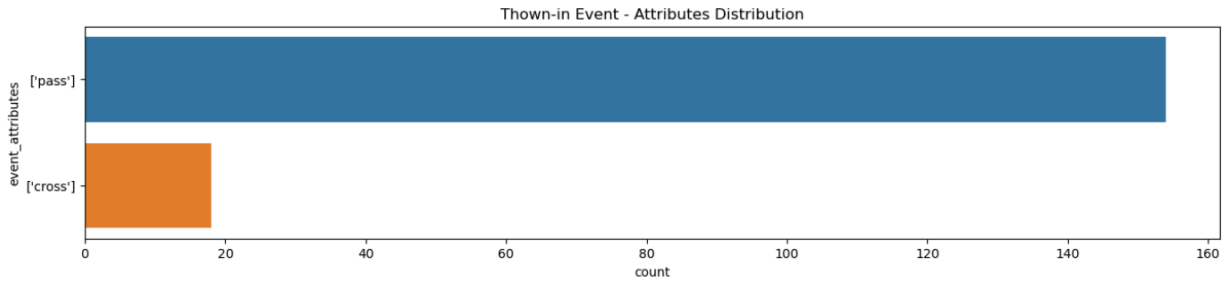


This represents the event attributes distribution in accordance with the event challenge.

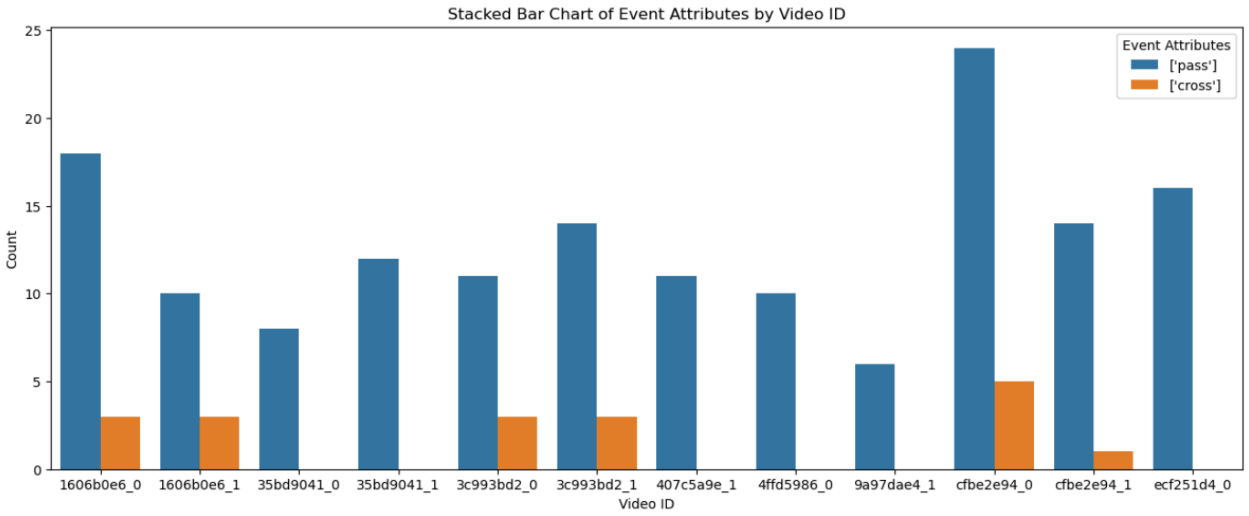


This represents the event attributes distribution in accordance with the video-ids.

Throw-in Event:

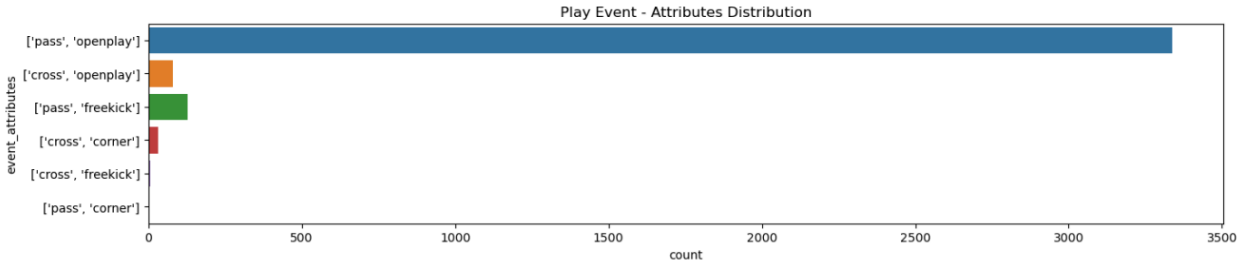


This represents the event attributes distribution in accordance with the event throw-in.

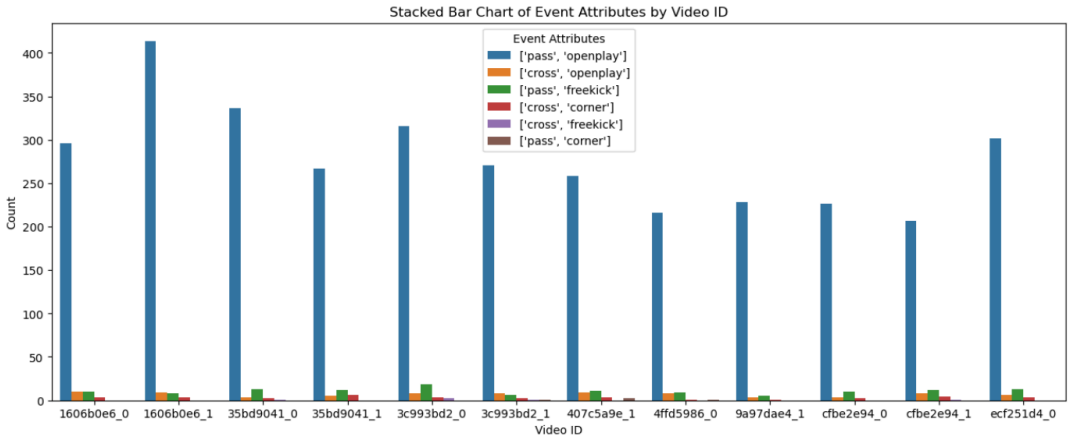


This represents the event attributes distribution in accordance with the video-ids.

Play Event:



This represents the event attributes distribution in accordance with the event play.



This represents the event attributes distribution in accordance with the video-ids.

YOLOv5x model

The YOLOv5x model represents a significant advancement in the field of object detection through its use in real-time applications requiring high accuracy and efficiency. It is part of the YOLO (You Only Look Once) series, which has gained widespread acclaim for its ability to perform object detection tasks swiftly and accurately.

Architecture and Design:

YOLOv5x is the largest and most precise variant within the YOLOv5 series, designed with a deeper and wider network architecture compared to its counterparts (such as YOLOv5s, YOLOv5m, and YOLOv5l). This model variant is particularly optimized for scenarios where detection accuracy is critical and can afford slightly more computational resources.

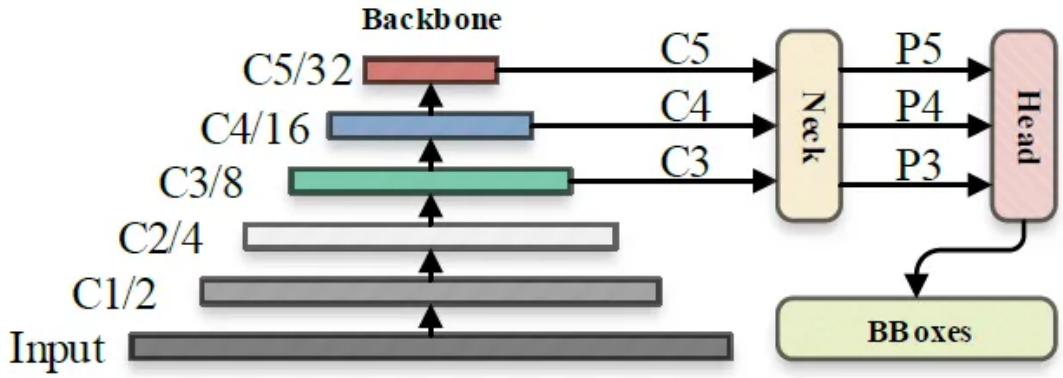


Figure 1. The default inference flowchart of YOLOv5.

The architecture of YOLOv5x utilizes a backbone for feature extraction, a neck that processes these features across different scales, and a head that predicts the bounding boxes and class probabilities. This configuration allows the model to detect objects at various scales effectively, which is essential for handling diverse sizes and aspects of objects within images or video frames.

Detection Mechanism:

YOLOv5x processes images by dividing them into a grid, where each grid cell is responsible for predicting bounding boxes and the likelihood of object classes within those boxes. Each bounding box prediction includes:

- Coordinates (x_center, y_center, width, height) defining the box's location and size.
- Objectness score that signifies the confidence level of the detection.
- Class probabilities indicating the likelihood of each class being present in the bounding box.

Training and Performance:

The training of YOLOv5x involves using a vast dataset with annotated images to teach the model how to accurately predict the presence and classes of objects. It employs various data augmentation techniques to enhance the model's ability to generalize across different environments and conditions. The performance of YOLOv5x is often evaluated using metrics like precision, recall, and the Intersection over Union (IoU) to assess the accuracy and reliability of its predictions.

Advantages in Video Analysis:

In the context of sports video analysis, such as in football games, YOLOv5x provides several advantages:

- Real-time processing capability allows for the analysis of video streams without significant delays.
- High accuracy in detection ensures that players and objects like the ball are consistently and reliably identified, even in complex dynamic scenes.
- Ability to handle variations in player appearances, environmental conditions, and camera angles due to the robustness of the model.

Key Features and Formulas of YOLOv5x:

The YOLOv5x model operates on the principle of dividing the image into a grid and predicting bounding boxes and class probabilities for each grid cell. The predictions involve several components:

Bounding Box Prediction: Each cell predicts bounding boxes through offsets from anchor boxes, with each box characterized by four coordinates (x, center, y center, width, and height), and a confidence score that indicates the likelihood of an object being present.

Class Prediction: Alongside the box coordinates, the network also predicts the class probabilities indicating the likelihood of the detected object belonging to a specific class.

The model utilizes the intersection over union (IoU) as a primary metric, calculated as:

$$\text{IoU} = \frac{\text{Area of Overlap between Predicted and Ground Truth Boxes}}{\text{Area of Union between Predicted and Ground Truth Boxes}}$$

This metric is crucial for training the model, as it directly influences how well the model learns to place accurate bounding boxes around the objects of interest.

Object Detection and Tracking, Team Assignment and Ball Possession Analysis using a Custom Trained Yolov5x model

Custom YOLOv5x Model Training

Dataset Utilization:

I utilized the Roboflow YOLOv5 Football-Player-Detection Image Dataset, a specialized dataset designed for the purpose of detecting football players within diverse game scenarios. This dataset contains images annotated to identify players in various poses and actions, captured under different game conditions and angles. It provides a robust foundation for training an object detection model that needs to recognize and differentiate between players in a complex, dynamic environment.

Model Selection and Training:

For the object detection task, the YOLOv5x variant of the YOLO (You Only Look Once) architecture was chosen. YOLOv5x is known for its effectiveness in providing a good balance between speed and accuracy, making it suitable for real-time video analysis applications where both factors are critical. The choice of YOLOv5x was driven by its advanced capabilities in handling high-resolution inputs and its efficiency in detecting small to medium-sized objects, such as footballs and distant players.

I trained the Yolo 5x model using the Roboflow's Football Detection dataset with the number of epochs set to be 100.

Model Evaluation and Selection:

The model weights were saved periodically during training, with particular attention given to the 'best' and 'last' weights. I further chose the best weights for the annotation tasks. These weights provide the most reliable detection while minimizing false positives and false negatives, essential for accurate tracking and analysis in subsequent stages of the project.

Modules:

1. Video Processing:

The process starts by loading and decomposing a video into individual frames. These frames are subjected to the custom YOLOv5x model to detect players and the ball. Each detected object is then tracked over time across the video frames, ensuring their movement is followed accurately.

2. Object Tracking:

This module focuses on maintaining the identity and trajectory of each detected object throughout the video. It includes handling occlusions or when objects go off-screen, employing

strategies to predict and interpolate positions, thus maintaining continuous tracking especially for the ball in this case.

3. Team and Player Analysis:

Initial frames are analyzed to assign team colors to players using a K-Means clustering algorithm on detected jersey colors. This assignment is crucial for distinguishing between teams throughout the video. The system continuously updates these assignments to account for any visual changes in the players' appearances due to external factors.

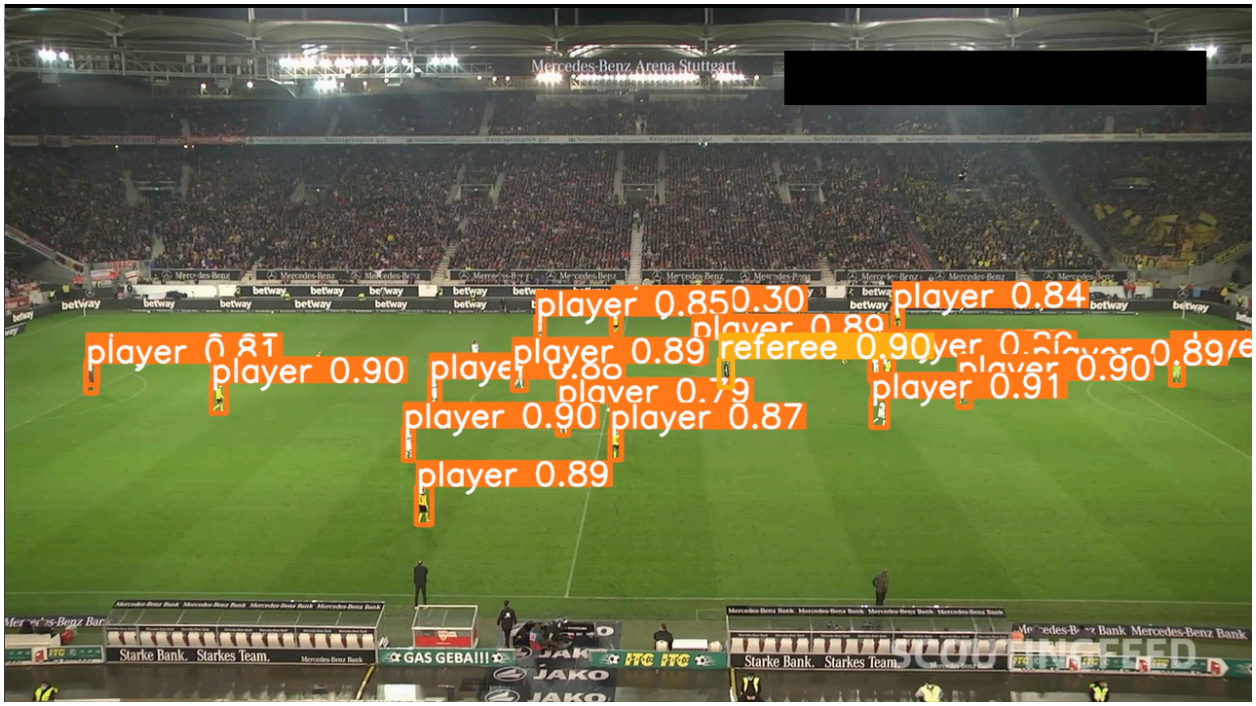
4. Ball Possession Analysis:

The system calculates distances between players and the ball for each frame. It assigns possession to the player closest to the ball, integrating spatial analysis to dynamically understand control of the game, which is pivotal for tactical evaluations.

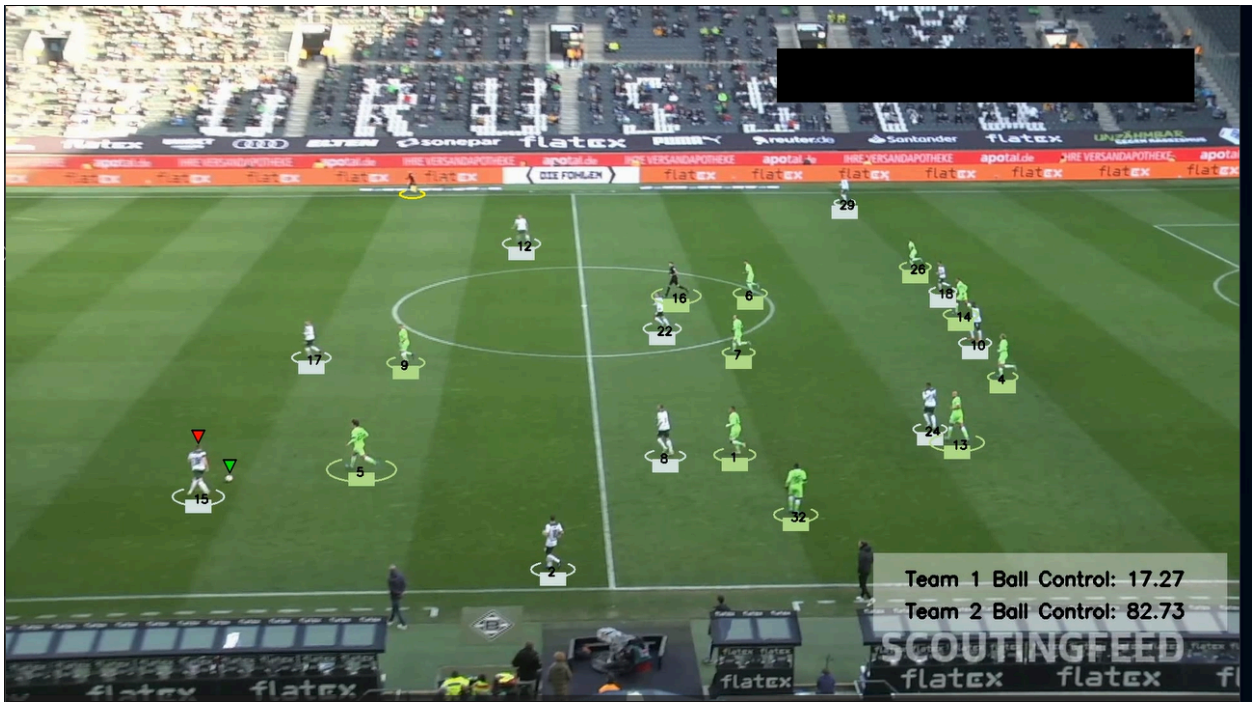
5. Annotation and Output:

Each frame is annotated with detailed tracking information, including visual indicators for player paths, ball position, team colors, and possession status. This rich annotation provides a layered understanding of the game dynamics, compiled back into a reconstructed video for review and analysis.

Input Video



Results applying Custom Yolov5x



Results applying Custom YOLOv5x with all module implementations

Frame Segmentation Using ResNet

Contributor - Shanun

In this section I along with my teammates worked on classifying and predicting images from the videos into different classes

The main training dataset is the following

	video_id	time	event	event_attributes
0	1606b0e6_0	200.265822	start	NaN
1	1606b0e6_0	201.150000	challenge	['ball_action_forced']
2	1606b0e6_0	202.765822	end	NaN
3	1606b0e6_0	210.124111	start	NaN
4	1606b0e6_0	210.870000	challenge	['opponent_dispossessed']
5	1606b0e6_0	212.624111	end	NaN
6	1606b0e6_0	217.850213	start	NaN
7	1606b0e6_0	219.230000	throwin	['pass']
8	1606b0e6_0	220.350213	end	NaN
9	1606b0e6_0	223.930850	start	NaN
10	1606b0e6_0	224.430000	play	['pass', 'openplay']
11	1606b0e6_0	226.430850	end	NaN
12	1606b0e6_0	228.955367	start	NaN
13	1606b0e6_0	229.390000	play	['pass', 'openplay']
14	1606b0e6_0	231.455367	end	NaN
15	1606b0e6_0	236.248227	start	NaN
16	1606b0e6_0	236.710000	play	['pass', 'openplay']
17	1606b0e6_0	239.350000	play	['pass', 'openplay']
18	1606b0e6_0	240.401851	end	NaN
19	1606b0e6_0	241.635933	start	NaN

Video_id - Identifies which video the event occurred in

Event - The type of event occurrence, one of challenge, play or throwin, Also present are labels *start* and *end* indicating the scoring intervals of the video.

Event Attributes - Additional descriptive attributes of the event

Time - The time in seconds, the event occurred within the video

```
err_tol = {
    'challenge': [0.30, 0.40, 0.50, 0.60, 0.70],
    'play': [0.15, 0.20, 0.25, 0.30, 0.35],
    'throwin': [0.15, 0.20, 0.25, 0.30, 0.35]
}
```

These error tolerances are essential part of the modeling process

These error tolerances represent the maximum allowable deviation between predicted event timestamps and ground truth event timestamps for each event class. These are used during the evaluation process to determine whether a predicted event is considered a True Positive (TP), False Positive(FP) or False Negative

Here’s what each part signifies

1. Challenge - [0.30, 0.40, 0.50, 0.60, 0.70] : For this event class predicted event timestamps are allowed to deviate from the ground truth event timestamps by up to the above values

2. Play: These are the error tolerances for events classified as "Play".

   [0.15, 0.20, 0.25, 0.30, 0.35]: For this event class, predicted event timestamps are allowed to deviate from the ground-truth event timestamps by up to 0.15, 0.20, 0.25, 0.30, or 0.35 seconds.

3. Throw-In: These are the error tolerances for events classified as "Throw-In".

   [0.15, 0.20, 0.25, 0.30, 0.35]: For this event class, predicted event timestamps are allowed to deviate from the ground-truth event timestamps by up to 0.15, 0.20, 0.25, 0.30, or 0.35 seconds.

Preprocessing

In the base Preprocessing i created more events and calculated frames present in each video

For each video if the event corresponding to that video is present is found in the error tolerance dictionary. It calculates the tolerance value based on the event type. It then generates additional events before or after the original event with the timestamps adjusted by **‘tol**’ and ‘**tol * 2**’

I also calculated the frame rate and total number of frames using the following function from opencv library

```
cap = cv2.VideoCapture("/home/ubuntu/bundesliga/src/Data/train/3c993bd2_0.mp4")
fps = cap.get(cv2.CAP_PROP_FPS)
print("fps:", fps)
df["frame"] = df["time"] * fps
```

Preprocessed Training dataset after the above step

	video_id	time	event	event_attributes	frame
0	1606b0e6_0	200.265822	start	NaN	5006.645548
1	1606b0e6_0	200.750000	pre_challenge	['ball_action_forced']	5018.750000
2	1606b0e6_0	200.950000	start_challenge	['ball_action_forced']	5023.750000
3	1606b0e6_0	201.350000	end_challenge	['ball_action_forced']	5033.750000
4	1606b0e6_0	201.550000	post_challenge	['ball_action_forced']	5038.750000
5	1606b0e6_0	202.765822	end	NaN	5069.145548
6	1606b0e6_0	210.124111	start	NaN	5253.102780
7	1606b0e6_0	210.470000	pre_challenge	['opponent_dispossessed']	5261.750000
8	1606b0e6_0	210.670000	start_challenge	['opponent_dispossessed']	5266.750000
9	1606b0e6_0	211.070000	end_challenge	['opponent_dispossessed']	5276.750000
10	1606b0e6_0	211.270000	post_challenge	['opponent_dispossessed']	5281.750000
11	1606b0e6_0	212.624111	end	NaN	5315.602780
12	1606b0e6_0	217.850213	start	NaN	5446.255329
13	1606b0e6_0	219.030000	pre_throwin	['pass']	5475.750000
14	1606b0e6_0	219.130000	start_throwin	['pass']	5478.250000
15	1606b0e6_0	219.330000	end_throwin	['pass']	5483.250000
16	1606b0e6_0	219.430000	post_throwin	['pass']	5485.750000
17	1606b0e6_0	220.350213	end	NaN	5508.755329
18	1606b0e6_0	223.930850	start	NaN	5598.271259

This dataframe is finally saved into a csv and will be used in further modeling

Preprocessing part 2 :

Now after creating a new training dataset I performed some basic steps like splitting the videos into training and validation steps

For low computational purpose and as the data corresponding to each video is huge(~80k frames per video)

I analyzed only 1 training and 1 validation video for generating baseline results

```
video_id_split = {

    'val': ['35bd9041_0'],

    'train': ["1606b0e6_0"]
}
```

```
IMG_SOURCE = "img_train" if not VAL else "img_val"
df_video = df[df.video_id == video_id]

print(video_id, df_video.shape)

# crr_statu => background, play, challenge, throwin
arr = df_video[['frame', 'event']].values
```

In this step i iterated through the rows of df_video and extracted frame and event information

- The event information is split into **‘start**’, **‘end**’ and **‘class**’ information
- Based on the event type, a **data** list is populated with dictionaries containing **start**, end and **class** information
- Creating a new dataframe ***df2* -** Here i iterated over the through the **data** list processing each event and frame
- For each frame we check if the corresponding image file exists
- If the image file exists, it appends the frame information along with the class and video id to the **out** list

BackGround Frame Generation

- After processing the events we check for background frames at regular intervals (BACK_INTERVAL or BACK_INTERVAL_VAL)

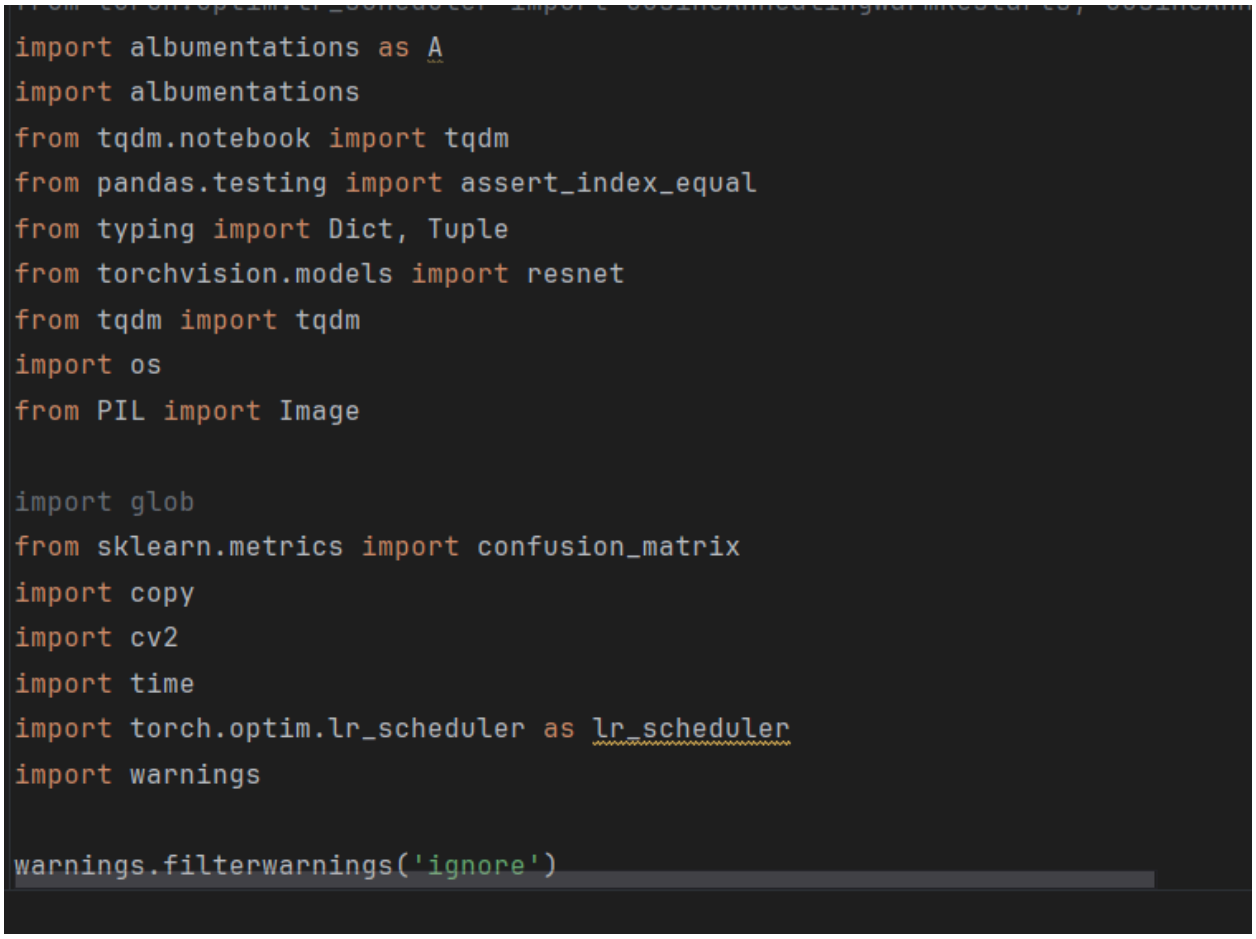
Training and Validation Datasets After PreProcessing

	frame	cls	video
0	4400	play	35bd9041_0
1	4400	post_play	35bd9041_0
2	9500	pre_play	35bd9041_0
3	14000	play	35bd9041_0
4	14300	post_play	35bd9041_0
5	14900	post_challenge	35bd9041_0
6	16300	pre_play	35bd9041_0
7	19000	post_play	35bd9041_0
8	20100	challenge	35bd9041_0
9	21200	play	35bd9041_0
10	23100	pre_play	35bd9041_0
11	25900	post_play	35bd9041_0
12	29800	play	35bd9041_0
13	30500	play	35bd9041_0
14	30600	pre_play	35bd9041_0
15	31600	challenge	35bd9041_0

(training split snippet)

Model Training

```python
import pandas as pd
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.optim import lr_scheduler
from torch.utils.data import DataLoader, Dataset
from torch.utils.data.sampler import SubsetRandomSampler, RandomSampler, SequentialSampler
from torch.optim.lr_scheduler import CosineAnnealingWarmRestarts, CosineAnnealingLR, ReduceLROnPlateau
import albumentations as A
```



```python
import albumentations as A
import albumentations
from tqdm.notebook import tqdm
from pandas.testing import assert_index_equal
from typing import Dict, Tuple
from torchvision.models import resnet
from tqdm import tqdm
import os
from PIL import Image

import glob
from sklearn.metrics import confusion_matrix
import copy
import cv2
import time
import torch.optim.lr_scheduler as lr_scheduler
import warnings

warnings.filterwarnings('ignore')
```

Parts of Model training Script

Custom Dataset Class: The dflDataset class is a custom dataset class derived from the torch.utils.data.Dataset class. It's used to load and preprocess the dataset for training and validation. It reads image files and their corresponding labels from the dataset and applies transformations such as resizing, augmentation, and mixup.

Data Transformations: The transforms_train and transforms_val define the transformations to be applied to the training and validation datasets, respectively. These transformations include random rotations, flips, brightness/contrast adjustments, resizing, and more. These transformations help in data augmentation, which is essential for improving the model's generalization ability.

Training and Validation Data Loading: Training and validation datasets are loaded using the custom dataset class and the defined transformations.

Model Definition: The net class defines the neural network architecture. In this case, a ResNet-18 model pretrained on ImageNet is used as the backbone, and the fully connected layer is replaced with a new layer with the desired number of output classes.

Loss Function: The nn.CrossEntropyLoss() function is used as the loss function. Cross-entropy loss is commonly used for classification problems with multiple classes.

Mixup: The mixup_criterion function implements the mixup loss, which is a data augmentation technique that blends pairs of input images and their labels to generate new training examples. This helps in improving the robustness of the model.

Training Loop: The training loop iterates over the training dataset for a specified number of epochs. In each epoch, it trains the model using batches of data, computes the loss, and updates the model parameters using the optimizer.

Validation Loop: After each epoch of training, the model is evaluated on the validation dataset to monitor its performance. The validation loss, accuracy, and other metrics are calculated to assess the model's performance.

Model Saving: After training, the model parameters are saved to a file for future use or further evaluation.

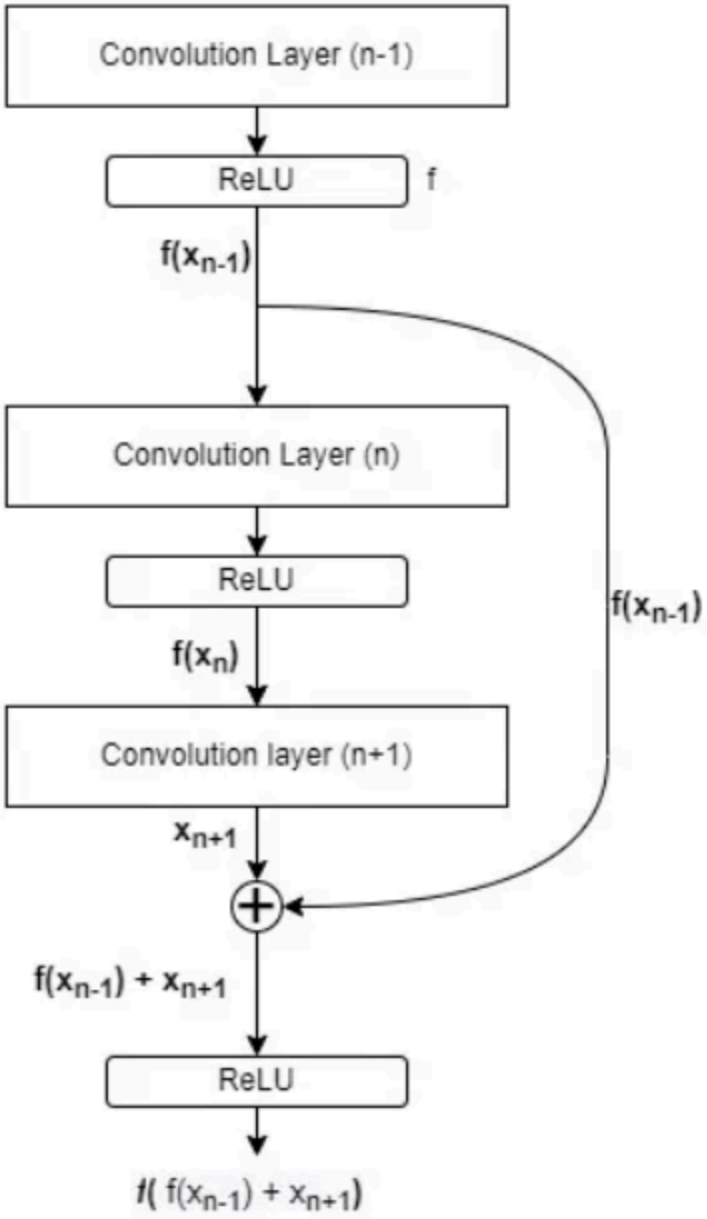
Employed Model - ResNet

We think that the deeper the neural network, the better the performance, but when researchers experimented with deeper neural nets,

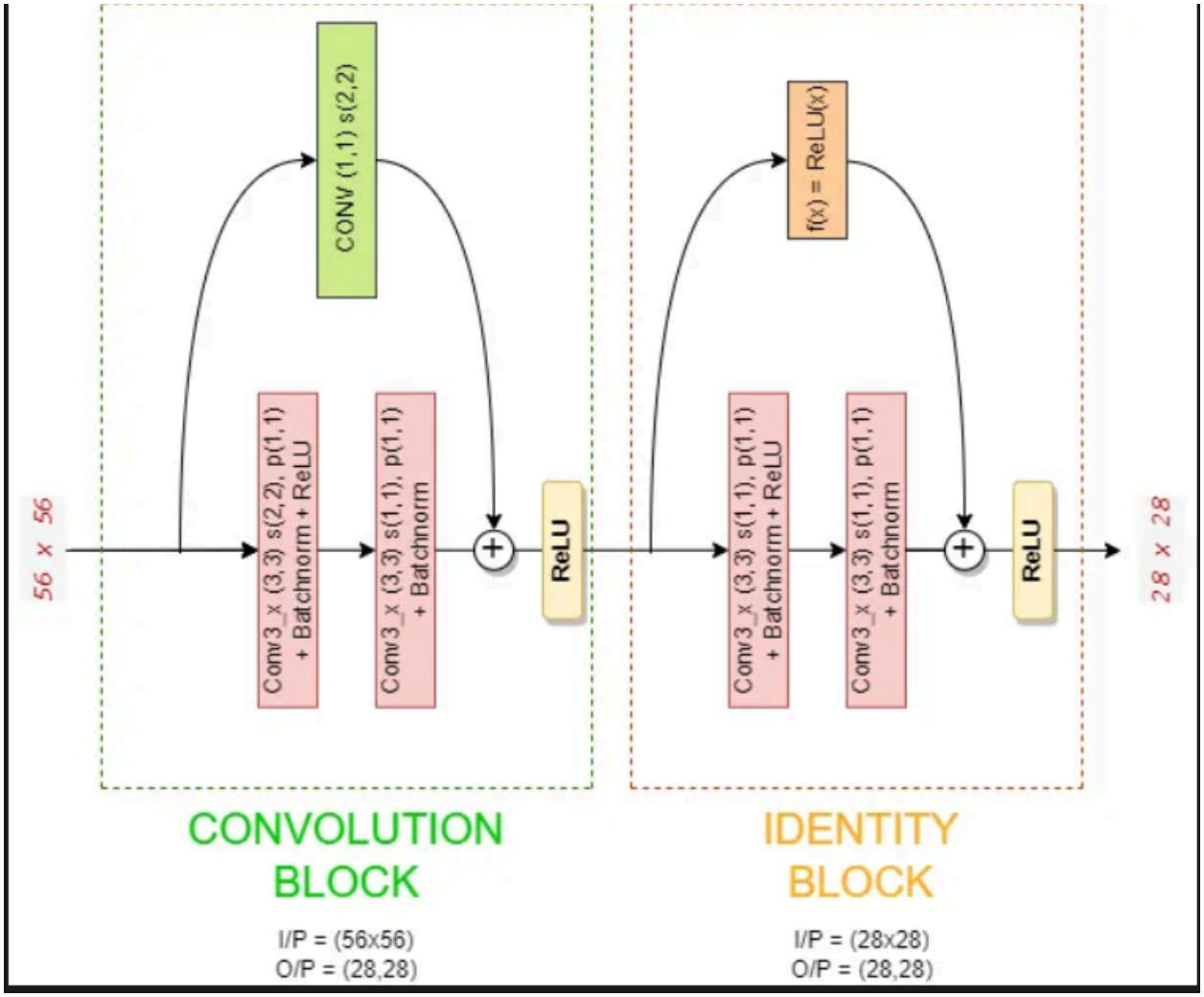
It was found that adding more layers to a deep network does not always add up to its performance but rather decreases it, which was due to vanishing gradients in very deep neural networks.

As a result, it was proposed that adding more layers to a deep neural network should either increase its performance or let it stay the same, but it should never decrease the performance. In order to achieve this, they came up with the concept of Skip connections/Residual connections, by which we can avoid loss of information flow.

A Skip/Residual connection takes the activations from an $(n-1)^{th}$ convolution layer and adds it to the convolution output of $(n+1)^{th}$ layer and then applies ReLU on this sum, thus Skipping the n^{th} layer.



(***RESNET18 ARCHITECTURE***)



Convolution Block

The input to the Conv3_x block is an image of shape (56x56) with 64 channels, The first Convolution layer transforms the image to (28x28) image with 128 channels using a 3x3 kernel and 2x2 stride with 1x1 padding, and applies ReLU on it. The second convolution layer takes this

image as input and outputs the image with the same shape (28x28x128). Now in order to apply residual/skip connection, we have to add the output of Conv2_x block which is (56x56x64) with the output of the second convolution which is (28x28x128), in order to do this we need to convert Conv2_x output to (28x28x128), This is done by applying another convolution with 1x1 filter and 2x2 stride, with input channels as 64 and output channels as 128.

Simply put, the Convolution block is responsible for converting the output from one block using convolution operation so that it could be effectively used to add with the output of another convolution block.

After the addition is done, Activation (ReLU) is applied to its output and is sent to the Identity block.

Identity Block

The input and output of an identity block are the same, thus to apply Residual/Skip connection we do not need to apply any transformation to the output of Convolution Block. Therefore to apply Skip/Residual connection all we need to do is to add the output of the Convolution block to the output of the 4th convolution layer in Conv3_x block, and then apply ReLU on it.

We now understand that whenever there is a need to adjust output to make it possible to apply a Residual connection, we would need a Convolution block, and whenever the input and output are the same we need an Identity block.

Model Results

The following results are coming from running a baseline resnet model trained on 1 video and validation is done on 1 video. I used less data to get baseline results, further improvements can be made to get better results

```
for class 0: accuracy: 99.94106739877247
for class 1: accuracy: 0.0
for class 2: accuracy: 0.23674242424242425
for class 3: accuracy: 0.0
for class 4: accuracy: 0.0
for class 5: accuracy: 0.0
for class 6: accuracy: 0.0
for class 7: accuracy: 0.0
for class 8: accuracy: 0.0
for class 9: accuracy: 0.0
```

```
for class 9: accuracy: 0.0
val df
```

	background	challenge	play	throwin	video_name	video_id	frame_id	score	event	time
0	0.820503	0.581450	0.797371	0.106265	/home/ubuntu/bundesliga/src/work/img_val/35bd9...	35bd9041_0	13080	0.584110	challenge	523.20
1	0.814149	0.577047	0.811599	0.096452	/home/ubuntu/bundesliga/src/work/img_val/35bd9...	35bd9041_0	12914	0.575648	challenge	516.56
2	0.814517	0.563060	0.797267	0.104495	/home/ubuntu/bundesliga/src/work/img_val/35bd9...	35bd9041_0	13058	0.564158	challenge	522.32
3	0.818563	0.563868	0.785508	0.114833	/home/ubuntu/bundesliga/src/work/img_val/35bd9...	35bd9041_0	12885	0.564148	challenge	515.40
4	0.804557	0.570516	0.789408	0.113614	/home/ubuntu/bundesliga/src/work/img_val/35bd9...	35bd9041_0	13005	0.563682	challenge	520.20
...	...	...	...	...	...	...	...	...	...	...
10513	0.889037	0.442059	0.645832	0.093430	/home/ubuntu/bundesliga/src/work/img_val/35bd9...	35bd9041_0	41411	0.645363	play	1656.44
10514	0.899064	0.431011	0.645640	0.089049	/home/ubuntu/bundesliga/src/work/img_val/35bd9...	35bd9041_0	40943	0.645075	play	1637.72
10515	0.884217	0.445958	0.644851	0.096166	/home/ubuntu/bundesliga/src/work/img_val/35bd9...	35bd9041_0	41449	0.644616	play	1657.96
10516	0.872836	0.453142	0.644585	0.102073	/home/ubuntu/bundesliga/src/work/img_val/35bd9...	35bd9041_0	41501	0.644017	play	1660.04
10517	0.867171	0.455034	0.643190	0.110127	/home/ubuntu/bundesliga/src/work/img_val/35bd9...	35bd9041_0	41525	0.642580	play	1661.00

```
recall:
event      tolerance
challenge  0.30         0.289348
           0.40         0.457758
           0.50         0.503002
           0.60         0.515297
           0.70         0.528527
play       0.15         0.173936
           0.20         0.251548
           0.25         0.335316
           0.30         0.362090
           0.35         0.380065
throwin    0.15         0.371849
           0.20         0.371849
           0.25         0.371849
           0.30         0.371849
           0.35         0.371849
dtype: float64
precision:
event      tolerance
challenge  0.30         0.023543
           0.40         0.040130
           0.50         0.043148
           0.60         0.044246
           0.70         0.048057
play       0.15         0.100771
           0.20         0.147229
           0.25         0.197624
           0.30         0.212479
           0.35         0.228693
throwin    0.15         0.016338
           0.20         0.016338
           0.25         0.016338
           0.30         0.016338
           0.35         0.016338
dtype: float64
```

```
event       tolerance
challenge   0.30         0.014685
            0.40         0.035655
            0.50         0.041684
            0.60         0.044654
            0.70         0.050578
play        0.15         0.036090
            0.20         0.073157
            0.25         0.125770
            0.30         0.148746
            0.35         0.165365
throwin     0.15         0.088323
            0.20         0.088323
            0.25         0.088323
            0.30         0.088323
            0.35         0.088323
dtype: float64
```

```
dtype: float64
val loss
0.9736581
accuracy
92.2765576272311
score
0.078533406372802
predictions
[[0.81160337 0.47780192 0.7554106  ... 0.25050387 0.6159804  0.13253576]
 [0.8151151  0.47644234 0.7564814  ... 0.24715114 0.62020147 0.13122953]
 [0.8130537  0.47599542 0.75159687 ... 0.24963681 0.6123869  0.13461837]
 ...
 [0.8693303  0.43677482 0.6709343  ... 0.17698604 0.45978913 0.1411031 ]
 [0.8649221  0.44020468 0.6727527  ... 0.18655503 0.46310964 0.14454508]
 [0.8687233  0.4370755  0.672178   ... 0.18456718 0.4613681  0.14489888]]
ubuntu@ip-10-1-3-180:~/bundesliga/src$
```

Precision: Precision measures the proportion of true positive detections among all the detections made by the system. In the context of event detection, precision tells us how many of the detected events are actually relevant (true positives) compared to the total number of detected events. A higher precision value indicates fewer false positives, meaning that the system is better at correctly identifying relevant events.

Recall: Recall, also known as sensitivity, measures the proportion of true positive detections among all the actual positive instances in the dataset. In event detection, recall tells us how many of the actual events were correctly identified by the system. A higher recall value indicates that the system is capturing a larger proportion of the true events present in the dataset.

Precision-Recall Curve: The precision-recall curve plots the precision values against corresponding recall values for different thresholds or confidence levels used in the detection process. It provides a graphical representation of how precision and recall vary with changing detection thresholds. Analyzing this curve can help determine the trade-off between precision and recall at different operating points of the system.

Average Precision (AP): Average precision is a summary metric computed from the precision-recall curve. It represents the average precision value across all recall levels or operating points. AP is commonly used to evaluate the overall performance of an event detection system. A higher AP value indicates better overall performance, with higher precision achieved across all recall levels.

Event Detection AP: The event detection AP computed in the code is a specific measure of performance for event detection tasks. It considers the precision-recall trade-off for each event class (e.g., challenge, play, throwin) at different tolerance levels. By analyzing the event detection AP, one can assess how well the system identifies specific events while considering different tolerance thresholds for matching detections with ground truth events.

STREAMLIT APP:

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

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

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

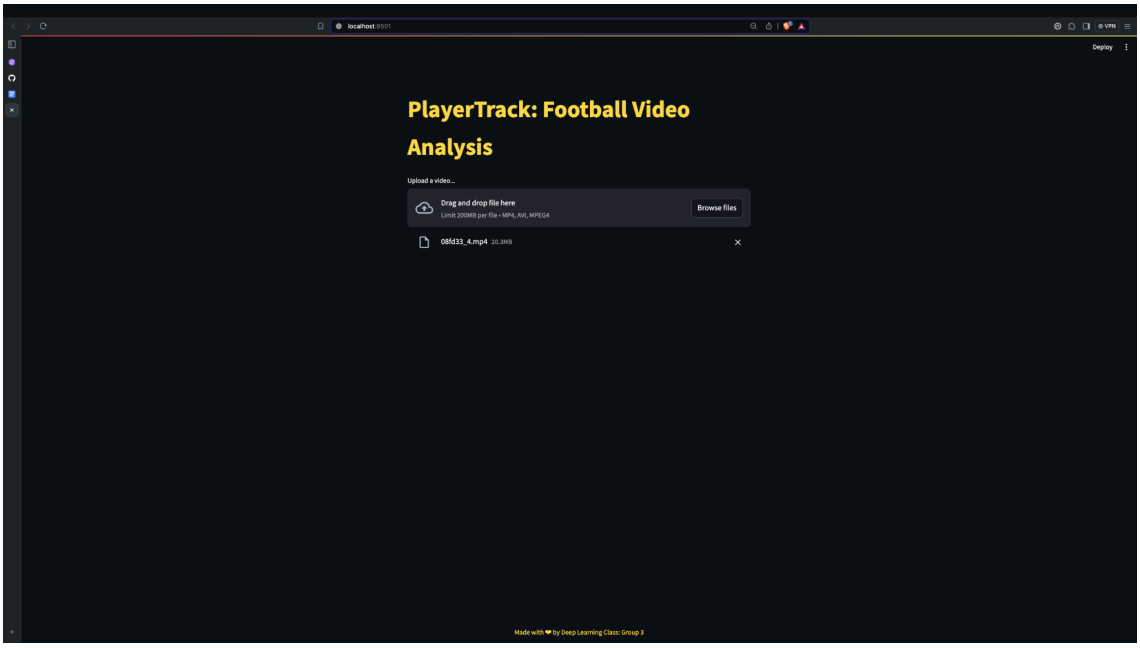
Object Detection: Once a video is uploaded, it is fed into the YOLOv8 model. The model uses the trained weights to perform object detection on each frame of the video. This process involves identifying the location of football players and bounding them with boxes.

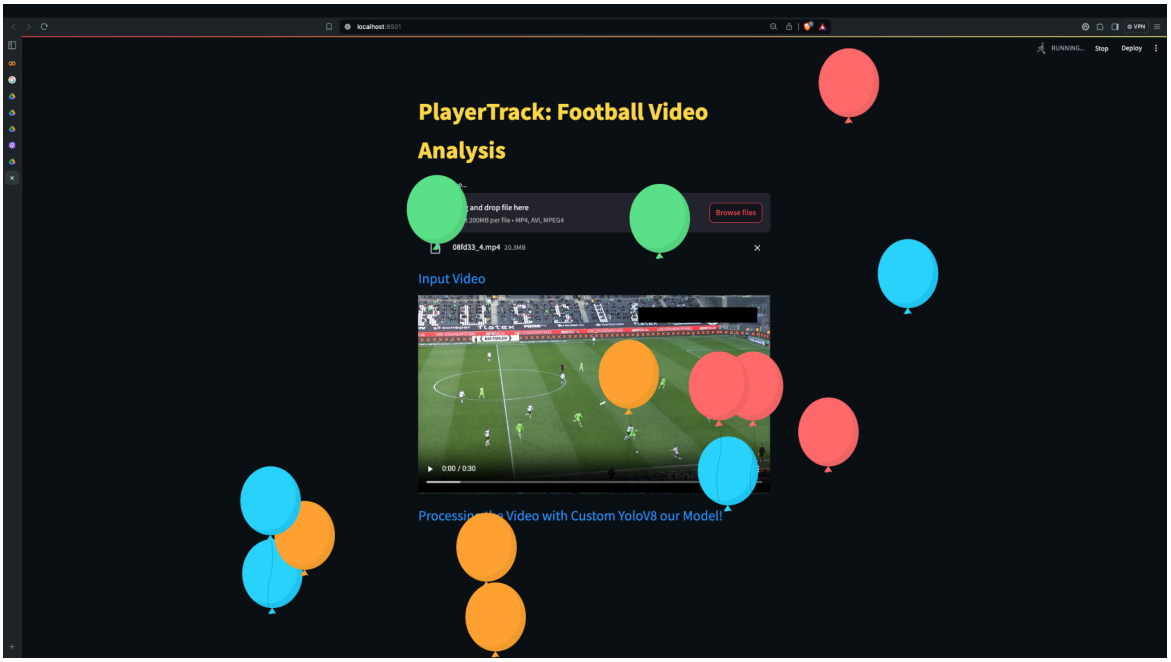
Display Results: After processing, the video with detected objects (players) is displayed on the Streamlit app interface. Users can see the processed video with bounding boxes around players, allowing them to analyze the players' positions and movements throughout the video.

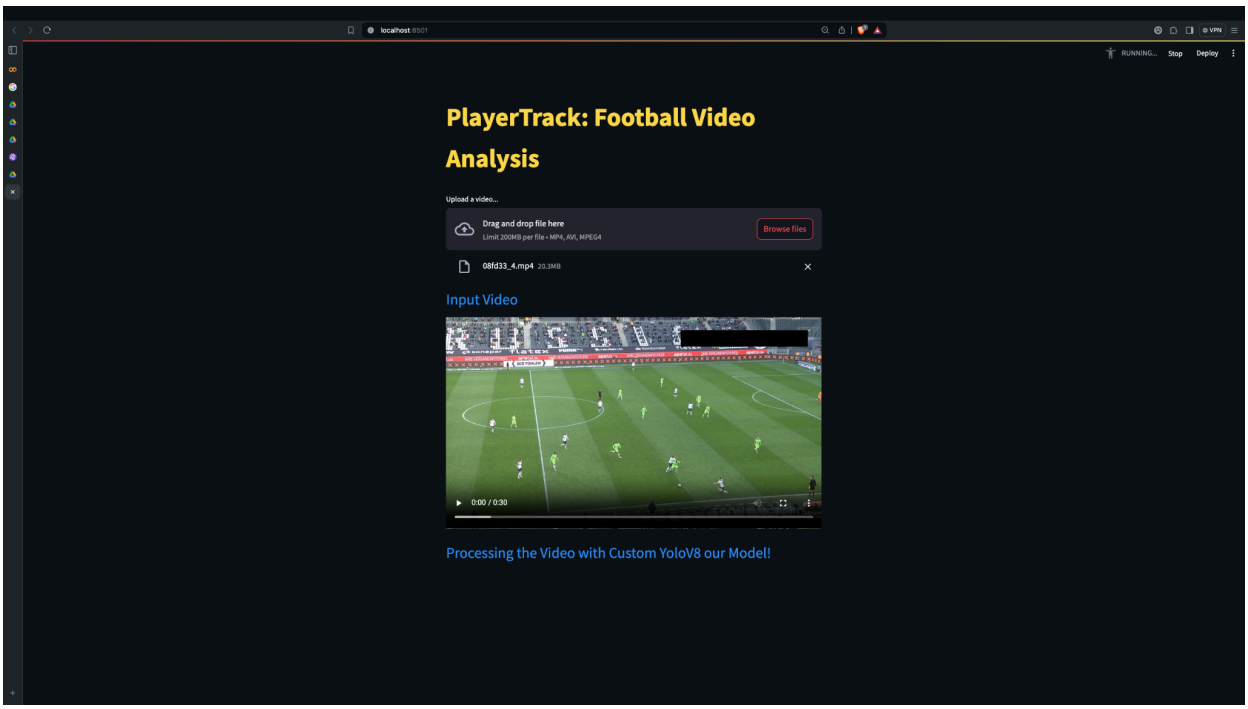
Balloon Animations: When a user uploads a video and the detection process starts, colorful balloon animations appear on the screen. This playful animation serves a dual purpose: it not only visually entertains

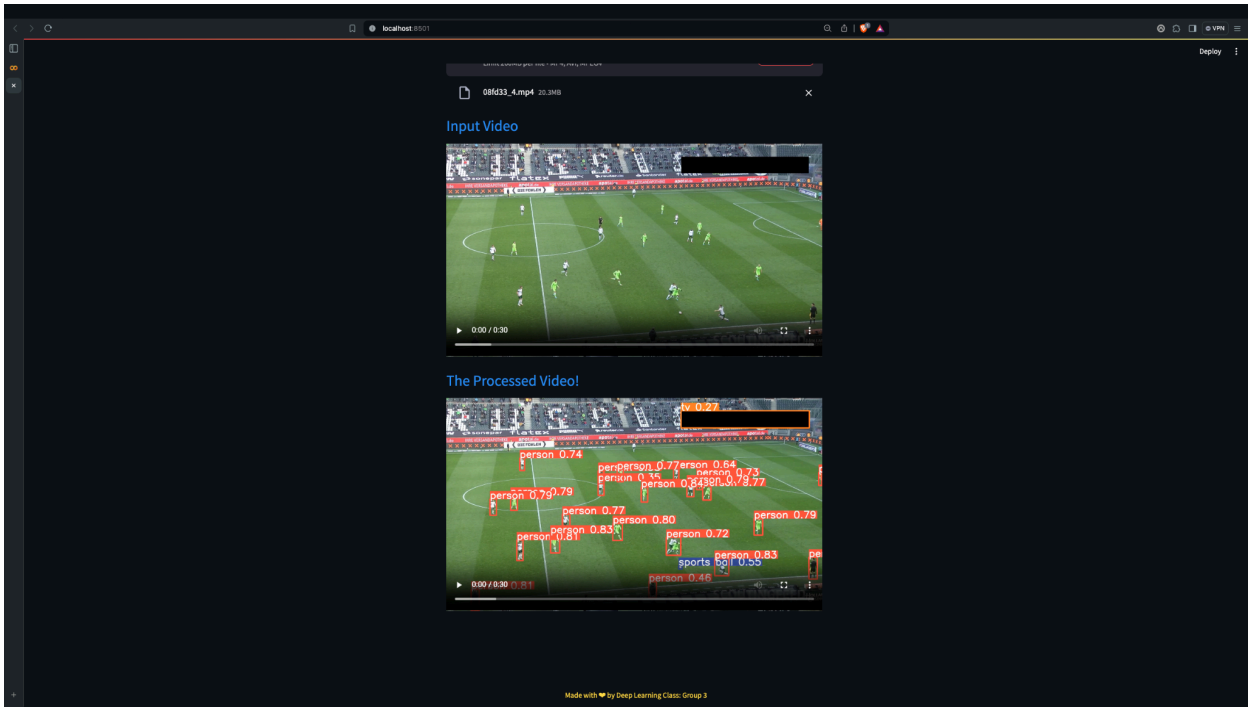
the user but also subtly indicates that the application is actively processing the input. The balloons float across the user interface, creating a light and engaging atmosphere that reduces the perceived waiting time.

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




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


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



Improvements

Feature Engineering:
I would tend to Explore additional features that could better represent event patterns, such as temporal features, spatial features, or contextual information.

Experiment with different feature representations, such as embeddings or higher-level abstractions, to capture more meaningful patterns in the data.

Model Architecture:
I will Consider using more advanced neural network architectures tailored for event detection tasks, such as recurrent neural networks (RNNs), convolutional neural networks (CNNs), or transformer-based models.
Experiment with ensemble methods to combine predictions from multiple models or model variants to improve overall performance.

Data Augmentation:
I will consider augmenting the training data by introducing variations, such as adding noise, applying transformations, or simulating different environmental conditions, to improve the model's robustness to variations in the input data.

Hyperparameter Tuning:
I will conduct rigorous and systematic hyperparameter tuning experiments to optimize model parameters, learning rates, regularization techniques, and other settings to achieve better performance on the validation or test datasets.

Multi-task Learning:
Will Explore multi-task learning approaches where the model simultaneously learns to perform multiple related tasks, such as event detection and event classification, to leverage shared information and improve generalization.

Conclusions:

In this comprehensive project, our team developed and trained advanced object detection models using the YOLOv5, YOLOv8, and ResNet architectures to identify and classify various elements in football game footage, such as players, referees, goalkeepers, and the ball. The primary objective was to enhance game analysis capabilities

by providing real-time, accurate detections during matches.

- Citations

https://www.kaggle.com/code/its7171/dfl-benchmark-inference

https://www.kaggle.com/competitions/dfl-bundesliga-data-shootout/data

https://docs.ultralytics.com/yolov5/