Soccer players Tracking and Team Classification

Abdelrahman Elsayed - Karim Fathy - Youssef Emad

September 2024

1 Introduction

1.1 Problem Statement

The objective of this project is to track various elements in a football game, including:

- Players of the team on the left,
- Players of the team on the right,
- The goalkeeper of each team,
- The referee and both linemen,
- The ball.

These elements are to be annotated with bounding boxes in the video frames.

2 Dataset

The main dataset, SoccerNet tracking, consists of 57 training samples of 30-second videos, which were provided as frames sampled at a rate of 24 frames per second. Each frame has a resolution of 1920×1080 pixels, with labels for the ball, players, referees, left goalkeeper, right goalkeeper, bounding boxes, and IDs.

The secondary dataset used initially was from Roboflow, containing 204 training samples of random video frames from different football matches, employed for player detection. The training set was augmented for each image with the following modifications: horizontal flipping, saturation changes between +25% and -25%, and brightness changes between +20% and -20%. This augmentation increases the amount of training data to 612 training images. The dataset included 38 images for the validation set and 13 images for the test set.

2.1 Baseline Architecture

The baseline architecture is based on ZTrackers, the 6th place winners in the SoccerNet tracking challenge 2023. While no open-source code or paper was available, their solution is summarized in the SoccerNet 2023 challenge results report.

Their approach is described as follows:

> "Our methodology involved combining the YOLOv5 Medium object detection model with the ByteTrack algorithm. We trained the YOLOv5 Medium model on annotated soccer game images to accurately detect players and the ball. The detections obtained from YOLOv5 Medium were then passed to the ByteTrack algorithm. We fine-tuned several parameters, including the tracking confidence threshold, frames buffer, and matching threshold, to optimize tracking accuracy, smoothness, and robustness. Additionally, we fine-tuned the parameters of the Kalman filter to enhance velocity and position estimates. By integrating YOLOv5 Medium, ByteTrack, and the fine-tuned Kalman filter parameters, we developed a robust system capable of providing real-time and accurate player and ball tracking in the soccer video game."

3 Method

Our first model aimed to detect players using a YOLO model and apply the ByteTrack algorithm for player tracking. We utilized the architecture of YOLOv5 for player detection, trained on the Roboflow dataset. Initially, we could not use the Soccernet dataset in Kaggle due to its size, as we were unable to download it in the Kaggle notebook. Therefore, we trained the model for 100 epochs on the Roboflow dataset, achieving an accuracy of 0.9. In the Roboflow small dataset, we assigned each player to a team based on the color of their shirt. We took a crop of the first frame based on the coordinates of the bounding box detection and sampled a point near the center of the top half of the box to detect the shirt color. Each color value was represented as an RGB vector, which we clustered using simple K-means into 2 clusters (representing 2 teams) and assigned each player to their respective team. This approach was the least accurate, as we used only the first frame for classifying players. Some players may have been obscured by an opponent, potentially leading to misclassification, as we might sample the color vector from another player.

The next step was to train a model on the Soccernet tracking dataset, we used the split of 50 videos out of 57 for training and the other 7 video samples for validation which yields the sum of 37510 training examples(video frames) and 5250 validation examples(video frames)

We trained multiple YOLO detection models to find the most promising one for our use case

we trained(yolo v8m-yolo v10n-yolo v10s-yolov10m)

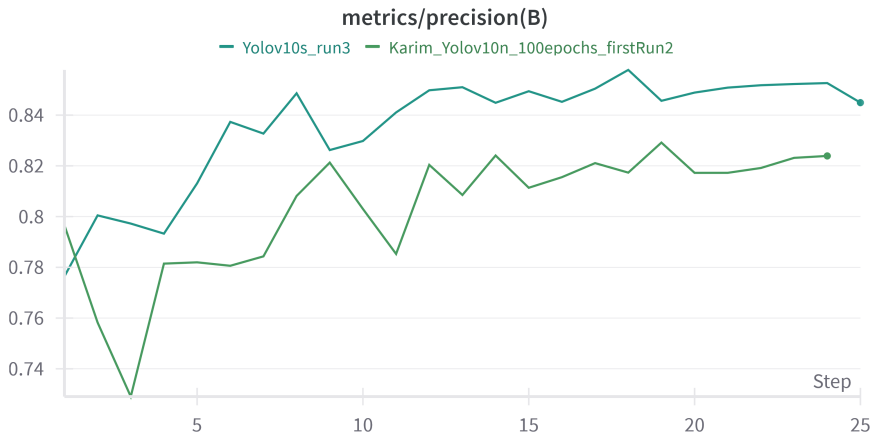

Figure 1: percision values of trained models

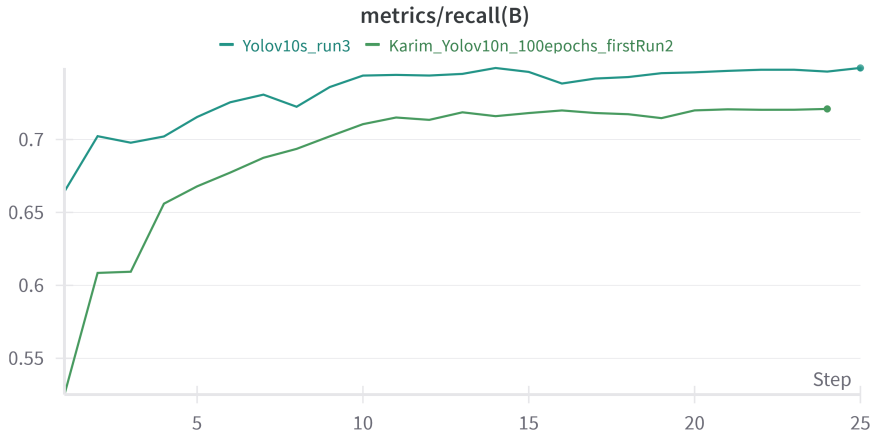

Figure 2: recall values of trained models

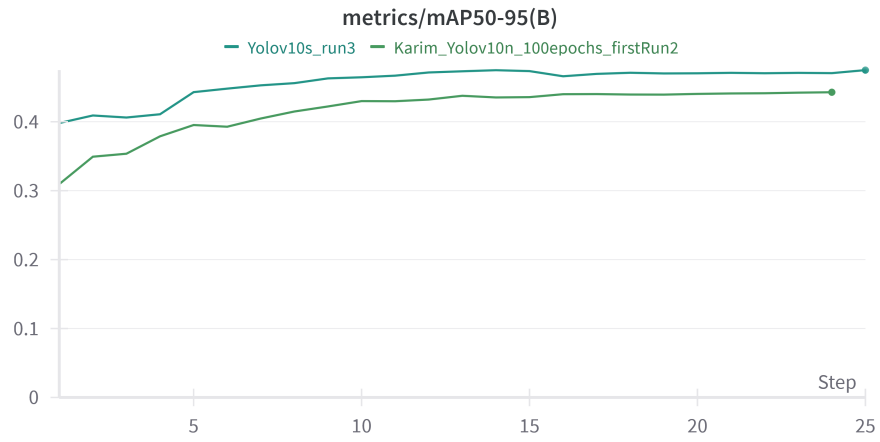

Figure 3: mean average precision of trained models

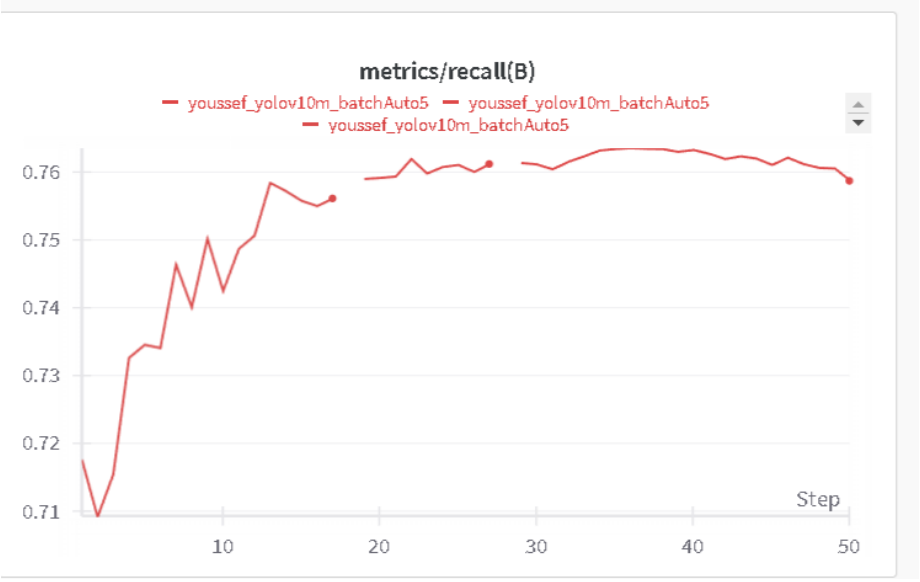

Figure 4: yolov10m recall

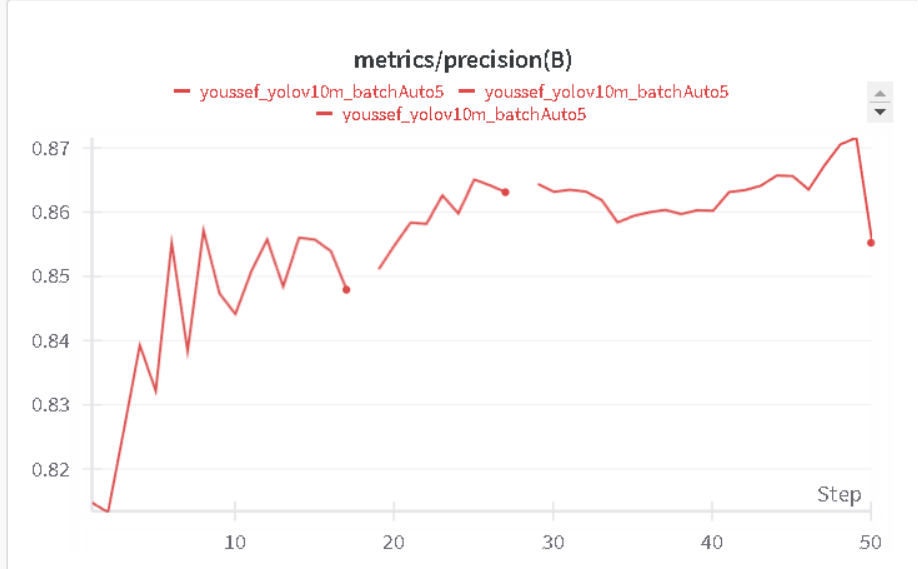

Figure 5: yolov10m precision

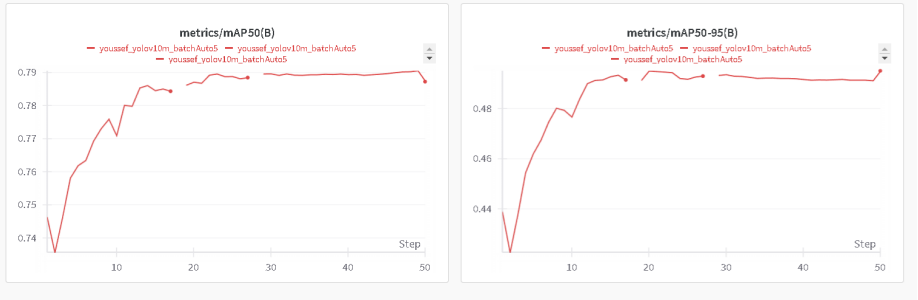

Figure 6: yolov10m mAP

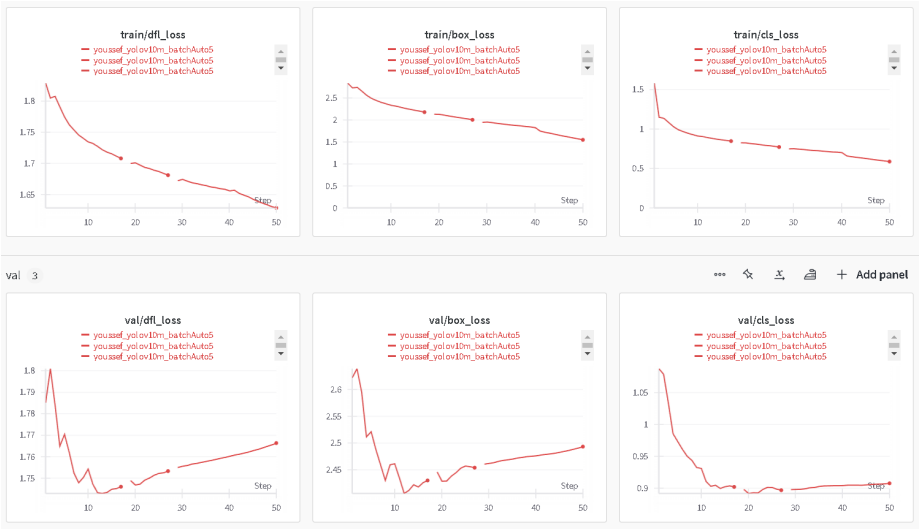

Figure 7: yolov10m losses train and val

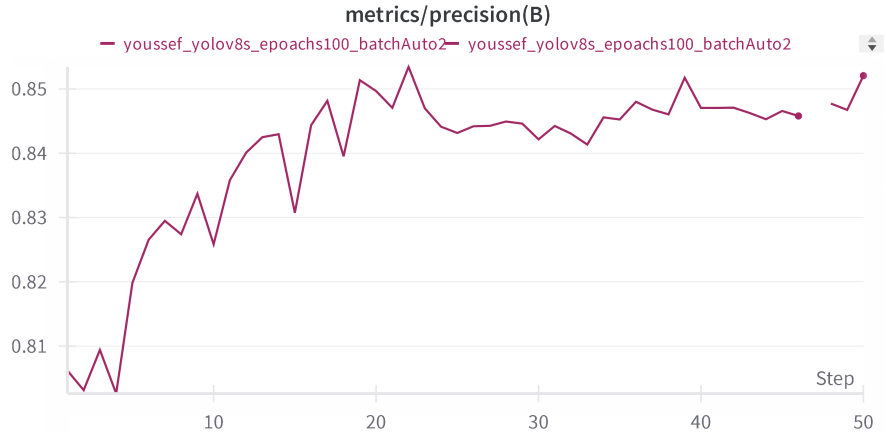

Figure 8: yolov8s pecision

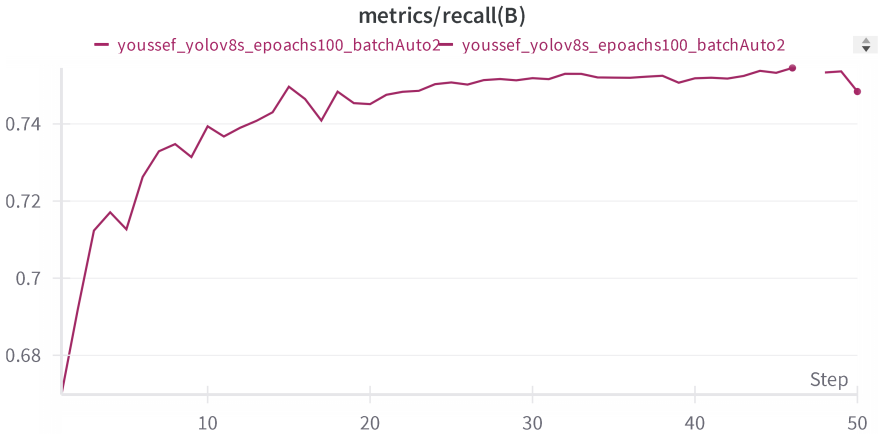

Figure 9: yolov8s recall

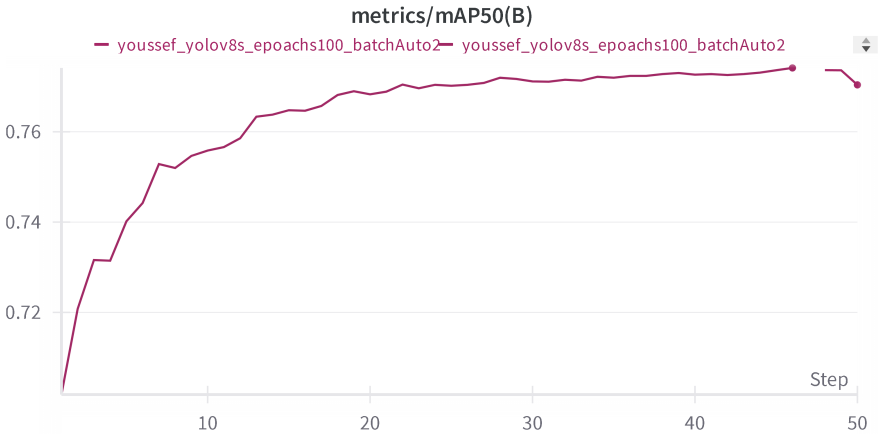

Figure 10: yolov8s mAP50

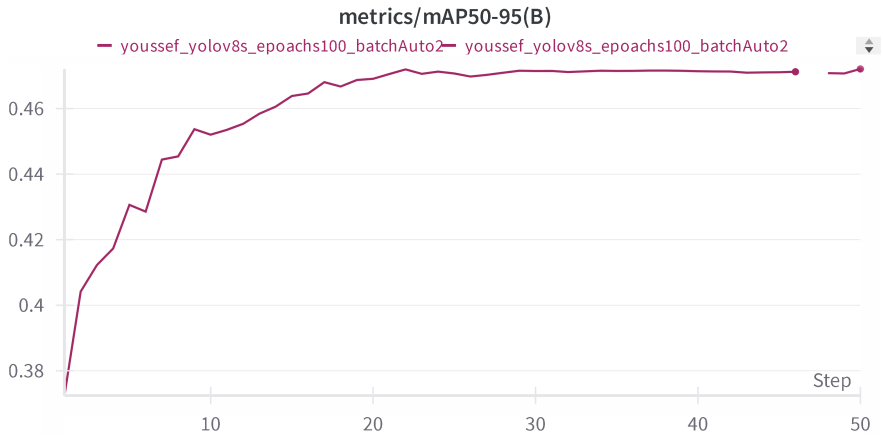

Figure 11: yolov8s mAP50

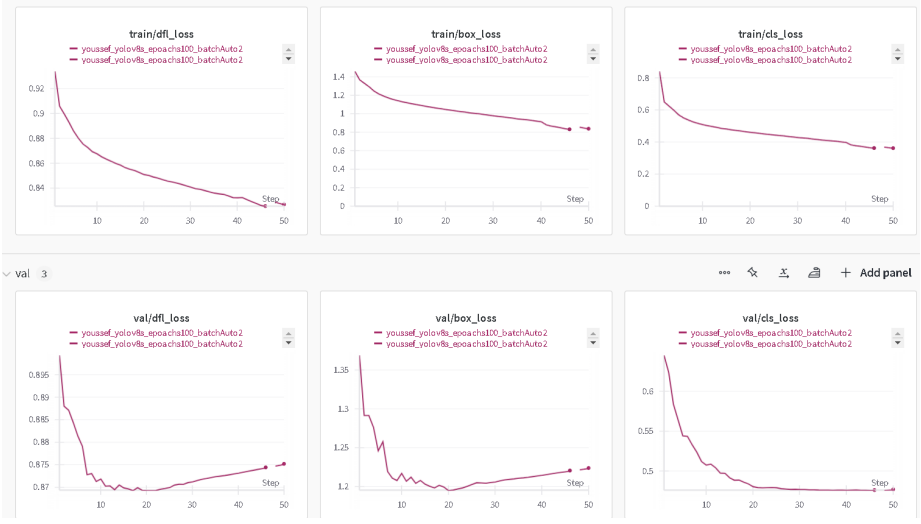

Figure 12: yolov8s losses

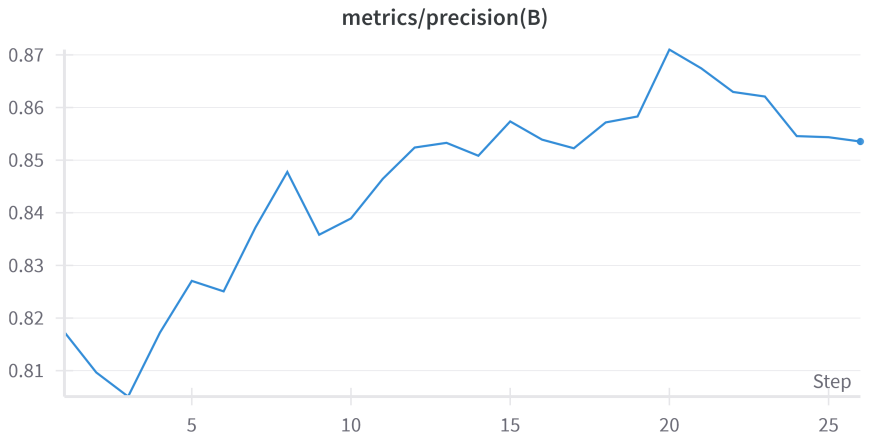

Figure 13: yolov8m precision

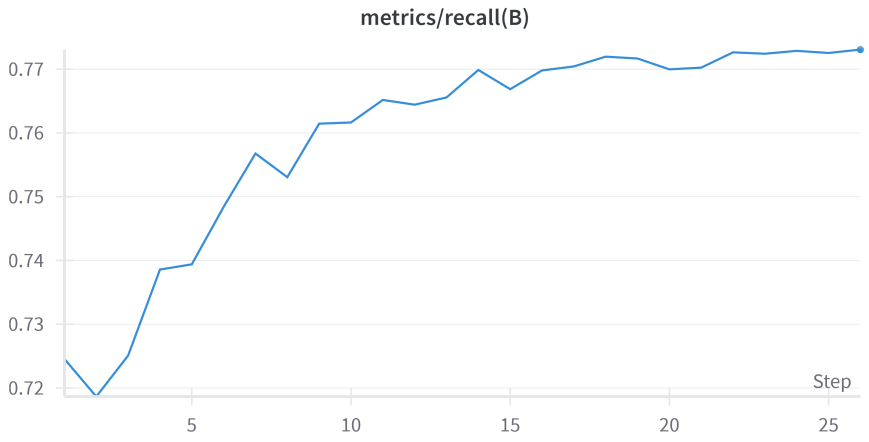

Figure 14: yolov8m recall

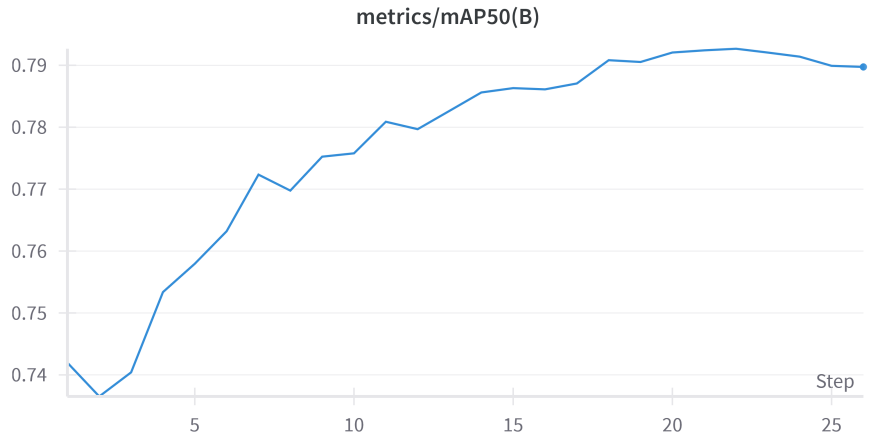

Figure 15: yolov8m mAP50

using the final model due to the less need for computational power we start improving the other aspects of the model for team assignment, we had two problems arising from our first approach (team assignment accuracy was too low due to the simple approach for assignment and the goalkeeper team assignment could not be based on the shirt color as the goalkeeper wears a different colored shirt than his teammates)

for the first problem, we used a different model for assigning players other than sampling a color vector from the shirt of the player by sampling a point or by taking the average color value from a region in the crop, we used the SigLIP vision transformer model to get an embedding vector from each player crop as players wearing the same color tend to have embedding vectors that are close in the provided vector space. the embedding vector size is (1, 768)

We sample a frame from each second in the video and extract player crops from those samples then we get the embedding vector for all the sampled player crops using the pre-trained SigLIP vision transformer. we perform dimensionality reduction on the extracted vectors to 3 dimensions (3-D space) using UMAP to preserve the relations between similar embeddings of the same player in different frames and players wearing the same kit,

we then perform simple Kmeans on the dimensionally reduced data to classify each cluster as a team

during inference, we extract the player crops from each frame and get the embedding vector for each crop using SigLIP we project each vector using UMAP and get the team classification using the nearest neighbor approach, that approach had its toll on performance as it took 30 minutes to analyze one 30 second clip but it had the most accurate results in terms of team classification

the second problem was to classify the goalkeeper to their team correctly. We tried two approaches for this issue. the first was to train the detection model

Figure 16: player crops using bounding box of each player

on classifying the goalkeeper as either a goalkeeper on the left or a goalkeeper on the right then we tried to determine its team based on the positions of the players on their movement but it was not a promising approach as the accuracy of classification has dropped down significantly {insert training results} which deem it not useful

the second approach was to assume the goalkeeper's team based on the position of the keeper and the positions of the players near him using K nearest neighbor with(K=5 and K=7) based on the fact that more defense players will be near the goal of their team than attacking players from the opposite team

that approach had good results and was the most practical one

fixing team assignment time long inference time issue

since using each frame from the video for team assignment was causing the inference to take so much time we started using the tracking IDs of the players to maintain the team classification of each player as we assigned the team to a tracking ID at each nearly 0.3 of a second (10 frames) which drops the number of getting embedding vectors processes and dimension projection processes from 720 to 72

this approach decreases the time of inference from 30 minutes to nearly 10 minutes at most

3.1 Why We Chose SigLIP over CLIP

SigLIP provides better performance in zero-shot detection, making it a more effective choice for this task. Zero-shot detection is a technique in computer vision where a model can detect and classify objects that it has never seen during training. This allows the model to identify and categorize new objects without requiring specific training examples for those particular classes.

Method	Image Encoder		ImageNet-1k				COCO R@1	
	ViT size	# Patches	Validation	v2	ReaL	ObjectNet	I → T	T → I
CLIP	B	196	68.3	61.9	-	55.3	52.4	33.1
OpenCLIP	B	196	70.2	62.3	-	56.0	59.4	42.3
EVA-CLIP	B	196	74.7	67.0	-	62.3	58.7	42.2
SigLIP	B	196	**76.2**	**69.6**	82.8	**70.7**	**64.4**	**47.2**
SigLIP	B	256	76.7	70.0	83.1	71.3	65.1	47.4
SigLIP	B	576	78.6	72.1	84.5	73.8	67.5	49.7
SigLIP	B	1024	**79.2**	**73.0**	**84.9**	**74.7**	**67.6**	**50.4**
CLIP	L	256	75.5	69.0	-	69.9	56.3	36.5
OpenCLIP	L	256	74.0	61.1	-	66.4	62.1	46.1
CLIPA-v2	L	256	79.7	72.8	-	71.1	64.1	46.3
EVA-CLIP	L	256	79.8	72.9	-	75.3	63.7	47.5
SigLIP	L	256	**80.5**	**74.2**	**85.9**	**77.9**	**69.5**	**51.1**
CLIP	L	576	76.6	72.0	-	70.9	57.9	37.1
CLIPA-v2	L	576	80.3	73.5	-	73.1	65.5	47.2
EVA-CLIP	L	576	80.4	73.8	-	78.4	64.1	47.9
SigLIP	L	576	**82.1**	**75.9**	**87.0**	**81.0**	**70.6**	**52.7**
OpenCLIP	G (2B)	256	80.1	73.6	-	73.0	67.3	51.4
CLIPA-v2	H (630M)	576	81.8	75.6	-	77.4	67.2	49.2
EVA-CLIP	E (5B)	256	82.0	75.7	-	79.6	68.8	51.1
SigLIP	SO (400M)	729	**83.2**	**77.2**	**87.5**	**82.9**	**70.2**	**52.0**

Figure 17: SigLIP