CV Project

Francesco Pio Monaco[1] Michele Russo[2]
1 2087638 2 2087637

Introduction

Human detection has been a problem known in literature for ages, many applications have been developed for pedestrian detection, surveillance systems and crowd analysis using various systems that range from traditional computer vision techniques to machine learning and deep learning.
Our goal was to develop a system capable of detecting humans in various images of sports; where different poses, heavy clothing and different background patterns may pose significant challenges. The system also needs to be capable of classifying such detected people into the two teams playing, and segmenting the image into background/playing field and humans (divided by team).
This project has been developed by minimizing the use of ML to lightweight and fast systems developed from scratch, using various methods studied in class (e.g. Canny, LBP, watershed) similarly as some literature studies to obtain robust results.

Detection & Classification

First Attempts

Initially some non ML implementations were tried like multi-scale template matching of sports clothing; then noticing that most of the players have a number on their clothing a number matching was implemented. These methods provided mediocre results, retrieving many false positives and providing boxes that needed heavy reshaping to obtain the full players. Examples are shown in (Figure 1).

Figure 1. Results of matching. The boxes of fixed dimensions needed reshaping, centering and a heavy clean-up for false positives.

HOG+SVM [2]

Despite being relatively old using an SVM with a HOG descriptor is still one of the most common methods for object recognition. This approach fitted very well our needs of a reliable tool for recognition while needing a relatively small dataset and faster training times w.r.t. deep learning.
A training/validation dataset was created from scratch using stills of players from the

various sports in our test dataset; the stills were carefully chose to display an array of positions, gender, skin types for the positive samples; empty field images, balls and stills with partial humans crowded the negative samples. The dataset is then augmented using an image augmentation object from the library `imgaug`, the pipeline creates images applying with some probability these transformations: an horizontal flipping, a $[-20°, 20°]$ rotation and a gaussian blur.
The HOG descriptor created using the OpenCV library is initialized with the following parameters:

```
winSize = (64, 128) window size
blockSize = (16, 16) block size
blockStride = (4, 4) block stride
cellSize = (8, 8) cell size
nbins = 12 num of bins
```

The descriptor is then scaled using a MinMaxScaler fitted on training data.
The use of this descriptor w.r.t. other descriptors introduced in class (e.g. SIFT descriptor, LBP) is justified in the literature as being more stable and better performing; in our experiments using more bins (12 w.r.t. 9) and a smaller stride (4 w.r.t. 8) provided better results.
The SVM was trained performing a GridSearch over a *linear* kernel and different *gamma*s and *c*s, in order to obtain an automatic regularization over the features.

Selective Search + Canny Length Pruning

To detect players on test data the simplest approach is using sliding windows of different dimensions (in our test dataset some people need square boxes or rectangles where the width is greater than the heigth) at different scales over the image and feeding them to the SVM for classification, this approach, other than being computationally heavy, produces a massive amount of windows to classify, resulting in a high probability of finding false positives.
The implemented approach consist in using `SelectiveSearch` (provided by OpenCV in the *ximgproc* library), the method performs various segmentations at different scales using grouping algorithms based on color, location and texture providing a number of diverse boxes that enclose the resulting segments, these segments may contain objects (the method doesn't know what object) [7].
To further reduce the number of boxes to analyze, we implemented`canny_len`. This method returns the height and width of the largest contour in the image analysis after applying *canny*. Subsequently, we use the function `cv.boundingRect` on the *max* of the return of `cv2.findContours`, to prune all the boxes outside the chosen interval $0.5 * canny_h|canny_w < box_h|box_w < 2 * canny_h|canny_w$ (e.g. the image from the 2006 football cup has canny values of $\simeq 100$, while the hockey ones of $\simeq 600$, there's no need for big boxes in the first one as there's no need for small ones in the second).
We then classify (as human/non-human), using the *SVM*, the remaining bounding boxes, and save them in a temporary folder.

Post-processing

Now we move to our main in C++ to remove False Positives and perform the other tasks.

Diffusion Mask

In this subsection we use the function `computeDiffusion` to create a diffusion mask of our images. This method, inspired by heat diffusion [4], computes the Canny image using

dynamic thresholds for optimal detection, the thresholds are computed in the following way:

1. Median of the gradient magnitude computation using a Sobel operator;
2. $threshold1 = \frac{9}{2} * median$, $threshold2 = 9 * median$ where the constant 9 is given in the paper [4].

Figure 2. Masked image. After applying the diffusion mask the only things left are the players and some noise.

Once obtained the Canny image it is processed for a number of iterations by 'expanding' the contour, then a max kernel is applied to fill internal holes. The mask is then applied on the original image, (Figures 1, 2) display how most of the background is removed by the masking.

Removal of uniform boxes

Each box found by our SVM is localized on the masked image, then the number of black pixels is computed on it, if below a threshold the box is removed (removes completely black boxes and boxes where the main component is black).
This simple approach is capable of removing most of the False Positives by just computing one time the mask for the whole image and without the need of analyzing the content of the boxes.

Boxes Merging

Since in some images the players are found by 2+ overlapping boxes, the algorithm `mergeRectangles` was implemented to merge the boxes whose intersection area is above a threshold into a bigger box containing both. The thresholds for merging are tuned to the dimensions of the images in the test set. In a general setting bigger images could be fed to the pipeline (e.g. bigger than $1900x1900$) and the merging would make some mistakes.

Team Classification

The classification of the final boxes in teams is obtained using the function `classify` which applies the following steps:

1. BGR2HSV conversion of the detected boxes, the H channel gives information decorrelated from the luminosity in the image;
2. Histogram computation, concatenation and normalization via `calcHist`;
3. Distance matrix computation between each pair of boxes via `compareHist`;

4. `kmeans` (from openCV) on the distance matrix.

The provided labels are the ones assigned to each box.
The method provides the last step of False Positives cleaning via the function `class_clean`, when an image has a False Positive box the classification puts all the humans in the same class and keeps the False Positive into another (due to strong differences in color), in case that a label has only 1 box associated we check the box height, if not similar to the others we remove the box and redo the classification on the remaining boxes.

Segmentation

Since we have very diverse images to segment and using only non-ML methods, the implemented approach consists in different phases that work at different scales (human/field).

Player Segmentation

In this section we will discuss about the approach we used to segment the players inside each image.

This method can be divided into six steps:

- image denoising;
- color clustering;
- apply canny filter;
- close the borders;
- merge color and spatial information;
- remove non-connected components.

Image denoising

Image denoising is a computer vision technique focused on noise removal.
In particular the method used for this task is called *Non-Local Means Denosing* [1].
The method is based on a simple principle: replacing the color of a pixel with an average of the colors of similar pixels. But the most similar pixels to a given pixel have no reason to be close at all. It is therefore licit to scan a vast portion of the image in search of all the pixels that really resemble the pixel one wants to denoise.
This technique has two main implementation, the one we used, taken from the opencv library and called with the method *fastNlMeansDenoising*, is the *Pixelwise*.
Given a 3 dimensional image, the denoised colored image is:

$$\hat{u} = \frac{1}{C(q)} \sum_{q \in B(p,r)} u_i(q) w(p,q)$$

where B indicates the neighbourhood centered at pixel p with size $(2r+1)(2r+1)$, the weight w(p,q) depends on the euclidean distance, each pixel value is restored as an average of the most resembling pixels, where this resemblance is computed in the color image. So for each pixel, each channel value is the result of the average of the same pixels. We use an exponential kernel in order to compute the weights w(p, q).

We used this approach in order to remove noise and to improve the robustness of the segmentation also in low resolution image (Figure 3).
In fact, we noticed a lack of precision on low resolution image, such as *im_1* and *im_4*.

(a) Before denoising. **(b)** After denoising.

Figure 3. Comparison between original and denoised images. We can notice how the grass is smooth in (b)

Canny and closing borders

We've developed two versions of this algorithm: one finely-tuned specifically for the given dataset, and another that offers greater robustness and can be applied to different datasets without sacrificing precision, additional testing has been conducted on this version, and the results are presented in **Supplementary Test Images**.
On the **first version** we extract the boxes, provided by the SVM and post-processed as discussed in **Detection & Classification**, then we apply, over the box, a Gaussian blurring before applying the Canny edge detector. Here there is an image after pre-processing phase:

Figure 4. Preprocessed Image.

Finally we start the player edge detection, using Canny. Since the images vary in both size and content, we used an adaptive algorithm; inspired from [3], to set canny thresholds. For each box we calculate the gradient by using *Sobel filter* in both directions (x and y), the upper and the lower thresholds of canny are determined as $TH = c * median(G)$ and $TL = TH/2$. Where, median(G) is the median value of gradient magnitude of the

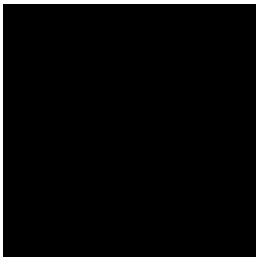
bounding box G. c is a scaling factor for threshold selection which is a positive constant. In our tests we used a constant c=5. Here there is one example:

Figure 5. Canny Image.

As previously mentioned, all the information related to the background are completely removed, but we still have some noise given by the nets or field irregularity.
Finally we close the border of the player by using morphological operators.

We tested many operators provided from the *OpenCV* library, at the end we decided to apply gradient which is the difference between the dilation and the erosion of a given image.
We also tested the *Opening operator*, since we wanted to enhance the connectivity and remove all the weakly connected edges, but after many tests we switched into gradient operator. Here there is a comparison of both operators:

(a) Gradient operator results.



(b) Opening operator results.

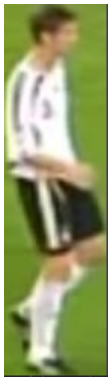
Figure 6. Comparison between gradient and opening.

As you can see the effect of erosion removes completely all the lines expanded by dilate.
Finally we apply another function called *create_lines*, this step is essential because numerous boxes do not encompass the entire player, making it impossible to generate an approximate segmentation able to detect the player inside the box.

At the end we filled the edges to get the approximate player segmentation using the *findContours* algorithm implemented in *opencv*. For each edge we use *drawContours* in order to fill closed lines. Here there is an example:

Figure 7. Filled image.

As we can see from the starting image, our canny segmentation is able to take the player inside the box, but it still has many false positives, due to some noise like the net and the presence of other players inside the box itself.
While the results in images that are not affected by noise are perfectly taken, as shown on the following example:



(a) Extracted box with blurring.

(b) Segmentation.

Figure 8. Player Segmentation Examples.

The **second version** is quite similar from the fist one, but it differs on the way we fill canny edges. We fill the image by applying the convex hull directly after applying morphological operators to the canny image. Following this, we draw the contours by using *drawContours*.
Here there is an example:

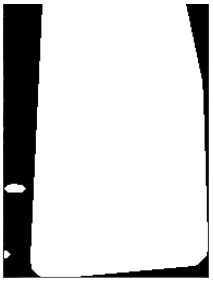


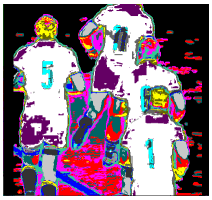
Figure 9. Canny Image.

Nevertheless, after completing the segmentation processing, the resulting image will closely resemble the one obtained through the first approach.

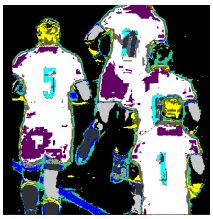
Clusterization

As seen on the previous section, canny images and all the post-processing made for each image in case of noise it occurs into an over-segmentation as seen in (Figure 7).
We solved these problems by using color clusterization on the whole image. Since the noise is caused by net or background, by using this technique we are able to assign a label to each of these noisy patterns and remove them quite completely from the image. Here there is an example to get an idea:

As you can see, similar patterns of noise will have the same label.
Color segmentation is performed by the `kmeans++` algorithm.
In order to exploit more information from the neighbourhood of each pixel, we decided to



(a) Segmentation before clusterization.



(b) Segmentation after clearing the clusterization.

Figure 10. Clusterization example.

add *Local Binary Pattern* features on clustering.
LBP technique is a feature descriptor, based on the inspection of pixel neighbourhood. In a nutshell our implementation considers a ray r, in which we can take the pixels, for each pixel inside the ray we take its neighbourhood, of size k. We move clockwise through the neighbourhood, we make a comparison between the i-th pixel's intensity value and that of the central pixel. If the neighboring pixel's value is greater than the central pixel's value, we calculate a value represented by 2^i, where 'i' denotes the position of the currently considered neighboring pixel in relation to the central pixel, and then we sum all values to obtain the LBP score of the central pixel. Here there is an explicative example of local binary pattern:

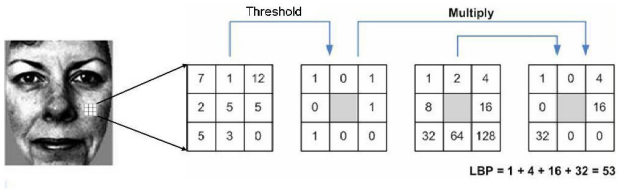


Fig. 1. Example of an LBP calculation

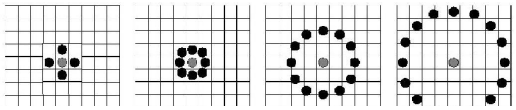


Figure 11. Simplificative example of LBP.

Here there is an example of the clusterization effect:

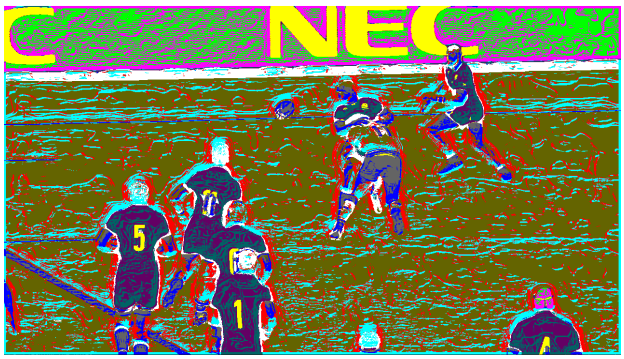
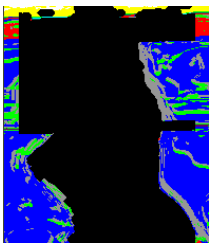


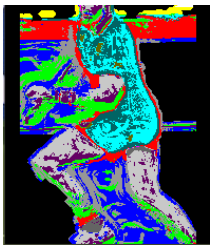
Figure 12. Clustering.

Merge features

In this section we will discuss about merging feature provided by segmentation and filling edges technique discussed in **Canny and closing borders**; that provides us a starting player segmentation, and color clustering information discussed in **Clusterization**.
The problem of the staring player segmentation (**Canny and closing borders**) is that it exploits only edge information so it is exposed to noise, that leads to an over-segmentation, as previously discussed.
In order to refine the border and eliminate all this noise, we perform a color clustering over the entire image.
Given a starting segmentation of box i and a clusterization of the image G, we take i_r that is the reverse of our segmented box, we localize it on G, we count all the pixels per label, provided by the clustering then we remove all the labels from the i that don't



(a) Reverse box with clustering labels.



(b) Segmented box with clustering labels.

Figure 13. Examples.

respect this constraint:

$$n_j > \frac{A}{L}$$

where:
n_j = Number of pixels of label j
A = Area outside the segmented area
L = Number of labels outside the segmented area.

The main idea is that mostly noisy labels inside the primitive segmentation are also outside the label and they have the same color and quite similar features, so by eliminating all the labels that we find outside, under the criteria above, we are able to remove quite all the noise and better define the player.
Here there is an example of the results:

Figure 14. Clustering removal of noise.

This method is implemented on the *super_impose*.

Remove Non-Connected Components

To clean up the final segmentation from any disturb the method `remove_components` finds all the connected components of the segmented sub-image and removes all the components whose

$n_i < mean\_n.$ where n_i is the number of pixels of the connected components i, and $mean_n$ is the average of the number of pixel of each region. Here there is an example of the results:

Figure 15. Segmentation after NCC removal.

Field Segmentation

The approach starts by removing the segmented players from the image using the function `player_elimination`, this ensures less noise in the subsequent `color_quantization`, which divides the image into 3 regions based only on color information; the clustering is then passed to the function `merge_clusters` which given as input a *distance threshold* merges together all the clusters that satisfy the condition, the distances are computed using the openCV function `norm`; after the merging the returned image is a basic segmentation between playing field and background with some noise in the two areas (Figure 16). This approach ensures that extra elements in the image do not interfere during the clusterization of field/background.
The field segmentation is then refined by the methods `line_refinement` and `court_segmentation_refinement`, whose goal is to use the distance transform watershed to create an uniform image with the watershed lines, using a high threshold for the distance transform we're able to obtain in all cases the strong lines dividing the image (as showed in Figure 17a); subsequently the Hough Lines function is called to find the longest horizontal line (usually the longest is the one diving the playing field from the background); if such line exists the initial segmentation is refined by keeping everything below the found line as is (in the playing field we may have objects that are segmented as background) and putting all that is above to background (Figure 17b).

Figure 16. Basic playing field detection.

(a) Watershed result. The line that divides the playing field from the background is clearly visible.

(b) Refined result. Everything above the detected line is put to background.

Figure 17. Examples from the refinement of the court detection.

Final Segmentation

The final segmentation is obtained through the function `unifySegmentation` by applying over the field segmentation, in the position given by the bounding boxes, the segmentation of the players using their relative label obtained through the classification.

Evaluation, Results & Visualization

Since no indication were given on how to assign labels to the teams, the evaluation works by comparing our results to the ones in $Mask$ and the ones with the two team labels reversed by taking the max.
Results on both boxes and segmentation may vary due to clusterization and different SVM states.

mAP: 0.11

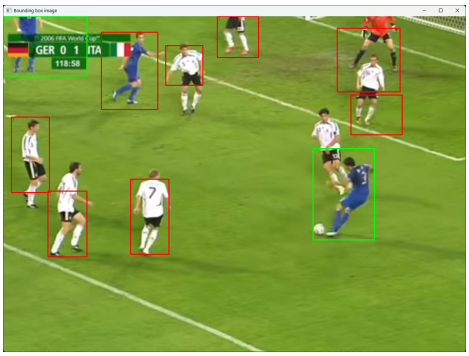
The threshold of 0.5 penalizes the majority of the boxes since they're either bigger or smaller; using a threshold of 0.3 (used in some literature to evaluate crowded scenes) the mAP rises to 0.5; despite this result the boxes are still good since the segmentation provides good results (the segmentation works on the boxes).

IoU: 0.47

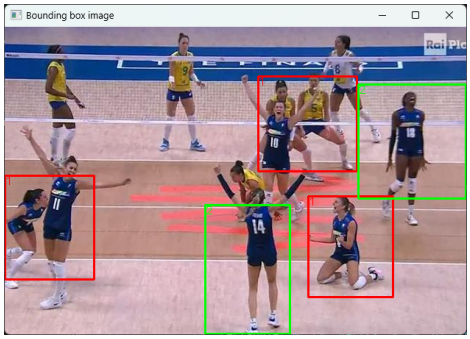
Since many errors are due to the classification (some boxes contain parts of other players and end up classified as the same team) an evaluation was tried with only 3 classes (background, playing field, human): in this case a IoU of 0.57 was obtained.

Bounding Boxes

Figure 18.

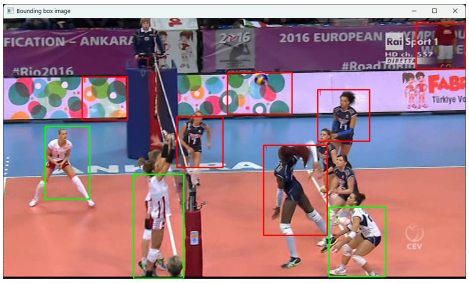


(a) AP: 0.2

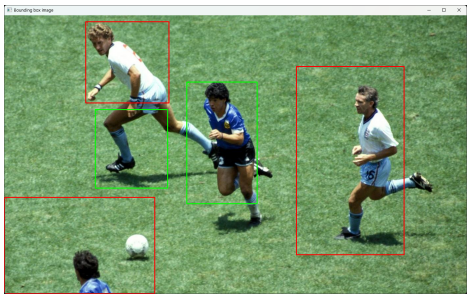


(b) AP: 0.1

Figure 19.



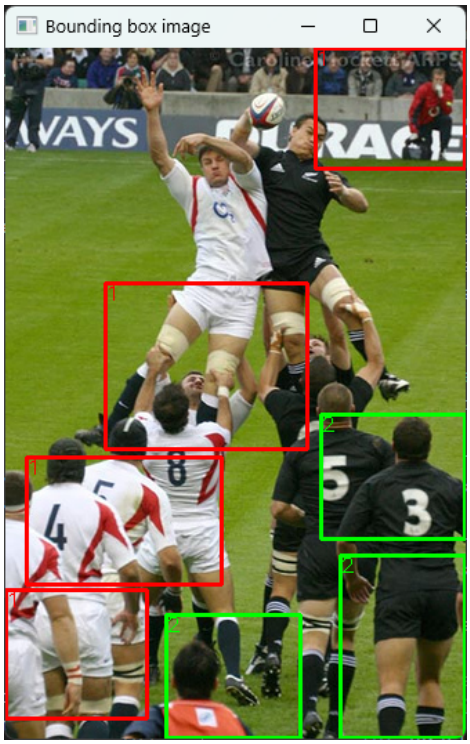
(a) AP: 0.2



(b) AP: 0.1

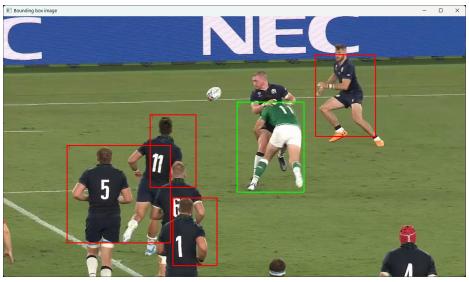
Figure 20.

(a) **AP:** 0

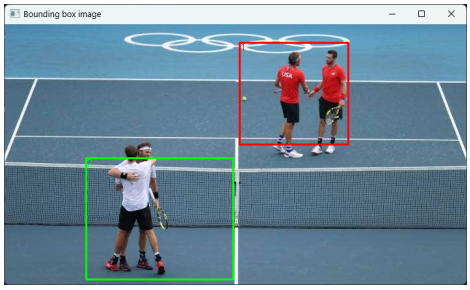


(b) **AP:** 0.1

Figure 21.



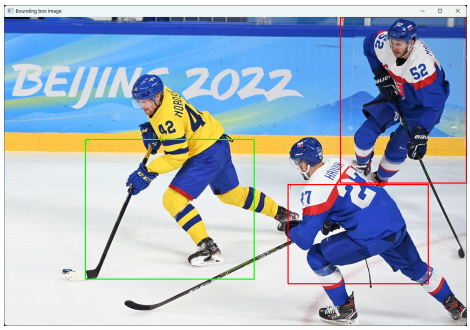
(a) **AP:** 0.6



(b) **AP:** 0

Figure 22.

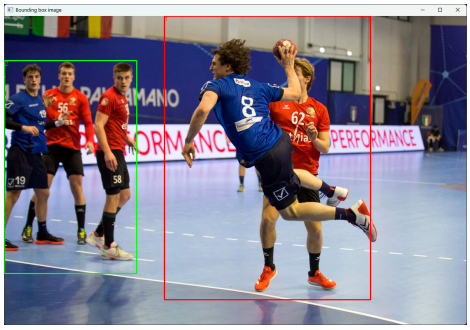
(a) **AP:** 0.5



(b) **AP:** 0.13

Figure 23.

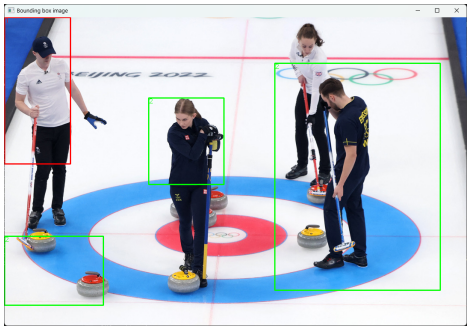
(a) **AP:** 0.6



(b) **AP:** 0.13

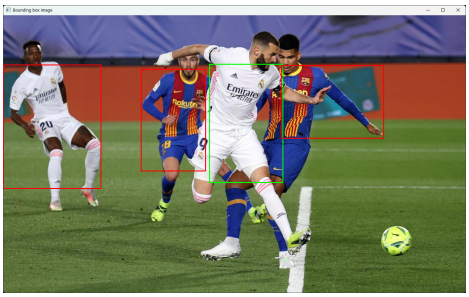
Figure 24.

(a) **AP:** 0.5



(b) **AP:** 0

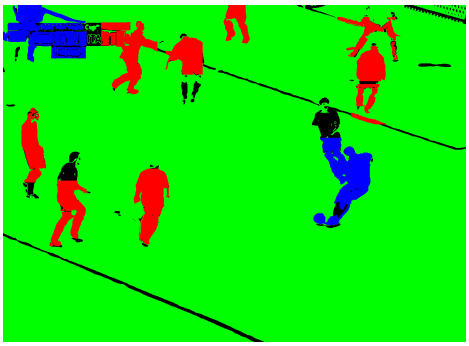
Figure 25.



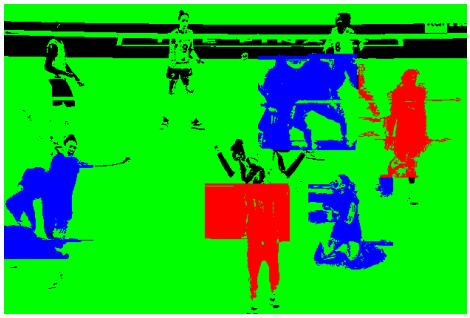
(a) **AP:** 0.13

Segmentation - Base

Figure 26.

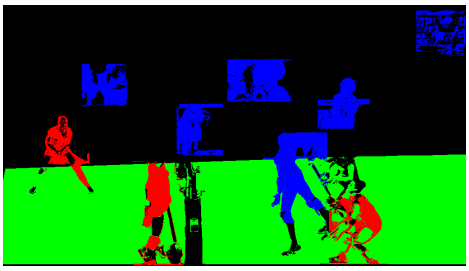


(a) **IoU:** 0.69

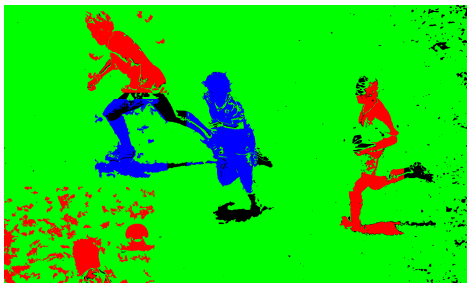


(b) **IoU:** 0.35

Figure 27.

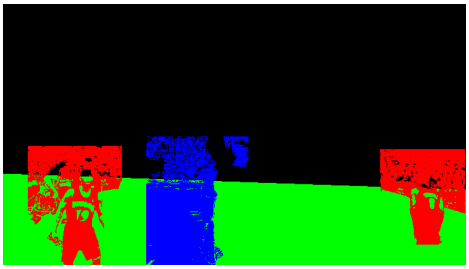


(a) **IoU:** 0.57



(b) **IoU:** 0.41

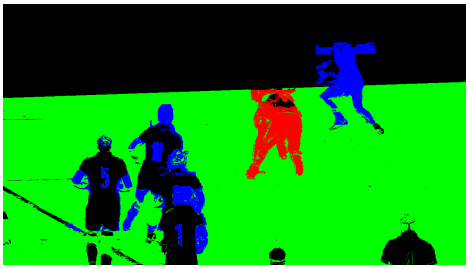
Figure 28.



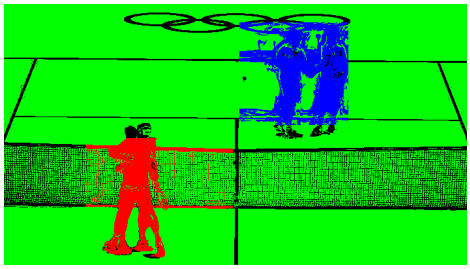
(a) IoU: 0.36

(b) IoU: 0.40

Figure 29.

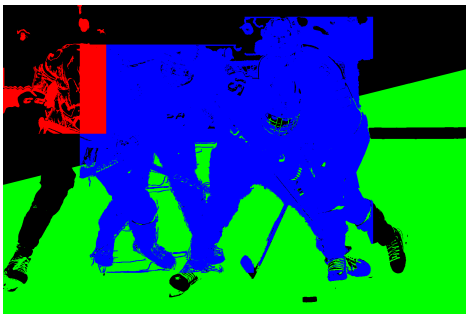


(a) IoU: 0.50



(b) IoU: 0.46

Figure 30.

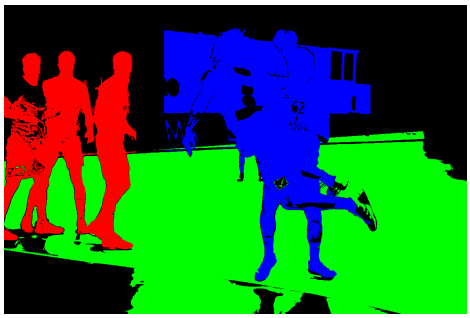


(a) IoU: 0.43

(b) IoU: 0.60

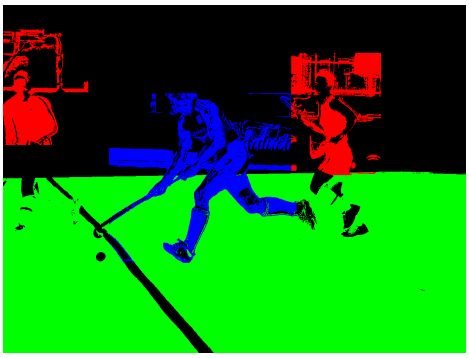
Figure 31.

(a) IoU: 0.43



(b) IoU: 0.83

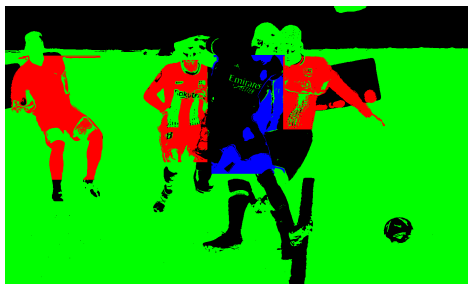
Figure 32.



(a) IoU: 0.41

(b) IoU: 0.35

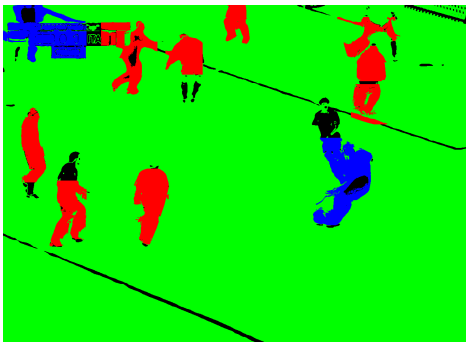
Figure 33.



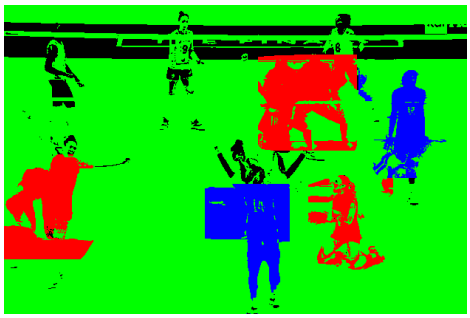
(a) IoU: 0.28

Segmentation - Robust method

Figure 34.

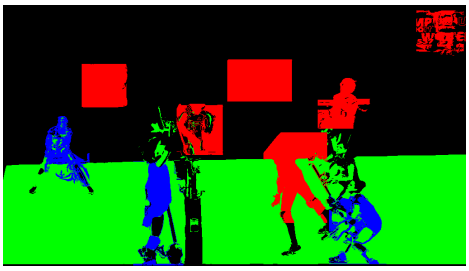


(a) IoU: 0.60

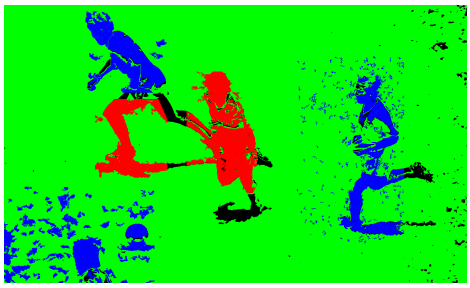


(b) IoU: 0.34

Figure 35.

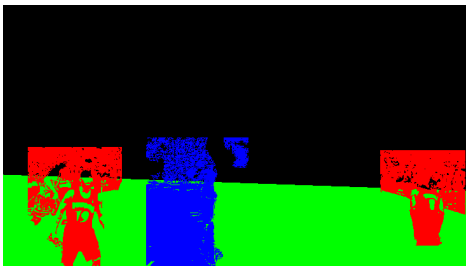


(a) IoU: 0.58



(b) IoU: 0.41

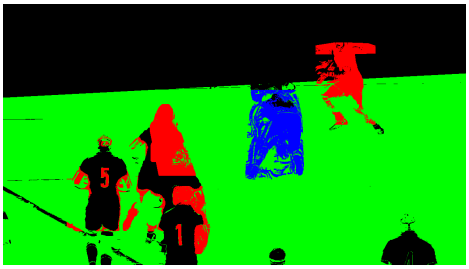
Figure 36.



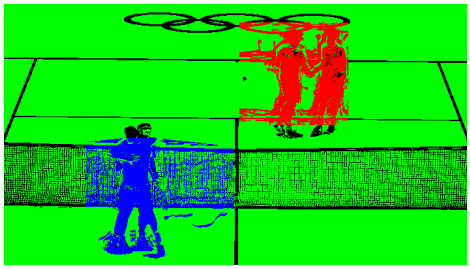
(a) **IoU:** 0.36

(b) **IoU:** 0.37

Figure 37.



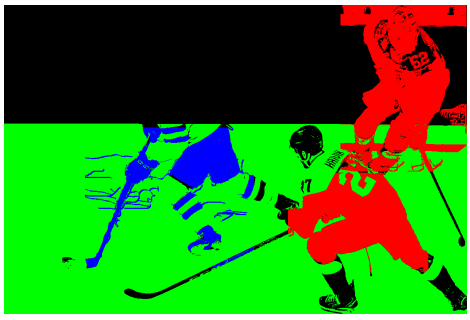
(a) **IoU:** 0.40



(b) **IoU:** 0.40

Figure 38.

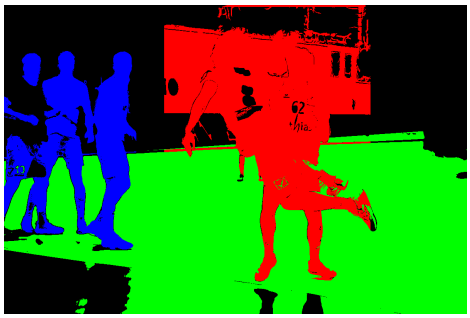
(a) **IoU:** 0.44



(b) **IoU:** 0.55

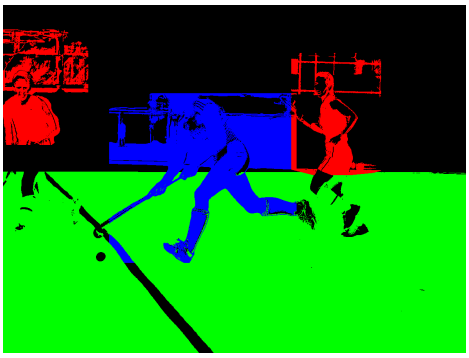
Figure 39.

(a) **IoU:** 0.55

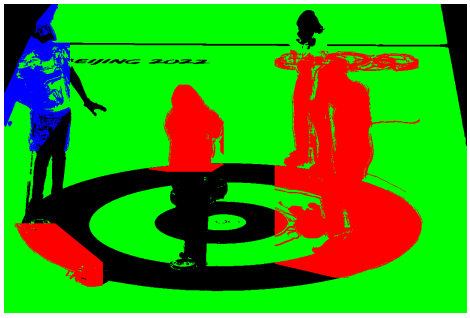


(b) **IoU:** 0.81

Figure 40.

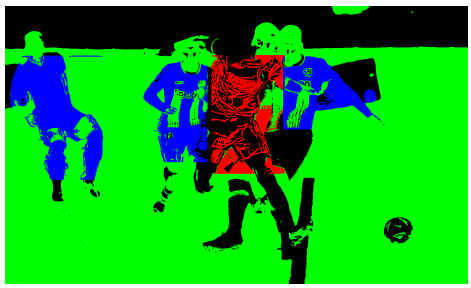


(a) **IoU:** 0.37



(b) **IoU:** 0.37

Figure 41.



(a) **IoU:** 0.28

Supplementary Test Images

Figure 42.

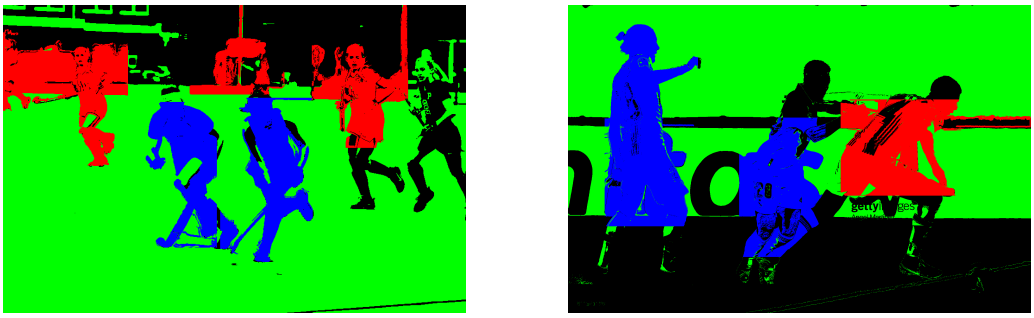


Figure 43.

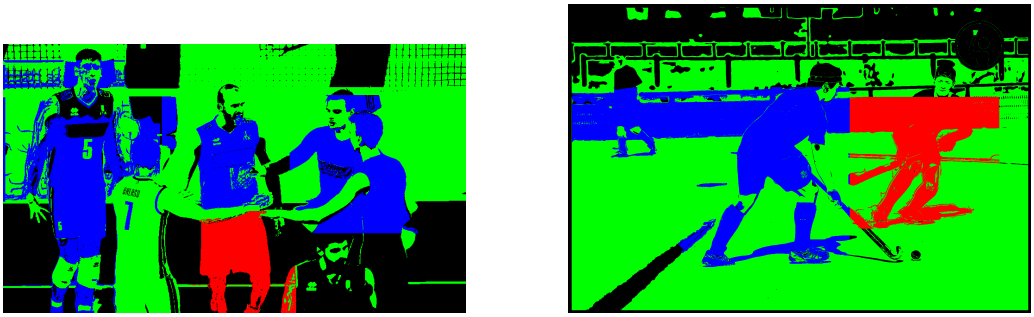


Figure 44.

Figure 45.

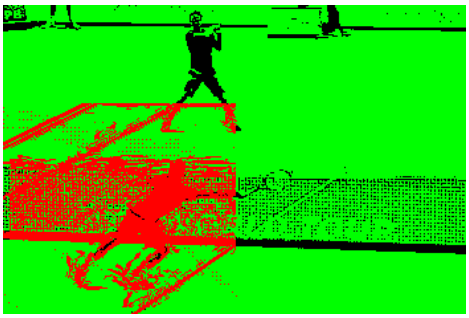
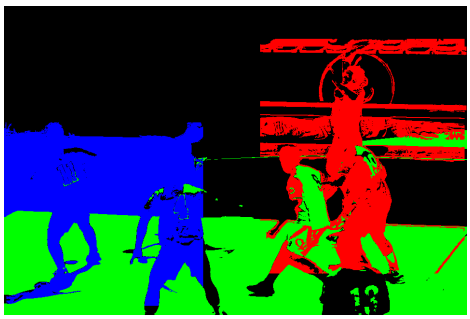


Figure 46.

Figure 47.



Conclusions

We employed a diverse set of tools and techniques to tackle this particular task, resulting in a highly adaptable and versatile codebase. Some of our approaches draw inspiration from computer vision course, such as the utilization of Local Binary Pattern (LBP) and Canny edge detection. Additionally, we've incorporated strategies and methods found in the existing literature.

The main problems found during detection happen in crowded images (some overlapping boxes of close people end up merged) and in images where due to perspective the net covers players, since the dataset contains many negative examples of nets, it ends up rejecting these proposed boxes (the problem remains when removing the samples from the dataset, causing furthermore false positives on nets in other images).
Another problem is given by the `canny_len` function, some noisy images provides an edge map where the edges aren't just the ones of the players, and counting the length to use for the boxes leads usually to bigger than needed ones, causing an involuntary merging in the post-processing phase (e.g. *im12.jpg*). This problem reflects in the **mAP**, some images have a performance of 0 despite providing correctly classified boxes.

In spite of these problems, it can be observed that our outcomes exhibit a degree of similarity to findings in the existing literature, especially when considering studies that did not utilize neural networks or R-CNNs. (Table 1) shows that our results are not that far away from similar tasks. On our pursuit of achieving precise image segmentation, we initially considered the *GrabCut* algorithm as our starting base. However, the initial results we obtained were not up to our satisfaction. Consequently, we decided to develop our own custom segmentation method, with the objective of surpassing the performance of *GrabCut*, as shown in (Figures 48, 49, 50).

Table 1. Comparison in detection rate at 0.25 with a similar task found in literature.

Our Results	Custom features [3]	HOG+SVM [3]	DPM + LSVM [3]	BS [3]
50.14	90.13	88.13	74.87	69.0

Table 2. Comparison in IoU over non-ML methods with similar tasks.

Our Results	MCCoSeg - Football players [5]	MCCoSeg - Baseball players [5]	Constrast-aware Co-segmentation [6]	Multi-spectral Clustering [6]
54.0	51.1	62.2	51.0	41.0

Figure 48. Comparisons between our approach and Grabcut.

(a) Our Approach (b) Grabcut. (c) Our Approach. (d) Grabcut.

Figure 49. Comparisons between our approach and Grabcut.

(a) Our Approach. (b) Grabcut. (c) Our Approach. (d) Grabcut.

Figure 50. Comparisons between our approach and Grabcut.

(a) Our Approach. (b) Grabcut.

The strength of this project is our segmentation approach that is able to detect the player inside the box, in most cases, especially when we don't have any noise ,(Table 2 shows some comparisons with literature methods). Furthermore as you can see in **Supplementary Image Tests** the tests we performed underline the robustness of our

method, in particular when we use the second approach for canny detection and fill image as explained in **Second Approach**.
Field segmentation performs exceptionally well in this specific task, demonstrating an high precision in detecting and distinguishing both the field and the background, even in challenging scenarios like those depicted in images 28a and 32a.
Moreover also classification is able to classify correctly in almost all the cases.

Future work

Since detection is the most problematic task due to the images complexity, implementing a deep learning approach would provide considerable positive effects on the whole project, fit boxes increase the success of the classification and a better segmentation. This approach while providing better results needs more computational power and a creation of a vast dataset.
Moreover also classification can have some improvements, particularly when dealing with situations where two players are within the same enclosure. To address this challenge, R-CNNs (Region-based Convolutional Neural Networks) emerge as an highly suitable tool. However, it's worth noting that implementing R-CNNs may demand increased computational resources due to its complexity.

A possible improvement for the segmentation can be a better clustering technique leveraging on similarity of patterns and also position pixels.

Systems used

On this section we want to insert the systems we used to run and develop this code:

- Windows 11, on AMD 2500U, 4 GB
- Windows 11, Ubuntu 22.04 on Intel Core i5-8300H, 8 GB

Contributions

The code has mostly been developed together, although the main focus of each one was:
Francesco Pio Monaco, 120h: SVM+HOG, Box clean-up tuning, Evaluation, Line refinement for playing field segmentation, Main file, Diffusion mask, Template matching approach, Canny line to choose the dimensions of the boxes approach
Michele Russo, 120h: Player segmentation, Playing field segmentation, Parameters tuning, LBP feature, Clusterization to clean the segmentation, Denoising and removal of non-connected components approach

References

1. Antoni Buades1, Bartomeu Coll2, J.-M. M. Non local means denoising. *IPOL* (2011).
2. Burges, C. J. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery 2*, 2 (1998), 121–167.
3. Cem Direkoglu, Melike Sah, N. E. O. Player detection in field sport. *International journal of computer vision* (2012).

4. Direkoglu, C., Sah, M., and O'Connor, N. E. Player detection in field sports. *Machine Vision and Applications 29* (2018), 187–206.

5. Joulin, A., Bach, F., and Ponce, J. Multi-class cosegmentation. In *2012 IEEE conference on computer vision and pattern recognition* (2012), IEEE, pp. 542–549.

6. Tsai, T.-Y., Lin, Y.-Y., Liao, H.-Y. M., and Jengg, S.-K. Precise player segmentation in team sports videos using contrast-aware co-segmentation. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2016), IEEE, pp. 1826–1830.

7. Uijlings, J. R., Van De Sande, K. E., Gevers, T., and Smeulders, A. W. Selective search for object recognition. *International journal of computer vision 104* (2013), 154–171.