Scuola universitaria professionale della Svizzera italiana
Dipartimento tecnologie innovative

SUPSI

Subbuteo Monitor

Student/s

Nathan Margni

Supervisor

Del Rio Giacomo

Co-relator

Host

IDSIA

Course

-

Module

Thesis

Year

2023

Contents

List of Figures

Abstract

Subbuteo is a traditional table soccer game, it offers an fun physical game where players use their fingers to kick pieces on a field and score goal. This thesis explores the possibilities and methodology of integrating computer vision techniques with the Subbuteo gameplay, aiming to detect and track in-game elements, such as the pieces, ball, and corners. We explored both basic computer vision and deep learning approaches, notably the YOLO algorithm, a fast state-of-the-art technology for multiple object detection, utilizing video data obtained with an overhead webcam.

The findings indicate that while basic techniques offered some utility, the YOLO model exhibited better performance in object detection and tracking. Moreover, challenges arise in differentiating similar colored/shaped elements especially with the basic approaches.

This thesis gives the foundation for potential future improvements, such as real-time gameplay insights and event notifications, statistical analysis etc. giving a more immersive experience for Subbuteo players.

The integration of this type of technology into a classic physical game not only enrich the gameplay experiences but also underlines the potential of computer vision in improving traditional recreational activities.

Chapter 1

Introduction

1.0.1 Objectives

Subbuteo is a table soccer game where players have to kick the pieces on a miniature soccer field with the finger in order to score goal. The objective of this thesis is to develop a program able to detect the position of the various elements in the field, such as pieces, ball and corners; this information can be subsequently used to detect events that occur in the game, like goals and offsides

1.0.2 Problem definition

The proposed solution involves various computer vision methodologies to process live video from a webcam placed above the Subbuteo field. This video data will be used as input for a Python application, which will take multiple steps to achieve the desired output. The steps include:

- **Data acquisition and preprocessing:** The webcam will record frames from a subbuteo game with a view from the top, incoming frames from the webcam are subjected to preprocessing techniques to optimize their quality and prepare them for analysis.
- **Object Recognition and Tracking:** Frame by frame analysis is executed to find the spatial coordinates of distinct elements such as player pieces, the ball, and corner markers. This involves the use of computer vision algorithms capable of identifying and tracking these entities.
- **Annotation and Visualization:** The acquired positional information is annotated into the original frame, creating an annotated representation of the game in a digital form.
- **Event Detection and Notification:** This final step would be a possible future improvement, using coordinate information of the various elements, to detect in-game events like goals and offside instances. These events could be signaled to players, enriching their engagement and awareness during or after the gameplay.

In summary, this thesis consists in creating an application that brings Subbuteo status into the digital form, giving players useful and interestintg insights during and at the end of the match

1.0.2.1 Existing technologies

There are not specific Subbuteo project like this in the existing literature, but there are some existing deep learning techniques that can be used and fine-tuned for this task, such as the YOLO (You Only Look Once) algorithm.

Chapter 2

Data

The quality of data is crucial for training algorithms and for their application during both training and inference phases. In this section, we'll explain the kinds of data used and how it was collected and processed.

2.0.1 Data acquisition

To realize this project, the first step was to find a way to obtain our input data, the frames. This involved the research, selection and configuration of cameras to capture the Subbuteo game using Ubuntu as operating system.

1. **Monitoring camera experiment:** The initial idea was the utilization of an avaiable Mako-g camera with PoE (Power over ethernet) connection (ref: [mak]). In order to configure this camera a series of driver and software were installed and configured and a dedicated IP address created, since a ethernet connection between the camera and the laptop was necessary.

   Unfortunately in the end the quality of the captured frames remained unstatisfactory, especially because of the poor quality of the images, even after trying with different lenses.

2. **Webcam experiment:** Next a decision was made to switch to a Logitech webcam with 4k resolution up to 60 Hz (camera: Logitech v-u0040 [log]). Unlike the previous camera, setting this system was much simpler, requiring less software intervention. But a challenge emerged when attempting to record with high framerate and resolution due to software limitations.

3. **Solution to limitations:** To address the issue of recording at high framerates and resolution the use of a terminal command was used to enable the use of all avaiable resources in the recording process. Increasing noticeably the limits.

The specific command utilized is the following (2.1):

Listing 2.1: Terminal command to record with maximum resources

```
ffmpeg -f v4l2 -framerate 30 -video_size 4096x2160 - input_format
   mjpeg -i /dev/video2 -c:v libx264 -preset ultrafast -crf 23
   output.mp4
```

Where ffmpeg is a recording software and the "-preset ultrafast" command allow to record a video at 4k and 30Hz exploiting all possible resources in the machine.

2.0.2 Data preprocessing

The next step was handling the acquired data, which consisted of video recordings of actual gameplay of Subbuteo. The following subsection outlines the data preprocessing procedures developed in a Jupyter notebook using Python.

1. **Video Import and Initial Analysis:** The initial step of data preprocessing was to import the data. In this process the VideoCapture function from the cv2 library is used import the video recording and read its attributes. These attributes include dimensions, frame count, and supplementary metadata, which contribute to the initial analysis.

2. **Resolution Reduction for Enhanced Efficiency:** A resolution reduction strategy is done to reduce the computational complexity since the algorithms have shown not to lose much accuracy after this reduction. The original resolution of 4096x2160 is reduced by half, rendering a refined resolution of 2048x1080, dividing overall the number of pixels by 4. This downscaling has also the nice side effect of slightly removing the noise in the frames.

3. **Frame Extraction and Conversion:** To facilitate the tuning phase and generalization without keeping all the frames only some n frames are extracted, keeping some gap between one frame and another to avoid similar frames. Next these frames were transformed into NumPy arrays, a versatile data structure to facilitate subsequent operations. The final format is a three dimensional array of 2048x1080xN having N frames.

4. **Example Frame Visualization:** Figure 2.1 provides a visual representation of a sample frame from the processed data collection.

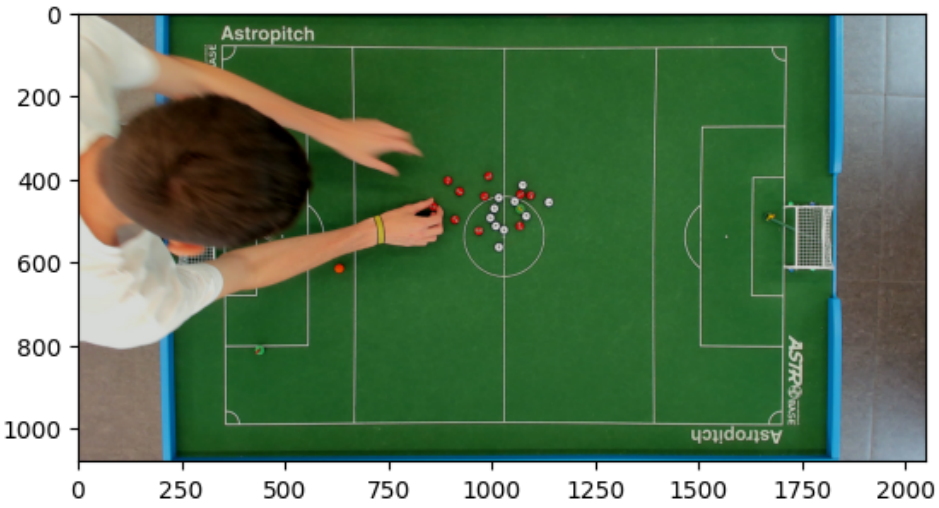

Figure 2.1: Example frame illustrating game elements.

2.0.3 Metrics

Evaluating the performance of the models in object detection is fundamental. As multiple methods are explored, having standardized, quantitative metrics is necessary for consistent evaluation and comparison.
The main metrics that will be used to evaluate the models are the following:

- **precision:** Measures the accuracy of the objects detected by the model, specifically it quantifies the ratio of correct detections to the total number of detections. Among the detections made by the model, how many were actually correct.
- **recall:** It represents the sensitivity of the model to the actual objects in the ground truth, in essence, of the number of actual objects actually present, how many did the model succesfully find.
- **map 0.5:** mAP stands for mean Average Precision. mAP is a common used metric to evaluate the performance of object detection models. The "0.5" refers to the Intersection over Union (IoU) threshold, Meaning that detections are considered a positive match if they have an IoU of 0.5 or more with the ground truth. This calculation is done also for the other metrics but its usually specified for mAP. This metrics its basically the area under the precision-recall vs confidence curves, since the confidence is needed this metric will be computed only when this information is present.

In summary, while both precision and recall offer invaluable insights into specific facets of a model's performance, they can be misleading if analyzed individually. For instance, a model with a precision of 1 might have made a solitary correct prediction, missing numerous other objects present. Therefore the value of mAP at an IoU of 0.5, as it considers both precision

and recall, offer a more general view of model performance. Its also important to consider that the various metrics importance is significantly correlated to the context, for example in a healtcare system, recall can be much more important than precision, to avoid false negative as much as possible despite having low precision.

Chapter 3

Basic detection algorithms

For the detection of the elements in the subbuteo gameplay there are an infinite number of possible approaches, from simple ones to the most complex. In this project the first experiments has been done using very simple methods and in this section its explained how they work.

3.0.1 Corners detection

The first step involves the detection of the four corners of the playing field to have a reference system position. Given that the field corners have distinctive quadrant shape, this feature becomes crucial in the detection methodologies.

Two distinct approaches were developed as preliminary corner detection methods.

3.0.1.1 Area ratio method

This method leverage on the fact that a the area of a quadrant and of the bounding box around it have a ratio of approximately 0,7.

The initial step is to create a white color mask to isolate the field's white sides (see figure 3.1). To address some gaps or distruptions in the lines caused by factors such as imperfect line or overlaying objects, a "closing" operation is executed with the method morphologyEx from cv2 ([Ope]) , using a (4,4) kernel parameter this procedure fill gaps smaller than 4x4 pixels in the frame, the following figures represent the example frame before 3.1 and after 3.2 the morphology process.

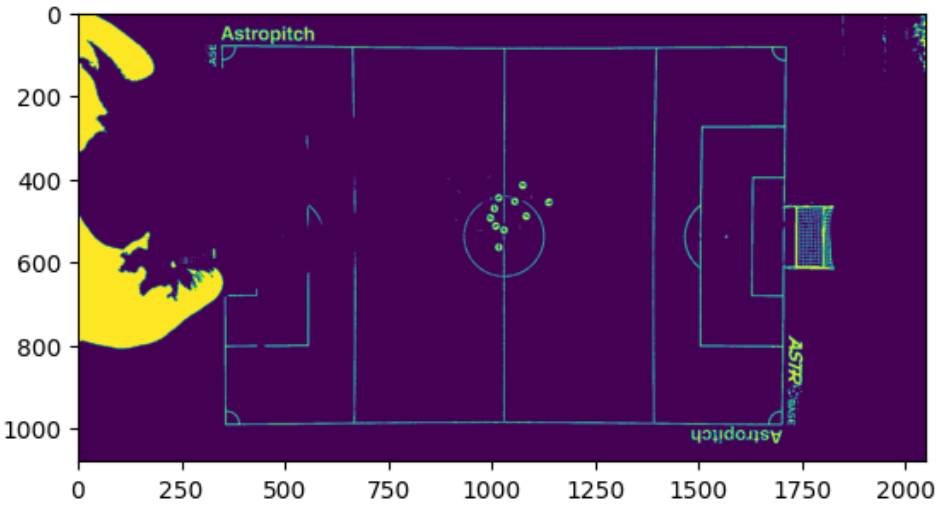

Figure 3.1: White color mask

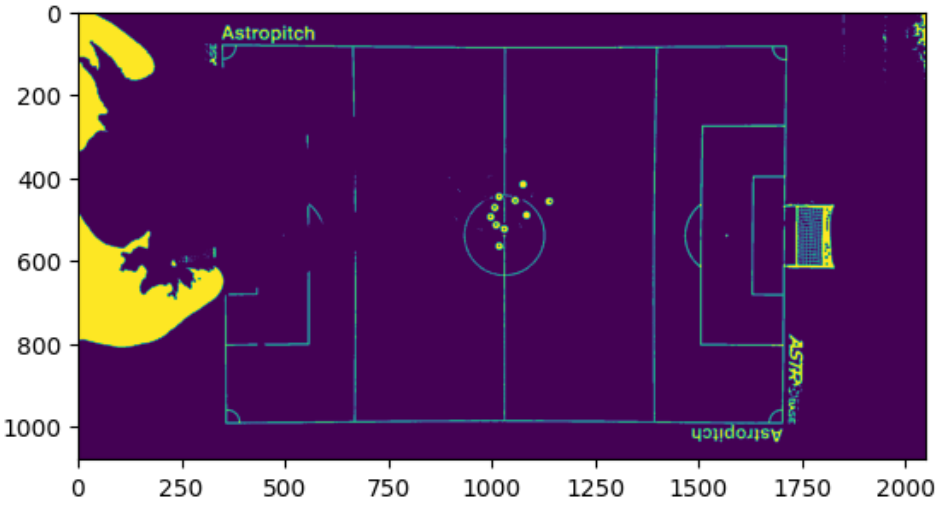

Figure 3.2: Filled white color mask

Next contours are extracted using the method "findContours" from "cv2", getting an array containing the points connecting the sides for each contour. All the contours for the frame example are visualized in the figure 3.3

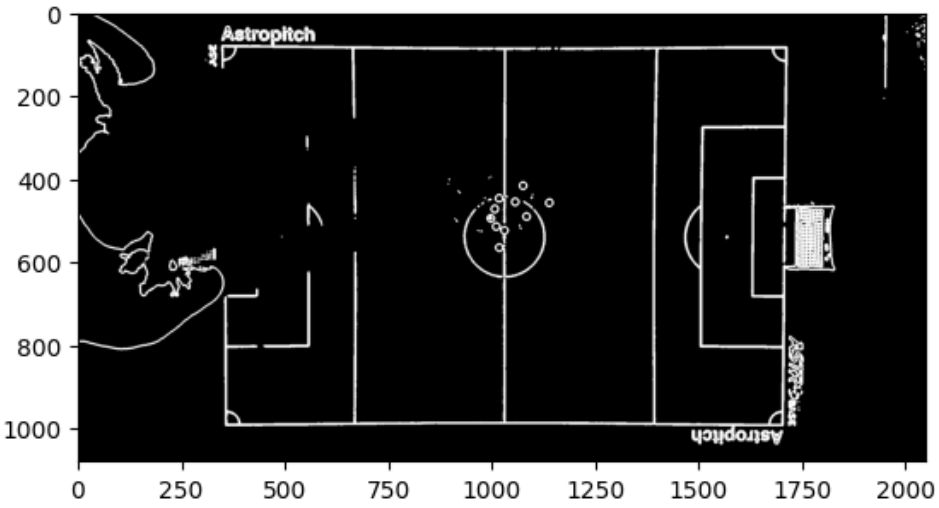

Figure 3.3: Contours

Now calculations of both quadrant and bounding box areas are performed for each contour after removing the ones with non square bounding box and that fall inside a determined area range. Proximity to 0.7 is computed and contours below a certain threshold are classified as corner. While effective, this method may exhibit sensitivity to shapes with comparable box area ratios, leading to potential false positives.

3.0.1.2 Points proportion method

This method is based on the distances distribution from the contour corner point and all the others points.

Diverging slightly, it shares the same preprocessing steps as the previous method then for each contour and each frame corner the contour nearest point to the frame corner is considered as quadrant corner, after this the distance between it and the other points in the contour are computed. Obtaining the distances distribution using fifteen bins reflecting the number of points within each bin range. This approach uses the unique distribution of points in a quadrant. Since the majority of these points (>60) tend to fall in the last bin, this technique allows to identify the quadrant by looking at the last bin proportion.

An example frame with the predicted corners annotation is showed in figure 3.4.

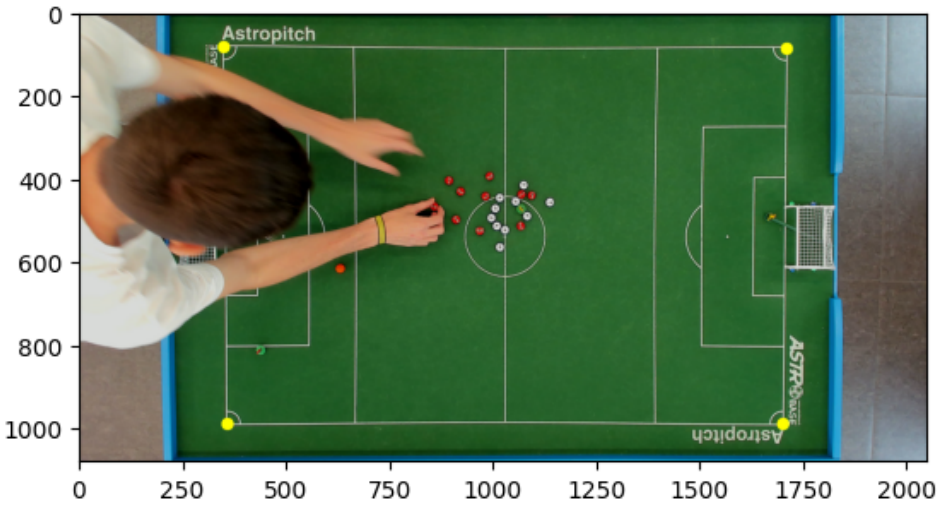

Figure 3.4: Frame with corners annotation

3.0.1.3 Limitations

Both methods are subject to their limitations. The area ratio method, while robust against field rotation with respect to the frame, may fail in detecting only quadrant shapes due to the potential for similar ratios among various shapes, resulting in potential false positives. Conversely, the points proportion method demonstrates sensitivity to field rotation, necessitating perfect parallel alignment with the frame. Nonetheless, once this alignment criterion is fulfilled, the method proves resilient in detecting only quadrant-shaped elements.

3.0.2 Pieces detection

The process of detecting pieces in the field involved the development of different basic methodologies, those methodologies are the same for both team colors with little changes in the parameters tuning.

3.0.2.1 Circularity method

The first approach involves creating a mask corresponding to the specific team color, then with cv2.morphologzEx pixel holes in the mask are filled. With cv2.connectedComponentsWithStats individual groups of pixels are identified.

Those components are then filtered by an area threshold, starting eliminating most of non-pieces elements. Finally, circularity check is executing, computing this value using the contour of each connected component, discarding the remaining non-pieces elements.

The circularity formula is as follow: $4 * pi * (area/(perimeter^2)$ where a maximum value of

1 indicates a perfect circle. However, this method presented some limitations, proving to be less reliable due to false detections of extraneous elements, also, it sometime fail to detect pieces placed over the lines (for the white team) perceiving them as all connected.

3.0.2.2 Area threshold with opening morphology

This alternative strategy is more simple and effective. It start with a sequence of morpgological operations after generating the mask, by using a combination of "CLOSE" and "OPEN" morphology the small pixel holes are filled and the noise elements such as the white lines are removed respectively. Next is just enough to filter the elements by a determined area range and the remaining elements are considered as pieces.

3.0.2.3 limitations

During the implementation of those methods more limitations and challenges than expected has been encountered. The most difficult part was the parameter tuning, in the research of finding the equilibrium between detecting all the pieces and excluding all other confounding elements emerged in an unavoidable trade-off necessity.

The parameter calibration encompasses color, area, shape and other threshold parameters, and the combined interactions between these parameters presented a very difficult puzzle, making it an arduous task to find the tuning for the perfect model. This because the modification to one parameters necessitate adjustments to related parameters, such as the morphological kernels and the area thresholds. A useful thing that hasn't been done for time reasons is to perform tuning in an automatic way, for example with Gridsearch, this would involve also the development of methods to evaluate the predictions to know if the parameters are well tuned.

One specific issue was the division of single red pieces into two elements, caused by the small human statue present whithin each piece that is not red, causing a gap in the center. While adjusting the morphological close parameters could address this problem, it would potentially lead to instead generate single clusters for groups of touching pieces.

These limitations shows how has been difficult to achieve a perfect and precise model for pieces detection. While the current methodologies encounter challenges that may not yield a parameter combination resulting in a perfect model, deep learning techniques can bring more promising results.

3.0.3 Ball detection

The detection of this entity was simpler since it has a shape similar to the other pieces. It was possible to use the same functions as them only modifying the size and color parameters.

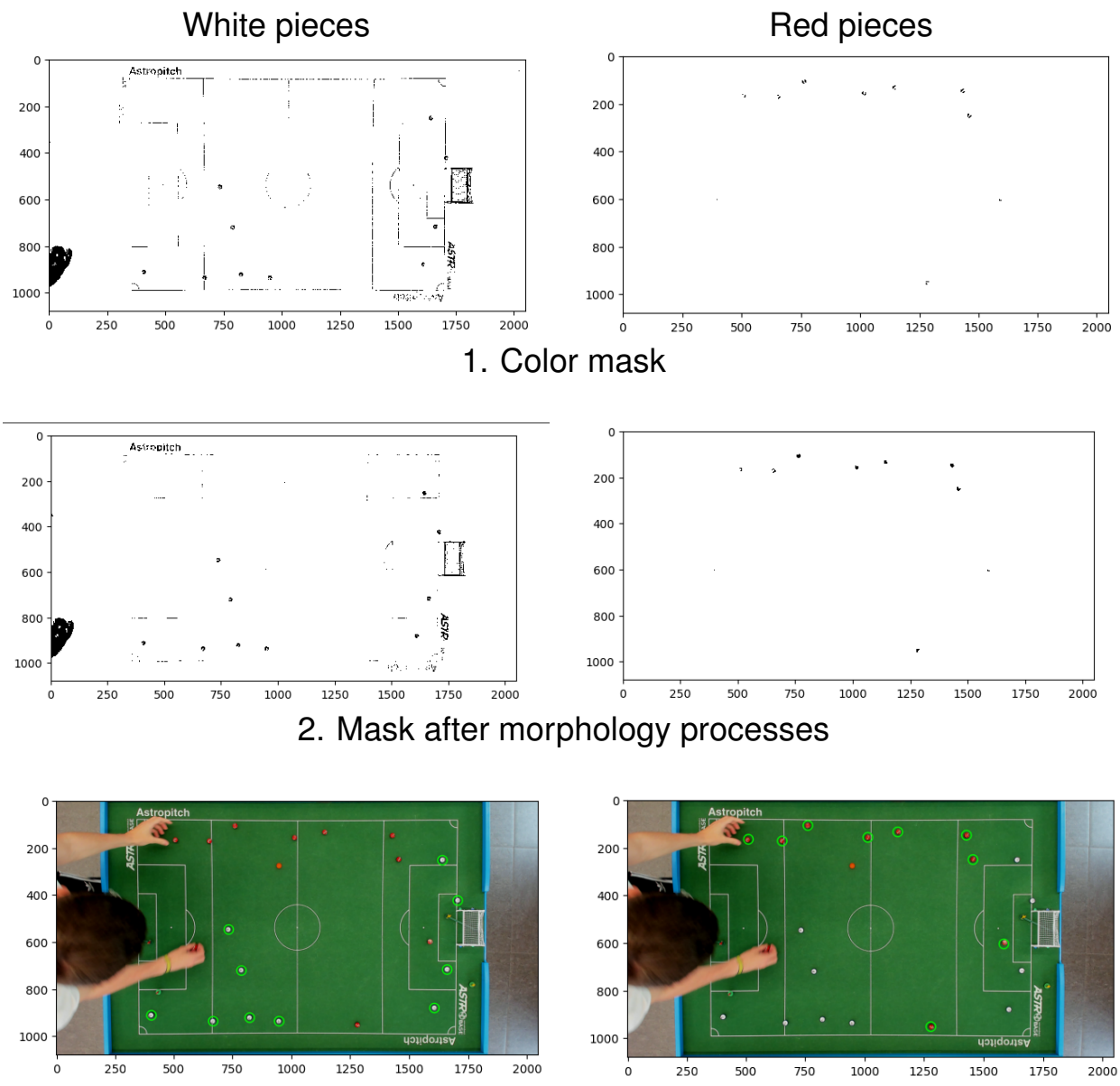

Table 3.1: Processes of white and red pieces detection with area treshold with opening morphology method

Althought this remained a challenge mainly for the similarity between ball and red pieces.

3.1 Results

In this section, the methods will be tested on images that has not been used to tune the parameters.

3.1.1 Inference

Test images are passed throught the function of this methods to obtain the predictions that can be then annotated in the original image and visualized in figures 3.5 and 3.6:

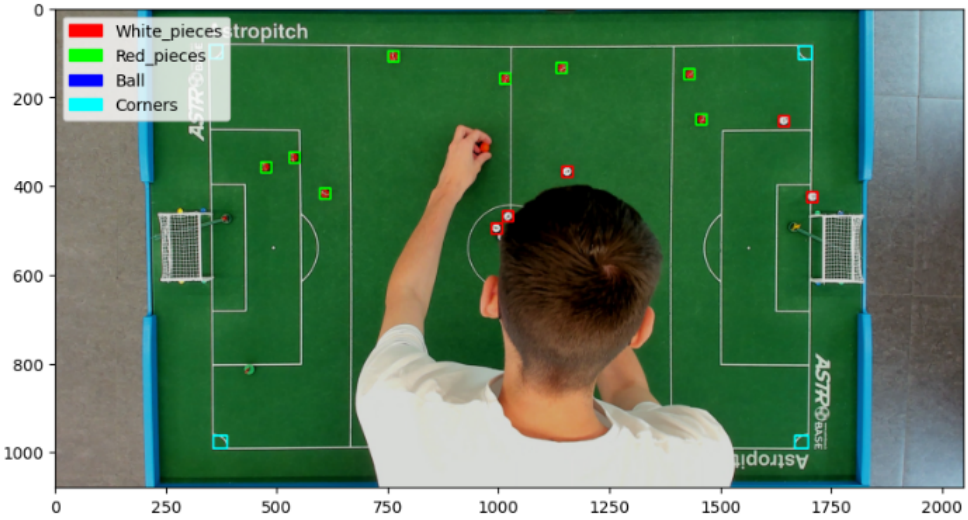

Figure 3.5: Predictions on test image

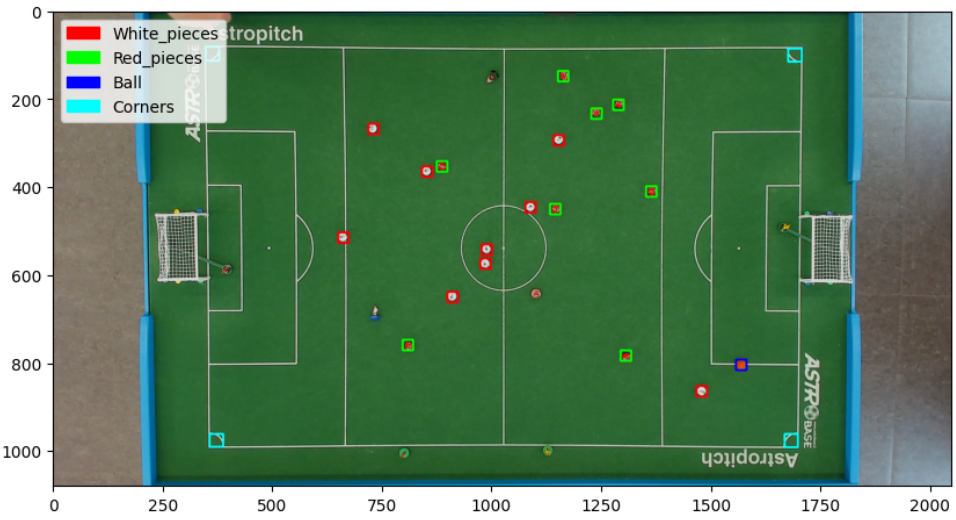

Figure 3.6: Predictions on test image 2

3.1.2 Quantitative evaluation

With functions manually implemented to compute precision and recall (but no mAP since confidence is not avaiable) a table showing the scores for each class and for all togheter is computed:

Results on test set				
Class	**Images**	**Instances**	**Precision**	**Recall**
All	8	163	0.94	0.82
White piece	8	62	0.98	0.87
Red piece	8	61	0.82	0.70
Ball	8	8	0.75	0.75
Corner	8	32	1	0.96

The values on this table confirm the difficulties exposed before, for example the ball have low precision and recall, this is due to the high similarity of the ball and red pieces color, also the red piece have a low recall meaning that many of the red pieces are missed, this due to the limitations explained before.

Chapter 4

Deep Learning

After many trials and experiments using basic techniques the project progressed to more advanced and promising methods. Deep learning technologies are widely used in computer vision and almost always outperform any other method (revision made on [MB22] paper). This section details the deep learning technologies developed and the results achieved.

4.1 Methodologies

Within computer vision, the task of object detection is addressed using several techniques. The following provides an overview of the most popular algorithms (ref: [Gan18] paper):

- **Sliding Window Approach**:
  - **Concept**: The model scans the image with different window sizes at various positions and scales to identify the presence of an object.
  - **Drawbacks**: Highly computationally intensive as it evaluates multiple overlapping patches of an image independently.
- **R-CNN Family (R-CNN, Fast R-CNN, Faster R-CNN)**:
  - **Concept**: Detects and classifies regions of interest in an image.
    - **R-CNN**: Uses an external method, such as Selective Search, to propose regions. Each region is processed by a Convolutional Neural Network, and then an Support Vector Machine (SVM) classifies it.
    - **Fast R-CNN**: Processes the entire image once through a CNN and maps region proposals onto the resulting feature map. Introduced RoI (Region of Interest) pooling and utilizes Softmax for classification.
    - **Faster R-CNN**: Incorporates a (RPN (Region Proposal Network)) directly, making region proposal an integrated and more efficient step.

  - **Evolutionary Advantage**: The progression from R-CNN to Faster R-CNN emphasizes the improvements in the computational efficiency and a more integrated approach, minimizing the reliance on external processes.
  - **Drawbacks**: Despite improvements, the two-step nature (region proposal and then classification) can still challenge real-time processing.

- **YOLO (You Only Look Once)**:
  - **Concept**: Divides the image into a grid. Each grid cell predicts a fixed number of bounding boxes along with class probabilities, therefore handling both bounding box prediction and classification in a single network pass.
  - **Advantage**: Significantly faster than the R-CNN family and the sliding window approach due to its one-shot detection, making it suitable for real-time applications.
  - **Drawbacks**: Earlier versions might have challenges with small or overlapping objects, although this has been addressed in subsequent versions, moreover sometimes R-CNN can still be more accurate.

In the context of detecting Subbuteo game pieces, processing speed is critical, especially when considering future enhancements such as live monitoring. Consequently, this project will employ the YOLO algorithm due to its efficiency and suitability for real-time applications. Development will be based on the official YOLOv5 code [YOLc] and documentation [YOLb]. Further insights were derived from the foundational YOLO paper [JR16].

4.2 YOLO Model

Unlike R-CNNs, YOLO is a One-stage detector and have the following structure (ref: [YOLa]):

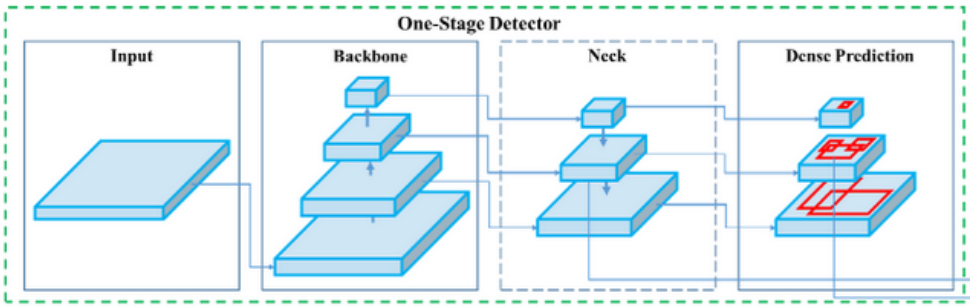

Figure 4.1: YOLO structure

1. **Input:** The image is fed into the network

2. **Backbone:** Deep CNN extract feature from the input image, usually consists of layers that gradually reduce the spatial dimension while increasing the depth (number of channels).
3. **Neck:** Those are additional layers that refine the feature maps obtained from the backbone. This step is crucial for multi-scale object detection.
4. **Dense Prediction:** The final layers generate prediction for object classes and bounding boxes coordinates.

After obtaining the dense predictions, NMS (Non-Maximum Suppression) is applied. This post-processing step retains the highest-confidence bounding box for an object and suppresses overlapping boxes. It operates by sorting predictions by confidence, selecting the top-scoring box, and eliminating boxes with significant overlap (e.g., IoU > 0.5).

Next version of the model had be choose, since there were different sizes of YOLO structure available:

- YOLOv5n $\approx$ 0.7 million parameters
- YOLOv5s: $\approx$ 7 million parameters
- YOLOv5m: $\approx$ 21 million parameters
- YOLOv5l: $\approx$ 47 million parameters
- YOLOv5x: $\approx$ 87 million parameters

In this case, YOLO5s appears to be the optimal choice. Given that the elements to detect are simple (e.g., red or white round pieces), and considering the images have a high resolution of 2048x1024 (subsequently reduced), the nano version would be inadequate. However, to ensure YOLO5s was the most suitable choice, we also tested other versions.
Additionally, we chose to utilize the pre-trained model, which means the model has already been trained on various objects, enhancing edge and general object detection capabilities. Given the limited data availability for this project, the pre-trained version is deemed preferable.

4.3 Custom dataset

To train a YOLO model on the Subbuteo data it was necessary to create a proper dataset. The images were already obtained, as explained in the Data section, by sampling frames from the subbuteo video. It's important to mention that the frame's resolution has been adjusted to 1280x1280, as it was found that reducing the resolution maintained comparable results while enhancing the model's speed.

4.3.1 Data annotation

Another essential information is necessary for training the model: the labels. In this case the labels are bounding boxes around the elements that we want to detect, the pieces the ball and the corners.

In the YOLOv5 manual it was suggested the use of roboflow for drawning the bounding box, but the software LabelImg (code: [lab]) was used instead for simplicity and previous experience with this software. Within this software labels are easily draw in the images, in the figure 4.2 is showed how the labelling process looks.

Figure 4.2: LabelImg software

In the end, approximately 40 images are annotated.

4.3.2 Data augmentation

YOLOv5 library already provided useful data augmentation techniques easly customizable in the file hyper.scrath-low.yaml present in the yolo directory. List of methods:

- **hsv_h**: 0.005, image HSV-Hue augmentation (fraction)
- **hsv_s**: 0.4, image HSV-Saturation augmentation (fraction)
- **hsv_v**: 0.3, image HSV-Value augmentation (fraction)
- **degrees**: 40, image rotation (+/- deg)
- **translate**: 0.2, image translation (+/- fraction)
- **scale**: 0.2, image scale (+/- gain)

- **shear**: 0.0, image shear (+/- deg)
- **perspective**: 0.0002, image perspective (+/- fraction)
- **flipud**: 0.4, image flip up-down (probability)
- **fliplr**: 0.5, image flip left-right (probability)
- **mosaic**: 0.0, image mosaic (probability)
- **mixup**: 0.0, image mixup (probability)
- **copy paste**: 0.0, segment copy-paste (probability)

Among the various augmentation techniques, some were especially effective, such as HSV adjustments, image rotation, translation, scaling, perspective and flipping (ref: [aug]).Those augmentation techniques give a simulation of possible variations in the real word, for example HSV adjustments simulate different lights conditions, perspective simulate angled cam view and so on.

Given the limited dataset, we opted to duplicate each sample. This approach is beneficial because it's improbable for two images to obtain identical augmentations, leading to identical samples. This is due to the numerous techniques of augmentation combinations possible. By duplicating samples, we essentially double the unique data in the training set.

The figure 4.3 displays examples of augmented images from the batch:

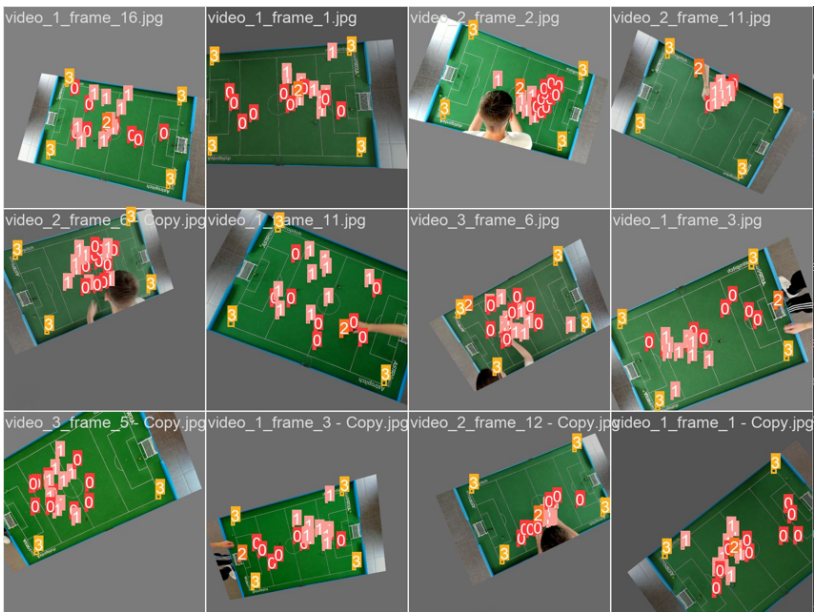

Figure 4.3: batch with augmented images

4.4 Hyperarameters

In the hyperparameters file there were also a set of tunable Hyperparameters, such as learning rates, regularization techniques, biases and various threshold. Those parameters are complex to tune due to the few amount of data and relatively long training time, altought an automatic hyperparameter tuning could be implemented to find the best combination the provided standard values were already giving good results.
The tuned parameters are the following:

- **learning rates**: both initial and final learning rates are reduced (from 0.01 to 0.005) since the training time to reach convergence was not too long this allowed a smoother approach to the convergence
- **anchor sizes**: anchors sizes are modified based on the mean size of the target elements, personalizing the model to better suit this specific dataset.

An improvement on this part would be to implement a program that automatically train the model multiple times with different combination of parameters to find the best one, this method is called *hyperparameter tuning*.

4.5 Training

To train the model with our custom data it was necessary to run the train.py file provided, this could be done by running the command and setting some parameters 4.1:

Listing 4.1: Command to train yolov5s model

```
python train.py  --epochs 100 --data D:\SUPSI\git\subbuteo_monitor
    \data\yolo_data\training_data.yaml --weights yolov5s.pt --batch
     16 --device 0
```

- **–epochs:** set number of epochs
- **–data:** set yaml file containing training and validation data paths.
- **–img:** Reduce resolution of images (not used.)
- **–weights:** Specify which pre-trained yolo model weights use.
- **–device:** Specify which device to use on the process (0=gpu).

A comet ML API key WAS previously been set to the enviniorment to save and monitor the training log.

4.5.1 Training Time and Resources

Training duration for computer vision tasks can be quite long, in this case a relatively high resolution of the images is necessary because of small dimension of the pieces. Despite this. the chosen small version of YOLO had much less parameters than the bigger versions, reducing the complexity and thus the training and processing time. The training process also has been done usign both CPU and GPU, specifically Intel(R) Core (TM) i7-10700f CPU 2.90GHz and GeForce RTX 2060 respectively, therefore the training time became much shorter, $\approx$ 2h for 500 epochs.

4.5.2 Losses

Other than the metrics some losses are being computed in the training process, this values not only offer a visual insight on how the model is learning but are also being used from the model to update its weights in the backpropagation steps. The computed losses during training are the following:

- **box_loss**: This refers to the bounding box regression loss. It measures how far off the predicted bounding box coordinates are from the correct one, this loss is used to update the weights of the model during the training.
- **cls_loss**: This is the classification loss. If you are detecting multiple classes of objects, this loss measures how well the model classifies which object is in a given bounding box.
- **obj_loss**: This loss relates to objectiveness. It helps the model decide whether there is an object in a given bounding box or not. While losses are computed on both training and validation set metrics are only computed on the validation set.

4.5.3 Learning curves

During training, losses are computed on both training and validation but metrics are only computed on the validation set.
To analyze the trend of those values during the epochs, lineplots are plotted, this visually give an idea of how the model is learning and for example to ensure that the model is not overfitting.

The plot 4.5 (with validation and train loss) shows that the model is not overfitting since the validation loss decrease in the same way as the training set, the other plots shows that the metrics improve until epoch $\approx$ 300, next the values are stable and therefore more epochs are not necessary. Since those plots give an overview of the scores obtained on the validation set the best model (configuration) can be chose and tested in the next steps. After

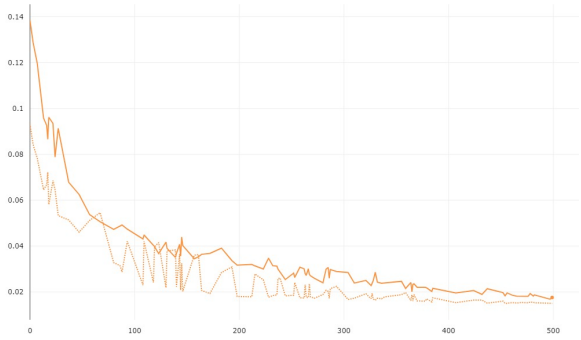

Figure 4.4: validation (dashed line) and train (solid line) box_loss loss

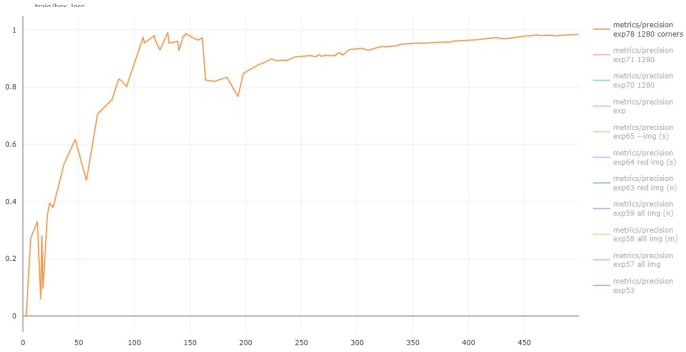

Figure 4.5: Precision

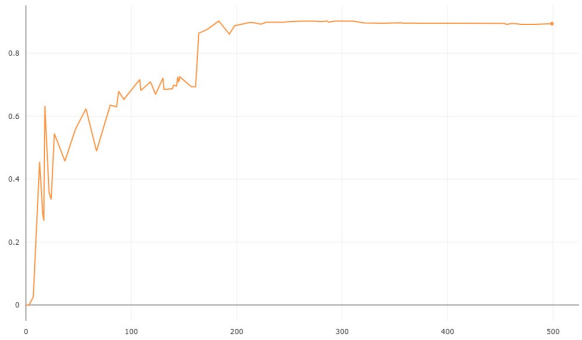

Figure 4.6: Recall

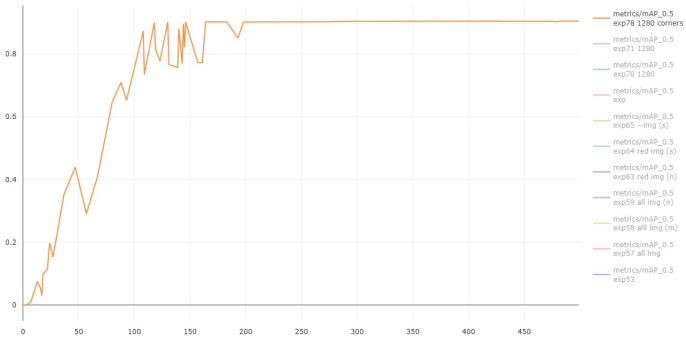

Figure 4.7: mAP 0.5

analyzing the curves for various model setting (such as image resolution, model version and some hyperparameters) the model yolov5s with duplicated images and 1280x1280 resolution (showed in the plots above) seemed to be the best choice The next plot at figure 4.8 compare the small and nano version of the YOLO model on the recall score (the precision was similar), and its visible that the nano version stops improving and converges at lower recall than the small version:

4.6 Results

In this section quantitative and qualitative results on test images that has never been used for the model in the training process neither for tuning are shown.

4.6.1 Inference

To obtain and visualize the predictions on the test images the model can be load and used with torch in python, or the file detect.py provided in the YOLOV5 directory can be directly used to make predictions with the following command 4.3:

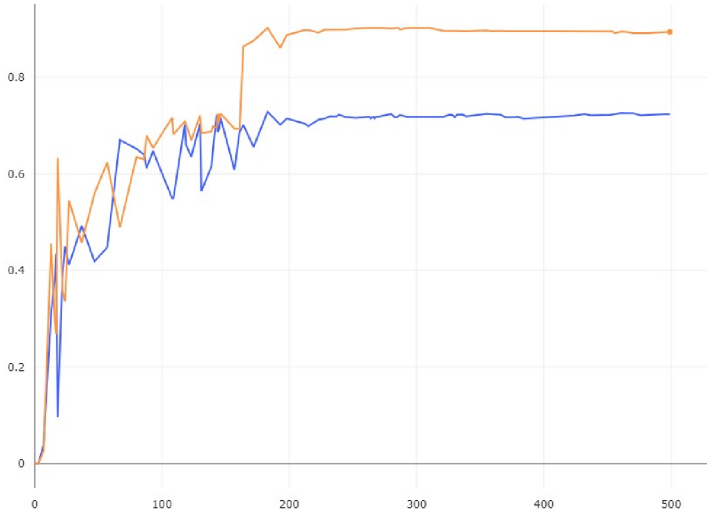

Figure 4.8: Recall vs epoch, small (orange line) and nano (blue line) yolo versions compared

Listing 4.2: command to run detect.py file on test images

```
python detect.py --imgsz 1280 --source D:\SUPSI\git\
    subbuteo_monitor\data\yolo_data_w_corners\test\images --weights
     D:\SUPSI\git\yolov5\runs\train\exp78\weights\best.pt
```

The script saves the frames with the predicted annotation and its confidence as showed in the figure 4.9 and 4.10:

Figure 4.9: predictions on test frame

Figure 4.10: predictions on test frame

The average speed for the process on a single frame is the following: 1.4ms **pre-process**, 20.9ms **inference**, 2.4ms **NMS** per image at shape (1, 3, 1280, 1280)

4.6.2 Evaluation

The best model is finally tested on a test set of images not used in the training process, neither for validation, since this set of images should not influence the choice of model and hyperparamers. To test the model the file "val.py" has been used on test images, to do it the following command 4.3 has been runned:

Listing 4.3: command to run val.py file on test images

```
python val.py --data D:\SUPSI\git\subbuteo_monitor\data\yolo_data\
    testing_data.yaml --weights D:\SUPSI\git\yolov5\runs\train\
    exp58\weights\best.pt
```

4.6.2.1 Scores table

The model is then evaluated using the same metrics explained before (except for the losses), in the table below are present the quantitative results.

Results on test set					
Class	**Images**	**Instances**	**Precision**	**Recall**	**mAP 0.5**
All	8	163	0.99	0.92	0.98
White piece	8	62	1	0.95	0.97
Red piece	8	61	0.98	1	0.99
Ball	8	8	1	0.78	0.98
Corner	8	32	0.98	0.96	0.96

From this table can be deduced that the model is performed very well, all the precisions are high, as well as the recalls, with exception only for the ball that have a lower recall, meaning that the model may have some difficulties in detecting this element, this can be explained from the very low number of instances of this element in the training set.

4.6.2.2 Confusion matrix

Also a confusion matrix is plotted to check for specific issues:

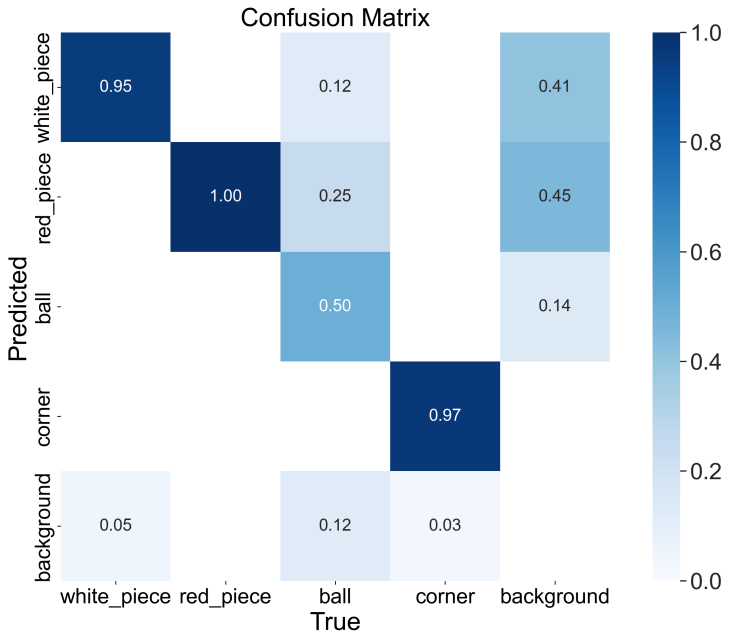

Figure 4.11: confusion matrix

From the Confusion Matrix (CM) it can be saw that there is a relevant number of times where the model mistakenly detect the ball as red piece, this because of the similarity between the two elements.

4.6.2.3 Precision-confidence curve

A Precision-cofidence curve is plotted to analyze how the confidence is ensuring a correct prediction.

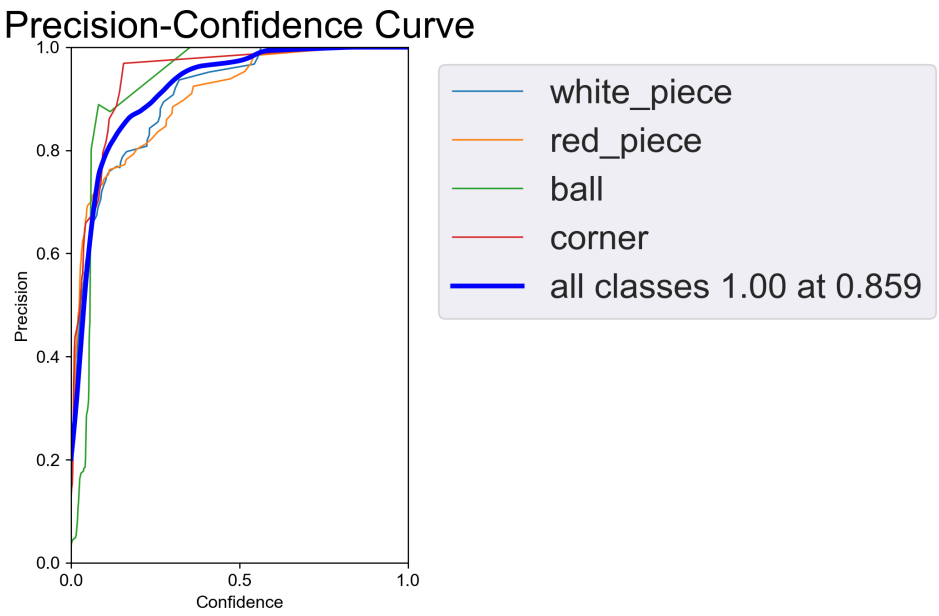

Figure 4.12: Prediction-confidence curve

The plot shows that after a confidence threshold of ≈ 0.6 the prediction is always correct, in the test set.

4.6.2.4 Recall-confidence curve

But it also important to visualize the recall confidence curve, since if we set a too high threshold confidence acceptance, ensuring high precision, many will be left undetected:

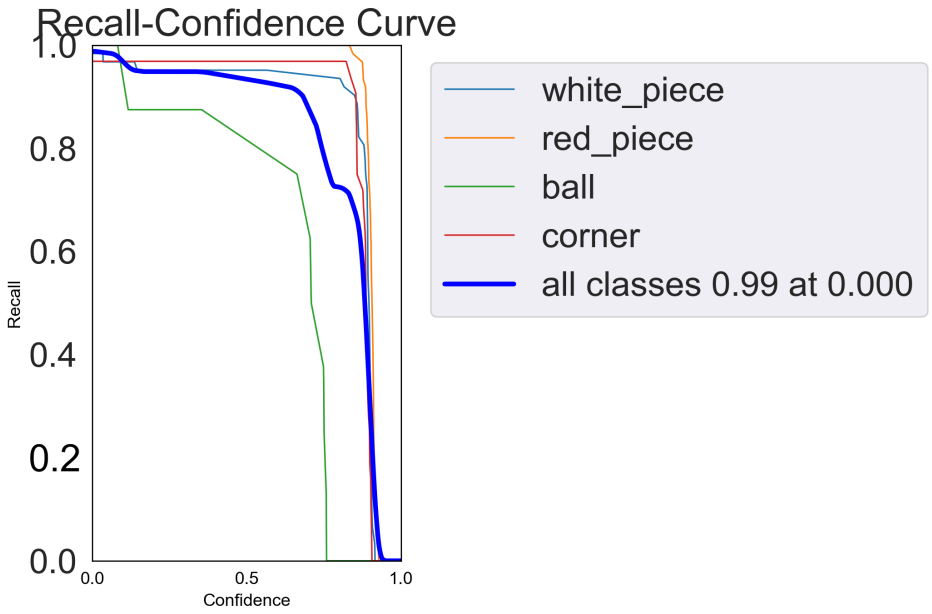

Figure 4.13: Recall-confidence curve

From the graph, it's evident that no detections occur beyond a confidence level of approximately 0.9. This indicates that while a model might have high precision after reaching a certain confidence threshold, it doesn't necessarily translate to overall good performance of the model. Thus, it's crucial to select a threshold that balances both metrics optimally. Furthermore, the green line, which represents the ball, highlights that the model consistently predicts this entity with lower confidence. This lower confidence is likely attributed to the limited instances of the ball in the training dataset and its similarity to the red pieces.

4.7 Improvements

A series of improvements could be done in this part of deep learning. The complexity of this model emerges in numerous possible combination of modalities, combining model versions, parameters and hyperparameters, therefore it would be quite challenging and resource-intensive to enhance the model for only a marginal increase in performance. This was not worth it since there was a bigger issue in the foundation, the data. Data was extremely limited since in the training set there were more or less 30 images, confronting with the suggestion of the YOLO developers of using at least 1000 images the gap is enormous.

So the first improvement would be to increase the number of frames with many different lighting condition and positions of the elements. Only after that a better hyperparameter tuning can be suitable. As explained in the hyperparameters section an automatic hyperparameter

tuning method such as random search can lead to better results.

4.8 Comparison

We compared the scores of basic methods with the deep learning (YOLO) approach. A table was made showing the differences in precision and recall, specifically, the scores from the basic methods were subtracted from the YOLO values to gauge its superiority in the following table

Difference between tables		
Class	**Precision**	**Recall**
All	+0.05	+0.10
White piece	+0.02	+0.08
Red piece	+0.16	+0.30
Ball	+0.25	+0.03
Corner	-0.02	0

The table reveals that nearly all values are positive, indicating YOLO's better performance. It was particularly more accurate in detecting red pieces (higher recall) and predicting balls correctly (higher precision).

4.9 Example Usage

4.9.1 Perspective Transformation

Perspective transformation, often referred to as homography or projective transformation, is a fundamental technique in the realm of computer vision and image processing. Its primary function is to transform a given image into a different viewpoint, allowing the viewer to see objects and scenes from another perspective, akin to changing one's viewpoint in the real world.

1. **Principle of Operation**: At the heart of this transformation is the principle that parallel lines in the real world will still appear parallel in the perspective-transformed image. This technique leverages this invariant to ensure that rectangular objects in the real world (like our Subbuteo field) are represented as rectangles in the digital rendition, even if the original camera view was at an angle.

2. **Determining Transformation Matrix**: Four points in the source image and their corresponding points in the destination image are used to determine the transformation matrix. In the subbuteo context, the source image are the detected corners in the field and the destination image their normalized position in the frame with some margin in

the border. By identifying these points the transformation matrix is computed using OpenCV's `getPerspectiveTransform` method.

3. **Applying the Transformation**: Once the matrix is determined, it can be applied to the entire image, thus the predicted bounding boxes coordinates, converting an eventual skewed or angled view of the field into a perfect top-down perspective.

4. **Benefits for Subbuteo Gameplay Analysis**: Implementing perspective transformation makes subsequent operations, like field reconstruction, goal detection and player movement tracking, much simpler and more accurate, this because it simplifies the spatial computations. By working in a normalized space, operations like determining if a ball is in the goal or a piece is offside become straightforward distance checks. In essence it ensures the digital view aligns with the actual gameplay dimensions, providing a more accurate and consistent basis for further analysis and event detection.

4.9.2 Visual reconstruction

Once the perspective transformation is complete, we can derive the coordinates for various zones of the field using standard proportions. These zones, in addition to the pieces, are then plotted onto a blank frame for clear visualization, the following are the steps involved:

1. **Field Boundary Construction**: Using the detected and transformed corners, the rectangular boundaries of the field are established by connecting them. If one§ or more corners are missing, positions from previous frames can be utilized to approximate their locations.

2. **Goal Position Determination**: Taking into consideration the proportion of the goal to the field sides, approximately $1 : 6.7$, the exact position of the goal net is determined.

3. **Vertical Lines determination**: The field is divided by several vertical lines which demarcate key zones, such as the "shooting area" and the central midline. By leveraging the known width of the field and standard proportions, the precise coordinates for these divisions can be obtained. Those lines are foundamental for the detection of specific events like offside.

4. **Placement of Game Pieces**: Finally, positions of all game pieces, including the ball, are plotted on the digital representation.

5. **Goal Event Detection**: With the spatial information now available, it becomes straightforward to determine if, in a given frame, the ball lies within the goal boundaries, implying a goal event. This event could be reset (indicating the end of a goal event) once the ball returns to regular play on the field.

After applying the described processes to an example angled game frame, we obtain the normalized positions in a digital representation, as shown in Figure 4.14.

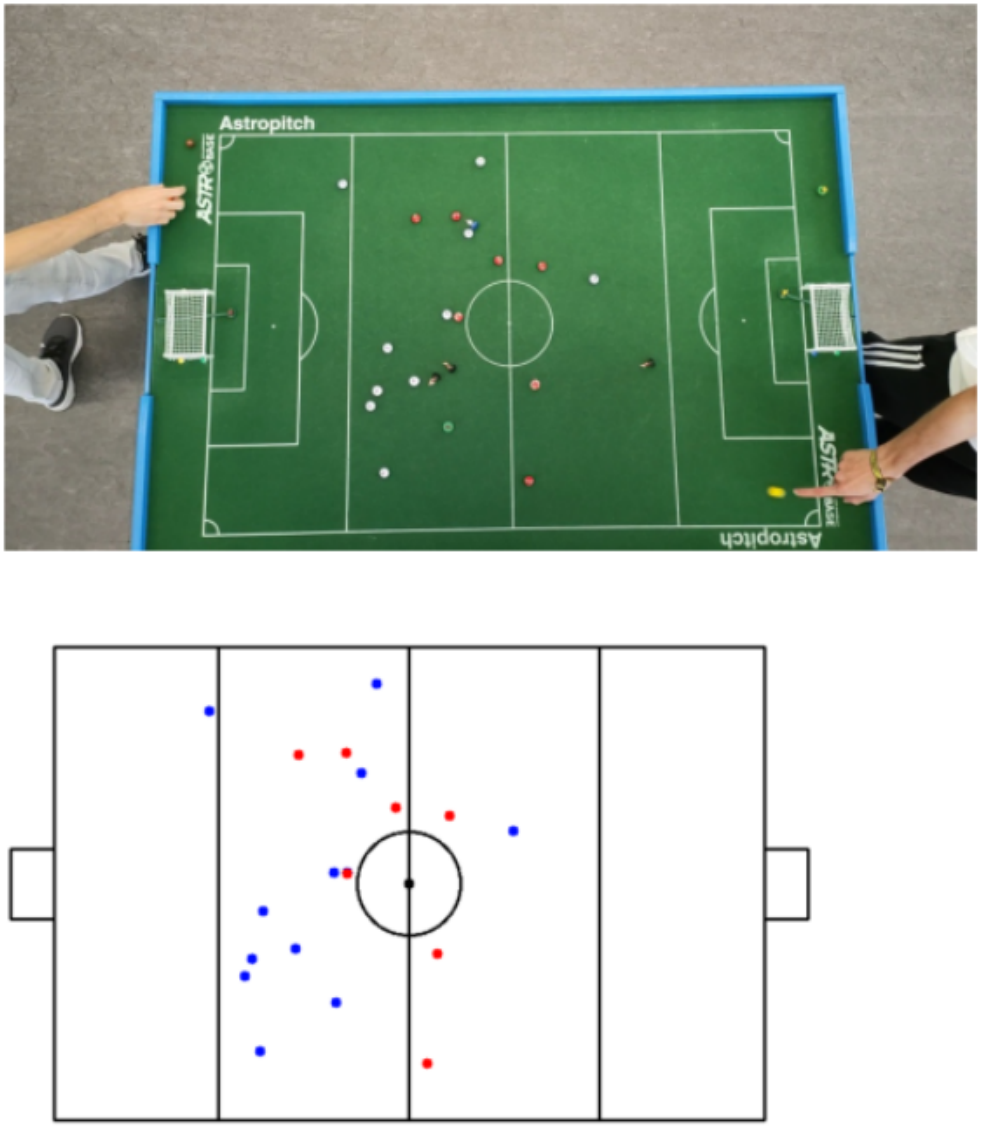

Figure 4.14: Reconstruction of transformed positions

With this foundation base, future work becomes simplified. The potential for real-time goal and other events detection becomes feasible. By converting this process to a live format and maintaining a log of detected goals, we move closer to a comprehensive system that can identify and record live events during gameplay.

Chapter 5

Conclusions

This thesis documentation explained the various steps involed in developing a computer vision system capable of detecting and tracking elements in a Subbuteo game, bringing the traditional table soccer game to the digital form by leveraging basic to state-of-the-art technologies.
The methodologies adopted in this work have resulted effective in recognizing essential in-game entities like player pieces, corners, and the ball. While the detection of certain elements, for example the ball, presented challenges, however, simply subsituting the ball, such as using a yellow ball instead of an orange one, would greatly improve the accuracy of the system in his detection.
In terms of comparative performance, deep learning approach (YOLO), outperformed basic computer vision methods. This was anticipated, given the inherent complexity of the game environment and the robust capabilities of this state-of-the-art DL technology.
Also is important to note that even if very few images (samples) were available it was possible to train the model in a decent way thanks to the augmentation and techniques and variability in the samples.

5.1 Future Improvements

The foundation of this research provides various opportunities for improvement:

- **Increase amount of data:** One of the primary improvements lies in expanding the dataset. A richer and diverse dataset will not only increase the accuracy of the model but also ensure its better generalization.

- **Color Diversity:** By incorporating pieces of various colors in the game setup, the model can be trained to recognize a multiple pieces teams. This augmentation will also give to the player the possibility to choose the teams that prefer.

- **Refined Element Detection:** As part of future enhancements, there's potential to further refine the detection capabilities to include game elements such as "fallen pieces" and "goalkeepers". Introducing these elements into the detection framework would be straightforward, following the established procedures used for other pieces. Once these elements are incorporated, the model can be retrained seamlessly. However, goalkeeper pieces could be harder to detect than other elements due to their small size and their tendency to be positioned behind the goal net. While increasing the data set is a general requirement for improvements, this becomes especially crucial for accurately identifying elements like this one.
- **Gameplay Insights:** An important goal is to utilize the spatial data of the game elements to generate real-time insights, such as detecting goals or offsides. This step lies on the quality of the basis developed in this thesis.
- **Real-time Monitoring:** Enabling the system to operate in real-time, with minimal latency between actual gameplay and system processing, would enrich the player experience. Instantaneous event notifications, for instance, can bring engagement to the players.

In essence, while the goals of this thesis were reached, the horizon of improvement remains vast. The intersection of traditional table games and computer vision technology gives plenty of possibilities, and this present the development of the basis for future development.

5.2 Acknowledgments:

I'd like to express my appreciation to Giacomo Del Rio, for his guidance and support throughout the development of this thesis. His expertise in computer vision was invaluable to my research.
I would also like to thank the IDSIA department for providing the necessary resources and environment for the research, and to anyone who offered feedback or insights along the way. Finally, gratitude goes to my family and friends for their understanding and patience during this period, and to the Subbuteo community for the inspiration.

Bibliography

[aug] Explanation of various image augmentation techniques. URL: https://iq.opengenus.org/data-augmentation/.

[Gan18] Rohith Gandhi. Object detection algorithms. *towardsdatascience*, Jul 2018. URL: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e.

[JR16] Ross Girshick Ali Farhadi Joseph Redmon, Santosh Divvala. You only look once: Unified, real-time object detection. May 2016. URL: https://arxiv.org/abs/1506.02640.

[lab] *LabelImg software for labelling*. URL: https://github.com/HumanSignal/labelImg.

[log] *Logitech BRIO camera manual*. URL: https://www.logitech.com/content/dam/logitech/it_ch/video-collaboration/pdf/brio-datasheet.pdf.

[mak] *Mako-g camera manual*. URL: https://www.alliedvision.com/en/camera-selector/detail/mako/g-234/.

[MB22] Kashifa Kawaakib Hussain Md. Bakhtiar Hasan A.B.M. Ashikur Rahman Md. Hasanul Kabir Mk Basha, Samia Islam. Multiple object tracking in recent times: A literature review. Sep 2022. URL: https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwiT9ezVvomBAxWj2wIHHWu2D-QQFnoECBcQAQ&url=https%3A%2F%2Farxiv.org%2Fpdf%2F2209.04796&usg=AOvVaw13PPwimA39ctiJB_afZbM3&opi=89978449.

[Ope] *Opencv documentation*. URL: https://docs.opencv.org/4.x.

[YOLa] Yolo structure explanation article. URL: https://iq.opengenus.org/yolov5/.

[YOLb] Yolov5 documentation. URL: https://docs.ultralytics.com/yolov5/quickstart_tutorial/.

[YOLc] Yolov5 github code directory. URL: https://github.com/ultralytics/yolov5.

Glossary

Confusion Matrix (CM) A table used to evaluate classification model performance by comparing actual and predicted values. 27

Hyperparameters Parameters in a machine learning model that are set prior to training and are not updated during training. Examples include learning rate, batch size, and number of epochs. 22

NMS (Non-Maximum Suppression) A post-processing step in object detection that retains only the most confident bounding box for an object while suppressing other overlapping boxes. Ensures each detected object is represented by a single, high-confidence bounding box. 19

RoI (Region of Interest) A selected subset of an image or dataset, typically focusing on objects or features of significance. In computer vision, it often refers to areas where potential objects are located. 17

RPN (Region Proposal Network) A network used in object detection to propose candidate object regions directly from feature maps, often integrated into architectures like Faster R-CNN. 17

Softmax A function that converts a vector of raw scores, known as logits, into probabilities for each class in multi-class classification problems. 17

Support Vector Machine (SVM) A machine learning algorithm used for classification and regression, focusing on maximizing the margin between data classes. 17

YOLO (You Only Look Once) An object detection algorithm that processes images in one pass, efficiently detecting objects in real-time. 4