



Pázmány Péter Catholic University

Faculty of Information Technology and Bionics

Movement detection on video recordings of water polo matches

Hubay Csenge

November 11, 2022

TDK thesis

Consultants: Dr. Horváth András, Magyar Bálint and Török Mihály

Acknowledgements

I would like to thank all my consultants, without whom this thesis would not exist. Words cannot express my gratitude to Magyar Bálint, who provided crucial professional help, suggestions throughout my work, and valuable feedback. Furthermore, I would like to express my appreciation to Török Mihály, who coordinated my work patiently. I would also like to thank Dr. Horváth András for his help in the administration of this research. Finally, I would like to thank my family, friends, and colleagues for all the support they provided.

Contents

Acknowledgements	2
1 Abstract	6
2 Introduction	7
2.1 About image processing, in general	7
2.2 Water polo and sport analytics	7
2.3 Antecedents	8
2.4 Goals of this research	8
3 Technological background	10
3.1 Basic concepts	10
3.1.1 Computer vision	10
3.1.2 Segmentation	10
3.1.3 Machine learning	10
3.2 Methods used in used for court segmentation	14
3.2.1 Canny edge detection	15
3.2.2 Hough transform	15
3.2.3 Template matching	15
3.3 Methods used for detecting moving objects	16
3.3.1 Histogram analysis	16
3.3.2 K-means clustering	16
3.3.3 U-net architecture	17
3.3.4 R-CNN architectures	18
3.3.5 Mask R-CNN architecture	19
3.3.6 YOLO algorithm	19
3.4 Methods used for tracking	20
3.4.1 Kálmán filter	21
3.4.2 DeepSORT	21

3.4.3	AlphaPose	22
4	Planning	24
4.1	Comparing traditional methods and machine learning	24
4.2	Choice of segmentation type	24
5	Implementation	26
5.1	Python packages used	26
5.1.1	OpenCV	26
5.1.2	PyTorch	26
5.1.3	SciPy	26
5.2	Other softwares used	27
5.2.1	Linux environment	27
5.2.2	JupyterLab	27
5.2.3	Kdenlive	27
5.2.4	CUDA	27
5.2.5	Docker	27
5.2.6	Conda	28
5.3	Preparing versions of the dataset	28
5.3.1	Extracting important information	28
5.3.2	Dataset labeling	29
5.3.3	Dataloaders	30
5.4	Court segmentation	31
5.4.1	Segmentation using Hough lines	31
5.4.2	Segmentation using template matching	31
5.5	Moving object detection, ball detection	33
5.5.1	Ball detection with traditional methods	33
5.5.2	Ball detection with handmade sequential network	34
5.5.3	Ball detection with U-net	34
5.5.4	Ball detection with YOLOv5	35
5.6	Moving object detection, player detection	35
5.6.1	Player detection with Mask R-CNN and U-net	35
5.6.2	Player detection with AlphaPose	35
5.6.3	Player detection with YOLOv5	36
5.7	Moving object detection, simultaneous detection	37

5.8	Tracking of moving objects	38
5.8.1	Naive approach	38
5.8.2	Application of Kálmán filters, in general	38
5.8.3	Player tracking with PoseFlow	38
5.8.4	Tracking using DeepSORT	38
6	Evaluation of results	40
6.1	Court segmentation	40
6.2	Moving object detection, ball detection	42
6.2.1	Average number of pixels in the testing dataset	42
6.2.2	Results on confusion matrices	43
6.2.3	Applying common evaluation metrics	43
6.2.4	Results by IoU scores	45
6.2.5	YOLOv5 results, an exception in the grading scheme	46
6.3	Moving object detection, player detection	46
6.3.1	Mask R-CNN results	47
6.3.2	YOLOv5 results	48
6.3.3	Comparison of networks	49
6.4	Moving object detection, simultaneous detection	49
6.5	Tracking	50
6.5.1	Ball tracking	50
6.5.2	Player tracking	50
7	Conclusions	53
7.1	Summary	53
7.2	Future plans	54

Chapter 1

Abstract

Image processing algorithms and deep learning-based methods provide great results for problems like movement detection. Later on, these can then be used to generate statistics useful in different fields of life, such as sport analytics.

The problems I intend to solve are related to the analysis of water polo matches. For the study of these, the first thing needed is to identify the key elements of the game. These elements are the court, the ball, and the players. After finding the required elements, the next step is to distinguish the players from each other and track the trajectories of the moving objects.

In this thesis, I would like to identify all these key elements on a single video footage. This sport provides new challenges compared to other ball sports, as the game is played in an aquatic environment, which sometimes covers our targets.

Chapter 2

Introduction

In this research, my goal is to build an analysis pipeline for water polo using various image processing methods.

2.1 About image processing, in general

Nowadays the importance of image processing is increasing rapidly. The use cases also cover a thriving area of topics from diagnostics to the military. There is also a possible application aiding sports teams; this technology is used in a wide variety of sports such as golf, football, and basketball. Amongst these, a possible field of use is water polo, which still seems to be unexplored so far.

Image processing algorithms and deep learning-based methods can provide an accurate solution for movement detection, which is necessary for generating game statistics.

2.2 Water polo and sport analytics

Like many modern ball sports, water polo, too, originates from 19th century Great Britain. In the next few decades, it gained popularity and spread worldwide in parallel with other sports, such as tennis or football.

In 1928, a breakthrough happened in the sport when Béla Komjádi, a Hungarian coach invented the "air pass" technique. Before, the ball was in contact with the water for most of the game's duration. He discovered that by throwing and keeping it out of the water in general, the dynamics of the game can be shifted. This technique and its perfections have earned Hungary dominance in this sport for many of the next decades making it one of our national sports. For Hungarians, water polo has a high historical and ideological value[15].

Sadly, the popularity of water polo has been on the decline for years even in Hungary. As football became the focus of attention in Europe and South America, it seems more profitable to finance sports analytics concerning that instead. This means that there is little to no research being made on water polo.

2.3 Antecedents

Calculating the trajectory of moving objects (also referred to as tracking) is a solved problem for many sports including football, volleyball, and golf. There is a particularly elegant solution made for basketball [19]. This work describes a general solution for the problem; detection and tracking of the players, pose estimation, and projection of their position on the basketball court. This approach might face difficulties for water polo, as the water covers most of the joints of the players.

Some research is already done for tracking and detecting people in aquatic environments. There is one thesis, for example, about the tracking of swimmers. In that study, they do not use machine learning; their implementation is solely based on traditional techniques, such as color correlation and histogram analysis [2]. There are some unexpected uses of pose estimation in aquatic environments; in 2019, one of the main topics of MediaEval conference's Satellite was flood severity estimation based on humans' poses in the images taken from different social media platforms[3]. This challenge combines the difficulties of an aquatic environment, that I am about to face and the possible solutions for the problem of object detection and tracking shown by the basketball pose estimation system mentioned before. Unfortunately, the dataset for this is not available anymore.

There are some papers about the tracking of water polo players, however, they use special equipment during data recording. There is one that uses 360° cameras to detect players and analyze different in-game situations [9]. In that implementation, the ground truth is provided in the form of bounding boxes; an idea that seems useful for my task as well.

2.4 Goals of this research

I aim to build an image processing system that can analyze water polo matches by segmenting and tracking the most important elements of a single match and also to provide know-how for broader use. I aim to solve the following problems:

- **Court segmentation:** this method filters out the background, which is irrelevant from the perspective of the match.

- **Moving object segmentation:** to gain information about the game itself, we need to follow certain objects in the water. These are the ball and the players. By knowing their actual location, it is possible to compute their trajectories.
- **Moving object tracking:** computing the tracks of the players and the ball can give important information about what is happening on the court.

Chapter 3

Technological background

In this section, I would like to give an overview of the technology used in my implementations. This consists of general concepts appearing in my work, specific methods, and algorithms.

3.1 Basic concepts

The concepts that I listed in this section appear frequently in the field of movement detection. It is important to state them to achieve a better comprehension of the subject; the background and also the implementation of the tasks I aim to solve in this paper.

3.1.1 Computer vision

Computer vision can be perceived as an implementation of human sight as computational models. The main goal of this technology is to gain information from images or videos or in some cases even real-time environments.

3.1.2 Segmentation

Segmentation is a subfield of computer vision. It includes a wide variety of techniques, and its main subtypes are semantic and instance segmentation. The first labels each pixel with a given set of classes, while the latter does not only label the pixels but also distinguishes between each of the objects from the same category[11].

3.1.3 Machine learning

There are several different approaches and hence, different definitions of machine learning. They all more or less agree that such algorithms are computational algorithms designed

to mimic human thinking and learning processes. Machine learning requires input data to gain the desired output without any programming needed concerning the procedure in use; it alters its own architecture throughout numerous iterations to fit a given task. The desired output is called ground truth, which can be given or absent depending on the type of machine learning. If there is ground truth for every single input, we are talking about supervised learning. If none of the inputs are labeled, it's unsupervised learning. Considering machine learning tasks, we can mention regression and classification. Regression is when the set of ground truths contain continuous values, for example in the case of estimating prices for housing, and in classification problems, the dataset is labeled with different classes representing different categories of things, these are discrete values) and bounding box regression (in which case the algorithm tries to guess the exact place of an object or more objects)[6].

Perceptrons

Perceptrons are the smallest working units of neural networks. They are intended to simulate the working process of a brain cell, so they are also called artificial neurons[25][41]. Their job is to transmit signals throughout special structures, the neural networks. The output of the i^{th} neuron is calculated the following way: $h_i = \sigma(\sum_{j=1}^N (V_{ij}x_j + b_i))$, where

- σ is an activation function,
- x_j is the perceptron's input(line vector, consisting of N elements),
- V_i is a weight vector belonging to the i^{th} neuron (this should be adjusted for each neuron until the output is sufficient)
- b_i is a bias enabling asymmetrical behavior.

Neural networks

Neural networks are machine learning algorithms containing multiple layers of perceptrons. The input of the first layer is the data sample and the output of the last layer is a prediction. The middle layers are called hidden layers[41].

Loss functions

Different measures can help us to calculate the distance between the predicted (y) and the desired (d) output. The ones I tried are:

- **Smooth L1 loss:** This function is commonly and effectively used for bounding box regression; it minimizes the difference between the predicted and the target values. Smooth L1 can be calculated as follows

$$L_{sL1} = \begin{cases} 0.5(d - y)^2, & \text{if } |d - y| < 1 \\ |d - y|, & \text{otherwise} \end{cases}$$

[23]

- **Binary Cross Entropy loss:** A loss commonly used in binary classification problems. Its formula: $L_{BCE} = d * \log(y) - (1 - d)\log(1 - y)$. [27]
- **IoU loss:** Intersection-over-Union or Jacquard index is commonly used in determining the similarity of two matrices. In our case, these two are the output of our network and the desired output. This can be calculated as: $L_{IoU} = \frac{Y \cap D}{Y \cup D} = \frac{Y \cap D}{|Y| + |D| - Y \cap D}$ [40]
- **GIoU loss:** GIoU or Generalized Intersection over Union of two areas A and B can be calculated as: $L_{GIoU} = L_{IoU} - \frac{|C \setminus (A \cup B)|}{|C|}$ where C is their smallest enclosing convex object.

Optimizers

The goal of neural network training is to find the perfect set of weights for a given problem, with which the distance between the actual and the desired outputs are minimal. In other words, if the global minimum of the loss function is found, the problem is solved. There are different approaches and methods of finding the optimal weights. The main principle could be that by calculating the gradient of the activation function at a given point from all the parameters, the algorithm can find the strongest negative where the minimum supposedly is due to its randomness. These gradient-based methods tend to struggle with local minima if no measures are implemented to step out of it. Optimizers have a parameter called learning rate, which decides the size of the steps they are taking on the gradient curve while changing the weights. I have experimented with different optimizers, which were:

- **Adagrad:** This is a gradient-based method with an adaptive learning rate. This means that it changes the learning rate based on the proximity of the solution; the steepness of the gradient curve while learning. This is a useful approach because at the start of the learning process the features are sparse; their parameters are less

dense than for the features at the end of the learning process. In other ways, later on, it takes nuances of changes to improve the results, while at the start bigger steps are optimal. $w_t = w_{t-1} - \eta_t' \frac{\partial L}{\partial w_{t-1}}$, where the derivative of the learning rate is $\eta_t' = \frac{\eta}{\sqrt{\alpha_t + \epsilon}}$, ϵ is a small positive number to avoid division by 0, and α_t is the summation of the gradient square[24].

- **Adam:** This optimizer is a combination of two others. It uses the same momentum concept as the Stochastic Gradient Descent with momentum, but with a learning rate added, such as in the case of Adagrad or Adadelta. The momentum can be regarded as a computational implementation of the physical phenomenon. It is used in the gradient function; the present gradient has a dependency on previous gradients. The new weight and bias with this algorithm can be calculated as: $w_t = w_{t-1} - \frac{\eta}{\sqrt{S_{dw_t} - \epsilon}} V_{dw_t}$; $b_t = b_{t-1} - \frac{\eta}{\sqrt{S_{db_t} - \epsilon}} V_{db_t}$ where S_{dw_t} and S_{db_t} are the exponential weighted averages of past squared gradients ($S_{dw_t} = \gamma S_{dw_{t-1}} + (1 - \gamma)(\frac{\partial L}{\partial w_{t-1}})^2$; $S_{db_t} = \gamma S_{db_{t-1}} + (1 - \gamma)(\frac{\partial L}{\partial b_{t-1}})^2$) and V_{dw_t} and V_{db_t} are the exponential weighted averages of past gradients ($V_{dw_t} = \beta V_{dw_{t-1}} + (1 - \beta)\frac{\partial L}{\partial w_{t-1}}$; $V_{db_t} = \beta V_{db_{t-1}} + (1 - \beta)\frac{\partial L}{\partial b_{t-1}}$) [24]
- **AdaMax:** This is a variant of the Adam optimizer. Adamax generalizes the L^2 based update rule used in Adam to an L^p based rule. The variants become unstable when p becomes too big of a number. If $p \rightarrow \infty$ the algorithm becomes simple again.

Convolutional layers

These layers are used for feature extraction. They convolve the input image with a special matrix (filter or kernel, usually smaller than the image itself) with changeable elements (parameters). Convolution is an algorithm where a smaller matrix is slid over a bigger one and element-wise multiplication is performed at each place. Important parameters are

- Kernel size (K): kernels usually are $K_x \times K_y$ matrices, they have the same height and width. Their size therefore can be described by giving a single parameter; K.
- Stride (S): the step of the sliding, means the distance of the kernel's middle element's position along a single coordinate; x or y.
- Padding (P): While sliding a convolutional kernel through the image, the iterations begin at the first place where the kernel can be completely fit into the picture.

That is why for each layer, the edge pixels of the image would be lost in the output. To solve this, padding can be applied. Without padding, convolution tends to downsize the images, but on the other hand, with it, the extra features increase the complexity of the network needed with no added information value.

The inverse method is deconvolution, which creates bigger pictures with fewer channels.

Pooling layers

Pooling layers are used for downsampling, which leads to less computation. This also has parameters like kernel size and stride. Different types of pooling layers are max pooling (the output of a step is the maximum element of the values covered by the kernel) and average pooling (where the output is the average of the covered elements). There are different inverse methods of this, which produce an output with the same size as the picture itself (either by filling the corresponding places with the pooling kernel's output or placing the output somewhere in the corresponding placing's reach and filling the rest with zeros).

Activation function

These are the functions that produce an output of each perceptron as well as the output of the network. The types I considered were:

- ReLU: ReLU or Rectified Linear Unit transforms the output to $[0; \infty)$ scale. $R(x) = \max(0, x)$. The issue with ReLU is that when the outputs are in the negative range, they can fix the perceptron's output to 0 (this is also called the dying ReLU problem)[35].
- Leaky ReLU: This is created to address the dying ReLU problem. $R_l(x) = \max(\alpha x, x)$, where $0 < \alpha < 1$ is a small number. This is a good solution, in the sense that by permitting some negativity it can distinguish between a little wrong and very wrong outputs[35].

3.2 Methods used in used for court segmentation

There are different pre-existing methods that I found useful during my work concerning the court's segmentation. These include methods from the field of traditional computer vision.

3.2.1 Canny edge detection

This is an operator used to detect a wide variety of edges. Its working mechanism can be divided into five steps.

- i Smoothing out the input with a Gaussian filter to get rid of false edges generated based on noise and imperfections
- ii Computing intensity gradients for the picture
- iii Apply gradient magnitude thresholding, an edge-thinning technique
- iv Double thresholding, finalizing the edges
- v Hysteresis; getting rid of weak edges that are not connected to the main, stronger edges

3.2.2 Hough transform

Hough transform is a special case of radon transform used to detect lines or other 2D objects, such as circles analytically. It iterates through the input image and calculates a confidence value. This value tells whether a given pixel lies on an object of the required type. Lines, for instance, can be defined in the form of equations, such as $r = x\cos\Theta + y\sin\Theta$, in which case the coordinates (r, Θ) identify one single line. The transform iterates through each point and each of the possible lines that might go through them. Next up, those are selected where the most matching points are found in the image. A Hough parameter is a pair of floating point numbers that represent a dot in the angle-distance space. This pair of parameters defines an exact line in the image space [21].

3.2.3 Template matching

Template matching is a method used to calculate similarities of two or more objects from multiple images using various mathematical methods. Usually, in these kinds of tasks, there are target and template images (at least one from each). The aim is to locate the smaller template image(s) on the target image(s). This requires a measure of similarity or dissimilarity between the images. The selection of such a method depends on the target and template images. There are several different approaches to this task. Some examples are listed below.

- **Feature-based approach:** the algorithm looks for similar features (such as curves or control points) on the target and the template image.

- **Area-based approach:** also called correlation methods are a combination of feature detection and matching. Matches are measured based on the intensity values of the template and the target images. It uses the eigenvalues and eigenspaces of the images if direct matching is impossible. Methods using this approach are considered template matching in a narrow sense.
- **Naive template matching:** this method is used when the template is directly on the target without any changes made. One error used to calculate the differences between the template and the target is the squared difference.
- **Image correlation matching:** here the template and the target may have different noise measures and distortions. The similarity metric used must be the correlation between the two images. Correlation metrics can be cross-correlation or normalized cross-correlation as well.

[12]

3.3 Methods used for detecting moving objects

For the detection of moving objects, I first tried some implementations with traditional image processing like K-means clustering. After I found these mostly ineffective, I opted for more modern and conceptually complex ways in the field of machine learning.

3.3.1 Histogram analysis

This method creates a histogram from single-channel images (either from one of the color channels or from an RGB image's single color channel). It takes the pixel values and arranges them by the intensity in ascending order. With the help of this, image sharpening is possible or one can also detect the similarity of images based on their histograms[14].

3.3.2 K-means clustering

In image processing, K-means clustering is a method of dividing a picture into color classes based on pixel intensities. The function has an integer K parameter which decides the number of classes to group the pixels into. The first step of this algorithm is to give k mean vectors (one for each cluster, c_k) of pixel intensities and an error tolerance value(ϵ). In the second step, we calculate the Euclidean difference of each of the pixels;

$d = ||p(x, y) - c_k||$. After each Euclidean distance is calculated, we reassign the values for all the mean vectors;

$c_k = \frac{1}{k} \sum_{y \in c_k} \sum_{x \in c_k} p(x, y)$. We iterate the Euclidean difference calculations and the reassignments of mean vectors until the error is smaller than or equal to ϵ [5].

3.3.3 U-net architecture

In most image processing tasks, an ideal output to have is a pixel-wise classified version of the input. This provides information about the class some objects belong to and shows us where exactly on the picture we can find it. U-net is an architecture published in 2015 by O. Ronnenberger et al., which aims to produce such outputs. This architecture has a decoding (contracting) and an encoding (expansive) path which consist of several layers of the same levels. Each decoding level sends its output to its corresponding encoding level as well as to the next decoding level as seen in figure 3.1.



Figure 3.1: U-net architecture, [33]

The first decoding level creates a 64-channel output and each similar layer doubles this. They consist of 2 ReLU-activated convolutions with 3x3 kernels and a max pooling layer with 2x2 kernels. The encoding layers also consist of two 3x3 ReLU-activated convolutions, but instead of a max pooling layer, they have up-convolution, which helps to return the series of outputs to the original image size. At the beginning of the encoding layers au contraire to the decoding ones, the output always halves in channel numbers.

3.3.4 R-CNN architectures

R-CNN [10] or Region-Based Convolutional Network is a tool used for object detection tasks. R-CNN’s original version consists of three modules. The first extracts region proposals. These regions are the ones that might contain target objects. The second module extracts the features of each of the first one’s input regions. The third part of the R-CNN classifies the regions using a linear support vector machine (SVM).

Region proposals

The first module proposes a large number of regions for each input picture. If its operation is ideal, all of the final regions with the desired objects are among these with high probability. There are different methods of computing these region proposals, from which selective search [38] is the fastest and most accurate. This method uses different graph-based algorithms to cluster the input based on different metrics of pixel similarities.

Feature extraction

The output of the first module consists of 2000 proposed regions. These are fed to the second module, a convolutional neural network, which outputs their 4096-dimensional feature vectors.

Linear SVMs

The first two modules result in regions with given values for each feature attribute. These attributes can be perceived as axes of a multi-dimensional data space in which points are assigned to each region unambiguously. Linear SVM-s aim to classify these points by choosing a hyperplane with the maximal margin. These hyperplanes can be constructed based on the closest data points of each class to the others called support vectors[8].

Versions and developments of the concept

R-CNN in itself is far from perfect. Its main flaw is that it takes too much time to compute. Classifying around 2000 regions per image is time-consuming. There are models, which took R-CNN as a base and aimed to reduce its computation time. Two such models are Fast R-CNN and Faster R-CNN. Fast R-CNN introduces RoIPool (Region of Interest Pooling) and Faster R-CNN adds a Region Proposal Network (RPN), which both aim to minimize the number of regions to be classified.

3.3.5 Mask R-CNN architecture

Mask R-CNN is a model used for instance segmentation. This task combines object detection and semantic segmentation and normally it has two outputs, a bounding box and a fitting label (such as in the case of Faster R-CNN). Mask R-CNN also outputs a binary mask as the objects' segmentation. The main addition, therefore, compared to Fast R-CNN is the mask segmentation output. The branch that computes this mask is parallel to the other that outputs the bounding box recognition branch [13].

3.3.6 YOLO algorithm

For humans or animals to recognize an object takes only one look; it is usually easy to identify objects seen before, even out of context, or a deformed version of them. YOLO (You Only Look Once) algorithm aims for the same effect. The processing of images with it is fairly simple. First, the input image is resized for frugality reasons. Then a convolutional network simultaneously predicts bounding boxes and class probabilities. Finally, the result is thresholded and the bounding boxes with higher probabilities are the result.

The benefits of this method compared to traditional object detection are its simplicity, precision, speed, global reasoning, and ability to learn generalizable representations. It creates a regression problem from object detection, so it does not require a complex pipeline with long runtime. It makes YOLO suitable for real-time detection on videos. Its global reasoning means that when the selection of bounding boxes happens, the network sees the whole image at once. This helps in the decoding of contextual information, which sliding window techniques lack. Learning generalizable representations provide YOLO with some ability of abstraction, it outperforms traditional methods in object detection on artworks after being trained on photographs.

An important flaw that YOLO has is its accuracy. While identifying objects works well, localizing them can be problematic for this algorithm[31].

Improved versions

Since the source code of YOLO is public, teams of data scientists have been improving its structure while creating different versions of this architecture. In this research project, I used YOLOv5. This was the first YOLO version implemented in Python instead of C. The structure of YOLOv5 is the following:

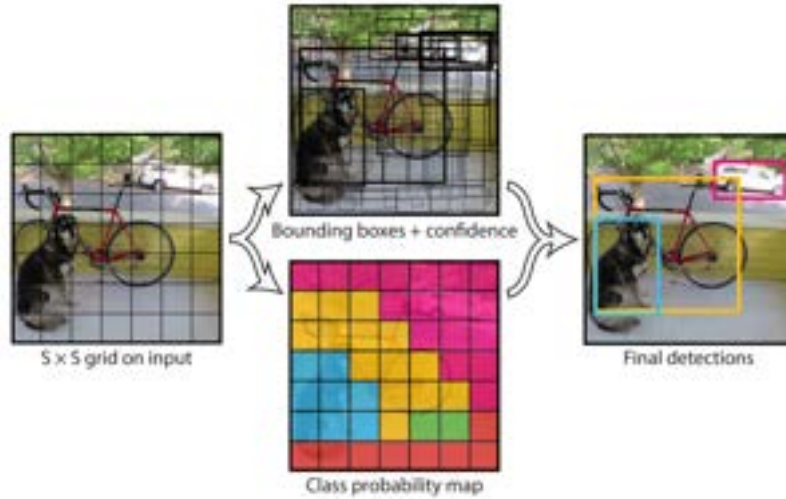


Figure 3.2: Model of the YOLO algorithm's workflow, [31]

- **Backbone:** Focus structure, CSP network

CSP refers to Cross-Stage Partial connections. In these kinds of networks, similarly to the U-net architecture, each layer is passed through convolution with denser output and then concatenated to the original input creating the new output.

- **Neck:** SPP block, PANet

SPP or Spatial Pyramid Pooling increases the receptive field and separates the important features resulting in a fixed-size output regardless of the input size.

- **Head:** YOLOv3 with GIoU loss

The backbone is the feature extractor part, Neck forwards combine and aggregate the features to the Head module where the detections are made [37].

3.4 Methods used for tracking

The base of most state-of-the-art tracking methods is the Kálmán filter. It has been mathematically proven to be the perfect solution to predict states of processes that can be characterized by discrete-time functions. As the equations of such a filter are sufficient for one target, multi-target trackers were developed. They all attempt to lead back the task to multiple single-target tracks where the Kálmán filter is applicable.

3.4.1 Kálmán filter

Kálmán filter is a recursive algorithm for the optimal solution to the discrete-data linear problem. The method consists of several mathematical equations which provide estimations of past, present, or future states of objects even without knowing their precise state (in our case, movement) equations.

The processes that the Kálmán filter applies are time-controlled and discretized, their states can be described with a linear stochastic equation (the equation below represents the prediction for the $k+1$ th state of x , an n -dimensional vector of real numbers)

$x_{k+1} = A_{k+1}x_k + Bu_{k+1} + w_{k+1}$, where

- x_k is the previous state of x process,
- A_{k+1} is the transition matrix,
- B is a control input matrix,
- u_{k+1} is a control input vector,
- w_{k+1} is a Gaussian noise.

The true (observed) state of x at the $k+1$ th moment is the following: $z_{k+1} = H_{k+1}x_{k+1} + v_{k+1}$. The use of this filter has two stages, predict and correct. This means that by comparing z_{k+1} and x_{k+1} after several instances the matrices used to compute x can be written and if the movement of the object changed (e.g. in direction or speed) they can then be corrected as well [42].

3.4.2 DeepSORT

Kálmán filter was initially designed to operate with one target to follow. If there are more targets, such as in the case of player tracking, a multi-object tracker (MOT) is needed. The DeepSORT algorithm combines the chosen deep learning method's output with a SORT algorithm.

SORT

SORT or Simple Online Real-Time Tracking in itself has 4 core components [44].

- Detection: In this case, the output of a neural network.
- Estimation: Using the Kálmán filter's linear velocity model, we propagate the detections to the next frame.

- **Association:** This component decides between the possible targets to assert each detection to, by assigning them to different tracks, respectively.
- **Track Identity creation and destruction:** This part ensures that when a target enters or disappears a new track is created for it or its track is getting destroyed. The disappearance of a track happens when the detector part of the system does not predict any bounding boxes that intersect with the ground truth bounding box belonging to that track for a few consecutive frames of the input video. On the other hand, if a detection's IoU loss is too big with all the tracks' current states, a new path is created for it.

3.4.3 AlphaPose

AlphaPose is a multi-person pose estimation and tracking system. Its backbone used to detect instances is by default a Fast R-CNN. The uniqueness compared to YOLOv5 is that it does not only provide information about the whereabouts of the instances, but it also estimates their limbs' exact location. Another difference is that this classifies only humans[7].

Input data

The input for training the detector module consists of images of people labeled by a stable number of fixed points of their bodies. These points are usually joints but they can also be facial features. The number of points depends on the input dataset, which declares it with different gaussian weight for each (the weighted sum of the fixed points indicate a certainty of detection).

Pose estimation

The module responsible for pose estimation in AlphaPose is called RMPE or Region Multi-person Pose Estimation network. Its pipeline is shown in Figure 3.3. First, a Spatial Transformer Network (A CNN-like spatially variant learning module), selects Regions of Interest from all of the proposals. On these regions, the SPPE (Single Person Pose Estimator) block finds the points of joints and identifies them, then connects them accordingly. Finally, an SDTN (Spatial Detransformer Network) maps the found pose structure back to the original image.

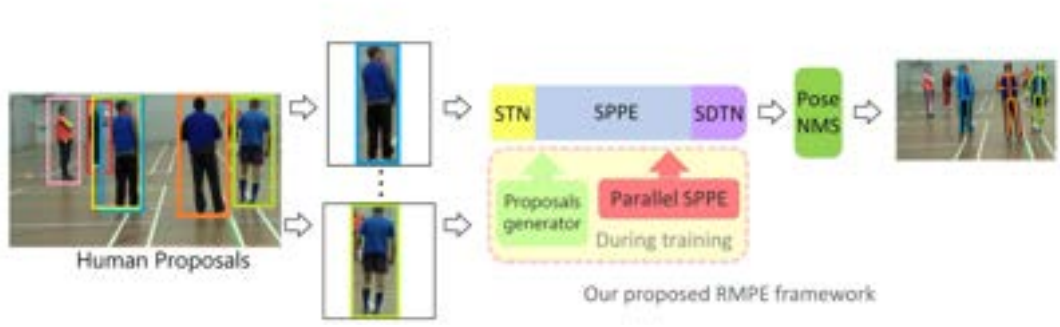


Figure 3.3: **RMPE pipeline** [31]

Modules for tracking

AlphaPose’s code package yields 3 different options for tracking people and their poses. These are PoseTrack, Towards-Realtime-Mot, and PoseFlow. They are all different kinds of multiple-object trackers. The principle behind them is similar to DeepSORT’s. They take detections from an ML architecture (in this case, after processing with the pose estimator module), then try to assign them to already existing paths. If they can not, a new observed path is created.

Chapter 4

Planning

4.1 Comparing traditional methods and machine learning

Tasks involving computer vision usually yield two approaches, traditional methods and the use of machine learning. Traditional methods, such as edge detection, template matching, or histogram analysis tend to be unsupervised processes. This means that they do not need ground truth labels for images or instances. This makes data preparation for them more comfortable, as the process of labeling can be meticulous. On the other hand, the fact that their datasets do not necessarily have labels makes it harder to evaluate their outputs. Differences between computation time and resources needed may vary depending on the complexity of the traditional algorithms and machine learning algorithms in use. For tasks involving moving object detection I mainly used machine learning algorithms, as they seemed more efficient for such tasks. For court segmentation, on the other hand, traditional methods seemed to do the job sufficiently and for object tracking the use of Kálmán filters is simply unavoidable (as they were mathematically proven to be the best solution for such tasks, there were no ML algorithms perfected for tracking).

4.2 Choice of segmentation type

As mentioned previously, the use of machine learning is an important part of object detection. There are, however, several different kinds of network outputs that could yield a potentially good solution to such a task. It's important to mention here, that in the case of players it's needed to distinguish between instances and identify players on the field individually.

Our first and perhaps least complex solution is bounding box regression. There, the creation of datasets is easy as labeling data manually requires only four coordinates per

instance. Networks like the different YOLO versions were designed to solve such tasks. On the other hand, pixel-wise segmentation is also possible. It has difficulties with instance segmentation, so that would not solve the identification for different players. Another problem with it is the labeling of our dataset. Segmenting the image manually is very precise and time-consuming even for just the ball detection. I still think that it is worth a try to feed such a network (eg. U-net) some images with binarized masks of bounding boxes for a ball detection task.

The fanciest solution to player identification is pose estimation by identifying the joints (with AlphaPose). Its advantage is that it can yield precise information about the situation and maybe even the activity of the players. A problem with this is that it requires even more precision and time to label such datasets, especially since the number of visible joints of players is not constant due to their environment. Also, in our case, such precision may not be by all means necessary. Because of this, I chose not to label any data for this task. There are open-source datasets online that can help; they don't contain many instances involving aquatic environments though. For this, I don't have high hopes, but will still try because it might yield the best possible solution.

Chapter 5

Implementation

5.1 Python packages used

When deciding between programming languages for this task I opted for Python. This language has many built-in functions which facilitate its use, also with its specialized packages such as Pytorch, it is the go-to choice of most data scientists and AI researchers. Below, I describe the packages which I found the most useful during my work.

5.1.1 OpenCV

OpenCV is a package optimized for computer vision tasks. It works with several other programming languages too, such as Javascript or C++. This package has spared me from the hours of meticulous work it would take to implement non-AI algorithms such as K-means clustering or Canny edge detection. It has implementations for the most common and difficult tasks of image processing.

5.1.2 PyTorch

This is a package optimized for deep learning. Torch.nn contains blocks that offer a comprehensible and easy way of building neural networks.

5.1.3 SciPy

SciPy is a library useful for scientific calculations. This came in handy when I wanted to create the trajectory of the ball by calculating the centers of each prediction.

5.2 Other softwares used

5.2.1 Linux environment

Linux is an open-source operating system. One of its main advantages is that it is free to use. Therefore, libraries like PyTorch were developed on it in the first place; their support on other operating systems is limited, and their performance is also better on Linux [1].

5.2.2 JupyterLab

JupyterLab is a web-based environment for creating and running code, notebooks, and data. Its interface allows the installation of different extensions, which facilitate the widespread use of different functionalities [30].

5.2.3 Kdenlive

Kdenlive is a free open-source video editor software. It facilitates the manual screening of important data from the whole input video [20].

5.2.4 CUDA

CUDA is a GPU (Graphic Processing Unit) handler software. It redirects the use of all the accessible GPU-s to compute non-graphics processes (such as most tasks in machine learning) as well. This is beneficial because in these tasks, using a GPU is more efficient than the default use of Central Processing Units [29]

5.2.5 Docker

Docker is a tool, which is used for the virtual delivery of software in packages called containers. It facilitates the sharing of applications between computers and servers. Its architecture can be seen in Figure 5.1. An important feature is that docker isolate containers from their environment and so they ensure their operating system independence and uniformity on different devices.

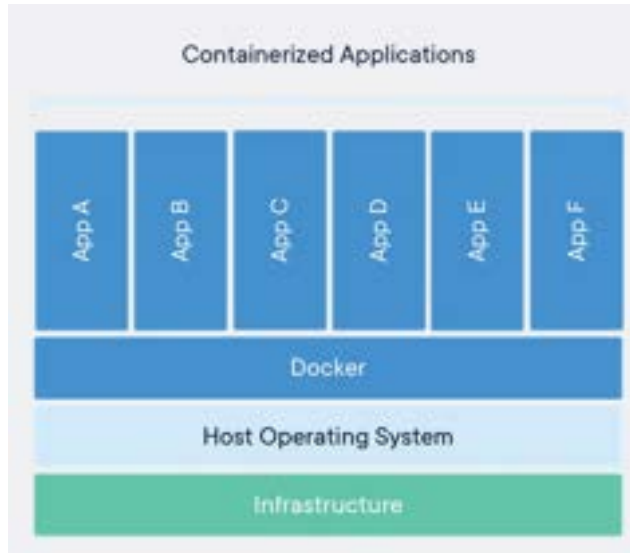


Figure 5.1: Docker architecture [43]

5.2.6 Conda

Conda is an open-source package and environment management software. It facilitates the management of python libraries and switching between their different versions (by creating new environments).

5.3 Preparing versions of the dataset

5.3.1 Extracting important information

I used a match from 2021/22's LEN Champion's League, which took place in Italy, Milan between the teams Pro Recco and Genesys OSC. It was a match on the 10th day of the preliminary round's day 10 (2022. February 2, 9 PM) and resulted in the victory of the Italian team. Only those parts of the footage were useful which depicted the match itself from the right camera angle (without close-ups of the players), so the first step of preparation was to cut the video. For this, I used Kdenlive in a Linux environment. I extracted about 48 minutes of useful data from an 86-minute-long video. This was already sufficient for methods not requiring labeled datasets (ie. for testing traditional image processing methods), but for machine learning, I needed some pictures where the objects' placement was pre-identified.

5.3.2 Dataset labeling

Creating the labels

My first instinct for machine learning was to use bounding box regression. The main reason for that is that it does not require high precision to create a labeled dataset. First, I cut some frames from the video based on their color similarity thresholds to a template image, then I used a Python program, that was already in use for labeling other kinds of data. Three versions of this exist now; one, which is used for labeling players en masse or by instance, and another for ball labeling. They create CSV files with the frames' filenames and coordinates of the bounding box (top left X, top left Y, bottom right X, bottom right Y, accordingly to the `cv2.rectangle()` method). For the en masse player annotator, these variables store lists of corresponding coordinates, while for the ball labeling, all the coordinate variables are integers. In the case of individual player annotation, the bounding boxes are the same, yet they are stored in CSV-s grouped by the players they belong to. This kind of labeling is needed for the case of tracking evaluation. In this case, I labeled a short video frame by frame, distinguishing the players. In some cases of ball annotation I could not or could only barely identify the object on the picture, these frames simply were subtracted from the machine learning dataset and the coordinates next to it are irrelevant; to distinguish these, the ball segmentations' CSVs contain a visibility variable as well.



(a) Annotating ball data



(b) Annotating player data
en masse



(c) Annotating player data
individually

Figure 5.2: Images of the annotator's User Interface after annotating one image

From these, the labeling of the ball took the least amount of time (for one picture), but this also adds the least amount of new information. The most difficult thing to label was the video with the individual players. In total, I labeled around 3400 images for the training, 1100 for the test, and 1100 for the validation datasets (these include only the instances where the ball was visible). In the case of en masse player labeling, from these

images (choosing them from the targeted datasets, respectively) I labeled 1500 training images, 300 test, and 300 validation images. Finally, I only labeled a 10-second video (about 300 frames) with the individual player annotation tool, as it serves only evaluation purposes.

Preprocessing annotations

As mentioned before, the annotating script's output is a CSV file. What is difficult in this format is that it can only store strings, yet we need to extract numerical information in the data loaders of U-net. To facilitate this, I transformed the data containing ball coordinates and the en masse player information into .json files. In the case of ball data, the keys of each image are the filenames and these contain coordinate names from CSV (eg. TOPLEFT_X or BOTTOMRIGHT_Y) with their values, respectively. Players' data is also stored grouped by image filenames but they are in a list of ordered lists of coordinates, where the outer list represents the players found on each image. It is important to note, though, that here individual players are not identified and their order inside these lists does not indicate their identity or belonging to either of the groups.

YOLO requires special input labeling, these are stored in .txt files individually for each image. The .txts contain lines representing each ground truth instance on the corresponding image. Some coordinate transformation was needed for this, as instead of the top left and bottom right corners, the sufficient input is the top left corner with the width and height of the bounding boxes. A line of information contained the class identifier and mentioned coordinates normed by the images' width and height in order. I made three training datasets like this, containing the players only, the ball only, and both listed.

5.3.3 Dataloaders

Several of the neural networks in use need their custom data loaders made with the annotations mentioned previously. For the sequential network used for ball detection by bounding box regression the data loader's input is the .json file with the ball coordinates by image, the folder where these images are found, and possibly transformations to preprocess the image with. Its outputs are the (possibly preprocessed) image, the bounding box coordinates, and the filename.

U-net is also used for ball detection. This requires a special data loader, which loads a binary mask with our data. This mask is made by projecting the bounding box on an

empty segmentation. The input made by this technique, therefore, is not very precise, it contains false positive pixels.

5.4 Court segmentation

This part of my task seemed more optimal using traditional image processing methods. There are several objects with fixed places on the water polo court; these usually do not move. Objects like this are buoys marking distances on the horizontal edge of the court, the goals and goal lines at the horizontal edge, other lines marking the edges of the court, and boxes in the corners.

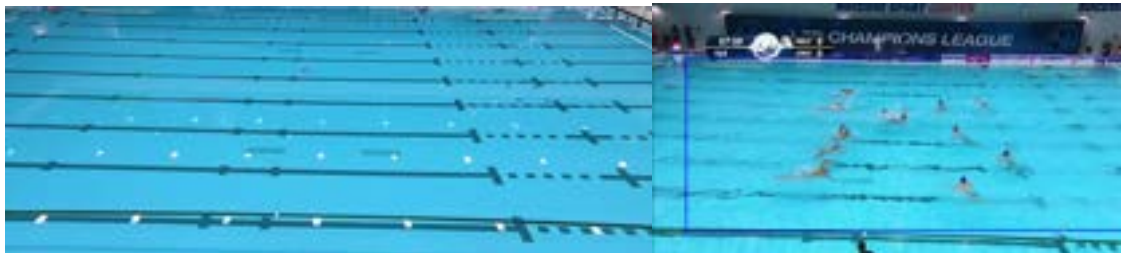
5.4.1 Segmentation using Hough lines

The idea behind this implementation is that the court is surrounded by horizontal and vertical lines colored red, yellow, and white. In theory, if we threshold the colors of the court and search for lines on it using the Canny edge detector and Hough lines, we can find these edges and segment the court based on them. A grave disadvantage of this method is that it contains complex calculations that can be time-consuming, so this method is not suitable for on-the-fly computations.

5.4.2 Segmentation using template matching

Template matching using the whole court as template

A naive approach was to cut out an image with the whole court visible on the frame and use it as a template to find. On one hand, the result of this procedure yields a surface with mostly water on it, but it is far from perfect, as it cuts out some of the court's area as seen on 5.3. This is because the template's size is constant, and after finding its most certain place CV2's algorithm prints its dimensions on the picture.



(a) Template image

(b) Target image matched

Figure 5.3: Template and target image of naive template matching

Template matching using fixed points

The idea behind this method is that there are buoys that mark the horizontal borders of the court and boxes and goals marking the vertical edge. By using template matching on the image and pictures of these fixed points as templates, the segmentation of the court is possible as written below.

There are 9 buoys (as seen in Figure 5.4 surrounding a horizontal edge of the court, four red, two yellow, and three white ones). Only the upper horizontal line is visible on the video, therefore searching for the 9 buoys and fitting a straight line through their bounding boxes' lower edge results in the horizontal segmentation of the court.

Due to the camera angle, the vertical line is not perpendicular to the horizontal one on

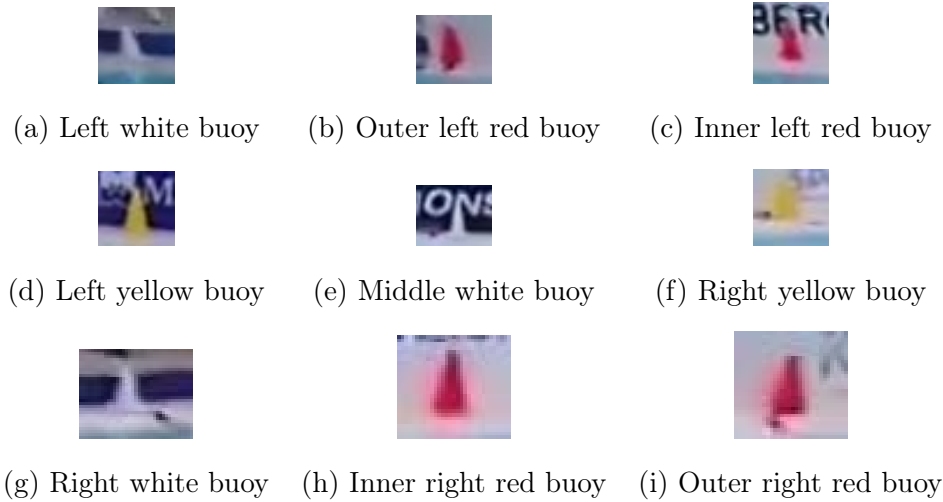


Figure 5.4: Template images used for finding buoys, horizontal alignment

the video. This means that more fixed points are needed to be selected for defining the vertical line. For defining the vertical line, the objects on the templates are red boxes and goals. From there, I selected fixed points on the found templates to fit our line through with the method of trial and error.

Other important place markers on the frames are boxes and goals (Figure 5.5). In the case of the left vertical line, these are the $2/5$ th of the bottom line of the red box's boundaries and its top right corner. For the right line, it is the $3/5$ th of the bottom boundaries and the top left corner.

Since we know the distances of all the fixed points in real life, in practice, we could transform all the found coordinates of the tracked objects on a 2D map defined by these points by an affine transformation. A grave problem with this concept is that the hor-

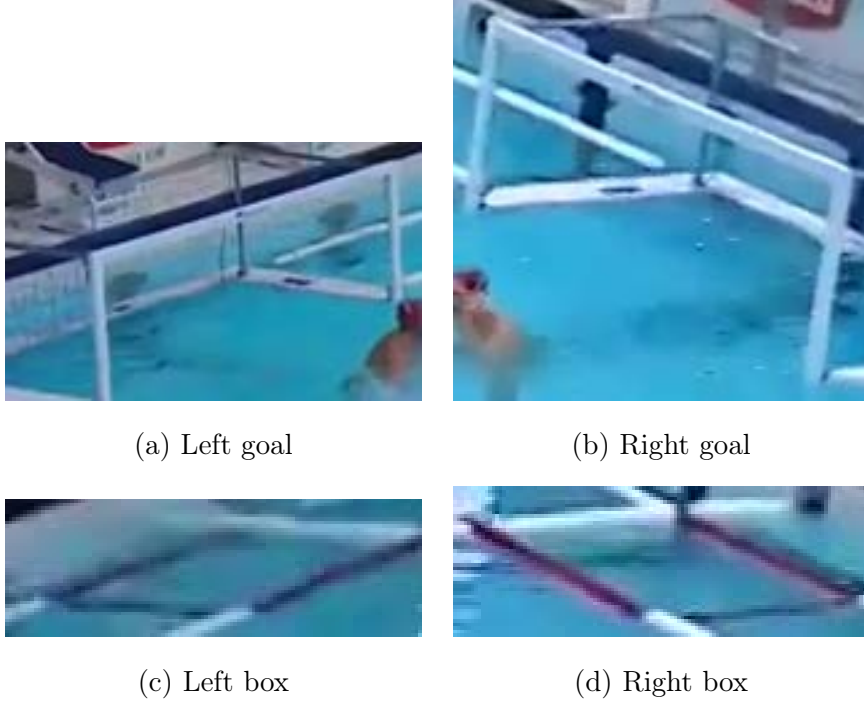


Figure 5.5: Template images used for finding goals and boxes, vertical alignment

horizontal boundary is usually visible, but the vertical ones are only occasionally present. This means that the fixed points found on our images are usually collinear, hence do not define a 2D space. Although the algorithm is fairly slow, it would be the ideal method for this problem, as it also provides information about the situation of the frame and the objects present.

5.5 Moving object detection, ball detection

5.5.1 Ball detection with traditional methods

Color thresholding

In this method, I tried to assign numerical values to things visible to the human eye. This is a simple and even naive approach. The main idea was that the colors of our target (the ball) are dark blue, yellow, and white, so if neither of these colors appears at a specific place, that means it is probably not the ball. This way, we can create a mask that contains the target among other things that we need to filter out. Here I used a threshold for the given colors rather than a single value because with different lighting conditions the same objects can appear lighter or darker in color.

Circle Hough transformation

Once we got a mask out of color thresholding, the next step would be to find the silhouette of the ball on it. The main shape feature of such is that it is round. So in theory, if we want to find the ball, chances are that it is the roundest object in the image.

Division by K-means

K-means clustering is another color-based approach. With that, the whole image is divided into classes by the pixels' color. I used two classes here so that the water can be subtracted from the image and create a mask of everything else. We then need another measure for all these non-water objects to select the ball. One of the ideas applicable here is using a circle Hough filter on the mask, another is to calculate a probability for each object being a circle. The roundness of an object can be calculated as: $P = \frac{4\pi A}{P^2}$, where A is the area of a coherent object and P is its perimeter. This measure returns 1 for filled circles [4].

5.5.2 Ball detection with handmade sequential network

Bounding box regression seemed evident for the ML data type, as I labeled the ball with bounding boxes. In this implementation, I used a network with a sequential pipeline which I developed (with my teammates) based on a torch tutorial for last semester's Neural Networks class competition. It was used for the regression of one value (the sharpness of its input picture) so I had to modify its last layer to give 4 numbers of output instead of only 1. I used it with Adagrad optimizer that had 0.0005 as its learning rate and a smooth L1 loss [18].

5.5.3 Ball detection with U-net

I tried detecting the ball using pixel-wise segmentation too. For this, U-net seemed the best; the base of which comes from an article about the network's Pytorch implementation [34]. My U-net has only two layers which means that it is significantly smaller than the one on Figure 3.1 and therefore has way fewer trainable parameters. It uses batch normalization and the ReLU activation function. The first loss function I tried for this network was binary cross-entropy. I had to change it because, for that loss, a prediction with only zeros is nearly optimal, as the ratio of positive pixels in the ground truth is small. So I opted for IoU loss which penalizes non-intersecting objects rather than false

positives. For optimization, I used Adamax, as it was found to be the best choice for U-nets [28].

5.5.4 Ball detection with YOLOv5

Another possible solution to ball detection with bounding box regression was YOLOv5 [39]. This method does not require custom data loaders, it only needs a custom configuration file and the .txt formatted ground truth with class 0 being the ball itself. I tried all the different subtypes of YOLOv5 architecture first, training for 10 epochs.

5.6 Moving object detection, player detection

5.6.1 Player detection with Mask R-CNN and U-net

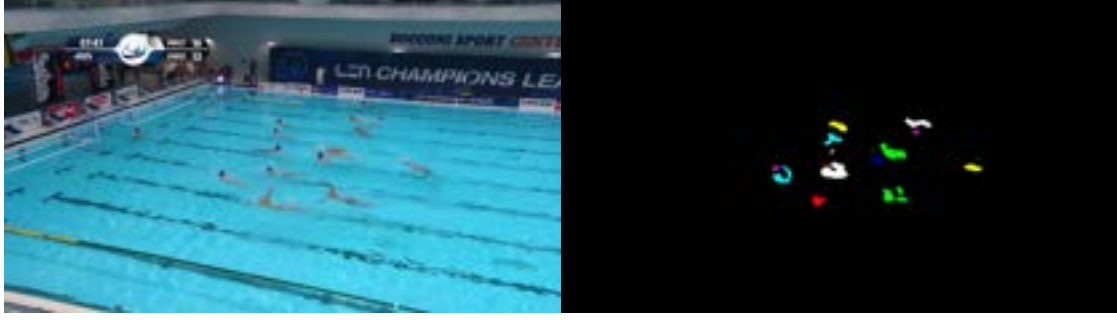
As mentioned before, Mask R-CNN takes the segmented image as an input together with the bounding boxes and the class labels (in my case, there is only one class). Since creating pixel-wise segmentation manually is roughly unsolvable, I had to choose a way to substitute it. The first idea was to create binary masks from the bounding boxes, but since one of the required inputs for mask R-CNN were, in fact, those, that would not give any new information to this network.

On the other hand, in the case of ball detection, I noticed that the segmentation of the U-net was more accurate than my bounding boxes, so I trained the U-net with the ball detection's configuration for ten epochs and used its output as masks. This, of course, does not yield as accurate results (Figure 5.6). As U-net is rather resource-intensive, I considered its further training on people a worst-case scenario, if nothing else works, and a waste of resources as it does not yield direct results. It is important to note that Mask R-CNN expects masks with different label numbers (colors) for each instance. I solved this by separating connected areas on the U-net masks and labeling them as different instances.

I trained Mask R-CNN for ten epochs, using Adamw as its optimizer with 0.001 as its learning rate.

5.6.2 Player detection with AlphaPose

As mentioned previously, the use of AlphaPose had not included training due to the lack of datasets constructed from this video [26]. I still ran the system with pre-trained data. Due to the lack of training information concerning aquatic environments, the pose esti-



(a) Original frame

(b) Generated segmentation

Figure 5.6: Example of U-net’s output for segmentation masks

mation was visibly unsuccessful for most images and players.



Figure 5.7: AlphaPose’s pose estimations, pre-trained on COCO dataset

As seen in Figure 5.7 AlphaPose pre-trained on universal datasets (in this case, COCO), gives almost perfect results in the case of viewers standing around the court, yet, almost always misses water polo players.

5.6.3 Player detection with YOLOv5

Data preparations

Preparing data for YOLOv5 did not require setting up data loaders. The only preparation needed was to create a custom config file for the dataset containing player data. After setting that up, the code was ready to use.

Trial of different configs

YOLOv5 exists in different complexities, which contain different numbers of parameters. These all are considered different architectures. These are YOLOv5x, YOLOv5l, YOLOv5m, YOLOv5s, YOLOv5n [39]. The last letter of their names indicates their complexity; vx (short for version extra large) contains the most parameters and is the most time-consuming to run, while vn (for version nano) has the least amount of parameters and the shortest runtime. First of all, I ran 10 epochs of training on each of these configurations, to get an idea of their accuracy.

	YOLOv5n			YOLOv5s			YOLOv5m			YOLOv5l			YOLOv5x		
Weights	none	P	Q	none	P	Q	none	P	Q	none	P	Q	none	P	Q
Max prec_val	0.4698	0.5381	0.5096	0.2482	0.1167	0.5202	0.2346	0.5661	0.5472	0.3101	0.5891	0.5330	0.3242	0.5664	0.5122
Max rec_val	0.4825	0.5115	0.5115	0.3392	0.2670	0.5265	0.6024	0.5493	0.6801	0.3658	0.5615	0.5228	0.3224	0.5404	0.5106
Max mAP_05_val	0.3673	0.4207	0.4100	0.1614	0.0974	0.4153	0.1506	0.4385	0.4202	0.2201	0.4539	0.4299	0.2150	0.4366	0.4115
Max mAP_05-95_val	0.0837	0.0981	0.0986	0.0378	0.0237	0.0975	0.0333	0.1043	0.0994	0.0523	0.1060	0.0995	0.0480	0.1035	0.0973
Min box_loss_val	0.0733	0.0712	0.0718	0.0282	0.0707	0.0715	0.0273	0.0707	0.0714	0.02804	0.0706	0.0715	0.0288	0.0710	0.0717
Min obj_loss_val	0.0772	0.07204	0.0758	0.0275	0.0764	0.0760	0.0387	0.0764	0.0735	0.0238	0.0761	0.0689	0.0279	0.0762	0.0420
a TP		2153	2165		2177	2204		2194	2147		2192	2172		2165	2094
a FP		1748	2005		1503	1888		1341	1730		1283	1726		1323	1631
a FN		1026	1014		1002	975		985	1032		987	1007		1014	1125

Figure 5.8: YOLOv5, full grid search results with two kinds of weights in pre-trained cases

The second step was to choose the one with the best preliminary results to run for more epochs (Figure 5.8). Out of the possible configurations, it was the YOLOv5l architecture that had the best preliminary performance using its pre-trained weights. It had the best precision, recall, mean average precision, and it was also among the best identifiers in terms of the number of true positives. I chose to continue the research with this base architecture. I trained that configuration for 100 epochs.

5.7 Moving object detection, simultaneous detection

Finding the ball and the players simultaneously would perhaps be the most elegant solution. A problem with this is that players are overrepresented and thus, the ball may get lost in the process. The network that is plausible for finding both can differentiate between instances (of players) and also locate objects. Both Mask R-CNN and YOLOv5 seem to do the job, yet, they all had their problems. YOLOv5 entirely failed the dataset containing only the ball, and Mask R-CNN is resource intensive and it is less confident in player detection. My chances seemed better with YOLO.

5.8 Tracking of moving objects

5.8.1 Naive approach

For ball tracking, I used a naive approach for U-net’s result. As the predictions were given and evaluated, this part was to calculate the midpoint’s coordinates of the biggest of the predicted patches and collect them into a list (as I previously described in the Implementation section). Its output can be visualized with an on-the-fly method, which draws the previous coordinates as red circles.

5.8.2 Application of Kálmán filters, in general

The filter works on discrete-time-controlled processes. Even though movement (eg. in sports) is continuous for our perception, videos that record it are built up of frames that discretize our progress well enough. There a step is $\frac{1}{30}th$ of a second if the video is recorded at 30 fps. In the case of the ball tracking, it did not seem to alter the tracking results significantly compared to my intuitive solution. For player tracking, the filter was never used in itself, yet it was always included in the MOT algorithms.

5.8.3 Player tracking with PoseFlow

PoseFlow is one of the tracking algorithms included in the AlphaPose package [26]. As it works with the previously made pose estimations which performed rather poorly, naturally, the tracking using this method was also visibly inefficient.



Figure 5.9: A frame from PoseFlow’s output on AlphaPose with ResNet backbone, pre-trained on COCO dataset

5.8.4 Tracking using DeepSORT

DeepSORT tracking is facilitated using YOLOv5 architecture. The first part was to interconnect the SORT architecture I found in [32]. Another slight modification I made

was to create multiple bounding boxes for each frame. This code normally showed only the DeepSORT algorithm's output but I also added the current detections for better comparison.

Chapter 6

Evaluation of results

6.1 Court segmentation

Since I used unsupervised methods for this task, my best chance of giving a quantitative evaluation is to select frames where both the horizontal line and at least 1 vertical line are visible. I chose 150 of these with 50-50 of the left and right goals visible and ones without the vertical goal lines. After this, I run the different segmentation methods and count the ones that I think were identified correctly. I gave up on the naive template matching, as it undersegments the court by far. The two that are left to compare are the ones using the fixed points as templates and the Hough line method.



(a) Accurately matched horizontal border with no vertical needed



(b) Accurately matched horizontal and vertical borders

Figure 6.1: Examples of accurate fixed point template matching results

As seen in Figure 6.4's a) subfigure, the Hough method always finds the horizontal borderline and misses around 30% of the vertical ones. This can be corrected with further refinement of the parameters.

Figure 6.4's b) subfigure, on the other hand, indicates that the template matching method needs further adjustments. It finds around 64% of the horizontal lines and hardly any of



Figure 6.2: Examples of faulty fixed point template matching

the vertical ones. The grave state of the vertical borders' matching might be explained by the game score's counter covering the left red box and the fact that with the movement of the camera angle, the angle of view on the red boxes changes, therefore needing different templates to precisely match it.

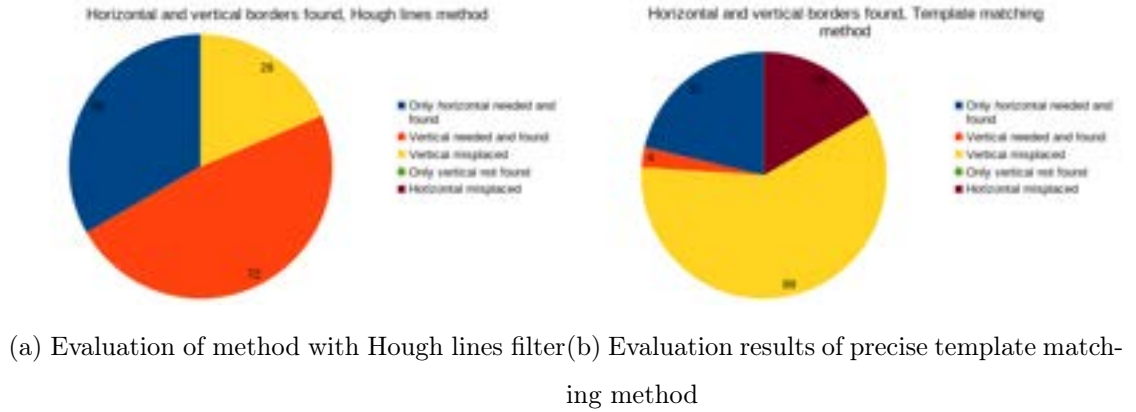
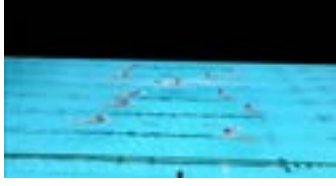


Figure 6.3: Evaluation of more advanced court segmentation methods

To sum up the state of the court segmentation, to date the Hough lines method works better by far. The perfection of the template matching method is more important though, because with it, the 2D parsing of the objects' data is possible, and this is one of this task's future problems to solve.



(a) Accurate horizontal border matching, no vertical line needed



(b) Faulty vertical border matching



(c) Accurate vertical border matching

Figure 6.4: Examples of accurate segmentations and a faulty vertical line segmentation using Hough filter

6.2 Moving object detection, ball detection

A metric that instantly came to my mind in the evaluation of this task was the confusion matrix. It computes the rates of true positives, false positives, true negatives, and false negatives. The problem with it is that in this dataset the ratio of positive to negative pixels is tiny, therefore in the prediction, true negative values are dominant, if the negative predictive value is not unacceptably poor (on average, a picture contains 140 positive and 129.459 negative pixels). Nevertheless, what I aim to see on these is the true positive pixels' ratio converging away from zero.

6.2.1 Average number of pixels in the testing dataset

The original video's width was 1920 pixels and its height was 1080 pixels. As working with images this size would require too much computation time and resources, I reduced the dimensions to $\frac{1}{4}$. The new frame size became 480x270. With this transformation, each frame contains 129600 pixels from which on average the frames contain about 140 with positive and 129460 with negative labels. Regarding the predictions by type, on average

- The **Kmeans**' frames contained 4 true positive, 879 false positive, 128583 true negative, and 138 false negative pixels;
- The **sequential network**'s predictions had around 0.05 true positive, 149 false positive, 129310 true negative, and 140 false negative pixels, and
- The **U-net**'s outputs had 93 true positive, 36 false positive, 129420 true negative, and 51 false negative pixels.

6.2.2 Results on confusion matrices

The confusion matrices did not tell me too many things about the efficiency of different methods. As it is shown in Figure 6.5, most of the pixels belong to the true positive category. At least it is safe to say that neither of the methods has a terrible true negative rate; they seem to recognize those more or less perfectly.

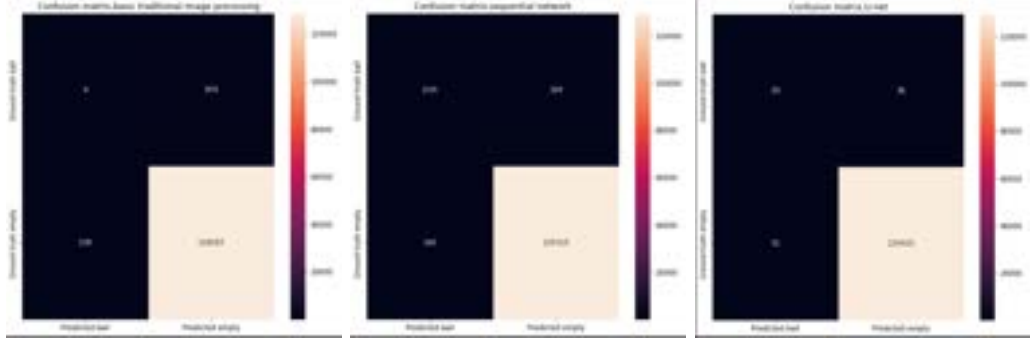


Figure 6.5: Confusion matrices (from left to right:Kmeans clustering method, sequential network, U-net)

6.2.3 Applying common evaluation metrics

As I mentioned, the majority of pixels are labeled as negatives. To emphasize the ratio of positive pixels, I applied some other metrics to emphasize the ratio of true positive and negative pixels independently from each other. The following metrics are in use:

- **Sensitivity:** true positive rate, $\frac{TP}{TP+FN}$
- **Miss rate:** false negative rate, $\frac{FN}{TP+FN}$
- **Fall-out:** false positive rate, $\frac{FP}{TN+FP}$
- **Specificity:** true negative rate, $\frac{TN}{TN+FP}$

From these metrics, a perfectly accurate classifier has 100% (or 1) sensitivity and specificity and 0% miss rate and fall-out. My tries had high specificity, yet their specificity could be improved according to these tests.

	Kmeans	Seq_net	U-net
Sensitivity	0.0282	0.0003	0.6458
Miss rate	0.9718	0.9997	0.3542
Fall-out	0.0068	0.0012	0.0003
Specificity	0.9932	0.9988	0.9997

Figure 6.6: Evaluation metrics applied to different methods of ball detection

K-means clustering and circularity

As figure 6.6 a) suggests, this method seems to find only about 3% of the pixels which contain the ball, while getting 1% of the negative pixels wrong. This means that the method finds approximately 308 times more false positives than true positives.

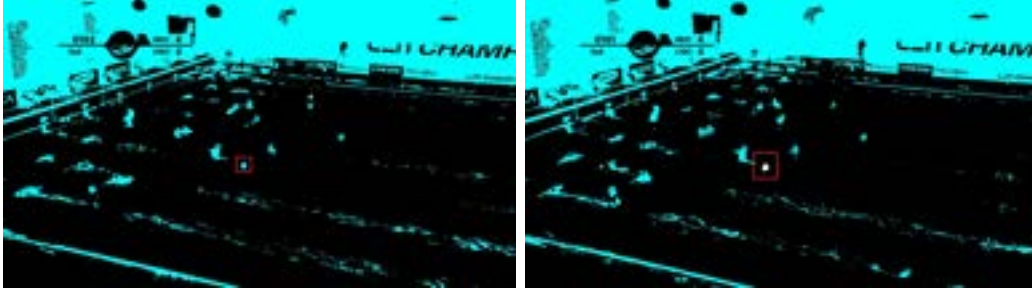


Figure 6.7: Two frames with Kmeans clustering (blue-black mask) and the circularity metrics (white patches) combined; to the right, the ball was found by the metrics, and to the left, it was only found by the clustering, among many other things. The area of interest is inside the red rectangles.

Sequential network

Figure 6.6 b) hints to even worse results in matters of sensitivity; here the amount of true positive pixels on the test data is so few that it proportionally does not reach even 1%. At least this method does not predict false positives either; the predictions seem almost always empty.



Figure 6.8: A frame with the ground truth (dark blue) and the predicted (red) bounding box, sequential network output

U-net

As we can see below (Figure 6.6 c)), this method is way more promising than any of my other tries. It finds 64% of pixels labeled positive on the test data, and the ratio of false

positives is below 1%. If we disregard the labeling and think about those pixels that contain the ball only (and not the water around it, which still is part of the bounding box that I made), this network seemed to be capable of even more (we can see this effect at 6.2.4, where the ground truth bounding box is labeled red, and the prediction's in green). It tended to label only those pixels inside the bounding boxes that contain the ball while throwing those out that do not. We can say that in some instances- this had done an even better job at finding the ball than I did, it certainly was more precise.



Figure 6.9: A frame with U-net's output applied to the ball detection problem (green) and ground truth (red) superimposed on it

6.2.4 Results by IoU scores

To check if these results are synchronous with the loss metric I originally used for back-propagating the U-net's predictions, I calculated the average IoU scores of the different methods. They turned out the following:

- $IoU_{Kmeans} = 0.003$
- $IoU_{SeqNet} = 0.001$
- $IoU_{U-net} = 0.475$

As expected, U-net yields by far the best IoU results from the tried method. It is important to note, though, that 0.475 for the IoU score is not a lot. This network might actually perform better than that because as Figure suggests, the segmentation tends to label only the pixels that belong to the ball itself and not its surroundings, as I did. This proves that using a segmentation network on bounding boxes might cause some problems. Still, this method has the best results by far; even including the performance of YOLOv5...

6.2.5 YOLOv5 results, an exception in the grading scheme

There only are preliminary results for this experiment. After being trained for ten epochs, none of the YOLO architectures (Figure 6.10) return any results. The fact that there was not a single true positive in the validation step's predictions alerts us of this problem. The problem might have been the unsatisfactorily amount of training data, which for this task contained only 1500 images with 1-1 instances on each; however, the recommended minimum amount is about 10000 instances. As there have already been better results (by U-net) I deemed further tests on this network unnecessary.

	YOLOv5n	YOLOv5s	YOLOv5m	YOLOv5l	YOLOv5x
Max prec_val	0	0	0.0028	0.0006	0
Max rec_val	0	0	0.0066	0.1333	0
Max mAP_05_val	0	0	0.0014	0.0003	0
Max mAP_05:95_val	0	0	0.0003	0	0
Min box_loss_val	0.0721	0.0720	0.0723	0.0722	0.0722
Min obj_loss_val	0.088	0.0899	0.0085	0.0083	0.0088
# TP	0	0	0	0	0
# FP	0	0	0	0	0
# FN	300	300	300	300	300

Figure 6.10: YOLOv5's grid search results, ball detection

6.3 Moving object detection, player detection

This case is different from the case of ball detection, as here there are multiple objects. This, of course, means more parameters; the neural networks that I used for object detection return a set of bounding boxes with confidence levels and classes aligned to each. The case of classes is evident; as we only have one class ('player') to detect, there is no need to examine losses concerning the misclassification of our objects. Some of the metrics that are in use for the evaluation of such networks' outputs are

- Precision: $P = \frac{TP}{TP+FP}$, where TP and FP are the amounts of True and False positive detections, respectively. This is also called Positive Predictive Rate because it indicates the chance of being correctly identified for objects that were predicted positive.
- Recall: $P = \frac{TP}{TP+FN}$ here FN represents the amount of false negatives. This metric is often referred to as the true positive rate and it represents the chance of correctly identifying an object labeled positive.
- mAP score: mAP or mean average precision is a rather complex metric. In statistics, average precision means the area under the precision-recall curve. In

such a figure, the found instances are ordered by their recall, which is represented by the X axis, and precision is plotted as a function of it on axis Y. As the number of objects detected is always finite, the formula for average precision is the following: $AP = \sum_{k=1}^n P(k) \Delta R(k)$, where $P(k)$ is the precision of the k^{th} element and $\Delta R(k)$ is the difference between the recall if the k^{th} and the $(k - 1)^{th}$ element. Knowing this, mAP can be perceived, such as $mAP = \frac{\sum_{n=1}^N AP(n)}{N}$, where N is the number of classes. Since in this case there is only 1 class, mAP is equivalent to AP. It is important to note, that the determination of the detections' positivity depends on the minimal IoU threshold of the bounding boxes; implicating, that with different thresholds, the results of mAP differ [17].

- F1 score: F1 score or F-measure is the harmonic mean of precision and recall. It can be calculated with the following equation:

$F1 = \frac{2PR}{P+R} = \frac{2TP}{2TP+FP+FN}$. The maximum of all F1 scores at different confidence levels is called the design point, where the pair of precision and recall are co-optimized. In this case, a high maximum F1 score and confidence are desired [22].

6.3.1 Mask R-CNN results

A frame with this model's output masks superimposed (with one color) and all the output bounding boxes drawn on it can be seen on Figure 6.11. Its pre-trained weights are from the first epoch because afterward, it started to overfit the bounding boxes on the training images.



Figure 6.11: A detection's output using maskRCNN

Based on the F1 curve belonging to this network (as seen on 6.12 b)), the threshold confidence is high, it is around 0.7. On the other hand, Mask-RCNN is not exactly optimal, as its maximum F1 score is only about 0.4. The Precision-recall curve belonging to this F1 score can be seen in Figure 6.12 a). The area under this curve is 0.2562.

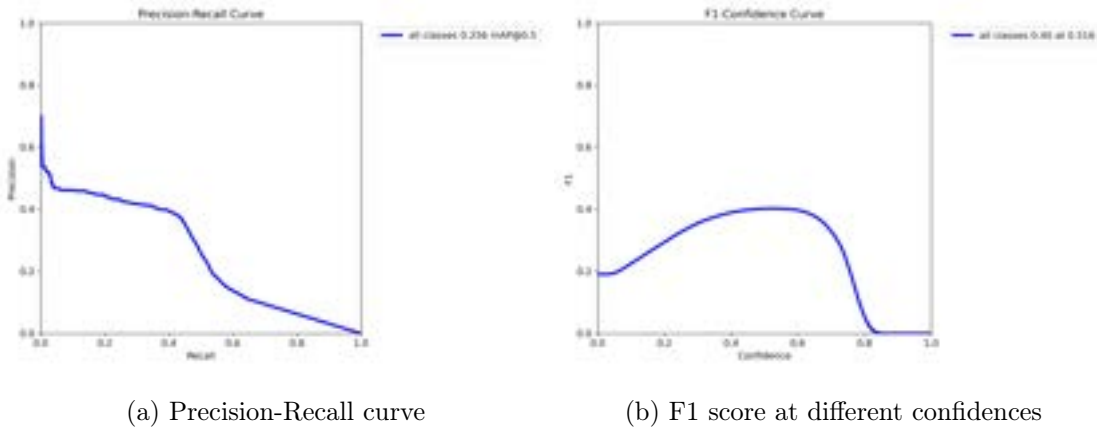


Figure 6.12: Precision and recall values of Mask R-CNN at different confidence score thresholds

6.3.2 YOLOv5 results

A frame from the resulting detections is in Figure 6.13. The numbers above the players' heads are confidence levels. This aspect is to be improved, as well as the habit of completely missing the left keeper, which might be a flaw in the training data.



Figure 6.13: A detection's output using YOLOv5l with pre-trained weights after 100 epochs

According to the F1 curve of this network (Figure 6.14 b), the threshold confidence is about 0.5 with an F1 score of 0.5. At this F1 score, the Precision-recall curve is present in Figure 6.14 a). The area under it is 0.3926, which indicates better performance, than in the case of Mask R-CNN.

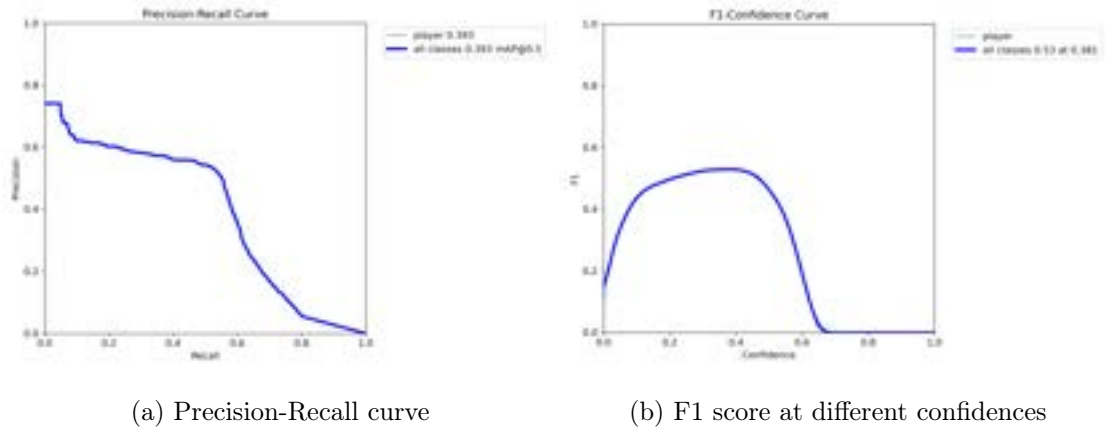


Figure 6.14: Precision and recall values of YOLO at different confidence score thresholds

6.3.3 Comparison of networks

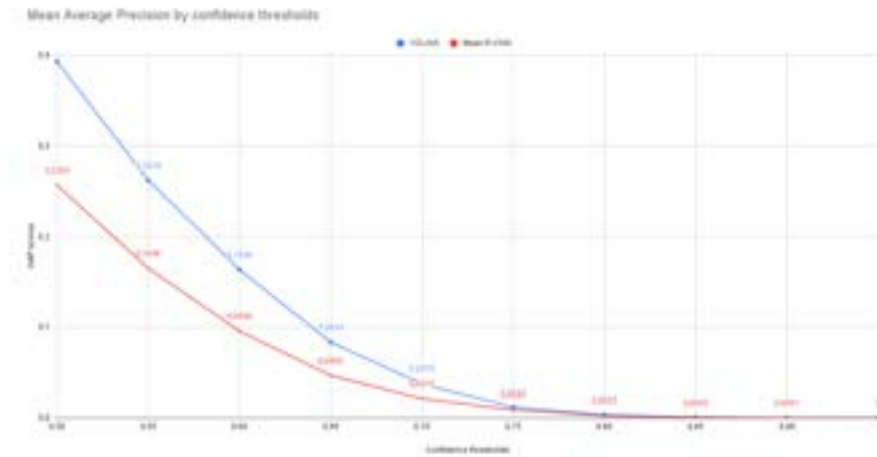


Figure 6.15: mAP values of YOLOv5 and Mask R-CNN at different IoU thresholds

As seen in Figure 6.15, both of the models' mAP-s visualized at the values of their maximum F1 scores' threshold confidence (these values being Precision and Recall), both of the confidence-mAP curves indicate some resemblance to exponential decay approaching 0 at confidence = 0.95. The average precision of YOLOv5 is better at most confidence thresholds.

6.4 Moving object detection, simultaneous detection

For this task, I made a smaller preliminary grid search on the YOLO architectures used before, first to get to know if the system manages to find the ball. As seen in Figure 6.16, the networks with no pre-trained weights still missed the ball after ten epochs of learning.

An interesting phenomenon occurred; the networks started to output more false positive detections with the ball present in the dataset than in the case of the one that contains only the players. This is unexpected behavior, and as such, I found it surprising.

	YOLOv5n	YOLOv5s	YOLOv5m	YOLOv5l	YOLOv5x
# Player predicted player	2074	2192	2144	2150	2143
# Ball predicted ball	0	0	0	0	0
# Ball predicted player	0	0	0	1	0
# Player predicted ball	0	0	0	0	0
# None predicted player	2204	2264	2083	1746	1894
# None predicted ball	0	0	0	0	0
# Player predicted none	1105	987	1035	1029	1036
# Ball predicted none	300	300	300	299	300

Figure 6.16: YOLOv5 preliminary grid search on the dataset containing data about both players and balls

6.5 Tracking

6.5.1 Ball tracking



Figure 6.17: Generating trajectory and prediction on-the-fly, green patches: prediction, red dots: previous and current midpoint coordinates

6.5.2 Player tracking

As the output of PoseFlow seemed insufficient, the best possible method for future uses is DeepSORT. Below is the evaluation of its results. In player tracking, there are also various metrics, that provide information about the correctness of the output. I calculated the following ones: [36]

- **IDF1:** F1 scores aggregated by the longest trajectory found in each track.
- **IDp:** Precision scores aggregated by the longest trajectory found in each track.

- **IDr:** Recall scores aggregated by the longest trajectory found in each track.
- **MT:** the number of mostly tracked trajectories.
- **PT:** the number of partly tracked trajectories.
- **ML:** the number of mostly lost trajectories.
- **IDs:** number of ID switches, it counts all the instances of changing the prediction ID of all the ground truth tracks.
- **MOTA:** or Multiple Object tracking Accuracy $MOTA = 1 - \frac{FN+FP+IDs}{GTP}$, where FN represents all the false negative instances on all the frames, FP respectively all the false positives and GTP is the accumulated number of positive instances in the ground truth.
- **MOTP:** or Multiple Object Tracking Precision accumulates the precisions on each frame in the following manner: $MOTP = \frac{\sum_{i,t} IoU_{t,i}}{TP}$ where I and t represents instances and frames.

IDF1	IDp	IDr	MT	PT	ML	IDs	MOTA	MOTP
0,437	0,481	0,4	0	12	1	30	0,179	0,628

Figure 6.18: A frame from DeepSORT’s output with YOLOv5l backbone, pre-trained for 100 epochs

I ran DeepSORT using the YOLOv5 backbone with the pre-trained weights made on YOLOv5l’s 100-epoch run. This resulted in the values shown in Figure 6.18. All in all, I had 13 tracks premade for this (as one of the goalkeepers is absent from the video), and 11 out of 13 tracks were partially found. The frequent switches seem to prevent the accumulation of full tracks. This means that DeepSORT tends to miss the players just long enough to forget their tracks and then find them again. The two players that were always missed were the goalkeeper and the swimming nearest to the bottom of the frame (closest to the camera). Missing the goalkeeper might be because of the lack of data concerning them (which also caused the problem of ball tracking with YOLO), as they are more distinguishable than the players in blue or white hats.



Figure 6.19: A frame from DeepSORT’s output with YOLOv5l backbone, pre-trained for 100 epochs

The constant switches between IDs of instances cause a low MOTA value, even though only two tracks are missing as a whole. On the other hand, regarding the results frame by frame, we notice high scores concerning detection accuracy. Due to this, the MOTP of the tracking is high.

Chapter 7

Conclusions

7.1 Summary

The code belonging to my implementations is available at my GitHub repository [16] or the other repositories I cited there.

My task was to create an image processing system for water polo match analysis by segmenting and tracking the most important features of a game (the court, players, and the ball). I have tried different implementations for all of the task's components.

I tried two approaches for court segmentation; template matching and finding lines at given angles by Hough segmentation. The latter turned out to be more effective in finding the horizontal border in 100% and the vertical in 72% of the cases.

In the case of ball segmentation, I labeled the dataset with bounding boxes. I implemented possible solutions using Kmeans clustering and circular Hough filter simultaneously (both of which are methods of basic image processing), with the help of bounding box regression networks (a custom sequential network and YOLOv5), and using U-net. Even though U-net is a network traditionally designed for segmenting images (thus, conversion of the bounding boxes to 2D masks seems inaccurate), this yielded the best results from all of the methods tried. This might be due to the insufficient amount of data in the case of YOLOv5. I computed the ball's trajectory by calculating the midpoint of the most extensive mass on U-net's output segmentation. Applying the Kálmán filter on that named point had no apparent extra effect.

In the case of player detection, there were two main types of approaches that I tried. One of these was using pose estimation with AlphaPose. I faced difficulties using this method. Labels to such networks include every joint of the instances, ordered. I found no such dataset containing people in aquatic environments, so I used a network pre-trained on a

general instance detection dataset’s (MS COCO) subset containing people. As this contained little to no pictures in my special context, AlphaPose failed on my data. The other kind of approach that I implemented was detection by bounding box regression. For this, I used Mask R-CNN and YOLOv5. Even though Mask R-CNN had higher confidence in its output bounding boxes, its precision and recall scores were worse than YOLOv5’s. All in all, YOLOv5 had the better results for this task. To sum up, for player detection, my go-to solution is using YOLOv5. Tracking was implemented by passing this network’s outputs to a SORT algorithm.

7.2 Future plans

The solutions proposed in this paper provide solutions for locating the most important elements of a water polo game. As everything, these can also be perfected, there is always room for improvement. It would be crucial to enhance the performance of the object-wise template matching at court segmentation as with the help of it, possibly by the introduction of more fixed points (preferably mostly visible points outside of the horizontal border), it is possible to parse all the data to a 2D surface with an affine transformation. This, of course, needs additional information about the camera’s degree of distortion. In the future, my other plan is to coordinate the results of the moving objects’ trackers with the court segmentation tool, perhaps by projecting all the substracted information to the 2D surface mentioned before (as seen in the study of extracting data about basketball players [19]).

Another possible use of this after computing such a system is to identify the maneuvers between the players. With a system capable of that, the analysis of water polo games would be easier, and this system would help coaches and clubs in their strategic decision-making.

The next step after that sounds a bit far-fetched, but in theory, such a system might enable the prediction of the players’ decisions or the proposal of the best strategies to use in any given situation. This, however, is a very long-term plan; some would say it’s only a dream at this stage.

Bibliography

- [1] Janus Atienza. *Why is linux popular for machine learning? top distributions to use*. Sept. 2022. URL: <https://www.unixmen.com/why-is-linux-popular-for-machine-learning-top-distributions-to-use/>.
- [2] D. Benarab et al. “Optimized swimmer tracking system by a dynamic fusion of correlation and color histogram techniques”. In: *Optics Communications* 356 (Dec. 2015), pp. 256–268. DOI: 10.1016/j.optcom.2015.07.056. URL: <https://doi.org/10.1016/j.optcom.2015.07.056>.
- [3] Benjamin Bischke et al. *Mediaeval benchmarking initiative for multimedia evaluation*. 2019. URL: <http://www.multimediaeval.org/mediaeval2019/multimediasatellite/>.
- [4] Bodhisattwa Chakraborty and Sukadev Meher. “A trajectory-based ball detection and tracking system with applications to shooting angle and velocity estimation in basketball videos”. In: *2013 Annual IEEE India Conference (INDICON)*. Mumbai, India: IEEE, Dec. 2013.
- [5] Nameirakpam Dhanachandra, Khumanthem Manglem, and Yambem Jina Chanu. “Image segmentation using K -means clustering algorithm and subtractive clustering algorithm”. en. In: *Procedia Comput. Sci.* 54 (2015), pp. 764–771.
- [6] Issam El Naqa and Martin J Murphy. “What is machine learning?” In: *Machine Learning in Radiation Oncology*. Cham: Springer International Publishing, 2015, pp. 3–11.
- [7] Hao-Shu Fang et al. “RMPE: Regional Multi-person Pose Estimation”. In: *ICCV*. 2017.
- [8] Rohith Gandhi. *Support Vector Machine — introduction to machine learning algorithms*. URL: <https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47>.

- [9] Rocío Alvarez-Cedrón García-Zarandieta. “Implementación de un modelo de detección y seguimiento de jugadores de waterpolo para el análisis de modelos de juego”. Madrid, 2020. URL: <https://oa.upm.es/62753/>.
- [10] Ross Girshick et al. “Rich feature hierarchies for accurate object detection and semantic segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.
- [11] Robert M Haralick and Linda G Shapiro. “Image segmentation techniques”. en. In: *Comput. Vis. Graph. Image Process.* 29.1 (Jan. 1985), pp. 100–132.
- [12] Nazanin Sadat Hashemi et al. “Template Matching Advances and Applications in Image Analysis”. In: *CoRR* abs/1610.07231 (2016). arXiv: 1610.07231. URL: <http://arxiv.org/abs/1610.07231>.
- [13] Kaiming He et al. “Mask R-CNN”. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)* (Oct. 2017).
- [14] Oliver Holub and Sérgio T Ferreira. “Quantitative histogram analysis of images”. en. In: *Comput. Phys. Commun.* 175.9 (Nov. 2006), pp. 620–623.
- [15] Mladen Hraste, Milivoj Bebić, and Ratko Rudić. *Where is today’s Water Polo Heading? An Analysis of the Stages of Development of the Game of Water Polo*. Nov. 2012. URL: <https://hrcak.srce.hr/file/153571>.
- [16] Csenge Hubay. *Movement detection, Github repository*. URL: https://github.com/hubaycsenge/Movement_detection/.
- [17] Ren Jie Tan. *Breaking down mean average precision (MAP)*. Mar. 2019. URL: <https://towardsdatascience.com/breaking-down-mean-average-precision-map-ae462f623a52>.
- [18] Daniel Johnson. *PYTORCH tutorial: Regression, image classification example*. Apr. 2022. URL: <https://www.guru99.com/pytorch-tutorial.html>.
- [19] Neil Johnson. “Extracting Player Tracking Data from Video Using Non- Stationary Cameras and a Combination of Computer Vision Techniques”. In.
- [20] *Kdenlive- Video editing freedom*. Oct. 2022. URL: <https://kdenlive.org/en/>.
- [21] V F Leavers. “Which Hough transform?” en. In: *CVGIP Image Underst.* 58.2 (Sept. 1993), pp. 250–264.

- [22] Peter Leibedzinski. *A single number metric for evaluating object detection models*. June 2021. URL: <https://towardsdatascience.com/a-single-number-metric-for-evaluating-object-detection-models-c97f4a98616d>.
- [23] Chao Liu et al. “Adaptive smooth L1 loss: A better way to regress scene texts with extreme aspect ratios”. In: *2021 IEEE Symposium on Computers and Communications (ISCC)*. Athens, Greece: IEEE, Sept. 2021.
- [24] Gunand Mayanglambam. *Deep learning optimizers*. Nov. 2020. URL: <https://towardsdatascience.com/deep-learning-optimizers-436171c9e23f>.
- [25] Warren Mcculloch and Walter Pitts. “A Logical Calculus of Ideas Immanent in Nervous Activity”. In: *Bulletin of Mathematical Biophysics* 5 (1943), pp. 127–147.
- [26] MVIG-SJTU. *GitHub - MVIG-SJTU/AlphaPose: Real-Time and Accurate Full-Body Multi-Person Pose EstimationTracking System* — *github.com*. [Accessed 06-Nov-2022]. 2018. URL: <https://github.com/MVIG-SJTU/AlphaPose>.
- [27] Ravindra Parmar. *Common loss functions in machine learning*. Sept. 2018. URL: <https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23>.
- [28] Anindya Apriliyanti Pravitasari et al. “On comparing optimizer of UNet-VGG16 architecture for brain tumor image segmentation”. In: *Brain Tumor MRI Image Segmentation Using Deep Learning Techniques*. Elsevier, 2022, pp. 197–215. DOI: 10.1016/b978-0-323-91171-9.00004-1. URL: <https://doi.org/10.1016/b978-0-323-91171-9.00004-1>.
- [29] *Programming Guide :: CUDA Toolkit Documentation*. URL: <https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html>.
- [30] *Project jupyter*. URL: <https://jupyter.org/>.
- [31] Joseph Redmon et al. “You Only Look Once: Unified, Real-Time Object Detection”. In: *CoRR* abs/1506.02640 (2015). arXiv: 1506.02640. URL: <http://arxiv.org/abs/1506.02640>.
- [32] Rishita32. *DeepSORT implementation repository*. Oct. 2021. URL: https://github.com/Rishita32/Yolov5_Deepsort_KalmanFilter.
- [33] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-Net: Convolutional Networks for Biomedical Image Segmentation”. In: (May 2015). arXiv: 1505.04597 [cs.CV].

- [34] Johannes Schmidt. *Creating and training a U-net model with Pytorch for 2D and 3D semantic segmentation: Model Building...* Apr. 2021. URL: <https://towardsdatascience.com/creating-and-training-a-u-net-model-with-pytorch-for-2d-3d-semantic-segmentation-model-building-6ab09d6a0862>.
- [35] Sagar Sharma. *Activation functions in neural networks*. July 2021. URL: <https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>.
- [36] Han Shen. *A python implementation of multiple object tracking(mot) evaluation toolkit(2d)*. Aug. 2019. URL: https://github.com/shenh10/mot_evaluation.
- [37] DoCong Thuan. “Evolution of YOLO Algorithm and YOLOv5: The State-of-the-art Object Detection Algorithm”. In: 2021.
- [38] J. R. R. Uijlings et al. “Selective Search for Object Recognition”. In: *International Journal of Computer Vision* 104.2 (Apr. 2013), pp. 154–171. DOI: 10.1007/s11263-013-0620-5. URL: <https://doi.org/10.1007/s11263-013-0620-5>.
- [39] Ultralytics. *GitHub - ultralytics/yolov5: YOLOv5 in PyTorch > ONNX > CoreML > TFLite*. 2022. URL: <https://github.com/ultralytics/yolov5>.
- [40] Qi Wang et al. “A comprehensive survey of loss functions in machine learning”. In: *Ann. Data Sci.* 9.2 (Apr. 2022), pp. 187–212.
- [41] Sun-Chong Wang. “Artificial Neural Network”. In: *Interdisciplinary Computing in Java Programming*. Boston, MA: Springer US, 2003, pp. 81–100.
- [42] Greg Welch, Gary Bishop, et al. “An introduction to the Kalman filter”. In: (1995).
- [43] *What is a container?- Docker container documentation*. Oct. 2022. URL: <https://www.docker.com/resources/what-container/>.
- [44] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. *Simple Online and Realtime Tracking with a Deep Association Metric*. 2017. DOI: 10.48550/ARXIV.1703.07402. URL: <https://arxiv.org/abs/1703.07402>.