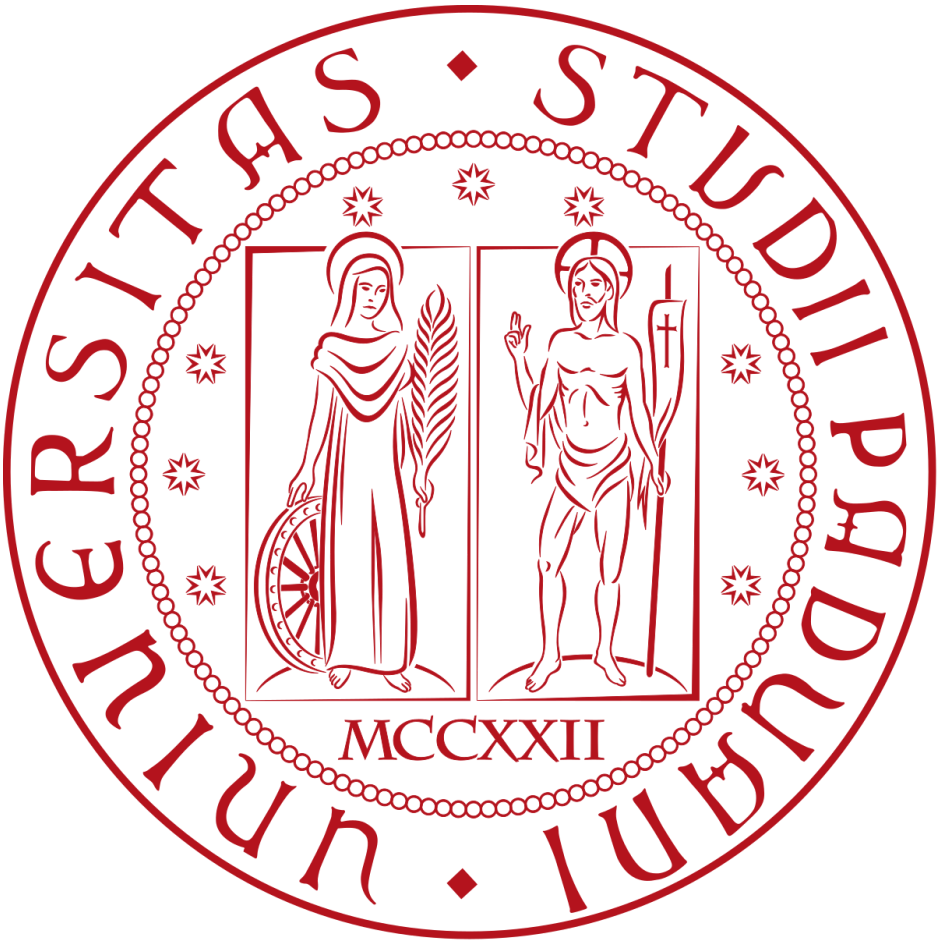
COMPUTER VISION PROJECT

Sport Scene Analysis



Pietro Girotto 2088245
Federico Gelain 2076737
Enrico D'Alberton 2093708

Pd, 24/09/2023

INDEX

The Project

The goal of this project was to conduct a comprehensive analysis of sports scenes. Our aim is to identify various elements within these scenes such as: players, the playing field, the background, and ultimately, determine the team to which each player belongs.
The given benchmark dataset comprises 15 sports scenes, each captured from different sports and viewpoints. To facilitate the evaluation of our performance, we have been provided with ground truth data, including grayscale masks, colored masks, and a text file containing bounding box information of the players. These masks utilize different colors to distinguish between the background, Team A, Team B, and the playing field.
The Github page of the project is the following: https://github.com/gp-1108/CV_Sport_Project.

Strategy used

We initiated our project by conducting an extensive brainstorming session to generate a wide array of ideas. After careful consideration, we selected a neural network-based approach as our initial strategy. We believed that this approach would be highly effective in accurately identifying individuals within the field.

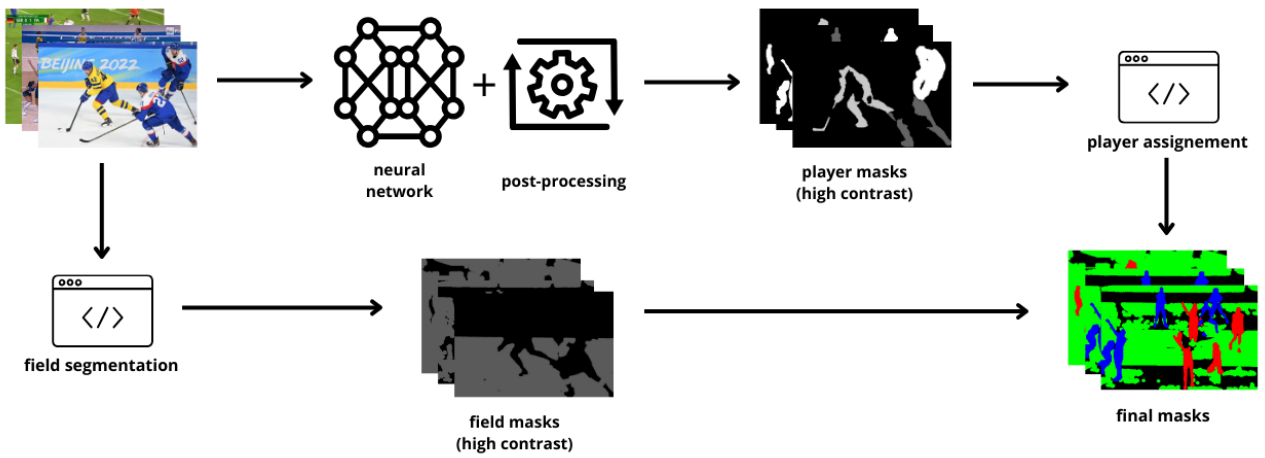
Following this decision, our project workflow was delineated into several key components:

- **People Recognition**: leveraging the power of neural networks, we aimed to efficiently recognize and identify individuals within the field.

- **Player Team Assignment**: this aspect of the project involved assigning recognized individuals to their respective teams.

- **Field and Background Recognition**: this part of the algorithm runs in parallel with people recognition to include the identification of the field and background elements.

This workflow is visually represented in the accompanying diagram.



Assigned tasks

We divided the workload in the following way:

- **Pietro Girotto**: spearheaded the "People Recognition" aspect, shouldering the responsibility for crafting a neural network dedicated to player recognition.

- **Enrico D'Alberton**: led the charge in the "Player Team Assignment" domain, crafting an algorithm designed to accurately assign each player's mask to the appropriate team.

- **Federico Gelain**: directed efforts toward "Field and Background Recognition," taking charge of an algorithm that adeptly discerns the sports field and its background in the scene.

It's crucial to note that despite this division of labor, we shared a robust spirit of collaboration and teamwork within the group. Each member actively contributed to one another's tasks with advice and new ideas, resulting in a highly synergistic workflow.
We want to specify that despite this synergy, each member of the group wrote their own code individually.

Segmentation Techniques

Field Segmentation (Federico Gelain)

Introduction to the problem

The main challenge of this task was to find a way to segment the field regardless of the sport. So a global method that would take into account the different configurations of the fields, the lines in which they are contained (if any), how “zoomed” they are (since they could cover the entire image or only a portion of it) and so on.

Ideas and Methods

The first idea that came to mind was to detect the contour of the field, using findContours, and color the field as the area within said contour. This proved to be unsuccessful because of images such the one shown below

since the view is zoomed in and there, and the following one

where the field line is of a very similar color to the field and occluded by the players.
For a similar reason, floodFill and watershed didn't work well, since defining the markers correctly proved to be more difficult than expected (even dividing the image in regions and choosing the markers using a uniform distribution).
So the focus shifted to color-based segmentation. Clustering based segmentation didn't provide good results because of the uncontrollable choice of the initial centroids, which didn't include the color of the field most of the time.

This last consideration sparked the hope that finding the color of the field and using it to segment the image could be a right approach, and indeed it provided the best results out of all the techniques described and attempted.
The process can be broken down into the following steps:

1. Color histogram computation (from the original images);
2. Field color candidates extraction (from the histograms computed in step 1);
3. Field color extraction (out of all the candidates extracted in step 2) and field mask computation.

Color Histogram Computation

For this task several color schemes were considered, in particular BGR, HSV (the one usually recommended for histogram computation) and YCbCr. BGR proved to give the best results, since the color of the field was detected correctly in all the test images.
It's important to note that for the computation of the histograms (one for each of the 3 channels, separately) 2 notable assumptions were made:

- The field is for certain located in the bottom half of the image. This works also if the image is zoomed (so the field covers the entire image), since the field to player ratio can be considered still greater than 1;
- The color of the field can be assumed to never be black or a color close to it. Also the matches are properly lighted to provide the best experience to the viewers, meaning that the darker areas are created only by the shadows of the players. In terms of code implementation, this meant that for each histogram the bins used for the computation of the candidates didn't start from 0, but from a slightly higher value suited for each one of the BGR channels (such as 20 or 30).

Field color candidate extraction

The color sample candidates were chosen by computing all the peaks (local maxima) for each one of the 3 histograms, which were previously smoothed (to keep only the most significant peaks), and picking an arbitrary number of the highest ones. For this project, choosing the first two peaks for each channel (resulting in 8 different candidates), provided the best results in terms of accuracy and computation. Picking the second peak regardless of its difference of frequency with respect to the first one (since usually it's recommended to pick it only if it's at least 80% of the maximum peak) proved to be the best choice, since there were cases, like the one shown below, were the highest peak is located in the darker bins (due to the players and the background) and is considerably higher than the second one (which corresponds to the correct color).

Field color extraction and field mask computation

Choosing the field color is similar to a clustering assignment and requires two steps:

1. Each one of the 8 candidates is considered as a centroid and to each pixel of the image (still considering the bottom half, not the entire image) is assigned the closest centroid. The distance function used is the following one:

$$dist = \sum_{i=0}^{2} \left| c_i - p_i \right|^2$$

where c is a centroid and p is the current pixel. The subscript indicates the current channel (since the color scheme considered is BGR, it refers to the blue, green and red channels), so c_i and p_i refer to the corresponding grayscale value for channel i.
After this, the centroids are sorted in decreasing order based on the number of pixels assigned to them;
2. Starting from the best candidate, generate a mask for the entire image, where all pixels whose distance from the candidate is below an arbitrary threshold are assigned the field label (3), the background label (0) otherwise. If the number of field pixels is enough (an arbitrary percentage of the total image area), then the candidate color is considered as the field color and the mask is returned.

At the end, some post processing to the mask is done. This aims to merge the smaller regions and attempt to remove the ones that were classified incorrectly. After testing several techniques, the ones used are the following:

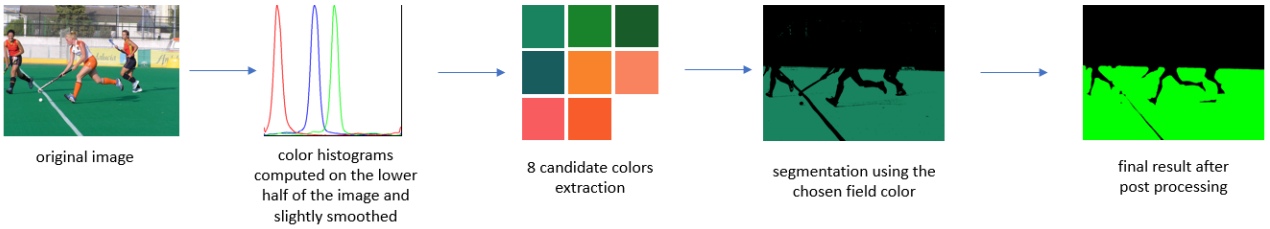
1. First, opening is applied to remove the smaller details, which can be considered as noise;
2. Then, to find the contours of the incorrectly segmented regions, the mask was first blurred using the Bilateral Filter (in order to preserve most of the edges), then the Laplacian Filter was applied to obtain a binary image of the edges and finally findContourswas used to obtain all the detected edges.
3. Each region detected whose area was below a certain percentage of the image (specifically 1.8%), was then reclassified as background.
4. At the end, closing is applied to reconnect weakly connected blobs that could have been lost from opening.

For both opening and closing, the morphological operator used is an ellipse, whose diameter is calculated as:

$$d = \frac{\alpha}{100}\sqrt{H^2 + W^2}$$

where H and W are respectively the height and the width of the image, α is an arbitrary parameter. (Cuevas et al.)

Below a scheme that shows all the steps applied to the thirteen image:



Conclusions and future work

There are clear improvements that could be done. First of all, most of the techniques used rely on arbitrary parameters that can be tuned, so certain combinations of them could result in greater results for some images, at the cost of greatly worsening others. The ones chosen for this implementation are the best trade off that yields the best possible results for all images.
Moreover, for post processing more possibilities could be explored. For example, a better way to obtain the edge mask (using for example Canny instead of the Laplacian filter, though it requires an extensive time to find the best parameters), better criteria to detect the contours to remove (other than their area) and more precise ways to merge the blobs at the end, instead of using morphological operations.

Player Segmentation (Pietro Girotto)

Introduction

As previously suggested, one of the hardest tasks was indeed players' detection and segmentation. To do so the team decided to fine tune a neural network for people segmentation. To do so different models and datasets were taken into consideration and tested.

Sample of segmentation output of the final predictive model

Datasets Used

For the datasets, the main issue was to decide which approach we wanted to use. On one hand we could have gone the “sports’ players detection” route by looking for sport-specific datasets, on the other hand we could have tried to create a “general people detector”.

Even though both strategies have their benefits and disadvantages, the final choice was to work towards a general people detector. The main motivations were indeed:

1. The benchmark scenes were heavily body-occluded, non-standard images of playing fields. As such, classical sport images taken from far away were not suitable for the task. Furthermore, the test images far more resembled general human segmentation datasets.
2. Datasets for sport scenes with provided segmentation for training were rare and not well distributed in all possible sports. Creating imbalances by training the network on specific sports by neglecting other ones was not suitable for the task and was clearly against the provided guidelines for the project that required a strong and robust model for various sports.

Two datasets were were the one utilized in this project for training:

- **MHP Dataset:** This dataset (which can be found here) consists of several images of multiple people in different scenarios with masks provided for each and every person in the frame for different body parts. Its key feature is the fact that there are many instances per image of humans, a crucial situation in most sport scenes. It was ultimately the one that led to best performances.
- **OCHuman Dataset:** The OCHuman dataset (found here) consists of several images of peoples with heavily occluded bodies between each other. This dataset was introduced in order to try to educate the system to better recognize occluded persons. However, given that the dataset was partially annotated (not all people were actually segmented for the ground truth) and the fact that only a couple of people were actually present per image, the models tested performed poorly and were struggling even more with the task.

A substantial part of training the neural networks was to traduce from one format of dataset to a suitable format for the model. Each model had its own metrics and standardized ground truth and it had been a task by itself migrating from one architecture to the other.

For the final model, it is possible to download the above mentioned datasets, and use the python script in the “dataset_manipulation” folder to actually create the final datasets used for training.

Models Used

For the models the ones tested were:

- **SegFormer B4 / B0:** This model is meant to be used for the semantic segmentation task. As such it was not properly thought of to be used for instance segmentation. Still, given its performances and capability it was tested as a first approach. B0 and B4 are just sligh variants of the SegFormer model where B0 is the lighter (less weights) one and B4 the heavier.
- **Mask R-CNN:** Mask R-CNN is a popular model created in 2019 designed for instance segmentation. As such it was a visible improvement over SegFormer even though it required more troubles to work with. Still performances were far from acceptable as it was clearly

struggling with partially occluded bodies, something really important given the sport settings and the benchmark images presented.

- **YOLOV8-Seg:** To overcome the occluded bodies issues, further research was made into discovering newer and more robust models than Mask R-CNN. The final choice was indeed YOLOV8-Seg as it was recent, easily exportable in ONNX format and proved to be the best one out of all the previous ones.

° SegFormer

SegFormer is a semantic segmentation model designed in 2021 with the purpose of resolving most tasks of zero-shot segmentation. Two versions of the model were tested: B0 and B4. B4 is the more advanced version with increased runtimes traded for more accuracy and complexity.
The instance trained for this project was pretrained on the ADE20K dataset and further fine tuned on an instance segmentation task for human recognition.
The two tasks are clearly not compatible at first, to somehow traduce an instance segmentation task into a semantic one the following approach was used:

1. Each person in the ground truth was given an integer index from 1 up to N. Background was labeled 0.
2. The true mask presented to the model was a matrix [width, height] in which each entry corresponds to one pixel and has the integer value of the corresponding label in the ground truth.

The outcome of such training was nothing short of miserable. The model selected for the task was clearly capable of distinguishing “person” from “non-person” pixels, however introducing all of that noise with different indexes for the same category (person) was causing some interesting jittering effects as well as disastrous final masks.
Both the OCHuman datasets and the MHP one gave similar results.

Sample output from SegFormer. Each color is an instance

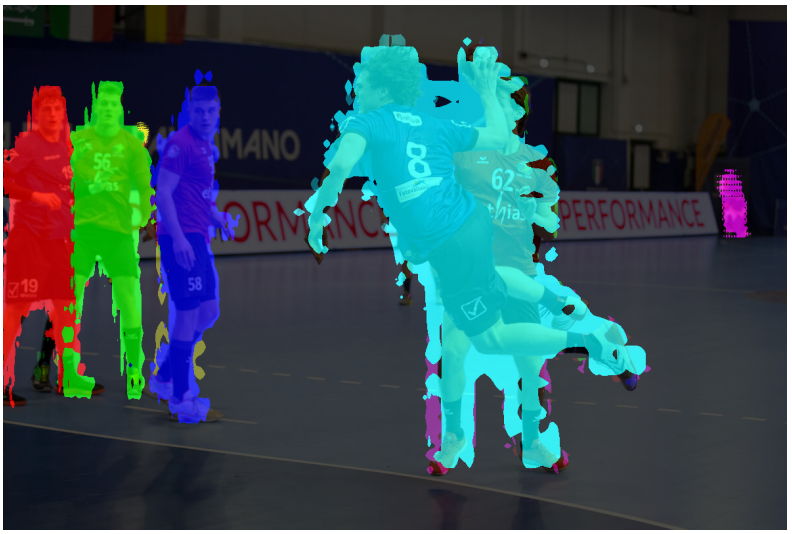
° Mask R-CNN

Mask R-CNN is a less recent model compared to SegFormer, as it was firstly released in 2019, but better suited for the task at hand. The model was taken from the PyTorch library and fine tuned on both aforementioned datasets. In this case results were far more promising, as the model was intended for such usage.

The model was tested with different fine-tuning combinations depending on:

- The dataset used (MHP or OCHuman)
- The part of the model fine tuned (just the decoder heads or the whole model too)

The results, even though there were clear improvements over the previous model, were still far from satisfying. The model persisted in those ringing effects as well as struggling greatly with semi-occluded bodies. No amount of parameters fine tuning seemed to even slightly change this situation, so further time was invested into researching either a more recent model, or one specialized in human parsing with occluded bodies.
Another interesting problem was found in managing the game ball, if present. MASK R-CNN was eager to label it as a person, no matter how far from the truth it was.



Sample from Mask R-CNN outputs, each color is an instance

° YOLOv8-Seg

After researching for a newer model, capable of better handling the occlusion problem, the final choice was indeed YOLOv8.
YOLOv8 comes in different flavors for different tasks (Localization, Segmentation, Classification, Pose), therefore the “-Seg” suffix, to indicate the one used.
This model was published in 2023, as such it was a great advancement from MASK R-CNN and was far better in recognizing people's shapes.
The model comes in different sizes (n-s-m-l-xl), the one used was the medium one (m) as it was way faster than the large version but with no noticeable detriment in performances.

This neural network was pre-trained on the COCO Dataset and then fine tuned as a people detector on the MHP dataset. Even though the OCHuman dataset was far more useful for occluded bodies, still the fact that on average just 2 people were present per frame led the neural network to precisely detect only a small portion out of the actual persons during inference.
This reason led to the usage of the MHP dataset, of which multiple variants were produced to fit the YOLO format for training.

While training this neural network, it is important to convert the dataset to a format YOLO-compatible. As such each segmented instance must be expressed by a polygon with a finite number of vertices.

However, such a situation can become extremely noisy when the true mask consists of unconnected components, in which case there was the need to find a way to properly add them together

Different variations were tested:

- **ALL VERTICES:** Simply stack together all vertices of all unconnected components, even if it can create potential noise and corrupted masks.
  Even though one might think it is far too simplistic, this approach was capable of expressing most of the masks quite right.
- **GREATEST POLY:** Find the greatest unconnected region per mask and simply discard any other polygon present.
  This method keeps the shapes intact, however aggravates the occlusion problem because the model would not be able to properly segment regions from unconnected masks as one single instance.
- **MASK APPROXIMATION:** Detects the center of the single mask. Given this centroid, compute the angle and distance of each point of the mask and for a given range of degrees (dividing the angles in bins) keep only the furthest point from the centroid to avoid adding points that would otherwise disrupt the mask.
  A balance between keeping the shape of the true mask and educating the system to occluded bodies.

The fine tuned model clearly understood the task correctly and was capable of segmenting correctly many images in the benchmark dataset.

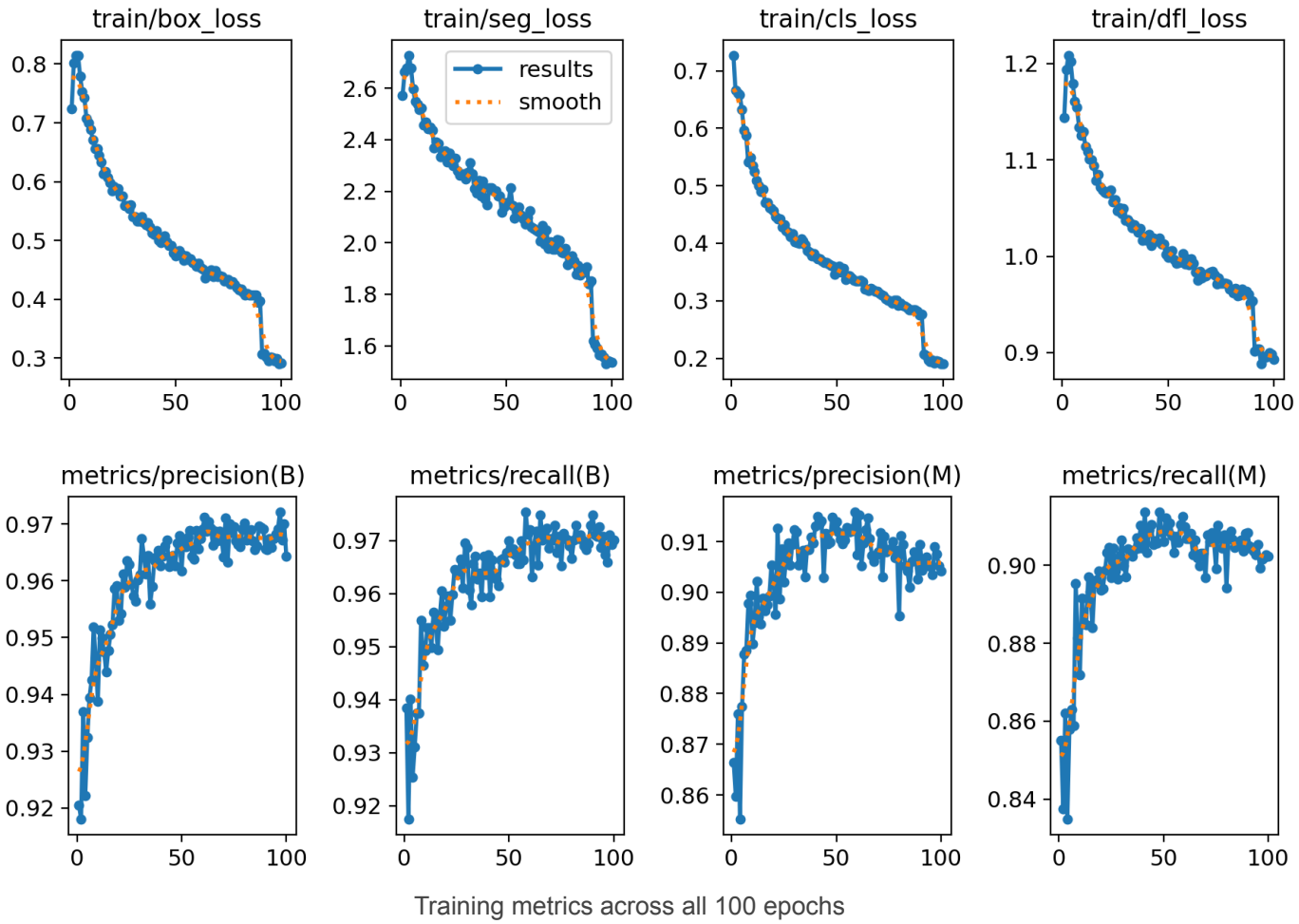
Interestingly enough, the dataset that allowed the model to perform the most precisely on the benchmark dataset was the “all vertices” one. Even though some masks were corrupted because of this raw approach, the model was still capable of understanding the task and creating correct masks as well as identifying the most number of players across all images.

It is to be noted that, even though model performances might seem underwhelming on the benchmark, it is a hard task even for currently SotA models to perform well in such images and conditions. The constraint of porting the model to C++ greatly reduced the number of options available and even for the best ones these were quite hard instances to segment. For more traditional people segmentation tasks the model performs amazingly well and precisely. Further confirming that the training was indeed effective can be found in the metrics shown during training (see training section for reference) as it clearly performs better and better on the task with the increase of the epoch number.

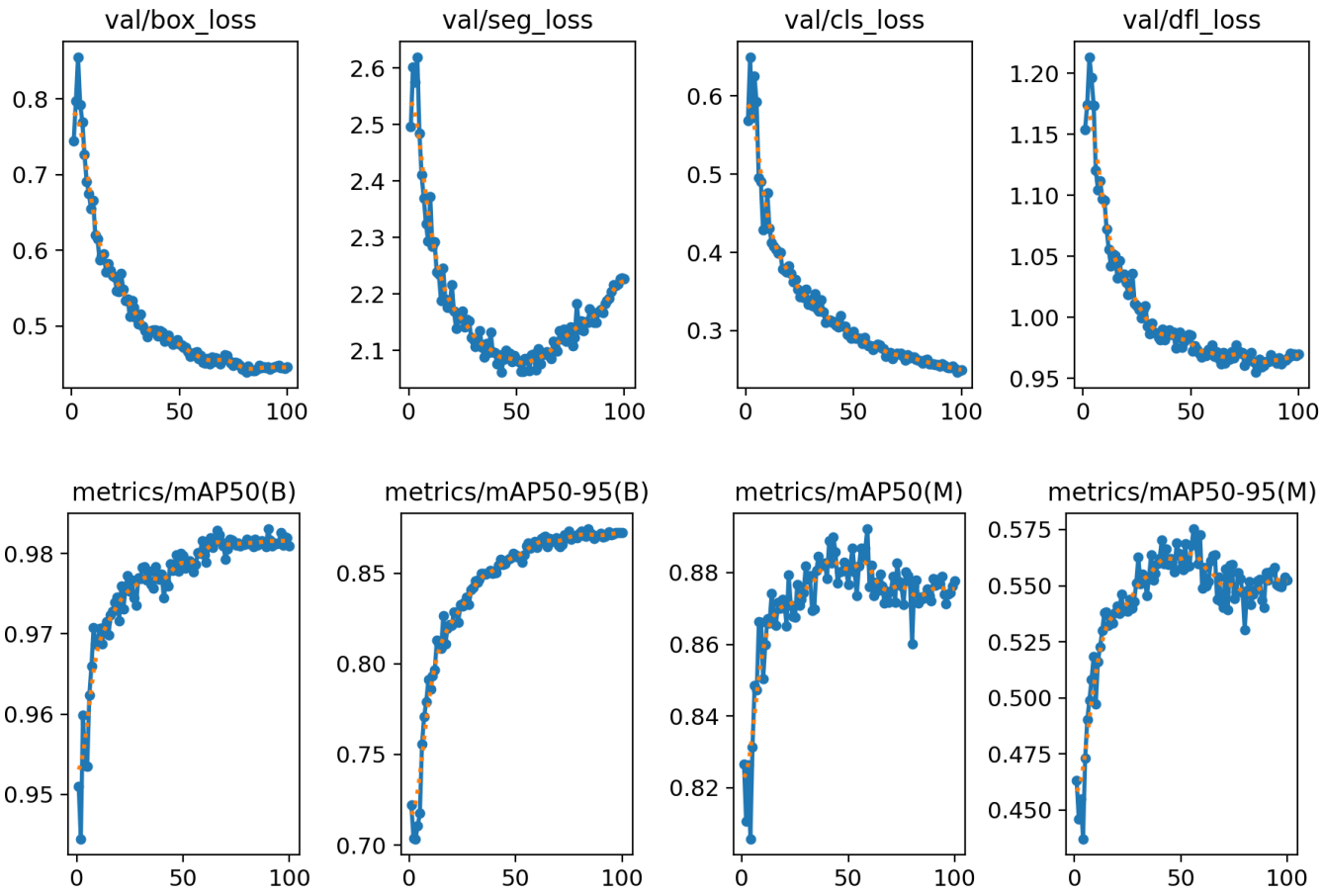
Training

With the exception of SegFormer, which was trained on Google Colab, all other models were trained on the ClusterDEI architecture with the usage of SLURM and Singularity on two GPUs.
Generally speaking training was performed with the SGD optimizer, using PyTorch (ULtralytics library) and inference results were taken from the models once ready for production.

The following chart is the train/validation losses from the training. The model picked was the best one on the validation set across all 100 epochs.



Training metrics across all 100 epochs



Validation metrics across all 100 epochs

It is clear that the model, in fact, learned from the training process to correctly identify persons in different contexts.

NN Evaluation and Results

The evaluation of the neural networks, from one to the other, did not necessitate any metrics definitions. Results were getting clearly better just by sight:

SegFormer | Mask R-CNN | YOLO V8

Each color represents one instance of class “person”

For the final results, the actual metrics used were the one on the whole architecture, as it was clear if the model was getting better or not. Some final output images of the validation set are therefore added to show the model final potential.

Conclusions and Future Works

There is clearly room for improvement. Using a Pose Estimator combined with a neural network for decoding and generating single masks could be an even more effective way of tackling the problem. With respect to YOLO, perhaps fine tuning the model on pose estimation and using it to feed a post-processing neural network could lead to significant improvements. Still, the constraint of producing a model that has to be run on C++ drastically limits the choice of a model suitable for this task, as most SotA NNs are indeed only available in Python and non-portable to C++ without investing a sensible amount of time into it.
Eventually, the model performed discreetly on the benchmark set and really well on non-occluded images. It is an acceptable performance as even SotA results struggled with such images.
Overall, with a greater time span, results could have been refined to SotA-comparable through the extension of the neural model.

Classification Task

Player Team Assignment (Enrico D'Alberton)

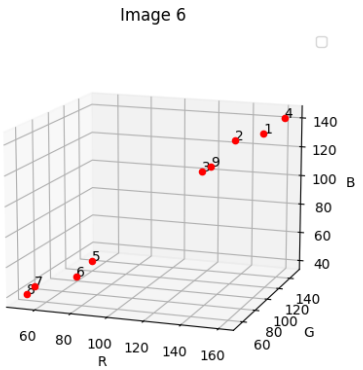
Introduction to the problem

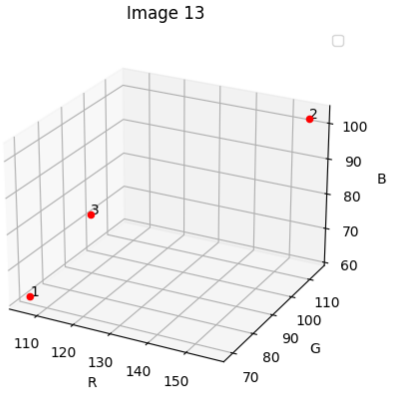
The primary objective of this task was to accurately allocate each player to their respective team based on the players' masks. Consequently, this aspect of the algorithm relies, to some extent, on the performance of the neural network responsible for player segmentation in producing reliable outputs.

Idea

Before diving into code development, I initiated a brainstorming session to explore various approaches for player clustering. Ultimately, I opted for the most intuitive method: grouping players based on the

colors of their team uniforms. To achieve this, I began by processing the players' masks through the application of filters, primarily GaussianBlur and pyrMeanShiftFiltering. Subsequently, I examined the histograms of the three color channels.
In the next step, I made the decision to represent each player using their average RGB values, which I then visualized in a 3D plot. In this unique representation, each axis corresponded to a primary color. Below, I share some of the obtained outcomes:

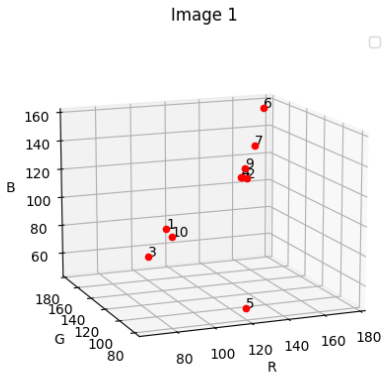


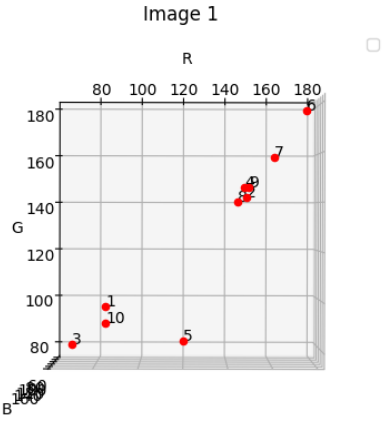


Observing the images I noticed that in most of the scenes, the players could have been divided in two different clusters.

Methodology

I opted to experiment with clustering by applying the k-means algorithm with k=2. The initial outcomes showed promise, leading me to commit to this approach. Notably, during my exploration of the data, I observed that when projecting the points onto a 2-dimensional plane (specifically, GB and RG), it produced a more distinct separation of the clusters. For instance:





As evident in these visual representations, Image 1 exhibits a clearer demarcation of two distinct clusters when we project the data points onto the RG plane. Consequently, I have opted to perform multiple

k-means clustering iterations with different "planes" and subsequently assess the outcomes through the utilization of the silhouette coefficient for comparison. Moreover, I have tried other clustering methods, but k-means was always the best one, so I decided to remove them.

The algorithm

The player assignment algorithm operates through a series of well-defined steps:

- **Invocation of Master Function**: To commence the process, the master function assignToTeams() is invoked individually for each scene. This function serves as the gateway to the entire set of functions within the corresponding .cpp file.

- **Mask Extraction**: Following this, the algorithm extracts colored masks of the players from the post-processed mask produced by the neural network.

- **RGB Average Calculation**: For each player, the algorithm calculates the average RGB values.

- **K-Means Clustering**: Next, the algorithm employs three separate instances of the k-means clustering technique. One operates on the RGB plane, another on the RG plane, and the final one on the GB plane.

- **Silhouette Coefficient Selection**: Among the three clustering outcomes, the algorithm selects the one with the highest silhouette coefficient, indicating the most optimal clustering solution.

- **Final Output Generation**: With the chosen clustering in hand, the algorithm generates the final images and a bounding box .txt file, which may then be optionally saved for further use.

Confidence computation

Throughout the algorithm, an estimation of a confidence value is generated for each individual player. This estimation is achieved by amalgamating the silhouette coefficient of the clustering along with the distance of the data point from its corresponding centroid.

$$confidence = silhouette * \frac{distance\ from\ the\ centroid}{max\ distance\ from\ the\ centroid}$$

This confidence value is later used for the computation of the performances.

Conclusions and Future Works

There is space for improvement by optimizing the filter combinations to boost performance. Additionally, we can explore alternative player representations, such as one that incorporates color variance of the player's mask. Moreover, it may be worthwhile to experiment with a new clustering method. There are numerous possibilities to explore, and through further research and testing, we can reach improved performances.

Results

Defined Metrics

For this project, the performances of detection and segmentation were measured respectively using the following metrics:

- mAP (mean Average Precision);
- mIoU (mean Intersection over Union).

mAP (mean Average Precision) (Kukil)

As the name suggests, this is computed as the average of the AP (Average Precision) over all detected classes. For this project, the classes detected are only 2:

- team A;
- team B.

since the detection only looks for players and decides to which team they belong to. So

$$mAP = \frac{1}{2}\left(AP_{Team\ A} + AP_{Team\ B}\right)$$

An extensive description on how the AP metric is computed for each class is given below.

AP (Average Precision)

AP is computed using the **PASCAL VOC 11 Point Interpolation Method**. This requires to compute the precision-recall plot first, and will interpolate the precision values using 11 recall values: {0, 0.1, 0.2, …, 1.0}. Before talking about how the interpolation works, let us focus on how the precision-recall plot is computed.
Like the name of the plot suggests, 2 quantities are needed:

- precision (P), which is given by the following formula: $P = \frac{TP}{TP+FP}$
- recall (R), which is given by the following formula: $R = \frac{TP}{TP+FN} = \frac{TP}{\#ground\ truth}$

Starting from the predicted bounding boxes, they are sorted in decreasing order based on their confidence value (which was described before), and then, one by one, compared with the corresponding ground truth bounding box. The comparison consists of computing the IoU (Intersection over Union) values, as:

$$IoU = \frac{\#intersection\ pixels}{\#union\ pixels}$$

Since for each predicted bounding box the exact corresponding ground truth bounding box isn't known, we choose the one that has the same label and yields the highest value of IoU.

The algorithm to compute IoU works in this way:

1. draw the bounding boxes separately in two different masks, filling the area within them. The masks are big enough to draw the bounding boxes in their original position in the sports images;
2. compute the intersection mask multiplying the two masks;
3. compute the union mask summing the two masks;
4. calculate #intersection pixels and #union pixels using the function nonZeroPixels for the respective masks (this function returns the number of pixels in the mask whose value is different from 0);

5. compute IoU using the formula above (it's clear that its value will be in the interval [0, 1]).

The value is then compared with a threshold (set to 0.5):

- if it's greater than the threshold, then we have found a TP (True Positive);
- if it's less than the threshold, then we have found a FP (False Positive). Since they are not required for the recall computation, FN (False Negatives) will be considered as FP, since they basically correspond to a IoU value of 0.

Keeping a counter for the number of TP and FP found so far, precision and recall are computed each time one of these values change and "stored" as a plot point (currentPrecision, currentRecall). For how the recall is defined, those points are sorted in non decreasing order of recall.
Note: until a TP is found, precision and recall will both be 0 (useless for the plot computation). This is why the predictions are sorted by confidence, since it helps to find a TP as soon as possible.

We're now ready to compute AP. For each recall value in the interval S = {0, 0.1, 0.2, …, 1.0}, let us call it interpolated recall, compute the interpolated precision value as the maximum precision associated with a recall value greater or equal than the current interpolated recall. As an example, if the interpolated recall is 0.7 and the precision recall plot consists of the points (0.7, 0.13), (0.52, 0.7), (0.26, 0.81), the interpolated precision value will be 0.52.
After computing the interpolated precision value for all the interpolated recalls, we can finally compute AP as:

$$AP = \frac{1}{11} \sum_{i \in S} interpolatedPrecision_i$$

mIoU (Mean Intersection over Union) (Kukil)

Like the name suggests, this is computed as the average of the IoU computed for each class of the image:

$$mIoU = \frac{1}{4} \sum_{c} IoU_c$$

For this project, the classes involved in segmentation are:

- background (label 0);
- team A (label 1);
- team B (label 2);
- field (label 3).

For each class c, IoU is computed in the following way:

1. given the full segmented predicted and ground truth masks (with all 4 classes) create the corresponding binary mask where all pixels labeled with label c are set to 255, everything else is set to 0;
2. compute the TP area as the intersection of the two masks, which is computed multiplying the two masks (basically applying the AND operation);
3. compute the union area summing together the two masks (computing separately the FP and FN areas is not necessary, it's only useful for printing purposes);
4. compute $IoU_c = \frac{TP\ area}{union\ area}$

Numerical results

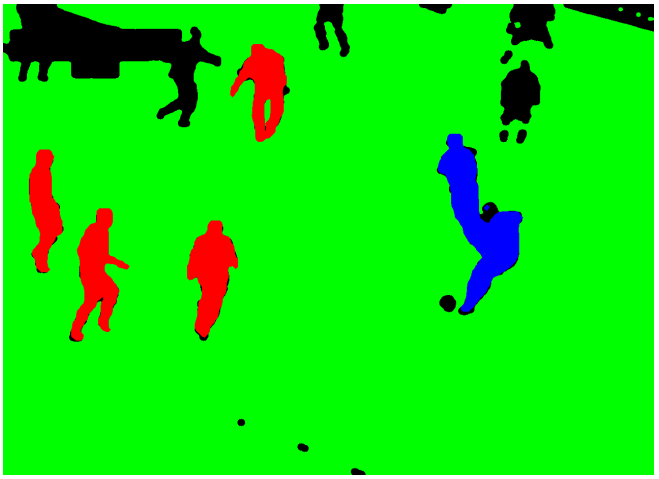
Average mAP (across all images): 0.733;
Average mIoU (across all images): 0.667;

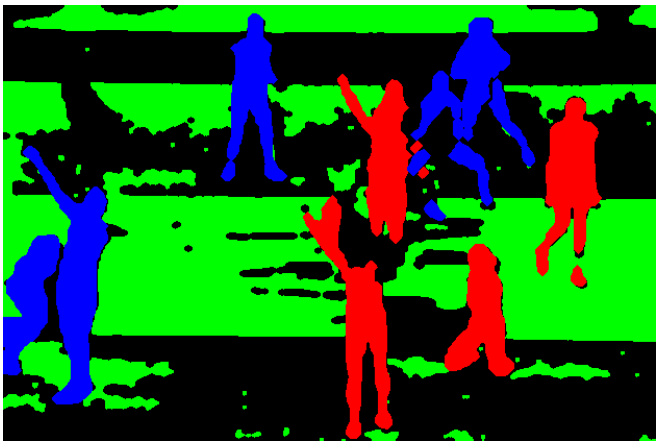
Image	mAP	mIoU
	0.273	0.540
	0.482	0.375
	0.606	0.675
	0.773	0.665
	0.864	0.782
	0.591	0.643
	0.773	0.758

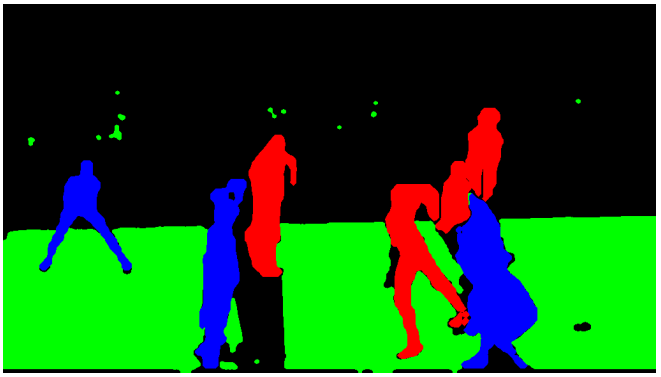
	0.773	0.753
	0.773	0.572
	1	0.715
	0.318	0.589
	1	0.649
	1	0.919
	1	0.663

	0.773	0.705

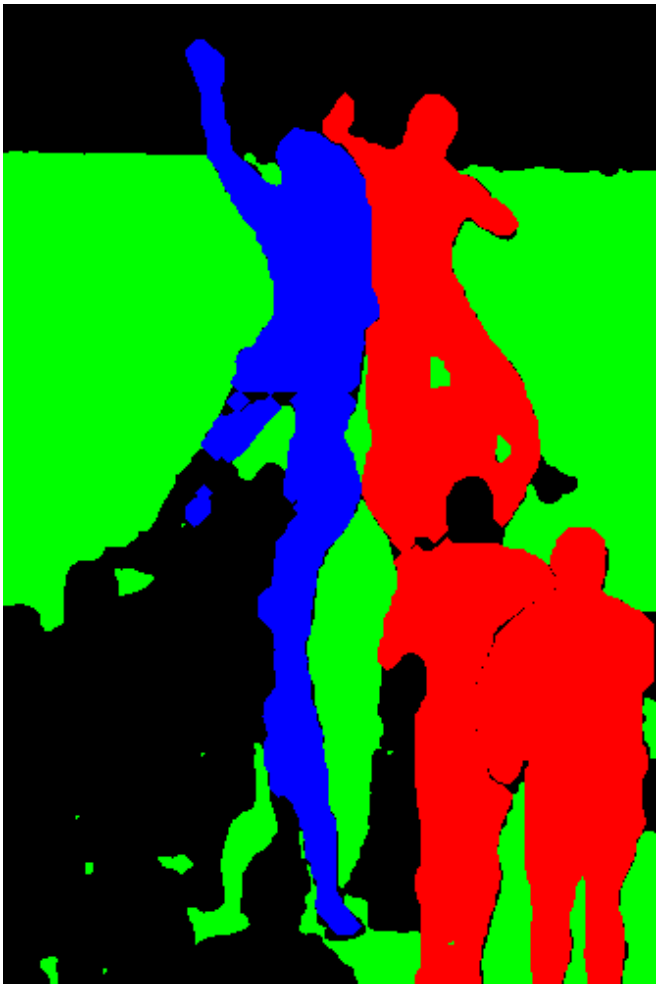
Images Results



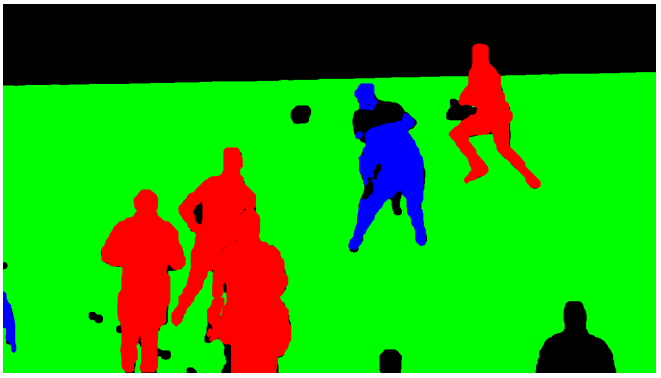


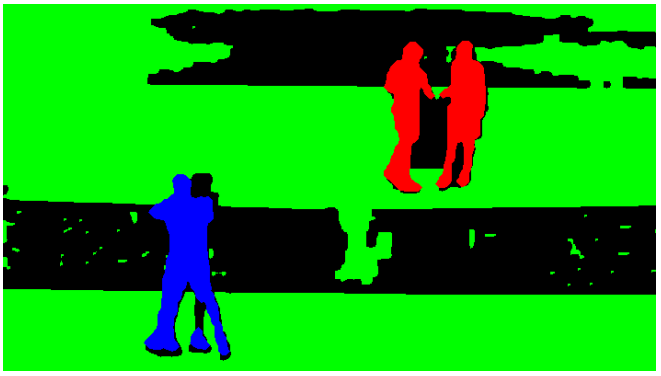



8
5
4
3

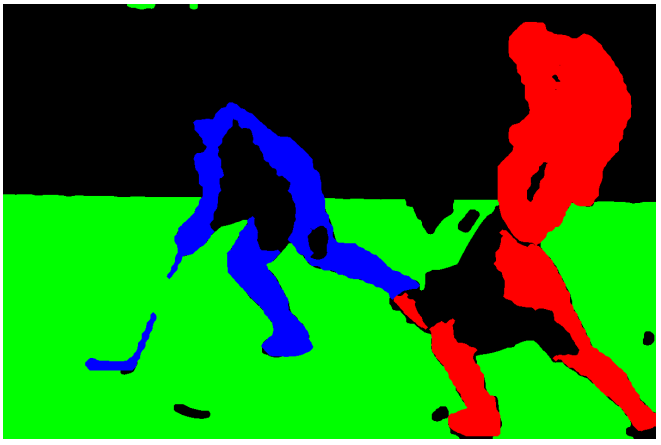



NEC



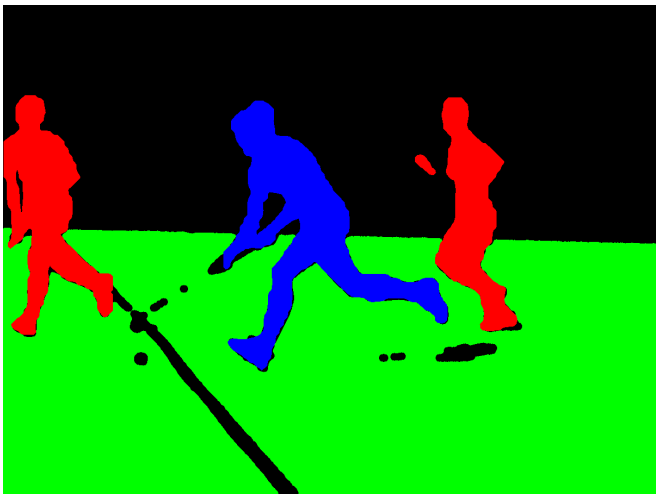


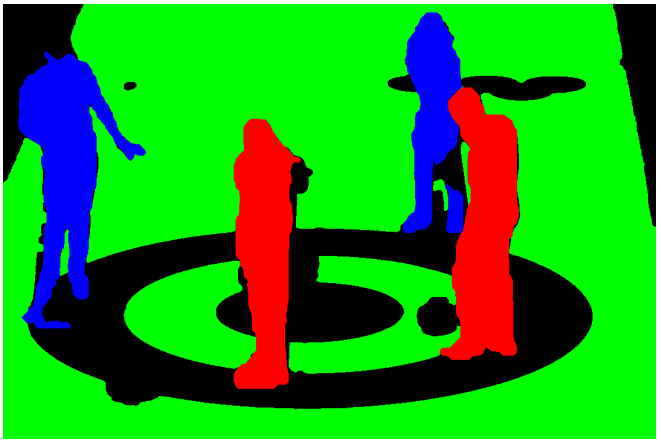

BEIJING 2022







Analysis

Challenges and problems

- **Federico Gelain:** Regarding field segmentation, the biggest challenge was to find a general method that would work regardless of the image conditions and that didn't need to rely on other parts' performances to perform well. The final approach used, for example, has to deal with unwanted contributions from the players when it tries to find the field color, since it has no information about where they are. It's also important to stress out that color-based segmentation has many limitations, due to differences in illumination in the image and similarities of the field color with non field areas. Out of all of the techniques attempted though, this proved to be the least concerning problem.
  Moreover, finding what could be done as post processing to improve the results of the color segmentation proved to be extremely difficult, since it essentially had to deal with the limitations described above, while keeping what was instead segmented correctly.
- **Enrico D'Alberton:** when it comes to the task of assigning players to teams, our primary challenge was to discover the optimal combination of filters that would guide the algorithm toward achieving the most precise clustering. It's worth acknowledging that finding the absolute best method proved difficult to catch, given the multitude of filter combinations at our disposal. Furthermore, we extensively deliberated on the importance of the confidence score, recognizing its crucial role in computing the average precision for each class and ultimately determining the

final mean Average Precision (mAP). We grappled with the question of which confidence score to use, given that various components of the algorithm contribute to generating the final mask for each player. Ultimately, our choice was to incorporate a confidence value that took in consideration how much a point (player) belongs to their respective cluster (team). This decision has been made by the realization that the overall quality of the clustering outcome heavily relies on the performance of other parts of the project's algorithm. If these other components were not functioning optimally, it would invariably weaken the quality of the clustering results.

- **Pietro Girotto:** For certain, limiting the number of models to those who can be ported in C++ with reasonable effort made the task exponentially harder. Not only I had to find a powerful model but, even before training it, it was required to port it (with all the related work required) to make sure it was a feasible solution in the first place. This workflow heavily affected the number of models that could be taken into consideration, as well as wasting time on models that ended up performing poorly regardless of the possibility of porting them. As highlighted before, the task itself is quite hard for CNN (even SotA ones) for occluded bodies. Therefore, fine tuning the network to find acceptable results was quite insidious.

Conclusions and future work

In general, the project is a good starting point for a true state of the art approach to the problem. With regards to field segmentation, the results are really promising with an overall accuracy of 80% and blazing fast speed for detection. For player assignment instead, the clustering could be improved by exploring alternative player representations, such as one that incorporates color variance of the player's mask. Moreover, this is probably the most time consuming task and some optimization might be needed depending on which hardware it is run onto. Finally, for the player segmentation task, more research and development on composite neural networks could lead to better results in occluded instances. Overall, performances, even though are not SotA comparable, are promising for future works.

Miscellaneous

To be noted: All commands are meant to be executed starting from the root folder of the project.

Usage

To run the following script it is required to have OpenCV 4 installed. As the usage, functioning and support of the cv::dnn module of OpenCV changes quite often it cannot be guaranteed that the model will run on any version >= 4. Specifically the project was developed, builded and tested with the following versions:

- **OpenCV 4.8.0**
- **OpenCV Contrib 4.8.0**

For reproducibility, in the folder “OpenCV_Environment” can be found the Singularity definition file “opencv_env.def” as well as the package list “packages.sh”.

To actually build the Singularity image use the following command:

```
cd OpenCV_Environment
sudo singularity build cv_sport_project.sif opencv_env.def
```

Building

To build the project (without singularity) use the following commands:

```
mkdir build
cd build
cmake ..
make
```

If the singularity image is being used then you need to execute the same commands, but through the singularity:

```
singularity exec OpenCV_Environment/cv_sport_project.sif bash
OpenCV_Environment/build_command.sh
```

Note that the above command does also the inference too.

Execution

Once you follow the instructions for building you are ready for inference.
By default it is possible to run the model without specializing any particular option as follows:

```
cd build
./main ../models/best.onnx ../Sport_scene_dataset ../Model_output
```

The results will be saved in the Model_output folder. The metrics computed will be inside the build folder.

If you want to, it is possible to adjust the parameters of such a command.
To do so execute the following command:

```
cd build
./main <path_to_model> <path_to_benchmark_dataset> <output_folder_path>
```

The path to model is the path to the .onnx file of the exported model. The benchmark path is the path to the folder structured by the rules defined in the project assignment.
This is also valid for the instructions contained in the OpenCV_Environment/build_command.sh

Workload

	Enrico D'Alberton	**Federico Gelain**	**Pietro Girotto**
Total hours estimated	80h	80h	100h

Bibliography

Works Cited

Cuevas, Carlos, et al. “A fully automatic method for segmentation of soccer playing fields.” 26 January 2023, https://rdcu.be/dmPxd. Accessed 23 September 2023.

Kukil. “Intersection over Union (IoU) in Object Detection & Segmentation.” *LearnOpenCV*, 28 June 2022, https://learnopencv.com/intersection-over-union-iou-in-object-detection-and-segmentation/#IoU-in-Image-Segmentattion. Accessed 20 September 2023.

Kukil. “Mean Average Precision (mAP) in Object Detection [TL;DR].” *LearnOpenCV*, 9 August 2022, https://learnopencv.com/mean-average-precision-map-object-detection-model-evaluation-metric/. Accessed 20 September 2023.