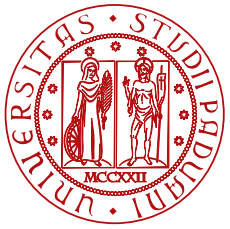


Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Report of Computer Vision project

Sport video analysis

Students

Alberto Dorizza

Student ID 2075216

Dario Mameli

Student ID 2087636

Sara Farris

Student ID 2075215

Professor

Prof. Stefano Ghidoni

Group Name

SADVision

Academic Year
2022/2023

Contents

List of Figures

[1] https://upload.wikimedia.org/wikipedia/commons/thumb/3/31/VJ_featureTypes.svg/640px-VJ_featureTypes.svg.png

[2] https://it.mathworks.com/discovery/support-vector-machine.html

[3] https://blog.roboflow.com/whats-new-in-yolov8/

[4] https://blog.roboflow.com/whats-new-in-yolov8/

[5] https://towardsdatascience.com/metrics-to-evaluate-your-semantic-segmentation-model-6bcb9

List of Tables

List of Algorithms

List of Code Snippets

List of Acronyms

CSV Comma Separated Values

HOG Histogram of Oriented Gradients

SVMs Support Vector Machines

YOLO You Only Look Once

ROI Region of Interest

1 Introduction

In this report, we as Team SADVision, will present our solution to the problem of developing a robust and accurate system for sports video analysis. We will delve deeply into the designing process and the main tools from the literature we took inspiration from.

Our multi-faceted system is built so as to put in comparison older techniques with the newest ones, with the ultimate goal of producing the best compromise system for speed and accuracy.

The report is structured as follows:

- Chapter 1: the introduction to the project.
- Chapter 2: an overview of the main algorithms adopted from the scientific literature.
- Chapter 3: the core analysis of the structure and functioning of the code.
- Chapter 4: the analysis of the performance using the classic measurement metrics.
- Chapter 5: a brief recap of the report with the final conclusions.

2 Literature

In this chapter, we will take a closer look at the literature techniques that were used to produce the code. How they were put into use will be explored in more detail in Chapter 3.

2.1 Classic algorithms

In this section, we will provide an overview of all of the algorithms that were adopted which do not include any form of machine or deep learning.

2.1.1 Gaussian Blur

Gaussian blur is a technique used to reduce noise and enhance image quality by applying a Gaussian function-based filter. It acts as a smoothing mechanism, effectively diminishing the impact of high-frequency details while retaining the low-frequency components in an image. This process is typically executed by convolving the image with a Gaussian kernel, a matrix that defines the blurring effect. In its 2-dimensional form, the Gaussian kernel is mathematically represented as follows:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2 + y^2}{2\sigma^2}} \tag{2.1}$$

In this expression, σ represents the standard deviation of the Gaussian distribution. The σ value plays a pivotal role in determining the extent of the blurring

around each pixel. A higher σ value results in a broader distribution, leading to more pronounced blurring, while a lower σ value produces a narrower distribution, resulting in a milder blurring effect.

2.1.2 Discrete Laplacian

In image processing, the discrete Laplacian is a 2-D isotropic measure of the 2^{nd} spatial derivative of an image. It is employed as a digital filter that highlights areas of rapid intensity changes within an image, making it particularly useful for edge detection. To reduce sensitivity to noise, it's often applied after smoothing the image with a Gaussian filter. This discrete Laplacian operator is essentially a combination of two one-dimensional second derivative filters, applied together in a two-dimensional fashion. The filter therefore implements the following equation:

$$L(x, y) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2} \tag{2.2}$$

Where I is the image intensity value at pixel (x, y).

2.1.3 Bilateral filter

Bilateral filtering is a useful image processing technique that strikes a balance between smoothing an image to reduce noise and preserving its critical features, particularly edges. It accomplishes this by considering both spatial proximity and intensity similarity. The method combines domain and range filters, with the latter distinguishing between similar and dissimilar pixel values. This allows bilateral filtering to maintain sharp edges and contours while effectively reducing noise and fine details in images.

In color images, bilateral filtering becomes even more valuable, as it operates in perceptually meaningful color spaces. It ensures that only visually similar colors are averaged together, reducing color artifacts while retaining edge clarity. Multiple iterations of bilateral filtering can further refine images, making it a versatile tool for various image enhancement tasks.

Further information is available at [7].

2.1.4 Canny Algorithm

The Canny algorithm is a widely used edge detector, first proposed by J. Canny in 1986 [1]. It's a filter addressing the following targets: low error rate, edge point well localized, and single edge point response.

The algorithm is divided in four main steps:

- Smoothing with a Gaussian filter: we smooth the image before evaluating edges to perform a noise reduction
- Gradient computation: we calculate edges by using vertical, horizontal, and diagonal masks. Specifically, the edges are distinguished based on the phase: 8 bins of 45° are created.
- Non-maxima suppression: used to reduce the thickness of the edges. Thin edges are desirable because the edge points are localized more accurately. The process is done by crossing an edge with particular masks and selecting the strongest point.
- Hysteresis thresholding: used to keep strong edges, to keep weak edges connected with strong edges, and to reject isolated weak edges. It's based on two thresholds T_L and T_H from which we obtain two images I_L and I_H. Pixels in I_H are considered part of an edge while pixels in I_L are considered part of an edge only if strong edges are around.

2.1.5 HOG Descriptor

Histogram of Oriented Gradients (HOG) is a widely-used feature descriptor in computer vision and image processing, primarily used for object detection. It operates by counting the occurrences of gradient orientations in localized regions of an image, focusing on the shape and structure of objects. Here are the key steps[1] to calculate HOG features descriptor:

- Resize the input image to a standardized dimension (typically 128x64 pixels).
- Calculate the gradient of the image by computing the gradient magnitude and angle for each pixel, considering blocks of 3x3 pixels.
- Divide the gradient matrices into 8x8 cells and create 9-bin histograms for each cell, where each bin represents a 20-degree angle range.

[1] https://towardsdatascience.com/hog-histogram-of-oriented-gradients-67ecd887675f

- The values of the bins are appended to a 9-bin histogram array for each cell. The resultant matrix after the above calculations will have the shape of 16x8x9.

- 4 blocks from the 9-point histogram matrix are clubbed together with a stride of 8 pixels to form a new block (2x2).

- Concatenate the histograms from all cells within a block to form a 36-feature vector for each block.

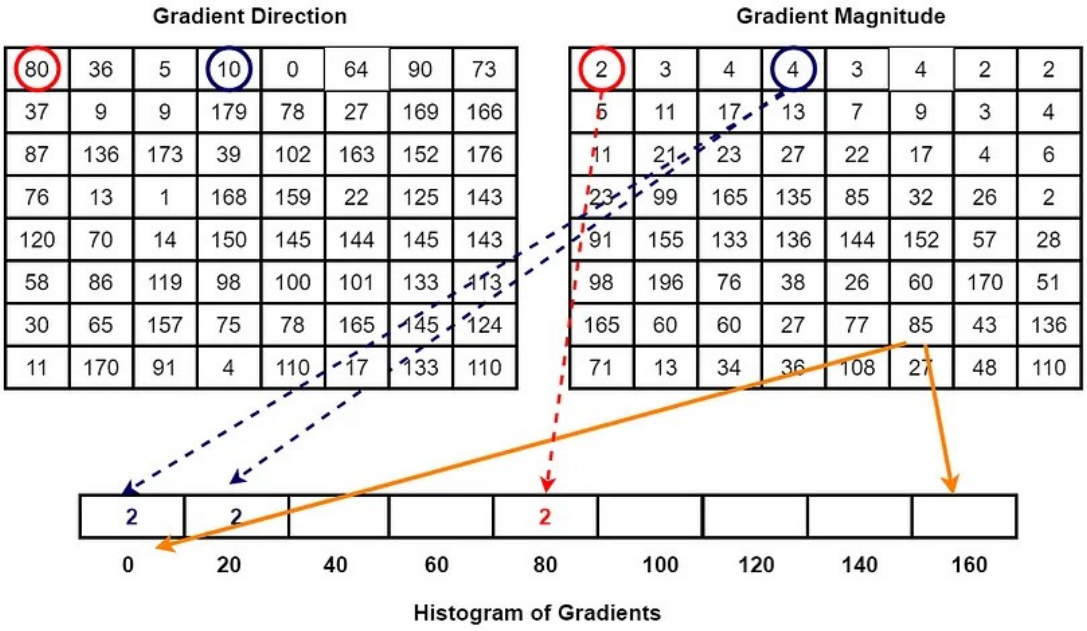
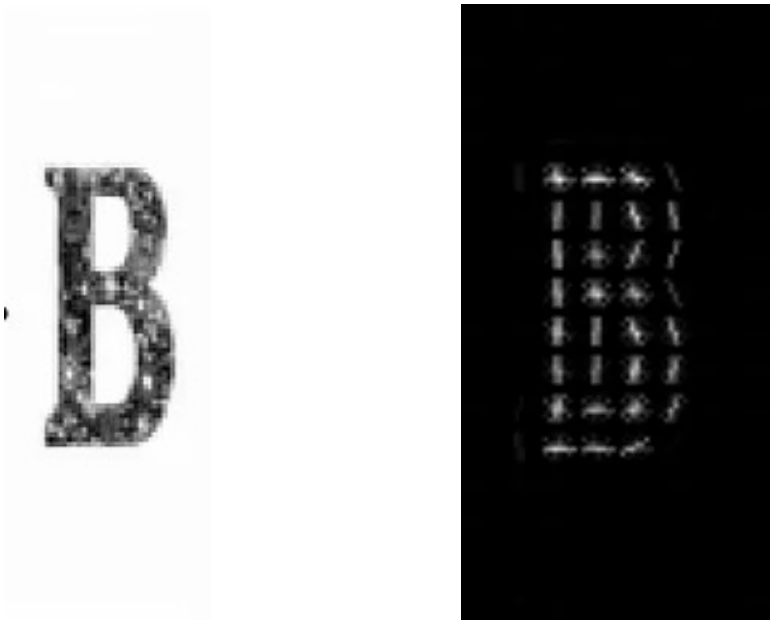


Figure 2.1: Method for calculation of 9 bin histograms

- Normalize the values of each block by dividing by the L2 norm to account for variations in contrast.

- Collect the 36-features vectors from all blocks to obtain the final HOG feature vector for the image.



(a) Original image (b) Visualization of HOG

2.1.6 K-means clustering

K-means is an unsupervised clustering algorithm used to partition a dataset into k clusters, where each cluster is represented by a centroid. The objective function is the following:

$$\min \sum_{i=1}^{k} \sum_{x \in C_i} d(x, \mu_i)^2 \tag{2.3}$$

Where C_i is the cluster i, x is a feature vector, μ_i is the centroid of cluster i, and d is some distance function (typically Euclidean).

A good heuristic to solve k-means is Lloyd's algorithm [1].

K-means seeks to minimize the within-cluster sum of squares, making clusters more compact and well-separated. It's a versatile algorithm used in various fields, such as image segmentation, customer segmentation, and pattern recognition. However, K-means requires specifying the number of clusters (k) in advance and is sensitive to initial centroid selection, which can lead to sub-optimal results.

Algorithm 1 Lloyd's Algorithm

Require: data points $X = \{x_i | i \in \{1, ..., m\}\}$
Ensure: clustering $C = (C_1, ..., C_k)$ of X; centers $\mu_1, ..., \mu_k$ with μ_i center for $C_i, 1 \leq i \leq k$

randomly choose $\mu_1^{(0)}, ..., \mu_k^{(0)}$
for $t \leftarrow 0, 1, 2, ...$ (until convergence) **do**
  for $i \leftarrow 1, ..., k$ **do**
    $C_i \leftarrow \{x \in X : i = argmin_j d(x, \mu_j^{(t)})\}$
  end for
  for $i \leftarrow 1, ..., k$ **do**
    $\mu_i^{(t+1)} \leftarrow \frac{1}{|C_i|} \sum_{x \in C_i} x$
  end for
  if convergence reached **then**
    return $C = (C_1, ..., C_k)$ and $\mu_1^{(t+1)}, ..., \mu_k^{(t+1)}$
  end if
end for

2.2 Machine Learning and Deep Learning

In this section, we will provide an overview of all of the algorithms that were adopted which include some form of machine or deep learning.

2.2.1 Haar features cascade classifiers

As in the Viola-Jones [8] object detection method, the cascade classifier is useful to significantly improve efficiency during the detection process. The cascade of classifiers is designed to quickly eliminate non-object regions, allowing for faster detection.

In the cascade structure, a series of stages are defined, with each stage containing a subset of the total features. The features adopted are called *Haar features*, which are simple rectangular filters introduced by Alfred Haar in 1909 [2]. They efficiently capture variations in pixel intensities thanks to their low computational cost. They are able to identify key image features in complex objects such as faces.

Initially, the first stage is composed of a relatively small number of these features. If a candidate window fails to pass this stage (i.e., it is deemed a non-object region), it is discarded without further evaluation, conserving processing

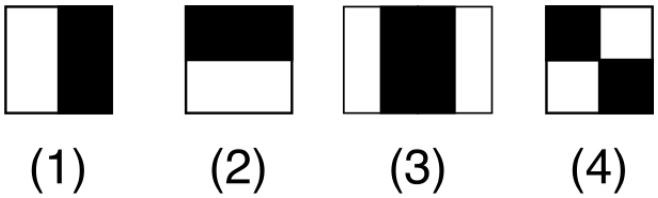


Figure 2.3: Example of Haar Features: Square shaped kernels [2]

time.

For windows that pass the first stage, they proceed to the subsequent stages, each with more features. This process continues until either a candidate window is rejected at a stage, or it passes all stages, confirming it as an object region.

This cascade approach significantly reduces the computational burden by concentrating efforts on regions that are more likely to contain objects. It allows for efficient real-time object detection.

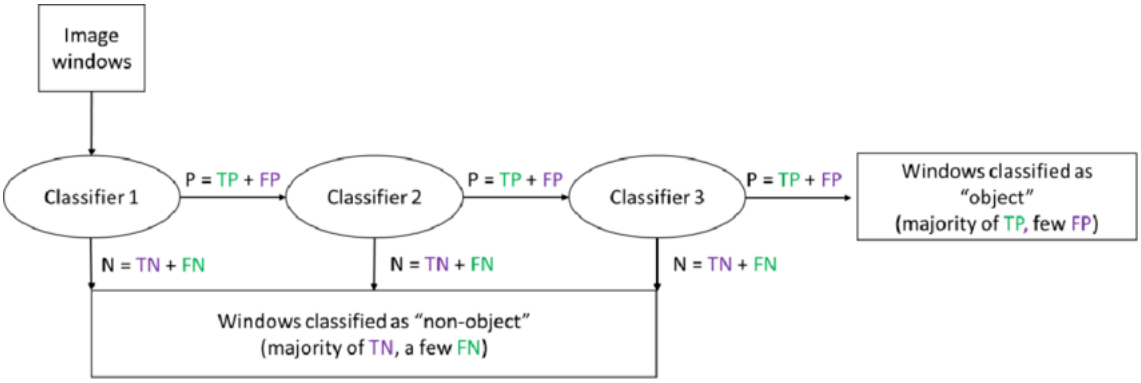


Figure 2.4: Sample schema of a cascade classifier [6]

2.2.2 Support Vector Machines (SVMs)

SVMs (Support Vector Machines) are machine learning algorithms used for classification tasks. Their primary objective is to find a separating line (or hyperplane in higher dimensions) between two (or more) classes of data points.

SVMs seek the optimal separating hyperplane using a specific approach:

- Candidate Lines: SVMs begin by considering various candidate lines or hyperplanes that could potentially separate the data.
- Support Vectors: They identify the data points that are closest to the decision boundary. These points are called support vectors. The key idea

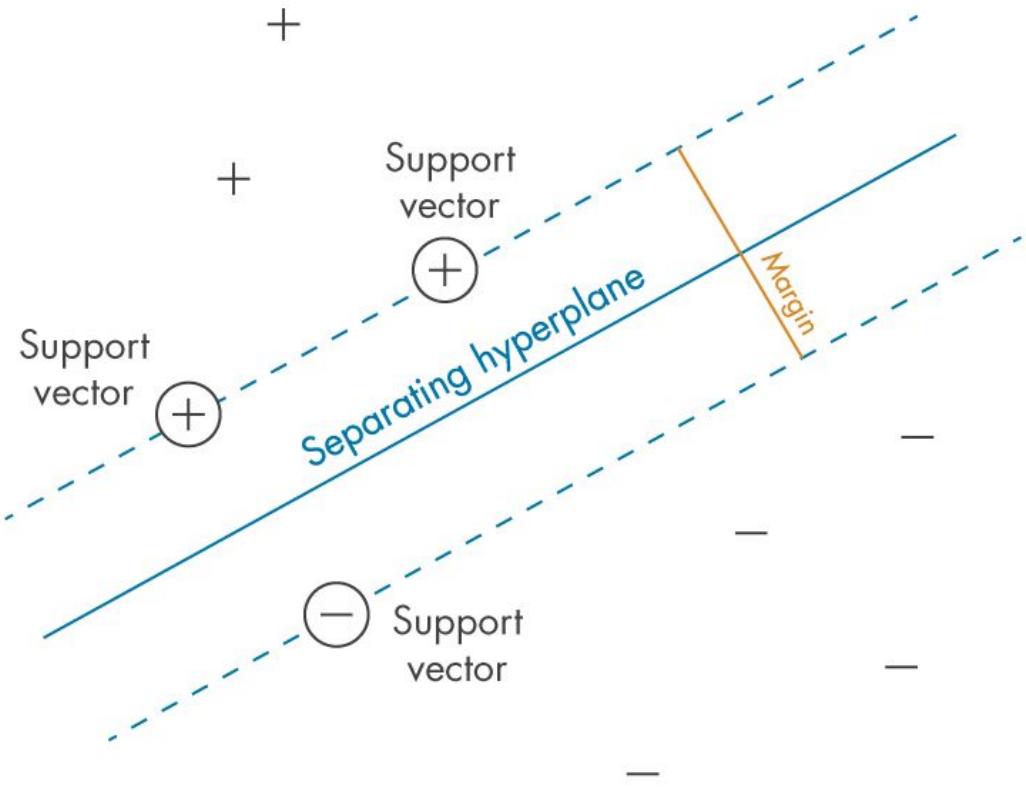


Figure 2.5: Graphical representation of SVMs in the two-dimensional case[3]

is that these points are able to completely determine the optimal separating hyperplane.

- Margin Maximization: The optimal separating hyperplane should have a maximum margin, which is the distance between the hyperplane and the nearest support vectors.

In cases where the data is not linearly separable, SVMs can transform it into a higher-dimensional space by adding dimensions based on specific mathematical functions that make the data linearly separable. SVMs then project the decision boundary back into the original dimensions.

Additionally, SVMs can use kernel functions to automatically perform these transformations, without explicitly altering the original data, a technique known as the "kernel trick."

2.2.3 You Only Look Once (YOLO)

You Only Look Once (YOLO) (You Only Look Once) is a family of computer vision models for solving complex tasks such as object detection and instant

segmentation. It has been one of the most popular go-to choices since Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi introduced the novel architecture in 2016 at CVPR, winning OpenCV's People Choice Awards. [4]

Introduction The original YOLO was written by Joseph Redmon [5] in a custom framework called Darknet which is very flexible due to the fact that it was written in low-level languages. It has produced a series of the best realtime object detectors in computer vision: YOLO, YOLOv2, YOLOv3, YOLOv4, YOLOv5, YOLOV6, YOLOV7, YOLOv8, and YOLO-NAS.

Since its inception in 2016, the YOLO family of models has seen continuous development. YOLOv2 and YOLOv3 were created by Joseph Redmon, while subsequent models were authored by different individuals.

The original YOLO model was groundbreaking as it was the first object detection network to seamlessly integrate two essential tasks: drawing bounding boxes and classifying objects. This integration was achieved within a single end-to-end differentiable network.

General architecture YOLO operates as a single-stage detector, managing both object identification and classification in a single network pass and excelling in terms of both speed and accuracy.

YOLO models are often characterized by their swiftness and compact size. This quality makes them faster to train and more straightforward to deploy, particularly on devices with limited computational resources.

YOLOv8 The latest entry in the family is YOLOv8 by Ultralytics [5], introducing a plethora of architectural enhancements and improvements compared to the previous iterations.

[4] https://blog.roboflow.com/guide-to-yolo-models/
[5] https://ultralytics.com/yolov8

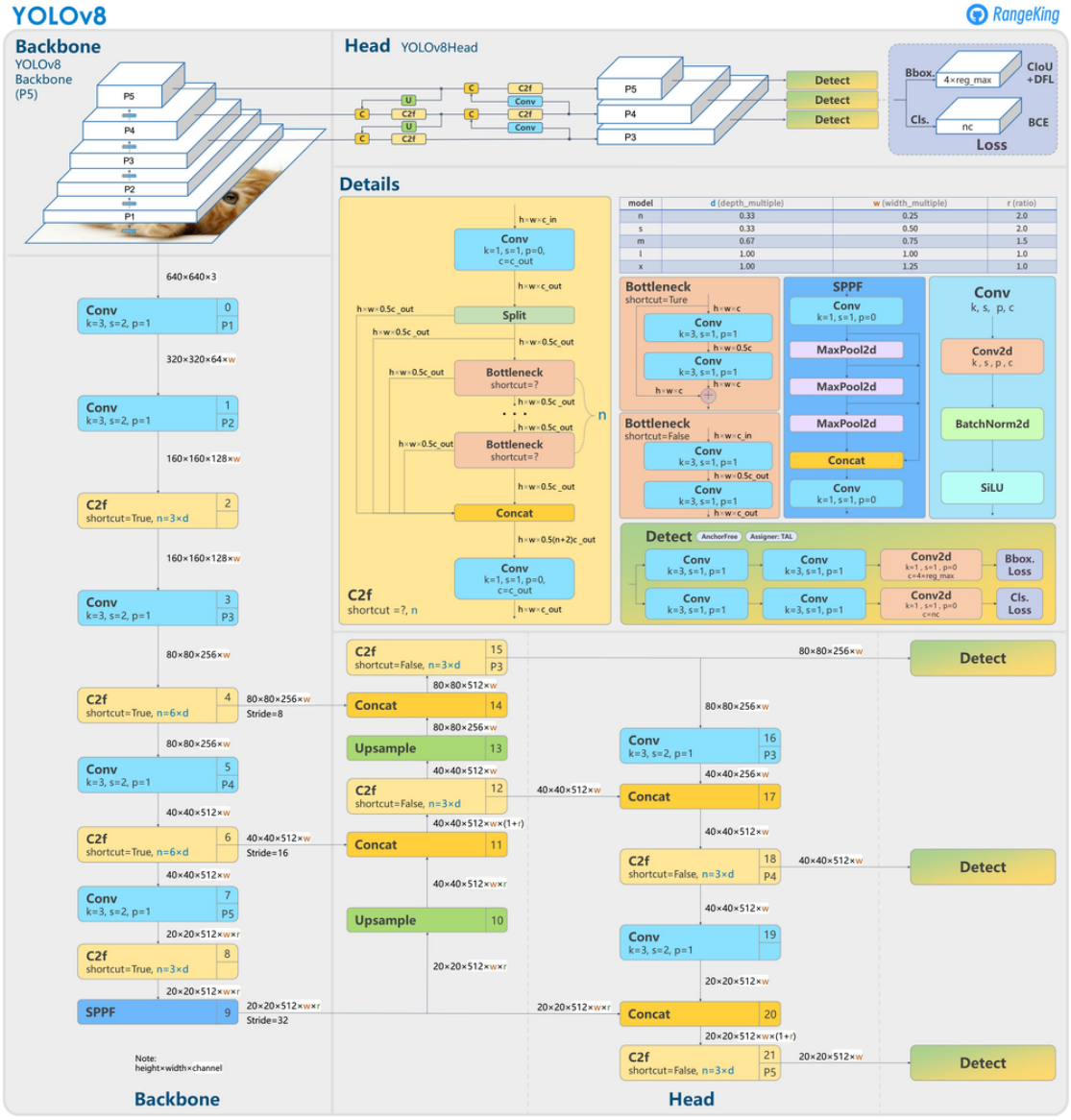


Figure 2.6: YOLOv8 Architecture, visualisation made by GitHub user RangeKing [6]

Going through the changes, we can find in particular:

- **Anchor-Free Detection**: YOLOv8 adopts anchor-free detection, predicting the object's center directly without relying on anchor boxes. Anchor boxes in earlier YOLO models could sometimes mismatch the distribution of custom datasets. This anchor-free approach reduces the number of box predictions and accelerates Non-Maximum Suppression during post-processing.

- **Architectural Changes**: YOLOv8 introduces architectural changes, such as replacing the 6x6 convolution in the stem with a 3x3 convolution. The primary building block is modified, and C2f is replaced by C3, impacting

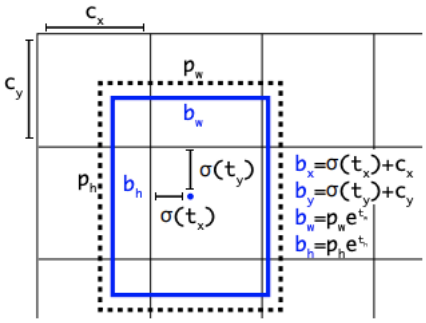


Figure 2.7: Visualization of an anchor box in YOLO [7]

the feature concatenation process. While the Bottleneck structure remains similar to YOLOv5, the kernel size of the first convolution changes from 1x1 to 3x3, aligning with the ResNet block design from 2015. In the neck of the architecture, features are concatenated directly without enforcing consistent channel dimensions, reducing parameter count and tensor size.

- **Mosaic Augmentation**: YOLOv8 employs mosaic augmentation during training, which stitches four images together to introduce variations and challenges such as partial occlusion scenarios, and diverse backgrounds. However, mosaic augmentation can degrade performance if applied throughout the entire training process, so it is not employed in the final ten training epochs.

3 The System

In this chapter, we will go in-depth into the development process and the core ideas that resulted in the final code.

3.1 First attempts with OpenCV

The first iteration of the system only makes use of OpenCV functions for solving the task of detecting players within each image, by predicting a bounding box for each instance and classifying it depending on their team. As it can be seen in Figure 3.1, the steps can be summarized into:

1. **Preprocessing**: each image is processed by four algorithms into 4 images as in Figure 3.2.
   (a) Identity: no processing involved, the output image is the input.
   (b) Gaussian blur: the image is converted to grayscale and then blurred with a Gaussian filter (2.1.1) of kernel size 5x5.
   (c) Gaussian blur + Laplacian: after gaussian blurring the laplacian (2.1.2) of the image is subtracted for sharpening effect.
   (d) Bilateral filter: the image is processed by a bilateral filter (2.1.3) with the diameter of each pixel neighborhood equal to 5, sigmaColor and sigmaSpace equal to 150 for strong filtering (cartooning effect).

2. **Localization algorithms**: one of three algorithms is chosen for the localization:
   - HOG + SVM: extracts the HOG features (2.1.5) from the input image predicting bounding boxes at multiple scales via SVM model (2.2.2) pretrained on people detection.

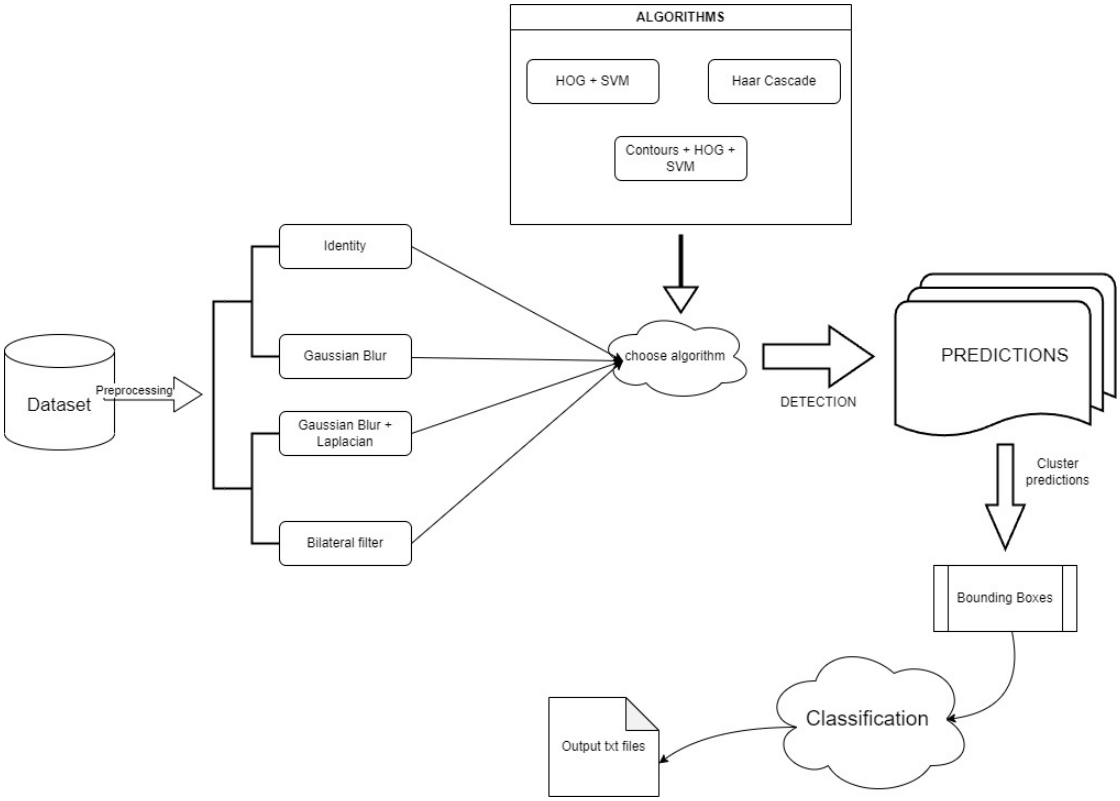


Figure 3.1: Summarizing schema for the architecture at the initial stage of the development

- Haar Cascade Classifiers: computes the Haar features (2.2.1) on the input image and predicts bounding boxes at multiple scales using 3 cascade classifiers pretrained for full, upper, and lower body detection.
- Contours + HOG + SVM: processes the input image with Canny algorithm (2.1.4) and identifies the strongest contours from a hierarchical structure. The image with contours is then processed as in the HOG + SVM method.

Outputs are shown in Figure 3.3.

3. **Clustering predictions**: the bounding boxes predicted for each image by the chosen algorithm for localization are then clustered together so that rectangles with significant overlap are merged into a single representative rectangle, as it can be seen in Figure 3.4.

4. **Classification**: finally the remaining bounding boxes are converted into feature vectors, where each vector is a flattened normalized histogram of the color distribution within each bounding box ROI. These feature vectors are clustered together using k-means (k=2), and the bounding boxes are therefore classified accordingly into one of the two teams. Bounding boxes are saved in a text file for each image. Output example is shown in Figure 3.5

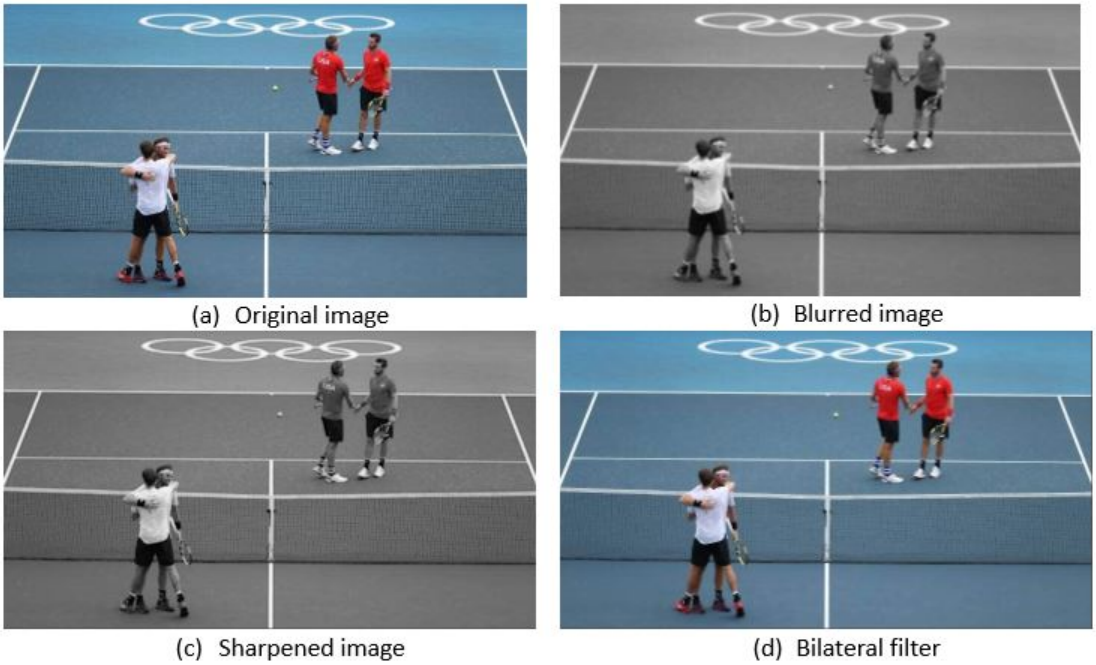


Figure 3.2: Example of preprocessing for img8.jpg

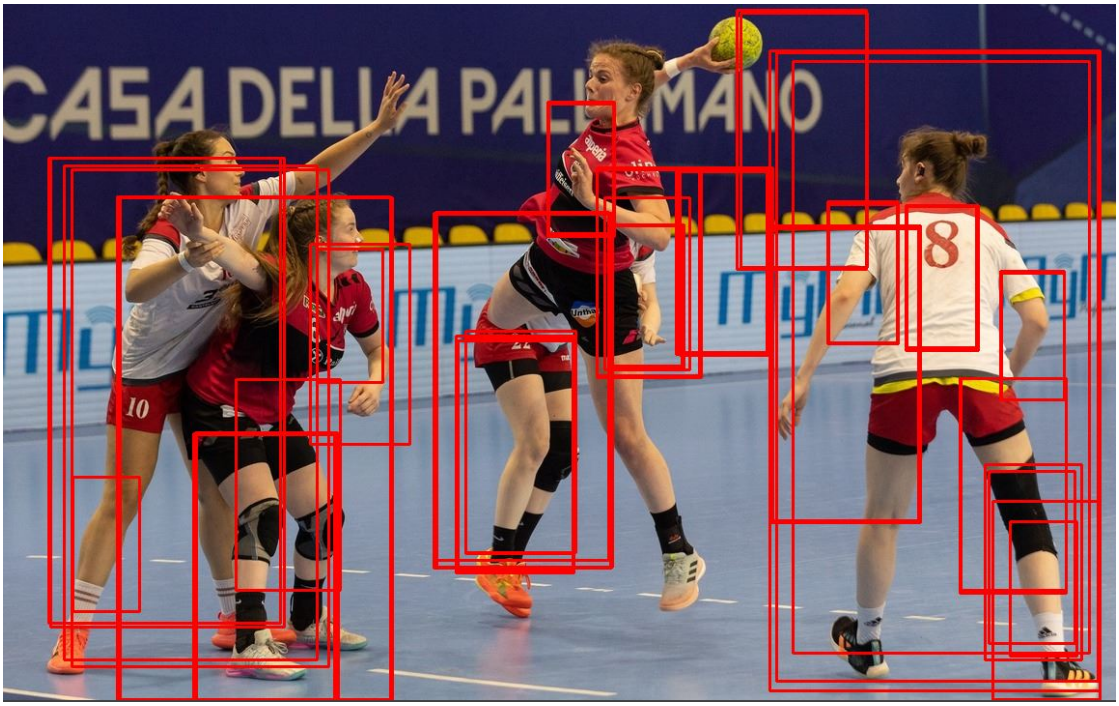


Figure 3.3: Example output of localization using HOG + SVM with all the bounding boxes for img11.jpg

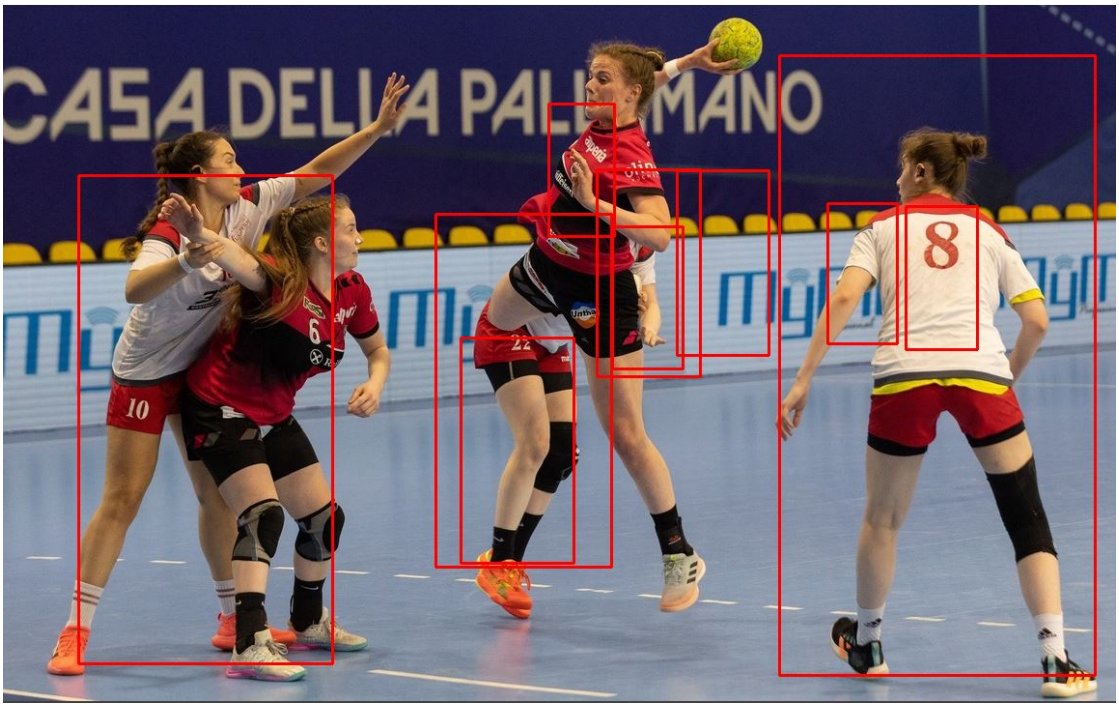


Figure 3.4: Example output of localization using HOG + SVM with clustered bounding boxes for img11.jpg

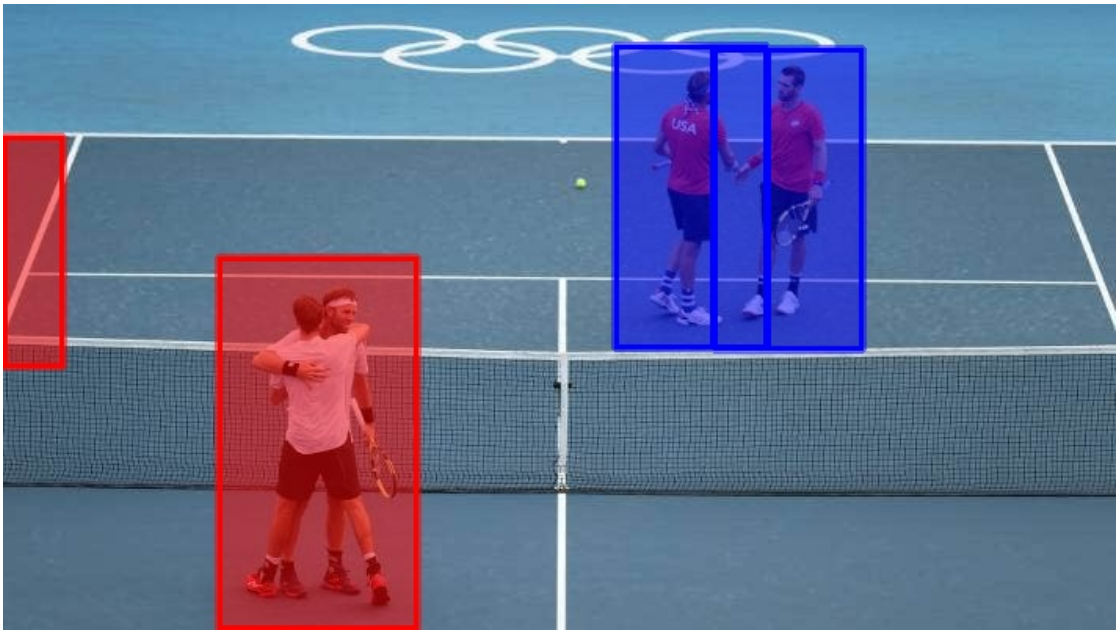


Figure 3.5: img8.jpg with classified bounding boxes predicted using HOG + SVM

3.2 Instance Segmentation with YOLO

The task of localization is clearly a non-trivial task. While tools such as HOG Features (2.1.5), cascade classifiers (2.4), and SVMs (2.2.2) are undoubtedly powerful in certain applications, for people detection they are unable to provide results which are even remotely as accurate as the current state-of-the-art models, that are, CNNs. The complexity of the task requires complex models for its solution, and being neural networks universal approximators they can reasonably represent any function given a sufficient number of trainable parameters. More discussion about this will be held in Section 4.3, where these statements will be supported by our experimental evidence.

The next stage of the development process is, therefore, that of performing instance segmentation of the players using YOLO with the Python library *ultralytics* (2.2.3).

3.2.1 Datasets

Two datasets have been assembled for the purpose of training models. The idea is that of using datasets of players during sports activity in order to reproduce as accurately as possible the eventual test set. However, for the task of instance segmentation, only basketball, tennis, and soccer images were found. Both datasets have 9292 images, split 83-12-6 for training, validation, and testing.

- **Dataset 1: Player-Referee**[1]: augmentation is performed in the form of random cropping of 0% up to 30%. The classes considered for the segmentation are *player* and *referee*.
- **Dataset 2: Person-Referee**[2]: same augmentation on the same images, but different class mapping: *player* is now *person* and *referee* is the same.

3.2.2 YOLOv8 models

Four models have been used by our system, three of them trained on the two datasets using the Python library *ultralytics* and Roboflow API for loading the datasets.

[1] https://app.roboflow.com/computer-vision-7iioc/players-segmentation-ecdrn/1
[2] https://app.roboflow.com/computer-vision-7iioc/players-segmentation-ecdrn/2

1. **YOLOv8 pretrained**: The first model that has been tested is 'yolov8s-seg.pt', which is a pretrained model for instance segmentation on the COCO dataset.[3].
2. **YOLOv8 from scratch trained on Player-Referee dataset**: This model is called 'yolov8s-seg-scratch_player_referee.pt' and, as the name suggests, it is the 'yolov8s-seg.yaml' model, which is untrained by default, then trained (from scratch) on the Player-Referee dataset.
3. **YOLOv8 pretrained trained on Player-Referee dataset**: This model is called 'yolov8s-seg-pretrained_player_referee.pt' and, as the name suggests, it is the 'yolov8s-seg.pt' model which has been fine-tuned on the new classes and trained on the Player-Referee dataset.
4. **YOLOv8 pretrained trained on Person-Referee dataset**: This model is called 'yolov8s-seg-pretrained_player_referee.pt' and, as the name suggests, it is the 'yolov8s-seg.pt' model which has been fine-tuned on the new classes and trained on the Person-Referee dataset.

3.2.3 Training

The training routine for models 2, 3, and 4 is run in Google Colab, using Google's GPU environment, and it is the following:

```
import os
HOME = os.getcwd()

!mkdir {HOME}/datasets

!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="gJ9KweLd5IYIbnxIPxFZ")
project = rf.workspace("computer-vision-7iioc").project("players-segmentation-ecdrn")
dataset = project.version(1).download("yolov8")

"""## Custom Training"""

!yolo task=segment mode=train model=yolov8s-seg.yaml data={dataset.location}/data.yaml epochs=25 imgsz=800 plots=True

```

[3]https://cocodataset.org/#home

```
"""## Save the output models

from google.colab import files

files.download('/content/runs/segment/train/weights/best.pt')
files.download('/content/runs/segment/train/weights/last.pt')
```

Code 3.1: The code for the training

3.2.4 Inference

Any of the networks are used for predicting binary masks, each representing the segmented instance of a player or person in an image. The binary masks are then saved for further analysis, which will be the topic of Section 3.3. The code for the inference step is the following:

```
# Choose model
model = YOLO("yolov8s-seg.pt")

folder = r"Sport_scene_dataset/Images" # Dataset location
output_folder = r"runs/segment/predict/masks"  # Where to save the masks
output_folder_images = r"runs/segment/predict/images" # Where to save the segmented images

# Create the output folder if it doesn't exist
os.makedirs(output_folder, exist_ok=True)
os.makedirs(output_folder_images, exist_ok=True)

# Iterate through each file in the folder
for filename in os.listdir(folder):
    file_path = os.path.join(folder, filename)

    # Check if the item is a file (not a subfolder)
    if os.path.isfile(file_path):
        print(f'Found file: {filename}')

        # Read the image for masks creation
        img = cv2.imread(file_path)

        # Run inference
        results = model.predict(source=file_path, retina_masks=True, classes=0)
```

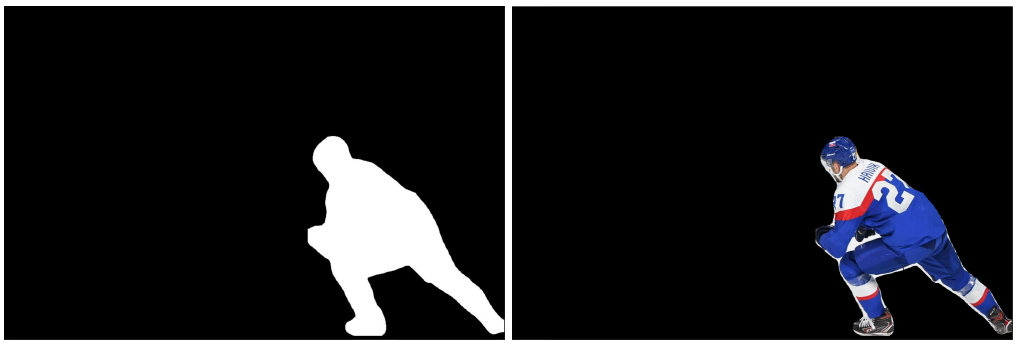
```
25
26          imageName = filename[:-4]
27
28          # Get the masks for people (class 0)
29          if results[0].masks is None:
30              continue
31          masks = results[0].masks.data
32          people_masks = masks if len(masks) > 0 else None
33
34          if people_masks is not None:
35              # Convert the tensor to a NumPy array
36              people_masks_np = people_masks.cpu().numpy()
37
38              i = 0
39              for instance_mask in people_masks_np:
40                  # Get each instance and save it after resize
41                  instance_mask_resized = cv2.resize(instance_mask, (
    img.shape[1], img.shape[0]), interpolation=cv2.INTER_LINEAR)
42                  instance_mask_toSave = (instance_mask_resized > 0).
    astype(np.uint8) * 255
43                  output_file_path = os.path.join(output_folder, f"{
    imageName}_mask_{i}.png")
44                  cv2.imwrite(output_file_path, instance_mask_toSave)
45                  i = i+1
46
47                  # Create a BGR image (3 channels)
48                  mask = np.zeros_like(img) # Initialized to 0
49                  mask[:,:,2] = instance_mask_toSave # Saved the values
     of the instance in the red channel
50
51                  # Overlay the red mask on the image
52                  img = cv2.addWeighted(img, 1, mask, 0.8, 0)
53
54              # Save the overlaid image
55              output_file_path = os.path.join(output_folder_images, f"{
    imageName}_segmented.jpg")
56              cv2.imwrite(output_file_path, img)
57
58          else:
59              print(f'No mask found for {filename}')
```

Code 3.2: The code for the segmentation inference

3.3 Player class and localization

At this point in the pipeline, we have access to the players' segments found by the network as binary images (masks). In order to speed up the inference, an encompassing method for the localization of bounding boxes and segments has been developed, which is based on the idea of using the masks for extracting the Region of Interest (ROI) from the image, effectively obtaining a BGR segment of the player for each mask. An example is shown in Figure 3.6.

Figure 3.6: Example of binary image (mask) and extracted BGR segment for img10.jpg



(a) Mask (b) BGR segment

Then, for each segment, an instance of an ad-hoc built class called *Player* is utilized, which holds the information for each player about the BGR segment and the bounding box that is generated after the segment. In particular, each bounding box is made of a rectangle (cv::Rect) which represents the coordinates of the best-fit contour encompassing the segment to which the Player refers within the image, in the format xywh (where (x,y) is the top-left point, w is the width and h is the height), and an integer *teamID* variable, representing the team affiliation.

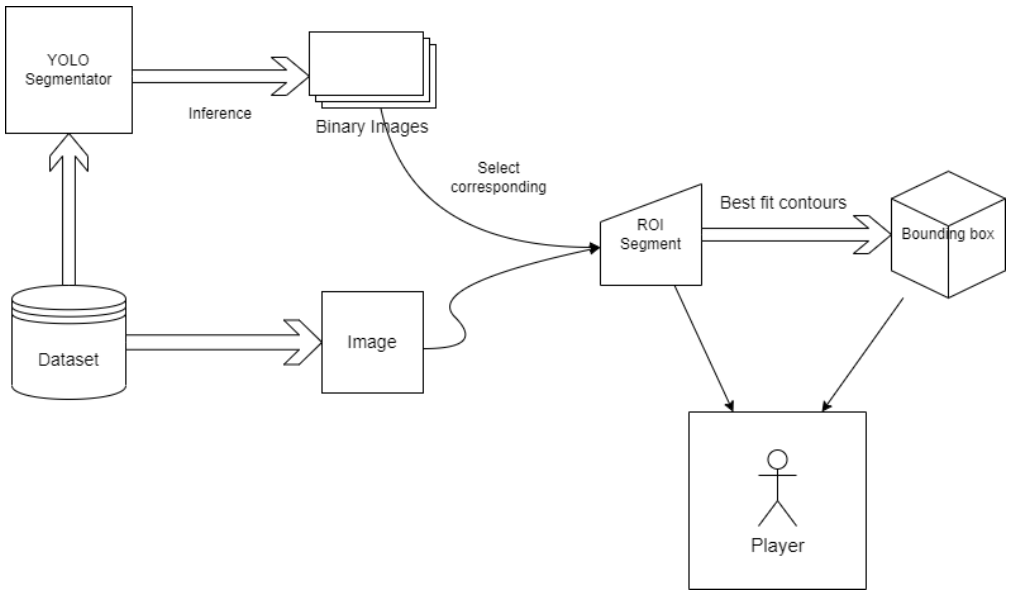


Figure 3.7: Simplified scheme of the building of Players

For now, the team affiliation is set to zero by default, since it will be decided at classification time, as will be explained in Section 3.6.

3.4 Color Features Segmentator

Given the variety of colours that the playing fields of various sports can have, the segmentation of the background and the playing field without the aid of neural networks is not a simple task. However, over the years, new techniques have been developed that perform this task without the aid of Machine Learning and Deep Learning techniques.

In particular, we have found and implemented a generalised field segmentation method using colour features, which was first presented in [3]. The method can be briefly described by the diagram in Fig. 3.8. More precisely, it first estimates the probability density function (pdf) of the colour components (cb, cr) of the input image, then applies a hill-climbing search to the pdf to generate different clusters, and finally the small regions are merged into the large adjacent regions to obtain the segmentation result.

In [3] they describe the estimation of the two-dimensional pdf obtained from the colour histogram of an image, the clustering method of local maximum with hill climbing (standard, fast 2D or approximate 1D) applied to the pdf that allows

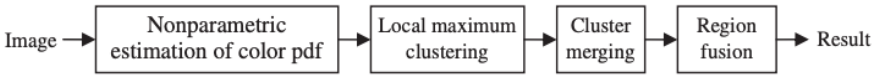


Figure 3.8: Flow chart of the color features segmentation proposed in [3].

for image re-quantization, and a four-colour cluster-merging method to segment the image into different regions. The application of a region-merging scheme to merge small regions into larger regions is only mentioned, but without providing details of the implementation they developed to achieve the results reported in the paper.

This segmentation method can be understood as a mode-seeking process within the cbcr histogram. It identifies the most prominent color bins, presuming that these modes correspond to red, green, or blue, which are the prevalent color components in various types of playing fields. Alternatively, these modes might correspond to grey, representing the average color of non-playground pixels.

Because the four-color mapping is very close to the actual scenes of generic sports videos, the mode search will be more effective and efficient since a good initial estimate of modes is obtained.

3.4.1 Nonparametric estimation of color pdf

As described in the research, colour characteristics can be extracted from various colour spaces, but the YCbCr colour space was chosen as it has a slightly higher average accuracy than the other colour spaces (e.g. Lab). Specifically, colour features were extracted from the YCbCr colour space as follows. First, the method extracts a (cb, cr) pair from each pixel of an image and calculates a two-dimensional (2D) histogram with respect to the Cb and Cr bins, as shown in Fig. 3.9. Second, using nonparametric estimation [4], they obtain an estimate of the pdf associated with (cb, cr), as shown in Fig. 3.10.

In practice, they obtain the estimate of a 2D pdf, $\hat{p}(x, y)$ by convolving an N-point Gaussian window, $w(k, l)$, with the 2D histogram, $h(x, y)$, as shown in Eq. 3.1.

$$\hat{p}(x, y) = \sum_{k,l} h(x - k, y - l)w(k, l), \quad -\frac{N-1}{2} \leq k, l \leq \frac{N-1}{2} \tag{3.1}$$

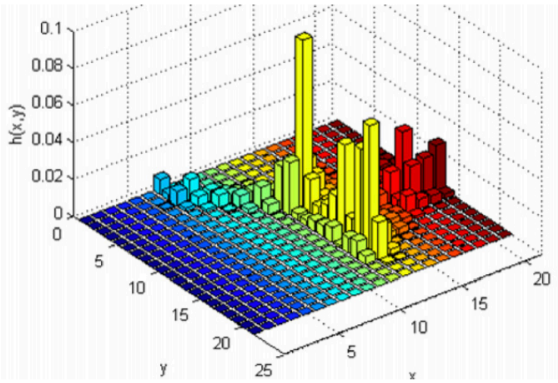


Figure 3.9: Example of a 2D CbCr histogram

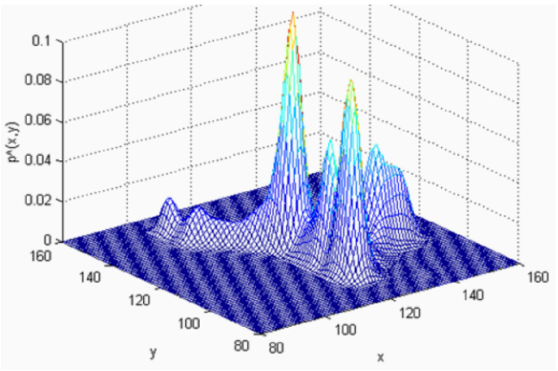


Figure 3.10: Example of a 2D PDF estimation

Where the N-point Gaussian window function is defined in Eq.3.1 with the reciprocal of the standard deviation α.

$$w(k,l) = exp\left(-\frac{1}{2}\left(\frac{\alpha}{N/2}\right)^2 (K^2 + l^2)\right) \tag{3.2}$$

Note that N and α determine the width of the Gaussian window function and, as reported in the article, they selected $N = 9$ and $\alpha = 2.5$, which yields the window shape shown in Fig. 3.11.

3.4.2 Local maximum clustering with 2D steepest-ascent hill climbing

Regarding the identification of local maxima of the estimated pdf Hung et al. [3] propose first a *Local maximum clustering with 2D steepest-ascent hill climbing* and then a faster version reported in Alg. 2.

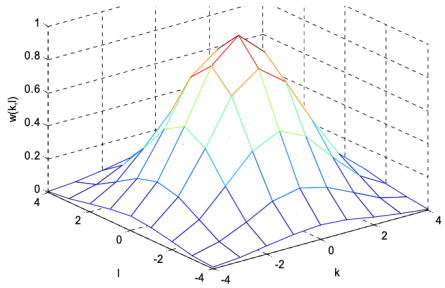


Figure 3.11: N-point Gaussian window function with $N = 9$ and $\alpha = 2.5$

By calling 'decreasing level' a density value that corresponds to the local minimum or saddle points of the pdf, this version of the algorithm designs a decreasing level to reduce the number of points to perform the hill-climbing process.

Since each level acts as a threshold dividing the curve or surface of the pdf into different regions, hill climbing is not necessary when a particular region contains only one local maximum point, because in this case all points in this region are signed at the maximum point, thus saving the search for a large number of points. This mechanism therefore only requires hill climbing for the remaining unlabelled points in the regions. As reported in the research, this fast scheme requires only about 10% of the search points compared to the original steepest-ascent hill climbing method.

By defining as a blob the set of points having the same associated local maximum and as the center of that blob the local maximum, we can then requantize the image by filling each blob with the representative color $(y_{mean}, cb_{max}, cr_{max})$, where (cb_{max}, cr_{max}) is the local maximum and y_{mean} is the average value of the Y component over all the pixels falling in the block.

3.4.3 Local maximum clustering with approximate hill climbing

In addition, the authors also developed an approximate hill-climbing scheme based on two 1D pdfs instead of the 2D pdf, which achieves a significant acceleration ratio at the cost of a small loss of segmentation accuracy.

Algorithm 2 Fast hill climbing.

```
//Initialization
find local minima and saddle points of pdf, p(x), and then sort and store them
in decreasing list, dec_list
find local maxima of pdf, p(x), and designate maximal points as C_1, C_2, ..., C_n

prepare a labeling array, lab(x), to record which maximal point will be as-
signed in x
assign all elements of lab(x) to be unlabeled
// decrease level and label points inside regions
for all d in dec_list do
  tf(x) ← p(x) > d?1 : 0
  // return a binary array, tf(x), if p(x) is greater than d, then tf(x) ← 1 else
  // tf(x) ← 0
  curr_regs ← binary_to_connected_region(tf(x))
  // return connected regions of tf(x)
  for all reg in curr_regs do
    if only one maximal point, c, exists inside reg then
      lab(x ∈ reg) is assigned to c
    else
      call conventional steepest-ascent hill climbing for each unlabeled point
      inside reg
    end if
  end for
end for
```

Figure 3.12: Example of input image to be segmented

Consider for example the image in Fig. 3.12 as input to this color features segmentation method. After the detection of the local maxima and local minima of 1D pdfs of cb and cr, as shown in Fig. 3.13 and Fig. 3.14.

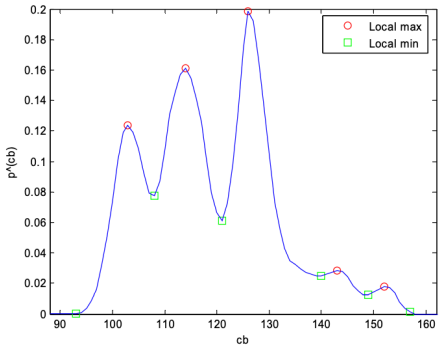


Figure 3.13: Example of local maxima and minima on the pdf of cb

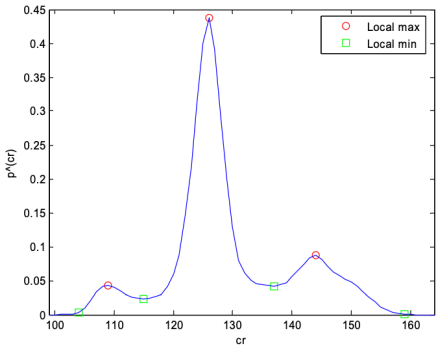


Figure 3.14: Example of maxima and minima on the pdf of cr

We then obtain a cross-maximal set, $\{(cb_{max}, cr_{max})|cb_{max} \in$ maxima on the pdf of cb, $cr_{max} \in$ maxima on the pdf of $cr\}$, shown as triangles in Fig. 3.15, enclosed and bounded by the local minima of the pdf of cb and the pdf of cr.

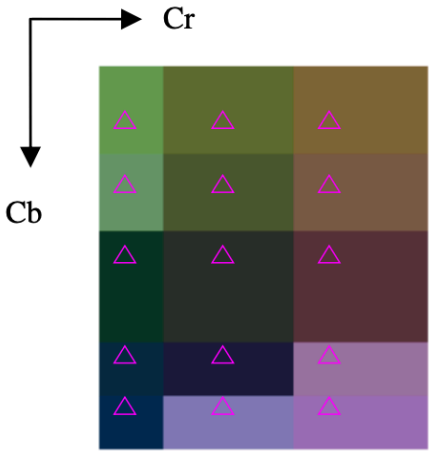


Figure 3.15: Example of a color map clustering

Then, all points in a block correspond to the same cross-maximum. Thus,

the block is replaced with the representative color, $(y_{mean}, cb_{max}, cr_{max})$, where (cb_{max}, cr_{max}) is a cross-maximum of the block, and y_{mean} is the average value of the Y component over all the pixels falling in the block, as was the case with the previously presented method. Thus, we can again obtain the requantized image using the representative colors as shown in Fig. 3.16.

Figure 3.16: Example of the corresponding re-quantized image

This approximate clustering scheme works mainly from the assumption that cb and cr are independent, so the 2D hill-climbing algorithm is simplified into two 1D search processes. In this way, the computational complexity of clustering is greatly reduced. Moreover, in [3] is shown how this approximate method achieves a speed-up of about 3.5-fold with only 0.07% loss of average segmentation accuracy.

3.4.4 Cluster merging and region fusion

The method presented by Hung et al. [3] is based on and works thanks to the observation that the colors of many playing fields used in various sports are often close to the three primary colors, red, green, and blue, due to the substances they are composed of. For example, baseball fields contain green grass and red soil, soccer fields are made up of green grass, and natural tennis courts are composed mainly of green grass or red sand. Additionally, many artificial fields are often covered with a blue plastic material, such as those used for basketball, volleyball, and tennis. Subsequently, the authors point out that other objects are visible during the game, such as players, spectators, commercial scoreboards, etc., which have a variety of colors that can be very different from the three

colors indicated above. But since non-playable objects are less important in the segmentation of the playing field only one gray color class is introduced for non-playing field objects.

Based on these previous observations they developed also a novel algorithm to segment playfields based on these four dominant colors. To define the four color classes, red, green, blue, and gray, they partitioned the CbCr plane into four parts according to the following rules. A pixel of an image, $x = (r, g, b)$, is classified into red, green, or blue using the simple rules reported in Eq. 3.3.

$$\begin{cases} x \in \text{red, if } r > g \text{ and } r > b \\ x \in \text{green, if } r > g \text{ and } r > b \\ x \in \text{blue, if } r > g \text{ and } r > b \end{cases} \tag{3.3}$$

The rules in Eq. 3.3 are reasonable and useful, as can be demonstrated by the experiment in Fig. 3.17

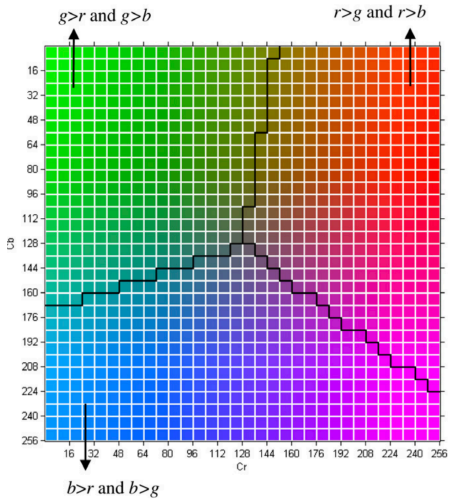


Figure 3.17: Color palette of the CbCr plane with Y = 100

By transforming the YCbCr color space into an RGB space using Eq. 3.4, we can rewrite the inequalities of the conditional expressions as Eq. 3.3.

$$\begin{bmatrix} r \\ g \\ b \end{bmatrix} = \begin{bmatrix} 1.164 & 0.000 & 1.596 \\ 1.164 & -0.392 & -0.813 \\ 1.164 & 2.017 & 0.000 \end{bmatrix} \begin{bmatrix} y - 16 \\ cb - 128 \\ cr - 128 \end{bmatrix} \tag{3.4}$$

$$\begin{cases} r > g \text{ or } r < g \Rightarrow 0.392 \cdot (cb - 128) + 2.409 \cdot (cr - 128) > 0 \text{ or } < 0 \\ g > b \text{ or } g < b \Rightarrow -2.409 \cdot (cb - 128) - 0.813 \cdot (cr - 128) > 0 \text{ or } < 0 \\ b > r \text{ or } b < r \Rightarrow 2.017 \cdot (cb - 128) - 1.596 \cdot (cr - 128) > 0 \text{ or } < 0 \end{cases} \tag{3.5}$$

It should be noted that the y variable of all terms is removed, i.e. the three variables (r, g, b) in Eq. 3.4 can be simplified using two variables (cb, cr) resulting in Eq. 3.5.

Regarding the gray class, a $x = (y, cb, cr)$ pixel can be classified as gray if (cb, cr) is in the neighborhood of (128, 128), i.e. using the Euclidean distance that measures the color difference between the test pixel and the reference color (128, 128), if the difference is less than the preset threshold, Th, the pixel is considered to belong to the gray class, as summarized by Eq 3.6.

$$x \in \text{grey}, \text{ if} \sqrt{(cb - 128)^2 + (cr - 128)^2} < Th \tag{3.6}$$

In [3] the gray color threshold value is obtained experimentally, as $Th = 11$.

By combining Eq. 3.3 and Eq. 3.5 the Cb-Cr plane is partitioned into four parts corresponding to each of the four color classes, as shown in Fig. 3.18.

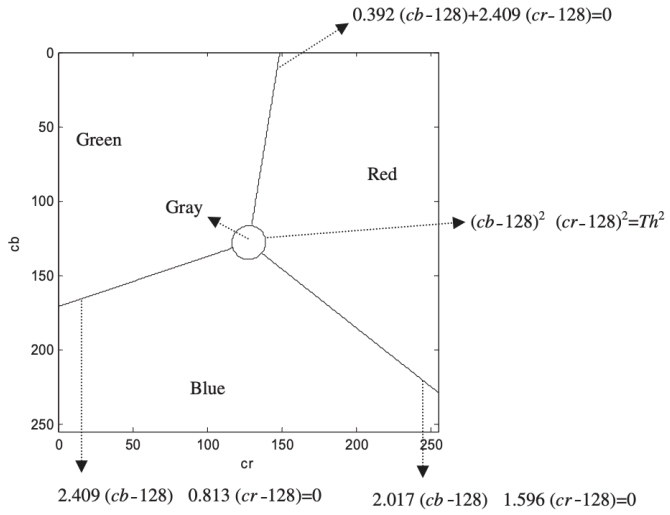


Figure 3.18: The decision regions of four-color classes.

After hill climbing, we obtain the clustering results in the CbCr plane, and

by inserting these clusters into the four color sections defined before we wish to categorize every cluster into one of the four classes.

For each cluster, we count the number of pixels falling within each of the four parts, which are denoted R#, G#, B#, and K# for red, green, blue, and gray, respectively, and based on this count, a cluster (C_i) can be categorized into one of the four classes using Eq. 3.7.

$$C_i \in argmax(R\#, G\#, B\#, K\#) \tag{3.7}$$

After categorization, the clusters belonging to the same class are merged, and, finally, each class is labeled and the colors of the original image with red, green, blue, and gray, respectively, are replaced as shown in Fig. 3.19. Note that after cluster merging, the number of clusters is ≤4 depending on the distribution of the original clusters.

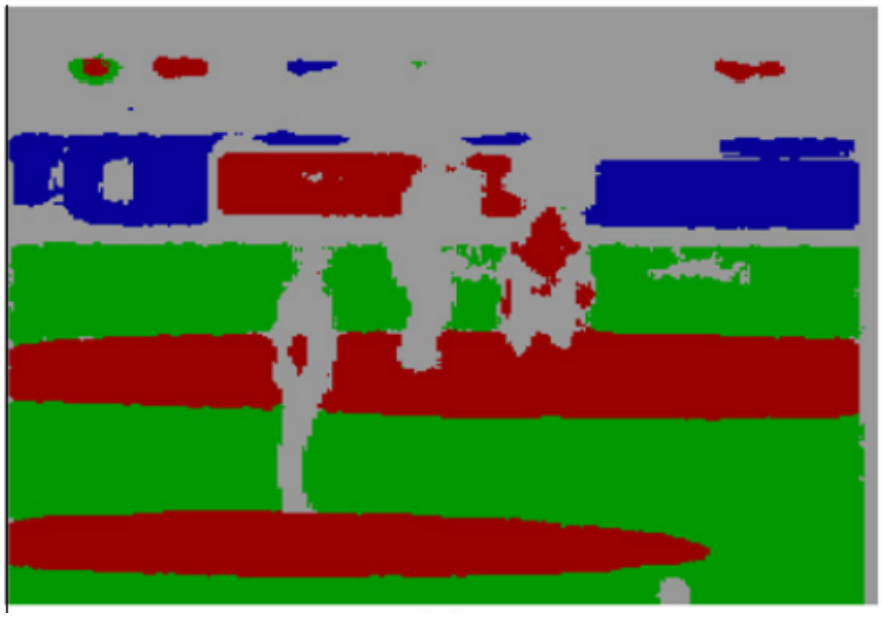


Figure 3.19: Labeled image based on cluster merging

Finally, the authors mention a method of eliminating some small regions (below a predetermined threshold) that have segmentation errors through a simple region fusion scheme to merge these small regions into the nearest large region. the final segmentation result after region fusion is shown in Fig. 3.20.

The method just described, as reported in the research paper, has been experimentally shown to outperform both the Gaussian Mixture Model (GMM)

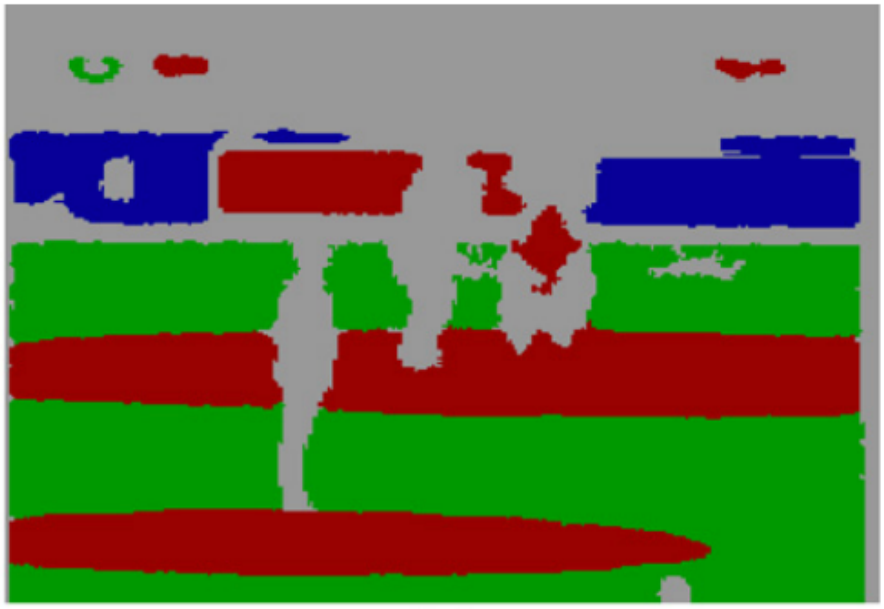


Figure 3.20: segmentation result after region fusing

method in terms of segmentation accuracy and computational efficiency, and the Mean-Shift method in terms of computational intensity while maintaining a comparable level of segmentation accuracy.

3.4.5 Our approach

We have implemented in C++ with the OpenCV library the generalised playfield segmentation method using colour features explained in the previous sections. In particular, we have implemented the local maximum clustering through the approximate hill-climbing scheme. This choice comes from our desire to have a faster system at the cost of a small loss of performance in segmentation.

Note that after several tests, in our implementation, pixels that are not classified, because they do not belong to any of the blocks found, are set to 0 (i.e. black).

In addition to implementing the method proposed by Hung et al. [3], we developed our own method of merging small regions.

The workflow of the merging method can be summarized as follows: for each of the four main colors (i.e., red, blue, green, and gray), we exploit the *connectedComponentsWithStats* method of OpenCV to find all connected regions of equal color. Then, for all regions that are composed of less than 0.05% of the

total pixels of the original image, the merge method inscribes a full ellipse in the region, enlarges the ellipse by 25 pixels (both width and height), and finds the dominant color in the area of the drawn ellipse without taking into account the small region. Finally, it sets the color of the region to the dominant color previously found.

Please note: you can see all the steps of the implemented method by demuting the comments inside the *segment* method in the *ColorFeaturesSegmentator.cpp* file.

3.5 Playing Field segmentation

After color features segmentation, each image displays a certain number of segments, as explained in Section 3.4. We shall now assume that the two main segments are the background and playing field. Therefore, we need to reduce the number of segments to just playing field and background so that we can subsequently paint over the segmented image the classified Players, as will be explained in Section 3.6.

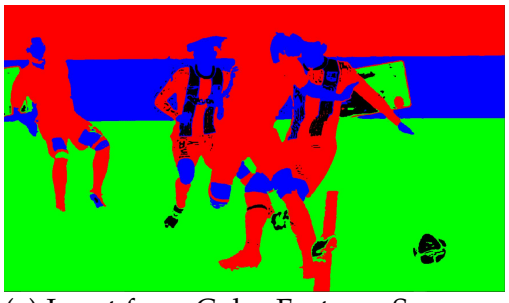
To this end, the image is segmented using k-means (k=2) with the pixel BRG values as feature vectors. As a consequence of our initial assumption, the two remaining clusters now represent the background and playing field. However, it is unclear which of the two clusters is the background and which is the playing field.

Therefore, to make this distinction, a heuristic function has been designed, assigning to each pixel on the image a cost which is the squared distance along the y-axis of the pixel w.r.t. the top of the image. The costs are summed for each pixel in the two clusters, and normalized by the number of pixels in each cluster. Then the playing field is defined as the cluster with the maximum total cost (and the background as the other). The heuristic procedure is shown in Algorithm 3.

Once the correct cluster for the playing field has been identified, the pixels are colored according to the mapping specified in the instructions.

An output example is shown in Figure 3.21.

Figure 3.21: Example of output of k-means + heuristic on the color features segmented image im15.jpg



(a) Input from Color Features Segmentator

(b) Output of k-means + heuristic

Algorithm 3 Playing field heuristic algorithm

Require: $maxNormalizedCost \leftarrow 0, playingField \leftarrow -1$
Ensure: the label of the correct cluster for $playingField$

```
for label ← 0, 1 do
  totalSquaredDistance ← 0
  clusterSize ← 0
  for pixel ← image do
    if pixel.label ← label then
      squaredDistance ← pixel.y * pixel.y
      totalSquaredDistance ← totalSquaredDistance + squaredDistance

      clusterSize ← clusterSize + 1
    end if
  end for
  if clusterSize > 0 then
    normalizedCost ← totalSquaredDistance/clusterSize
  else
    normalizedCost ← 0
  end if
  if normalizedCost > maxNormalizedCost then
    maxNormalizedCost = normalizedCost
    playingField ← label
  end if
end for
```

3.6 Player classification and merging

Once we have the Players and the playing field segmentation for our reference image, the last step is to paint the Players' segments over the segmented image, according to their team affiliation.

Before explaining the classification process that assigns the class to each Player, the last step is that of dropping some of the Players that are not located in the playing field. This step is necessary to reduce the false positives, since in the localization process some people in the background may have been mistaken for Players.

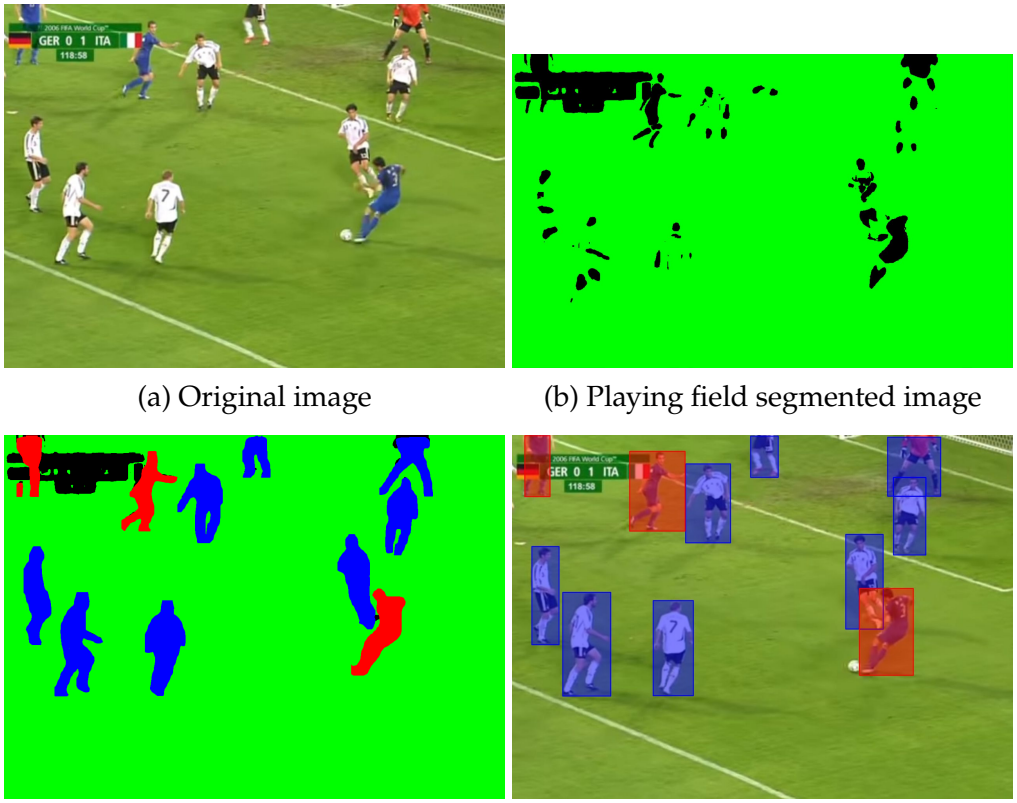
To this end, the bounding boxes of each Player are overlapped with the playing field segmented image in order to verify that at least 10% of the bounding box ROI is made of playing field pixels. If that condition is not met, then the Player is discarded.

For the classification, we have decided to use k-means (k=2) in order to choose which Players belong to Team 1 and which to Team 2. The flattened histograms of the color distribution within each segment of each Player have been used as feature vectors since the best discriminating factor for each Team is the color of the clothing.

Once the Team ID has been established, the newly colored segments are painted over the playing field segmented image. Finally, the masks are saved together with the bounding boxes' coordinates and classes in the required formats. Additionally, images displaying the detected bounding boxes are saved.

Outputs are shown in Figure 3.22. A summarizing schema is given in Figure 3.23.

Figure 3.22: Example of merging and classification for im1.jpg



(a) Original image

(b) Playing field segmented image

(c) Colored segmentation mask

(d) Image with detections

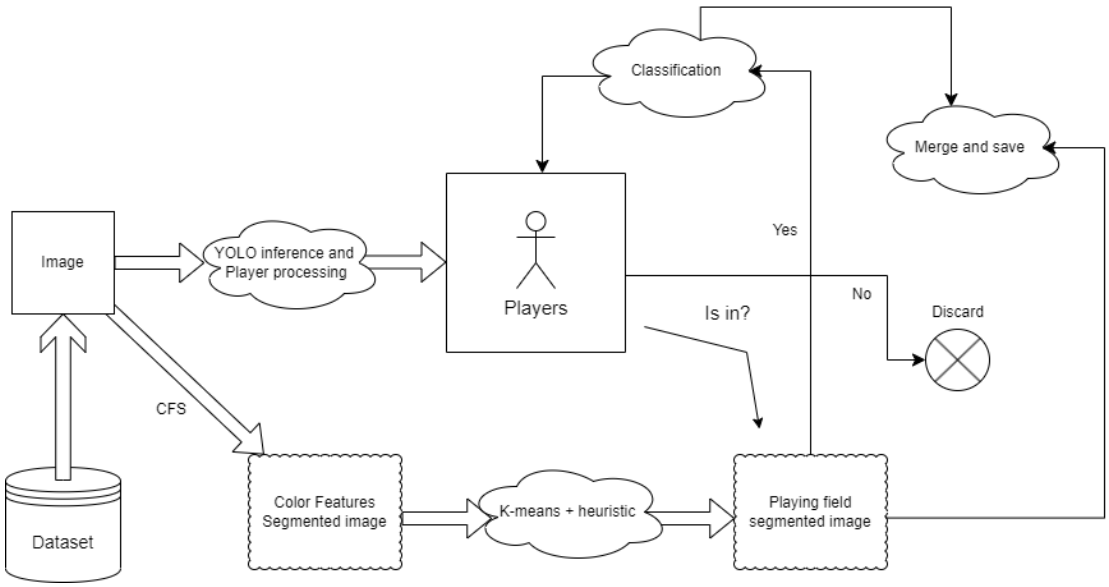


Figure 3.23: Summarizing schema of the process leading to the final outputs

4

Performance measurements

In this section, we discuss the performances of the models used. In particular, the Mean Average Precision was used for player detection, and calculated at IoU threshold 0.5. For player and playing field segmentation the Mean Intersection over Union was used.

4.1 Mean Intersection-Over-Union

The Intersection-Over-Union (IoU), also known as the Jaccard Index, is a commonly used metric in semantic segmentation. The IoU is the area of overlap between the predicted segmentation and the ground truth divided by the area of union between the predicted segmentation and the ground truth.

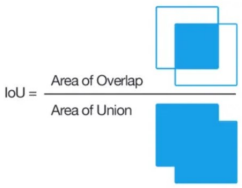


Figure 4.1: IoU calculation graphically [1]

This metric ranges from 0 to 1 with 0 signifying no overlap and 1 signifying perfectly overlapping segmentation. For binary (two classes) or multi-class segmentation, the mean IoU of the image is calculated by taking the IoU of each class and averaging them.

4.2 Mean Average Precision

Mean Average Precision (mAP) is a performance metric used for evaluating machine learning models. To better understand what mAP is we need to start by considering the Precision P and Recall R.
The Precision P measures the proportion of predicted positives that are actually correct. It's computed by considering the True Positives out of the total detections and the value ranges from 0 to 1.

$$P = \frac{TP}{TP + FP} = \frac{TP}{TotalPredictions} \tag{4.1}$$

The recall R measures the proportion of actual positives that were predicted correctly. It computed as the True Positives out of all Ground Truths and it also ranges from 0 to 1.

$$P = \frac{TP}{TP + FN} = \frac{TP}{GroundTruths} \tag{4.2}$$

Than considering the Average Precision (AP) as the area under the precision-recall curve, we can define the mAP as the average of AP over all detected classes.

$$mAP = \frac{1}{n} * \sum_{i \in class} AP_i \tag{4.3}$$

Where n is the number of classes.

4.3 Our results

Considering the metrics mentioned above, it now follows the analysis of the results we obtained in the different approaches we've seen.

4.3.1 Detection using OpenCV

As we have seen in Section 3.1, three algorithms were implemented to solve the detection problem. The metric we considered is mAP with an IoU threshold of 0.5.

1. **HOG + SVM:** While it had been one of the best algorithms prior to the advent of deep learning, even if in some pictures it's able to discreetly

detect the players, on average it fails to detect even one person correctly. In Figure 4.2 we can see an example of the localization.

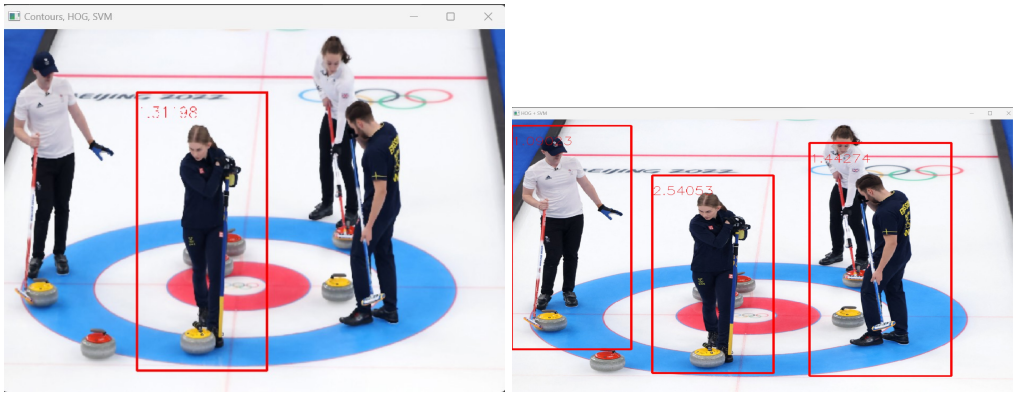
Figure 4.2: Comparison between examples: in (a) we still have a false negative and in (b) a lot of false positives

(a) Example of good localization (im14) (b) Example of bad localization (im09)

2. **Haar Cascade:** This approach gave us the worst results. It may be mostly because the Haar features work better when there are some fixed characteristics that it can rely on, like when detecting faces, hands, or singular body parts. The large intra-class variance of the players might have played a big role in its failure.

3. **Contour, HOG and SVM:** This approach in terms of performance is in the middle between the previous two, though it is clearly heavily leaning on the worse side. The idea of exploiting edges for better detection accuracy has clearly not yielded a positive outcome. An example is shown in Figure 4.3.

Figure 4.3: In this comparison we see how with the same image im14 (b) gives good localization results, while (a) doesn't



(a) Contours, HOG and SVM (b) HOG+SVM

Since the results using only OpenCV were severely under-performing, as seen in Table 4.1, we decided to raise the complexity of our system by focusing on the use of CNNs, as explained in Section 3.2.

Table 4.1: mAP results for OpenCV algorithms

Image	HOG + SVM	Haar Cascade	Contours + HOG + SVM
img1	0.167	0.0182	0.227
img2	0.333	0.295	0.091
img3	0.091	0	0
img4	0	0	0
img5	0	0	0
img6	0	0	0
img7	0.045	0	0.045
img8	0.409	0.273	0.273
img09	0	0	0
img10	0	0	0
img11	0.3	0	0.091
img12	0.197	0	0
img13	0.273	0	0
img14	0.273	0	0
img15	0	0	0
mean mAP	0.139	0.039	0.048

4.3.2 Detection and segmentation after YOLO

Now we will focus on the analysis of the results of the detection and segmentation after the predictions given by the four YOLO models.

Again, for the detection mAP with IoU threshold of 0.5 is considered, while for the segmentation mIoU is considered.

- **yolov8s-seg.pt**: it is the best-performing model among the four. We believe this is due to the intense and well-developed training which takes into account many different scenarios. The network has in fact learned to segment people very well, and the results are noticeable in Table 4.2.
- **yolov8s-seg-scratch_player_referee.pt**: as expected, it is the worst-performing model. It is almost surely due to its' relatively fast training of just 25 epochs, which is generally very low for training deep networks from scratch, especially for datasets almost as big as 10,000 images. The brand new Player class for this dataset has evidently not been figured out sufficiently well. The dataset itself did not represent well what was going to be seen in the test set, due to the lack of available data on sports other than soccer, basketball, and tennis.

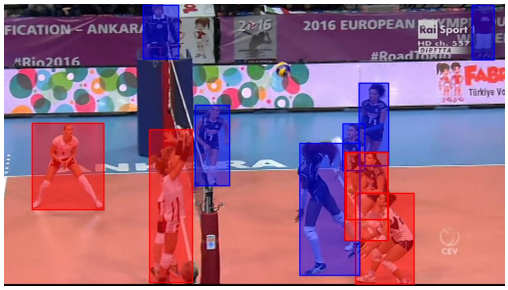
  Results are shown in Table 4.3.
- **yolov8s-seg-pretrained_player_referee.pt**: for this model, performances can be located in the middle between the former two. This is due to the pre-trained nature of the network, which gets a head-start in performance

compared to the version from scratch. Yet the same considerations for the dataset, player class, and training can still be made valid. Despite this, we can see the potential for this network in one example in particular, shown in Figure 4.4. The full results are shown in Table 4.4.

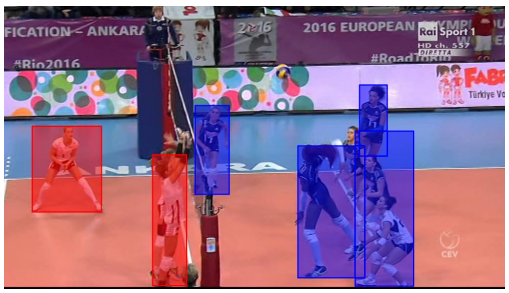
- **yolov8s-seg-pretrained_person_referee.pt**: again we have mid-performances, and we believe the reasons are ultimately the same as the previous model. The person class was already well represented by the pre-trained model, and this additional and too small training ended up tarnishing its' previous knowledge. Results are shown in Table 4.5.

By looking at all the tables, we can see that between mAP and mIoU, the most discriminating metric for the performance evaluation is mAP. The mIoU does not change significantly possibly because the playing field and the background are usually comprised of more pixels than the players; therefore, bad segmentation, and ultimately detection, of players may not hinder the mIoU performance as much, provided a good playing field segmentation.

Figure 4.4: An example of the detection output given the predictions of two models: the referee and the background person are correctly ignored by (b), as wished for, but are false positives for (a). This has led to the wrong team classification of some instances on the right-most portion of (a), which does not occur for the predictions of (b).



(a) Predicted bounding boxes given the segments of yolov8s-seg.pt



(b) Predicted bounding boxes given the segments of yolov8s-seg-pretrained_player_referee.pt

Table 4.2: yolov8s-seg.pt results

Image	**mAP**	**IoU**
img1	0.982955	0.960055
img2	0.581818	0.891395
img3	0.689394	0.730014
img4	1	0.978624
img5	1	0.875572
img6	0.666667	0.575312
img7	0.613636	0.958556
img8	0.772727	0.735738
img9	0.5	0.829129
img10	1	0.902431
img11	0.681818	0.704125
img12	0.818182	0.80664
img13	1	0.688353
img14	1	0.904524
img15	0.833333	0.928529
-	Mean mAP=0.809369	Mean MIoU= 0.831267

Table 4.3: yolov8s_seg_scratch_player_referee.pt results

Image	**mAP**	**IoU**
img1	0.272727	0.946378
img2	0.484848	0.890464
img3	0.681818	0.657495
img4	0	0.959828
img5	0.272727	0.660354
img6	0.136364	0.366628
img7	0.386364	0.937632
img8	0.772727	0.735743
img9	0	0.780613
img10	0.272727	0.756098
img11	0	0.564211
img12	0	0.690383
img13	0.272727	0.688353
img14	0.272727	0.897117
img15	0.136364	0.793104
-	Mean mAP=0.264141	Mean MIoU= 0.752472

Table 4.4: yolov8s_seg_pretrained_player_referee.pt results

Image	**mAP**	**IoU**
img1	0.560606	0.956395
img2	0.590909	0.89117
img3	0.818182	0.73399
img4	1	0.976346
img5	0.5	0.675977
img6	0	0.501427
img7	0.727273	0.922449
img8	0.772727	0.735747
img9	0	0.780613
img10	0.272727	0.739796
img11	0.333333	0.662242
img12	0	0.689761
img13	1	0.685572
img14	0.772727	0.902542
img15	0.272727	0.750331
-	Mean mAP=0.489899	Mean MIoU= 0.773624

Table 4.5: yolov8s_seg_pretrained_person_referee.pt results

Image	**mAP**	**IoU**
img1	0.655844	0.958794
img2	0.681818	0.887519
img3	0.818182	0.662673
img4	0.424242	0.978291
img5	0.818182	0.844333
img6	0.227273	0.56009
img7	0.386364	0.92253
img8	0.772727	0.735716
img9	0	0.780613
img10	0	0.739843
img11	0.454545	0.690372
img12	0.590909	0.774235
img13	1	0.680995
img14	0.272727	0.899467
img15	0.545455	0.811626
-	Mean mAP=0.509885	Mean MIoU= 0.79514

5

Conclusions and Future Works

5.1 Conclusions

The goal of this project was to identify and track players on sports teams and generate semantic information about the game, such as highlighting the boundaries of the playing field and recognizing the players' dynamics.

We know this kind of analysis is important to identify patterns, correct players' behaviors and so much more. We tried to address the problem from various points of view, developing different solutions.

Our first approach to use classic algorithms for solving the detection problem, as explained in Section 3.1, has led to poor results, as we have seen in Section 4.3.1, due to the highly complex nature of the task, together with the large variance within the classes that we were trying to detect.

Our second approach involved the use of deep learning tools, as explained in Section 3.2. In this case, as we have analyzed in Section 4.3.2, performances improved significantly, yet our own custom trainings of the networks did not meet the requirements to reach the pre-trained version in terms of quality of the results.

5.2 Future Works

Now that it has been established the superiority of deep learning based techniques over the classic algorithms for solving this complex task, any future

improvement should focus on the expansion of the dataset and prolongation of the training.

In particular, we have seen the potential of the Player-Referee dataset for the resolution of this task, as discussed in Section 4.3.2, where ideally the network is able to learn the difference between any person and a player and a referee, for top-level accuracy. However, reaching this goal requires more data which we were not able to get our hands on, and more extended training, which we were not able to accomplish due to the temporal limitations imposed by the Google Colab GPU environment.

In conclusion, we believe that if these two conditions were met, then our software would be able to solve the task with possibly state-of-the-art performances.

5.3 Individual contributions

- Sara Farris 2075215: ideas and implementation of the first stage of the system, that is the OpenCV algorithms for detection. Number of working hours: 70
- Dario Mameli 2087636: ideas and implementation for the use of YOLO as a segmentation network and related python files. Datasets assembling, training of the networks, and detection and segmentation of players exploiting the outputs of YOLO. Merging of playing field segmentation and player detection and segmentation. Number of working hours: 120.
- Alberto Dorizza 2075216: ideas and implementation for the playing field segmentation. Help in the training process for the networks. Development of the performance measurements code. Number of working hours: 90.

References

[1] John Canny. "A Computational Approach to Edge Detection". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-8.6 (1986), pp. 679–698. DOI: 10.1109/TPAMI.1986.4767851.

[2] Alfred Haar. "On the Theory of Orthogonal Function Systems". In: (1909), p. 37.

[3] Mao Hsiung Hung et al. "Generalized playfield segmentation of sport videos using color features". In: *Pattern Recognition Letters* 32 (7 May 2011), pp. 987–1000. ISSN: 0167-8655. DOI: 10.1016/J.PATREC.2011.01.022.

[4] E J Pauwels and G Frederix. "Finding Salient Regions in Images Non-parametric Clustering for Image Segmentation and Grouping". In: *Computer Vision and Image Understanding* 75 (2), pp. 73–85. URL: http://www.idealibrary.comon.

[5] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. arXiv: 1506.02640 [cs.CV].

[6] Laurent Thomas. "Computer vision and data-analysis solutions for phenotypic screening of small model organisms". PhD thesis. Jan. 2021.

[7] C. Tomasi and R. Manduchi. "Bilateral filtering for gray and color images". In: *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*. 1998, pp. 839–846. DOI: 10.1109/ICCV.1998.710815.

[8] P. Viola and M. Jones. "Rapid object detection using a boosted cascade of simple features". In: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*. Vol. 1. 2001, pp. I–I. DOI: 10.1109/CVPR.2001.990517.