Computer Vision - Sport Video Analysis Project

Authors

Alberto Zerbinati—2053861
Marco Sedusi—2055716
Marco Calì—2079446

Introduction

This project aims to develop a computer vision system for **analyzing sports video footage**.
The primary goals are to identify and track players on the field, segment the playing area, and classify players based on their teams. The system needs to be adaptable for various team sports like soccer, basketball, rugby, etc.
The performance will be evaluated using metrics like mean Average Precision (mAP) for player localization and mean Intersection over Union (mIoU) for segmentation tasks.
The project is developed in C++ using the OpenCV library, with the use of Python/Pytorch for the deep learning component.

Implementation

We identified three main tasks to carry out the project:

- **Players Detection/Localization**: given a sports image, find the bounding boxes around each player inside the playing field;
- **Players Segmentation**: isolate individual players from the rest of the image content, starting from their bounding boxes;
- **Field Segmentation**: identify and isolate the playing field from the rest of the image.

Other tasks to complete the project were:

- **Team Specification**: classify the detected players into their respective teams;
- **Metrics Computation**: implement metrics like mean Average Precision (mAP) for player localization and mean Intersection over Union (mIoU) for segmentation tasks to evaluate system performance;
- **Pipeline Definition**: define and document the complete pipeline starting from image input, pre-processing, player and field segmentation, team classification, and finally the computation of metrics.

By breaking down the project into these specific tasks, we were able to simplify and clarify the development process, which nonetheless proved *extremely* challenging.

Player Detection (Alberto Zerbinati)

Various **ML/DL methods** were considered for this task.

HOG Features + SVM Binary Classification

The method proposed in [1] by Dalal and Triggs appeared simple and effective on paper, focusing on Histograms of Oriented Gradients (HOG) for human detection. However, it performed poorly in our project. The high rate of false negatives and positives was unacceptable. Even after switching to a more complex MLP classifier, the model still wasn't capable enough. Nonetheless, it served as a quick **baseline** for our subsequent tests.

It also served to see how to implement a sliding window approach based on a binary classifier, which will turn out to be our final solution (with a different model).

After seeing the poor results from our initial attempts with hand-crafted features, we realized a change in approach was necessary. Two options caught our attention:

- Fine-tuning an object detection model specifically for player detection;
- Scaling up the binary classification approach.

SoTA Fine-Tuning

Following this official PyTorch tutorial, we managed to create a model that could segment pedestrians on the Penn-Fudan dataset [2]. Transitioning from this to player detection seemed feasible. We even planned to train the model on human detection using the "person" split of the COCO dataset [3].

However, according to guidelines from the project forum (link), such an approach wouldn't be acceptable. The base model we considered for fine-tuning (Mask R-CNN [4]) was already pre-trained on human detection. Therefore, fine-tuning it further on player detection seemed to go against the project's rules.

The same restriction applied to another tutorial we found on fine-tuning YOLO [5] on a custom dataset. Given these limitations, we decided not to pursue these approaches.

> **Note**
>
> We'd like to raise a point for clarification. Close to the deadline, we discovered that other groups had interpreted the forum post differently and believed it was acceptable to fine-tune a pre-trained state-of-the-art object detection models like YOLO, while we believed it to be prohibited.

YOLO Reimplementation

Next, I worked on what I found to be the hardest and longest task of the project for me. I started with the YOLO paper and built its setup in Pytorch (model, loss, dataset, train/valid loop). The tough part was figuring out the Loss function:

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2$$

I used a tutorial to guide me and tweaked it to detect two classes: "background" and "person." To speed things up and avoid overfitting, I simplified the network's design. My training set wasn't huge; it was the "person" split from COCO, with about 5k images. Here are some of the WandB curves from the training:

As I feared, while the training loss would decrease during training, the validation loss would remain high. The model overfit the training set no matter what simplification to the architecture I tried. There's also to notice that the training time was considerable, and so this whole experiment took lots of energy and time (~1 week / 30h) without providing the expected results.

Vanilla CNN for Binary Classification

Excluding the SoTA object detectors, and seeing the poor baseline ML results, we moved to our next idea: the advantage of Deep Learning over Machine Learning is largely represented by its capability for representation learning. A well-trained deep model can learn to represent the input in a manner suitable for various end-tasks, such as binary classification. This concept led us to our next solution: training a fully custom CNN with a binary classification head.

This approach had the same advantage as the HOG + SVM method in terms of simplicity in implementation and training but delivered much better results.

The model was trained to differentiate between crops of humans and background scenes. Initially, we used a generic dataset of human crops available online and observed encouraging results. However, to improve performance for our specific task, we realized that a custom dataset focused on sports images was necessary.

ResNet Finetuning for Binary Classification

Before settling on our final approach, we decided to experiment with a pre-trained classification CNN. We chose ResNet34 due to its inference speed and performance. Fine-tuning a pre-trained network has the benefit of speeding up the training process: you only need to train the classification head and perhaps adjust a few weights from the pre-trained model. This approach can also yield excellent results because the initial model already has a good understanding of key features.

In our case, switching from a custom CNN to a pre-trained ResNet34 led to a modest improvement in performance and also sped up the training time. Therefore, we included it in our final solution.

Custom Dataset

We created a custom dataset comprising 108 sports images, including soccer, rugby, ice hockey, and basketball. The goal was to train the model to distinguish players from the field and other background elements. The images we selected offer a good range of field and team colors as well as player positions like running and jumping. However, there are some limitations in terms of camera angles. **We primarily chose television views** for ease of automatic dataset processing. While this made annotation easier, it limited the model's performance in handling more dynamic or unconventional angles. This will be discussed later in the Limitations section. We also have versions with more images, more sports, and more variance in aspects like lighting and camera views, but we weren't able to improve metrics with them.

For dataset annotation, we took an automated approach:

- We used YOLOv3 to identify people in the images, setting the confidence threshold at 0.9. These bounding boxes served as positive samples;
- For negative samples, we randomly cropped sections from the same images, taking care to minimize overlap with positive samples.

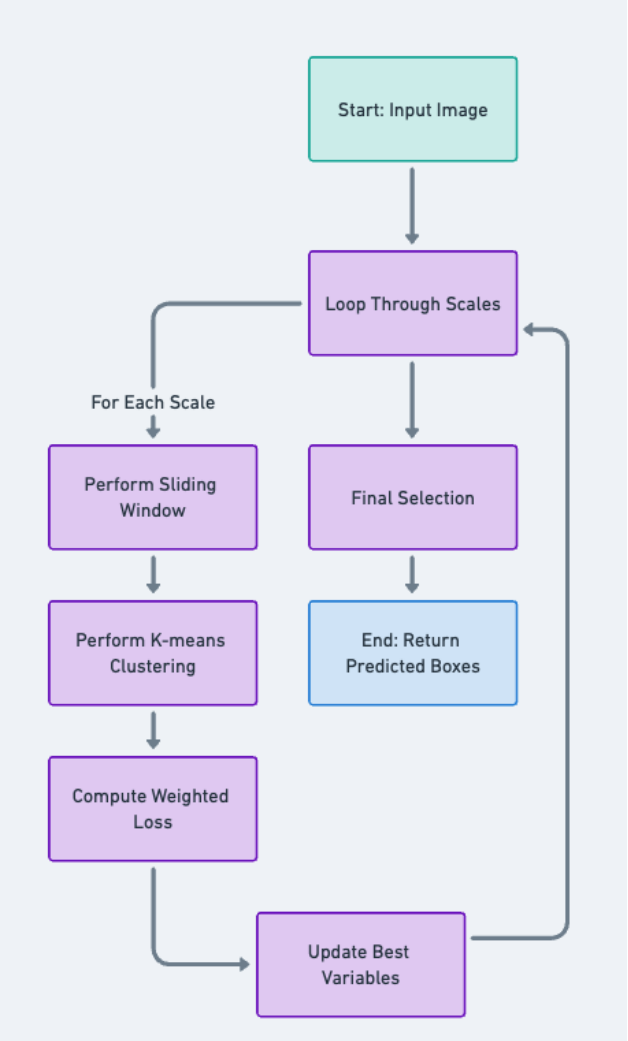
This gave us around ~750 crops (both positive and negative, train/valid/test split). We trained the model over 20 epochs.
The automation scripts for dataset creation can be found in the data/data_processing folder.

Our trained model reached an accuracy of ~70% on this dataset: not impressive, but the easy ways to improve the model didn't appear to transfer directly to the final metrics, so we didn't spend too much more time on this, since it already took a lot.

The Final Players Detection Technique

Here we outline the complete Player Detection Technique.

First, we imported the model into C++ using Torch Script, achieved through the Tracing technique outlined in the PyTorch documentation. A crucial step was to ensure that image pre-processing was consistent between the Python and C++ code. Specifically, we resized images to 100x100 pixels and performed normalization. Initially, skipping the normalization step led to NaNs and Infinities when we imported the model and performed inference in C++.



Here's an overview of our Player Detection Pipeline:

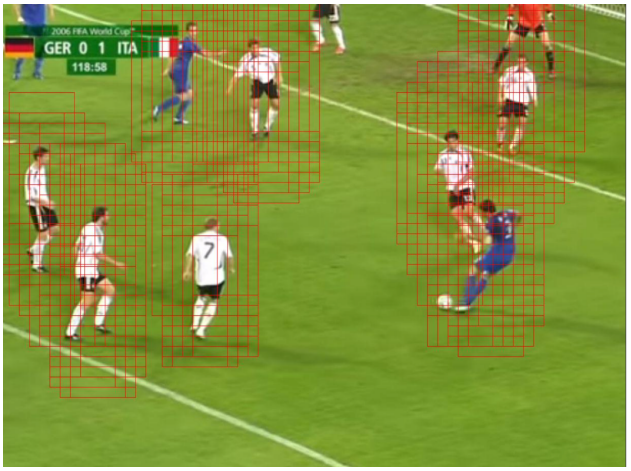
- **Scale Handling**: we start by considering multiple scales of the input image, as players can appear at varying sizes;
- **Sliding Window**: on each scale, we apply Sliding Window. The window size is determined by the image's dimensions and the current scale, while the step size is fixed at 50 pixels;
- **Clustering Positives**: the model often returns positive detections for windows that only partially contain a player. As a result, we found that each player typically corresponds to a cluster of windows;
- **Loss Function on Clusters**: we use k-means clustering on spatial features (the centers of the positive detections) and calculate a weighted loss function on its

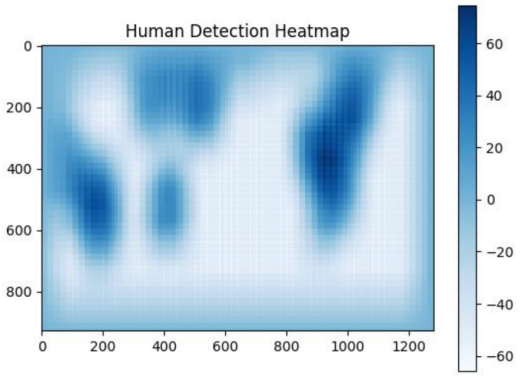
output. Since the number of players in an image is unknown, and k-means' cost favors a higher value for k, we run k-means for various values of k, penalizing higher ones. We also add a penalty for larger scales, which the model seemed to favor. This weighted loss acts as a regularization method.

- **Final Output**: We return the clusters of windows that result in the lowest weighted loss.

Our approach is effective, but with some limitations. For instance, one is scale uniformity: all players are detected at the same scale, which isn't ideal for varying player sizes. This remains an area for future improvement.

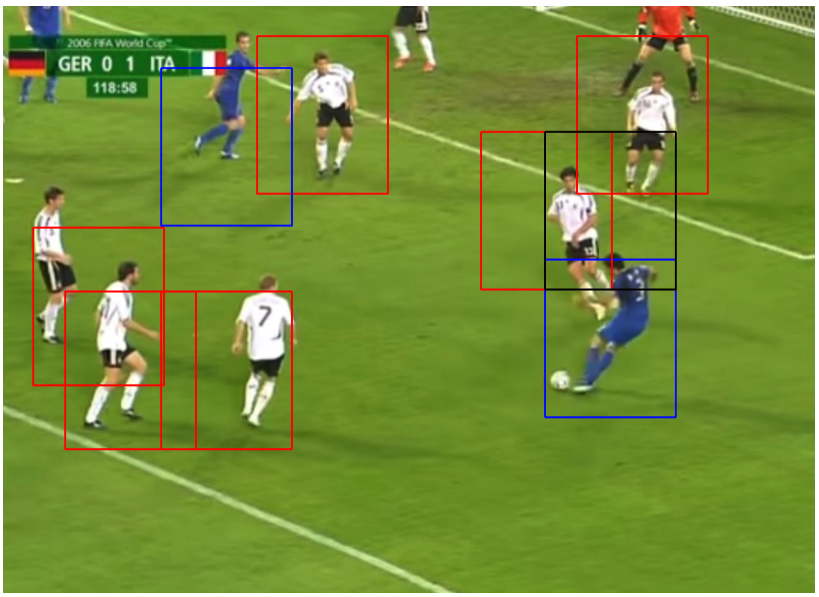
Here is a result when applying a sliding window without any clustering or regularization: it's hard to understand the boundaries of single players when they are close to each other.



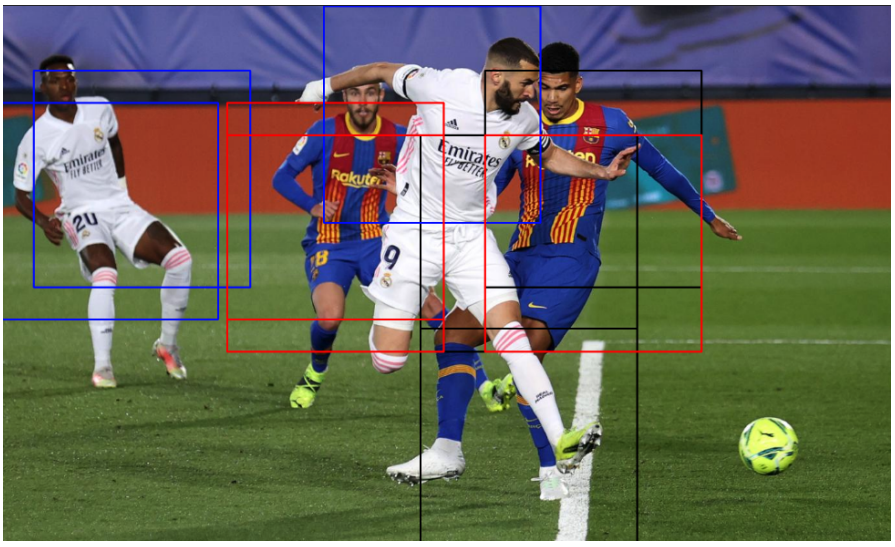


This shows that all the relevant areas (players in the field) are spotted. Going from this representation to the final bounding boxes is not trivial.

Our attempt, with k-means clustering and weighted loss, yields these final bounding boxes (colors specify the team, done at later stages):



Unfortunately, the current regularization technique is not always good enough for accurate scale detection. We were restricted to employing windows at only two scales: 0.8 and 1.2. This is not sufficient for images with very large players, where a scale of 2 or 3 would be needed. The main issue lies in our inability to appropriately calibrate the regularization parameters within the weighted loss function. When multiple scales are available, the model displays a strong bias towards the largest scales. This leads to inaccuracies in the output bounding boxes, both in terms of scale and placement.



An example of bad bounding boxes: the scale is too small, and there is high degree of overlapping. however, no "false negatives" are present (all the "players' area" is covered by the bounding boxes).

As it stands, the Player Detection pipeline is **best suited for images shot from a standard broadcast view, where players are spaced well apart and are roughly the same size**. The algorithm struggles with more varied scenarios, making this a core area for future improvements.

Player Segmentation (Marco Sedusi)

This task was about segmenting the players' area from a given bounding box.
Several methods were considered. Initially, we attempted classical approaches, but they quickly yielded poor results.
We began with a simple segmentation by thresholding but due to the wide variety of sport situations, it proved challenging to cover all possible scenarios with this basic method.
Nonetheless, this classical approach may still be useful in subsequent steps.
Secondly, a blob detection method with Gaussian and Laplacian filters with different kernel size was performed, but also in this case it was very tricky to generalize due to the different scale presented from the test images.

Afterward, we delved into more complex approaches that demanded significant implementation efforts. Nevertheless, the results obtained were better but not as satisfactory as we had hoped for.
Based on the observations mentioned above, we realized that better results could only be achieved through the combination of multiple methods.

GrabCut

We have considered the GrabCut algorithm [6]. In general, it is an interactive Foreground Extraction method using iterated graph cuts that allow us to segment an object of interest from an image with a small user interaction.
Here's how GrabCut works:

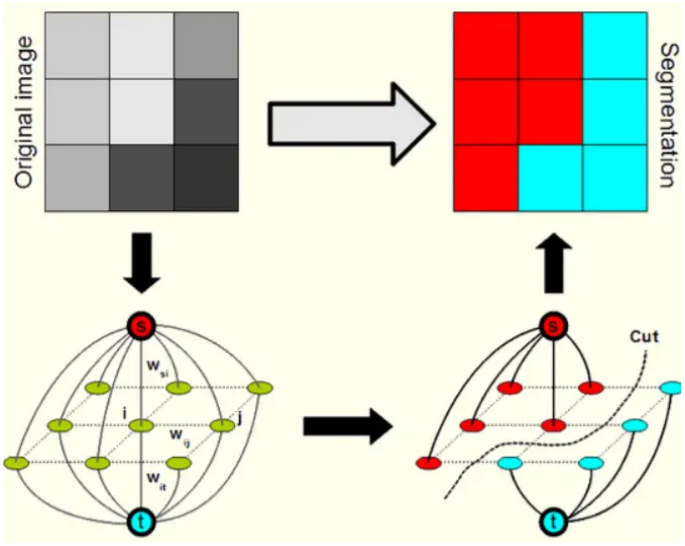
(1) Initially we define a Bounding Box around the foreground region (considered as hard-labeling which means they won't change in the process). Everything outside this rectangle will be taken as a sure background, instead everything inside the rectangle is unknown (this Bounding Box should include both foreground and background regions). It labels the foreground and background pixels (1-3 values for foreground, 0-2 for background).

(2) A Gaussian Mixture Model is used to model the foreground and background. We can calculate color statistics (mean and covariance) for both the foreground and background regions within the bounding box.
We initialize two Gaussian Mixture Models (GMMs) to represent the color distributions of foreground and background pixels.

(3) A generic graph structure is built from the pixel distribution, where nodes are pixels. We consider two special nodes, the Source node and the Sink node where every foreground pixel is connected to Source node and every background pixel is connected to Sink node and then we connect neighboring pixels with edges.

(4) The weight of edges connecting pixels to Source node/Sink node are defined by the probability of a pixel being foreground/background, instead, the weights of other edges are defined by pixel similarity. Indeed if there is a large difference in pixel color, the weight will be low.

(5) We use graph cut algorithms (such as max-flow min-cut) to find the optimal cut that separates the graph into two disjoint sets: the foreground and the background. The cost function is the sum of all weights of the edges that are cut. After the cut, all the pixels connected to Source node become foreground and those connected to Sink node become background. We update the GMM parameters (mean and covariance) for the foreground and background regions based on the pixels assigned to each segment after the graph cut and iterate the process until it converges.



GrabCut is a good starting point for our task, but we need to perform some modifications:

(1) We need to have zero user interaction, indeed for this task the object where we start is the bounding box received from the people-detection, hence no user interaction is involved.

(2) Secondly, we need to establish a general criterion for intelligently selecting the crucial rectangle for segmentation in order to always take the subject.

One possible way to solve these issues is based on bounding boxes' aspect ratio considerations.

Analyzing the bounding boxes returned by the player detection step, we drew the following heuristics:

- If the bounding box has a 4:3 aspect ratio, we notice high variance in the player's position inside the box, so approximately 90% of the image surface is considered as a rectangle for Grabcut. The starting points are set to (width/8, height/20).
- For 16:9 bounding boxes, a smaller surface is considered, since, in most cases, the person is in a central position. The starting points are set to (width/4, height/6).
- Finally, for 1:1 images the rectangle is positioned between the two previous cases. The starting points are set to (width/6, height/9).

Skin Detection

As mentioned before, performing only one method is not enough in order to achieve an acceptable result, hence, we have decided to also perform a simple skin-detection method in order to add small components relating to the person's skin.
The skin-detection method performs as follows:

(1) The original image is converted from the BGR color space to the HSV (Hue, Saturation, Value) color space. HSV is often used in computer vision tasks because it separates the color and intensity information, making it easier to work with color-related features.

(2) Define lower and upper bounds for skin Color in HSV:
Scalar lowerBound(0, 30, 70);
Scalar upperBound(27, 150, 220);
These values are based on the Hue, Saturation, and Value respectively in HSV color space.

(3) Next, we create a binary mask for skin Color, where the **inRange** function is used to create a binary mask **skin** that identifies pixels in the **img_HSV** matrix that fall within the specified HSV color range. Pixels that match the skin color criteria will be white, while others will be black in the skin matrix.

(4) Perform Morphological operations in order to remove small noise with the Opening operation, instead Closing operation is used to fill small gaps in the detected regions. We have decided to define an elliptical structuring for the kernel, which defines the neighborhood for these operations.

(5) For the final step we have used the **bitwise_and** function in order to combine the original image with the **skin** mask obtained. This operation keeps only the parts of the original image that correspond to the white regions of the **skin** mask, effectively highlighting the detected skin regions while blacking out other areas.

The Final Player Segmentation Technique

Now we have two different masks, one obtained from the GrabCut algorithm (person segmentation without some details) and one from the skin-detection (image with only the skin detected), it is therefore enough to merge the two masks in only one.
After the merging we need to clean up the image, because some details from the HSV color space are still present in the final image, hence we use the original image to restore the real value for affected pixels.

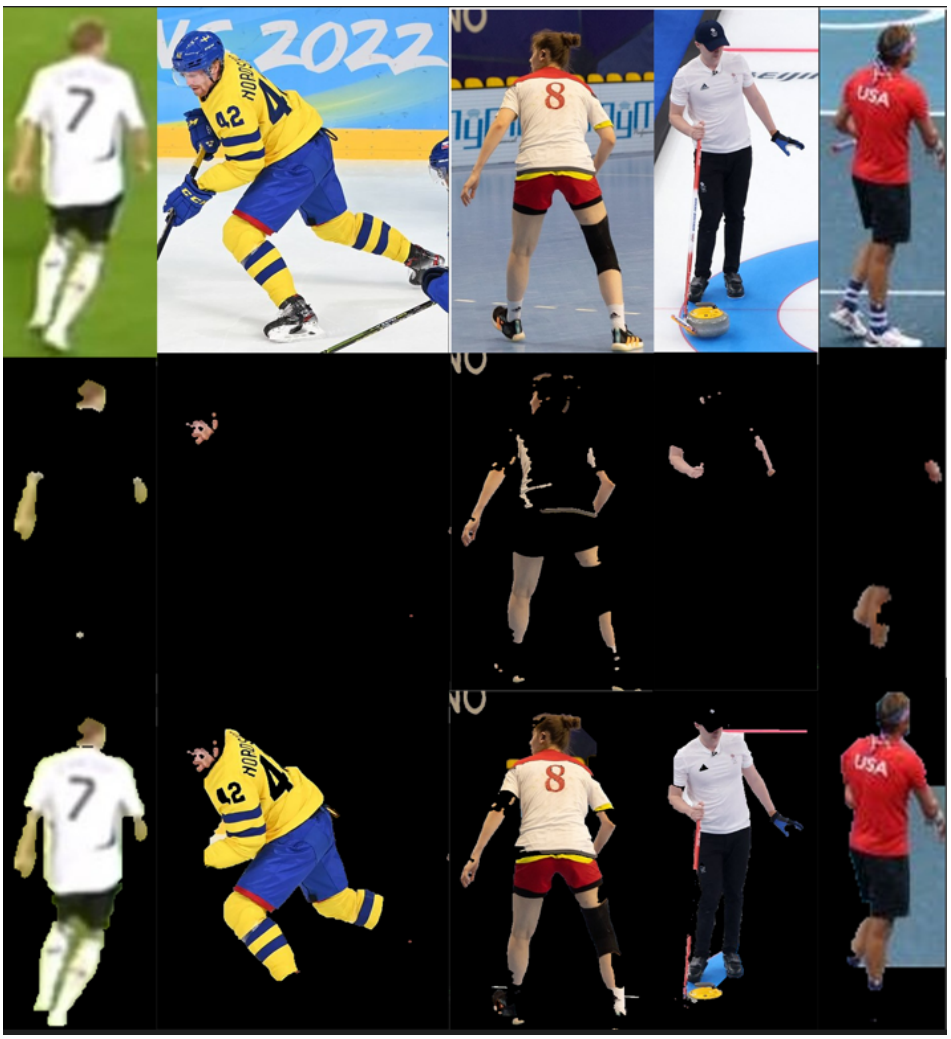


Figure:the first row shows the original image, the second row shows the skin-detection and finally the third row shows the Grabcut and skin-detection merged result.

Remarks

The proposed segmentation technique returns acceptable results in almost all the test images, on ideal bounding boxes.
The crucial weakness of the method is its reliance on the "quality" of the bounding box received, because if we receive a bounding box that is widely displaced respect the true one, the GrabCut algorithm performs only partially and the overall segmentation performance decreases.

One way to enhance people-segmentation would be to select a better starting rectangle for GrabCut, through more intelligent heuristics. For instance, to try to recognize the player's head inside the bounding box or try to recognize the number in the player's shirt. These would be great starting points for the initial rectangle.

We observed that GrabCut often included parts of the field in the player mask. To address this, we experimented with removing the field from the segmentation, as detailed in the subsequent image. However, this turned out to be less critical. As per your pipeline design,

the field segmentation is considered more reliable. Therefore, any field portions incorrectly included in the player segmentation would be overridden and removed in the final output segmentation masks.

Figure: Better results for segmentation with the K-means algorithm for finding dominant colors for the field received from the pipeline at the initial step. First column shows the original image, the second column shows the Grabcut and skin-detection result, the third column shows the inverted mask and finally the fourth column shows the result after K-means algorithm application.

Methods tested but not implemented

1) Hough-transform [7]: this is the simpler technique tested to improve the final result, the idea was to find the head of the player, but unluckily this method has important limitations, indeed it does not handle scale and rotation variations of shapes, a key concept for our task.
   We have also tried to implement the extension of this technique (Generalized Hough Transform) but also with this generalization we can not handle the many different scenarios and different sizes proposed.

2) Viola-Jones [8]: Viola-Jones uses a fixed-size sliding window approach to scan through the bounding box at different scales. This means that it may miss objects that are too small or too large relative to the chosen window size. To detect objects at

various scales, the algorithm has to be applied multiple times, which can be computationally expensive.
Also, this method is sensitive to variations in pose, lighting conditions, occlusions and allows you only to find the front faces. These limitations are crucial for our task, hence we have decided to discard this approach.

3) OCR: Optical Character Recognition (OCR) is a widely used approach for detecting and recognizing text, including numbers, within images. This is useful in order to detect the number in the t-shirt of the player.
Unfortunately also this approach has some significant limitations:
   - Difficulties with highly stylized or distorted text.
   - It doesn't work with numbers with small resolution.
   - OCR can be computationally expensive.

Team Specification (Alberto Zerbinati, Marco Sedusi)

We used color recognition to identify which team each player belongs to. This method also helped us remove false positives. For example, referees and fans usually wear different colors than the teams, so they're automatically filtered out.
As a note, we didn't try to identify goalkeepers or players in special uniforms within the same team. To do that, we'd need to consider their positions on the field, not just their colors, and would have required us additional time on the task. It's part of the possible future improvements.

Our method uses k-means clustering in the color space to find dominant colors in the image, with k the number of colors returned.
In nearly all cases, the dominant color vector returned by k-means corresponds to the field colors. For this reason, we discard the most present color and choose the second one. More specifically, since the bounding boxes passed to the team specification algorithm are already segmented (colored are of the player, surrounded by black where the field was), we can simply remove the black detection and return the other color.

With our dominant color function defined, the Pipeline is in charge of creating an std::map containing colors as keys and counters as values. Each bounding box (i.e. each possible player) votes for a color, and we use a threshold to cluster together similar colors. Once each bounding box has voted, we extract the two most frequent colors as team colors and assign each bounding box to the corresponding team. If a bounding box voted for a color that was not selected as a team, it is assigned a no-team label.

Field Segmentation (Marco Calì)

Chromaticity-Based Analysis Method

For field segmentation, the idea was to first solve a specific case, and then generalise to other types of fields of different sports. An interesting and fully automatic approach for soccer is suggested in [9], which yields very good results. This algorithm is also able to perform excellently in other sports where the field is green, such as rugby.

1. The algorithm, thanks to the opening preprocessing operation, is able to include the white lines of the field in its segmentation. We assumed that the white lines size is proportional to the size of the image, thus deriving a circular structuring element of size:

   $$d_e = \boldsymbol{\alpha}_e \sqrt{H^2 + W^2}/100$$

   where H and W are respectively the height and width of the input image.
   This assumption, while being reasonable, is not robust to the high variety of images we've been working with, indeed, when the zoom is high, the white lines are usually excluded from the final mask, as the opening operator is not large enough to blend the lines with the field. Moreover, all small white elements are blended into the field, such as the player socks, even more so if they are white.
2. Then, the green chromaticity matrix is computed, by calculating, for each pixel, the following quantity:

   $$g(r,c) = \frac{G(r,c)}{R(r,c)+G(r,c)+B(r,c)}$$

   where by $G(r,c)$, $R(r,c)$, $B(r,c)$ we indicate the green, red and blue components of the pixel.

   One of the advantages of chromaticity is its luminance-invariance, which is crucial in a context where shadows may significantly alter the colors, thus making a threshold-based method less effective and robust.
3. We estimate the probability density function of the green chromaticity by using the Gaussian Mixture Model with $N_G = 6$ Gaussians. This is done by using the Expectation-Maximisation algorithm with random initialisation. After estimating the pdf, due to the expected higher component of green in the field, we impose that pixels constituting the field must satisfy $g > 1/3$.
4. Thus the first mask is obtained by selecting the pixels with values of g greater than T_G, where

   $$T_g = max(1/3, m_0)$$

   with m_0 being the argmin of the first local minimum after $g > 1/3$.

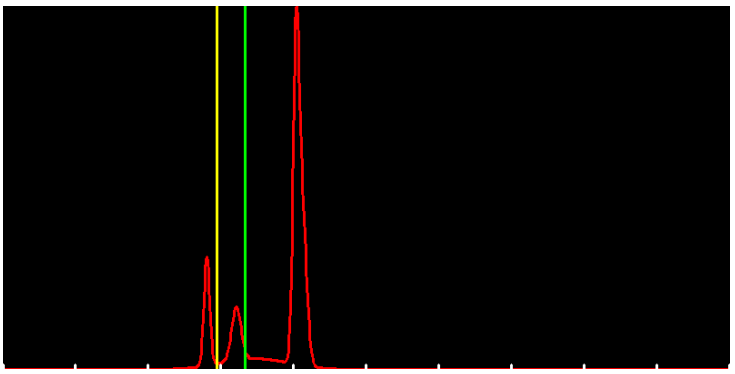


Figure: example of a PDF obtained from one of the images.
The yellow line is at $g = 1/3$, the green line at $g = T_G > 1/3$

5. Each of the modes of the pdf above T_g represents data with a similar level of green, however, there can be some colors such as cyan or yellow that have a strong component of green while still not being part of the field, such as billboards. In order to solve this problem we do as follows:
6. Using the Gaussians envelope, we find all the local minima after T_g and classify all the pixels in ranges that go from one minimum to the other, also including the range $[T_g, first\ minimum]$ and [last minimum,1]. Of course, the range that goes from 0 to T_g is not of interest.
7. For each of such ranges, we compute the mean color and then compare every element of the cluster u with the mean v, thus deriving the chromaticity deviation coefficient. This, from geometric considerations, is computed as follows:

$$cd = \frac{||u_{\perp}||}{||u_{//}||}$$

where $u_{//} = \frac{\langle u,v \rangle}{\langle v,v \rangle} v$ is the component of u along v, and $u_{\perp} = u - u_{//}$.
8. We finally create the second mask, by refining the first, adding a threshold constraint on the chromaticity distortion value, namely $T_C = 0.15$.
9. To improve results, we apply the opening operator again.

Below, some results are shown:

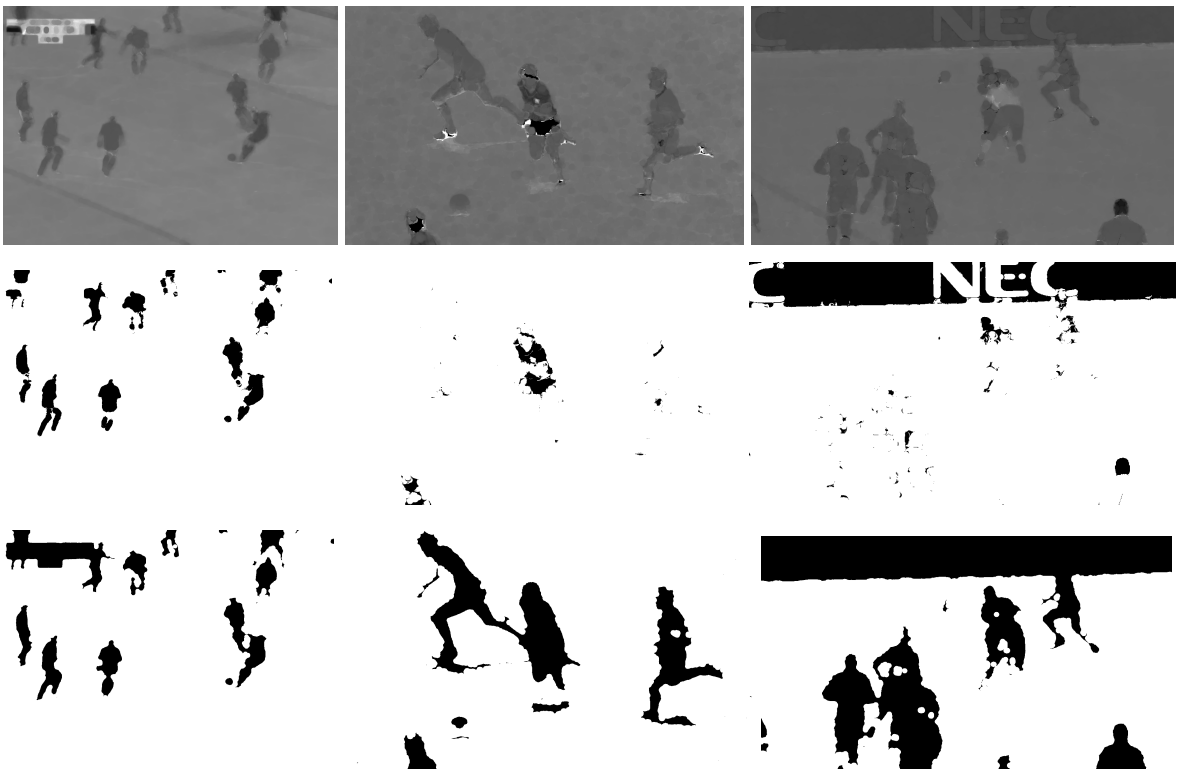
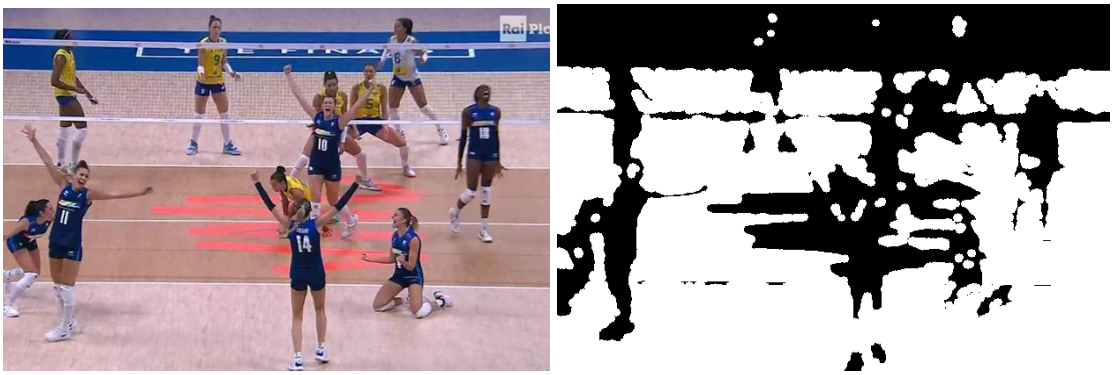


Figure: in the first row, three input images with different framings, the last one is instead a rugby image
In the second row, their respective green chromaticity images. In the third row, the first mask, obtained as described in step 4. Finally, in the fourth row, the resulting masks are shown.

Given the quite satisfactory results, we tried to generalise this approach to different field colors. The first attempt, for example, was to adapt the code to work with blue chromaticity for gyms and tennis courts.
Unfortunately, results were not consistent and had to be manually tuned to work properly.

Finally, for the gym courts, using red chromaticity, results were very poor, with some exceptions:



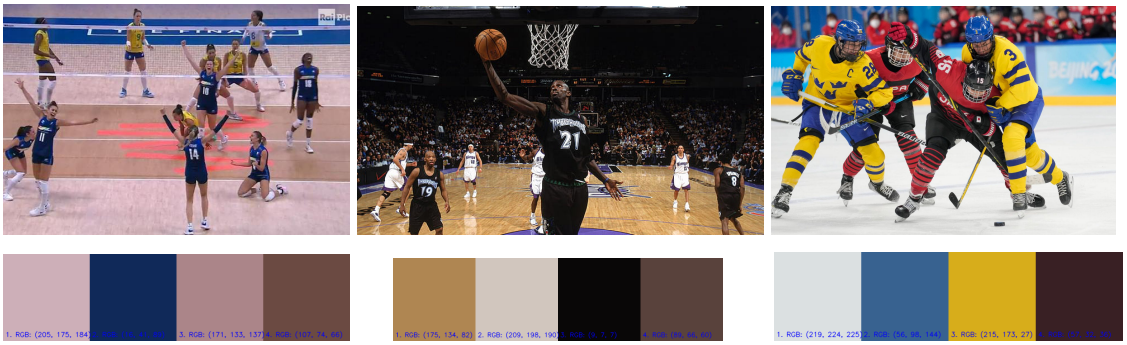
The implementation of the algorithm was challenging and it took several days to make it work properly. Thus, this algorithm was only kept for green fields, while a simpler approach was taken for the generic case, falling back to threshold-based color segmentation.

Threshold-Based Segmentation

Since the generalization of the previous method wasn't as easy or successful as expected, we decided to explore a different approach, based on the dominant color of the image.

The assumption is that the full image's most prominent color is the color of the field. With this assumption we perform K-Means, using K=4, and from that we heuristically extract the field's color.

This change was made after completing the pipeline, because we weren't getting the expected results in terms of quality of the segmented field.
In order to improve field segmentation. Indeed, the pipeline was supposed to pick the dominant color passed from the player segmentation subprocess. While this idea could have worked in theory, by supposing that in a bounding box, what is not the player is on average the field, this did not work in practice. By making field segmentation independent from the previous steps, we greatly improved the overall performance of our program, even mostly so when the field is not green.



After finding the means with K-Means, we measure the size of each cluster and determine the most dominant color, which is assumed to be the color of the field. Of course, it may happen for images strongly zoomed, that the main object portrayed almost completely overshadows the field, thus making the task very difficult. Another challenge is deriving some reliable heuristics about sports field, but given the variety of sports and frames, this is

no easy feat. Another challenge is managing multi-colored fields, such as some basketball courts and gyms.
After deriving the main color, the method proposed by the paper or the threshold-based segmentation are performed. In the latter case, since shadows and lighting can ruin the segmentation, we make considerations on the areas of these white or black blobs and also apply a closing operator.

Other Methods

Other approaches tried for the task were:

- Hough Lines Transform: the idea was that of recognising the external boundaries of the field, however, given the high variety of frames, with some showing no lines at all, this approach had to be discarded.

Some alternative and interesting approaches:

- It would have been interesting to apply camera calibration and geometric priors in order to detect fields [10], but the great variance in the test images and the little prior knowledge about the camera positions made this approach unfeasible.

Metrics Computation (Marco Sedusi, Marco Calì, Alberto Zerbinati)

mIoU

The Mean Intersection over Union (mIoU) is a widely used metric for evaluating the performance of image segmentation tasks, particularly in scenarios with multiple classes or categories. It provides a measure of how well a segmentation model can distinguish different objects or regions within an image.
In our context, there are four classes (background, team 1, team 2, playing field) involved.
IoU quantifies the overlap between the predicted segmentation mask and the ground truth mask for a specific class. It is calculated as the ratio of the intersection of pixels classified as that class by both the prediction and ground truth to the union of pixels classified as that class by either the prediction or ground truth.
We talk about Mean IoU when we want to evaluate the overall performance of a segmentation model across multiple classes, indeed you compute the IoU for each class and then take the average of these IoU values.
The mIoU metric provides a single scalar value that represents the overall quality of the segmentation across all classes.

mAP

Mean Average Precision (mAP) serves as a pivotal metric for assessing object localization performance, particularly in our context, which involves localizing players. Calculating mAP involves a multi-step process.

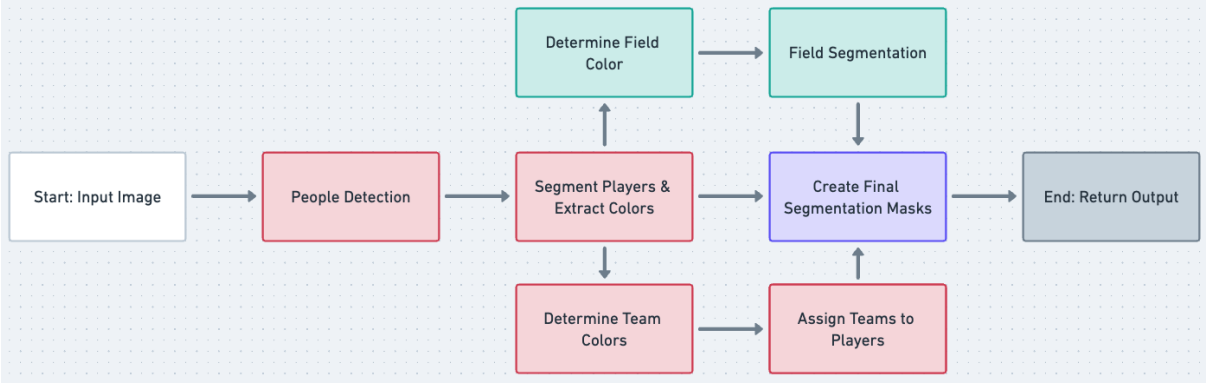
First, for each class (in our case, different player teams), we compute the Average Precision (AP). This is not a mere average of Precision values, but rather, it represents the area under the Precision-Recall curve.
To construct this curve, we start by ordering our detections in decreasing order based on their confidence scores. Subsequently, we calculate precision and recall row by row. This entails keeping track of True Positives and False Positives, determined through the Intersection over Union (IoU) threshold, set at 0.5.
Once the Precision-Recall curve is plotted, we apply the PASCAL VOC 11-point interpolation method to derive a comprehensive measure of the model's performance.
Averaging the AP values across all classes ensures that the model's effectiveness is evenly assessed, preventing it from excelling in one class while faltering in others. This approach promotes a holistic evaluation of model quality and its ability to localize objects accurately.

Pipeline Definition (Alberto Zerbinati)

The Initial Pipeline Proposal

We initially considered a linear pipeline with the following steps:

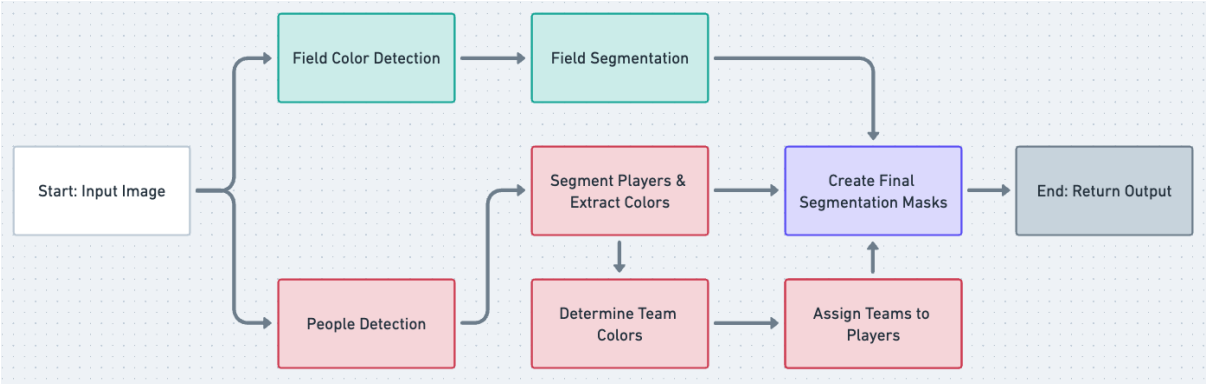
- Detect players;
- Segment them;
- Extract both team and field color information from each bounding box using a voting scheme;
- Use the dominant field color from the voting to segment the field.



The hypothesis was that the field color would be better captured from the non-player areas of the bounding boxes, given obvious spatial considerations (players are on the field), rather than from the full image.
However, this design had its shortcomings. Its linear nature made it unstable and error-prone. Any minor mistake at any stage would propagate and get magnified throughout the pipeline. Specifically, field color detection was not performing well.
As a result, we switched to a different design where field segmentation and player segmentation are independent tasks, mitigating the issues found in the linear approach.

The Final Pipeline

Here is the diagram for the final Pipeline:



Starting from the input image, we perform two **independent tasks** in parallel: field segmentation and player segmentation.

- **Field Segmentation** starts by detecting the color of the playing field, and then calling the appropriate method to segment the image, based on the detected color, as explained in the Field Segmentation section;
- **Player Segmentation** is a multi-step process:
    - First we perform Player Detection, using the CNN + Sliding Window + K-Means approach described;
    - Then we segment each bounding box to get the players' area;
    - Each bounding box also votes for a team color;
    - Votes are collected, and in the end, we select the two most frequent colors and assign players to teams;

The Pipeline concludes by putting together all the collected information. The output consists of a "binary" segmentation mask, with values representing the background, team1, team2, and the field, a corresponding colored mask, and a set of bounding boxes. We save the bounding boxes both in a text file and overlay them on a newly saved image for inspection.
Something worth mentioning is that the Pipeline considers the field segmentation more reliable than the player segmentation. For this reason, whenever a pixel is classified both as field and as player, we assign it to the field class,

Pipeline Evaluation

The evaluation process is included in a separate function of the Pipeline. It simply relies on the implementation of the Metrics (mIoU and mAP), which are computed between the Pipeline's output and the ground truth.

Results and Discussions

We report in the table the results in terms of mIoU, mAP, and output segmentation mask.

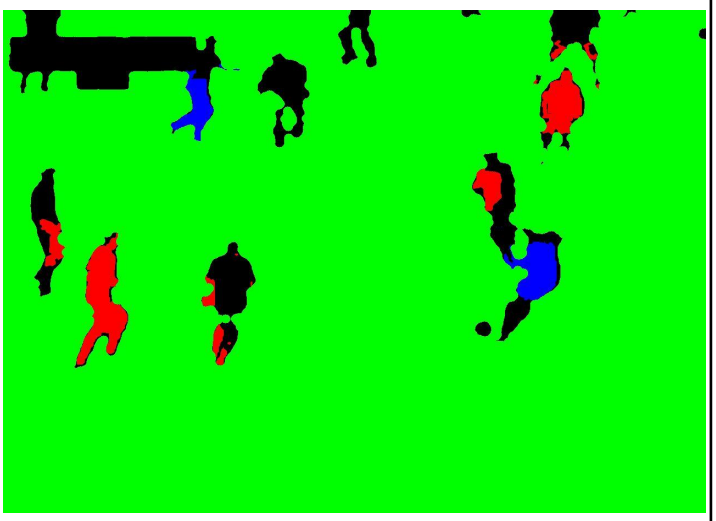
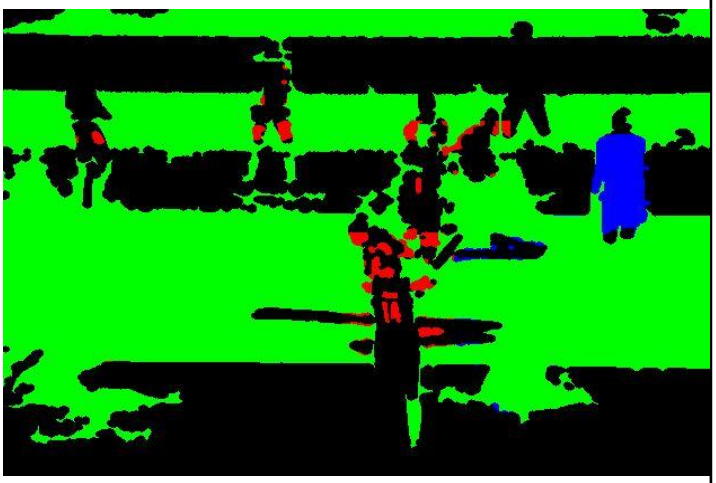
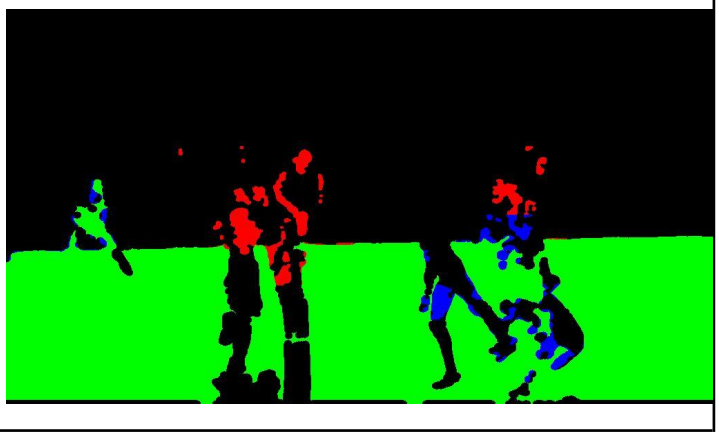
image	mIoU	mAP	segmentation mask
image 1	0.481458	0.181818	
image 2	0.166378	0.181818	
image 3	0.456778	0.0909091	

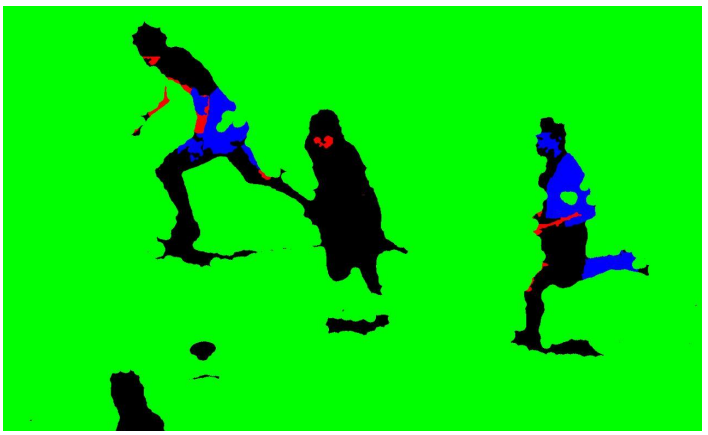
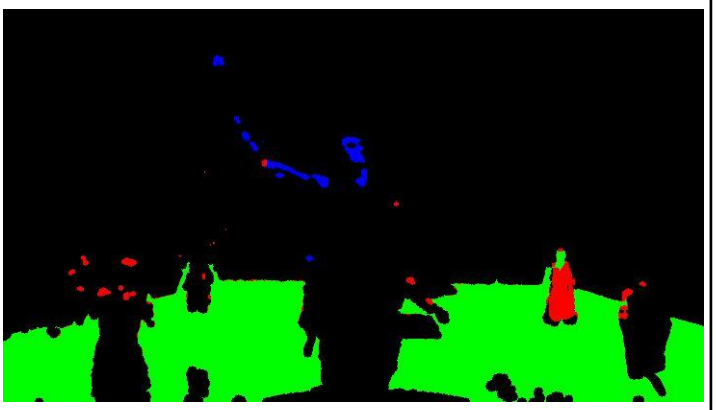
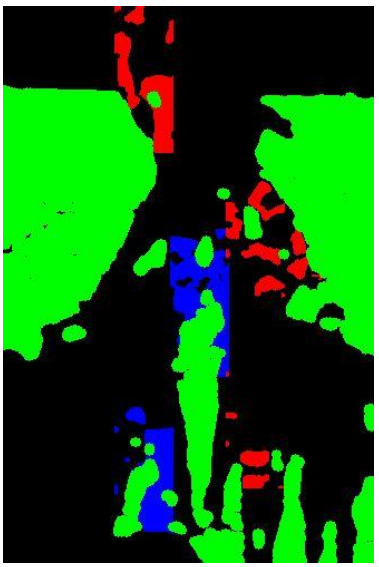
image 4	0.306785	0	
image 5	0.432831	0	
image 6	0.315273	0	

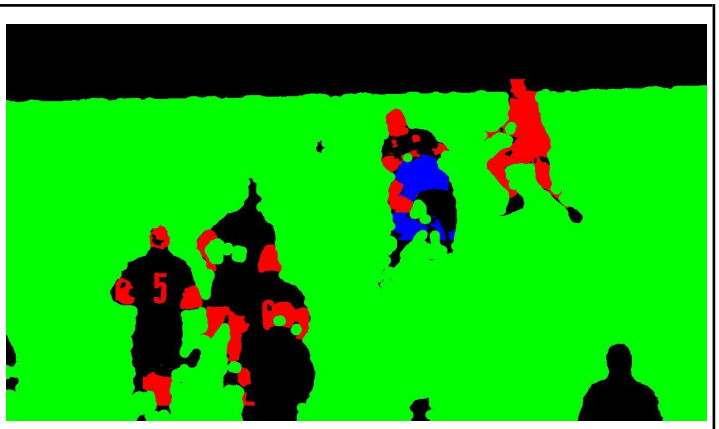
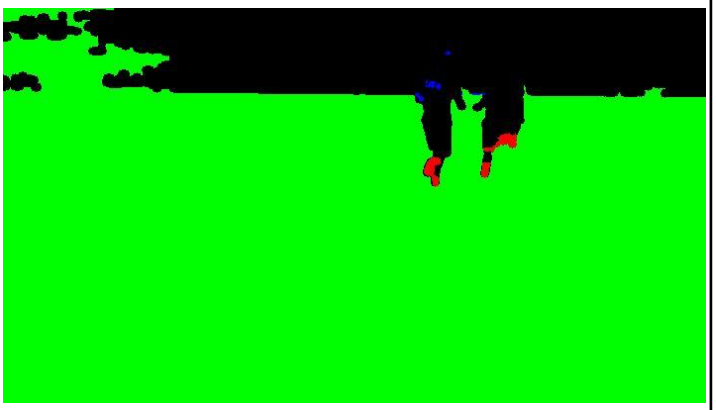
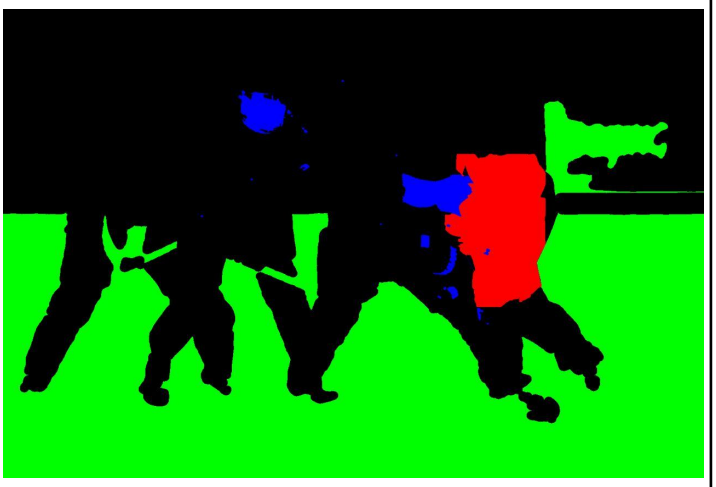
image 7	0.519466	0.0681818	
image 8	0.276812	0	
image 9	0.303138	0	

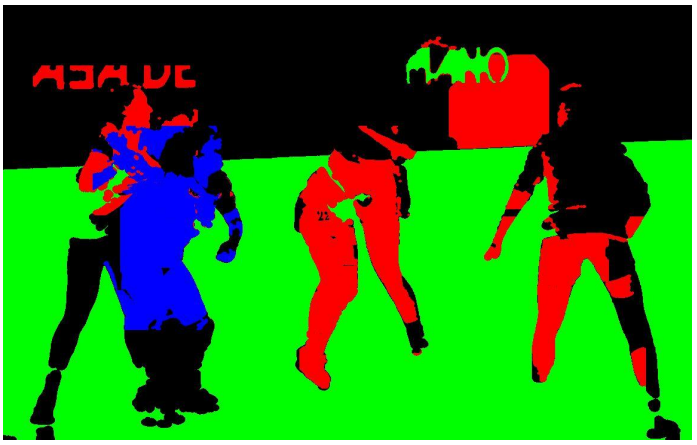
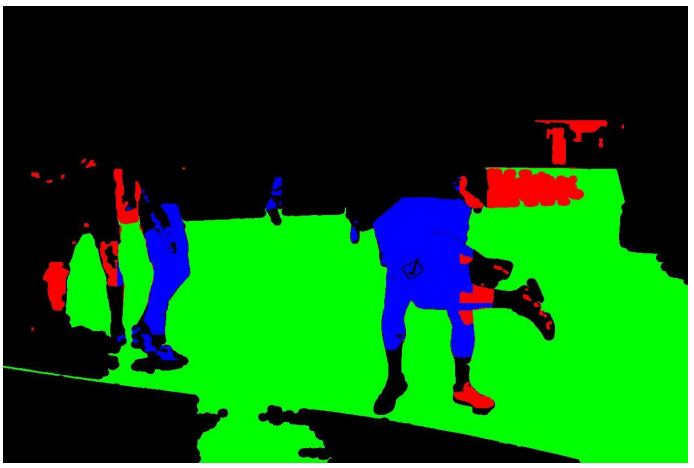
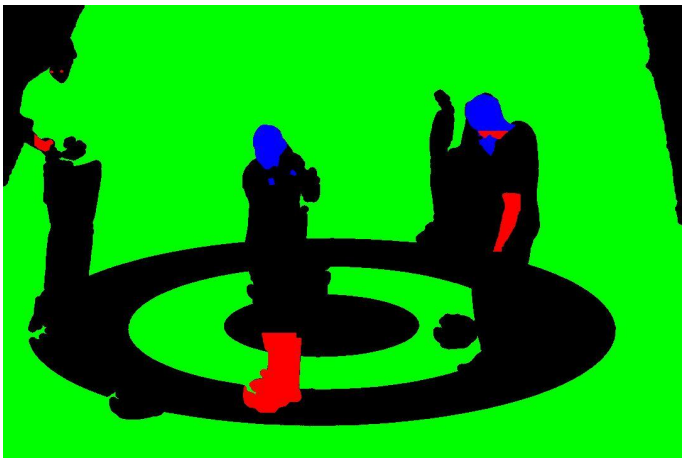
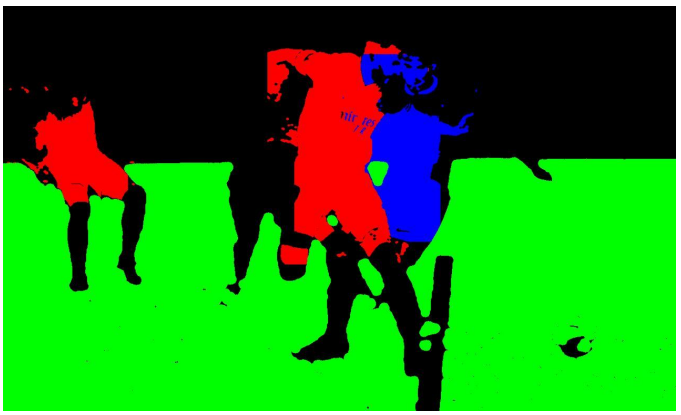
image 10	0.378268	0	
image 11	0.419325	0.272727	
image 12	0.34546	0	

image 13	0.539926	0.0909091	
image 14	0.266715	0	
image 15	0.404646	0.272727	

Discussion

The **average mIoU and mAP** on the test set are respectively ~0.38 and ~0.08.

We can observe that the results of people segmentation are not always the desired ones, also due to imperfect bounding boxes returned by people detection.
The mIoU metric, which assesses segmentation performance, generally performs well in most test images. However, in the case of image 2, the segmentation performance is

negatively affected due to the image's complexity. This complexity arises from a field with multiple dominant colors and a high number of people to detect in the scene.
The same argument turns out to be partly valid also for image 14.

The most concerning metric is mAP, which equals 0 in various cases. We attribute that to the incorrectly detected scale and general imprecision of the bounding boxes. Having set an IoU threshold of 0.5, the mAP will result in 0 every time the boxes don't allow to overcome this threshold. This is certainly the most critical aspect of the integration of our work: it's missing a refinement of the bounding boxes detection, plus a second pass to improve player segmentation.

The team specification also suffers as a consequence of the two issues highlighted above.

An interesting consideration is on image 8, where the output doesn't capture the players on the bottom left. This effect is due to the greater relevance we give to field detection rather than player segmentation. We indeed inspected the bounding boxes and segmentation, and there were some detections in that area. The pipeline would however overwrite them with the field. This is another area where more careful considerations might help.

Note: these results were obtained with **multiple runs of the pipeline for each image** (3-5), taking the best outcome. There are indeed some stochastic processes involved (e.g. K-means). Also, the player detection model itself depends on the training process which is not fully deterministic and can make the results vary. Most surprisingly, we tried running the same model and pipeline on different hardware configurations and for reasons not fully clear to us, we were getting different results, even using a shared Docker container. We invite the tester to try our model but also train a couple of other ones.

Miscellanea

Running the project

Follow the instructions on the README to run the project. We optionally provide a Docker image to make the installation easier.

Tests and Project Management (Alberto Zerbinati)

The test definition and implementation, as well as all the software engineering tasks (project architecture definition, git repository management, tasks specification, time and resources management) were carried out by Alberto Zerbinati.
When *making* the project, not only the *sports_human_detection* executable will be produced, but also a couple of test executables. They try to cover both unit and integration tests. For instance, there is a test (based on visual inspection of the output) for the person detector, one for the integration of the person segmentation into the person detector, etc.

These tests were useful to evaluate the quality of the project while developing it. They served also as the base code for the main Pipeline.
In our final deliverable, we've included only two out of the four total tests initially planned. During the development phase, indeed, frequent changes to class and function interfaces made keeping the tests updated increasingly cumbersome and the build times longer, outweighing the benefits of full test coverage.

Besides this, I was also responsible for conducting the tests and evaluating the performance improvements of the changing parts of the Pipeline.

Limitations and Future Improvements

We were quite surprised and sad to see that the integration of our work resulted in poor results in some cases. We acknowledge the limitations of the provided solution and would like to discuss possible future improvements that we considered but couldn't implement due to time constraints:

- **Better player detection**: our current approach using CNN, sliding window, and clustering tends to produce a number of false positives, and struggles to find the correct scale of the players. We considered refining the bounding boxes based on certain metrics. Another option could be to merge closely located boxes, especially after segmentation, where we have more detailed information about each subject;
- **Better player segmentation** through face and number detection: we considered detecting faces and jersey numbers on players. However, many players are seen from behind, making face detection challenging. We tested a simple Viola-Jones approach but were unsure how to integrate it into the pipeline due to inconsistent accuracy. For number detection, we believe a separate CNN training would be required, and we didn't have time to collect a proper dataset;
- **Better dialog between different Pipeline's steps**: for instance, there's a clear correlation between team color detection and field color detection. We could use field color to eliminate false-positive team detections or apply similar techniques.

We've found that the **integration of individual components** is a critical aspect of a software project. In our case, we're generally satisfied with the performance of each part (Player detection performs okay, particularly on televised images; Player segmentation is highly effective when provided with accurate bounding boxes; Field segmentation is reliable when the field color is correctly identified) but the integration is doesn't satisfy us fully.

We believe that employing a more refined approach for player detection could significantly improve the overall performance of the pipeline.

Overall, we faced significant challenges during project development for various reasons, including:

- The task is complex and multifaceted, requiring many parts to work in unison. We were confident about integrating everything smoothly for a working product, but even minor errors had a magnified impact, leading to lower-quality results than expected;
- Communication and organization within the team: instances occurred where someone spent a lot of time on tasks that didn't align with our planned direction. For example, the field segmentation algorithm wasn't generalized to any color and took over a week for one team member to develop;

Number of hours worked

	Alberto Zerbinati	Marco Sedusi	Marco Calì
hours worked	~100	~75	~75

Bibliography

[1] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), San Diego, CA, USA, 2005, pp. 886-893 vol. 1, doi: 10.1109/CVPR.2005.177.

[2] Object Detection Combining Recognition and Segmentation. Liming Wang, Jianbo Shi, Gang Song, I-fan Shen. To Appear in ACCV 2007

[3] Tsung-Yi Lin, Maire, M., Belongie, S. J., Bourdev, L. D., Girshick, R. B., Hays, J., … Zitnick, C. L. (2014). Microsoft COCO: Common Objects in Context. CoRR, abs/1405.0312. Retrieved from http://arxiv.org/abs/1405.0312

[4] He, Kaiming, et al. "Mask r-cnn." Proceedings of the IEEE international conference on computer vision. 2017.

[5] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. In 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 779–788). https://doi.org/10.1109/CVPR.2016.91

[6] Interactive graph cuts for optimal boundary & region segmentation of objects in ND images. Yuri Y Boykov, M-P Jolly

[7] Duda, R. O., & Hart, P. E. (1972). Use of the Hough transformation to detect lines and curves in pictures. Communications of the ACM, 15(1), 11–15.

[8] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001, Kauai, HI, USA, 2001, pp. I-I, doi: 10.1109/CVPR.2001.990517.

[9] A fully automatic method for segmentation of soccer playing fields | Scientific Reports. Link: https://www.nature.com/articles/s41598-023-28658-1

[10] Sports Field Localization via Deep Structured Models by N. Homayounfar, S. Fidler and R. Urtasum, University of Toronto