DIPARTIMENTO
DI INFORMATICA
SAPIENZA
UNIVERSITÀ DI ROMA

Football Analytics

A Machine Learning model to analyze football matches
AI LAB: COMPUTER VISION AND NLP

Professor:
Pannone Daniele

Project developed by:
Massaroni Flavio
Salinetti Andrea
Scappatura Leonardo
Ugolini Lorenzo

Academic year 2023/2024

Abstract

The purpose of the Football Analytics program is to extract the key statistics of a football match and show them on screen concurrently with the vision of the video.

In order to accomplish this goal, the approach was divided into 4 main tasks:

- *Objects recognition*, to identify the players and the ball in the field.
- *Team recognition*, to be able to assign the detected action to the right team.
- *Environment recognition*, to be aware of where the actions are performed.
- *Action recognition*, to display the summary of how each team performs.

For a more complete experience, a GUI has also been implemented.

Objects Recognition

The hardest part of the project is the classification of all the objects shown on screen. Creating a Neural Network to accomplish this goal was nearly impossible, both for computational limitations and for the lack of datasets. This because, there are very few datasets available that are eligible for training a neural network for this specific purpose whose job is to recognize players and sports balls.

To overcome this problem, the natural solution was to pick a pretrained model, and the choice was YOLOv8.

YOLOv8

YOLOv8 is the 8th generation of the You Only Look Once (YOLO) object detection algorithm. It is widely recognized for its real-time performance, accuracy, and compact model size. YOLOv8 builds upon the advancements of previous versions and incorporates various enhancements to improve its capabilities.

The primary application of YOLOv8 is object detection, enabling the identification and localization of multiple objects within images or video frames.

We adopted this model, among all the available versions after an accurate analysis of the trade-off between performances, accuracy, and computational/inference time.

As we can see from the specification, YOLOv8m lies perfectly in the middle but despite it all, it performed better than the “v8l” and “v8x” versions.

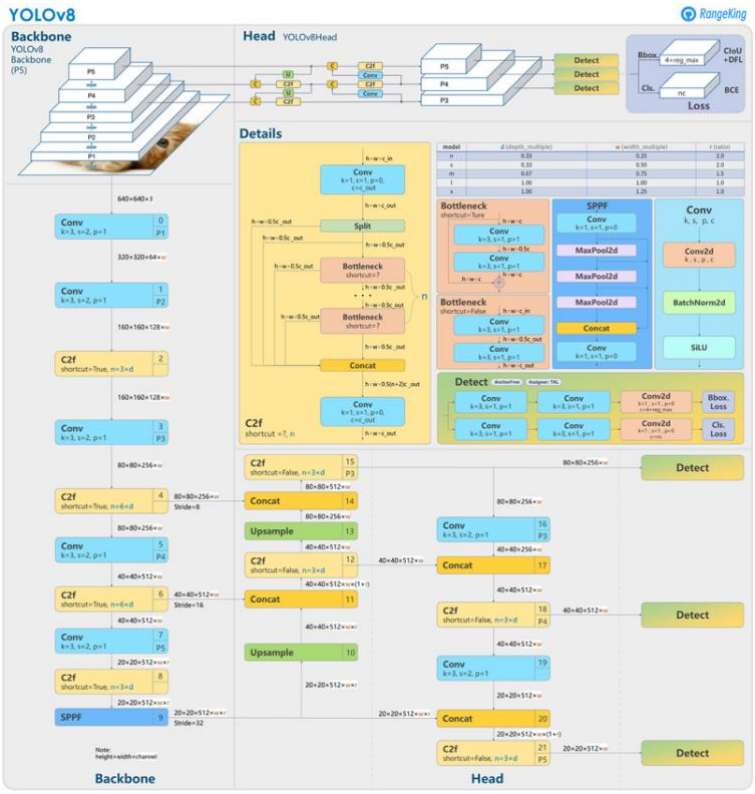
Model	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

To adapt YOLOv8 to our specific needs, we employed a technique called "fine-tuning." This technique consists of the utilization of a pretrained model as foundation and in its subsequent customization to satisfy our specific requirements. In our case, we focused on training the model to recognize football players and the ball within a football field. To achieve this, we created a personalized dataset on Roboflow that included annotated images of players and balls. The ultralytics library simplified the training process by providing the "$model.train()$" method, giving the possibility to access training and test data through a YAML file. The library gives the possibility to export the model as well, so that it can be imported and used to make prediction on the input data.

By exploiting the capabilities of YOLOv8 and utilizing our customized dataset, we aimed to enhance the model's detection accuracy at identifying footballers and balls in a football environment.

Nevertheless, probably due to the lack of datasets, the results were not the same as our expectation, for this reason we decided to adopt the stock model.

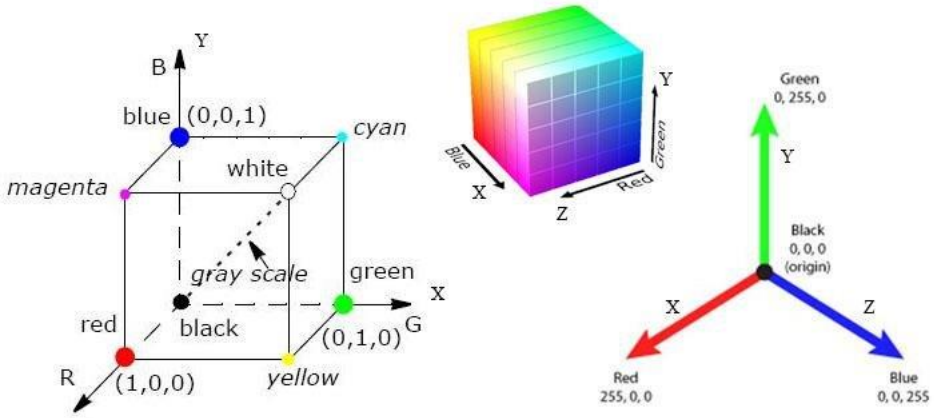
Here we provide a detailed visualization of the network's architecture.



Team recognition: a K-means Approach

To recognize what team a player belongs to we chose to differentiate teams based on the colour which better represents them.
With the purpose of raising the probability of characterising a player with the correct colour, the model picks only a fraction of the footballers' box located slightly above the centre of the rectangle, the section that we empirically determined as the most significative. This because on the one hand choosing a section too low would imply the detection of a part of the pants, which usually has a different colour from the t-shirt, on the other hand one too high can be misleading because of the different colour of the number or of the skin or, even, he could include the football field.

We indeed studied the RGB colours vector space, which is a subspace of $\mathbb{R}^3$ and because each axis can take values in [0,255] it is represented as a cube of edge 256.



Since each pixel can be seen as a vector belonging to this vector space, once the section has been identified the mean vector of each section is computed according to the formula:

$$\mu_i = \frac{1}{N}\sum_i^N x_i$$

Each x_i is a pixel, and it is inserted as vector into the initially empty subspace of $\mathbb{R}^3$.
With the help of a threshold, we constrain every colour shade to fall into a specific RGB combination, so that the function *"color_picker()"*, recognises no more than 8 colours, corresponding to the vertices of the colour cube.

Once the algorithm has collected enough points, the K-means algorithm is run with $k = 2$, where k is the number of clusters.

The model identifies two colours which are the 2 having the highest probability of representing teams' t-shirts. This algorithm is run only once at the set-up phase, but the decided colours will be held for the whole duration of the main loop.

During each frame, the model associates each identified player to one of the two teams. This is done by computing the vector mean for each object box of the players with the above procedure and computing the Euclidean distance between the calculated mean and the two colours chosen at the beginning.

The player will be associated to the team represented by the colour which is spatially closer to the mean vector.

These is the formula we used:

$$\gamma = argmin_{c_i}(||c_i, p||) \qquad i \in \mathbb{N},\ 1 \leq i \leq 2$$

where:

$$||c_i, p||_2 = \sqrt{\sum_j^3 (c_{ij} + p)^2}$$

c_i is the i^{th} team colour assigned at the beginning and c_{ij} its j^{th} component

p is the predicted colour

γ is the assigned colour

Environment recognition

Detecting how the field is composed is essential to fully comprehend what is happening with the flow of the frames.

To do so, computer vision techniques are applied to analyse the frame.

The first step is to highlight the field lines:

1. the frame image is turned into HSV format.
2. a green mask is created and applied to the image to isolate the field portion.
3. from the masked frame, only colour values in a specific range are taken to visualise the lines.

Once we have the lines highlighted, the $cv2.FindContours()$ method is applied. The output of this method is a list of lists, each one containing the points that are more likely to compose a border. Among these lists, is necessary to do a selection of which ones are actually considerable as contours. The first selection parameter is the size of the area formed by each one.

Then the $is\_contour\_straight()$ function is called in a loop iterating among the contours: each list of points is passed as input, and inside the function the $r\ coefficient$ is calculated between the $x's\ and\ y's$, to have an idea of how much correlated the points are.

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 (y_i - \bar{y})^2}}$$

After the correlation is verified, $np.polyfit()$ function is used to fit a straight line on the set of points.

This fitted line is used to understand where the line might be positioned in the environment. For this purpose, the slope of the line is used as indicator of its inclination, and based on this parameter the line is identified as *upmost, rightmost, lowest, leftmost*. Successively this classification will be used for the study of the ball position.

Action recognition

The main section of the program is designated to study what is happening, frame by frame. This resolves into a summary of the main actions that take place during the match.
This section is implemented mainly inside the "$while\ True$" loop, governed by the "$cv2.cap()$" method. This method reads the input video and divides it in frames with the purpose of analyse each of them. Either if the video is not found or if the user presses the key "q" the program stops raising an error.

Before the While loop, the set-up phase

To make the colour picking the most accurate as possible, the program samples n random frames from the whole video input until 1000 boxes of the wanted boxes are detected, it computes the average colour and fits it in the RGB vector space to run the K-means algorithm after that enough frames are collected with the procedure described above. In this way the program is able to identify the main colours of the t-shirts of the football players iterating throughout the sampled frames with the "$color\_picker()$" method.

A *Team* class is defined, created passing the recognized colours. Each class instance has its own statistics parameters that are going to be modified when necessary.

The While True loop

During each iteration, the current video frame is fit into the model to identify the footballers and the sport ball that are going to be subsequently analysed.

First, each detected object is assigned the label belonging to the corresponding class, and a rectangle box is drawn around it: the box is going to be red for identifying the ball, and for every player it's going to be of the same colour of his own team.
If the identified object is a player, as well as assigning a colour with the above procedure, his coordinates are put in the "$players$" array of respective team class.

If the label is "ball" instead, its coordinates are stored in the $last\_ball\_coords$ variable.

If the algorithm is in the initial phase and the Boolean variable $settings$ is set to True, the procedure described in the "***Team recognition***" section is used to compute the teams' colours and set them as the colours of the statistics in the UI, otherwise an ordinary loop is performed.

Now the algorithm calculates the statistics to show on screen.
The underlying assumption is that time flowing is determined by the frames of the video; in particular, each frame is a unit of time.

Ball possession

The possession of the ball is assigned to the team having the closest player to the ball.
This is done by computing the Euclidean distance between the centre of the ball's box and the centre of each footballers' box.
The team satisfying this condition, gains a "unit of possession", this is used at the end of each iteration to determine the total percentage of possession, given by the following formula:

$$ball\_poss(t_i, M) = \frac{\sum_j^M \delta(u_j = \mathrm{i})}{\sum_k^M u_k}$$

where:

- t_i is the i^{th} team.
- M is the total number of past frames.

$$\delta(u_j = \mathrm{i}) = \begin{cases} 1\ if\ u_j = i \\ 0\ otherwise \end{cases}$$

Of course, the possession of the ball is not assigned regardless of the least found distance from the ball to a player; a minimum pixel distance is set, so that the possession is considered valid.

For each iteration, the team currently holding the possession is registered into the new_pos variable and the old value of new_pos is assigned to $last\_pos$ variable.

If the possession changes, there is a tackle, which is assigned to the respective team.

Ball movements

Another statistic we decided to calculate are the **passages** performed by each team.
The $passage$ variable, set to a Boolean value, is indispensable to understand when to memorize the starting coordinates (x, y) of the passage; the coordinates are saved into the variable $ball\_position$.

Since the beginning of the passage, this variable will not receive any variation; this characteristic is essential in the understanding of the ball movement.

When the ball ward off 30 pixels from the player who began the passage, this is recognized to be started, therefore the program sets $passage = True$, so that the already saved $ball\_position$ will not change.
From now on the program is going to check for any player receiving the ball, at least 30 pixels far from the “sender”:

- If the *receiver*'s team is equal to the *sender*'s, the passage is completed, and the team's counter will be increased by one.
- Otherwise, a **tackle** is detected, and the opposite team's tackle counter is increased by one.

Every frame, the ball position is checked to be inside the field limits. If this is not the case, then it's over the upmost, lowest, rightmost, or leftmost line; when this there is an **out**, and the responsibility for the infraction is assigned to the last teams that had the ball possession.

User Interface

To make the results easy to look at, in parallel to the video display, we decided to implement a graphical user interface to let the user not to struggle with comparisons among different open files. In this way either statistics or the edited video with the corresponding boxes is gathered in a unique interface.
Here is shown an example of the UI at the beginning of a match:

The statistics are shown on the left column while the video on the right.
Statistics are written with the same colour of the teams' ones and each detected footballer is surrounded with the correct colored box.
The UI is designed so that it is resizable and automatically adapts to the video size, maintaining the form factor.
At the end of each frame analysis the statistics are updated, in this way the results are in real time.

Bumps on the road

Developing our software, we went through a lot of ideas to make the project the most accurate and reliable on the tasks it is supposed to accomplish and rich of interesting features.

Number recognition

The original development plan included the creation of a custom neural network able to recognize the numbers on the footballers' t-shirts. Our first idea was to recognize the players using his number, to assign a precise label on each player so to keep track of the individual statistics.
We tried to develop neural networks of different dimensions and structure alternating between FCN and CNN technologies.
Unfortunately, the lack of available datasets and the difficulty in creating them, together with a small availability of computational power, made the task quite challenging; after having built our networks and having created our custom numbers' dataset, we never managed to have good results on training and testing as expected, considering that the files on which the model was trained on were of low to medium quality and that the targeted portion of the frames is very small.

Ball movements' prediction

Unfortunately, YOLOv8 standard model is not always able to recognize every element on the frames; it's not unusual that the ball is lost during rapid passages and shots.
To overcome this problem, we tried to implement a Linear Regression model, with the aim of predicting the future ball position through the past registered x and y coordinates of the ball position. This would have helped to have a better idea of the ball movement, with a consequent improved precision of the final statistics.

Compressing the data

An important problem of our program was the responsiveness: in fact, the software can be very heavy sometimes, because of the complexity of the calculations it must perform. To solve this problem, we thought about compressing the data, developing a PCA, being careful in not losing more than the 5% of the information. But, compressing the data would have meant compressing colours' information, and their RGB values to change accordingly. But colours' shades are fundamental to compute team recognition, making data compression not a good solution.