Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Scott Gullaksen & Eirik Lie Morken

Automatic Player Tracking in Single-Camera Soccer Videos

Master's thesis in Computer Science
Supervisor: Frank Lindseth
June 2021

NTNU
Norwegian University of Science and Technology

Scott Gullaksen & Eirik Lie Morken

Automatic Player Tracking in Single-Camera Soccer Videos

Master’s thesis in Computer Science
Supervisor: Frank Lindseth
June 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Abstract

AI and machine learning have taken roots in today's soccer world as an automated means to deliver sports analytics. The analytical data help soccer stakeholders gain a competitive advantage in ways previously thought not to be possible. Particularly, automated tracking of soccer players has proved to play an essential role in providing such data. It can allow real-time assessment of team formations and tactics of high value for sports reporters and opponent teams alike. Furthermore, individual players can be assessed based on speed, acceleration, and total distance traveled, which is valuable in athlete performance analysis. However, the technology is primarily reserved for professional soccer organizations and individuals, and the entry cost is high. The technology requires expensive and specialized equipment, such as multi-camera setups and GPS trackers. In contrast, this thesis applies recent computer vision methods on *single-camera soccer footage*. The work has the potential to contribute to the development of low-cost software that provides similar analytical data that premium products offer.

To be more precise, this thesis is concerned with determining whether state-of-the-art computer vision algorithms can automatically detect and track players over time from soccer footage captured with one camera. The results are promising and even reveals that real-time processing speeds are within reach. It is also demonstrated how the processed footage can be transformed into full-fledged analytical data that may be directly used for soccer analytics purposes. At the heart of this thesis is a traditional machine learning methodology. To that end, we propose a semi-automatic labeling approach that significantly reduces workloads for machine learning-based multiple object tracking (MOT) tasks. We also propose a method for obtaining MOT metrics during the optimization of MOT system components. The method gives deep insight into how component optimization affects system-wide MOT performance.

Sammendrag

AI og maskinlæring har befestet seg i dagens fotballverden som en ressurs for å automatisere sportsanalyse. Analytiske data hjelper fotballens involverte med å oppnå et konkurransefortrinn på måter man tidligere ikke trodde var mulig. Særlig automatisk tracking av fotballspillere har vist seg å være essensielt for å kunne levere slike data. Tracking kan tillate sanntidsvurdering av lagformasjoner og taktikker, noe som er verdifullt både for sportsreportere og motstanderlag. Videre kan individuelle spillere vurderes ut fra hastighet, akselerasjon og total tilbakelagt strekning, noe som er særdeles nyttig i en analyse av utøverens individuelle ferdigheter. Dessverre er teknologien primært forbeholdt profesjonelle fotballorganisasjoner, og produktene er dyre. Teknologien krever kostbart og komplisert utstyr, for eksempel multikameraoppsett og GPS-trackere. Som en motsetning til dette, bruker denne oppgaven moderne datamaskinsynmetoder på *fotballopptak filmet med ett enkelt kamera*. Avhandlingen har potensiale til å bidra i utviklingen av rimelig programvare som leverer tilnærmede analytiske data som det premiumprodukter tilbyr.

Vi ønsker å undersøke om toppmoderne datasynsalgoritmer automatisk kan oppdage og følge spillere over tid fra fotballopptak filmet med ett enkelt kamera. Resultatene er lovende og viser til og med at sanntids-prosseseringshastigheter er mulig. Arbeidet demonstrerer også hvordan resultatene fra disse metodene kan videreutvikles til fullverdige analytiske data, og som kan brukes direkte til fotballanalyse. Kjernen av dette arbeidet består av en tradisjonell maskinlæringsmetodikk. I tråd med dette foreslår vi en ny semi-automatisk annoteringsmetode som markant reduserer arbeidsmengden for maskinlæringsbaserte tilnærminger for flerobjekts tracking. Videre foreslår vi en metode for å oppnå metrikker for flerobjekts tracking under optimalisering av individuelle systemkomponenter. Metodene viser hvordan optimalisering av forskjellige deteksjon - og trackingkomponenter påvirker den totale ytelsen av systemet.

Preface

This thesis was written during the spring of 2021 as a final deliverable to obtain a master's degree in Computer Science at the Norwegian University of Science and Technology (NTNU).

We would like to thank our supervisor Frank Lindseth for his valuable insights and his continued support throughout this endeavor. We would also like to thank the Department of Computer Science (IDI) for providing essential computing resources placed at our disposal.

Eirik Lie Morken & Scott Gullaksen
June 14th, 2021

Table of Contents

List of Figures

List of Tables

1 Introduction

For the past decade, a tremendous research effort has been dedicated to the the complex field of computer vision (CV). The payoff for these endeavors has been outstanding: Cars are provided with visual systems for automated driving [1], medical images can be scanned for automatic detection of tumors in patients [2], and many more exciting applications exist. With its rise, computer vision research has begun to explore interesting applications for sports. In fact, with the field having dedicated an annual conference [3], research is blooming. Some of the research investigates how computer vision methods can be applied to a video feed from various sporting events to extract useful information such as player pose, position, and ball trajectory. This thesis is concerned with similar issues. In particular, we investigate computer vision techniques for the automatic extraction of player tracking data from single-camera soccer videos.

1.1 Motivation

The use of computer vision to provide insights from sporting events is a special case of a phenomenon popularly known as *sports analytics* [4]. Sports analytics is generally regarded as data that can be used to inform teams and individuals to gain a competitive advantage. Raw data are collected and analyzed to provide statistics to coaches, players, and other stakeholders to make informed decisions, both before and during sporting events. For example, coaches and players can use detailed player fitness data and statistics to discover strengths and weaknesses that would otherwise be hard to identify with the naked eye [5, 6]. It can also give commercial organizations valuable insights to help improve ticket sales, merchandise or even optimizing fan engagement [7]. In other words, sports analytics can help businesses with decision-making to ensure growth and higher profitability. Sports analytics has been incorporated by some of the world's biggest sports teams and has been vital to their success [8].

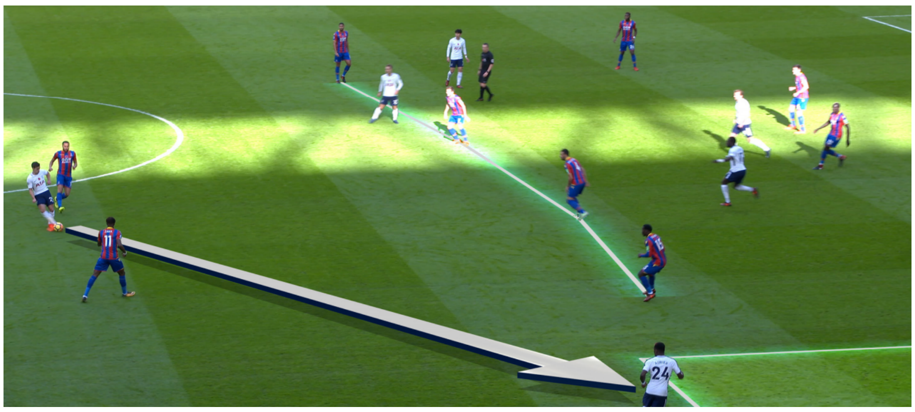


Figure 1: Computer vision techniques are often a prerequisite for enabling overlay graphics. Adapted from [9].

In recent years, sports analytics data is achieved automatically by innovative applications of computer vision techniques. For instance, when broadcasting sports events over television, computer vision is often used for automatic calibration of the cameras using positions-on-the-scene features [9, 10]. It is also becoming common to jointly and automatically detect the players on the calibrated image. Automatic calibration and detection is a prerequisite for what is shown in Figure 1. The figure shows what is so familiar to fans worldwide when watching their favorite team play a broadcasted game on television: overlay graphics. Such graphics has become the gold standard in broadcasting sports events as they provide an intuitive visual understanding of critical moments and tactics. It makes it easier for reporters to analyze and communicate, and creates a more engaging experience for viewers.

As another example, nearly every team in the NBA basketball league has, for a long time, used a computer vision-based camera system, STATS SportsVU [11], to track player and ball trajectories [12, 10]. Such details provide information on how players move, their speed, change of directions, how well they shoot from different positions, and how well they guard. The data, when correctly interpreted, can help stakeholders assess team performance and help make impactful decisions based on data previously not attainable. For example, SportsVU helped NBA teams identify that 3-point shots are one of the most effective ways to score points. The number of 3-point shots per game has dramatically increased since. Consequently, NBA teams were forced to fundamentally restructure how they strategize and play their games.

Unsurprisingly, not only basketball has made use of the recent advancements in CV-based tracking. European soccer has made itself well acquainted with the technology. SportsVU is also being used by some of the world's biggest soccer leagues, and individual teams [11]. These include Italy's Serie A and individual teams such as FC Bayern Munchen and Paris Saint-German F.C. (PSG). For soccer, the SportsVU system usually operates with 3 clusters of HD cameras [10]. Numerous other tracking systems have been adopted. The Spanish top soccer league, LaLiga, supplies a video analysis platform that is being used by a total of 42 clubs [13]. The platform can deliver player speeds, distance covered, and ball movements due to its tracking system, ChyronHego's TRACAB [14]. The usual configuration for the system consists of two clusters of 3 HD cameras [10], but can also use up to 16 cameras per stadium [13]. TRACAB is commonly recognized as the most accurate optical tracking system for soccer and is subsequently one of the few that is FIFA certified [15]. TRACAB is the official tracking data provider for the top soccer leagues in Germany, including the Bundesliga [16]. Some individual English Premier League teams also use TRACAB [17]. The system has also been used in some of the world's largest soccer events, including the UEFA Champions League and the FIFA World Cup [10]. For the English Premier League, Second Spectrum is the official optical athlete tracking system and it has been since the 19/20 season [18]. The system can be configured with up to 10 cameras per stadium [19].

It is evident from the examples above that computer vision has yielded valuable tools that contribute to the field of sports analytics. The examples also reveal the critical driving forces for utilizing sports analytics. Soccer, in particular, has made considerable use of autonomous tracking technologies, which implicitly highlights its usefulness. Unfortunately, the examples also reveal that existing applications are expensive products targeting professional-level teams, athletes, and organizations. They require specialized setups of multiple high-resolution cameras, player sensors, calibration software, and more customized hardware. In amateur-level sports, these premium methods are primarily regarded as unavailable. To be precise, there exist few low-cost solutions that attempt to provide the same analytical data that professional soccer clubs have access to [20]. To emphasize the demand, the visual computing group at IDI, NTNU, was approached by soccer club Ranheim TF to investigate the possibility of applying computer vision on their soccer matches filmed with a single camera. The untapped market suggests that it is worthwhile to investigate computer vision methods to enable sports analytics from inexpensive camera setups.

1.2 Goals and Research Questions

This thesis aims to assess various computer vision methods on soccer videos that use single-camera setups and their implications for delivering sports analytic data.

We aim to determine to what extent various recent methods can perform automatic detection and tracking of soccer players. Thomas et al. [10] notes that *fully automated tracking of soccer players remains an open challenge* and that tracking is at the core of providing sports analytics. While previous work has utilized complicated setups such as specialized cameras [21], GPS microchips [22], or multiple cameras setups [11], the methods in this thesis are intended to work with *soccer matches filmed with only a single camera.* Also, the video data in this work is of lower quality than what is usually required by current applications. It will serve as a testimony to the applicability of such methods on data that is realistic to obtain for stakeholders in all levels of sports. Furthermore, we aim to demonstrate if such a setup is sufficient for providing analytical data that may be used for sports analytics purposes.

Below are some specific research questions (RQs) we will try to address:

RQ1 - Is automatic player *detection* possible from a single-camera soccer video?

RQ2 - Is automatic player *tracking* possible from a single-camera soccer video?

RQ3 - Is automatic player tracking from a single-camera soccer video achievable in real-time?

RQ4 - Do the applied methods display usability in sport analytics applications?

1.3 Methodology

In order to meet the goals of this thesis, a mix of novel supervised machine learning methods [23], classical AI techniques [24], and traditional CV approaches [25] will be used. For the player detection and tracking tasks, various deep learning-based object detection algorithms are adopted due to their extraordinary success in recent years [26, 27, 28]. Multiple Object Tracking (MOT) has also been given considerable attention recently, which is directly due to the success in object detection. The deep object detectors performance seem to be highly transferable to the MOT domain, which makes sense as object detection is a necessary pre-processing step for fully automated tracking [29]. Subsequently, a mix of traditional and novel deep learning-based extensions has been added to the object detectors in order to transform them into full-fledged MOT systems [30, 31, 32]. This thesis takes a similar approach, and the object detectors employed will indeed be extended with MOT capabilities for the detection and tracking of soccer players. Particularly, three different online MOT systems will be assessed. Each of these fundamentally vary in how object detections and existing object trajectories are associated. The assessment of each will help to determine if one is more suited for soccer player tracking than the others.

The three object detection algorithms employed belong to the class of supervised machine learners. Therefore, it entails the need for labeled data. As mentioned, the data for assessment will be videos of soccer matches filmed with a single camera. Unfortunately, labeled soccer videos are tough to come by; especially MOT labeled ones. To that end, a semi-automatic labeling approach is proposed to reduce labeling efforts for supervised MOT tasks significantly. After a sufficient data amount is labeled, system component optimization will be performed. During optimization, we monitor for overfitting on an isolated validation set, such that the best performing models can be selected. This is achieved by monitoring MOT metrics during the optimization of components. Components can then be selected based on performance in system tracking instead of, for instance, object detection performance. To our knowledge, no work to date detail a similar approach. After the optimization is complete, each MOT system is assessed with both established detection and tracking metrics. Lastly, we will further process the outputs from the MOT systems and show its significance in the soccer domain. This will further rigorously assess the methods for applicability in sports analytics applications. The results will form the basis for comparing the different systems and help address our research questions.

1.4 Contributions

All work presented in this thesis will contribute to the field of applying computer vision to enable sports analytics from soccer footage. Current commercial products facilitate organizations and individuals almost exclusively at a professional level, and entry costs are high. The work can aid the development of products that have the same functionality but is obtainable at all levels of professionalism. The specifics are:

- A description of a machine learning methodology for multiple player tracking in the soccer domain.
- A comparison of three MOT systems capable of detecting and tracking soccer players from single-camera footage.
- Implementation details and assessment of a proposed semi-automatic labeling approach

- A proposed practice for MOT metric evaluation of the overall tracking system during optimisation of separate MOT system components.
- A demonstration on how to enable sports analytics features from the tracking data

1.5 Thesis Outline

A brief outline of the succeeding sections follows.

2. Background & Related Works
This section will explain essential concepts that are necessary to understand before reading the work presented here. It will serve as an introductory text to concepts that, in reality, are much more complex. It will, however, cover the essentials, providing the minimal foundation needed to understand the sections that follow. The section also provides a list of work that is similar to what will be presented here.

3. Methodology
The section will explain in detail the practical work conducted in this thesis. It includes tasks such as dataset creation, architecture selection and overview, optimization, and evaluation. Methods for creating sports analytics data are also provided. Details on the semi-automatic labeling approach and the MOT metric validation method can also be found in this section.

4. Experiments & Results
Details from architecture experiments are listed. The section provides quantitative and qualitative results from the experiments in the form of tables and labeled images, respectively. Following each experiment result, a discussion concerning the experiment in isolation is provided.

5. Comparative Discussion
The section provides a comparative discussion of the results obtained from the experiments. Addressing the research questions introduced in Section 1.2 will be the basis for comparison.

6. Conclusion & Future Work
The section addresses how well the research questions have been met and to what extent the goal was achieved. It also provides a detailed list of further work to be done in the field. It includes addressing the downsides of the work presented and future endeavors we failed to attend due to time restrictions.

2 Background & Related Works

In this section, methods, architectures, and technologies that this thesis builds upon will be presented. The following content represents a bare minimum of what should be understood before reading the subsequent sections of this thesis. The section will start with the fundamentals, explaining the basic building blocks that are artificial neural networks. The section then gradually moves onto more complex themes, such as object detection architectures and multiple object tracking methods. A list of tools used in the practical work of this thesis is also provided. Finally, the section lists various other research efforts concerned with computer vision in the soccer domain.

2.1 Artificial Neural Networks

Artificial neural networks (ANN) is a general class of supervised machine learning models that is able to create arbitrary complex functions that map inputs to output targets, or in other words, map problem instances to correct target values [33].

To summarize Nielsen [34], different artificial neural networks are created by combining the basic units, *neurons*, in different ways. Neurons make up the essential building blocks of these networks. Two types of networks are depicted in Figure 2 and 3. The neuron accepts multiple values $x_i \in [0, 1]$ often represented as a single input vector $\vec{x}$. The "body" of the neuron is composed of a set of tuneable weights, $w_i \in \mathbb{R}$, which is also often represented as a vector $\vec{w}$. A neuron functions by performing the linear transformation $\vec{x} \cdot \vec{w}$ to produce the real-valued scalar a. The scalar is then fed to an activation function f, such as a sigmoid, which becomes the final output of the neuron, $\hat{y}$ [34]. The output is then a number ranging from 0 to 1 and can function as a probability estimate, or commonly known as a *confidence score.*

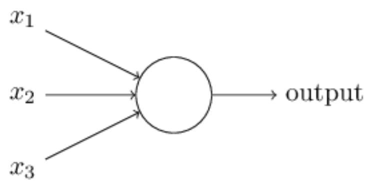


Figure 2: The basic building block of artificial neural networks: The neuron. It accepts multiple normalised scalar values x_i as input and performs a linear transformation to produce the output. Adapted from 34.

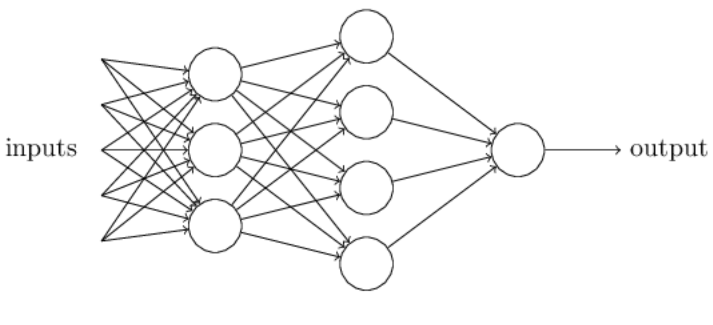


Figure 3: The basic units from figure 2 is combined to create what is known as a fully connected neural network. Vertically aligned neurons and inputs comprise what is known as a layer. All outputs from a layer are connected to every neuron in the preceding layer. Adapted from 34.

Figure 3 shows what is known as a *fully connected* (FC) neural network. It is *multilayered*, which means it is comprised of stacks of layers. A layer consists of a predefined number of unconnected neurons. In Figure 3, these are vertically stacked. However, each neuron in a layer is connected by every neuron in the previous and preceding layer. The first layer is called the input layer, and each neuron is just the elements of the input vector. Layers that are between the input and the output layer, is known as *hidden layers*.

We now know the overall structure of ANNs and how they can produce outputs. However, there is still something to be said of how such structures produce *correct* outputs. Since it is a supervised approach, it needs to *learn* from labeled data, referred to as *examples*. Examples are used by the *backpropagation* algorithm to learn artificial neural networks how to perform a mapping from the training set's inputs to their respective targets. This is a process of optimisation, or more specifically known as *training*. Usually, an additional set of labeled examples, called the *test set*, is isolated from the training procedure. It is used to assess the trained network's ability to classify new, previously unseen examples correctly. In other words: its ability to *generalize* beyond the training set. After all, being able to classify already seen examples correctly is not very impressive, and we are usually more interested in how the machine performs on new problem instances.

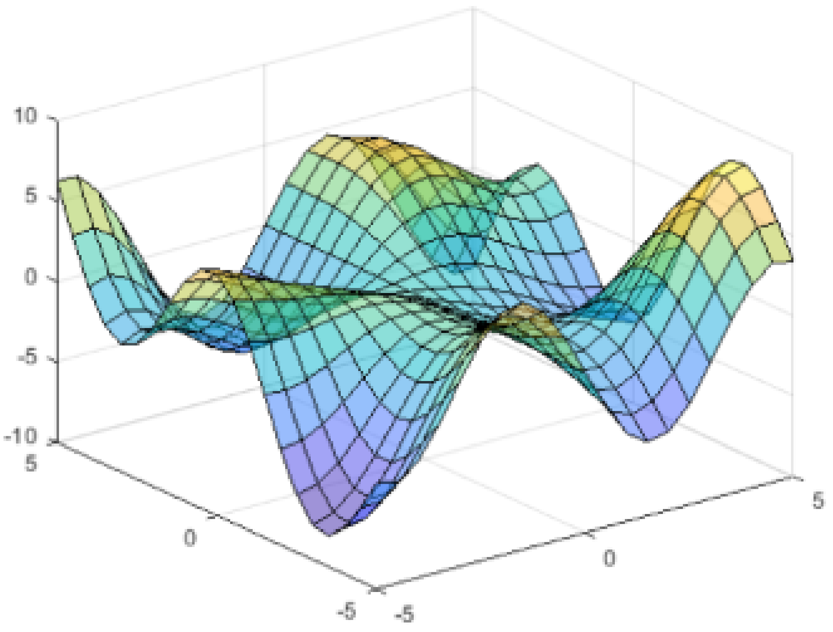


Figure 4: The cost function is a hilly landscape when regarded as a function of weights. In this case, it is a function of two weights, represented by the horizontal plane. Gradient descent works by iteratively taking steps towards the direction of steepest descent, which would eventually lead to a minima.

Norvig [33] nicely states how the backprogation algorithm works. In order to assess the performance of the machine both during and after training, a *cost or loss function*, C($\mathbf{y}$, $\mathbf{x}$, h_w), is defined. The cost function gives a numerical value reflecting the utility of the outcome $h_w(\mathbf{x})$= ŷ when the correct answer is $\mathbf{y}$. The metric is usually calculated with some sort of distance metrics between ŷ and $\mathbf{y}$, and a complexity measure of h_w.

Backpropagation is essentially the *gradient descent algorithm* specialized for artificial neural networks. In gradient descent, a gradient vector for the weights $\mathbf{W}$ is computed with respect to the cost function. For each weight w_i, the gradient vector defines the change needed to increase the cost function most efficiently. The cost function can be viewed as a hilly landscape in a high-dimensional space where the axis consists of the weights w_i and the cost C itself. This is depicted in figure 4. If one were to visualize the gradient vector in that same space, it would be an arrow pointing from the current value of the cost function to the direction of steepest ascent in that hilly landscape. Thus, the minimum value of the cost function can be obtained by repeatedly computing the gradient and adjusting the weights with a small *negative fraction* of the computed gradient. In the hilly landscape, this would be the equivalent of taking small steps towards the direction of steepest *decent*. The problem with artificial neural networks is that while computing the gradient for the output layers is straightforward by the use of partial derivatives, computing them for hidden layers is non-trivial since there are no targets directly available from the training data. Backpropagation solves this by propagating the errors from the output layers backward, making it possible to compute gradients. The backpropagating part is essentially an applied case of the chain rule for derivatives.

Below, a few concepts that will appear frequently in the text is listed.

- *Optimizer* - A specific algorithm that performs gradient decent. Some common ones are the Stochastic Gradient Decent (SGD) [35] and ADAM [36]
- *Epoch* - A complete pass through the examples of training set during optimization
- *Iteration* - One gradient decent-, or optimizer step.
- *Batch* - A subset of the training examples that are used in one optimizer step.

2.2 Convolutional Neural Networks

Generally, one can think of a *Convolutional Neural Network* (CNN) as an ANN with a specialization for detecting and recognizing patterns and making sense of them [37]. This ability to detect patterns is what makes them great at analyzing images. The reason CNNs excel at pattern detection is that they contain hidden layers called convolutional (conv.) layers, in addition to the traditional connected layers that is found in ANNs. In the convolutional layers, there are several *filters*, also known as *kernels*, that detect different kinds of features using the convolution operation. Filters in early layers of networks usually detect low-level geometric features like edges and shapes, whereas filters in later layers are more sophisticated and can be used to detect complex objects such as cars, dogs or people. The term *deep* comes from the fact that many layers, and thus filters, are stacked to create complex *models*.

This makes CNNs translational equivariant, m ANNs can be used for image analysis, but CNNs outperform them because of some properties that arise from the convolutional layers. One of these properties is that conv. layers inherently capture the spatial features of an image, in essence; the location of pixels in relation to each other and the spatial relationship between the features. CNNs are translational invariant, meaning the object's position does not have to be fixed to be detected by the network. Lastly, for an ANN, the number of parameters increases drastically corresponding to an increase in resolution of an image. CNNs counter this by utilizing parameter sharing. This is achieved by sliding the same filter, using shared parameters, over the entire image, creating a *feature map*. The parameter sharing property is quite logical; for instance, a filter that detects horizontal edges can be applied across the entire image, with no need to relearn the filter parameters for different parts of the image.

In addition to the convolutional layers, most CNNs contain one or more layers called *pooling layers*. Like the convolution layer, the pooling layer has a filter that glides across the input and performs a function. The most commonly used pooling functions are max pooling and average pooling. Max pooling will, as the name suggests, return the max value for all the values inside the filter, whereas the average pooling returns the average of the values. The purpose of max pooling is feature reduction by aggregating the features into more compact representations. This process makes the network more computationally efficient and contributes to the translational invariance. Furthermore, the pooling layers are parameter-free, performing the same functionality independent of the input, and requiring no training.

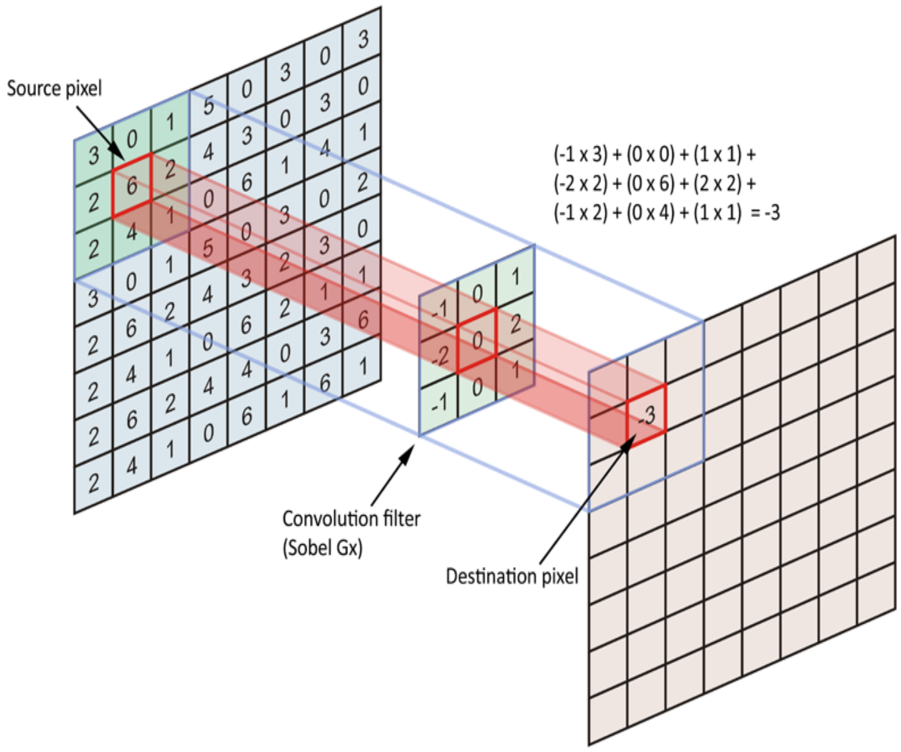

Source pixel
(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3
Convolution filter
(Sobel Gx)
Destination pixel

Figure 5: A convolutional filter sliding over the input and returning output to the next layer. Adapted from 38.

In figure 5 it is illustrated how the convolutional filter is applied to the input. Each time the filter slides across the input, it calculates the dot product between each pixel and the corresponding value in the filter. The products are summed up and returned as the output of the new layer, as shown by the calculations in the top right of the illustration.

2.3 Deformable Convolution

Traditional CNNs have an inherent problem concerning how to adapt to geometric variations in objects. The geometric variations include; deformation, differences in pose, and considerable dissimilarity in scale. A traditional approach to this issue has been to artificially create more variations in the dataset, by using data augmentation on the existing samples. Another approach has been to use translation-invariant features and algorithms. Both of these methods have drawbacks. By assuming fixed and known geometric transformations, the methods would not generalize on new and unknown transformations. Furthermore, handcrafting invariant features and algorithms can be both time-consuming and exceedingly challenging depending on the complexity of the transformation, and that is when the transformations are known. To solve the issue, deformable convolutions was developed [39].

In contrast to the rigid sampling grid used in normal convolutions, deformable convolutions allow for a more free-form sampling grid that adapts to local, dense transformations dynamically, and it does so end-to-end with no additional training supervision required. An illustration of this can be shown in figure 6. Deformable convolutions can readily replace regular convolutions in CNNs and improve the adaptability of the network, yielding higher accuracy on detection tasks.

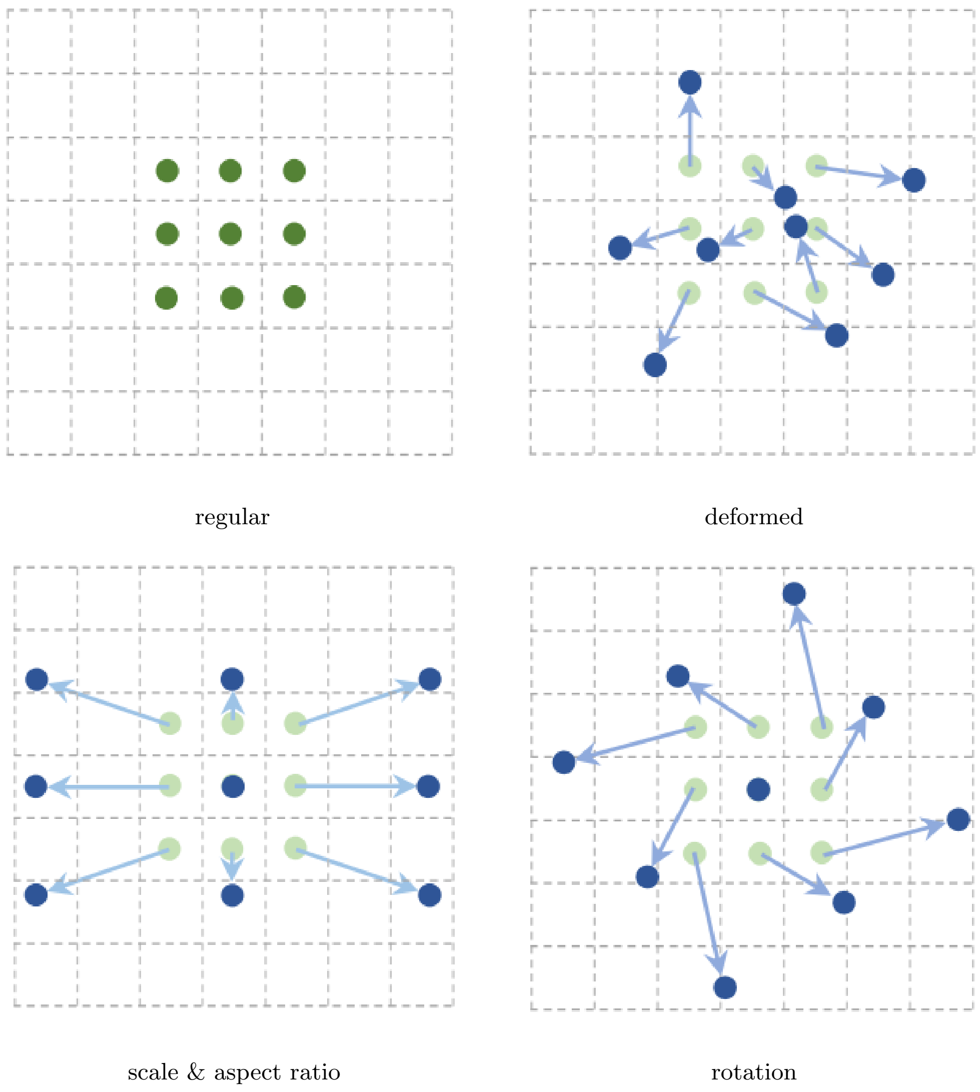
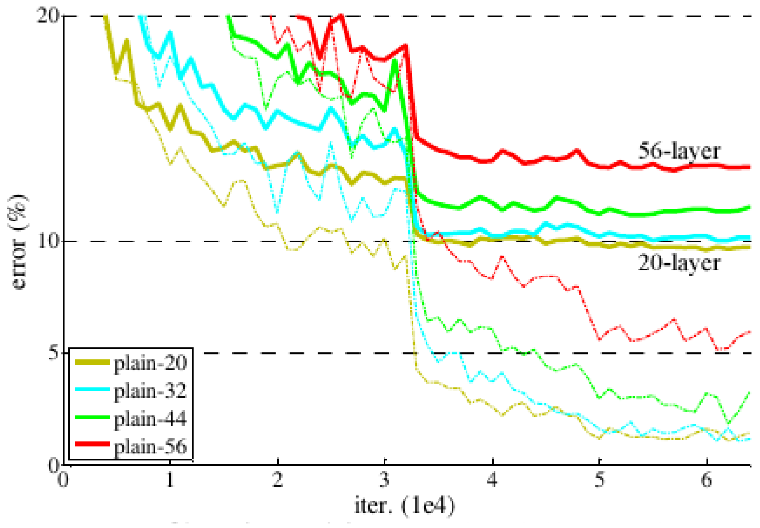


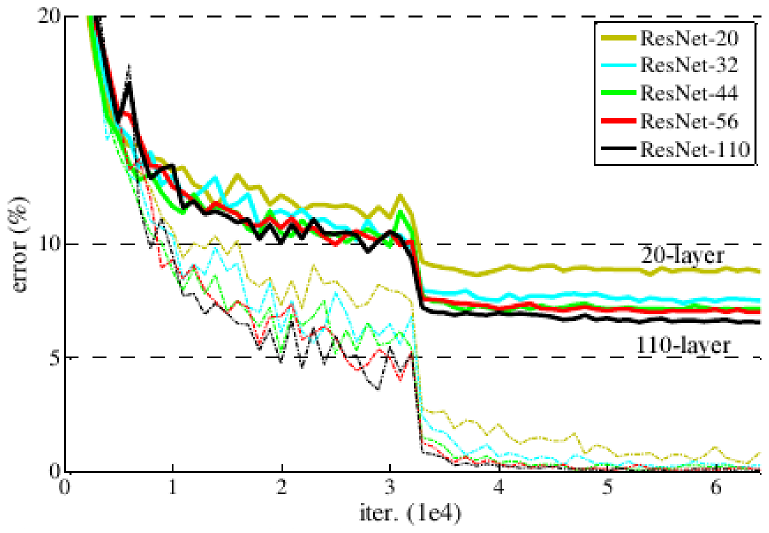
Figure 6: Illustrations of the adaptable sampling grid of a deformable convolution. Adapted from 39.

2.4 ResNet

Consider the hierarchical nature of deep learning, where one obtains low, mid, and high-level features by iterating through each layer of a model. A natural next step in deep learning development would be to add more layers to increase the CNN's performance. A deeper model should conceptually be able to detect features of higher complexity and yield better results. In practice, however, this was not the case. Before ResNet [40], models rarely went beyond 30 layers in depth. Figure 7 shows that by simply adding convolutional layers beyond a certain number causes the training error to increase.



Plain network



Resnet

Figure 7: The two illustrations show how the training error, depicted by the dotted line, and the test error, depicted by the bold line, is affected by adding more layers to a model. The top illustration shows a plain network, where the error increases when using more than around 30 layers, and the bottom illustration shows the ResNet network, where the model with 110-layers has the lowest error. Adapted from 40.

The creators of ResNet reasoned that deeper models should be able copy the shallow models performance by using identity mappings, also known as *skip connections*. This is illustrated in Figure 10, which shows how the skip connection is added to create the *residual block* used in ResNet. Using residual blocks in the construction of the network allowed ResNet to have far more layers, without being punished by the *vanishing gradient problem* and the *degradation problem* [40], both issues that traditionally increase the network error. At the time of release the architecture became state-of-the-art for classification tasks and is frequently used today as a *backbone* network for numerous object detectors. It simply means that it was used to extract deep, highly general features to inform higher level tasks such as object classification.

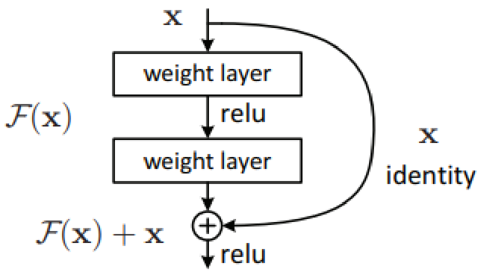

x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 8: The illustration shows a skip-connection, where the information from the first layer can skip over the weights of the next layer by being copied directly by the identity function Adapted from 40.

2.5 Deep Layer Aggregation

Visual recognition in images requires the ability to recognise anything from simple geometric shapes to complex objects. In the deeper layers of a CNN, low-level features extracted by the earlier layers get replaced by complex features of high semantic value. This is not ideal, because the features from earlier layers provide more fine-grained spatial information than the later ones [41]. The skip-connection was, in part, created to mitigate this issue. It allows the flow of information from shallow to deep layers without altering the information itself. Deep Layer Aggregation (DLA), introduced by Yu et al. [41], uses the same concept of carrying features from shallow layers through the network, although with a more refined approach than skip-connections. Where skip-connections only use simple operators to combine layers, DLA merges features iteratively and hierarchically for each layer to create a better and more sophisticated representation of the information extracted from the previous layers. The method is similar to other feature aggregation approaches, such as the *Feature Pyramid Network* (FPN) from Lin et al. [42]. An illustration of DLA's process is shown in figure 9.

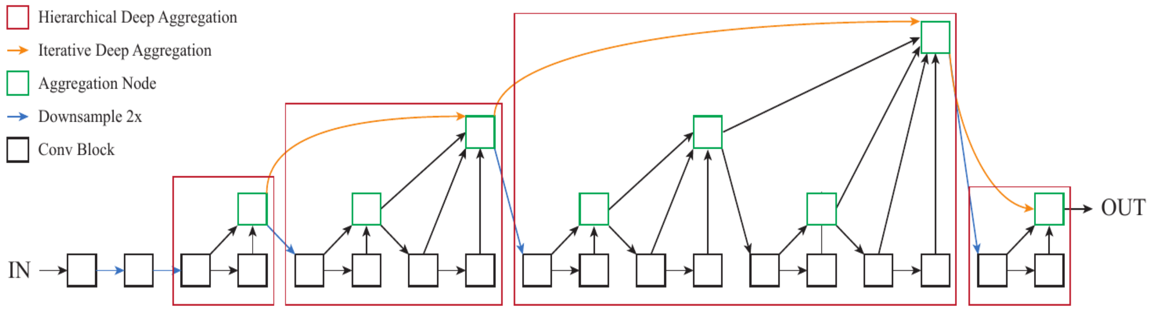


Figure 9: DLA merges features iteratively and hierarchically to better the extraction of features from the input (adapted from 41).

In particular, when a network is intended to solve a variety of computer vision tasks, it needs features that facilitate solving each of them. For example, object detection requires deep and complex features to predict object class and position, whereas a re-identification task is more dependent on low-level appearance to distinguish the individual objects [32]. Thus, to facilitate more tasks in the same network, the multi-layer feature fusion performed by DLA is a powerful tool. It allows the different tasks to extract the features they need when necessary.

2.6 Faster R-CNN

Introduced in 2015, Faster R-CNN [26] is an established *two-stage* object detection algorithm. That is, it is an algorithm capable of localizing and classifying multiple objects from an input image. Objects are detected with *bounding boxes* and labels.

It belongs to the family of *Region-Based Convolutional Neural Networks* (R-CNN) and is the third iteration of its kind. R-CNNs generate region proposals, or *Regions of Interests* (RoIs), representing possible areas within an image that may contain objects. Historically, the region proposal was made using an algorithm called *Selective Search* [43]. This method creates RoIs by merging superpixels based on low-level features, using a greedy approach. Selective Search uses a CPU-based implementation, instead of a GPU-based one, and uses more than two seconds per image to finish. For Fast R-CNN (Faster R-CNN's predecessor) it became clear that the region proposal process was the bottleneck. This was evident as Fast R-CNN only used 0.32 seconds for the entire object detection process, excluding time spent during region proposal.

As a consequence, creating a faster way of performing region proposals became the main objective. The solution was to create a separate network to perform the task; the *Region Proposal Network* (RPN). The RPN would be applied first, followed by Fast R-CNN, to make up the entire object detection network. The first step with the RPN is to feed the input image through a backbone network to extract a feature map. As speed is a main concern, the RPN shares the backbone features with Fast R-CNN, thus avoiding extra computations. Followingly, the RPN moves across the feature map in a sliding window-fashion. For every location of the sliding window, it predicts multiple region proposals simultaneously. These proposals are created by placing "anchors" at the center of the window. Anchors are essentially boxes of different size and aspect ratios, representing possible object shapes. After the sliding window has passed over the entire feature map, the region proposals are complete. These RoIs are used as input for Fast R-CNN to perform detection. Fast R-CNN firstly performs RoI-pooling, which essentially consists of partitioning all the RoIs into a fixed number of sub-windows and performing max-pooling on the windows. The outputs of the RoI-layer are forwarded to a fully connected layer, which in turn passes its result to the classification and regression branches for performing the final results.

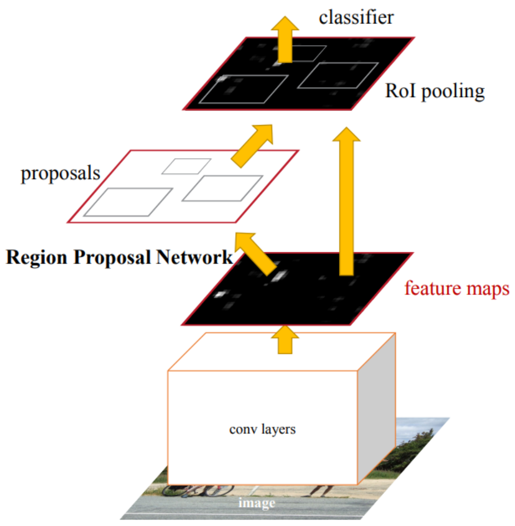


Figure 10: A high-level illustration of the Faster R-CNN architecture. Adapted from 26. The illustration also shows how the feature maps from the convolutional backbone are shared between the RPN and the detection branch.

The addition of the RPN significantly increases test speed and helps the model make a large leap towards real-time object detection. The RPN also improves training speed. With the new addition, the network is end-to-end trainable in contrast to Fast R-CNN, which has an expensive, piece-wise training phase. The training phase of Faster R-CNN consists of iterating between fine-tuning the FPN and subsequently fine-tune the object detection while keeping the region proposals fixed.

2.7 YOLO: You Only Look Once

YOLO is a *single-stage*, real-time object detector first introduced by Redmon et al. [27] in 2016. YOLOv1, YOLO's first iteration, was a major step forward in the real-time detection field at the time of release, both in terms of benchmark results and architecture concept. Previous works in object detection were largely based on repurposing classifiers to perform object detection. For instance, classifiers were used in a sliding window fashion. Another example is the more involved method described in Faster R-CNN, which uses a sliding window on top of a feature map while applying a Region Proposal Network and subsequently performing classification on the proposed regions. Considering the extra steps necessary for the above methods, such as post-processing and eliminating duplicate entries, makes this prior way of performing object detection a complex and arguably slow pipeline. YOLO throws out the pipeline and proposes a single convolutional network, which simultaneously performs classification and bounding box regression. In the years following YOLOv1's release, several new iterations of YOLO have been presented, each pushing the state-of-the-art to new heights. The subsequent sections will present the main aspects of YOLOv1 before presenting some of the improvements from the newer versions.

The general functionality of the model is depicted in Figure 11, which provides an overview of how the network performs detection on an image. The model receives an input image and divides it into a SxS grid. A number of objects, and thus bounding boxes, is predicted for each cell. The cells containing the center of an object should be recognized as an object. The number of boxes per cell, and the number of grid cells per image, are adjustable and could, for example, vary depending on how populated images are. For each of the predicted bounding boxes, the network computes an object confidence score. The object confidence scores are a probability estimate of the box containing an object and how precise the box captures it. Additionally, the grid cells predict conditional class probabilities for all the classes in the applied dataset, yielding a class probability map. Combining all the previously mentioned results outputs the final predictions.

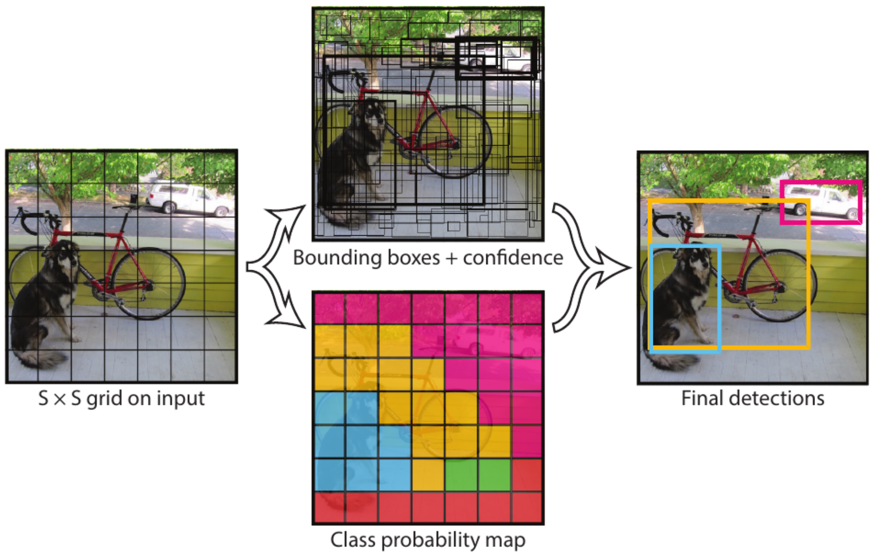


Figure 11: YOLO's SxS-grid and how it computes the object proposals. Adapted from 27.

The model's architecture consists of a single convolutional network. The network is comprised of 24 convolutional layers, followed by two fully connected layers. The network uses the initial convolutional layers for extracting features and the fully connected layers for predicting object coordinates and class probabilities. The architecture was inspired by Szegedy et al. [44]'s GoogLeNet from 2014. A notable distinction from GoogLeNet is that it moved away from inception modules, exchanging them with 1 x 1 dimensionality reduction layers, followed by 3 x 3 convolutional layers. The complete network for YOLOv1 is illustrated in Figure 12. Before training the entire network, the YOLO team first performs an extensive pre-training of the first 20 convolutional layers for approximately one week on ImageNet's [45] 1000 classes. The full architecture is realized by having four convolutional layers and two fully connected layers with randomly initialized weights. They apply a linear activation function for the last layer and a leaky rectified linear activation for the previous layers. Multiple detections can arise due to large objects being covered by multiple grid cells or objects placed near the border of multiple cells. The excess detections are dealt with by filtering out detections corresponding to the same ground truth object, but have lower confidence scores. The approach is commonly referred to as *non-maximum suppression*.

The final layer outputs both bounding box coordinates and the class probabilities. The bounding box coordinates are normalized according to image width and height. During training, the network applies the squared error sum of the combined localization error and classification error. According to Redmon et al. [27], applying the squared error sum as a loss function is not ideal regarding localization errors, but it is used because it is easily optimizable. However, the error function is modified to overcome some of its shortcomings. The modifications include decreasing the loss from the confidence predictions of the boxes that do not contain an object. Further, they increase the localization loss of the bounding box predictions. All training and inference of YOLOv1 were conducted using the Darknet framework.

When testing the network on PASCAL VOC 2007 dataset [46], YOLOv1 was the second-best detector after Faster R-CNN. However, Faster R-CNN was significantly slower. The only other real-time detector at the time was DPM [47], with its 30 Frames Per Second (FPS) and was thus significantly behind YOLOv1 in both speed and accuracy. Redmon et al. [27] also presented a smaller and faster version of YOLO called Fast YOLO, detecting at 155 FPS, truly pushing the speed limits of object detection

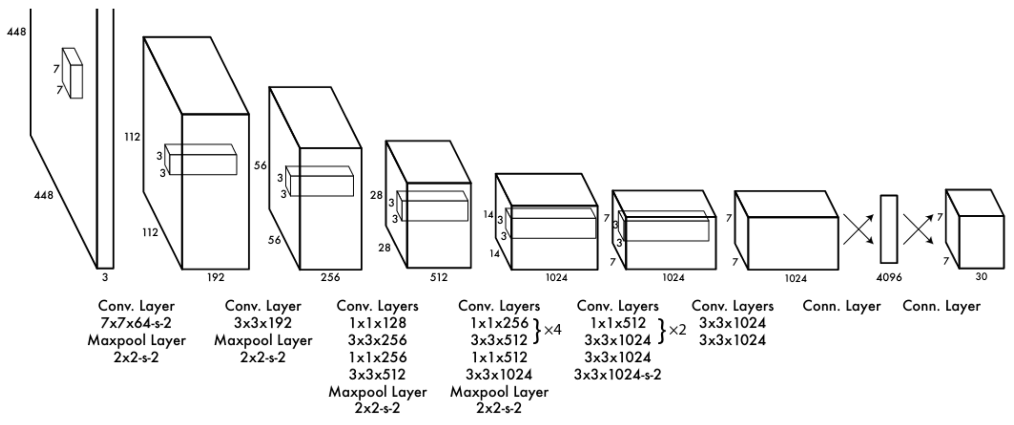

448
7
7
448
3
112
3
3
112
192
56
3
3
56
256
28
3
3
28
512
14
3
3
14
1024
7
3
3
7
1024
7
7
1024
4096
7
7
30
Conv. Layer 7x7x64-s-2 Maxpool Layer 2x2-s-2
Conv. Layer 3x3x192 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x128 3x3x256 1x1x256 3x3x512 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x256 3x3x512 }×4 1x1x512 3x3x1024 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x512 3x3x1024 }×2 3x3x1024 3x3x1024-s-2
Conv. Layers 3x3x1024 3x3x1024
Conn. Layer
Conn. Layer

Figure 12: The YOLOv1 network. Adopted from 27.

YOLOv1's architecture is relatively simple compared to its competitors, but the simplicity is essential for speed. Although detection speed is arguably the model's greatest strength, it is not the only advantage over the traditional models. For instance, the model can reason globally about an image. This is in contrast to evaluating images part by part, such as the sliding window approach. YOLO extracts features from the entirety of the input image in one forward pass and applies all these global features when predicting each bounding box. The global evaluation is advantageous for extracting contextual information about the image. This leads to YOLO making very few background errors, where other detectors might mistake a section of the background as an object.

Another notable strength is the network's proficiency in certain aspects of its training phase. Primarily due to the simple architecture, the network is end-to-end trainable, making it easy and fast to optimize. This is in contrast to the pipeline approach, where optimization is slower and more complex. YOLO also generalizes extraordinarily well. In training, it learns generalizable representations of objects and can use these representations when performing detection on new data. Keeping all of these merits in mind, YOLOv1 is lagging behind the contemporary state-of-the-art detectors in terms of accuracy by an arguably significant margin. YOLOv1 makes localization errors more frequent than, for instance, Faster R-CNN. Moreover, YOLO's grid cell method applies strict spatial constraints, such as there can only be a certain number of objects per cell and that all objects per cell must be of the same class. Taking this into account, if accuracy is undoubtedly the most important metric, then a network such as Faster R-CNN likely yields better results. However, if the speed-accuracy tradeoff is important, YOLO certainly has its merits.

Approximately a year after the publishing of YOLOv1, Redmon and Farhadi [48] YOLO9000, named after its ability to classify more than 9000 classes. Some of the improvements from YOLOv1 to YOLO9000 were introducing batch normalization to the convolution layers, replacing the fully connected layers used for prediction with anchor boxes, adding k-means clustering for automatically adjusting anchor box dimension, and introducing a high-resolution classifier and detector.

In April 2018, Redmon and Farhadi [49] published the paper "YOLOv3: An incremental Improvement" [49]. This version had a slightly larger network and was a bit more accurate than its predecessor. According to Redmon and Farhadi [49], however, YOLOv3 did not include any major steps forward, only a few improvements. One of the more considerable changes was replacing the feature extractor with Darknet-53 [49].

In the *mosaic augmentation* process, an image is selected and resized quadratically to a given image size - for example: 640x640. Three other random images are selected, resized to 640x640, and seamed together at the edge, totaling a 1280x1280 image. Finally, an area of the seamed image is cropped out, with the original size of 640x640, and placed in a training batch. The mosaic data augmentation helps detect objects of a smaller size than usual and introduces a translational shift that helps for generalization.

The release of YOLOv4 [50] came in April 2020. Bochkovskiy et al. [50] added several new features resulting in a 10 % accuracy increase and a 12 % FPS increase compared to YOLOv3 on the COCO dataset [51]. In June 2020, a couple of months after the release of YOLOv4, YOLOv5 was published by Jocher [52], a data scientist that had worked with YOLOv3. Unfortunately, the release of YOLOv5 came without a paper and it was more difficult to evaluate the network. publishing very promising results. It did, however, receive high praise from the community data which advocated for its promising results [53].

At the time of writing, it is still up for debate whether YOLOv5 is better than YOLOv4, with researchers backing both models. Followingly, the actual naming of YOLOv5 has been the cause of some controversy in the computer vision community as it is not a continuation of the YOLOv4 project, but was, in fact, developed by a different team. Both architectures have very similar essential components. They both resort to CSP Bottleneck [54, 55] as a backbone. Both employ the PA-Net [56] as a model neck for aggregating features. All things considered, YOLOv4 and YOLOv5 are in the same family of models, sharing most of the key characteristics, even though they are created by different developers. Notably, both projects have released improved versions of the networks after the initial release. Scaled YOLOv4 [57] was published in November 2020, and the YOLOv5 project is in continuous development, with its fifth and latest release on April 11th, 2021. All in all, both YOLOv4 and YOLOv5 are highly capable models, performing at state-of-the-art levels.

2.8 CenterNet

CenterNet is an object detection architecture that is claimed to be "more accurate, faster, and simpler" than contemporary models [23]. It works by modeling objects as their center points. This is different from the traditional approach, where object locations are predicted directly as bounding boxes. From the estimated center-point, the detector regresses the remaining object properties, such as box size, 3D location, orientation, and pose.

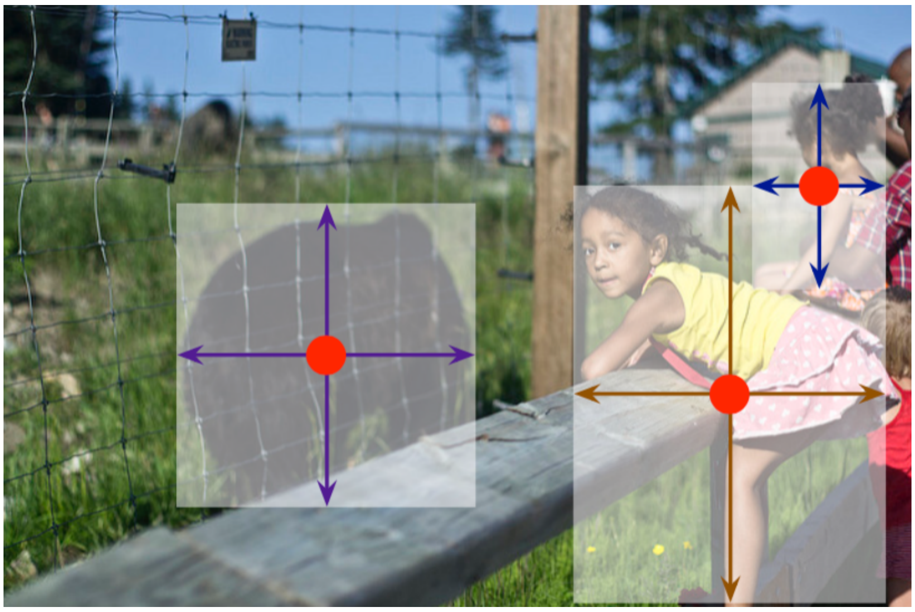


Figure 13: CenterNet estimates objects as points and regresses to other properties such as height and width. Adapted from 23.

While object detectors such as in Ren et al. [26] and Redmon and Farhadi [49]'s works are demonstrating good performance, Zhou et al. [23] identifies that a lot of the computation is wasteful. The main source of inefficiency originates from the anchor-based approach where fixed regions of the image (anchors), often at multiple scales, are exhaustively enumerated in order to produce bounding boxes that may capture objects at different locations. However, the magnitude of bounding boxes introduces significant post-processing overhead. For instance, NMS is usually applied to discard bounding boxes belonging to the same object, which consists of many IOU calculations. In addition, a great deal of the boxes is being forwarded for classification, even when most of them will be discarded. Also, NMS is hard to differentiate (train), making it complicated for networks that use it to be end-to-end trainable.

CenterNet proposes a solution that completely removes NMS and the complications it introduces, making the network end-to-end trainable with zero post-processing overhead. Instead of generating a multitude of anchors for each object, a single heatmap is generated in one forward pass. The heatmap's peaks represent object centers, which correspond to exactly one object. Consequently, the need for NMS is removed. Features extracted from object center locations can be further forwarded for regression tasks.

In order to produce a heatmap of sufficient high-level and spatial value, an encode-decoder network is used with feature aggregation at each layer, such as DLA. The heatmap uses a stride of 4, resulting in a larger feature map than what previous models would normally use. Given an input image of width W and height H, a feature map $F \in \mathbb{R}^{\frac{W}{4} x \frac{H}{4} x C}$ is produced representing the heatmap . The variable C, indicates the number of classes or types of keypoints desired. A heatmap location value of 1 indicates an object center, and a value of 0 indicates no object is present at that location. In order to learn to produce such heatmaps, ground truths need to be available first. Object centers are obtained by calculating the center of their bounding boxes and dividing by the stride, 4. The value of the ground truth heatmap at this position is 1. In addition, ground truth centers are "splatted" onto neighboring points using the kernel in Equation 1.

$$-exp(\frac{(x-c_x)^2+(y-c_y)^2}{2\rho_p^2}) \tag{1}$$

Equation 1's variables c_x and c_y denotes the calculated ground truth spatial position in the ground truth heatmap. ρ_p is an object-adaptive standard deviation. The training objective for the heatmap is a pixel-wise logistic regression loss with focal loss [58]. To account for the error introduced by the stride, an L1 loss function is used for training object center offset estimation.

As mentioned above, features extracted at object centers can be used to regress other properties. The 100 top values neighboring a peak point are used as features for further regression. This is done for each peak.That is, it is used for each object. In CenterNet, a $3x3$, followed by a $1x1$ convolution, is used to regress width and height in order to produce bounding boxes for the estimated object centers. This is depicted in Figure 13.

At the time of release in 2019, the model achieved the best speed-accuracy trade-off on the COCO dataset and was capable of real-time processing.

2.9 Multiple Object Tracking

Multiple Object Tracking (MOT) extends the task of object detection by maintaining the identities (IDs) of localized objects through a series of frames from video sequence [29].

In order to create a fully automated pipeline that extracts object trajectories from a video sequence, by far the most common approach is to use a *Detection-Based Tracking* system [29]. In this setup, solving the task of multiple object detection is an essential prerequisite in order to solve the task of object tracking. This leads to the two-stage design seen in Figure 14: First, a detection model outputs object proposals, and second, an association model associates the proposed objects with existing trajectories (often simply referred to as *tracks*). Association models can also be referred to as *re-identification* (re-ID) models. It partly explains the recent interest in detection-based tracking algorithms. Object detection has entered a revolutionary stage with the introduction of deep neural networks. Accordingly, tracking benefits directly from that by inventing clever ways to adopt them to MOT scenarios.

According to Luo et al. [29], tracking systems (or MOT systems) can be further categorized by their processing methods. Trackers can be *online*, meaning that tracking targets can be estimated on each incoming frame. This is in contrast to *batch*-processing methods that rely on the entire sequence to output targets. A motivational factor for using online trackers is that they allow the possibility of real-time systems.

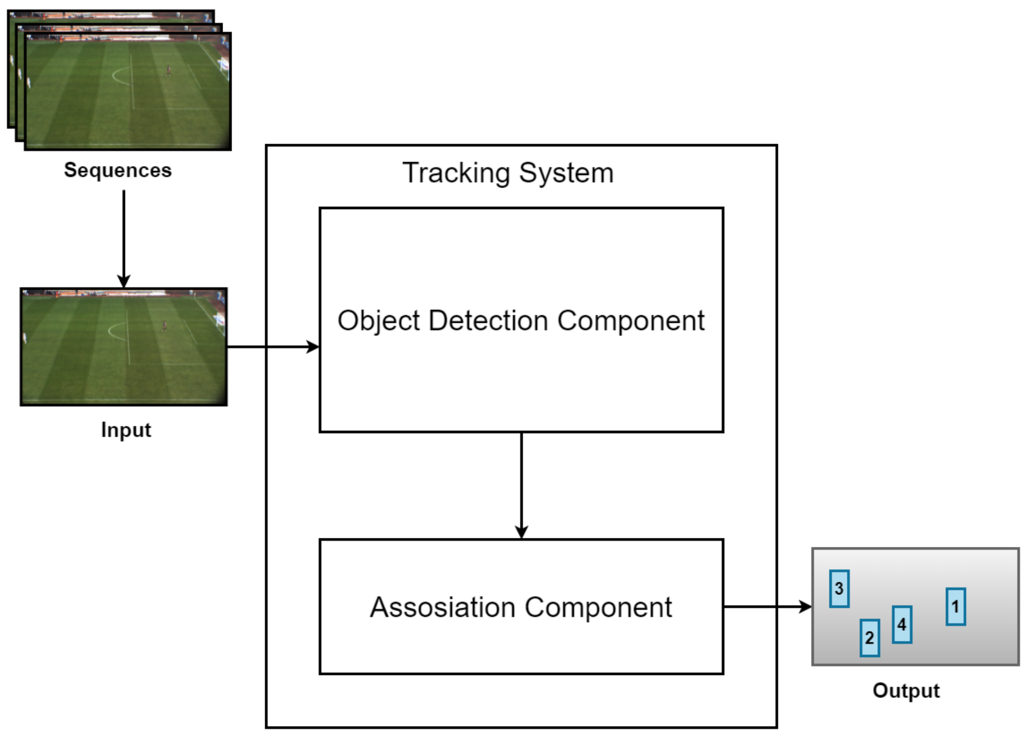


Figure 14: General structure of online tracking systems. Single images (frames) from a video sequence are served, one at a time, to adhere to the online behavior of the system. For each incoming frame, the two components sequentially collaborate to produce object locations with IDs attached.

In the context of an online tracking system, the detector simply accepts images as input in a sequential manner and remains oblivious to other frames from the sequence. The sequence, or the video, is read frame-by-frame and converted into images in an online fashion and is consecutively served to the detector. For each image input, the detection output is then forwarded, still, in an online fashion, to the association model.

The association component's job is to further process the objects localized by the detector, frame by frame. This involves the association between detections and tracks, or simply re-identification of objects, by recognizing objects from previous frames. The result is a complete system that outputs a list of coordinates representing bounding boxes and that are further tagged with IDs, for each frame it receives.

2.10 SORT: Simple Online Realtime Tracking

One of the more popular methods for tracking is the SORT algorithm. Published by Bewley et al. [30], SORT is a multiple object tracker that can perform in real-time.

Its desirability lies in its simplicity and flexibility. The algorithm is specifically developed to be used with object detector outputs. Thus, it only uses bounding box positions and sizes to perform the tracking task. This means a variety of different object detectors can be used with SORT. However, it also places a complexity restriction on the method. Specifically, since only bounding boxes are used as input, re-identification of objects after longer occlusion times is not possible. For example, if an object where to move completely out of the frame for a significant time period, the ID would simply be destroyed and it would be given a new ID on re-entry.

$$\mathbf{x} = [x, y, s, r, \vec{v_x}.\vec{v_y}, \vec{s}]^T \tag{2}$$

Equation 2 shows the state representation SORT uses to track objects. x and y are object center coordinates. s and r are scale (area) and aspect ratio, respectively. An IoU-based affinity matrix [59] is created with the projected tracks and detections for the next frame in question. The Hungarian algorithm [60] is used to associate tracks with detections optimally. This completes the association step on a frame-to-frame basis. An IoU-threshold is used to prevent far-fetched associations. After the association step, the states are then updated using a Kalman filter [24].

Whenever a detection has an IoU-overlap below the specified threshold, it is marked as untracked, and a new identity is given. The corresponding track state is initialized with zero velocity and needs to gather evidence in the following frames in order to correctly model movement. If the track state has an IoU-overlap below the threshold for a set number of frames, the track is discarded.

SORT was ranked the best open-source multi-object tracker on the MOT Challenge dataset [61], at the time of publication. Although the algorithm struggles with occlusion and re-occurring objects, it functions well as a fast and lightweight tracking algorithm.

2.11 DeepSORT

Only a year after the publication of SORT, Wojke et al. [31] released a new iteration of the algorithm, appropriately named DeepSORT. The new iteration was developed to mitigate the shortcomings of its predecessor, specifically its deficiency in occlusion scenarios. In essence, DeepSORT introduced a *deep association metric* to better inform the association step, allowing for more robust re-identification. Consequently, DeepSORT adds functionality for re-identification at the cost of introducing some of the complexity that SORT was trying to avoid. However, DeepSORT still has a relatively simple implementation and can track in real-time. The additional complexity is arguably well justified when considering that it significantly reduced the occlusion difficulties that plagued SORT, resulting in a decrease in ID switches by 45 %.

DeepSORT is essentially the same algorithm as SORT in nearly every aspect. First and foremost, it is still an online MOT algorithm. Secondly, it still works in conjunction with replaceable object detectors to process their outputs. Third, it still relies on the same object state vector from Equation 2, and Kalman filters to estimate its future states. Lastly, the association of existing object trajectories and newly arrived object detections are still treated as a min-cost linear assignment problem.

The changes from SORT to DeepSORT can be summarized as follows:

- A CNN module that creates feature descriptors (vectors) from cropped bounding boxes of the objects.
- An updated association metric that incorporates both motion and appearance information.
- A *Matching Cascade* algorithm to solve the linear assignment problem.

The CNN Module

Following the object detectors prediction output, DeepSORT crops out the bounding boxes and resizes them for further processing. The cropped images will be input to a CNN pre-trained on a large-scale person re-identification dataset. For all detected objects, the CNN outputs a feature vector that is incorporated in the association metric described next.

Association Metric

The association metric is used to assign costs in the creation of the affinity matrix for existing tracks and detections. It is a weighted sum of two parts. The first part measures the distance in motion between a projected track and detection using the Mahalanobis distance [31]. This is in contrast to SORT, which only uses the IoU metric to create the affinity matrix. The Mahalanobis distance has the advantage of automatically dealing with state estimation uncertainty and unlikely associations. Still, it provides a rough estimate at best in the case of non-linear or random movements, for instance, in the case of rapid camera displacements. It is a strongly informed metric for short-term prediction when state uncertainty is low.

Each track is accompanied by at most 100 appearance features produced by the CNN. These correspond to the objects associated with the particular track in previous frames. Thus, the second part of the association metric measures appearance similarity by the min-cosine distance between the appearance feature of a detection and the 100 appearance features of a track. The inclusion of appearance information in the association metric allows for recovery from long-term occlusion.

Matching Cascade

While SORT resorts to solving the association problem globally by directly applying the Hungarian algorithm, DeepSORT uses a different approach. Since the specific details of the algorithm are outside the scope of this thesis, a short description is provided instead. Essentially, the Mahalanobis distance causes a priority towards tracks with high uncertainty during association. This is undesired as the metric should increase the cost due to larger uncertainty. Therefore, the association step is sequentially solved for increasing track age and unmatched detections.

2.12 FairMOT

FairMOT [32] is a next-generation, multiple object tracking system. It is a one-shot model, which impressively makes it possible to perform real-time tracking.

The progress in the separate fields of object detection and object re-identification has been vast, yet progress on the joint effort to produce stand-alone multiple object tracking systems is still in its early stages. Zhang et al. [32] points out that in these architectures, re-identification is not fairly learned as they treat it as a secondary task in favor of the object detection task. This causes a significant bias towards the former, which hurts tracking accuracy.

Table 1 lists several issues Zhang et al. [32] identifies with previous MOT tracking architectures and the corresponding proposals to deal with these. The improvements proposed are also reflected in FairMOT's architectural design, which is shown in Figure 15 and described below.

Issue	Proposal
Non real-time processing in multi-stage models	Single shot model
ROI pooling in anchor based approaches extract irrelevant re-ID features	CenterNet to estimate object points
Re-ID feature extraction biased towards detection task	DLA + High-Res feaure maps
High dimensional re-ID features hurts detection accuracy	Only 64/128-dimensional re-ID vector
Cascaded style trackers (anchor based) biased towards detection task	Parallel re-ID and detection heads

Table 1: A list of issues Zhang et al. [32] identifies with previous tracking approaches. Corresponding proposals to each issue is listed on the right-hand side.

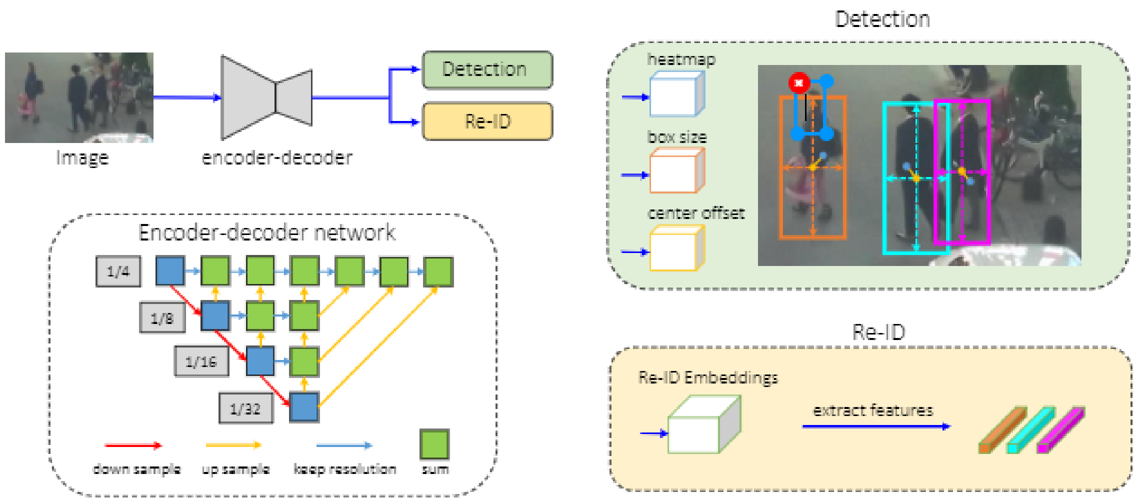


Figure 15: The architecture of FairMOT. Adopted from 32. A high-level overview is depicted in the upper left corner. Notice the simplicity of its design. The Encoder-decoder network (bottom left) consists of ResNet with DLA on top and deformable convolutions (”DLA34”). Two homogeneous branches are added on top of the encoder-decoder network to facilitate fair learning of the detection and re-ID task. The detection branch (upper right) is borrowed from CenterNet to regress object centers. The re-ID branch (bottom right) is responsible for producing object re-ID embeddings.

A variety of state-of-the-art feature extractors can serve as a backbone for FairMot. ResNet-34 is used in their benchmark model as it provides a good trade-off between speed and accuracy. A new modified version of DLA [23] is added on top to create multi-layer aggregated features of high resolution. Furthermore, deformable convolutions are used in all convolution layers to perform up-sampling, increasing network adaptability and accuracy. Zhang et al. [32] refers to the resulting backbone as ”DLA34”. A detection branch, adopted from CenterNet with very little modification, is added to the backbone. It is responsible for creating object heatmaps and offsets for object centers and boxes. A parallel, separate re-ID branch is added to the backbone, which performs a convolution to transform the backbone output into a 128 x W x H feature map. Estimated object centers can then be used to extract 128-dimensional vectors as re-ID embeddings. Joint training of the separate branches ensures that shared features facilitate both tasks. All in all, these new additions and the philosophy of fair training leads to FairMOT being seriously competitive with current state-of-the-art methods on the MOT Challenge dataset.

2.13 Metrics

In order to qualitatively assess performance of object detectors and MOT systems, various metrics are needed. First the basic components of the different metrics are described. At the end of the subsection, several metrics are listed.

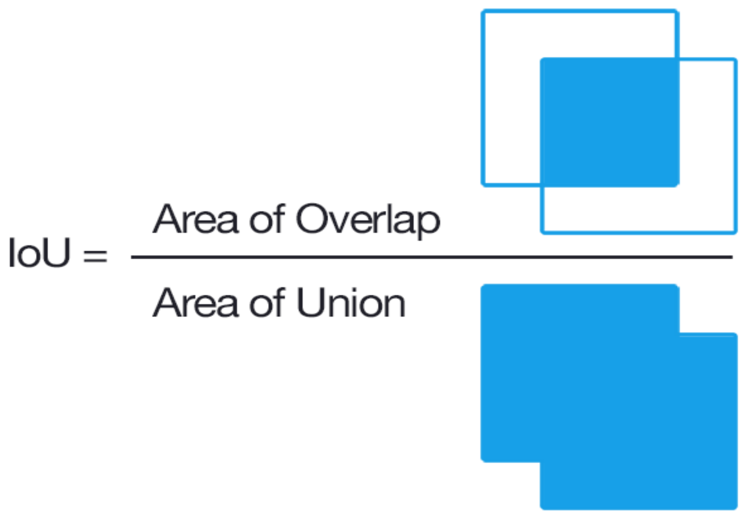


Figure 16: A visualization of how the IoU is computed. Adapted from 62.

Ground truth data for object detection tasks consists of bounding boxes in the input image, marked by pixel coordinates. In order to determine if a proposed bounding box counts as a correct prediction, the concept of IoU was introduced. Figure 16 explains how the IoU is computed. If the predicted bounding box has an IoU overlap with a ground truth box above a certain threshold, it is considered a correct prediction, or a *true positive* (TP). If a prediction has an IoU score below that threshold it is considered a *false positive* (FP). If a ground truth box has no IoU overlap above the threshold, it is considered a *false negative* (FN), or a missed detection. Usually, the threshold is chosen to be 0.5.

$$\text{Precision} = \frac{TP}{TP + FP} \quad (3)$$

$$\text{Recall} = \frac{TP}{TP + FN} \quad (4)$$

Equation 3 and 4 shows how precision and recall values are computed from TP, FP, and FN. These are essential in understanding the mean Average Precision (AP) object detection metric.

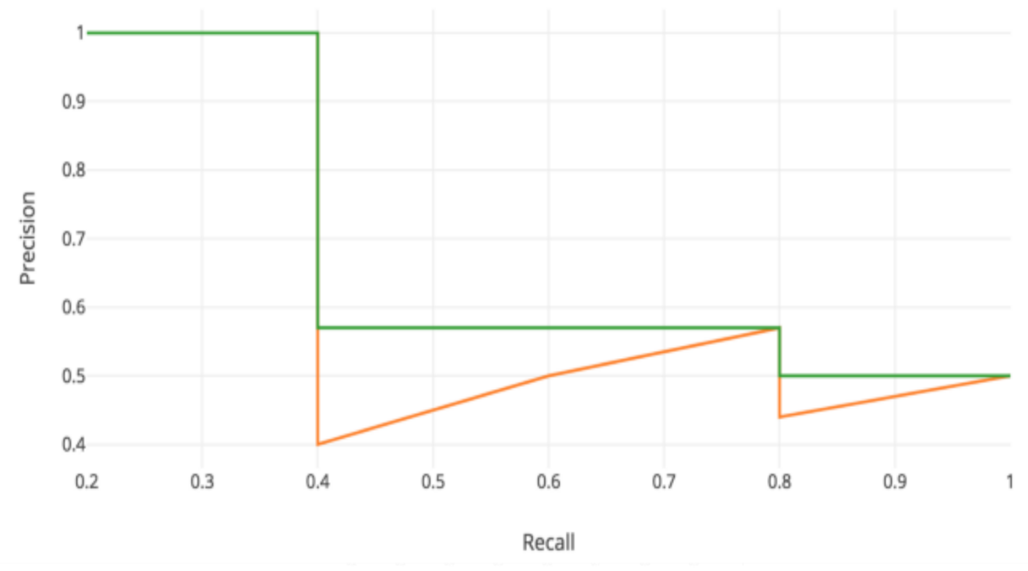


Figure 17: The precision-recall curve. The yellow zig-zagged line represents the real recall and the green line represents the interpolated recall at max value. The green line is used when calculating AUC. Adapted from 63.

In Figure 17, the yellow curve is constructed by calculating precision and recall values for increasing confidence scores of the detections. However, the curve often appears zig-zagged and is therefore interpolated at usually 11 equidistant recall points. The interpolation creates the green line depicted in Figure 17. The Area Under the Curve (AUC) is then computed from the green curve. To obtain the final AP, the AUC is calculated for each class, and then averaged.

In terms of MOT metrics, the CLEAR MOT metrics [64] and ID metrics [65] have gotten considerable attention in recent years. The mapping of ground truth tracks to predicted tracks is for both metrics dealt with as a min-cost assignment problem. The main difference between the two is that the ID metrics finds a globally optimal solution, while the CLEAR metrics does it on a per-frame-basis.

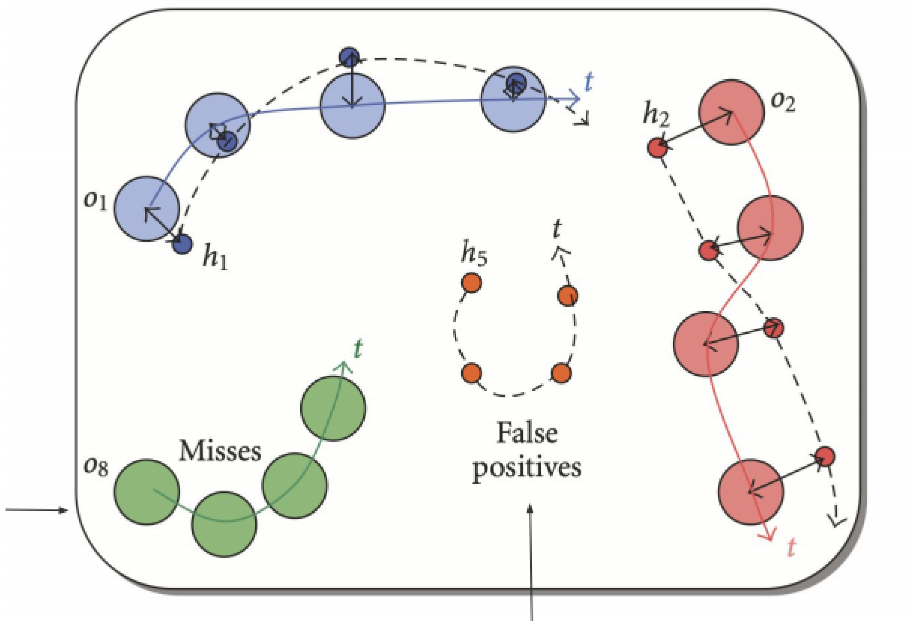


Figure 18: Adapted from 64. h_i are *hypothesises*, or predicted trajectories. o_i corresponds to ground truth object trajectories, or at each time step: g_t.

Figure 18 depicts a tracking scenario through several time steps. The figure illustrates concepts such as misses (m_t) and false positives (fp_t) in terms of tracking. Figure 19 further demonstrate how false positives and misses are counted on a per-frame-basis. The figure also depicts how mismatches (mme_t) occur.

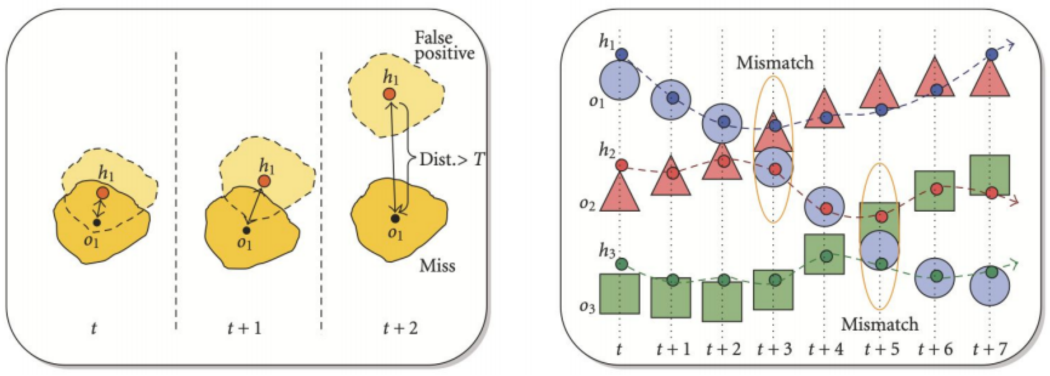


Figure 19: Adapted from 64. False positives occur when a previously tracked object drifts off course, determined by a certain threshold and usually computed with IoU. A mismatch occurs when new correspondences contradict the previous frame.

Equation 5 and 6 shows the CLEAR MOT metrics. MOTA evaluates tracking accuracy, while MOTP is concerned with localisation precision. d_t^i is the distance in overlap between matches, and c_t is the total number of matches made. The MOTP sums over all matches and all frames. It is worth noting that the MOTA metric has a range of $-\infty$ to 100. If the value is negative, there is most likely a large number of false positives present.

$$\text{MOTA} = 1 - \frac{\sum_t (m_t + fp_t + mme_t)}{\sum_t g_t} \quad (5)$$

$$\text{MOTP} = \frac{\sum_{i,t} d_t^i}{\sum_t c_t} \quad (6)$$

Equation 7, 8, and 9 comprise the ID metrics. IDF_1 is defined by computing the F_1 score using ID-precision (IDP) and ID-recall (IDR). Equation 9 shows how to compute it. IDP and IDR are, in turn, comprised of the metrics; ID True Positives (IDTP), ID False Positives (IDFP), and ID False Negatives (IDFN). Note that this is essentially identical to the recall and precision equations. Below IDP and IDR is described.

- IDP: Fraction of computed detections that are correctly identified. "Correct" means that there is a 1-to-1 mapping for each ground truth instance
- IDR: Fraction of ground-truth detections that are correctly identified.

$$\text{IDP} = \frac{IDTP}{IDTP + IDFP} \quad (7) \qquad \text{IDR} = \frac{IDTP}{IDTP + IDFN} \quad (8)$$

$$\text{IDF}_1 = \frac{2 \cdot IDP \cdot IDR}{IDP + IDR} = \frac{2IDTP}{2IDTP + IDFP + IDFN} \quad (9)$$

Other metrics that are frequently used in object detection and multiple object tracking is summarized below [66, 31]. Note that AP's correspond to the COCO API definition when calculating AP [51].

- AP: (Actually AP50:95) Mean of AP calculated with thresholds from 0.50 to 0.95 with a step size of 0.05
- AP50: Average Precision calculated with an IoU-threshold of 0.5.
- AP75: Calculated in the same way as AP50, only with an IoU threshold of 0.75.
- APsmall (APs): AP for small objects: area $< 32^2$ px
- APmedium (APm): AP for medium-sized objects: $32^2 <$ area $< 96^2$ px
- APlarge (APl): AP for large objects: area $> 96^2$ px
- ID switches: The number of times when an ID of an object is changed.
- Fragmentations: The number of times a track is interrupted by a missing detection.
- Transfers: A particular case of ID switches; the number of times where one object's ID is transferred to another object.
- Mostly tracked (MT): The percentage of ground-truth tracks that have the same label for at least 80 % of their life span.
- Mostly lost (ML): Percentage of ground-truth tracks that are tracked for at most 20 % of their life span.

2.14 K-means Clustering

K-means clustering is one of the most used unsupervised machine learning algorithms, as well as being one of the most intuitive. K-means' objective is to group unlabeled data samples based on their similarities. In simpler terms, it groups points together based on how close they are to each other. Due to the algorithm's generality, it can be applied in a broad range of research fields. The algorithm works in an iterative fashion, with a guarantee of convergence. K-means starts by positioning cluster centers or *centroids*, in the data space, usually done at random. Data points are then assigned to the centroid, to which the distance between the centroid and the data point is the smallest. A usual method for measuring the distance between data points is the Euclidean distance. After all the data points have been assigned to a cluster, the cluster centroids are recalculated based on the average position of the data points connected to them. After the centroids have been recalculated, or re-positioned in the data space, the process starts again by assigning all data points to their nearest cluster. As the centroids are in new positions, some of the data points may be closer to a different centroid than previously, thus changing their association. If none of the data points change its cluster during an iteration, the algorithm has reached convergence. Commonly, a stop-criteria of a predetermined number of iterations is set in place to stop the algorithm early in case it runs too long. Thus by the end, the algorithm has hopefully discovered hidden structures in the data and clustered the data points together accordingly.

2.15 Detectron2

Detectron2 [67] is a an open-source framework that implements state-of-the-art computer vision algorithms. It was developed by the Facebook AI Research group. The framework provides cutting-edge functionality for common computer vision tasks such as object detection, instance segmentation, and panoptic segmentation. The framework is built on top of the PyTorch library [68]. It contains several different backbones, meta-architectures, and utilities, so one can easily tailor the framework to best fit a certain use-case. Models initialized with pre-trained weights from their model zoo [69] can be used out-of-the-box for direct predictions or further training. Because of its high modularity, it is fairly straightforward to extend the framework with user-defined models and training from scratch using customized training loops, given some familiarity with PyTorch.

2.16 CVAT: Computer Vision Annotation Tool

In order for machines to learn object detection and tracking from videos, it is necessary to obtain examples specific to this. In order to get to useful training examples, the videos must be annotated. That is, creating labels showing the model what the correct answer is when given a task. The Computer Vision Annotation Tool (CVAT) [70] can be utilized for this task. CVAT is an open-source annotation tool developed for labeling images and videos for machine learning purposes. It provides a rich and friendly user interface that can be hosted on the web for easy access. Multiple users labeling the same data are allowed, and CVAT provides tools that facilitate concurrency and collaboration for increased productivity. Completed annotations can be downloaded in a variety of different dataset formats, including the standardized COCO format [51]. Figure 20, shows the typical workspace a user is exposed to when working with this tool.

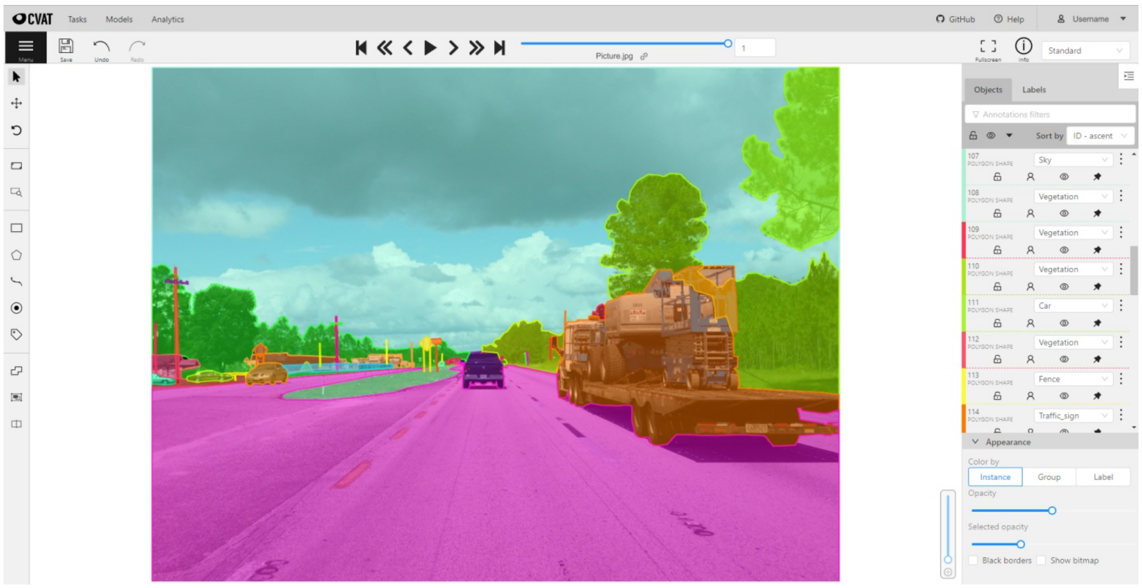


Figure 20: The CVAT UI. The screenshot is from the web app. The tabs on the top provide controls for playback of the video. The interactable functionalities on the left contain the work tools used to create annotations of different kinds. On the right-hand side, a display lets the user manage current annotations from the current frame. There, one can, for example, change and merge annotation IDs.

2.17 Docker

Docker is a prevalent software designed for simplifying the development and deployment of applications [71]. It removes repetitive configuration tasks that are common when running and deploying applications on new machines. This is achieved by packaging infrastructure with the application code to create isolated, platform-independent software. At the heart of this approach are Docker *Images* and *Containers*.

Images

Dockerfiles are blueprint-like configuration files. They specify operating system tools, application code, environment settings, run times, and more. When ran with Docker, Docker Images are created. These are executable software packages available to Docker, and they contain everything specified in the Dockerfile.

Once an image is created, it can be created on any machine regardless of the underlying hardware or operating systems, as long as Docker is installed. This is very useful from a developer's perspective. Everything an application needs to run is contained in the image, which can be created from the Dockerfile bundled with the source code.

Since application environments are essentially bundled as code, Docker images have extensively been subject to re-use. In fact, Docker-Hub is an ecosystem of reusable images shared by various organizations and individuals. For example, if one would need the latest Ubuntu system tools to create an application, one could simply fetch the official Ubuntu image from the Hub. Docker images also conform to the concept of inheritance, so one could easily extend pre-written images with additional or user-specific features.

Containers

If Images are executable packages of software, then containers correspond to images at runtime. They run only in the context of the image that spawned it, isolated and contained from everything else on the machine.

2.18 Early Stage of Tracking and Identification in Sports Analytics

In 2009, Emad Monier *et al.* published the paper "A Computer Vision Based Tracking System for Indoor Team Sports" [72]. At the time of publishing, the explosive growth of CNNs and deep learning in the Computer Vision field was arguably still a few years away. For that reason, many of the techniques used were significantly different than what is currently being used in object detection and tracking models. That being said, the subject is very much related, and the paper showcases how a different form of computer vision techniques can be applied to the same issue. The sports used in the study were handball and basketball.

As a first step before starting detection, the paper makes a closed-world assumption. Following this assumption, nothing outside the frame on which the tracking is applied can interfere with the tracking problem. This assumption would not be valid in our current experiments, as both the players and the ball move in and out of the camera frame on several occasions in the input video. It is albeit a very useful assumption to make when possible, as it would remove the issue of re-detecting and, more difficultly, re-identifying players, which is not a trivial task.

To detect the players, the paper applies background detection. In this process, pixels taken from an empty background image are subtracted from the image with players in it. Then they apply template matching where a template, i.e., a sub-image they wish to match, is glided over the frame. The template is matched pixel by pixel, subsequently returning the best matching area for that template.

The primary metrics returned from the tracking system are a speed profile, which determines whether players are jogging or sprinting, and field coverage, i.e., tracking a player's position through a match. These metrics are of equal importance in the sports analytics field now. As a parallel to this, it should be noted that even without the major advancements in the computer vision field of recent times, the desire to apply computer vision techniques in sports analytics was still evident.

2.19 Self-Supervised Small Soccer Player Detection and Tracking

In the recent paper published by Samuel Hurault *et al.* in October 2020 [73] they show how state-of-the-art tracking models achieve exceedingly good results on the task they are trained for, but still fail on more complex tasks, such as tracking soccer matches. Moreover, they believe these difficulties are mainly caused by the players' small size and that they are very similar to each other within each team. Thus, in the paper, they present a MOT-model for detecting and tracking soccer players, with special attention to the detection of small, low-resolution players in varying conditions. In addition, they propose self-supervision, enabling the model to train without ground-truth labels.

In their paper, they discuss some issues that are very relevant to our experiments, with three issues in particular. Firstly, they conduct their experiments with a single, fixed position, low-resolution camera. They purposefully do this to create a simple and efficient system that can be implemented with limited camera resources. This is also an essential point of our experiments. To test the level of performance possible when only using a single camera, in contrast to what is used by major vendors such as media productions and premium sports analytics firms.

Secondly, they heavily focus on how the player size and the similarity of players within each team provides a demanding challenge for a MOT-model. The issue is, in some ways, a direct consequence of the camera implementation. Player sizes are already small in a high-level media production, but the problem becomes exceedingly difficult when using a single, low-resolution, wide-pan camera. In attempts to solve this issue, they perform data augmentation, where they purposefully reduce image sizes to help the model train for detecting the smaller objects.

Lastly, they also discuss the differences between using a two-stage model versus a one-stage model. In particular on how two-stage models, where firstly the detection task is performed, and subsequently the detection output is used as input to the re-identification task, are heavily dependent on the accuracy of the object detector to achieve good overall performance of the tracker. As they apply a two-stage model in their experiment, they admit to this drawback, although they attempt to mitigate it. This is highly relevant to our FairMOT experiment, where one of the main objectives is to treat both object detection and re-identification fairly. Predominantly, to avoid treating object detection as the main objective, with re-identification as a secondary task.

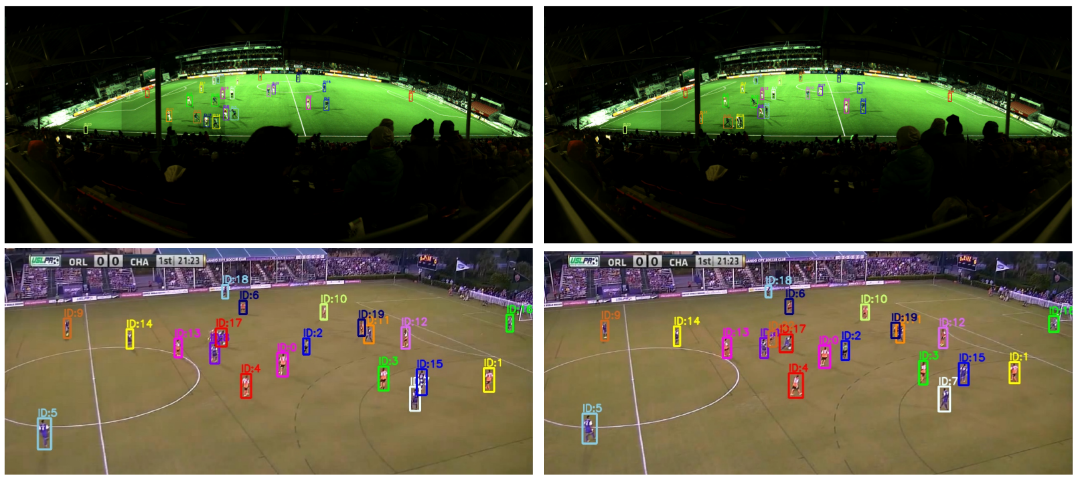


Figure 21: The visual output after applying the model on two different soccer matches. The two top frames correspond to the same game, and the bottom two frames to a different one. In this example there is a one-second temporal difference between the left-side and right-side frames to illustrate tracker behavior.

For the model's architecture, they use the tried-and-tested Faster R-CNN [26] network with a Feature Pyramid Network (FPN) [42] as the object detector, with slight modifications to increase accuracy on small object-detection. They go for a self-supervised approach for the backbone, using a teacher network and a student network, being ResNet50 and ResNet18, respectively. They apply the teacher network, pre-trained on a source domain, to train the student network on a target domain. Specifically, they use a teacher network pre-trained on human detection. They apply some corrections and fine-tuning and use this to train the student network, removing the need for having labeled ground truths in the target domain. Their main argument for this technique is the lack of publicly available annotated datasets and that producing the annotations manually is very time-consuming. Furthermore, they elaborate on how several recently published soccer trackers are only trained on a couple of hours of annotated test data. This is arguably significantly less than the required amount to capture all the various conditions found in soccer matches, such as weather conditions, kit color, and pitch conditions.

All in all, the paper presents a robust and accurate model, with no manual annotation required. According to their own tests, they achieve a MOTA score of 92.9 and an IDF$_1$ score of 80.9. In figure 21 we can observe some qualitative results from tests on two different soccer matches. Although the model is very accurate, it is far below real-time processing, with a throughput of only 6.5 FPS. However, as they utilize a two-stage model, some decrease in speed compared to one-stage models is to be expected.

2.20 FootAndBall: Integrated Player and Ball Detector

Published January 2020, FootAndBall by Komorowski *et al.* [74] present a lightweight and efficient detector for detecting soccer players and ball in high-resolution, long shot video of soccer matches. The method proposed by the paper is a fully convolutional, one-stage object detector. One of the main goals when designing the model was to increase performance. To achieve this, they greatly emphasized reducing complexity. To give an example, when comparing with SSD300 [75], FootAndBall has two orders of magnitude fewer parameters, with 199,000 parameters compared to SSD300's approximate 24 million parameters. The paper argues that the reduction in parameters does not impair detection, as FootAndBall only predicts on two classes and thus requires significantly fewer anchor boxes, in contrast to more general-purpose object detectors.

Their architecture is based on a Feature Pyramid Network design, starting with five convolutional blocks that process the input bottom-up and extracts feature maps. The output of conv-layer 2, 3, and 4 are each passed through a different 1x1 convolution shown in figure 22, which reduces the number of channels, so that the number stays the same for each output. Followingly, they add three heads to process the feature extraction; a player classifier, a player bounding box regressor, and a ball classifier. The player classifier and ball classifier each produce a confidence map, representing the probable locations for players and ball, respectively. The player bounding box regressor produces boxes in the same areas where the player confidence map denotes having a high probability of containing a player. With the addition of the FPN, they can use features extracted from both low-level convolutions and high-level convolutions. This is quite useful, as low-level features provide high precision for spatial location, and high-level features have a larger receptive field, enabling them to capture more contextual information from the input. Thus, the addition of an FPN is powerful in terms of accuracy and is especially significant for ball detection.

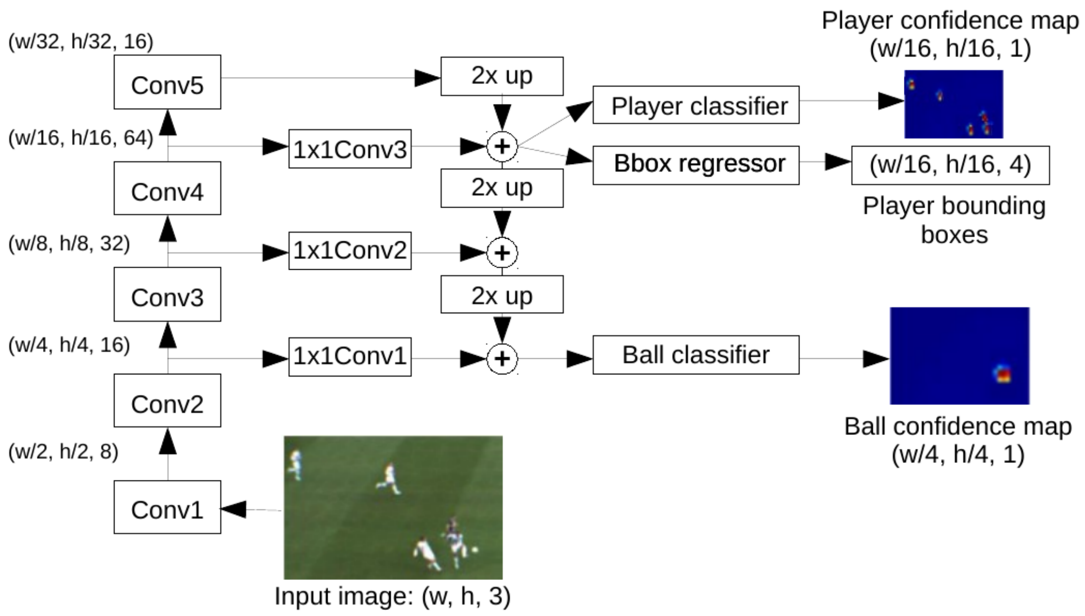


Figure 22: A high-level overview of the FootAndBall architecture (adapted from 74).

In their experiment, they trained their model on the ISSIA-CNR dataset [76] and Soccer Player Detection dataset [77], and tested it on a full-HD video stream, using a low-end GeForce GTX 1060 GPU. With this setup, they achieve real-time processing for detection, delivering a frame rate of 37 FPS with an AP50 of 0.909 for ball detection and an AP50 of 0.921 for player detection. The test data was taken from ISSIA-CNR. In comparison, a standard Faster R-CNN detector ran far slower, with a throughput of 8 FPS, player AP50 of 0.874, and ball AP50 of 0.866, using the same setup.

2.21 Soccer: Who Has The Ball? Generating Visual Analytics and Player Statistics

In the paper published by Theagarajan et al. in 2018 [78] they have a slightly different approach, angling their research towards measuring individual player ability. Rather than creating a general-purpose system for analyzing soccer, they assume the position of a talent scout, specializing in finding the best up-and-coming players. They argue that the normal process for identifying and selecting talent is subject to coaches underlying biases and preconceived notions of a good player. Therefore, in their paper, they propose a model for automating the talent identification process using computer vision and deep learning techniques that would operate free from bias and give results purely based on data.

To be fair, designing a system that truly determines the best player is, by today's standards, an extraordinarily difficult task. The system would need to be able to detect and measure a whole range of different attributes. Some of them are arguably quite straightforward to measure, such as physical attributes, and others that are a lot harder, such as technical attributes, mentality, or even "soccer-IQ", i.e., how well players can read and control games. That being said, developing a system that can give helpful insight into this matter is achievable, as several of the decisive attributes are, in fact, measurable. The authors decided upon using two key metrics: how much a player is carrying the ball and how well a player is able to complete passes to their teammates successfully.

For the task of localizing the players, they utilize the YOLO9000 network [48]. The network is comprised of a single CNN, that as a final output, returns a feature set including a bounding box prediction for each soccer player in an image. The YOLO-model is trained on the PASCAL VOC 2007 dataset [46], the COCO 2016 keypoints dataset, and the Imagenet dataset [45]. It should be noted; the model was trained explicitly on the "Person"-class, as all these datasets are pretty varied, containing multiple classes.

As per the work in section 3.3.2 in our thesis, they apply DeepSORT as the tracking component. For identifying which player has the ball, they test various CNNs, including ResNet-18 and ResNet-34 [40]. The network that achieved the highest performance was VGG-19, [79] with an accuracy of 86.59 %. For detecting successful passes, they track the changes in who is controlling the ball. They choose the criteria where a pass is successful; if the change of who is controlling the ball goes from one teammate to another. For the combined task of predicting which player controls the ball, and when a successful pass is made, VGG-19 produced an accuracy of 84.73 %.

Lastly, the paper conducts a different experiment in which the objective is to perform data augmentation. They use a Deep Convolutional Generative Adversarial Network (DCGAN) [80], where they train two CNNs, one generator, and one discriminator, against each other to generate test images. Example results from this method are presented in figure 23. As soccer matches are highly recorded events, there can be discussions made of the general effectiveness of data generation to improve a MOT model. Nonetheless, it is an intriguing addition to pursue, especially when not having an abundance of training data.

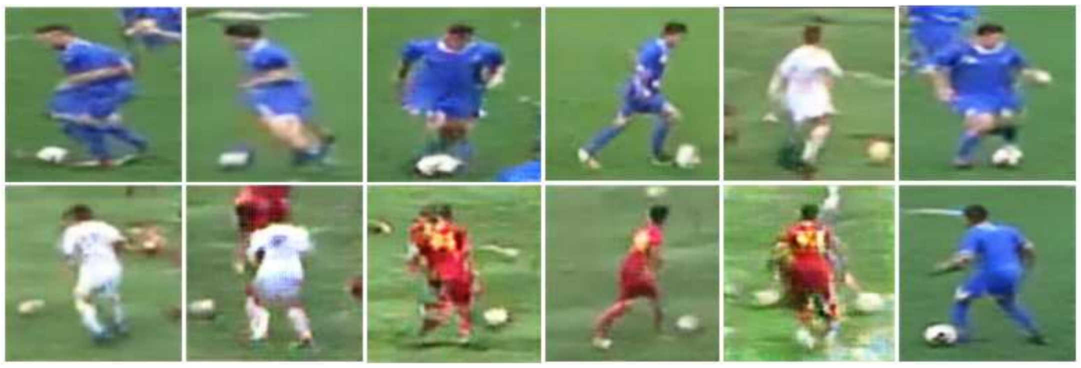


Figure 23: Outputs from the data augmentation method. All the above images are model-generated examples of a player with a ball.

2.22 SoccerNet

Created by Silvio Giancola *et al.* in 2018, SoccerNet [81] is a scalable dataset for action spotting in soccer games. The dataset is vast, consisting of 500 complete soccer games, making it a total of 764 hours of game time. The matches are gathered from the top six leagues of European soccer. The dataset contains in its entirety 6,637 temporal annotations, which are parsed from online match reports, with a one-minute precision, yielding one action per 6.9 minutes. These annotations are, in turn, manually refined to second precision. Following the creation of this dataset, SoccerNet turns their focus to creating and training a model for detecting soccer events. For their best model, they trained a ResNet-152 network with a special form of pooling called NetVLAD [82]. This model achieved an AP of 67.8 for detecting goals, yellow/red cards, and substitutions.

Although SoccerNet's focus lies in action spotting rather than object detection and tracking, the paper serves as evidence showing sports analytics' growing popularity in recent times. As an extension of the models used in this paper, action spotting can serve as a beneficial addition towards creating a multi-use sports analytics platform. Moreover, as the SoccerNet dataset is readily available online, free-to-use, it can serve as a great resource for training ML-models on soccer-related tasks.

2.23 Research Outside the Sport Analytics Field

In June 2020, Fritz-Olav Myrvang published, with the supervision of Frank Lindseth, the Master's thesis "Interaction detection, tracking and pose estimation in lobster farming". The goal of this research was to detect and track lobster behavior, especially concerning aggression. As aggression between lobsters is a major problem for lobster farming, the farmers use selective breeding to reinforce less aggressive traits. The thesis proposes an automation of tracking lobster behavior, aiding the lobster farmer in the task of selective breeding, using state-of-the-art computer vision methods. Although lobster farming may be far from the field of sports analytics, most of the same computer vision techniques are equally applicable. It should be noted; as we shared supervisor Frank Lindseth with this project, we inherited much experience in terms of useful computer vision methods and tools.

For dataset creation, some of the same data preparation methods were applied as in our work. A longer video segment was segmented into 1-minute segments, uploaded to CVAT, and annotated. While we gradually developed a more involved method, we started out utilizing the same method. A difference from our work is Myrvang's annotation for pose estimation, as opposed to solely focusing on detection. For the detection task, a range of different methods was examined. As real-time inference was a goal, main efforts were using single-stage detectors such as RetinaNet [83] and YOLOv3. Two-stage detectors like Mask R-CNN [84], and Faster R-CNN were, however, used as baselines. All of the models except YOLOv3 were implemented inside Detectron2. For tracking, the project applied the SORT algorithm. The previously mentioned pose estimation using keypoint detection was also implemented. This was accomplished using a Mask R-CNN network adjusted for keypoint detection.

As mentioned previously, a fair amount of inspiration was taken from this project. This mainly being the common use of CVAT, Detectron2, SORT, and YOLO.

3 Methodology

In this section, a detailed overview of the work conducted in this thesis follows. In particular, we describe the steps performed that ultimately led us to produce the results listed in Section 4. As argued for in Section 1.3, deep learning-based algorithms are adopted. Followingly, the steps taken are familiar to most supervised machine learning methodologies; dataset retrieval, labeling, model selection, training, and evaluation (result extraction). With the time restrictions imposed, four main experiments were conducted. Accordingly, each of their methodologies is described in separate sections. They mainly describe architecture and methods. Subsections before this describe steps intended to be shared or that are beneficial for all experiments.

There are steps described that are not strictly required in order to produce the results in Section 4. Nevertheless, they contributed valuable insights to the problem, helping us narrow down a strategy by figuring out what worked and what did not.

3.1 Environment

Our working environment is depicted in Figure 24. Each component is described in the following subsections.

3.1.1 Hardware

As this thesis is part of the university study program Computer Science at NTNU in Trondheim, we were lucky enough to be granted access to some powerful computing resources. The resources were allocated on an NTNU-server, accessible using remote access via Secure Shell Login (SSH). The server is powered by a cluster of two Intel(R) Xeon(R) Gold 6132 CPU @ 2.60GHz CPUs and two Tesla V100 General Purpose- (GP-) GPUs, each with 32 GB of memory. The system also has 754 GB of RAM at its disposal.

3.1.2 Workspace

Our coding editor of choice, Visual Studio Code (VS Code), can be installed with a useful extension that lets the user log onto remote servers via SSH [85]. The user space on the server is displayed in VS Code as if the user was working on a local machine. This proved tremendously useful in our case because we could edit and run code directly on the server through VS Code's user interface, all whilst utilizing its many excellent features. The traditional approach would be to develop locally, push the code to the remote servers, log onto the servers, and finally test the code. Obviously, this is a much more time-consuming, if not at least tedious, approach in comparison to the aforementioned.

Therefore, all of our coding efforts and data storage were conducted on the servers provided by NTNU via VS Code's SSH remote development feature.

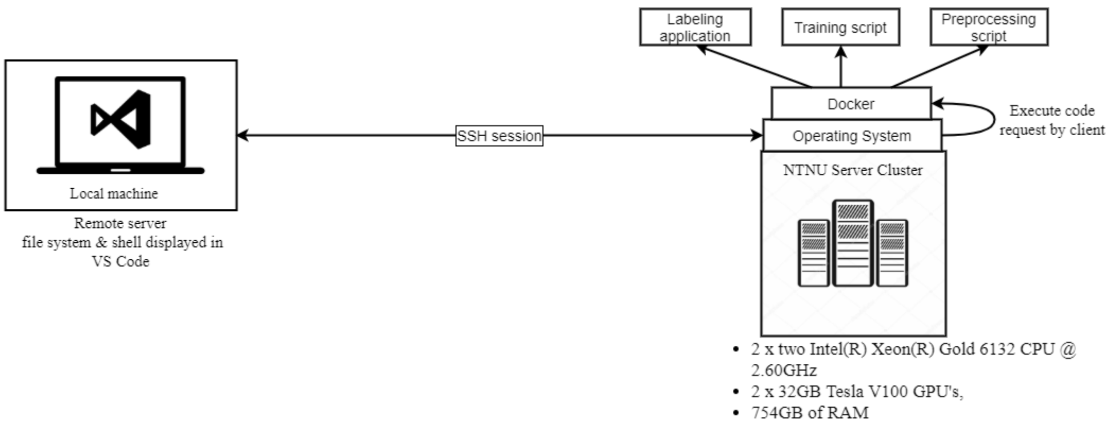

Labeling application
Training script
Preprocessing script
Docker
Execute code request by client
Operating System
SSH session
NTNU Server Cluster
Local machine
Remote server file system & shell displayed in VS Code
• 2 x two Intel(R) Xeon(R) Gold 6132 CPU @ 2.60GHz
• 2 x 32GB Tesla V100 GPU's,
• 754GB of RAM

Figure 24: The working environment illustrated.

3.1.3 Docker

We made use of Docker (see Section 2.17) to set up the software environment in the granted servers. This is practical as Docker provides a contained environment for installing software dependencies that is easy to set up and tear down, and thus avoids installing pollution in the borrowed servers, as requested by the server owners.

Additionally, if we had to switch servers, it would not be an issue because we could just run the Docker configuration files (Dockerfiles) on the new machine. In fact, this was precisely the case. The server providers requested us to migrate to another server cluster due to system restructuring. Thanks to Docker, this was more or less a painless transition. All we had to do was to copy all of our files to the new system, and build the environment with the Dockerfiles with a few simple commands. The process took us no more than a few hours in total.

Another reason for using Docker was that environment blueprints (Docker Images), tailored for our use case, were already written to a large extent. After only slight modifications, they were ready to use, and we could rapidly initiate a productive software environment. The re-used Docker images were either open-sourced, as with the case of FairMOT, or adapted from previous research efforts.

In total we utilized four different "dockerized" environments:

1. For hosting a CVAT application instance. Thus, the labeling step was centralized with user access over the internet. Enabled seamless collaboration.
2. For creating a Detectron2, GPU-enabled environment. Enabled us to perform common machine learning tasks such as pre-processing, training and evaluation.
3. For creating a FairMOT, GPU-enabled environment. For the same reasons as in 2.
4. For creating a YOLOV5 + DeepSORT, GPU-enabled environment. For the same reasons as in 2 and 3.

So, for example, if we needed to run a simple script that prepared a dataset for YOLOv5, we would simply execute the script in the context of the YOLO Image (that is, as a Container) and save the result to disk.

3.2 Dataset Creation

When working with automated soccer player tracking as the chosen domain, one quickly realizes the sparsity of publicly available datasets. Thanks to video streaming platforms like Youtube, which offer soccer footage in many different formats, such as with different lighting, angles, and so on, getting the footage is not the real issue. The issue arises when looking for *annotated data.* The matter is further complicated when the footage needs to be annotated with tracking data, which involves bounding box labeling of each player on the field, including ID tagging - for every single frame.

Numerous approaches to circumvent the dataset creation problem can be made. One approach, which we have included as an experiment, would be to use a pre-trained detector for finding people in combination with a non-deep tracking algorithm such as SORT. It effectively removes the need for domain-specific annotated training data because the detector is trained to detect humans. Soccer players are, after all, humans as well, so it would be fair to assume that this approach should be able to compete. Non-deep tracking algorithms such as SORT does not need any data in advance to perform the tracking step. They rely on traditional approaches to solve the data association problem, such as Kalman Filters [24] and the Hungarian algorithm [60]. Even though this approach does not need annotated data to create a practical algorithm, a dataset to compare with ground truths would still be required for quantitative evaluation, which would simplify comparison with other methods.

Hurault et al. [73] and Cioppa et al. [86] use a different approach where they automatically create their annotated datasets. It should be noted that these methods only label with detection data and not tracking data. That is, they omit the ID tagging part. They create the dataset with a distillation approach, where the knowledge of a "teacher network" is distilled into a "student network". More specifically, the teacher network is applied to unlabeled video footage. With various specialized techniques, the output is processed to fix noisy annotations, such as removing false positives and adding false negatives. The final result is an annotated dataset that can be used to train the Student network. Typically, the teacher network is much more complex and robust than the student network.

A third approach would be to use a tracking system pre-trained on people, and then *finetune* it with small amounts of manually annotated data. In contrast to the first approach, some labeling and, consequently, optimization has to be done. Nevertheless, it would require less annotation work than when training from scratch to achieve the same results. This is more or less the approach we used. It was chosen based on the fact that manual labeling is the most precise way of creating non-noisy labeled data, and we wanted the tracking system as accurate as possible. In Section 3.2.5, we propose a semi-automatic pipeline that alleviates the time-consuming process of manual labeling, enabling us to produce more annotated data in less time.

As Ranheim TF partially initialized the thesis, we received footage of their team playing in training matches and in practice scenarios. The footage comes with different angles, lighting environments, and weather. For reference, we will call it the *Ranhiem dataset.* Additional unlabeled data was retrieved from Pettersen and Johansen [87], which is high-resolution footage of matches played in Alfheim stadium with a complete view of the field. We will refer to the latter as the *Alfheim dataset.* Both of these footage sources were subject to our labeling process, which is described below. As noted, minimal amounts of annotated data existed. Fortunately, we were able to retrieve a good chunk of data from D'Orazio et al. [76], which the author named, and referred to as the *ISSIA dataset.* Besides being in a peculiar XML format that needed conversion and pre-processing, the data fitted fine for our experiments. Particularly, the pre-annotated dataset was labeled with multiple object tracking tasks in mind. That is, players were labeled with unique IDs. We also retrieved some unlabeled footage with decent angles and environmental conditions from Youtube. These were used for qualitative testing and will thus be referred to as the *test set.* The retrieved datasets will be further described in the first few subsections, including; dataset format, use cases, and troubleshooting.

In the rest of the subsections that follow, we describe the steps taken in order to prepare the data for the Experiments in Section 4. That is, we provide an overview of the process and a description of the tools used to realize it. We also detail a method of effectively splitting the annotated data into a training and evaluation set.

3.2.1 Ranheim dataset

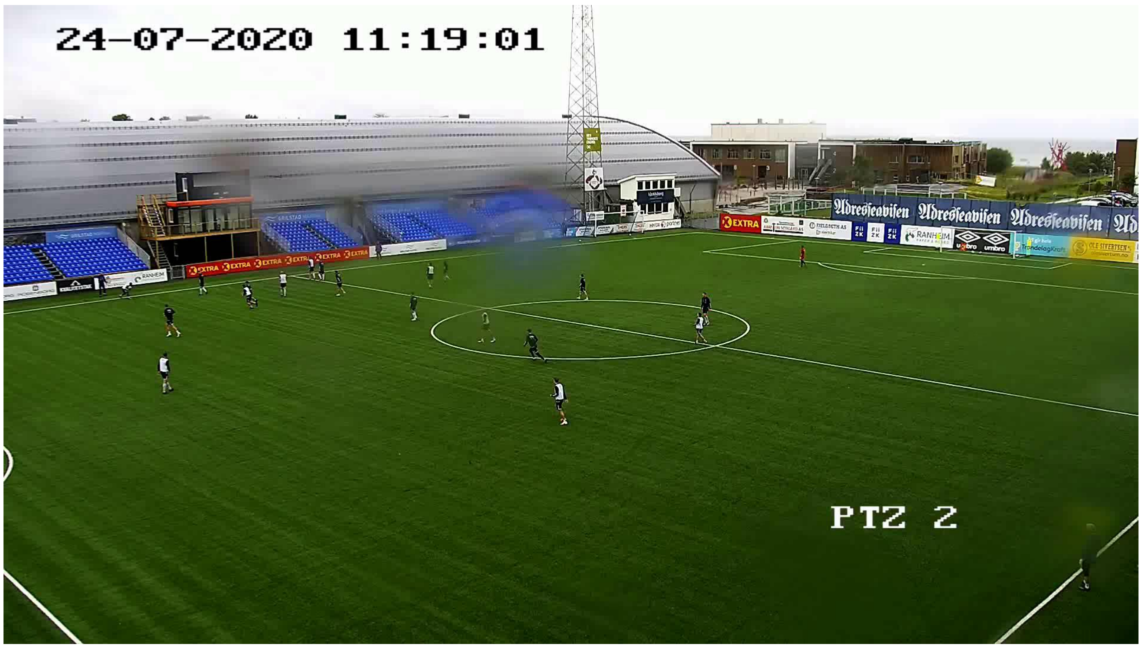
The footage recieved Ranheim TF was the first data that was subject to our labeling process. We received video footage from three different soccer matches, separated into different video files. Snapshots from each video can be seen in Figure 26. Each match had a playtime of about 15 minutes. The resolution is reasonably high, though the color contrast could be sharper. Each video varies in context; two videos capture training matches on a full pitch, and the last video capture a training exercise with a reduced pitch size.

The videos are filmed with a high-angle, skewed-from-the-corner view. This presents several challenges. From such an angle, player occlusion occurs at frequent intervals, often with more than two players involved. For instance, in an event such as a corner kick, multiple players may find themselves in a tight spot with many occluding one another. If, in addition, the camera is placed on the opposite side of the free-kick event, the scenario might prove itself incredibly challenging. An example of this can be seen at the bottom illustration of Figure 26. The lens has a partial view of the field where the closest goal is out-of-view. This means that, more often than not, the keeper guarding this goal is also out-of-view. Also, any event taking place on this side of the field will thus be undetectable.

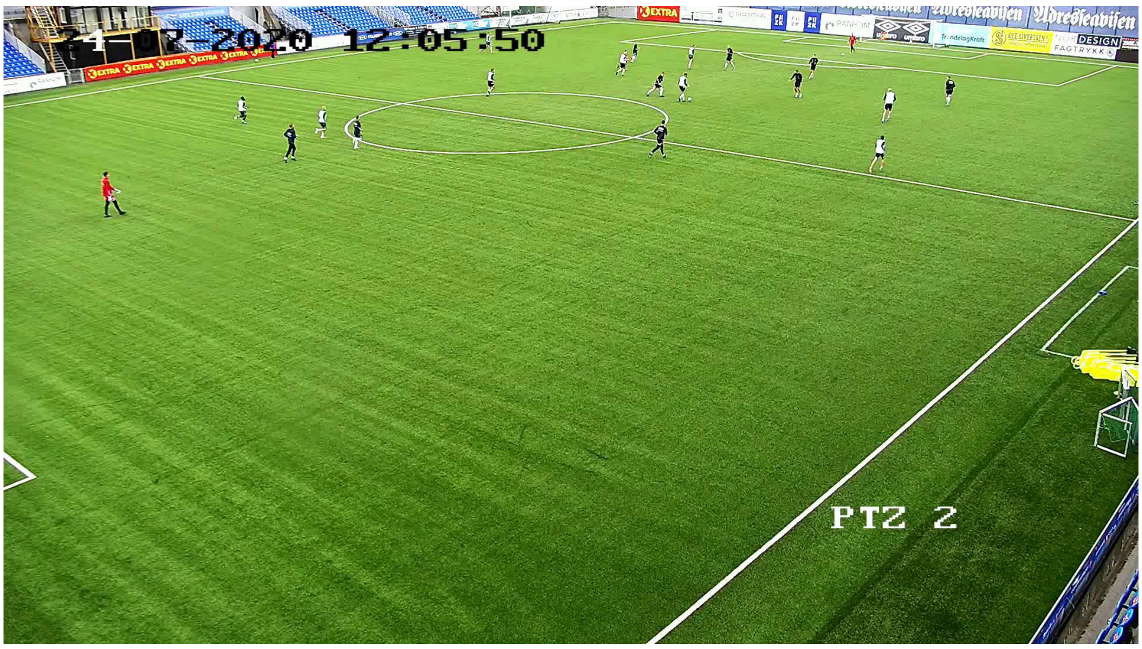
Light conditions are fairly consistent between the videos, considering everything was shot during the daytime. There are slight variations, however, due to environmental changes such as rain and overcast. Unfortunately, in rainfall cases, droplets can occur on the lens and partially obscure the view, which certainly does not make it less challenging. See the first image of Figure 26. Non-player humans on the footage include coaches and bypassing patrons. A summary of details of the Ranheim dataset can be found in Table 2.

	Ranheim dataset
Image resolution	1920x1080
Format	Traditional Linear
Framerate	25 FPS
Viewpoint	From the corner, skewed, high angle
Pitch coverage	Partial: nearest goal missing
Color scheme	Normal
Light conditions	Daylight
Environment	Sunny, rain, overcast
Supporters	No
Other non player humans	Coaches, benchwarmers, casual spectators
Number of videos retrieved	3
Camera movement	None
Average duration per sequence	15min

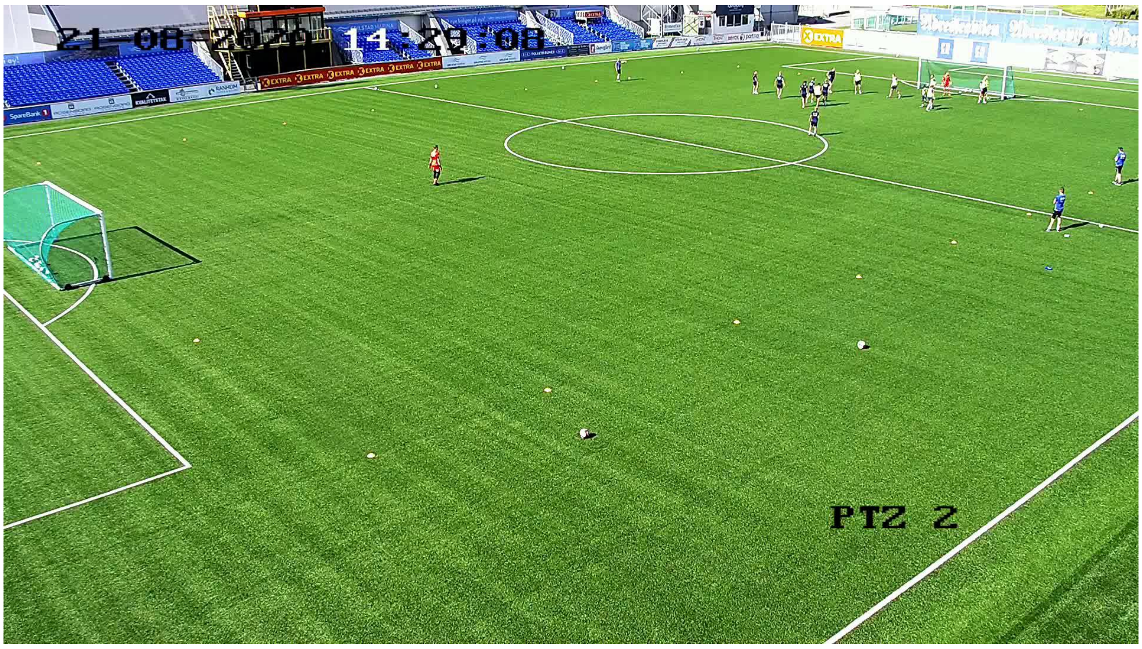
Table 2: Details of the unlabeled video footage received from Ranheim TF.



From the first video. Note how the overcast is influencing light conditions. Rain cause droplets to form on the lens.



From the second video. Normal light conditions. Note how the goal nearest to the camera is out of view.



From the last, and third video. Sunny weather provides good light conditions. Densely populated events in front of the goal cause severe occlusion scenarios.

Figure 26: Snapshots from each of the three videos provided by Ranheim TF.

3.2.2 Alfheim dataset

The footage retrieved from Pettersen and Johansen [87] was the second and final data that was subject to our labeling process. The video footage is different from the Ranheim dataset in pretty much every aspect. This is a good thing, as we want as much variance as possible in our combined dataset to create as robust methods as possible. The footage is of quality, offering 4k resolution with high contrast colors, making it easier to distinguish players from the background. The camera view is constructed from five 2K-camera arrays, placed high and central in the stand. It creates the high-angle panorama view of the entire pitch seen in Figure 27. Due to the camera setup, the video has a “fisheye lens”-effect, i.e., the video appears to curve. Even though the footage is shot during nighttime, the use of high-quality floodlights prevents this from being an issue, offering surprisingly good contrast colors to the scene. Again, and not surprisingly, challenges could arise when players move and possibly cluster towards the opposite side of the camera lens.

There are videos from three different games publicly available, filmed using various camera setups. One of the games was filmed with the 2K-camera array mentioned above, giving it an unobstructed view of the entire pitch. It was also the video we decided to retrieve, with the entire pitch-view being an attractive attribute in terms of tracking. The video had a duration of just below 40 minutes.

The details of the footage retrieved from Pettersen and Johansen [87] is summarized in Table 3.

	Alfheim dataset
Image resolution	4450x2000
Format	Curved panorama (fish-eye)
Framerate	25 FPS
Viewpoint	From the sideline, centered, high angle
Pitch coverage	Full coverage
Color scheme	High-contrast
Light conditions	Night + floodlights
Environment	Night
Supporters	Yes
Other non player humans	Coaches, benchwarmers
Number of sequences retrieved	1
Camera movement	None
Average duration per sequence	40min

Table 3: Details on the footage retrieved from Pettersen and Johansen [87]

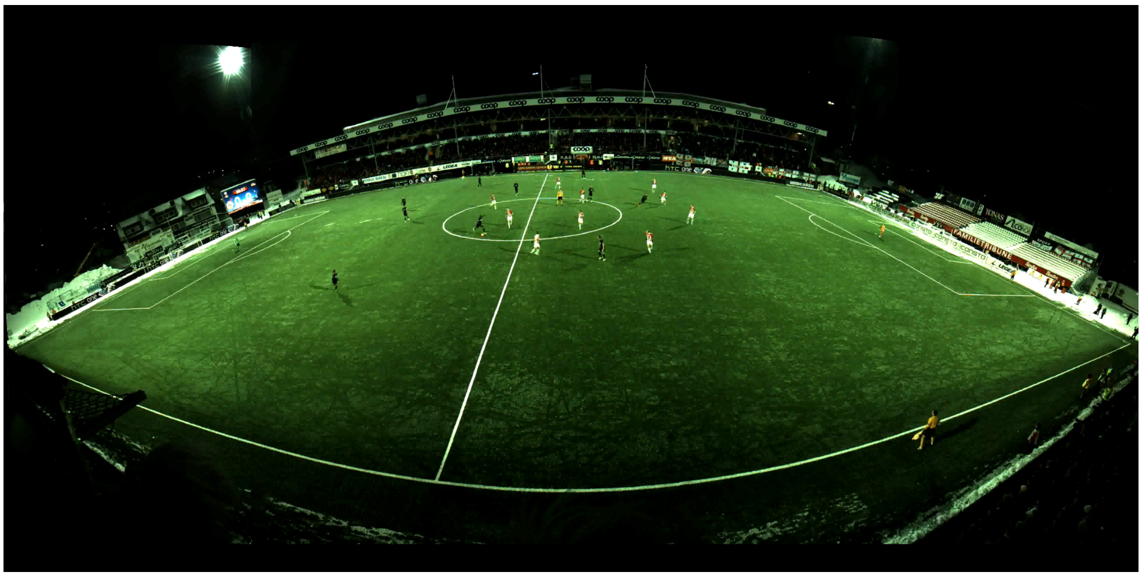


Figure 27: A snapshot from Alfheim stadium, retrived from Pettersen and Johansen [87]

3.2.3 ISSIA dataset

With the shortage of soccer tracking datasets publicly available, the ISSIA dataset from D'Orazio et al. [76] was the only pre-annotated dataset we could find. There are others available, but most in a format not compatible with our methods.

A total of 10 minutes of player tracking data from the ISSIA dataset was compatible with our work. However, in order to make use of the data, we first had to convert it. Our target format was the COCO annotation format [51], primarily because this is the format of the annotation tool we used (see Section 2.16). However, we also found this format to be quite intuitive. The source dataset was packaged in a complicated XML format specifically designed to work with the video annotation tool used in the work of D'Orazio et al. [76]. We discovered that the annotations were wrongly associated with frames too far ahead in the sequence. The annotations had to be "dialed back" for 6-8 frames to synchronize them. In addition, the 300-400 first frames of each video did not contain annotations. We removed these to avoid noise in our final dataset.

The ISSIA dataset is comprised of six videos capturing two elite soccer matches with three cameras. Thus, each video corresponds to a single camera capturing either of the two possible matches. Three of the videos, or cameras, jointly capture the entire field of a game. Figure 29 shows snapshots of how the three cameras cooperatively capture a single match. Each video has a duration of about one and a half minutes. However, even if the dataset holds tracking data for the entire pitch, appropriate combining of the data is not trivial. Positional data was only provided concerning the camera that captured it and did not consider the other cameras or the entire pitch. Player IDs, however, were kept between cameras capturing the same game. To our knowledge, there was no additional data on how to synchronize the cameras, or their corresponding annotations, neither temporally nor spatially. Without such data, the task becomes frighteningly complex and likely very time-consuming. We decided to treat each video and corresponding annotations as provided; as an independent sequence — for simplicity.

Due to the videos only having a static, third-part view of the field during the entire duration of the game, players leaving and entering the scene is widespread. As mentioned earlier, it is expected to introduce additional challenges and complexity for the chosen methods. Since four videos have a view of the part containing the goal, it is expected that there will be less occurrence of objects here than in the videos having a view of the middle part of the pitch. The reason is that there is usually more player traffic and events on this part of the pitch, likely due to player formations and transitioning between attack and defense formations.

Otherwise, we can see from Figure 29 that the videos provide reasonably high resolution and decent light conditions. Colors are, as in the case with the Ranheim dataset, a bit bland and nowhere near the high contrast footage that Pettersen and Johansen [87] provides. An advantage is that the footage contains very few non-field objects. The camera frames primarily consists of the field and little else, such as background environments, supporters, and stands. It lowers the chance for false positives and provides larger players to detect. A summary of the video properties is provided in Table 4.

	ISSIA dataset
Image resolution	1920x1080
Format	Traditional Linear
Viewpoint	From the side line, high-angle
Pitch coverage	Partial
Color scheme	Normal
Lighting conditions	Daylight
Environment	Overcast
Supporters	No
Other non player humans	Coaches, bench warmers, spectators, camera men
Number of videos retrieved	6
Number of matches	2
Number of cameras per match	3
Camera movement	None
Average duration per video	1.5 min

Table 4: Details of the annotated video footage retrieved from D'Orazio et al. [76]

First camera of the first game.

Second camera of the first game.

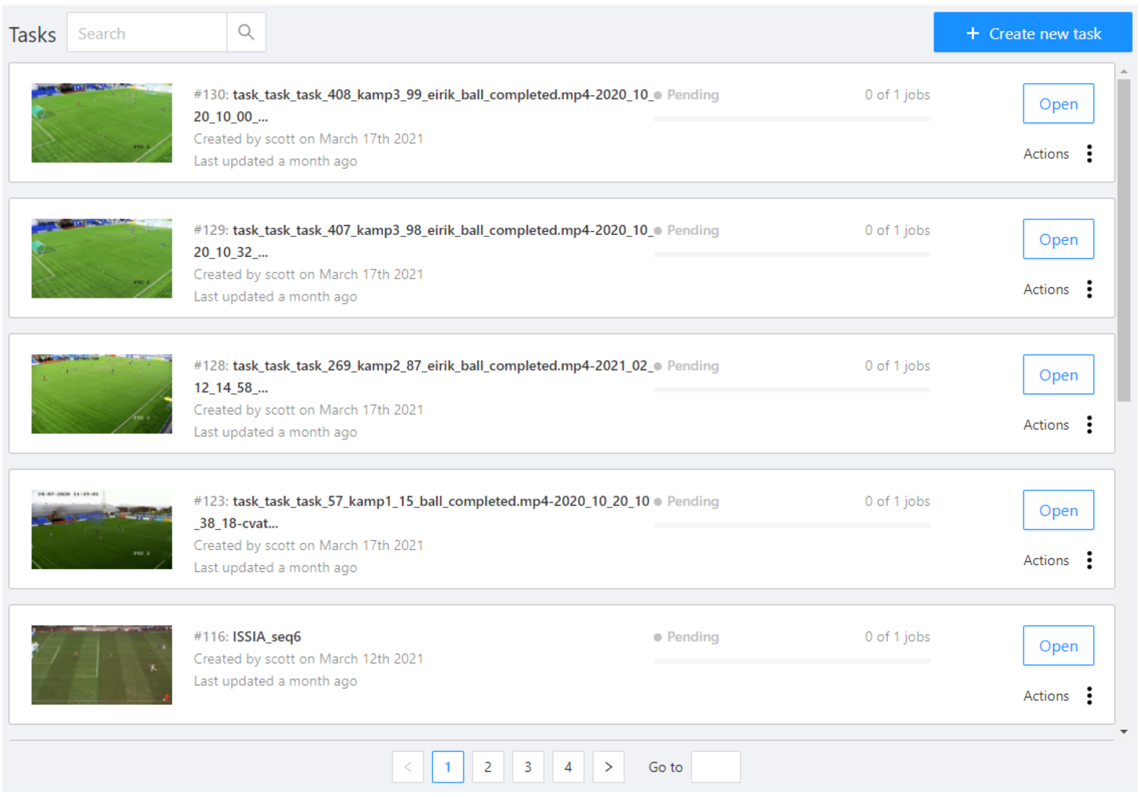
Third camera of the first game.

Figure 29: Snapshots of three videos from the ISSIA dataset.

3.2.4 Annotation process

The annotation tool of choice was the CVAT Application described in Section 2.16. It was set up according to the working environment described in Section 3.1, using the remote accessed machines that NTNU provided and Docker. The annotation software was made available online. Consequently, the tool was easily accessible from anywhere at any time. This enabled us to do collaborative labeling, which proved beneficial in terms of workflow.

As mentioned, the Ranheim and Alfheim dataset was selected for labeling (the ISSIA dataset was pre-annotated). Uploaded videos are in CVAT encapsulated in *tasks* which represents a unit of annotation work. Unfortunately, a task can not be shared between users, and annotations have to be completed before downloading it. This is troublesome when the length of the videos is a couple of minutes each, requiring a massive amount of labeling before using it. As a workaround, the larger video files were segmented into chunks of 250 consecutive frames. However, with the many chunks created, manually uploading each file proved to be very time-consuming. Luckily, CVAT came bundled with a useful Command Line Interface (CLI) that lets users interact with a deployed CVAT application. This enabled us to create another script that automated uploading a batch of video files as separate tasks. Conversely, the script uploaded all the newly created video chunks as individual tasks on the server, ready for labeling. The result is depicted in Figure 30.


Tasks Search + Create new task

#130: task_task_task_408_kamp3_99_eirik_ball_completed.mp4-2020_10_20_10_00_... ● Pending 0 of 1 jobs Open
Created by scott on March 17th 2021
Last updated a month ago
Actions

#129: task_task_task_407_kamp3_98_eirik_ball_completed.mp4-2020_10_20_10_32_... ● Pending 0 of 1 jobs Open
Created by scott on March 17th 2021
Last updated a month ago
Actions

#128: task_task_task_269_kamp2_87_eirik_ball_completed.mp4-2021_02_12_14_58_... ● Pending 0 of 1 jobs Open
Created by scott on March 17th 2021
Last updated a month ago
Actions

#123: task_task_task_57_kamp1_15_ball_completed.mp4-2020_10_20_10_38_18-cvat... ● Pending 0 of 1 jobs Open
Created by scott on March 17th 2021
Last updated a month ago
Actions

#116: ISSIA_seq6 ● Pending 0 of 1 jobs Open
Created by scott on March 12th 2021
Last updated a month ago
Actions

< 1 2 3 4 > Go to

Figure 30: Videos where divided into smaller chunks and uploaded as individual annotation tasks on the CVAT server instance. This effectively reduced the amount of work required to yield an annotated sequence that could be immediately included in our dataset. A user may be working on several tasks at a time, in which case it is displayed to other users. This avoids multiple users working on the same task, which is not supported. The combination of smaller jobs and multiple users working on mutually exclusive units of work ensures high throughput of labeled sequences.

The process consists of manually placing a bounding box around each player and possible coaches and referees for a given frame. Each bounding box is also tagged with an ID to keep track of an object during the entire sequence. The process is then repeated for each frame of the video segment. The process of labeling a single frame is depicted in Figure 31. CVAT provides a helpful interpolation feature, making it possible to annotate several frames simultaneously, given that the player moves somewhat linearly and at roughly the same speed. Even considering the features CVAT provides to speed up the labeling, the overall process is still time-consuming. In the following subsection, we detail an approach that removes a lot of the labeling efforts otherwise required and nicely integrates with the features CVAT provides.

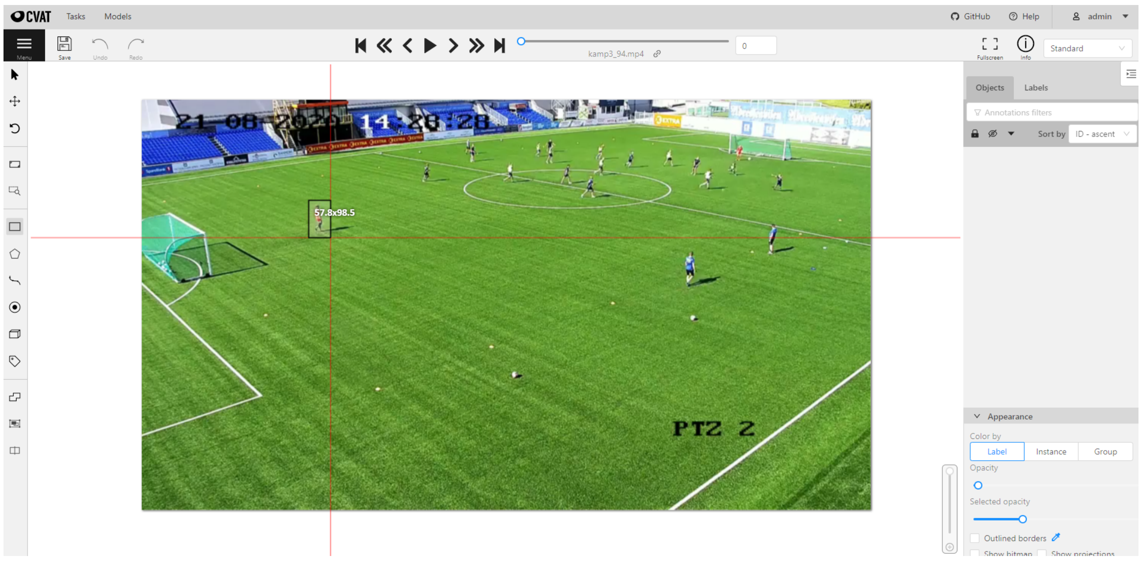


Figure 31: A snapshot of the labeling process. We used the marker to place boxes around the players and labeled each with an ID. The process is then repeated for each frame in the sequence. The same ID was used for corresponding objects between frames.

A task was marked as completed once each of its 250 frames was annotated. Again, we utilized the CVAT CLI to create a script that automatically downloaded all the tasks that were recently completed. This proved very useful; when re-engaging in annotation work, we could pull completed tasks from the server and work with an updated dataset. CVAT supports downloading completed tasks in the COCO annotation format, which it nicely integrates with Detectron2's interface. The format is quite intuitive: Each frame is a JPEG image file and has an accompanying annotation file in JSON format. Each annotation is represented with pixel coordinates.

After downloading a completed task, the annotated sequence was packaged as 250 images with an annotation file that localizes ID-tagged players on each image. The fact that the ID is retained throughout the sequence is an essential prerequisite to train the re-identification models used in our second and third experiments. If some of the IDs were to change, player objects would have had ambiguous target IDs, which would obviously hurt training and performance.

3.2.5 Semi-automatic Labeling

As one can imagine, labeling video sequences with MOT annotations can be incredibly time-consuming. Consequently, *here, we propose and detail the use of a semi-automatic labeling approach for MOT annotations.* Specifically, we wanted to use our chosen deep learning models to annotate the data for us. Initially, we would use pre-trained models and apply them on unlabeled videos to produce partially correct annotations. The annotations are then manually corrected, and the result is integrated into an ever-growing dataset. Finally, we use the updated dataset to finetune the models we just used to annotate. This process is repeated until we have enough data or the model is adequately robust. The concept is depicted in Figure 32. The idea is simple enough, but implementation-wise it required some carefully designed solutions to the following considerations:

1. How do we correct the noisy output from the models?
2. How do we integrate the corrected annotations with the existing dataset?
3. How do we integrate the models with the unlabeled data stored on the CVAT server?
4. How do we train the models on the updated datasets?
5. How do we update existing models with models trained on updated datasets?

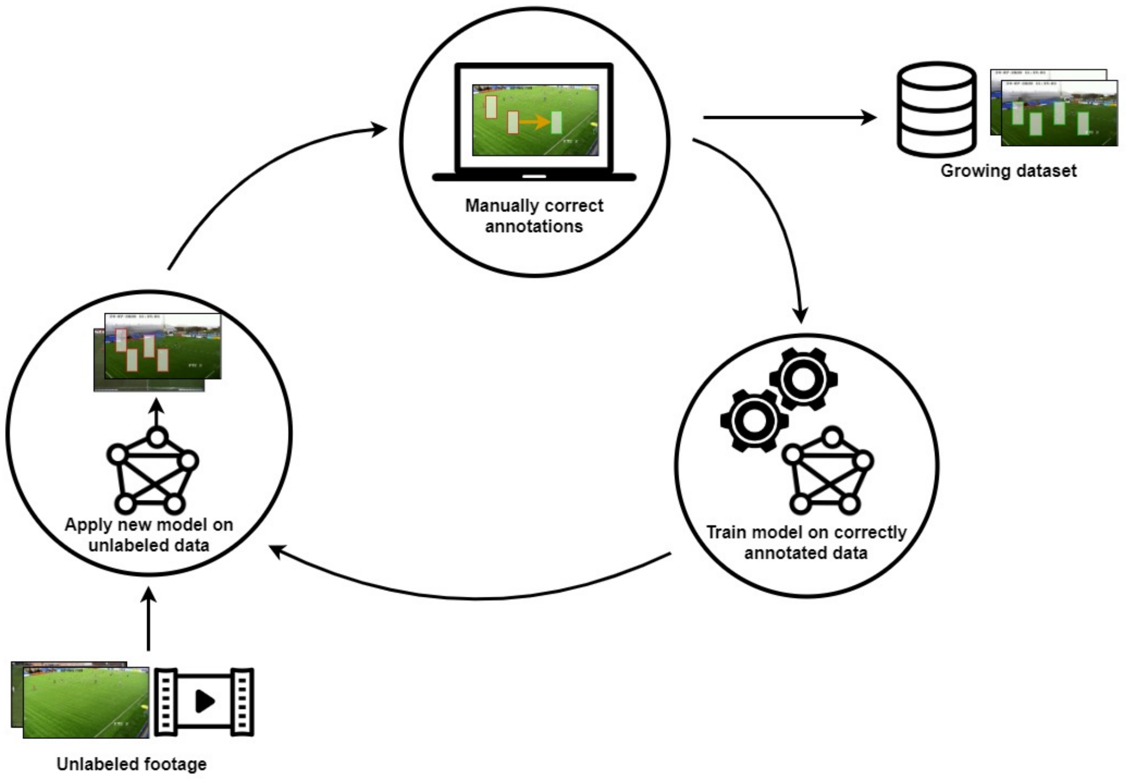

Manually correct annotations
Growing dataset
Apply new model on unlabeled data
Train model on correctly annotated data
Unlabeled footage

Figure 32: The circular, iterative process of semi-automatic labeling, specialized for our tools and workflow. First, left step: tracking model is applied to unlabeled video footage and produces noisy annotations for each frame. Initially, we use a tracker pre-trained on humans. Second, upper step: Noisy annotations are manually corrected in CVAT. The result is added to the dynamic dataset. Last, right-step: Correctly labeled data are used to train the last iteration of the model to produce a new iteration.

To answer the first, second, and third questions, the CVAT annotation tool seemed to be the solution. We had already structured our working environment such that we automatically pulled annotated sequences from the CVAT server and included them in the dataset. Additionally, chunks of unlabeled sequences were already on the server, packaged in tasks ready to be annotated. It seemed perfect if we could somehow apply the models directly to individual tasks on the server and then display the result in the editing page of CVAT (see Figure 31). The automatically generated labels could then be manually corrected by using CVAT's intuitive user interface. Surprisingly, this feature seemed to be in the development stage of the CVAT project. However, it only provided support for using the built-in models, which consisted of light-weight, CPU-only implementations such as YOLOv3. In addition, only object detection was supported.

The built-in model support is implemented by having the server-side of CVAT execute HyperText Transfer Protocol (HTTP) requests with byte-encoded images as payload. The endpoint, which is *serverless function* extensions of the detection algorithms, is then run on the received image, and the resulting annotations is sent back as a response to the CVAT server. The server then parses the response and stores the annotations for the respective image in the database. When viewing the image in the editing page of the tool, the annotations are overlayed and subject to editing. It works fine for object detection tasks. Each request uses a payload of a single image, which is everything the function needs to compute a stateless response.

However, in the case of multiple object tracking, the matter becomes more complicated. Serverless functions adhere to the principle of *referential transparency.* They do not store any state between function invocations, and the result is purely computed from the function call's arguments. This is contradicting to the methods we deploy. Since the object detection part is essentially a complicated mathematical function, they are pure. Hence, they are stateless. However, the association component of a tracking system is not. When the association component receives bounding boxes, it uses existing tracks (state) to assign IDs. Accordingly, for the serverless function to create tracking data for a particular frame, it needs to have the full context of the previous frames in the sequence passed as a parameter. The naive approach would invoke the serverless function with the entire sequence instead of individual images. This is a practically inefficient approach as the payload would become extremely large. The resulting request-response cycles would take a long time, possibly halting either of the two endpoints. Instead, we propose a different approach that still invokes the serverless function with single images. However, we add a second argument that encapsulates the state of the tracking system. The proposed and adopted solution is illustrated in Figure 33.

Instead of sending an entire sequence for the endpoint to process, we keep function calls lightweight by requesting annotations for a single image at a time. Providing historical context to the tracking system in the receiving end is circumvented by introducing a state object. As contrary as this sounds, it works because the state is never actually stored anywhere. It is continuously passed between the CVAT server and the serverless function as part of request and response transmissions. The state object is essentially the tracking system's association component. The object detection model is kept with the serverless function, effectively reducing the payload size of the state object to a tiny chunk of data.

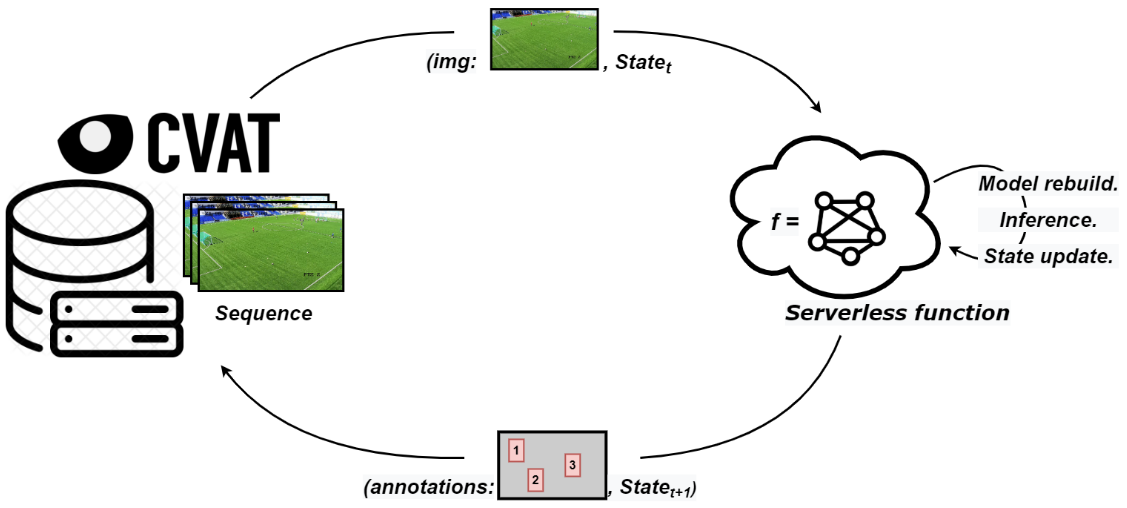


Figure 33: Overview of the implementation solution for semi-automatic labeling in the CVAT application. The tracking system is implemented as a serverless function. The CVAT application requests annotations for individual images. It does so for every frame in a sequence, sent in order. A state object is introduced as part of the payload in request-response transmissions to circumvent the referentially transparent nature of serverless functions and the contrary fact that tracking systems are stateful. It encapsulates the history needed by the serverless function to perform online tracking on the received image.

The state object starts its life cycle on the serverless side when a request for the first frame of a sequence is made. It is indicated by the state being a NULL object. From there, it creates a new and complete tracking system, and the image from the request is passed through it. In order to create a response, a few preparations are in order. First, the detection algorithm needs to be detached from the tracking system and is discarded. Second, the association component has to be serialized and encoded for transmission. Third, the annotations are formatted to a JSON, response-friendly format. Finally, annotations and the tracking system are bundled together to form a response object, ready to be sent back to the requesting server.

On the CVAT server-side of the cycle, the perspective is quite simple. It just keeps requesting annotations for each frame in a sequence, sent in the correct order. However, it needs to add the state received from the last invocation to the payload.

3.2.6 The Dataset Split

Once there was a sufficiently large dataset, the next order of business is to split the dataset into mutually exclusive sets called the *training, validation, and test sets.* In this thesis, *only the validation set, and not the test set, have annotated data.* The main reason is that annotated data was scarce, and we wanted to make the most out of the data available. The reader should also bear in mind that we are not trying to benchmark against other works simply because no dataset exists to support it. Consequently, we argue that metrics from the validation set provide sufficient data to compare the methods in this work. The test set is not annotated and is further described below. The test set provides additional support for the comparison of the different models.

As mentioned in Section 3.2.4, annotated sequences came in chunks of labeled frames. Accordingly, the training and validation split would have to consist of several such chunks. Having both sets consist of entire sequences is essential. If one used randomly drawn images from the annotated dataset, images in the sets could be extremely similar due to neighboring frames in a sequence being almost identical. This would first and foremost lead to a validation set incapable of assessing generalization capabilities. Secondly, having entire sequences in the validation set is necessary to assess tracking performance.

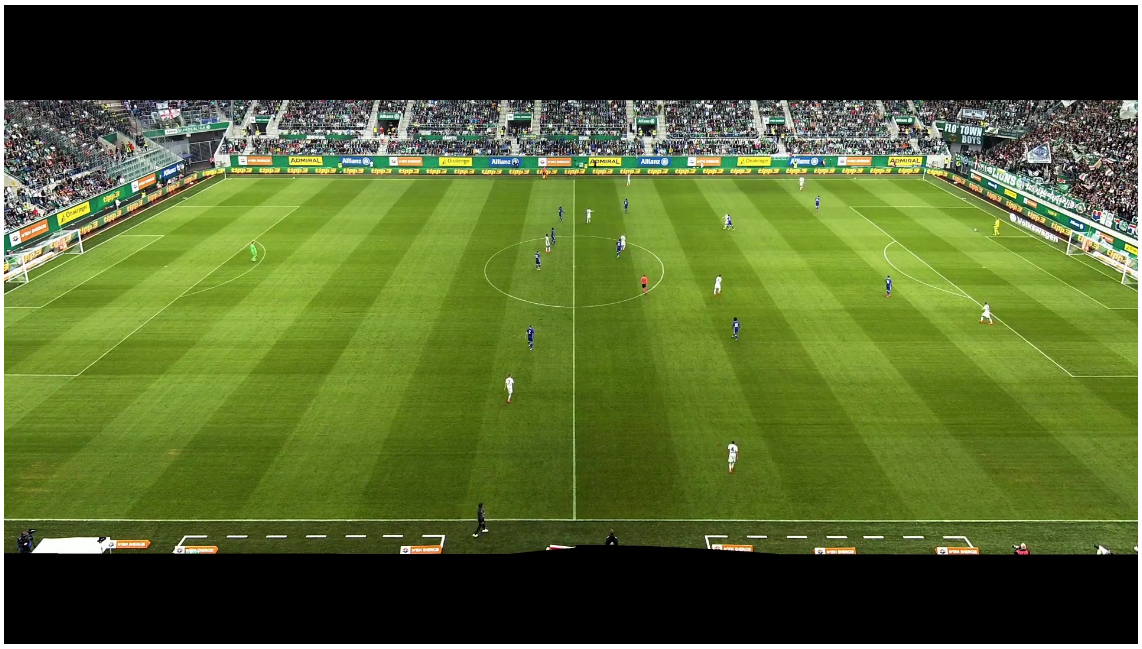
Considering a generally limited dataset size, structuring the validation set becomes fairly important. It needs to have a sufficient size to reduce the probability of "having luck" with sequences evaluated. It also needs to incorporate enough diversity to create a sufficient representation of real-world use-cases. At the same time, we would not want to use too much of the available data that would otherwise be used in training and creating adequately robust models. Also, when the number of total validation sequences becomes scarce, it is accordingly vital to choose rich sequences because they contain many objects in several positions and poses. The result from the dataset creation is listed and further discussed in Section 4.1. The models in both of the experiments used the same training and validation set. Using the same validation and test set is of essence to make a fair and reasonable comparison of the two models.

3.2.7 Additional Test Set

We retrieved three additional soccer footage videos for qualitative testing purposes. These are handpicked YouTube videos that we found to offer good showcasing qualities for the various experiments in this thesis. For instance, they offer decent angles with a good view of the field, relatively high resolution, and decent colors. They are regarded as representative of the type of data these models should work with. The intention for these videos was to test model generalization abilities further. A frame from each of the videos is displayed in Figure 35.

Snapshot image of the first YouTube video [88]

Snapshot image of the second YouTube video [89]



Snapshot image of the third YouTube video [90]

Figure 35: Snapshot samples from the YouTube videos retrieved and used for qualitative testing purposes.

3.3 Architectures

Following environment setup and dataset creation, the natural next step is applying multiple object tracking algorithms to the data acquired. To that end, we have selected three different tracking architectures. They belong to the family of *online* tracking algorithms, and following recent advancements, we only deploy online trackers, some of which relying on deep learning-based methods. Consequently, the architectures will be illustrated similarly as the general architecture of online MOT trackers shown in Figure 14 (Section 2.9). In the following sections, we explain each in order, including implementation details, rationale, and further considerations. The architectures will form the basis for our experiments in Section 4.

3.3.1 Architecture 1: Detectron2's Faster R-CNN + SORT

Being the first architecture of choice, we chose an approach that had the following desirable qualities:

- Easy to implement and deploy.
- Relying on tried-and-true methods.
- Conceptually intuitive.
- Based on recent advancements in CV techniques.

Having such features allows for rapid experimentation and providing a basis for comparison.

The choice fell on a *detection-first track-second* approach, arguably the most prevalent setup for MOT systems [32, 91]. It works by having an object detection algorithm compute target positions, simply a list of pixel coordinates relative to the input image. The output is then forwarded, in an online fashion, to a separate association model that classifies each object with an ID. Thus, the system is entirely realized by the association component *on top of* which handles all object tracking logic. The idea is recognized in Figure 36, which is an architectural overview of the adopted system. This approach is desirable because it yields highly competitive results, despite its simplicity. For example, because of its loosely coupled structure, components can be selected according to the latest and greatest advancements within their respective fields, continuously having an up-to-date performing model. Further, the detection and association modules can separately be optimized and still achieve a system-wide increase in performance [91]. For detection, we use Faster R-CNN, and for the *association* step, we use SORT. We first describe the detection part.

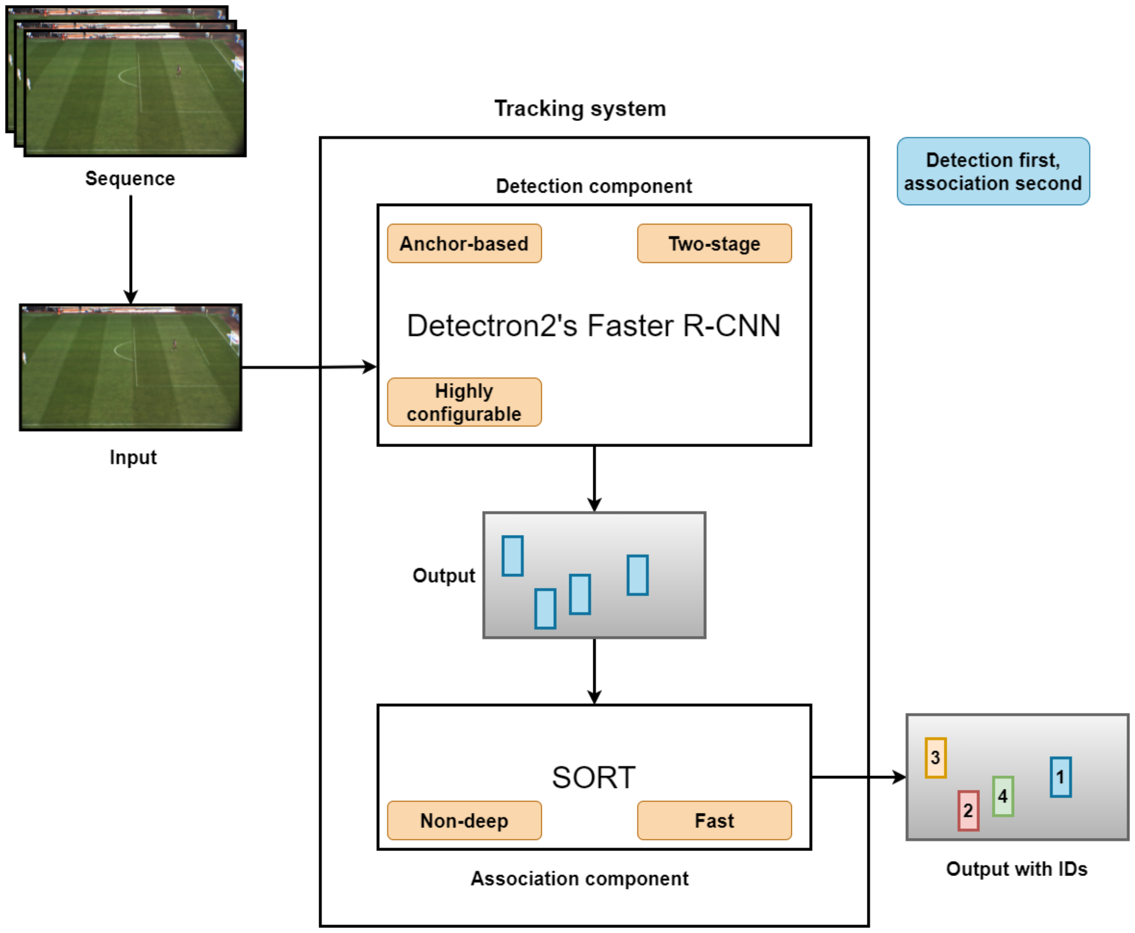


Figure 36: Architecture overview of Detectron2's Faster R-CNN implementation with SORT on top. The detection and association component is treated as black boxes for simplicity. Refer to Section 2 for details on implementation. However, key features of each component are listed in the yellow and blue labels. Faster R-CNN transforms the input into a set of bounding boxes. These, in turn, will be the sole input for the SORT algorithm, which associates and labels them with existing IDs.

Detection Component

The Detectron2 library provides a Faster R-CNN implementation ready to use with minor configuration. The detector is well known for being an established contestant within the family of CNN-based object-detectors, yielding competitive results [92]. Additionally, Faster R-CNN was also the object detector of choice by the authors of the SORT tracking algorithm, which is also our intended tracking method. Detectron2 enables several configuration options for Faster R-CNN, including switching backbones and whether or not to use feature aggregation techniques such as a Feature Pyramid Network (FPN) [42]. The chosen configuration was based on having an appropriate trade-off between detection accuracy and inference speed.

Association Component

SORT is a popular choice for multiple object tracking in real-time applications. Since its release in 2016, it has consistently maintained its presence in the MOT Challenge charts [93]. This is especially impressive when considering its simple implementation and high throughput. Furthermore, the code-base for the original SORT paper is publicly available, so integration with our work required minimal overhead. An interesting caveat with SORT is that it relies on traditional statistics and optimization algorithms to perform the association step. That is, unlike many of the methods described in this thesis, it is not a machine learning algorithm. Accordingly, SORT does not require any training. The detection part mainly introduces the variance of the performance in the overall system. The exception is the few manually adjustable parameters of SORT, such as overlap thresholds and the maximum number of initialized tracks. Also, since the only trainable part of the system is Faster R-CNN, only data labeled with object positions (and not IDs) is required to optimize the entire system.

3.3.2 Architecture 2: YOLOv5 + DeepSORT

For the second experiment, we replaced the two-stage detection architecture Faster R-CNN with the single-stage detector YOLOv5 and substituted the traditional, non-deep tracker SORT with its deep-learning counterpart DeepSORT. Thus, both our first and the second architecture used a detection-first, track-second approach.

The choice between YOLOv5 and YOLOv4 was not trivial, and there was definitely a need for investigation before picking one. As both networks deliver state-of-the-art results and are backed by different parts of the CV community, the choice was largely based on preference. The fact that YOLOv5 was developed in PyTorch was an incentive, due to its practicality and ease of use. The other architectures were also PyTorch implementations, so it was familiar. With this in mind, the choice fell upon YOLOv5. However, we feel confident we would have reached highly usable results with both models.

The YOLOv5 project is open-sourced, and the code-base is available on GitHub [52]. To make the initial setup process more streamlined, we decided to make use of an already implemented YOLOv5 and DeepSORT combination [94]. The code-base used a DeepSORT version written in PyTorch as well. With the necessary installation requirements in one place, it was a smooth task to containerize the entire project in Docker.

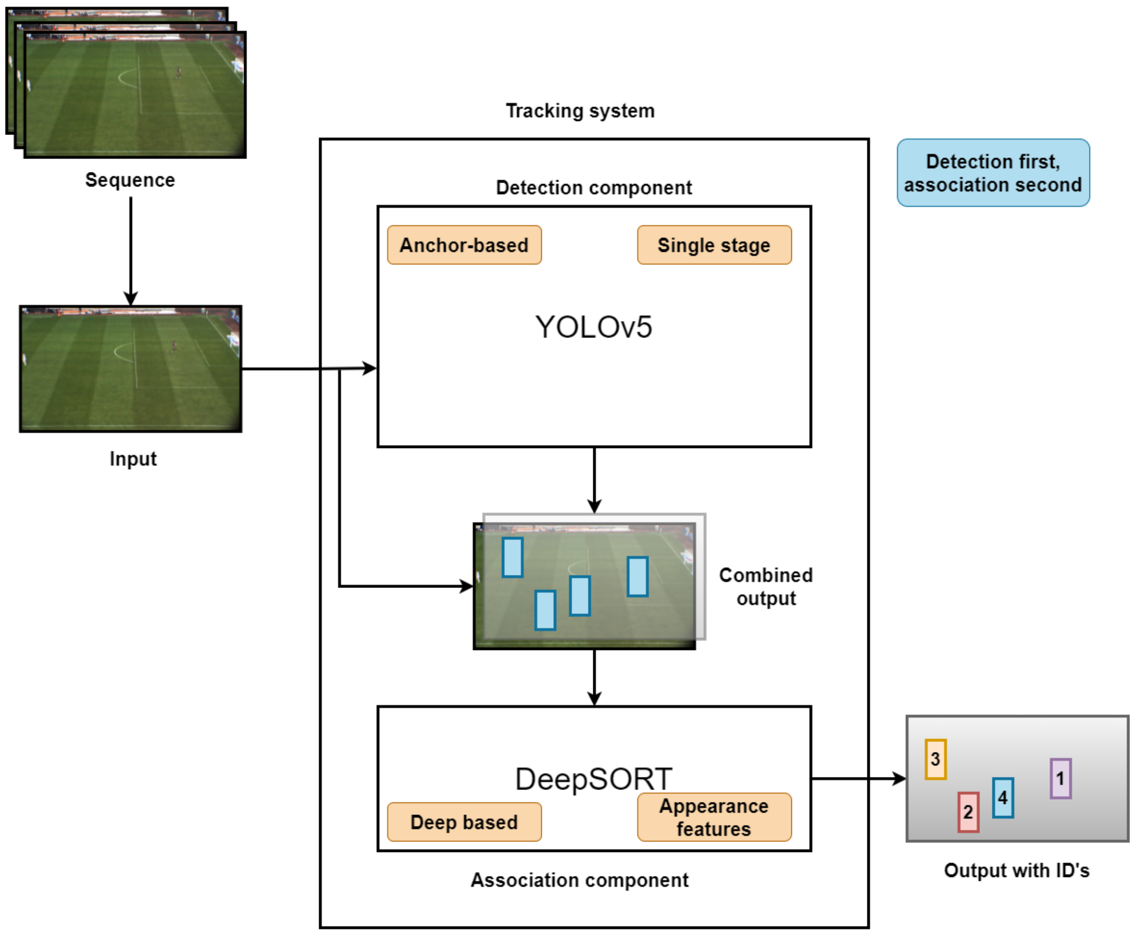


Figure 37: Architectural overview of YOLOV5 + DeepSORT. Note how DeepSORT requires both the original input image and the box proposals. It can then crop out areas of interest and extract deep appearance features that better inform the association task.

Detection component

YOLOv5 offered eight different models to choose from in its latest release [95], with different architecture sizes, speeds, and input image resolutions. The smallest and fastest model, named YOLOv5s, contained 7.9 million parameters, whereas the largest and most robust model, named YOLOV5x6, contained 141.8 million parameters. Four of the models were pre-trained on a 640x640 resolution, while the remaining four were pre-trained at 1280x1280. The speed differences between the models were significant. According to the YOLOv5 creators' test results, YOLOv5s delivered a top-speed of 500 FPS with a 640x640 input size. YOLOv5x6 had a top speed of 45 FPS with a 1280x1280 input size. This speed-to-accuracy trade-off was worth investigating, considering the superior detection results of the larger models and the superior speed of the faster models. Additionally, the difference in input resolution was an important complexity concern.

After carefully considering the different options, we decided upon proceeding with the models YOLOv5x and YOLOv5l6. Both of these models showed relatively high AP scores while remaining reasonably fast. Moreover, YOLOv5x was pre-trained on 640x640, and YOLOv5l6 was pre-trained on 1280x1280. Thus, it allowed us to test both resolutions and whether this would significantly impact the final performance.

We choose to use image size 1920 during both training and testing. With this input, YOLOv5 resized the image to a quadratic 1920x1920 and applied mosaic augmentation. Although this image size was quite memory intensive, its use was not without reason. In most of the sequences in the annotated dataset, the player sizes are very small. Thus, we chose to train at this high resolution to provide more fine-grained details.

Association component

To optimize DeepSORT for the soccer domain, each player's ID in a sequence is treated as its own class. The problem of re-identification is then reduced to a classification task. This made training quite different than for object detectors. We needed to convert the dataset specifically for DeepSORT to learn to map player appearances to their corresponding IDs. The dataset conversion involved a few steps. First, we cropped out the ground truth bounding boxes for all frames in the sequence and then resized them to DeepSORT's default input resolution of 64x128. Secondly, cropped player portraits were grouped by their IDs. The IDs simply became the labels, and DeepSORT was trained like a regular, deep-based classifier.

DeepSORT operates similarly during inference. It automatically crops out player portraits based on box proposals from YOLOv5. Then, it performs classification on the portraits, one at a time. Different from optimization, it omits softmax and extracts the last feature layer instead. These become the appearance descriptors of the player.

3.3.3 Architecture 3: FairMOT

For our third architecture, a quite different approach was used to solve the problem of tracking. In contrast to most current methods, FairMOT is a one-shot tracker, which means it jointly estimates object positions and re-identification embeddings in a single forward pass. Its implementation details is explained in Section 2.12. An overview of the model can be seen in Figure 38. The system is depicted similarly as illustrations for previous architectures for better comparison. A practical implication of this is that training and inference of ID descriptors can be made end-to-end. Since part of the optimization objective is to produce descriptive ID vectors, an ID-labeled dataset must be provided. This is significantly different from previous experiments. Then, we first had to train and optimize an isolated detection model independently before passing its outputs to a separately optimized tracking algorithm.

FairMOT's deep-based feature extraction part does not make much sense without the association step. The two components are too closely coupled. Accordingly, FairMOT is arguably just a moniker for describing the complete tracking system. Thus, FairMOT is provided as a standalone, self-contained system, while the previous architectures require a fair amount of coding to implement the complete functional system.

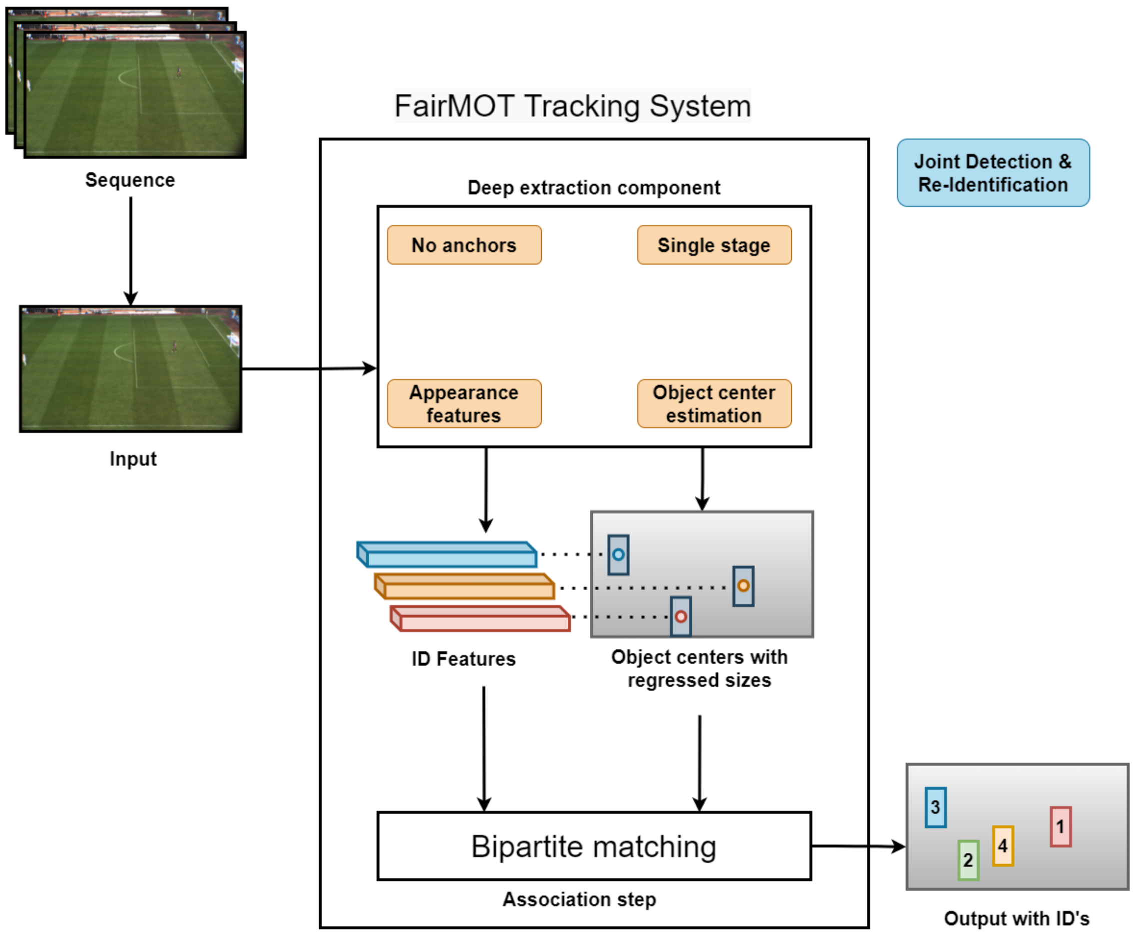


Figure 38: Architecture overview of the FairMOT tracking system. The FairMOT component corresponds to the one-shot model described in Section 2.12. The tracking setup is exactly as described in the work of Zhang et al. [32]. The association step is a simple bipartite matching algorithm that jointly uses ID features and box overlaps to compute costs. It functions as an additional post-processing step to output IDs.

There were several reasons why we chose FairMOT as our third architecture. These are listed as follows:

- Since its release, FairMOT has consistently been among the top open-sourced trackers on the MOT challenges [96].
- FairMOT's paper was released with a PyTorch implementation available on Github.
- It comes with weights finetuned on humans, which makes it highly transferable to our work via *transfer learning*.
- The architecture was revised as late as September 2020 (a few months prior to this work). Using the latest cutting-edge technology ensures our work is up-to-date.

FairMOT is a joint detection and re-identification (JDE) network, meaning that it works with ID-labeled datasets, conveniently referred to as JDE format. However, our COCO-formatted dataset is primarily intended for object detection tasks. IDs were merely included in an ad-hoc manner to this format. Accordingly, in order to use the FairMOT network, we had to convert our dataset to the object-ID intuitive JDE format.

A handful of scripts was provided with the FairMOT codebase that allowed for initiating common tasks such as training and producing demo videos from user-specified input. A downside of using the FairMOT repository was its somewhat rushed implementation. A disorienting file structure and lack of code readability made it obvious that the implementation was mainly written as an ad-hoc extension to the work of Wang et al. [91]. Whenever we wanted to add functionality, it required a massive investment to properly understand its problematic structure. For example, the possibility of evaluating validation sets was not a feature. This was implemented according to the evaluation method that will be described in Section 3.5. Consequently, we could evaluate with ID metrics, which surprisingly, was not a feature in the original code-base either.

3.4 Training

Mostly the same training strategies were used for all of the three tracking systems. The training was conducted with periodical rounds of evaluation of the validation set. *The same training and validation sets were used for all models.* Whenever we saw the validation set performance start to decrease for a significant number of iterations, we would cease training. We developed a logging module that conveniently logs many metrics obtainable from the training and evaluation phases. It would log metrics from all systems in the same format as long as it was integrated in the training loops. In turn, the logs allowed us to launch the metrics-per-train-step in graphs for easy inspection. Such a setup would allow for easy comparison and provide more fine-grained insight into the optimization process, making it easier to know when to interfere. Accordingly, the optimization of each model was recorded as described above. The graphs are included in Section 4 for the reader to inspect training and evaluation performances during optimization.

As alluded to earlier, in the detection-first, track-second approaches, the object detectors were independently optimized according to their own detection metrics on the validation set, for example, total loss or AP. Such was the case for both the first and the second architecture described in Section 3.3. This was done first because the association model directly benefits from better object proposals. Thus, yielding optimal object proposals yields an optimal starting point for the association model. If applicable, optimization of the association model was commenced second to produce the fully optimized tracking system.

In the case of the re-identification methods, the training process was a bit peculiar. Although their functionality in a deployed system generates descriptive re-identification features, their training objective is to minimize cross-entropy loss. In other words, during training, re-identification is treated as a classification problem, where object IDs are translated to classes. This had some implications for the validation procedure, which we discuss in the following subsection. For now, all we need to know is that the MOTA metric was heavily weighted in the stopping criterion during validation rounds.

It is well known that hyperparameter tuning, in general, is a time-consuming process. It is especially the case with the memory and computationally expensive models deployed in this thesis. Therefore, following time constraints, mainly default parameters were used when training. The occasional exception was when certain parameter adjustments would obviously benefit system performance. We would use some initial training rounds to assess the different configuration options available, so the number of iterations was generally kept below 1000. The configurations included deciding on backbones, efficient batch sizes, thresholds, and the number of RoI proposals. Once these were decided, training for a final model could be commenced. The resulting hyperparameters for each system can be found in Section 4.

3.5 Evaluation

Quantitative evaluation of the systems and their individual components was done on a separate, isolated validation set. The set was in common for all methods applied in this thesis. Validation sets are primarily used to determine optimal systems. In some cases, evaluation rounds could be done before, during, and after optimization. We did not use a test set but rather showcased our systems on the unannotated video footage described in Section 3.2.7. Bear in mind that we were not benchmarking against other models or a competition, simply because no such dataset exists to support it. In addition, we wanted to make the most use of our limited annotated data. Therefore, we found it sufficient to test on unseen videos qualitatively, and if needed, refer to the validation results for the metrics. In Section 4, quantitative evaluations are listed in the form of tables and graphs containing metrics, and qualitative evaluations are listed in the form of sequences of images with annotations drawn on top.

We used the same multiple object tracking metrics as in similar works such as Zhang et al. [32], Wang et al. [91], Bewley et al. [30], Wojke et al. [31]. These are mostly the ones that can be found on the leaderboards of the MOT Challenge [93] as well. Particularly, the CLEAR MOT metrics described in Section 2.13 are of greater interest to us, but the others add to providing a more detailed view of how specific models are performing and may reveal differences among them that are not captured by MOTA or MOTP metrics. Although optimizing object detection metrics was not the ultimate goal, it does have something to say about tracking performance, especially in the detect-first track-second case [30, 31]. It can be the crucial factor when identifying reasons for the difference seen in tracking performance. Accordingly, we used the COCO evaluation metrics for evaluating the detection parts of the trackers. It provides several APs to measure model performance in different scenarios. We found this to be the most informative object detection metric while being intuitive and simple. Additionally, a number of related works such as Jocher [52], Zhou et al. [23], Ren et al. [26] used the same method. *To summarize, we use all of the metrics listed in Section 2.13 to evaluate the systems.*

A MOT Validation Approach

In Section 3.2, we discussed the fact that the training and validation sets are comprised of one or more sequences of soccer footage. In the case of evaluating object detectors, the latter fact is ignored. That is, when evaluating object detectors, the validation set is treated as one large set of unrelated images (which corresponds to frames in the sequences). This works perfectly fine because object detectors are only intended to work with ground truths that relate to one image and not spanning a sequence of related ones. As such, metrics such as total loss and AP can be used when evaluating these models. However, tracking metrics can be obtained during component optimizations to assess the system-wide effect of the isolated training procedure. Next, we describe how such metrics can be obtained.

Interestingly enough, during training of re-identification components, the sequential structure of the training set is also ignored. This is because, during optimization, re-identification is treated as an object classification problem. Problems arise if the validation set was to be evaluated with the same metrics as used in the training procedure. Object IDs do not and should not be required to appear in both validation and training sets. If so, one would either have to use the same sequence in both sets, or one would need two sequences where the objects appear in both. In Section 3.2.6, we argued that the former would cause too similar images to appear in both sets, and could ultimately hurt performance. If this is not the case, during evaluation, the re-ID models would have to classify with labels they have never used before, which obviously would not go so well. In any case, classification is not the end goal and should accordingly not be the criterion for evaluation. In other words, *object classification metrics are not suited to evaluate re-identification models on a validation set.*

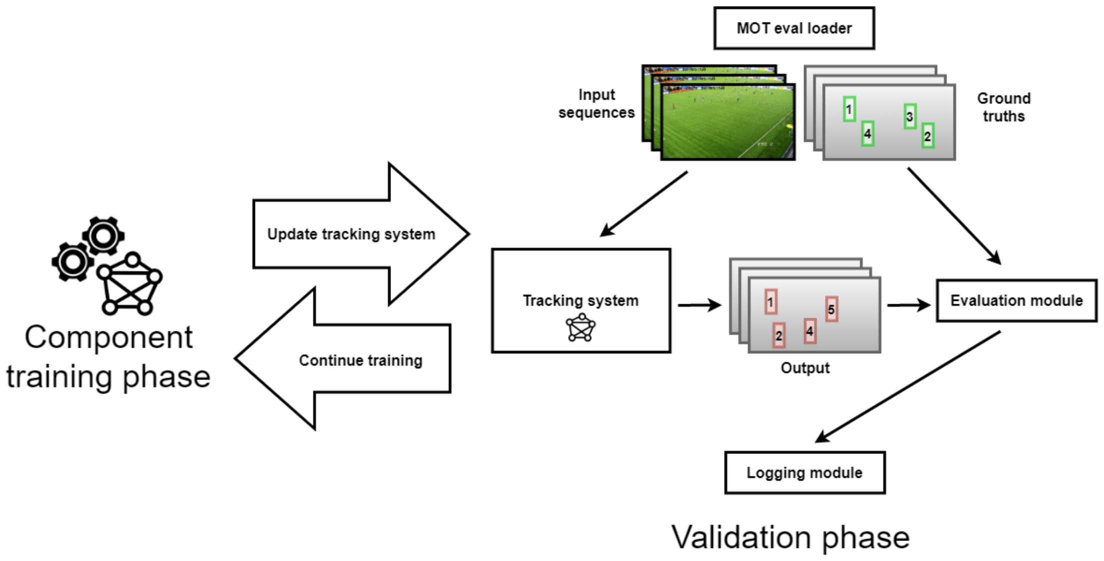


Figure 39: Our proposed evaluation approach. Component optimization can at any time be interrupted and switched to the validation phase. The complete tracking system is updated with the newly optimized component. Then, the system in its entirety is evaluated with MOT metrics on the validation sequences. Consequently, optimizable component's performance can be evaluated for the complete tracking system.

Instead, we propose not only to use the MOT metrics for the final evaluation round of the entire, complete system, but to use them during optimization of any component. Thus, for example, the re-identification components stopping criterion can be selected based on performance in relation to the final, complete tracking system. We find this to make sense because we optimize individual components to optimize the complete system, and the proposed method directly highlights how the component impacts system performance. To our knowledge, the proposed approach was not adopted in the works of Wojke et al. [31], Wang et al. [91], Zhang et al. [32], nor in the exhaustive inspection conducted of various implementations.

In order to realize this, custom tracking evaluation modules were developed to ease the task of component performance assessment. Indeed, some of the tools we used did not have any validation methods implemented whatsoever, so it was rather considered a necessity. Further, we could provide a unified way of evaluating and visualizing the performance of the different methods, enabling easy comparison. In Figure 39, the reader may inspect an overview of the implementation. Below we list some of the components needed to realize the approach.

- A custom dataloader that takes care of loading images/labels in the correct order according to the sequence they originate from.
- A modular tracking system interface that allows for swapping components during optimization.
- An evaluation module that accepts outputs from the tracking systems and evaluates them against ground truths.
- The logging module mentioned in Section 3.4 - Creates easy to inspect graphs from the data of the Evaluator module.

The components mentioned above, when properly interacting with each other, make up the validation phase. A validation phase can be executed at any time of the component optimization. It can happen before, after, and during optimization. Usually, it is done after each epoch of training. When entering a validation phase, the tracking system is updated with the current state of the component that is being optimized. Tracking system inference is then done on every sequence in the validation set, as provided by the MOT evaluation dataloader. Then, the evaluation module records every output from the tracker and compares them against ground truth data, which is also provided by the dataloader. The evaluator aggregates and computes metrics for each sequence in the validation set. These are then recorded by the logging interface, which constructs the graphs that can be seen in Section 4.

3.6 Extracting Sports Analytics Data

So far, this thesis has been concerned with achieving detection and tracking. Although these results are valuable in their own right, there is still a gap to fill before potentially being used in the sports analytics field. With this in mind, we investigated how the results from the MOT systems could be further processed and the possible challenges it would bring. Followingly, we developed some additional features as a means to demonstrate a hypothetical sports analytics tool. These features include calculating metrics for total distance covered, speed, and acceleration for each player. Also, we show how to obtain an overview of the players positioning and formation. Lastly, we assign teams to players using clustering, which enables calculating team metrics. These initial methods serve as a baseline that can be further developed, and they give an indication of how the MOT systems in this thesis can be used for sport analytics purposes.

3.6.1 Perspective Transform

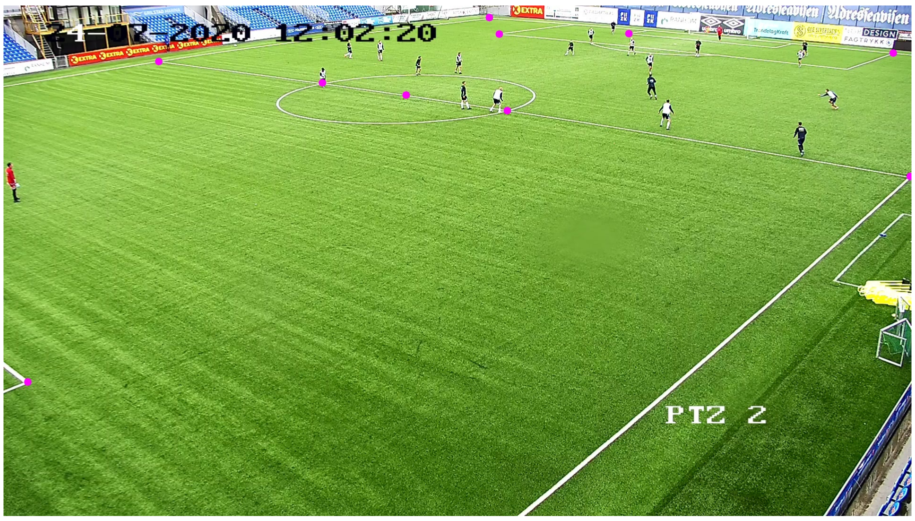
The first step we performed for extracting sports analytics data was performing a perspective transformation. With the perspective transformation, it was possible to transform the position of the players on the video stream to coordinates on a top-down visualization of a soccer pitch. Knowing player positions is essential for features concerning team formation. Moreover, if the coordinates are scaled with precision, they can be used for calculating metrics concerning player movement.

To obtain the player coordinates in a top-down view, we performed a perspective transformation from the input frame to the overview map. The method requires at least four pixel coordinates on the input and the corresponding ones on the output map, to be known. An example of this in soccer terms could be the four corresponding corner flags on the original frame and the output map. The points were found manually by matching the pixel coordinate from the input frame to the mapping illustrated in Figure 40. A transformation matrix is then obtained. The matrix could later be applied to transform any coordinate from the input frame to its corresponding coordinate on the output frame. Thus, after obtaining the transformation matrix, it was possible to transform all object position proposals to positions on an overview map.

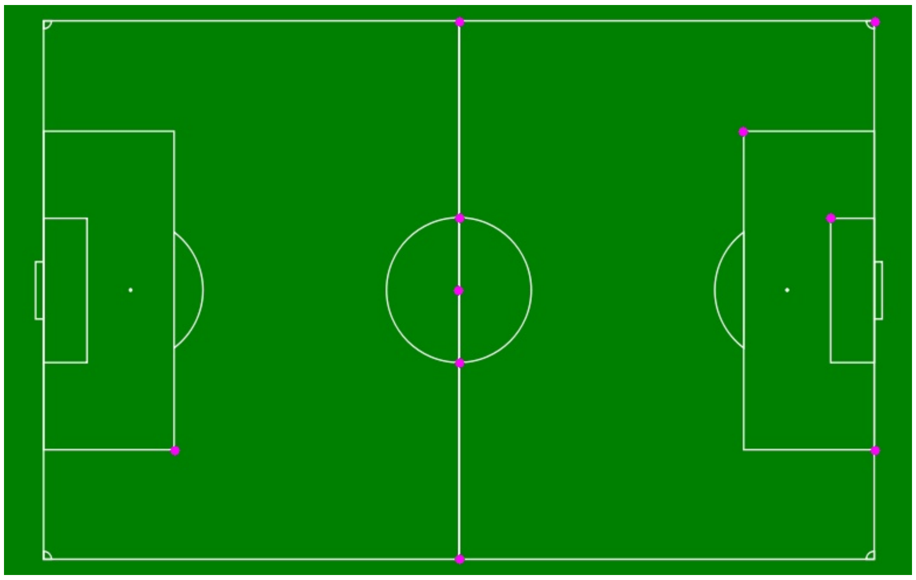
With the player positions transformed, this new representation could be used to generate useful metrics such as player speed and distance covered. By scaling the map to the size of an actual soccer pitch, it presents opportunities to monitor player movements on a metric scale instead of in pixels. This re-scaling can be simply calculated by taking either the length or width of a real pitch and divide it by the number of pixels for the equivalent distance in the overview map. As most soccer pitches are quite similar in size, a constant value could be used. However, for more accuracy, one would need prior knowledge regarding the exact size of the pitch in question. As this experiment was primarily a proof of concept, we used a standard 100 m for length and 60 m width.

We interpolated the transformed coordinates, and thus player movement in between every eighth frame. The interpolation helped in smoothing observed “back-and-forth” movement, giving the video of the overview map a more natural look and simultaneously yielding more sensible calculations concerning distance, speed, and acceleration. Given large fluctuations in movement, the calculated metrics would be significantly affected.

Using the methods described above, we created a handful of features for extracting sports analytics data. These features extracted metrics for total distance covered, average speed, top speed, and top acceleration. All the calculations were done using the transformed coordinates and with a pixel-tometer scaling constant, yielding results in the metric scale. The total distance was calculated by cumulatively adding the euclidean distance between each coordinate for a player, from one frame to the next. The speed and acceleration were calculated every tenth frame, by dividing the distance covered in those frames by the corresponding time interval. The acceleration was calculated using the same method, only replacing the distance covered with the change in speed.



Input frame from the Ranheim dataset.



Overview Mapping

Figure 40: Selecting 10 corresponding coordinates as input to the perspective transformation. Overview map adapted from 97.

3.6.2 Team Association

For supporting team-based features, it is essential with functionality for assigning teams to the detected players. The detection models are solely trained for detecting players and not, for instance, detecting if the player belongs to team one or team two. To solve this issue, we tried to apply the k-means clustering algorithm to associate players with teams.

As an initial experiment, we attempted to cluster the detection purely based on the bounding box proposal contents. That is, the k-means clustering would receive cropped out images using bounding boxes, and based on the pixel values, assign it to a cluster. To accomplish this, optimal clusters were made from bounding box proposals from the first frame in the sequence. New incoming detections were then immediately assigned to one of the clusters. However, using the raw pixel values from the detections directly did not yield optimal results, and the method was in need of refinement. The first step was to improve the cropping method for each detection. We decreased the bounding box size and moved it slightly upwards to target as much of the player shorts and jersey as possible and less of the surroundings. Moreover, some experimentation was done with various color spaces such as RGB, HSV, GRAY, and CIE $L^*a^*b^*$, to observe possible differences in accuracy and speed. Ultimately, the CIE $L^*a^*b^*$ was chosen, as this color space was created to differentiate between colors in the same way human vision conceives color [98]. Soccer kit colors are, after all, intentionally chosen to be easily differentiable for humans, not for being easily separable using, for instance, RBG-values.

After converting the color space, the mean pixel value in each channel of the cropped images was used as input to k-means. This was done to reduce data dimensionality, considering that k-means performs worse when the dimensionality is too large. Thus, the pixel values were compressed into three key features, one for each of the channels in $L^*a^*b^*$. By using these values, k-means primarily clustered all the players into one of two clusters. However, the goalkeepers, referees, and coaches were assigned to different clusters. The kit colors worn by these participants are normally significantly different than the outfield players. Since k-means was configured with two clusters, the clustering of these particular participants would not provide attributes that represented them. To improve on this, we made use of outlier detection. That is, the detection of which entries that were far away from both cluster centers. The aim was to assign the 20 outfield players to the two clusters and for the remaining detections to be labeled as outliers. The outlier detection was performed by calculating the euclidean distance between each of the entries and the cluster centroid it was assigned to. If the distance compared to the others was larger than a given threshold, the entry would be marked as an outlier.

3.6.3 Automatic Keypoint Detection for Perspective Transformation

The idea of this last sub-experiment was to test if it was feasible to automate the perspective transformation process. Choosing corresponding keypoints manually is a cause for a human-in-the-loop step to enable useful data, which is undesirable. As an initial plan, we wanted to try out some non-deep computer vision methods. The reasoning behind this was that the key features of a soccer pitch are the edges and corners that make out the pitch shape. Non-deep computer vision methods have traditionally worked well on detecting features of this kind. Even so, a deep-learning approach was considered, but it was decided not to be moved forward with. The requirements of this approach were deemed too extensive and subsequently out-of-scope for this work.

Pre-processing and Field Line Detection

Field line detection is a natural first step for automatic perspective transform. If we managed to detect the lines accurately, it would effectively give a sequence of points to choose from in the perspective transformation. For this task, we first applied the Hough line detection method [25]. The input image first had to be pre-processed. Bilateral filtering was applied to retain edges while smoothing out noise. Next, to keep pixels containing the soccer pitch and remove the background, such as the stadium, a green mask was applied. To remove the remaining noise, we found all the contours of the image after the green mask, then extracted the largest contour and used it as a filter. With this method, only the pixels from the pitch remained, and the rest of the frame was blacked out. As a final touch to fix contour outlines, an opening morphology was applied. The resulting image was passed through a canny transform, with adjusted thresholds: a low threshold of 98 and a high threshold of 260, for best detecting the edges. Finally, the output from Canny was passed as input to the Hough transform for performing line detection.

Automatic Feature Detection Algorithms

The last effort was focused on feature detectors such as SIFT [99] and ORB [100]. These are methods designed for detecting features in images and can be used to detect image similarities. Thus, with these methods, one could detect corresponding points between the input frame and the overview map. The resulting points could then be used in a perspective transformation. SIFT and ORB were tested on the unprocessed input frame, and with a combination of the different techniques described earlier, such as the green mask contour and the Hough line detection. SIFT proved to be the most reliant of these methods and is subsequently included in the results.

4 Experiments & Results

In this section, we detail a list of conducted experiments in order to help address the goals and research questions from Section 1.2. The result from the dataset creation process is listed first. There, the reader may find details on the resulting training and validation split. Also, it will help in evaluating the semi-automatic evaluation process described in Section 3.2.5. Then, as individual experiments, each of the tracking architectures described in Section 3.3 are optimized and evaluated. We train the models as described in Section 3.4. Further, methods are both qualitatively and quantitatively evaluated as described in Section 3.5. *That is, every metric, table, and image listed from here on out, is obtained by evaluating on unseen data.* We describe the setup, the results obtained, and provide a discussion of the results for each experiment.

Before inspecting the results, it may be helpful to consult the paragraphs below. They explain important aspects of the tables and images that will be presented in each experiment.

Setup Tables

The hyperparameters are listed for each architecture experiment, namely in Table 8, 10 and 12. The chosen hyperparameters will be used for the models in the training procedure. These are only a subset of the full list of configurations, which can be found in the Appendix, Section A. Here, only the most common or crucial parameters are listed - for brevity's sake.

Note that hyperparameters are necessarily specific to the implementation of the architecture. For instance, Detectron2 lets the user specify the number of FPN outputs to use in the RPN stage (see Appendix A.1) for the Faster R-CNN model, while other implementations may not. If such parameters are not listed, whatever is detailed in the model introduction paper or its official code release, is used. That is, we use the default parameters.

Training Graphs

The graphs listed in each architecture experiment display the performance on *the validation sequences* during the optimization of a component. They show the result from the proposed evaluation procedure discussed in Section 3.5. That is, *these are validation graphs.* The data points collected from any particular step corresponds to a complete evaluation round of the respective validation sequence. A step corresponds to one iteration or one optimizer step.

Performace metrics

For each architecture tracking experiment, a table of performance metrics is included. The metrics correspond to the ones discussed in Section 3.5 and 2.13. Each table will provide metrics for *the validation sequences* individually and for *the entire validation set.* Refer to Section 3.2.6 for the rationale on why metrics are obtained by evaluation on the validation set. Metrics are included before and after optimization of each architecture, indicated by "Pre" and "Post" columns, respectively.

For tables that show performance metrics after optimization, significantly improved numbers will be marked in bold, and usually means that improvement is at least doubled.

The metric tables shown in this section is only a subset of the complete list of metrics we were able to obtain. The latter can be found in the Appendix, Section B.

Visual results

There are included several images that demonstrate detection and tracking performance for the fully optimized systems. For detection demos, object detection proposals are drawn as boxes of the same colors on top of the input image. Each proposal's confidence score is drawn on top of the boxes. They range from 0 to 1.

A tracking demo will be provided for each optimized architecture. Four input frames are shown that are *about one second apart* in the video sequence. The system's outputs are overlayed in a similar manner as for the detection demos. However, model object ID proposals are indicated by different colors on the bounding boxes and ID numbers above the boxes. The complete demo videos that the frame snapshots originate from, can be viewed at `here`, and in [101]. Note that in the videos from the link, IDs may be initialized with different numbers than in the tracking demos shown below.

Some images are zoomed in on to provide more finegrained visualisations of the bounding boxes.

4.1 Dataset Creation

4.1.1 Experimental Setup

The setup for this step is as described in the dataset creation section (Section 3.2). Here, we list the results from that process. Recall that we acquired the video footage from three different sources. For each of these sources, Table 5 lists statistics that reflect the work achieved from manual and semi-automatic annotation. We also had a thorough discussion regarding the dataset split in Section 3.2.6. The resulting split is detailed in Table 7, and further discussed below. From the tables below, note that the labels Small, Medium, and Large annotations correspond to the COCO API's definition when calculating APs, APm, and APl, respectively.

4.1.2 Experimental Results

Dataset Summary

	Ranheim	Alfheim	ISSIA	TOTAL
Num. of annotated sequences	4	8	6	18
Num. of annotated images	1000	1729	**15593**	18322
Avg. duration of sequence (s)	10	8.6	**104**	-
Num. of annotations	20311	43089	**129288**	192688
Num. of tracks	84	228	144	456
Avg. annotation per image	20.3	24.9	8.3	10.5
Avg. Annotation resolution	23.9x50.9	36.3x80.6	46.4.6x96.9	41.8x88.4
Num. of Small annotations	9348	8192	1	**17541**
Num. of Medium annotations	10963	32213	**124554**	167730
Num. of Large annotations	0	2684	4733	7417

Table 5: A summary of the dataset creation process described in Section 3.2. Of the sources listed, only Ranheim and Alfheim were subject to the labeling process. As mentioned, ISSIA was pre-annotated. Note that ISSIA is a significantly larger dataset than what we were able to create.

ISSIA Dataset Summary

	Seq 1	Seq 2	Seq 3	Seq 4	Seq 5	Seq 6
Num. of annotated images	2598	2598	2599	2599	2599	2600
Duration of sequence (s)	104	104	104	104	104	104
Match origin	1	1	1	2	2	2
Part of field visible	1/3	3/3	2/3	2/3	1/3	3/3
Num. of annotations	10989	13120	**37983**	**37924**	15774	13698
Num. of tracks	27	21	27	24	26	19
Avg. annotation per image	4.2	5	**14.6**	**14.6**	6.0	5.3
Num. of Small annotations	0	0	1	0	0	0
Num. of Medium annotations	10296	13051	37033	36044	15574	12556
Num. of Large annotations	693	69	949	1880	**0**	1142

Table 6: Annotations provided by the ISSIA dataset. Each of the sequences that comprise the dataset is listed with statistics. Note that they mostly differ in the number of annotations. This is directly related to the view of the field. The sequences with 2/3 of the field visible has considerably higher amounts of objects due to its centered view. "Match origin" refers to whether the sequence originates from the first or the second soccer match.

Dataset Split Summary

	Training set	Validation set
Num. of annotated images	**16525**	1797
Num. of sequences	15	3
Num. of annotations	**160601**	32087
Avg. annotation per image	9.7	**17.6**
Avg. Annotation resolution	41.6x87.5	42.4x92.7
Num. of Small annotations	14215	3326
Num. of Medium annotations	140196	27534
Num. of Large annotations	6190	1227
Num. of classes	1	1
Sequences from Ranheim	3	1
Sequences from Alfheim	7	1
Sequences from ISSIA	5 1/2	1/2 (seq 4)*
Image % from Ranheim	**0.05**	0.14
Image % from Alfheim	**0.09**	0.14
Image % from ISSIA	**0.86**	0.72
Annotation % from Alfheim	0.23	0.18
Annotation % from Ranheim	**0.10**	0.16
Annotation % from ISSIA	0.67	0.66

Table 7: A summary of the final dataset split. The split is created according to the approach described in Section 3.2.6. There, it was also explained why the split only consists of a validation and training set, and not a test set. *****Due to ISSIA sequences large duration, only half of the second sequence in the dataset was included in the validation set. This is also the sequence with more annotations of the six.

4.1.3 Discussion

From Table 5, observe that we managed to get hold of a total of 18 annotated sequences. The data acquired constitutes a total of 18322 annotated images and 192688 annotated objects. This is certainly not bad given the time restrictions and the limited publicly available datasets. For instance, Dendorfer et al. [102] provides a total of 8 sequences per challenge. However, most of those sequences have a duration of 2-3 minutes, and only the six sequences originating from the ISSIA dataset have a duration that matches those. Additionally, the sequences from Dendorfer et al. [102] have considerably more populated sequences than the ones we have obtained, yielding a higher number of annotations and tracks.

In Section 3.2, and specifically Table 2 and 3 we saw that the duration of each video was at least 15 minutes. In Table 5 however, these sequences, when annotated, are shortened to 10 seconds each. This is due to the segmentation of larger videos to smaller sequences to produce manageable annotation work units. An in-depth description and discussion for this process were provided in Section 3.2. It is also the reason why we see more sequences annotated than originally available.

The difference in seconds, images, and objects annotated between ISSIA and the two other datasets introduce a severe imbalance in the dataset. For instance, ISSIA contributes a large portion of medium-sized objects, and particularly in the size of 46x96 pixels, to the overall dataset. It causes the 17540 small-sized objects contributed by the two other data sources to be less significant. It will not be surprising to see if optimized models are biased towards sequences with a similar appearance as the ISSIA sequences. Note from Table 5 that even though the total size of the ISSIA dataset is larger than the other two by a large margin, the number of player trajectories, or unique objects, is still of comparable sizes. This is related to the total number of sequences each dataset origin contributes. Even though the sequences have a longer duration, mostly the same objects are present in the scene, and rarely are new tracks introduced. This is a direct consequence of the closed world scenario that is a game of soccer. Accordingly, the size of ISSIA contributes a lot more variance in regards to detection training data, but not necessarily for re-identification training data.

The ISSIA dataset is so different from the rest of the annotations that a separate table (Table 6) is provided for it. There, key numbers are presented for each sequence in the dataset. The most noticeable of which is the variance in the number of annotations. Most likely, it is the positioning of the camera that captured the footage and the resulting view of the pitch that causes the variance. The sequences marked with the "2/3" part of the field visible have a view of the center where most objects are positioned. 1/3- and 3/3 parts of the field visible have a view of the part of the field containing the goal. Thus, not as many objects will be present in these sequences. For example, the average number of objects per image is 14.6 for the centered-view footage and usually below 6 for the rest. Sequence number 4 contributes zero large-sized objects. Given that all sequence was captured with cameras of equal distance from the pitch, this lack of large objects is probably attributed to the players never being near the camera placement. Otherwise, a lot of the numbers are consistent between the sequences within the ISSIA dataset.

Table 7 holds a summary of the dataset split process described in Section 3.2.6. The training set is considerably larger than the validation set, as seen by the numbers marked in bold. This is not necessarily critical, as long as the validation set itself is sufficiently sized. The critical factor is to have the validation set large enough to house the variations needed to assess generalization capabilities. In that respect, we argue that 1797 images and 32087 objects are sufficient. However, we have reasons to believe that the actual contents of the validation data do not have the diversity needed to assess generalization capabilities. The sequences from the validation data are, after all, from the same dataset sources as in the training set. For example, both splits have sequences from the ISSIA dataset, which are not identical but very similar. The biggest difference in the latter is mostly object positions and appearance. The background remains largely the same. However, this is one of the pitfalls of a shortage of available data: creating diverse training and validation data becomes very hard. In that regard, we may have made the best out of the data available. As mentioned in Section 3.2.7, we included additional unlabeled footage to to better assess model generalisation.

The dataset imbalance mentioned above is inevitably also reflected in the dataset split. For instance, only 5% and 9% of the training data consists of Ranheim and Alfheim images, respectively. Luckily, in terms of object examples, Alfheim and Ranheim are slightly better represented, constituting a total of 33%. In the validation data, more Alfheim and Ranheim images are present. In terms of objects, the percentages are about equal for the validation and training data. After realizing that centered-view footage provides more objects per frame, sequence 4 from the ISSIA dataset was purposely included in the validation set. This would hopefully increase diversity a small amount by heavily populating the validation set with more objects to track.

CVAT and Semi-automatic Annotation

We found the CVAT application to be a handy tool for annotating videos. Intuitive controls and mechanisms for collaboration made it able to maintain a high throughput of annotated sequences. However, there were a few concerns that halted the annotation process. These are primarily features we felt were missing and are listed below:

- No real support for larger video files.
- UI provides no way to download or upload multiple tasks.
- No multiclass lableling support.
- IDs on tracks are automatically assigned and cannot be switched.

The first issue is that there is no way to upload a large video file and segment them into different tasks such that collaboration on a single video can be done. It had to be manually implemented by scripting with the provided CLI tool. Similarly, we had to implement functionality to upload and download multiples sequences and annotated tasks, respectively. While straightforward to implement, it was time-consuming. As a workaround to multiclass labeling, we could create multiple boxes for the same object, and label each with different classes. This is inconvenient, especially if one could just label more than one class per drawn box. Even though our experiments were concerned with single-classed objects, we would still like to include more labels to create a feature-rich dataset that could support various future work approaches in the CV-for-soccer domain. We painfully found no solution to the fourth issue. This is particularly annoying when we would like to maintain object IDs between annotated sequences.

We found the semi-automatic annotation process described in Section 3.2.5 to be extremely valuable. Models were performing surprisingly well even without optimization; thus, they could be adopted at an early stage. As hoped, the automatic annotation removed a lot of the work. The annotation process was largely reduced to fixing noisy trajectories. As models were optimized, or we were trying out different models, the automatic annotation continued to improve and reduced the amount of interference required. The only caveat was that since semi-automatic annotation was still under development in the CVAT project, we had to implement it ourselves by altering the source code. This took a fair amount of time to accomplish. By the time we were done, we were left with little time to get some actual annotation work done.

4.2 Experiment 1: Tracking using Faster R-CNN and SORT

4.2.1 Experimental Setup

Faster R-CNN and SORT Hyperparameters

Detection architecture	Faster R-CNN
Tracking architecture	SORT
Backbone	ResNet50 with FPN
Weight initialisation	Detectron2's COCO benchmark weights [69]
Optimizer	SGD
Learning rate	10k its. @ 1e-2 + 40k its. @ 1e-3 + 26K @ 1e-4
Data Augmentations	Random flip
Hardware	1 x Tesla V100 PCIe 32GB
Training time	30 hours
Iterations	76000
Network input size	1080 x 1920
Image batch size	6
Confidence threshold	0.5
Number of classes	1
SORT IoU threshold	0.3

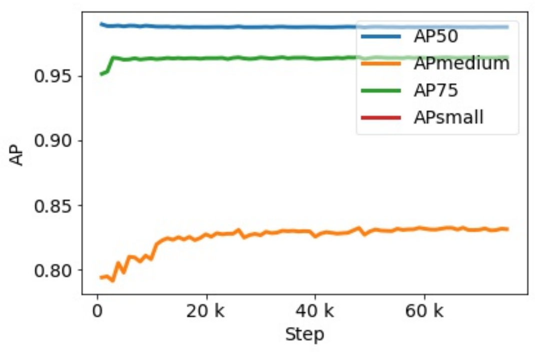
Table 8: A subset of the Faster R-CNN and SORT configurations used. The complete hyperparameter list can be found in the Appendix, Section A.1

Table 8 lists the most important parameters used for Faster R-CNN and SORT in our experiments. The weights used are Detectron2's best performing on the COCO challenge [51] and the model in question. They were obtained from [69].

4.2.2 Experimental Results

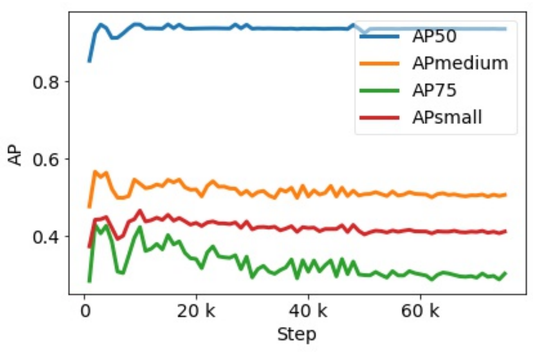
Faster R-CNN AP Validation Curves

ISSIA



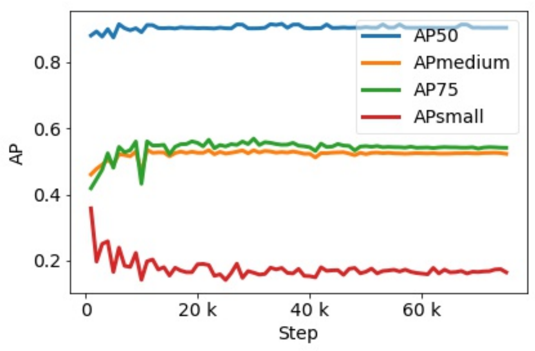
AP50 reaches a peak quickly. APmedium has a steady incline until the end.

Ranheim



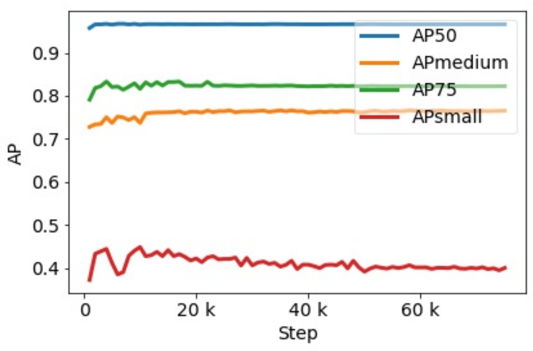
The metrics indicate an unstable performance start for the Ranheim dataset. Metric performance varies synchronously.

Alfheim



All metrics, except for APsmall, seems to increase until iteration 20K.

Overall

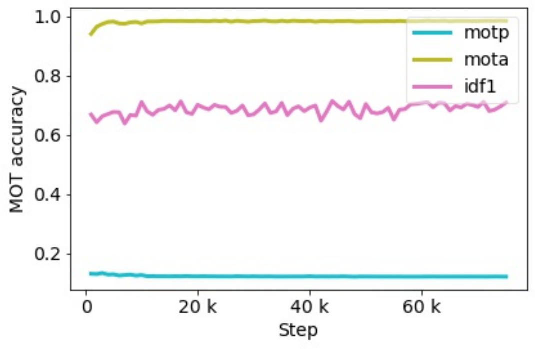


AP metrics increase rapidly to its maximums. The decrease in APsmall is also reflected in the overall performance.

Figure 41: AP validaton curves obtained during training of Faster R-CNN. Each subplot corresponds to periodic evaluation on each of the validation sequences. The bottom, right-most plot aggregates metrics from all the sequences.

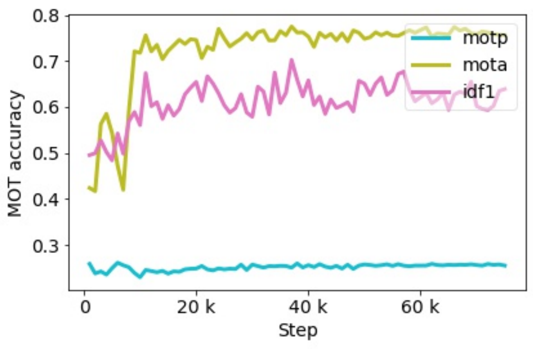
Faster R-CNN MOT Validation Curves

ISSIA



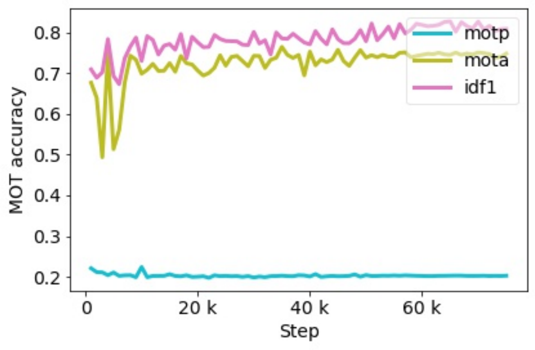
The MOT metrics show little variance on the ISSIA sequence.

Ranheim



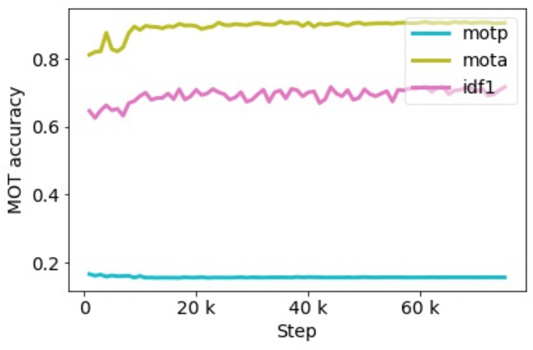
The MOT metrics are unstable and inclining in the first 40K iterations.

Alfheim



Tracking performance seems to slightly still increase on the Alfheim sequence.

Overall



Overall, the MOT metrics remain largely unchanged.

Figure 42: MOT accuracy metrics obtained during training of Faster R-CNN by regularly evaluating on the validation set.

Faster R-CNN Training Curves

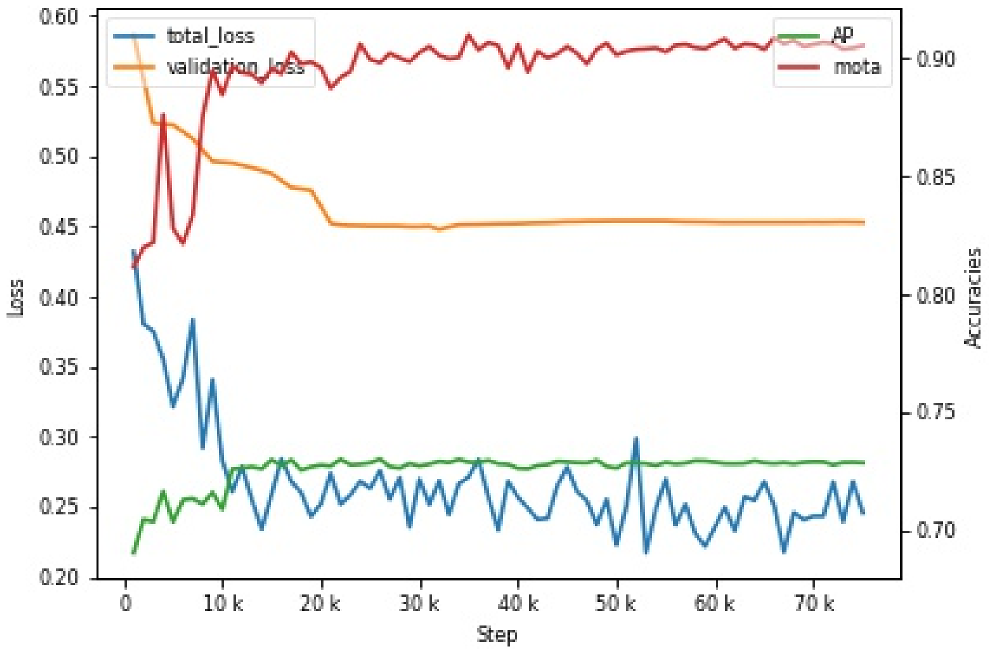


Figure 43: A graph showing the training and validation loss side-by-side during optimization of Faster R-CNN. AP and MOTA were also included to examine better the effect detector performance tuning has on the entire system. At exactly 60K iterations the final model is extracted.

The plots above are all obtained from the same training procedure of the Faster R-CNN model detailed in Table 8. The learning rate decayed from 0.01 to 0.001 at iteration 10k. The total number of iterations trained was 76K. Note that SORT is not a machine learning algorithm, and therefore no training is needed. Also note that Faster R-CNN needs to be part of a complete tracking system to obtain the MOT metrics during optimization. To see how this was implemented, refer to Section 3.5.

Faster R-CNN + SORT Performance Metrics

Metrics	**Alfheim**		**ISSIA**		**Ranheim**		**Overall**	
Pre/post train	Pre	Post	Pre	Post	Pre	Post	Pre	Post
MOTA ↑	43.5	**84.7**	-12.2	**98.6**	12.7	**89.3**	01.8	**94.6**
MOTP ↑	29.4	20.1	14.6	12.2	27.9	25.4	18.7	15.6
IDF_1 ↑	60.6	87.3	42.4	71.1	46.9	71.6	45.6	74.1
ID switches ↓	42	12	60	28	53	35	155	**75**
Fragmentations ↓	114	28	85	35	115	73	314	**136**
AP ↑	22.8	**51.7**	69.8	**83.1**	23.1	**42.4**	55.2	**72.8**
AP50 ↑	62.8	**90.2**	91.7	98.6	57.7	**91.8**	82.4	96.4
APsmall ↑	00.1	**16.7**	-	-	17.5	40.7	14.0	**39.6**
APmedium ↑	25.1	**52.5**	70.0	83.0	33.8	49.5	59.4	76.5
Recall ↑	82.1	0.925	98.4	98.9	73.7	92.3	91.6	96.7
Precision ↑	68.5	0.924	47.1	**99.8**	55.1	**97.6**	50.6	98.1
False positives ↓	2208	445	23452	33	2997	113	28657	**591**
False negatives ↓	1047	439	333	228	1316	387	2696	**1054**

Detection FPS: 12 **Association FPS**: 167 **Total FPS**: 11

Table 9: Validation performance metrics for Faster R-CNN and SORT. The ”Pre” and ”Post” columns refers to the metrics obtained before and after training, respectively. The final performance metrics shows that almost all numbers are improved. The change in performance is directly due to optimisation of the detection component. Note that the metrics are obtained by evaluation on the validation set. Refer to Section 3.2.6 for the rationale.

In Table 9, negative MOTA values are present. Recall that from Section 2.13, the MOTA metric ranges from $-\infty$ to 100.

Faster R-CNN Good Detection Demo

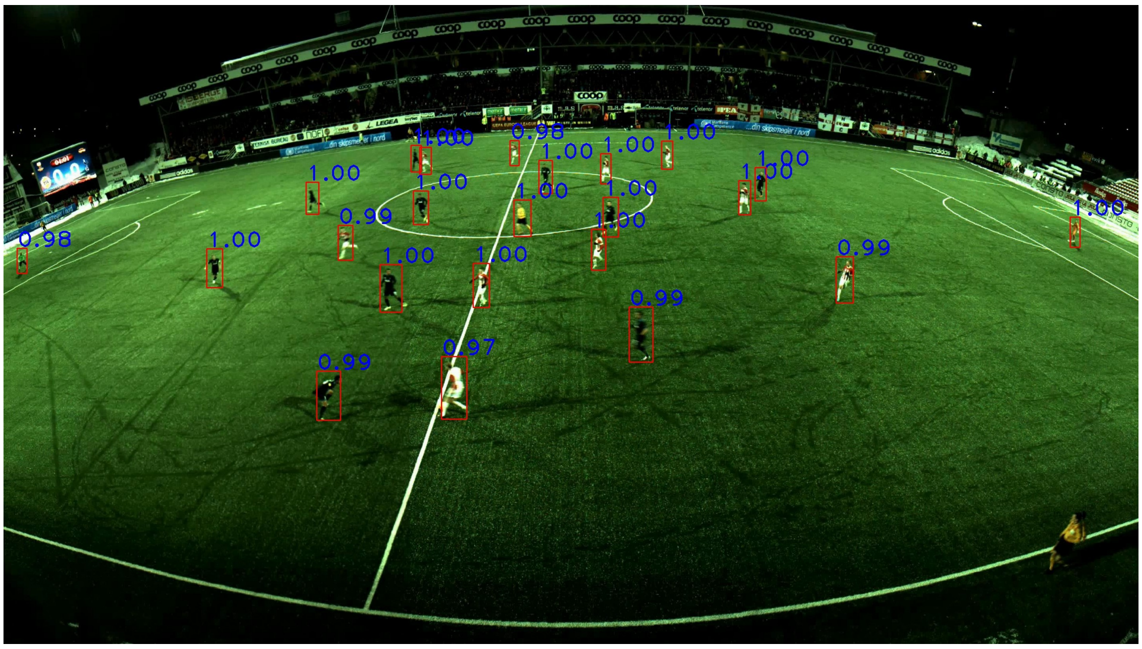


Figure 44: Examples of good detections by the finetuned Faster R-CNN model. All objects are detected with near max confidences and tight bounding boxes.

Faster R-CNN Bad Detection Demo

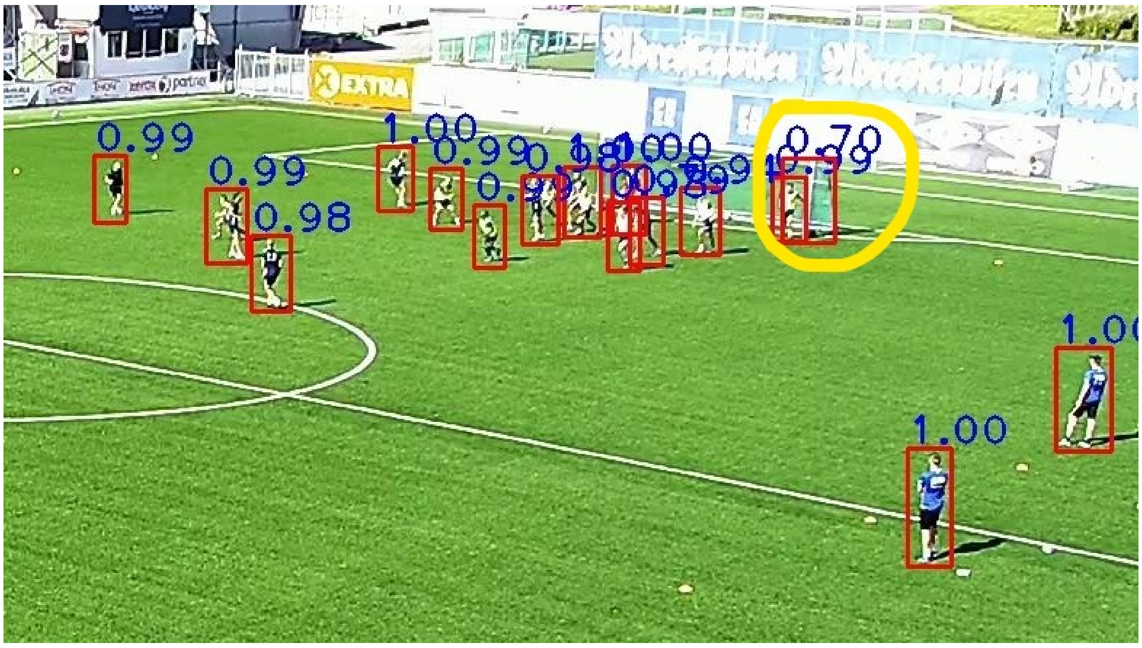
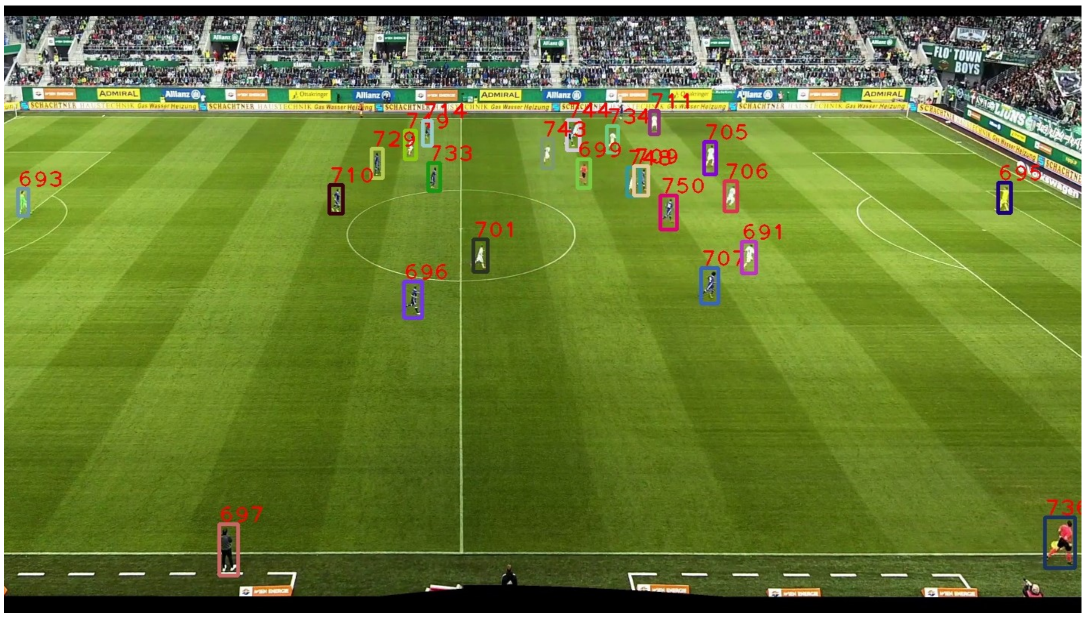


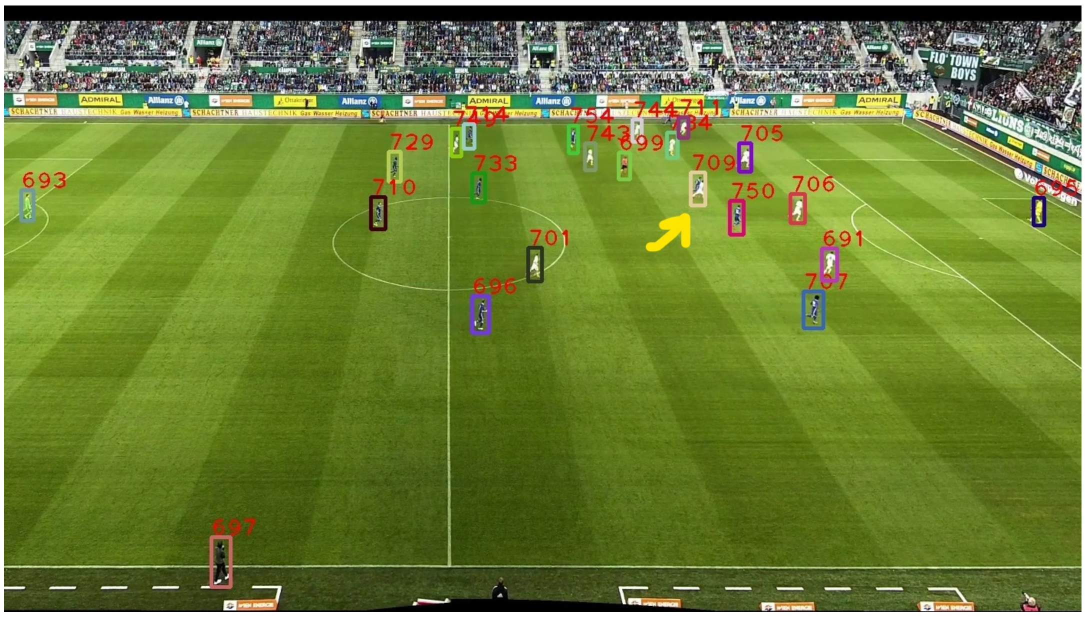
Figure 45: Examples of bad detections by the finetuned Faster R-CNN model. The goal post is wrongly classified as a player, although with lower confidence than the other detections.

Faster R-CNN + SORT Tracking Demo



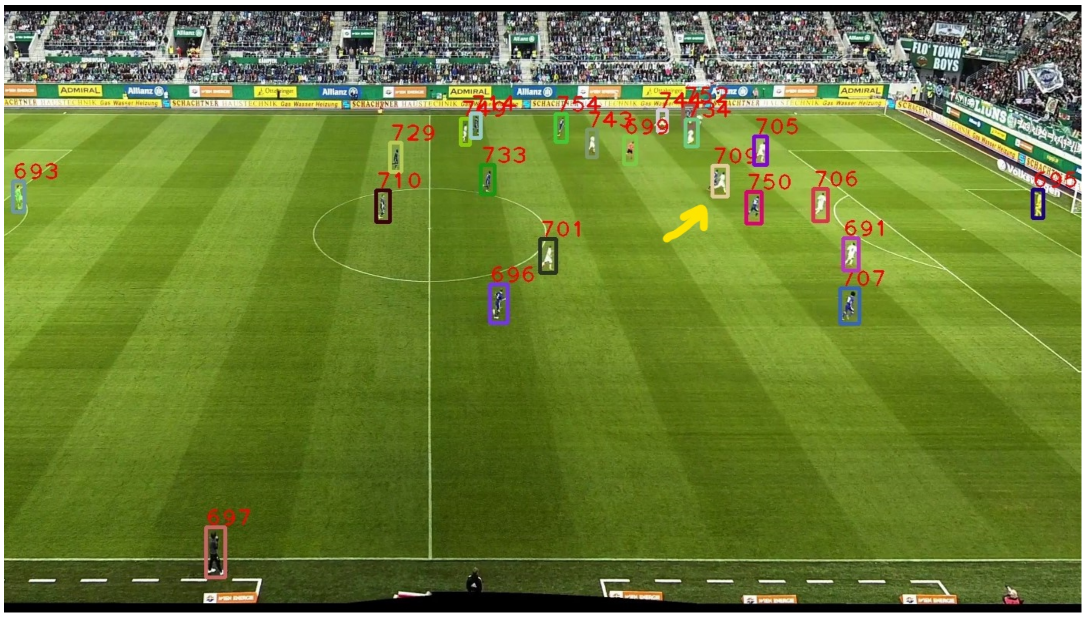
t=1

Most objects are correctly localized in the first frame.



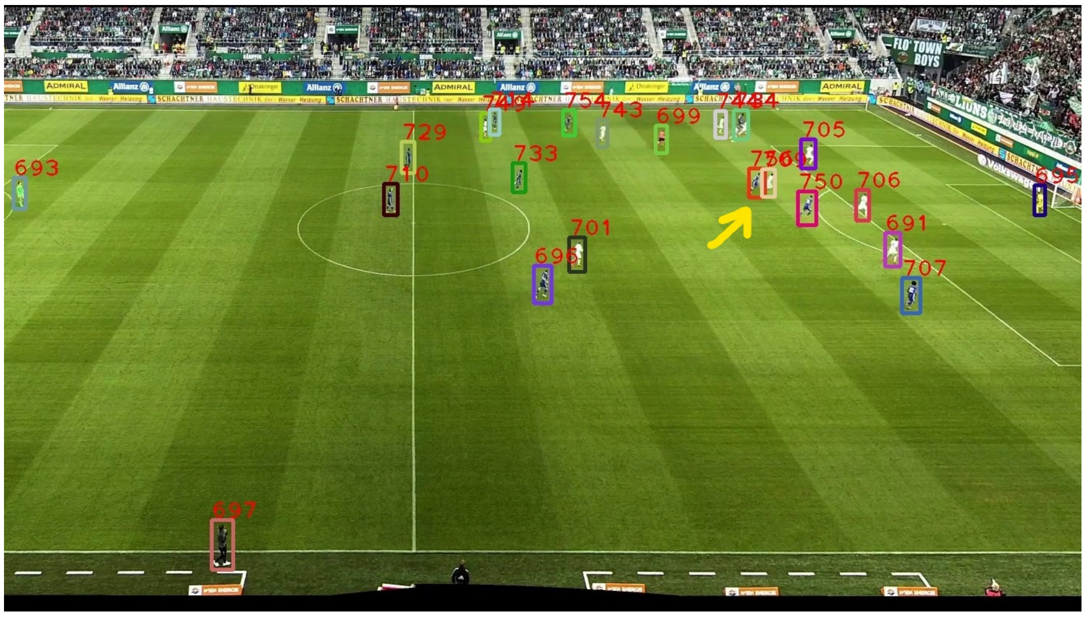
t=2

A player is being occluded behind player 709 (white), and is no longer detected.



t=3

The player behind player 709 is still in occlusion, and subsequently still not detected.



t=4

The object behind player 709 has emerged from occlusion and has been given a new ID (red).

Figure 47: Faster R-CNN and SORT tracking demo on one of the additional test sequences, described in Section 3.2.7. The objects are small. Frames are about one second apart. The whole demo can be viewed `here`, and at [101].

4.2.3 Discussion

From the validation curves in Figure 41, 42, and 43, observe that most performance metrics have the largest increase in the first 1000 iterations. Additionally, most of the metrics are highly unstable in the first 10K iterations. This is probably due to the high learning rate that was set in the first 10K iterations. In the following 10K iterations, there is a slow an steady increase until most of them peak at 20K iterations, which nicely aligns with decayed learning rate.

There are some exceptions, however. For instance, in the Ranheim curves in Figure 41, there is a shaky start, but for all APs except the AP50, the metrics seem to decrease instead stably. It suggests that Faster R-CNN improves localizing soccer players with an IoU overlap of 0.50 but fails to improve at higher thresholds. It is likely the case both for medium- and small-sized players because APsmall and APmedium are both declining. Similarly, there is an apparent decline in APsmall performance on the Alfheim dataset. There may be numerous reasons for this. Firstly, it may be due to the significant imbalance in the small-to-medium-sized object ratio. The model may be heavily biased towards performing well on medium-sized objects. Secondly, it may be because the model fails to detect objects of this appearance and size and that they demonstrate to be too hard examples for the model to handle effectively.

Furthermore, the high correlation between AP75 and APmedium suggests that medium-sized objects dominate the Alfheim validation sequence. Faster R-CNN requires little training on the ISSIA sequence to reach a high optimum with AP75 and AP50. The APmedium requires more steps before reaching its maximum, suggesting that it continues to improve on proposals with overlaps between 0.75 and 0.95. The fact that Faster R-CNN reaches convergence quicker on the ISSIA sequence than the other two is likely due to the imbalance in the overall dataset. The first few iterations use batches that are likely to contain a high proportion of images originating from the ISSIA dataset. Thus, a few more iterations are required in order for the model to have "seen" enough examples from the two other datasets. The "Overall" graph reflects the significant ISSIA contribution to the overall dataset by having very similar curves.

From Figure 42, the MOT subplots show a high correlation to its respective counterparts in the AP plots in Figure 41. For example, the AP instability in the Ranheim and Alfheim plots is also seen in the MOT plots, although with slightly more amplitude. In addition, Faster R-CNN and SORT quickly achieve convergence in terms of MOT metrics, exactly as discussed for the AP curves. The validation MOT plots, alongside the AP plots, clearly show the correspondence between detection and tracking performance that was heavily emphasized in the works of Bewley et al. [30] and Wojke et al. [31]. In Figure 43, the overall MOTA and AP performance further confirms this. There, the training and validation loss of Faster R-CNN is also included. All of the four curves roughly converge at the same point. It is indubitably clear that MOTA, thus tracking improvements, is solely dependent on an increase in detection performance. The Ranheim MOT curves converge on top despite their declining AP counterparts. It makes sense because for SORT, IoU overlaps of 0.50 might be enough to make the correct associations, and the AP50 mostly converges in an incline.

Table 9 demonstrates the drastic improvements in pure numbers between the out-of-the-box model that Detectron2 provides and the optimized version. For instance, the MOTA increases from a minuscule score of 2 to an impressive 94.6 score. Recall that this increase is achieved with little training. Further, the number of false positives drops from 28K to 591 after optimization. The improvements in AP are also significant. It is clear that the default Faster R-CNN, and consequently SORT, has significantly benefited from the optimization process.

The impressive results from Table 9 is further reflected in the detection demos showed in Figure 44 and 45. These images are drawn from the validation set, and the optimized Faster R-CNN bounding box proposals are drawn on top. The impressive AP scores make a lot more sense when inspecting the tight, perfectly placed bounding boxes that it produces. The images and the high recall value suggests that most objects are indeed detected. In addition, the object scores are nearly maxed for all instances. Even in the "bad" examples, most objects are localized with high confidence. However, there are a few cases where the detector wrongly place bounding boxes, for instance, when assigning a player box to a goal post. Luckily, these false positives have a lower confidence score than the rest and are few in between, as confirmed by the high precision value in Table 9.

Figure 47 showcases a challenging tracking scenario for Faster R-CNN and SORT. The sequence is neither from the Ranheim, Alfheim, or ISSIA datasets and thus rigorously tests the system's capabilities for generalizing beyond these datasets. The objects are small, and the background differs distinctly. Most objects are detected with high accuracy, and tracking works for almost all objects in the time frame shown. There are, however, a considerable number of fragmentations and ID switches. The cause might be the slight model weakness towards detecting small players, and especially in occluding scenarios. SORT is also known for not being able to perform well in re-identification after long occlusion periods. This is precisely what is demonstrated in Figure 47, where a player is given a new ID after 2 seconds in occlusion. The whole demo can be viewed `here`, and here [101], for the reader to inspect.

4.3 Experiment 2: Tracking using YOLOv5 and DeepSORT

As mentioned in Section 3.3.2, two YOLOv5 models were considered, namely YOLOv5x and YOLOv5l6. The latter was deemed the best fit for our domain due to its appropriate trade-off in accuracy and processing speeds. Hence, the following setup and results are from YOLOv5l6. The results from YOLOv5x are listed in the Appendix, Section B.2.

4.3.1 Experimental Setup

YOLOv5 and DeepSORT Hyperparameters

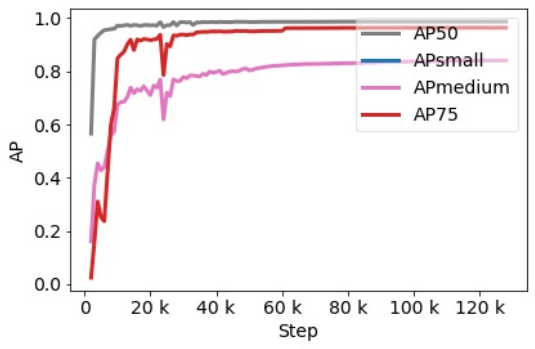
Detection component	YOLOv5
Tracking component	DeepSORT
Backbone	Modified CSPNet [55]
Weight initialisation	YOLOv5's COCO benchmark weights [95]
Optimizer	SGD
Learning rate	0.01
Data Augmentations	Scaling, color space adjustments, mosaic augmentation, flipping, translation
Hardware	2 x Tesla V100 PCIe 32GB
Training time	60 hours
Iterations	128000
Network input size	1920 x 1920
Image batch size	4
Confidence threshold	0.4
Number of classes	1
DeepSORT crop size	64 x 128
DeepSORT IoU threshold	0.3
DeepSORT iterations	31360
DeepSORT batch size	512
DeepSORT training time	12 hours

Table 10: A subset of the YOLOv5 and DeepSORT configurations used. The complete hyperparameter list can be found in the Appendix, Section A.2

4.3.2 Experimental Results

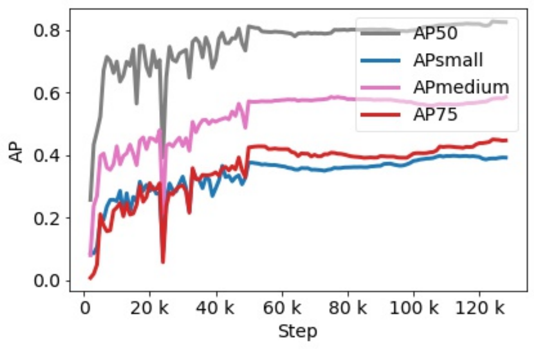
YOLOv5 AP Validation Curves

ISSIA



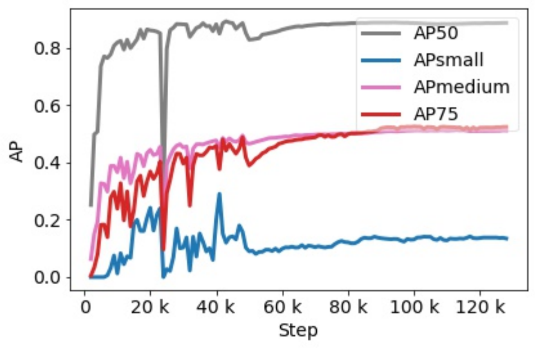
APs steadily increases and flattens out towards the end of the training.

Ranheim



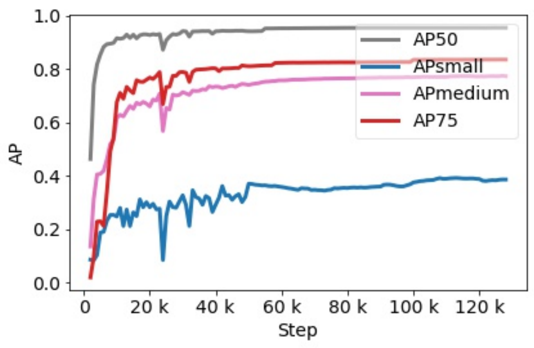
APs are unstable in the beginning, but ultimately flattens out on top.

Alfheim



APs are unstable in the beginning.

Overall

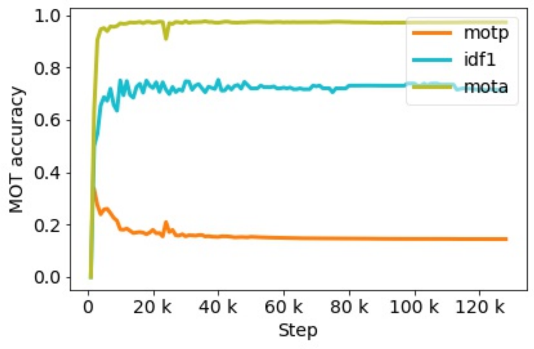


Overall, all COCO APs seem to incline until the final iterations.

Figure 48: COCO AP validation curves obtained during training of YOLOv5.

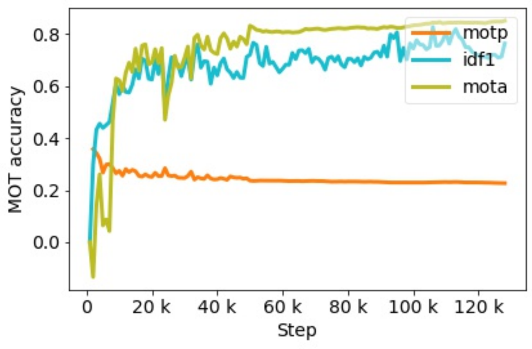
YOLOv5 MOT Validation Curves

ISSIA



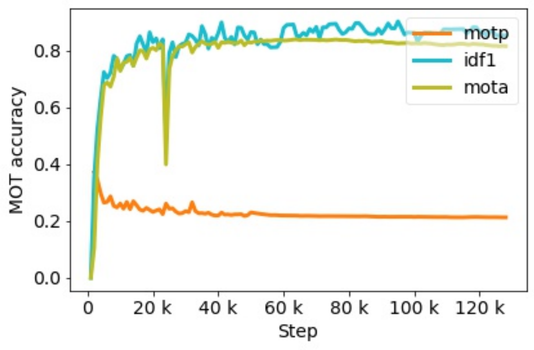
MOTA and IDF$_1$ values have peaked after 60K iterations.

Ranheim



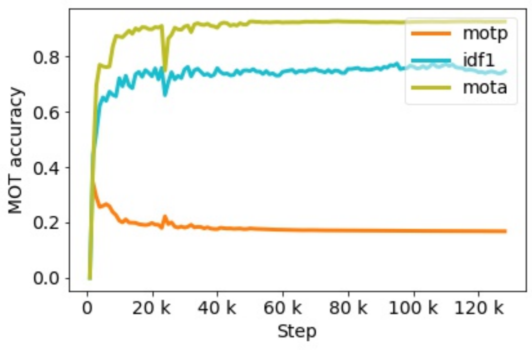
The MOTA and IDF$_1$ values are highest at 90-120K.

Alfheim



There is a rapid and steady increase until around 60K steps. After that the MOTA starts declining.

Overall



Overall, it seems the highest IDF$_1$ and MOTA values can be found at step 110K.

Figure 49: MOT accuracy metrics obtained during training of YOLOv5 by regularly evaluating on the validation set. Note that DeepSORT has not been optimized at this point.

YOLOv5 Training Curves

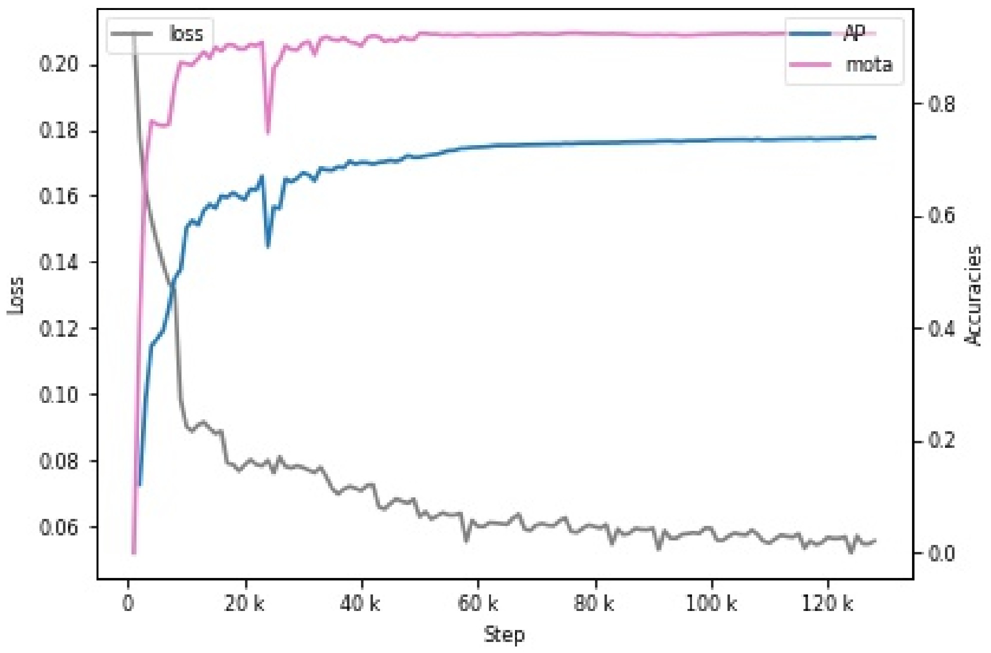


Figure 50: A graph showing the training loss side-by-side with the validation MOTA and AP during optimisation of YOLOv5. Validation loss was unfortunately not provided by YOLOv5's implementaton. AP and MOTA are stable and maxed until the end. Consequently, training is allowed to finish before extracting the final model.

The plots above describe the validation set performance of YOLOv5 and DeepSORT during the optimization of YOLOv5. All metrics were obtained from the same training procedure. The learning rate was not reduced at any point. The training lasted for a total of 130K iterations. The final model was trained of 130K iterations as well. YOLOv5 needs to be part of a complete tracking system to obtain the MOT metrics during optimization. To that end, DeepSORT with pre-trained weights was used. In the figure below, the metrics from the optimization of DeepSORT are shown. There, DeepSORT gets its inputs from the already optimized YOLOv5 model.

DeepSORT Training Curves

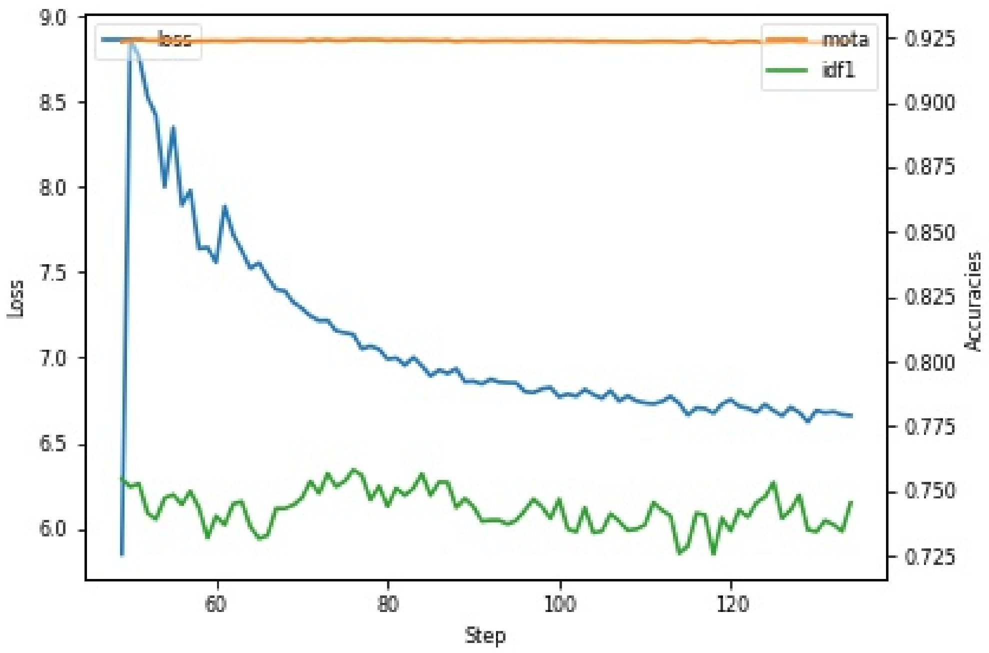


Figure 51: A graph showing the validation loss and ID accuracy metrics side-by-side during optimisation of DeepSORT. The steps correspond to epochs (one pass through the entire train set). The starting point is a DeepSORT pre-trained for 40 epochs, explaining the high number of epochs on the graph. Although validation loss is declining, there is no considerable change in performance at any point during training.

YOLOv5 + DeepSORT Performance Metrics

Metrics	Alfheim		ISSIA		Ranheim		Overall	
Pre/post train	Pre	Post	Pre	Post	Pre	Post	Pre	Post
MOTA ↑	66.7	80.7	92.7	97.1	58.3	**91.2**	82.7	**93.2**
MOTP ↑	27.7	21.0	16.1	14.1	27.4	23.4	19.2	16.7
IDF$_1$ ↑	73.8	88.4	68.9	72.7	61.8	77.6	68.9	**76.4**
ID switches. ↓	21	6	26	26	23	26	70	58
Fragmentations ↓	97	37	137	60	127	64	361	**161**
AP ↑	25.3	**49.9**	75.3	85.5	21.0	**47.6**	62.3	**74.9**
AP50 ↑	67.7	88.1	94.7	98.8	48.5	92.8	87.5	96.5
APsmall ↑	0.	27.9	nan	na .	6.3	47.0	6.1	**46.2**
APmedium ↑	27.1	59.0	75.1	88.9	36.6	61.1	65.7	81.4
False positives ↓	626	657	819	**266**	58	20	1503	943
False negatives ↓	1288	464	697	322	2005	393	3990	**1179**

Detection FPS: 25 **Association FPS**: 13 **Total FPS**: 9

Table 11: Validation performance metrics for YOLOv5l6 and DeepSORT. The performance metrics for YOLOv5 and DeepSORT after the optimisation of both components, showcased in Figure 50 and 51, respectively, are shown in the ”Post” columns. Metrics for YOLOv5l6 + DeepSORT before any optimization are included in the ”Pre” columns. Note that the metrics are obtained by evaluation of the validation set. Refer to Section 3.5 for the rationale.

YOLOv5 Good Detection Demo

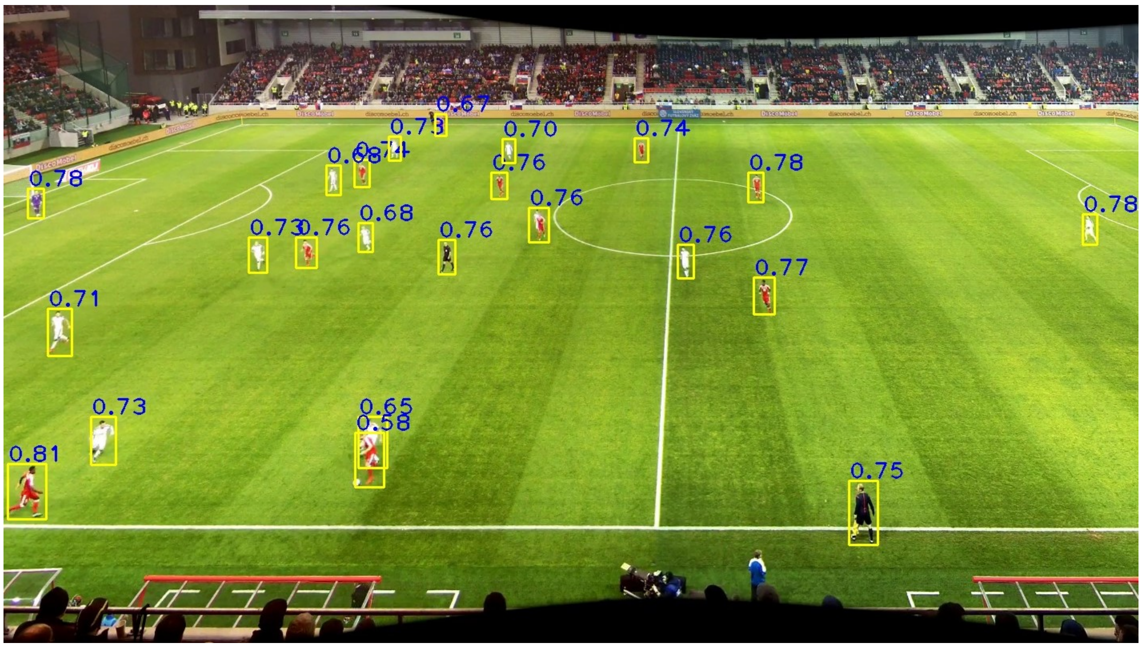


Figure 52: Examples of good detections by the finetuned YOLOv5 model. All objects are localized on an image from the additional test set described in Section 3.2.7. Objects are small.

YOLOv5 Bad Detection Demo

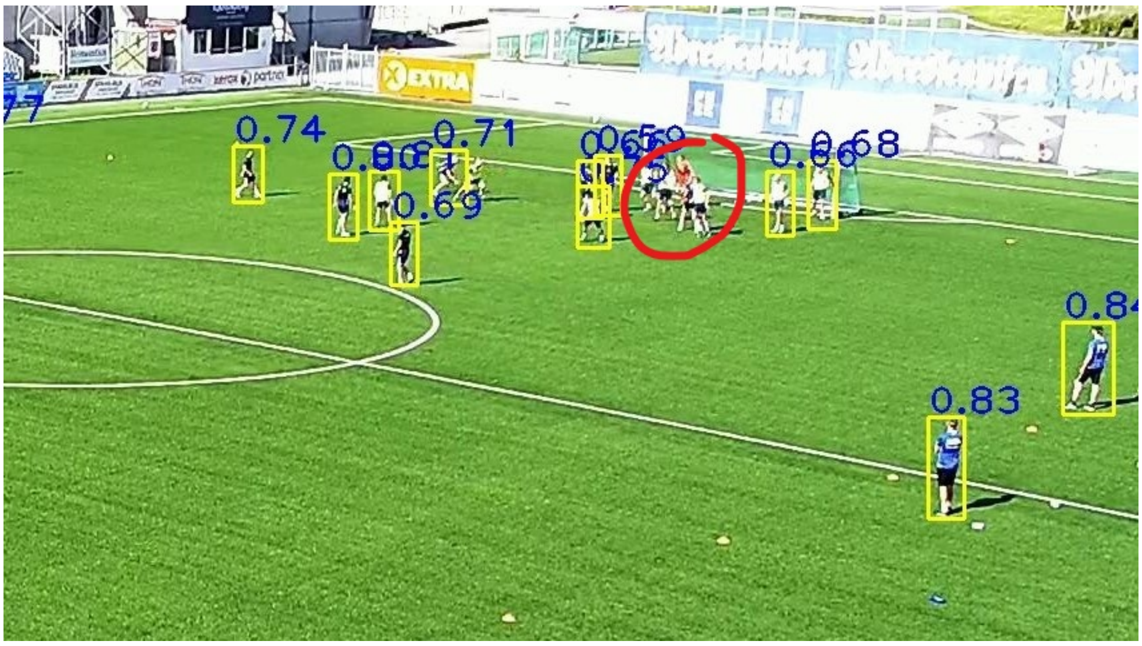
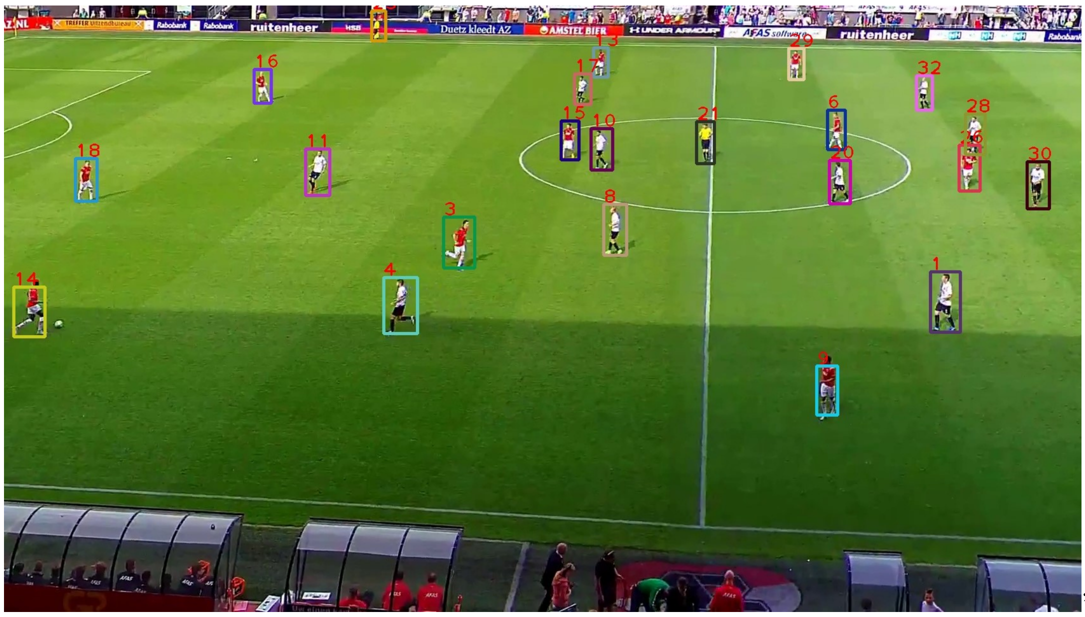
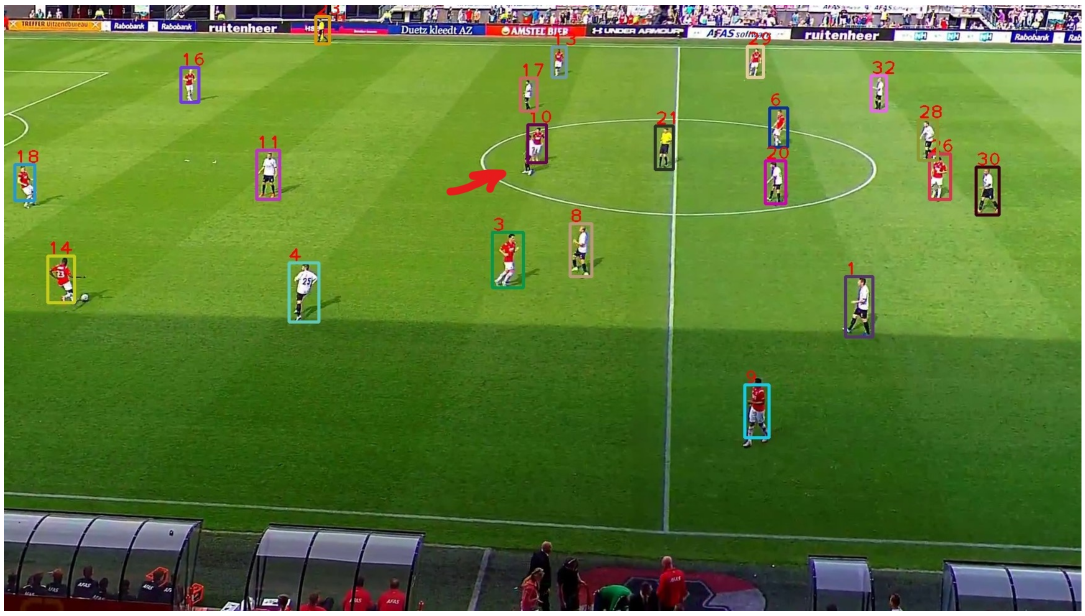


Figure 53: Examples of bad detections by the finetuned YOLOv5 model. On Ranheim validation frames, YOLOv5 frequently miss groups of players. There is a tight cluster of players in front of the goal that are not detected.

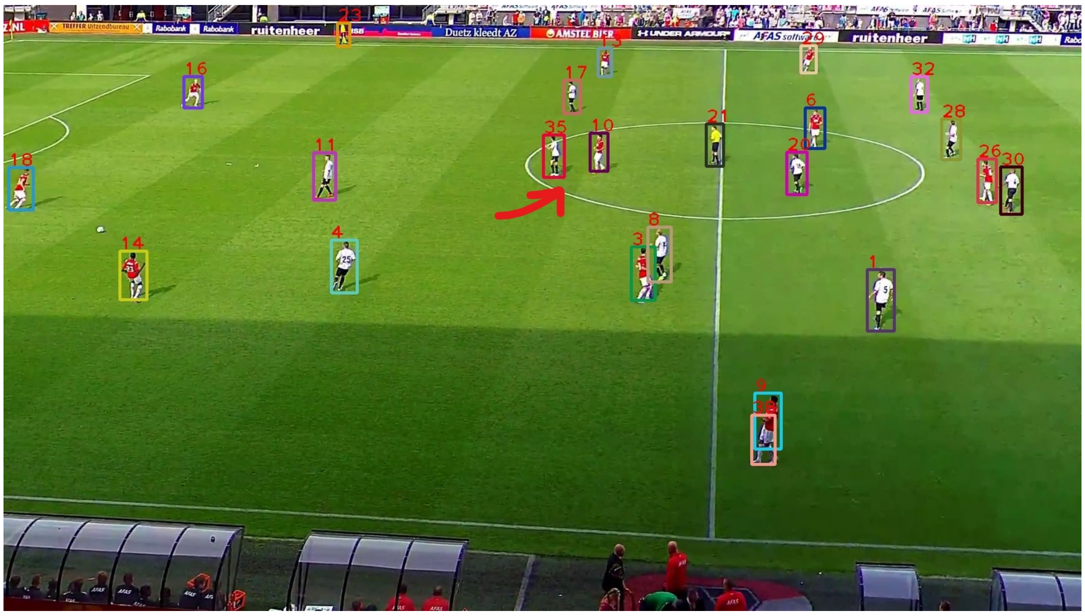
YOLOv5 + DeepSORT Tracking Demo

t=1 ,

All players are correctly detected.

t=2 

There is an ID transfer between player 15 and 10.

t=3

ID 15 has been destroyed, and ID 35 has been given to the wrong player.

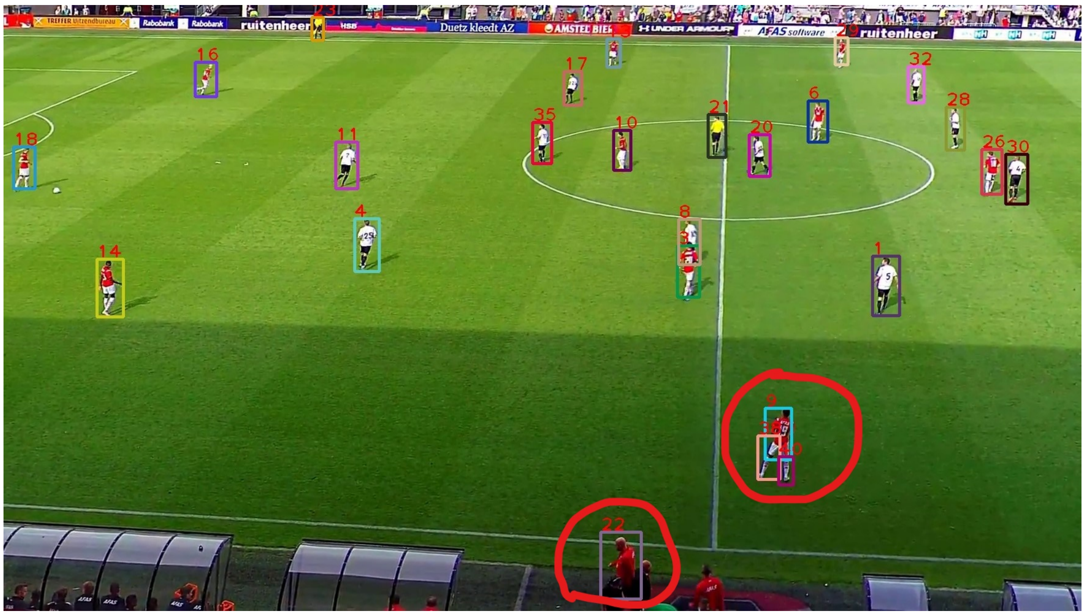
t=4

Tracks are mostly carried between frames for the sequence shown. There are some odd detections on the camera-end of the field, however.

Figure 55: YOLOv5 + DeepSORT tracking demo on one of the additional test sequences, described in Section 3.2.7. Frames are about one second apart. The whole demo can be viewed `here`, and here [101].

4.3.3 Discussion

The default YOLOv5 model showed an impressive AP of 62.3. It also appeared to generalize well on the additional test data, backed by the reasonably high AP across all validation sequences. All in all, the results were rather promising before training. From the graphs in Figure 48, observe how the different AP curves flatten out during training. The AP curves converge fastest for the ISSIA dataset, subsequently for the Alfheim dataset, and lastly for the Ranheim dataset. It lines up well with the example difficulty that each of the respective datasets presents. Similar to Faster R-CNN, the APsmall on the Alfheim dataset is highly unstable, but in contrast, YOLOv5 stabilizes and obtains a slow increase before flattening out. The overall AP peaks at around 120k iterations. The MOTA and IDF$_1$ scores in Figure 49 show a steep incline in the first 20k iterations for all the datasets. Performance on the Ranheim dataset is arguably the most unstable and slowest to converge in terms of the MOT metrics, similar to the respective AP scores. Alfheim and ISSIA are most stable and converges fastest on the MOT metrics. The overall performance best takes place towards the end, with the Alfheim MOTA slowly decreasing, and more importantly, the IDF$_1$-score dropping significantly just after the 110K mark.

By inspecting the training graphs of DeepSORT in Figure 51, there is observed no improvement for MOTA or IDF$_1$ during the entire training phase. While the training loss for DeepSORT is gradually improving, interestingly, it does not affect the performance of the overall MOT system. This result is somewhat unexpected and showcases the value of our evaluator. Nonetheless, the result is not inexplicable. When initially employing DeepSORT, there were worries that the re-identification task would be difficult because of the considerable similarities between tracking targets. The players are generally small, so the resolution makes it difficult to capture detailed features such as the face or jersey number. Adding to that, all the players are wearing matching soccer kits, mirroring the appearance of their teammates, making the tracking task substantially harder.

After completed optimization, YOLOv5 improves upon its previously mentioned starting AP of 62.3, reaching a total AP of 74.9, which is impressive. Taking into consideration an FPS of 25 further underlines the result. The APsmall saw a significant increase from 6.1 before optimization to 46.2 after. Given the under-representation of small players in the dataset, this increase speaks volumes for the quality of the data augmentation performed during training. The overall tracking system also yields competitive results with a MOTA of 93.2 and an IDF$_1$ of 76.4. The number of ID switches is relatively low as well, supporting the high IDF$_1$-score. The DeepSORT FPS of only 13 is arguably the most considerable letdown, bringing the total FPS down to 9. Notably, the MOTA, IDF$_1$, and AP were highly proficient before optimization, indicating the model had brilliant performance out-of-the-box.

In Figure 52, YOLOv5 produces tight bounding on all the players. The players are small, showing YOLOv5's highly respectable performance in detecting smaller objects. Bear in mind that the example is drawn from the additional test set, which supplies considerably different examples than the ones seen in the training data. There were observed cases in further qualitative testing where the detector struggled with players that are tightly compacted. An example of this is depicted in Figure 53, which is from the Ranheim dataset. There, significantly more situations occur where player occlusion and clustering are present, largely caused by the skewed camera angle. Although YOLOv5 generally detects small players well, it does struggle more compared to detecting the medium players. YOLOv5 does perform extensive data augmentation, mitigating some of the dataset's overweight of medium annotations of players compared to small ones. However, there are still some deficiencies when the players are small enough.

In Figure 55, a tracking demo for the optimized YOLOv5 and DeepSORT tracking system is provided. In the images, players are more spread out than in Figure 53. YOLOv5's output is faultless, predicting tight bounding boxes on all the desired cases. Even though DeepSORT failed to finetune deep appearance features for the soccer examples in this thesis, the example above clearly demonstrates that the algorithm can mostly perform accurate tracking. It appears that DeepSORT's spatial knowledge prevents it from failing. It might also be the case that the pre-trained weights in DeepSORT are to some degree transferable to the soccer domain but fail to specialize further. In the above example, the system struggles with the two players occluding each other. Given properly learned features, this particular switch could have been avoided by following the different colors of the jerseys'.

4.4 Experiment 3: Tracking using FairMOT

4.4.1 Experimental Setup

FairMOT Hyperparameters

Architecture	FairMOT
Backbone	DLA34 (See Section 2.12)
Weight initialisation	FairMOT's MOT benchmark weights [103]
Optimizer	Adam
Learning rate	2K its. @ 1e-3 + 20K its. @ 1e-4 + 8K its. @ 1E-5
Data Augmentations	Rotation, scaling, jittering, translation
Iterations	30000
Image batch size	6
Hardware	2 x Tesla V100 PCIe 32GB
Training time	25 Hours
Network input size	1056x1920
Confidence threshold	0.4
Number of classes	1

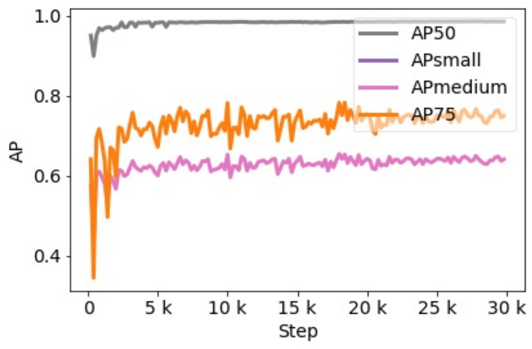
Table 12: A subset of the FairMOT configurations used. Consult the Appendix, Section A.3, for a full list of the hyperparamters.

In Table 12, note that FairMOT does not need different architectures for detection and tracking but is rather included in combined single shot architecture that does both tracking and detection. This is in-line with the architecture description in Section 2.12 and 3.3.3. The weights are obtained from the official FairMOT repository [103]. These are the same Zhang et al. [32] used to produce benchmark results for the MOT Challenge.

4.4.2 Experimental Results

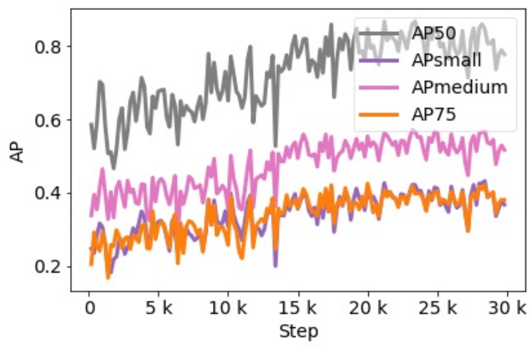
FairMOT AP Validation Curves

ISSIA



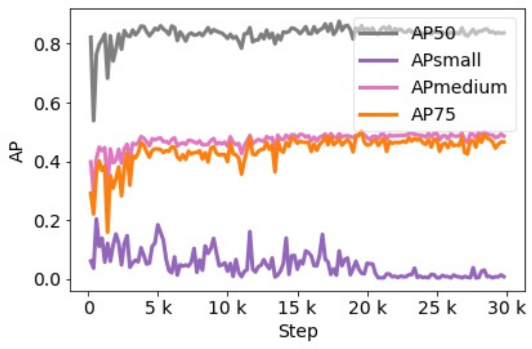
AP50 is peaked after 5K iterations on the ISSIA sequence. All metrics safe to assume peaked after 10K iteratons.

Ranheim



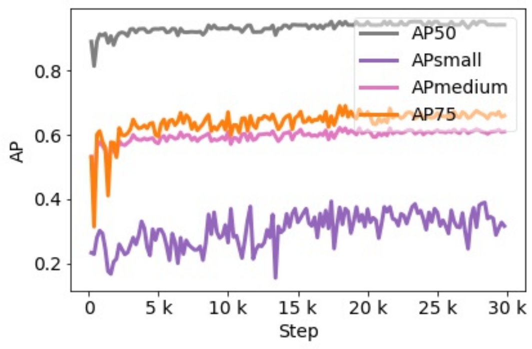
While still increasing, the AP performance is unstable.

Alfheim



APsmall seems to decline in the Alfheim sequence.

Overall

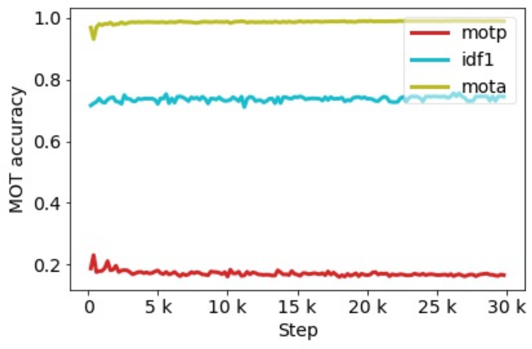


Overall, the AP metrics seem to have reached maximum after 10K iterations.

Figure 56: Detection metric curves obtained by evaluating FairMOT on the validation sequences during training.

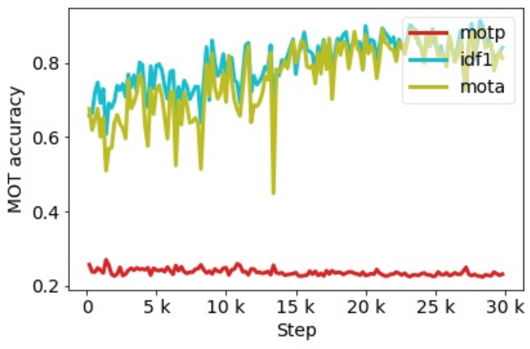
FairMOT MOT Validation Curves

ISSIA



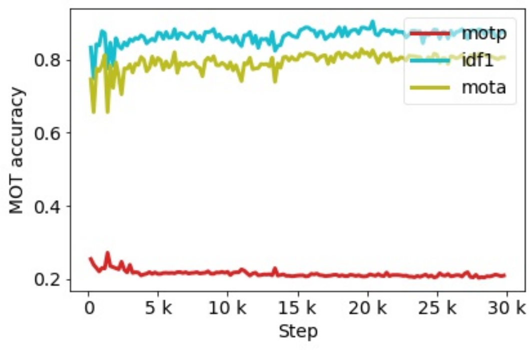
The remaining 25K iterations does not seem to affect MOT performance on the ISSIA sequence.

Ranheim



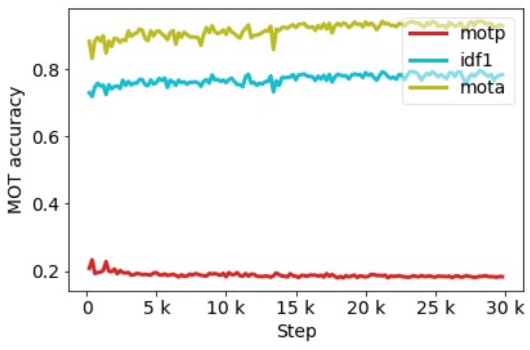
FairMOT performance on the Ranheim sequence is unstable, but increasing.

Alfheim



In the beginning, MOT performance is unstable on the Alfheim sequence.

Overall



Overall, it seems FairMOT performance is stable and flat throughout the training procedure.

Figure 57: FairMOT MOT performance metrics monitored during training by regularly evaluating on the validation data. In contrast to detect-first, track-second approaches, picking optimal models based on MOT performance is a reasonable approach.

FairMOT Training Curves.

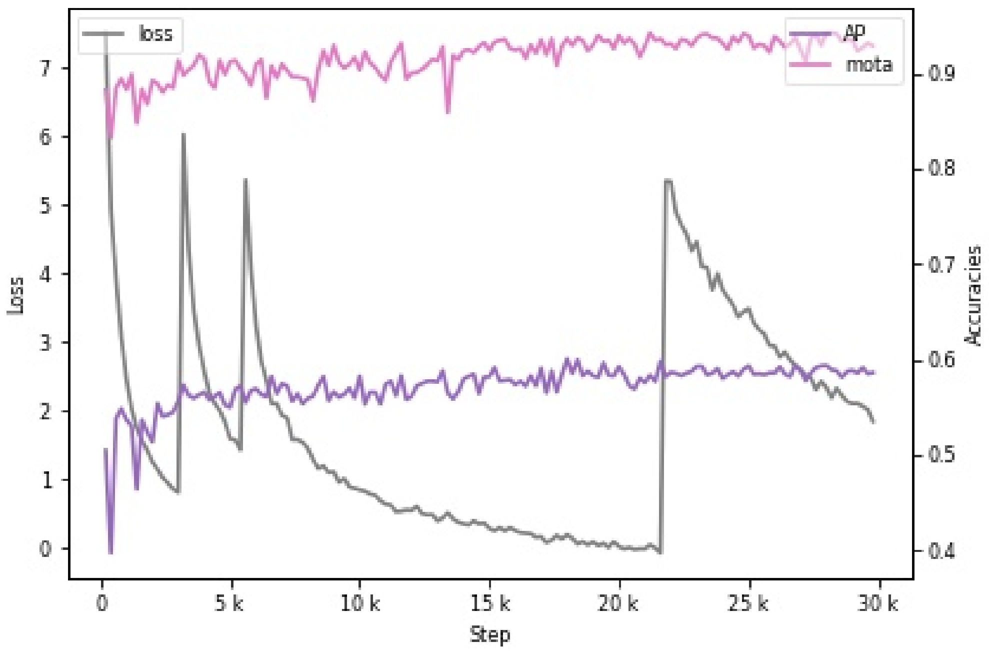


Figure 58: A graph showing the training loss side-by-side with the COCO AP and MOTA during optimisation of FairMOT. Validation loss was unfortunately not available. The AP and MOTA is saturated towards the last iterations. The final model is trained for all iterations.

The plots above describe the validation set performance during the optimization of FairMOT. All metrics were obtained from the same training procedure. After the initial 2K iterations, the learning rate was set down from 1e-3 to 1e-4. Then, it was held at this learning rate for another 20K iterations before setting it down to 1e-5 for the final 8K iterations. The training lasted for a total of 30K iterations. The final model was trained of 30K iterations as well. The spikes in the training loss in Figure 58 corresponds to points where the training processes were paused, for example, to adjust parameters. Despite the momentary increase in loss, MOTA and AP is still behaving ”normally.”

FairMOT Performance Metrics

Metrics	**Alfheim**		**ISSIA**		**Ranheim**		**Overall**	
Pre/post train	Pre	Post	Pre	Post	Pre	Post	Pre	Post
MOTA ↑	18.6	**83.0**	-8.7	**97.1**	65.4	79.2	7.8	**91.8**
MOTP ↑	26.7	0.231	22.3	18.7	28.9	24.9	23.5	20.3
IDF$_1$ ↑	37.7	**89.7**	47.4	73.2	62.5	78.8	48.0	**77.0**
Number of tracks	30	30	24	24	20	20	74	74
ID switches. ↓	29	12	23	24	39	71	91	107
Fragmentations ↓	213	64	225	58	295	206	733	**733**
AP ↑	08.7	**42.7**	34.2	58.8	22.2	36.0	31.3	**58.9**
AP50 ↑	17.4	**85.5**	63.7	**97.4**	57.5	**74.0**	60.1	**92.9**
APsmall ↑	0.	10.2	-	-	20.0	41.4	0.	**40.7**
APmedium ↑	8.9	**53.4**	34.5	66.7	27.2	55.2	32.4	63.5
False positives ↓	545	453	22410	**366**	279	115	23234	**934**
False negatives ↓	4177	**527**	662	219	1413	855	6252	**1601**

Total FPS: 12

Table 13: Validation performance metrics for FairMOT. Metrics after the optimisation shown Figure 56, 57, and 58 are shown in the ”Post” columns. The change is due to an end-to-end joint training of box estimation heads and re-identification heads. Almost every metric shows an improved number compared to the numbers in the ”Pre” columns. These detail detection and tracking metrics for FairMOT before any optimization. Especially significant improvements are marked in bold. Note that the metrics are obtained by evaluation of the validation set. Refer to Section 3.2.6 for the rationale.

FairMOT Good Detection Demo

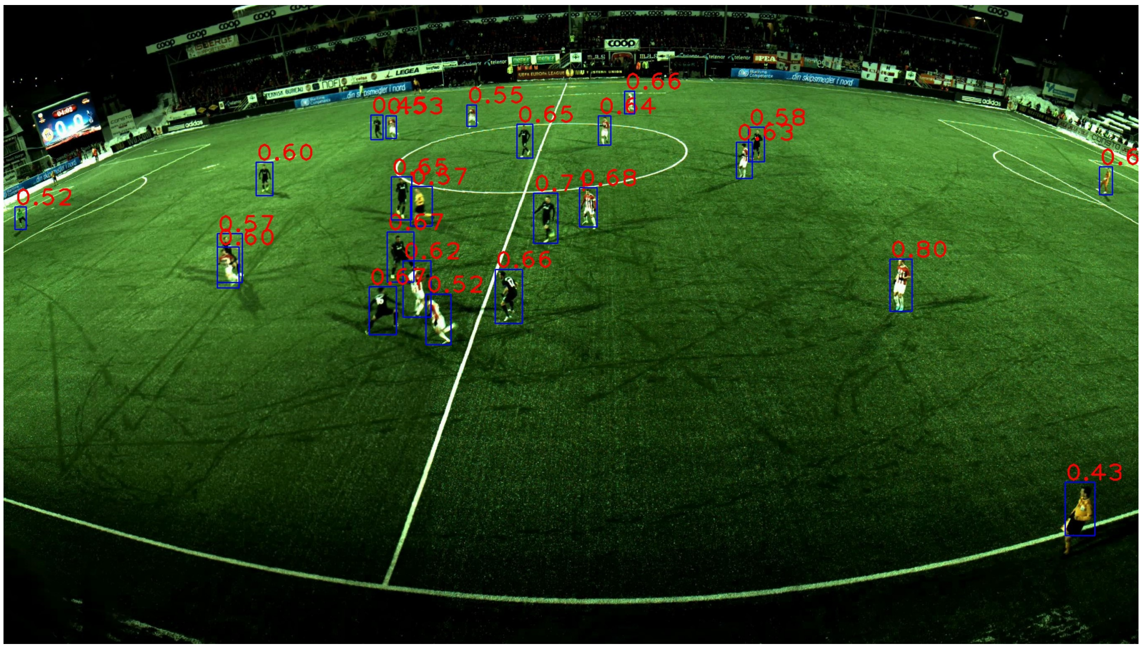


Figure 59: Examples of good detections by the finetuned FairMOT model. All objects are correctly localized on this image from the Alfheim validation set. Most objects proposal confidence scores lies in the range of 0.50-0.70.

FairMOT Bad Detection Demo

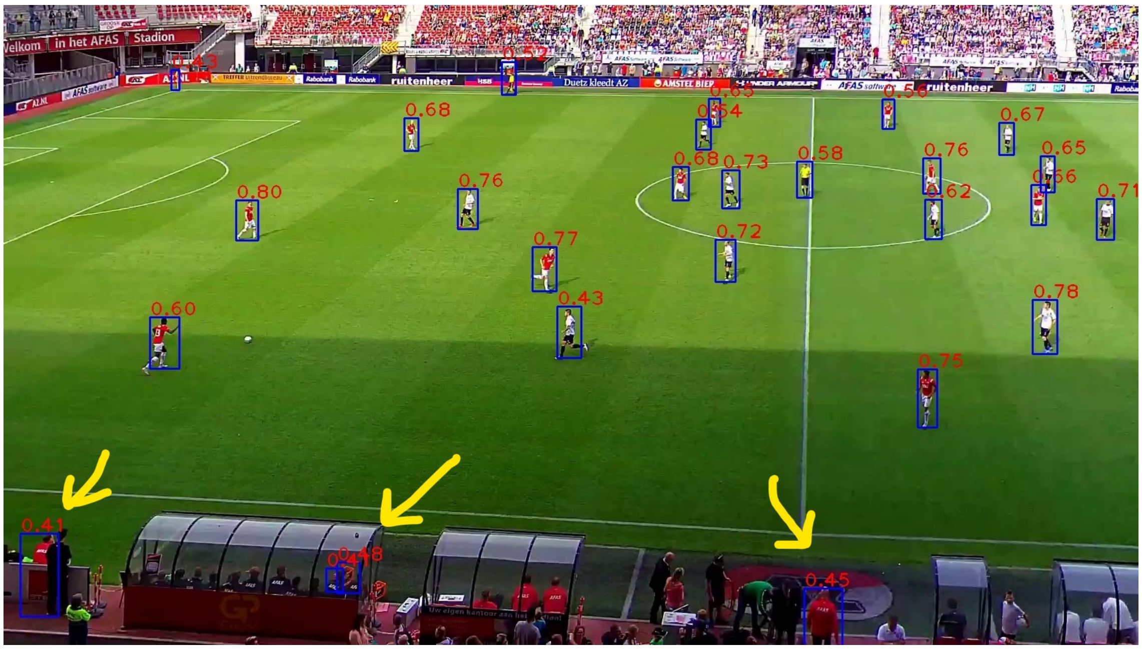
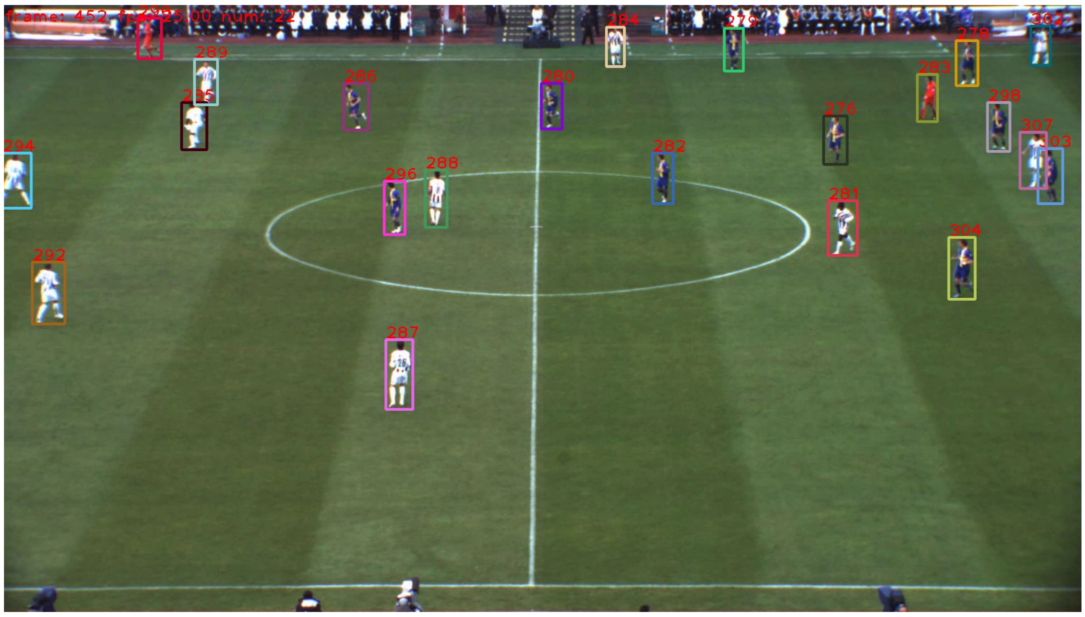
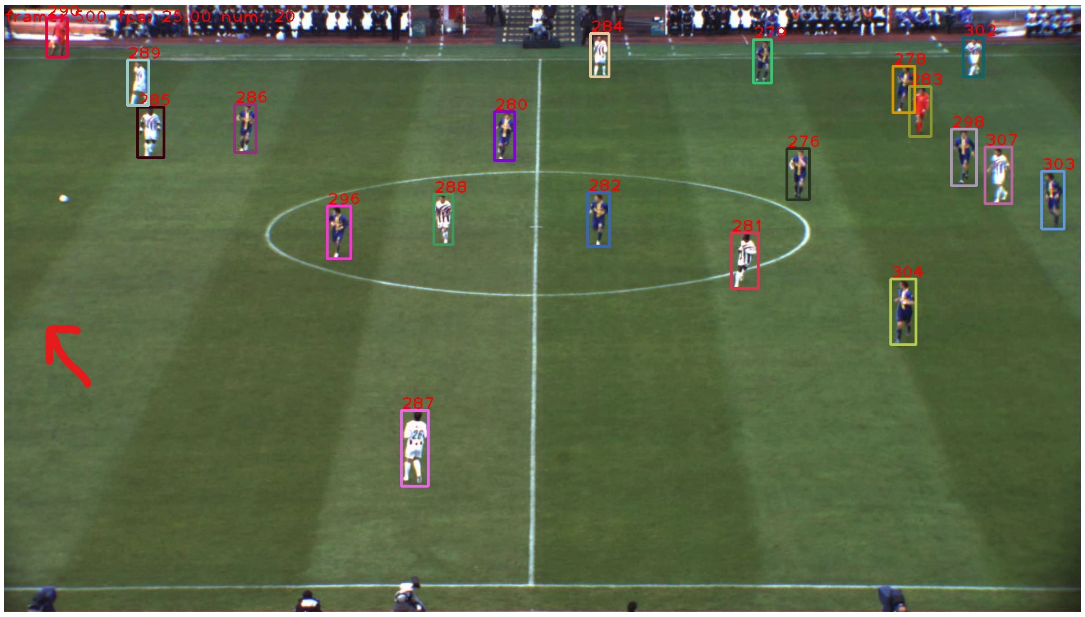


Figure 60: Examples of bad detections by the finetuned FairMOT model. The players are correctly localized, but a lot of false positives are present outside of the field.

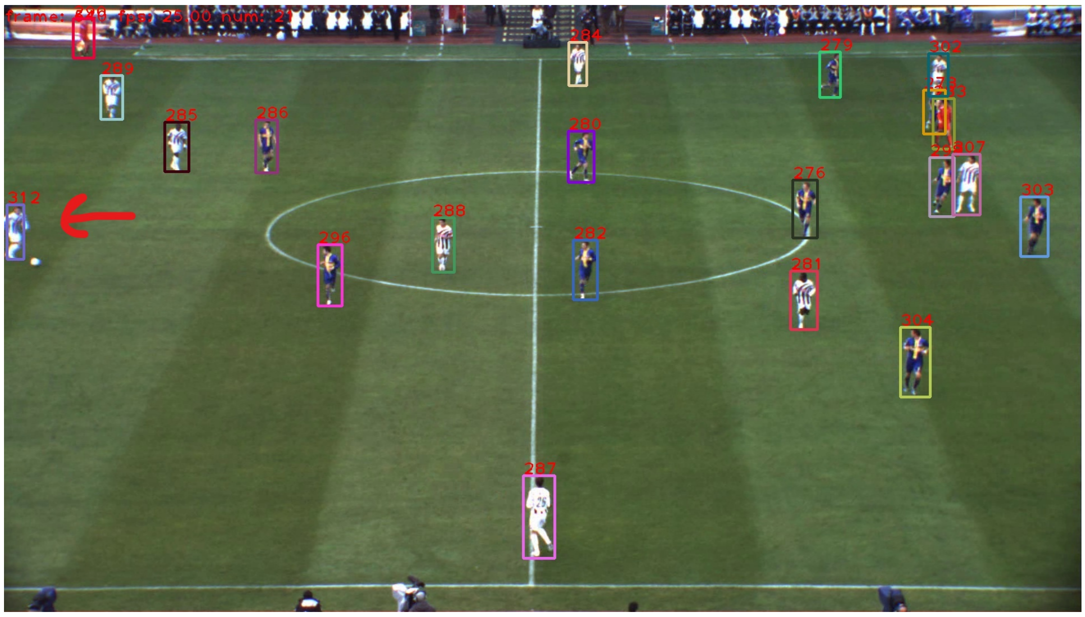
FairMOT Tracking Demo

t=1 

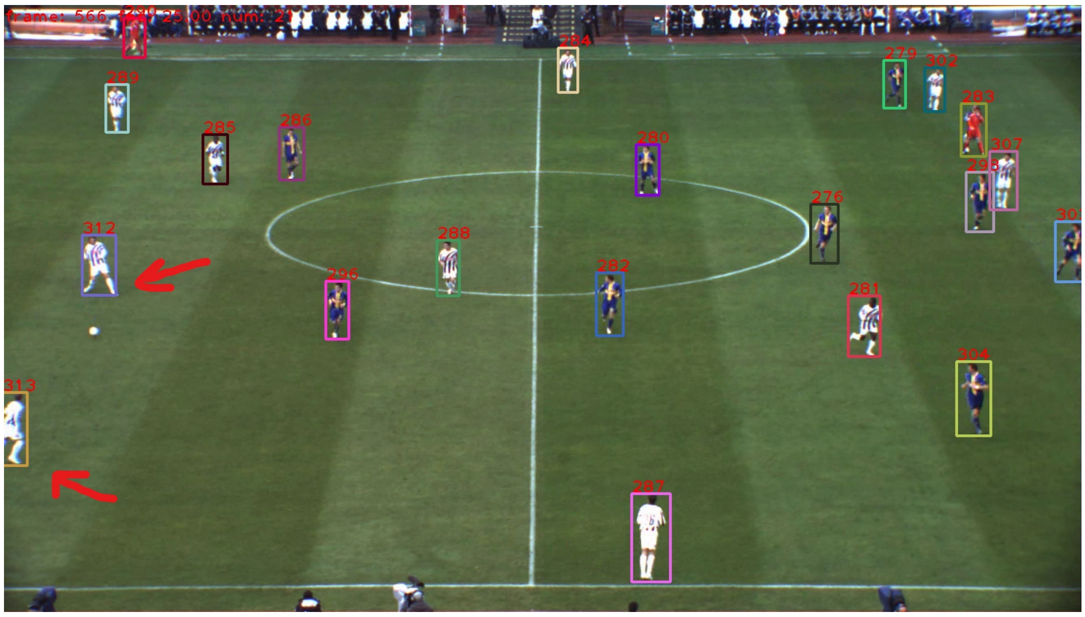
Players are correctly identified in the first frame.

t=2 

Player 294 and 292 has left the scene one second later.

t=3 

Player 294 has re-entered the scene, but is given a new ID.

t=4 

Player 292 has re-entered the scene, but is given a new ID.

Figure 62: FairMOT tracking demo on the ISSIA validation sequence. The data is similar to what the model was trained on. Players are mostly correctly tracked. The whole demo can be viewed `here`, and here [101].

4.4.3 Discussion

From the plots in Figure 56, it is clear that there is a significant jump in detection performance in the first 1000 iterations. This is expected because we set the learning rate to a relatively high value in the first 2K iterations. It would account for the extreme oscillations in those first rounds as well. The learning rate was further set down by a factor of 10 for the last 8K iterations because there were still some unstable curves with a learning rate of 1e-4. Most of the plots seem to be more stable after that. The training curves for the Ranheim sequence differ from the other two sequences. The learning curves are highly unstable, and FairMOT needs a lot more iterations to converge. Observe that the decayed learning rate less affects the curves. There may be several reasons for this. As seen earlier in the text, the Ranheim sequence consists of hard examples where players are small (further away), heavily occluded, and almost indistinguishable even for the human eye. When considering the small fraction of images the Ranheim sequence contributes to the overall dataset, it becomes clear why FairMOT struggles to improve on this particular sequence steadily. FairMOT has a declining AP for small objects on the Alfheim sequence, which makes sense because recall that from Table 5 that there are very objects of this size in the Alfheim sequence.

It is interesting to note the similarity between the MOT curves in Figure 57 and the AP curves of Figure 56. For the MOT curves, one can observe the same degree of oscillations and instability as discussed in the AP plots above. For instance, the Alfheim and ISSIA sequences have the same rapid and stable MOT performance incline as their respective AP counterparts. Furthermore, the Ranheim AP and MOT plots seem to be exact replications of one another, differing in amplitude and equilibrium point. Their oscillations vary very much synchronously. This is highly unlikely to be coincidental and consequently implies just how much FairMOTs tracking capabilities depend on detection accuracy. For instance, at around 13K iterations, there is a large dip in AP50 performance - roughly from 0.7 to 0.5. The corresponding dip in MOTA is from 0.82 to 0.45. That is, in this particular case, the effect AP50 has on MOTA is nearly doubled. This heavy dependence on detection performance is a bit surprising considering that FairMOTs performance on the MOT challenge demonstrates impressive tracking results due to the robustness introduced by its re-identification head. It might imply that FairMOT does not significantly benefit from the re-identification branch on this particular dataset. It might also, however, just be the case for the Ranheim sequence. In Figure 58, observe that, overall, the MOTA and AP are less correlated.

From Table 13, observe that the optimisation process recorded in Figure 56, 57, and 58, has yielded a significant performance increase. Perhaps the most interesting is the jump in MOTA performance, which leaps from 7.8 to 91.8. It seems as most of the increase came from improvements on the Alfheim and ISSIA sequences. Conversely, it also interesting to note how well the initial performance on the Ranheim sequence is, which improves the MOTA from 65.4 to “only” 79.2. Perhaps the players in this sequence resemble the MOT challenge dataset FairMOT was optimized for, which indeed, in both cases, have small objects in very crowded scenarios.

Table 13 also shows that while the improvements in detection are significant, the final result is not perfect. AP50 might be the exception, which reaches an impressive 92.9. Then again, the initial performance was quite good. The detection performance is further confirmed by inspection of the detection demos that are included above. In particular, Figure 59 demonstrates cases where the model localizes every player on the field correctly and does so even when many players are tightly positioned. Nevertheless, if the reader inspects the confidence scores above the proposed bounding boxes, it is clear that most of these lie below 0.80 and sometimes even below 0.50. Figure 60 demonstrates a bad detection example where mostly the same scores can be observed. The image is actually from the additional test set described in Section 3.2, and indeed demonstrates that the detection performance is representative for data beyond what is annotated in this thesis. However, the same image introduces several new challenges for the system. Mainly, non-player humans with a green background regularly confuse the model to output wrongly placed proposals. It was suspected that FairMOT does not have a detection performance that matches the current full-fledged object detectors. After all, it has a different optimization objective to satisfy object detection and re-identification jointly.

Despite detection performance not being perfect, Table 13 reveals that FairMOT is still able to produce impressive tracking results. The MOTA and IDF_1, especially, show great promise. A tracking demo can be seen in Figure 62. There, snapshots from FairMOT’s outputs are overlayed on the ISSIA validation sequence. The figure demonstrates both highlights and downsides of FairMOT’s performance. Observe that across all frames, nearly all players are correctly being tracked with high accuracy. However, at time step t=2, two players have left the scene, and in the subsequent steps, both players have been assigned new IDs. Cases of this type are unfortunately consistent. It is especially disappointing considering that FairMOT was specifically designed to deal with lacking robustness in longer occlusions of objects. Indeed, FairMOT may deal with longer occlusion times more robustly. However, if state estimation uncertainly is too high, which is likely when the object leaves and re-enter the scene, re-identification might become very hard. Adding to this that objects are extremely similar in appearance and have small sizes, the problem may be overwhelmingly complex for FairMOT to handle.

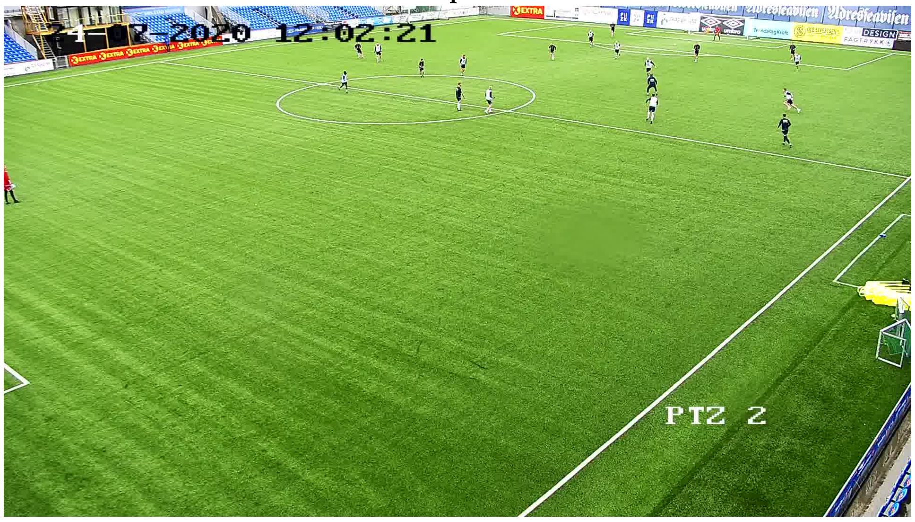
4.5 Experiment 4: Extracting Sports Analytics Data

In this section, different results from attempts to extract sports analytic data will be presented. Particularly, we show how the detection and tracking results from the previous sections could be further processed to deliver such data. The methodology was discussed in Section 3.6.

4.5.1 Obtaining Team Formations and Player Stats

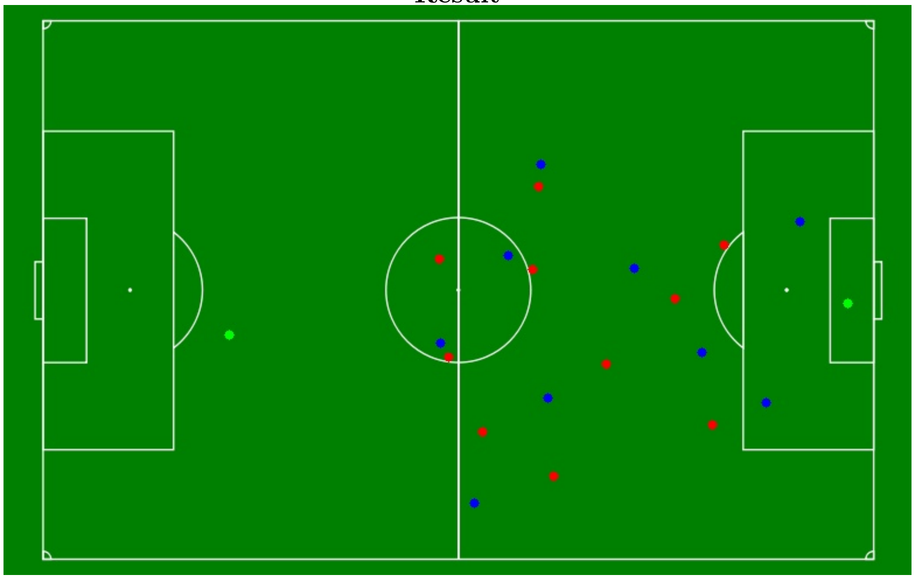
Perspective Transform and Team Association

Input



Example input frame from the Ranheim dataset. The input video is available at `this link` and at [101].

Result



All the outfield players are correctly labeled to each team. The goalkeepers are correctly labeled as outliers. A demo video without interpolating the player position is available at `this link` and at [101].

Figure 63: Overview transform and team association with K-means clustering. The team in black kits is represented by the blue dots and the white team with red dots. Outliers are represented with green.

Distance and Speed Metrics

ID	Total Distance (m)	Avg. Speed (km/h)	Top Speed (km/h)	Top Acceleration (m/s^2)
1	0.96	0.72	3.67	0.65
2	28.3	13.9	27.6	1.29
3	39.7	18.9	29.6	2.30
4	33.4	11.7	19.3	1.46
5	29.2	14.8	26.5	1.96
6	26.4	14.1	25.1	1.17
7	29.4	10.3	15.8	2.66
8	37.3	13.0	17.4	2.42
9	26.2	9.23	15.5	1.69
10	33.3	12.1	26.7	4.27
11	39.7	16.3	24.1	3.23
12	30.1	10.5	18.2	1.77
13	26.9	10.0	14.7	2.10
14	14.4	12.6	20.8	6.26
15	32.6	11.4	16.2	1.71
16	47.9	17.2	24.5	3.89
17	31.9	13.2	20.6	6.01
18	32.4	11.3	16.1	2.15
19	35.2	14.8	22.6	1.06
20	16.2	5.58	11.2	1.96
21	35.4	13.3	20.9	3.64
24	3.43	3.93	5.82	0.65
22	20.7	10.3	12.9	1.77
23	18.7	10.7	17.8	2.89

Table 14: Total distance covered, speed, and acceleration, calculated for each of the players from the example video in Figure 63. The input video with annotated IDs matching the IDs in this table, is available at `this link` and at [101].

Overview Map with Speed Metrics

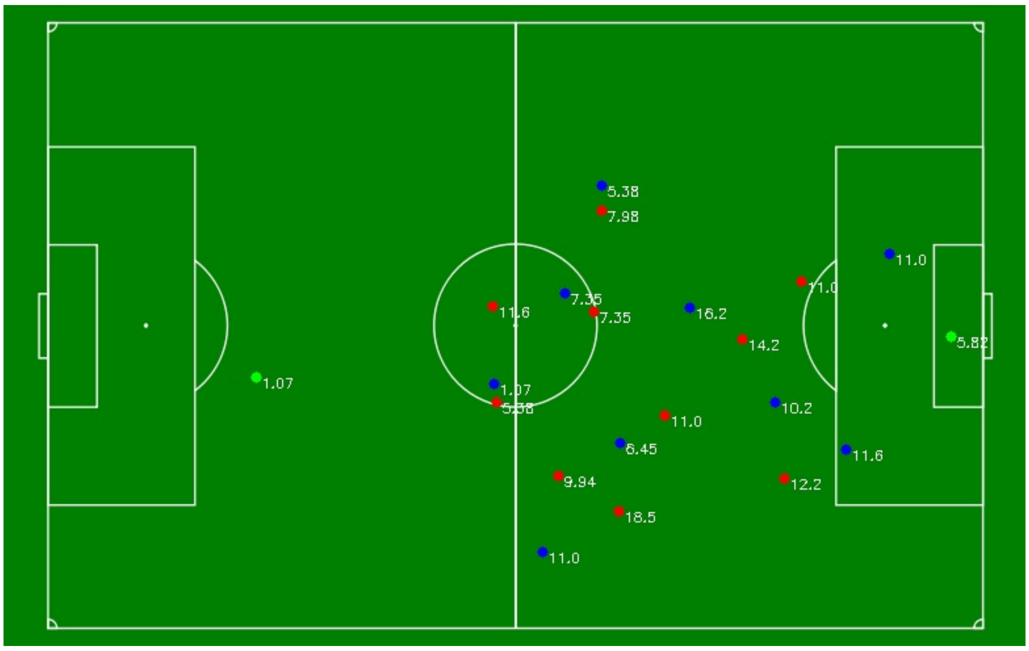
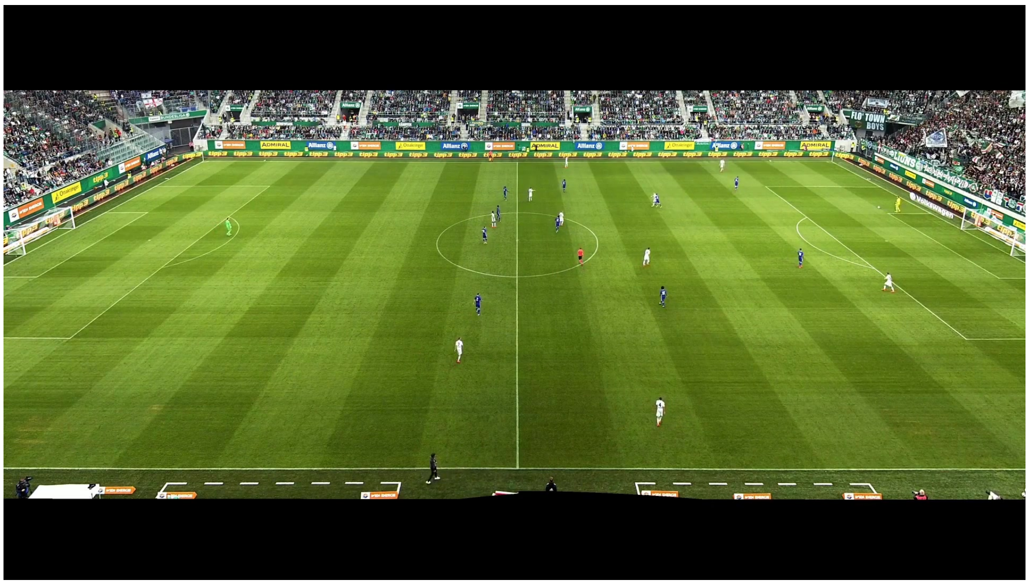


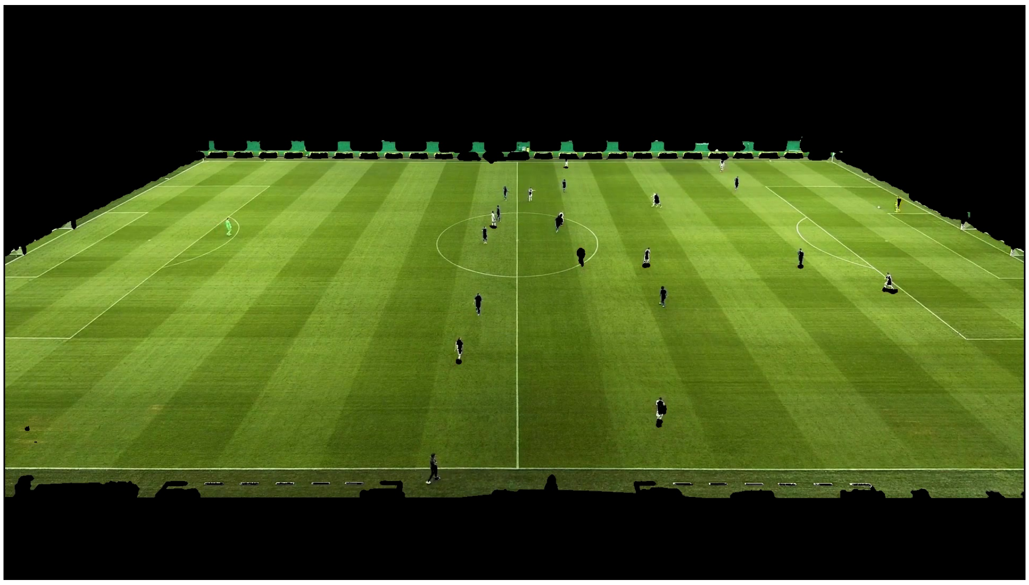
Figure 64: An overview map of player speeds measured in km/h. An video demo is available at `this link` and at [101].

4.5.2 Automatic Detection of Transformation Coordinates

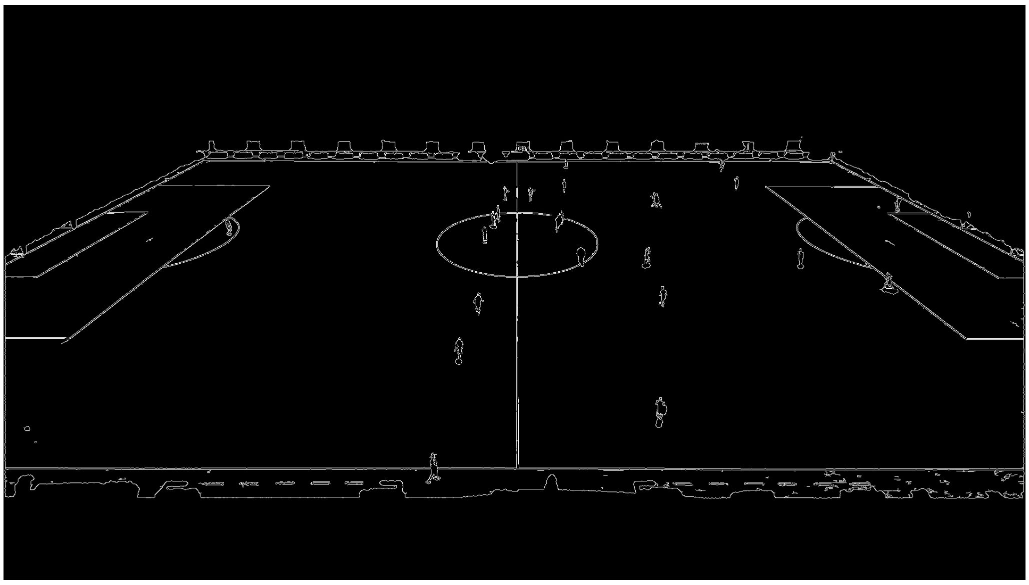
Image Pre-processing



(a) Input image for sports analytics data extraction. The image is retrieved from [90].



(b) Output after applying green mask, largest contour extraction, and opening morphology. The contour is still rough along the edges, despite the opening morphology.



(a) Canny transform applied on the image after the contour bit-mask. Lines from the pitch are clearly visible. The rough edges from the contour are still present. Players are also present, adding further noise to the transformed image.

Figure 66: Results from gradually processing an input frame, such that automatic point estimation for perspective transformation can be realized.

Line Detection

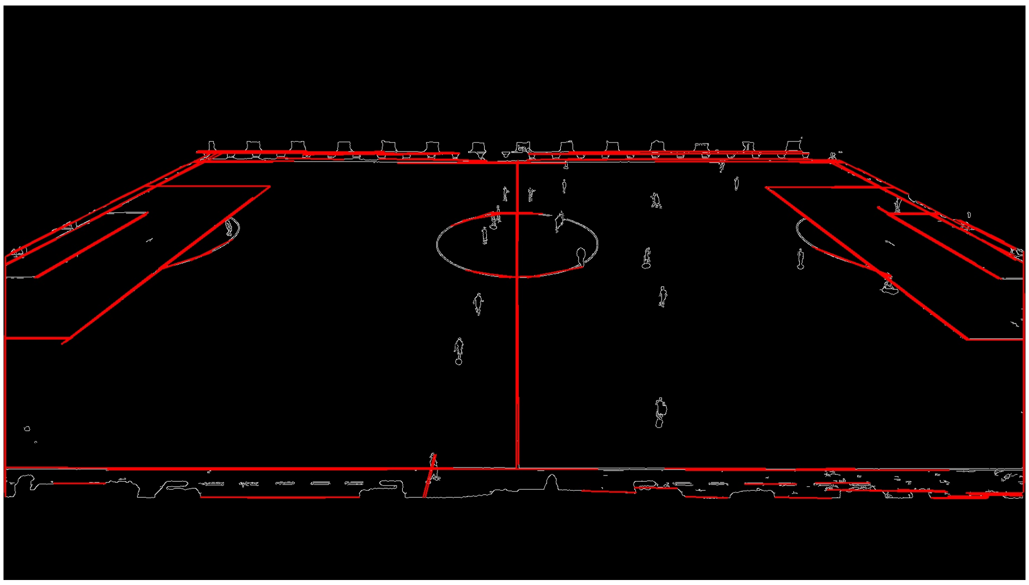
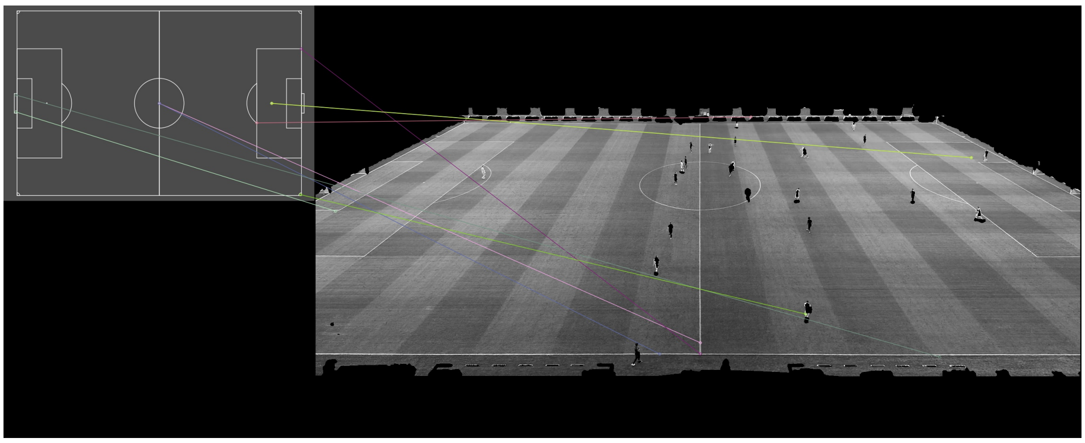


Figure 67: Hough line detection on the pre-processed image. Almost all the lines are detected, but several noisy lines are detected as well. Parts of the center circle and the semi-circles outside the penalty areas are also detected.

Local Feature Detection Algorithms


SIFT applied on the overview map and the input frame after the largest contour was found. Both images were converted to gray before applying SIFT. The drawn lines do not align with the correct points on the overview map.

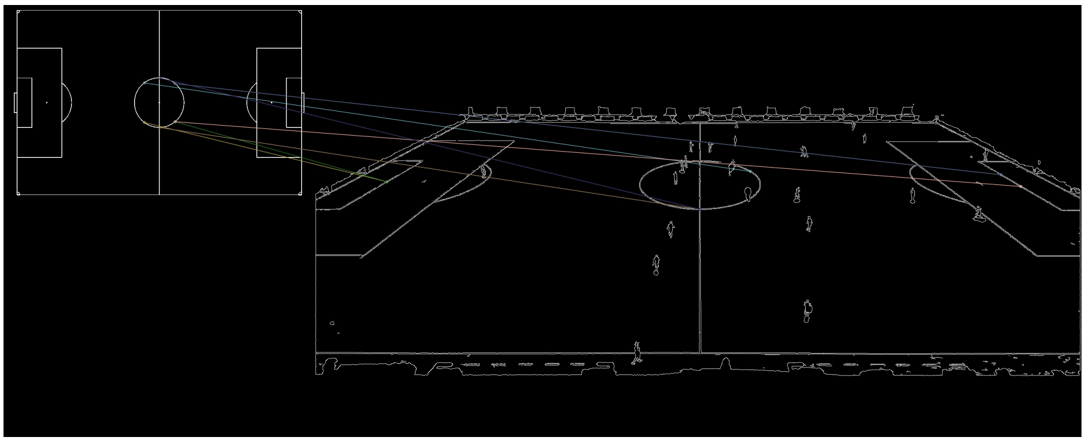

SIFT applied on a black-and-white overview map and the fully processed input frame. The third-to-last bottom line aligns with the desired target accurately, but the rest of the lines do not. The algorithm mistakes the ball for the penalty spot.

Figure 68: SIFT feature detection result

4.5.3 Discussion

Perspective Transform

From the figures above, we can observe that the perspective transformation can deliver realistic and accurate player-to-field coordinates. It is fair to assume that this is largely due to the high player localization precision on the pre-transformed image. It is so precise that one can observe and analyze player formations and, for instance, how the defenders mark the attackers. However, in the video demo linked in Figure 63, which was not interpolated, there are rapid, non-smooth, and unstable movements on the dots that correspond to players furthest away in the test video. As mentioned in Section 3.6.1, this "back-and-forth" movement is caused by slight variations in bounding box proposal placements and sizes. Further, the variations might be amplified during perspective transformation because of the video's flat angle view and the distance between the players and the camera. For instance, if a bounding box was placed on a player's hips instead of their feet, the transformation could, in very skewed situations, move the player meters away from their actual position. For that reason, some way of mitigating this movement, such as interpolation, is essential for more accurate results.

In Table 14, realistic values are observed, coherent with the players' movements in the input video linked in Figure 63. However, these values are not compared against actual metrics. Thus there is no way to validate the accuracy of the results. There are still instances where some unnatural, "back-and-forth" movement occurs, affecting the measurements, but it is not too detrimental to the overall results. Regardless, it is still a useful tool for measuring general player movements, and it demonstrates how the results from our MOT systems could potentially be utilized for sports analytic purposes.

Team Association

Clustering to obtain team membership works, but with a few notable drawbacks. The current method is dependent on the tracker not swapping IDs between the players. The players are clustered and associated to a team upon detection only, so a swap in IDs would not be noticed. An example of this is seen in the video from Figure 63, where player 2 swaps IDs with player 5, but the team associated with the player's ID stays the same. Consequently, the result displays the incorrect team for the players after the ID swap. In this case, K-means is not to blame, but the approach fails whenever the tracking system yields incorrect results. To counter this, one could re-cluster the detections at each frame, or at certain time intervals, at the cost of the system's overall speed. In such a case, K-means also appears to struggle with outliers. While it correctly assigns both keepers as outliers in Figure 40, when performing outlier detection later in the same sequence, the right goalkeeper is assigned as an outfield player. Also, with increased outlier thresholds, some of the outfield players were detected as outliers. An alternative to outlier detection is to add more clusters, but this introduces other issues. For instance, the goalkeepers could have the same kit color as the referees, making it uncertain whether to use three, four, or five clusters. Nevertheless, the $L^*a^*b^*$-features cause K-means to, in most cases, label outfield players correctly. It effectively demonstrates the $L^*a^*b^*$-attribute's usefulness. However, to summarize: the algorithm fails in several scenarios and improvements are necessary.

Automatic Detection of Transformation Coordinates

When inspecting the experiment's overall results, a deep learning method might have been more suited to detect corresponding transformation coordinates with the desired degree of automation. The input frame and the overview map may be similar in lines and shapes of the pitch, but they also have significant differences that complicate the task. The test frame is considerably more complicated than the overview map. It contains more objects, noisy backgrounds, and non-perfect geometric shapes. A non-deep feature detection algorithm could manage a task of this type, given excellent pre-processing. However, it would need to be quite intricate to handle all possible inputs effectively. Although the Hough Lines method rendered lines that could be used for a perspective transform, it would need additional logic concerning which lines to choose, such as finding the longest lines using the Hough space. Still, it is not certain that a method specific to one stadium would work with others. Input streams from the soccer domain contain a notable degree of variety, so the ability to generalize is paramount. Usually, deep learning-based methods handle this well.

5 Comparative Discussion

In the previous section, a thorough discussion was provided for each of the architecture experiments in isolation. There was also a comprehensive discussion of the results from the dataset creation process and the sport analytics results. Here, a comparative discussion of the experiments is provided instead. At the center of the discussion, and indeed the foundation for comparison will be the research questions formulated in Section 1.2. Each subsection will deal with the research questions in the same order as presented in Section 1.2. For easier comparison, an alternative representation of the results from the previous section is included in the tables below. There, metrics essential in discussing the research questions are listed for each architecture used in the experiments. Recall that the numbers in the tables are obtained from evaluating all three optimized MOT-systems on the validation sequences. Refer to Section 3.2.6 on why metrics are only provided for the validation set.

Overall Performance Metrics

Metrics	**Faster R-CNN + SORT**	**YOLOv5 + DeepSORT**	**FairMOT**
MOTA $\uparrow$	**94.6**	93.2	91.8
MOTP $\uparrow$	15.6	16.7	**20.3**
IDF$_1$ $\uparrow$	74.1	76.4	**77.0**
ID switches. $\downarrow$	75	**58**	107
AP $\uparrow$	72.8	**74.9**	58.9
APsmall $\uparrow$	39.6	**46.2**	40.7
AP75 $\uparrow$	82.1	**83.3**	57.7
False positives $\downarrow$	**591**	943	934
False negatives $\downarrow$	**1054**	1179	1601
Detection FPS $\uparrow$	12	**25**	-
Association FPS $\uparrow$	**167**	13	-
Total FPS $\uparrow$	11	9	**12**

Table 15: A comparison of the key performance metrics. The metrics are obtained from evaluating all three optimized MOT-systems on all validation sequences.

Alfheim Performance Metrics

Metrics	**Faster R-CNN + SORT**	**YOLOv5 + DeepSORT**	**FairMOT**
MOTA	**84.7**	80.7	83.0
IDF$_1$	87.3	88.4	**89.7**
ID switches.	12	**6**	12
AP	**51.7**	49.9	42.7
APsmall	16.7	**27.9**	10.2
AP75	**55.1**	53.5	37.5
False positives	**445**	657	453
False negatives	**439**	464	527

Table 16: A comparison of the key performance metrics for the three MOT-systems after final optimization. The metrics are obtained from evaluating on the Alfheim validation sequence.

ISSIA Performance Metrics

Metrics	**Faster R-CNN + SORT**	**YOLOv5 + DeepSORT**	**FairMOT**
MOTA	**98.6**	97.1	97.1
IDF$_1$	71.1	72.7	**73.2**
ID switches.	28	26	**24**
AP	83.1	**85.5**	58.8
APsmall	-	-	-
AP75	96.2	**96.6**	66.7
False positives	**33**	266	366
False negatives	228	322	**219**

Table 17: A comparison of the key performance metrics for the three MOT-systems after final optimization. The metrics are obtained from evaluating on the ISSIA validation sequence.

Ranheim Performance Metrics

Metrics	**Faster R-CNN + SORT**	**YOLOv5 + DeepSORT**	**FairMOT**
MOTA	**89.3**	91.2	79.2
IDF$_1$	71.6	77.6	**78.8**
ID switches.	35	**26**	71
AP	42.4	**47.6**	36.0
APsmall	40.7	**47.0**	41.4
AP75	30.2	**37.3**	29.1
False positives	113	**20**	115
False negatives	**387**	393	855

Table 18: A comparison of the key performance metrics for the three MOT-systems after final optimization. The metrics are obtained from evaluating on the Ranheim validation sequence.

5.1 Detection

RQ1: "Is automatic player *detection* possible from a single-camera soccer video?"

Accurate object detection is an essential prerequisite for the composite task that is multiple object tracking [29]. In that regard, all of the three architectures included in our experiments demonstrated sufficient capabilities for multiple player detection in soccer videos. From Table 15, observe that all architectures managed to achieve an AP of well beyond 50 on the validation data. Faster R-CNN and YOLOv5 showed a roughly equal superiority over FairMOT, with YOLOv5 arguably the slightly better detector of the three. Faster R-CNN does, however, show a better false positive count than its peers by a large margin. The difference may be notable when taking the soccer domain into account. In Section 4, the different MOT systems were also qualitatively tested for detection demo. There, it was also evident that all systems show multiple players detection is obtainable with impressive accuracy.

The same spread in detection performance can be seen in Table 16, 17, and 18, which show the various systems performance on the Alfheim, ISSIA, and Ranheim validation sequences, respectively. For example, again, YOLOv5 does seem to have the better AP metrics, while Faster R-CNN generally has a lower false-positive count than the rest. FairMOT, while obtaining good detection results, performs significantly worse than the other two. As previously mentioned in the discussion of Section 4.4, FairMOTs lower detection performance is expected. Recall that FairMOT has a different optimization objective than the other two detectors, namely balancing object localization and producing descriptive re-ID features for better tracking performance. Although there is a consistent ranking between MOT-systems across the validation sequence performances, the actual values between the validation tables differ significantly. Performance metrics for the Ranheim sequence are the most notable, representing a clear drop in detection performance for all detectors. To summarize previous discussions, it is most likely due to a severe under-representation of images of this type in the training set. To make matters worse, players from this dataset are small, have similar appearances, and are heavily occluded. Conversely, all models perform exceptionally well on the ISSIA validation sequences. Images from the ISSIA dataset are abundant and make up a large portion of the training set. Additionally, the objects of this dataset represent much "easier" examples than the ones seen in the Ranheim dataset.

The detection demos from the previous sections showed that a good portion of the false positives seen in Figure 15 could be attributed to the placement of boxes to non-player humans near the pitch. These include assistant referees, coaches, benchwarmers, supporters. Mostly, they belong to the assistant referees. The probable reason is that the ISSIA sequence purposely labeled these indistinguishable from the players on the field. Unfortunately, we only realized this too late, and it would have been an overwhelming workload to remove this manually. As a consolation, many of these have lower confidence scores than the actual players on the fields, and the false positive count may be reduced by a more careful selection of confidence thresholds. To summarize, a means to filter false positives outside the pitch would be required to increase robustness in this aspect. Similarly, all detectors fail to detect at least 1000 objects in the entire validation set, which has a total playtime of about 2 minutes. While not a game-breaking number, it might cause fragmentation issues for the complete tracking system, which may have significance in the soccer domain. Again, a means to post-process such missed detections may be the solution to increasing robustness in this. The obvious solution to improving both false positive and missed detection counts is to optimize the detectors further. However, it requires more (diverse) soccer footage to be labeled and an inefficient means to increase robustness. The use of domain-specific post-processing methods could be both efficient to implement and more effective. We discuss these further in future work efforts in Section 6.1.

5.2 Tracking

RQ2: "Is automatic player *tracking* possible from a single-camera soccer video?"

Above, the detectors of this work were established to supply the foundational basis for tracking: namely, localizing every player on the field with high accuracy - in every single frame of a soccer video. The next order of business is to discuss tracking performance when these detectors are turned into complete multiple object tracking systems.

The MOTA scores in tables 15, 16, 17, and 18 suggests that tracking is nearly flawless for all MOT-systems. It is not the full story, however. In particular, observe that the number of ID switches is not insignificant. The problem is further reinforced when considering that the validation set is about 2 minutes of video footage with roughly 80 players to track. Such a high proportion of switches may be insufficient when using tracking results for analysis in a soccer match. The IDF$_1$ and, especially, the MOTP values show more conservative results that better reflect this high proportion of ID switches. From the tracking demos of the experiments section, it is clear that longer durations of player absence cause most ID switches. For example, whenever there is an event that causes high player clustering, occlusions are introduced, and the detector may suffer from missed detections in many subsequent frames (see Figure 47). In this particular case, the tight compacting of players worsen the difficulty and is also cause to most of the ID transfers seen (see Figure 55). For sequences that do not fully view the pitch, longer absence durations are due to players frequently leaving and re-entering the scene. ID switches are consistently introduced in such cases (see Figure 62).

It is interesting to note the similar MOT performances that the different systems display. All tables above and the tracking demos from Section 4 confirm this. Faster R-CNN and SORT's performance is nearly interchangeable with the performance of YOLOv5 and DeepSORT. While their similarity in detection performance aligns with high rankings in benchmark object detection competitions, the tracking performance similarity justifies further commenting. DeepSORT is known to produce better results due to the inclusion of appearance features when associating detections with existing tracks. Recall from Section 4.3.3 that a discussion was provided on the disappointing fact that DeepSORT fails to contribute to improvements to tracking performance. It is the most obvious cause for the tracking similarity between the two systems. It seems that when DeepSORT fails to make use of the re-ID embeddings it produces, it is reduced to the baseline tracking performance of SORT. Consequently, a pairing of YOLOv5 with SORT would probably produce very similar results, if not even better.

Even FairMOT has highly comparable tracking results, despite having significantly worse detection performance. FairMOT is a more focused model than its peers since it regards tracking as the end goal instead of detection or association. It might explain the comparative tracking results despite having the lowest detection performance. For YOLOv5 and Faster R-CNN, the MOT validation curves in their respective experiment sections revealed a clear dependence on detection performance. Contrarily, FairMOT showed less covariance in overall AP and MOTA and seemed to be generally less impacted by heavy shifts in detection performance. On the other hand, from Section 4.4.3, we also discussed that the same graphs revealed just by how much FairMOT's tracking is dependent on detection performance. Despite the comparable results of FairMOT, it is by no means the better tracker of the lot. While being on top in terms of IDF_1 and MOTP, it is consistently the weakest in MOTA and ID switches. Conversely, even though FairMOT outperforms the competition in the MOT Challenge, the superiority is not noticeable in the soccer domain. As touched upon in Section 4.4.3, the reason may be related to the quality of the dataset.

From the tracking demos in the Experiments & Results section, it is still clear that tracking mostly works to a high degree. The inadequacy of the resulting ID switch count is undoubtedly reinforced by the strictness that tracking a game of soccer introduces. Considering that there is no domain-specific knowledge in the applied algorithms, all tracking architectures displayed encouraging results that promote further work. The tracking demos even displayed high usability in video sequences beyond what was annotated in this thesis. An additional impressive caveat of these various MOT systems is the high performance they offer out of the box. Although the MOTA score might disagree, a qualitative inspection of tracking on actual soccer footage revealed that these models offer decent tracking without optimization. This was excellent news since it allowed for adopting these systems early on for the semi-automatic labeling approach, discussed in Section 3.2.5. The use of YOLOv5 and DeepSORT was especially favorable due to their superior off-the-shelf performance.

From the architecture experiments of Section 4, there was consistently a drop in MOTP for every architecture when comparing values before and after optimization. While indeed suspicious, we argue that the value gets worse because the tracking accuracy improves. As discussed earlier, every architecture showed a drastic decrease in false negatives. Consequently, more objects are being evaluated by the MOTP metric after optimization, and there is not necessarily an equally large increase in bounding box overlap precision.

5.3 Real-time Tracking

RQ3: "Is automatic player tracking from a single-camera soccer video achievable in real-time?"

Keeping in mind that footage typically has an FPS of 25, Table 15 reveals that the combination of detectors and association methods in our experiments does not demonstrate real-time usage. Zhang et al. [32] advocates for the real-time processing times of FairMOT, but that assumes input image sizes of half the ones that this thesis employed. Indeed, this was confirmed by rescaling the soccer footage to a 1086x568 resolution. However, at this input size, much contextual information is lost. It was decided to keep the original resolution in favor of additional tracking accuracy, considering the hard object examples that the soccer footage in this thesis introduces. Furthermore, it was desirable for FairMOT to process feature-rich images to produce equally feature-rich appearance descriptors for improved tracking results. Separate detection and association FPS was not obtainable from the FairMOT implementation. Due to its closely coupled construction, only the total system FPS is obtainable.

For YOLOv5 and DeepSORT, DeepSORT is the bottleneck for the system with its disappointing

FPS of 13. Being the most mature detector of the three, Faster R-CNN clocked in at an expected FPS of 12, which is also the bottleneck for its respective system. From discussing the tracking results above, we noted that different pairings of detectors and association models would most probably produce the same results. Considering this, a pairing of YOLOv5 and SORT would construct an MOT system capable of real-time processing speeds. Therefore, multiple object tracking from soccer videos is arguably achievable in real-time.

Furthermore, the YOLOv5 implementation is highly configurable in terms of model complexity. In fact, during testing, we found that specific models yielded nearly the same detection accuracy at half the processing speeds. Consequently, a YOLOv5 and SORT configuration are theoretically well within the real-time threshold. This leaves a large room for additional complexity placed elsewhere in the algorithm, such as adding domain-specific post-processing methods.

5.4 Extracting Sports Analytics Data

RQ4: "Do the applied methods display usability in sports analytics applications?"

The results from Section 4.5 are some steps behind what can be obtained from a complete, top-class sports analytics tool. However, the features show how to use the MOT systems described in this thesis for generating necessary baseline data in the sports analytics domain. The overview map demonstrates that it is possible to represent player positions and team formation using a single-camera setup. The distance and speed metrics demonstrate how valuable sports analytics results can be extracted from the overview map. However, there is room for accuracy improvements, mitigating the "back-and-forth" movement caused by the bounding boxes and fine-tuning the distance and speed metrics.

Notably, with the current methods, the player-ID-specific features are dependent on the tracking performance, and the positional features are dependent on the detection performance. Thus, to improve, more accurate detection and tracking would help. A means to detect and counter-act when the MOT system makes a mistake would equally be of use. Nonetheless, the methods illustrate that our MOT-systems' output can be applied for creating sports analytics features.

6 Conclusion & Future Work

In conclusion, we have successfully been able to assess recent computer vision techniques on single-camera soccer videos. We argue that insights from this thesis are of high value for the further research of producing sport analytics data from such a setup. We have established that three highly profiled deep learning-based object detectors can nearly perfectly detect players on frames from single-camera soccer footage. *This answers the first research question. To address the second research question*, we conclude that the assessed multiple object tracking algorithms can track soccer players on footage filmed with a single camera, but not necessarily of the sufficient quality required by sports analytics. For this to be realized, further work in the field is required. For instance, it could be achieved either by increasing training set quality or incorporating domain-specific knowledge into the algorithms. During the assessment, we jointly identified that the feature descriptors from the deep association networks do not significantly contribute to soccer player tracking, at least not on the dataset utilized here. We also found that tracking of this quality is achievable in real-time, *answering the third research question.* We also demonstrated how domain-specific features could be extracted from the tracking results and used in downstream sports analytics applications. These include bird eye-view transformations of the players and pitch to view team formations and extract player speeds. *This concludes our fourth and final research question.* In addition to meeting our research goals, we applied and assessed a proposed semi-automatic labeling approach for machine learning-based MOT tasks. The proposed method significantly reduced labeling efforts and should consequently be further investigated. As the last contribution, we also proposed a method to obtain MOT metrics during the optimization of MOT components. The method gave us great insight into how the optimization of individual components has a system-wide effect on MOT performance. To our knowledge, such a method is never detailed in similar works, and while simple to implement, it bears significance that justifies attention from future work.

6.1 Future Work

Like so many thesis authors before us, there comes the point where one realizes that there is not enough time to explore everything that was planned. This thesis is no exception. While the body of this work is indeed extensive, it is still a mileage away from supplying algorithms that could be directly used in potential soccer analytics applications. Here, the aim is to highlight the work's key problems and possible solutions for mitigating them. Next, a list of most of the interesting possible extensions to this work is presented. These are consistent with the ones we indeed had in mind to explore but could not due to time restrictions.

Dataset Quality

Dataset quality is arguably the biggest drawback of this thesis. Firstly, as discussed in Section 4.1.3, there is a severe imbalance in the dataset in regards to where the images originate from. Consequently, most of the images are very similar to one another. Related to this is the lack of diversity. We only managed to source our material from three different places, and images from the same source appear very much the same, differing primarily in object appearance and object location. The background remains largely the same. More efforts in labeling sequences from different sources, and of equal sizes, are needed to increase generalisation capabilities and robustness in the deep learning-based methods employed. If enough data had been labeled, an additional test set could also have been included.

Improving Deep Learning Models

On the model side of things, there are various knobs to tweak that could increase generalization performance, requiring further attention. For instance, hyperparameter tuning was only conducted to a limited degree. There exist various approaches that efficiently test the different configurations to approximate optimal models. In the Appendix, there are exhaustive lists of the different parameters to adjust for each model. Future endeavors should refer to these for inspiration. Freezing early weight layers would probably decrease optimization times and maybe even at no cost in performance, due to the high generality of these early layers. However, this needs further testing to confirm. Both the FairMOT and Faster R-CNN implementations let the user alternate between different backbones. It might be worthwhile to experiment with different combinations.

Improving Tracking

The possible improvement mentioned above applies to the association methods employed in this thesis, such as tuning various settings and improving dataset quality. However, the biggest issue for these methods is probably attributed to the lack of any domain-specific adjustments. The need for adjustments arises from the closed world scenario that a game of soccer is. For example, all objects of interest are at all times during the sequence within the pitch lines. Consequently, players can not suddenly disappear forever in the middle of the field. In addition, a means to automatically filter out-of-the-pitch detections could potentially remove a lot of false positives. Also, rarely are new objects of interest introduced, and thus initially available ID tags, and their destruction, should be restricted and frequently re-used. Conversely, if 21 out of 22 players are successfully re-identified, it is fairly obvious what the remaining player's ID should be. The same logic could be applied if a player has moved out of the scene: it could be forced to choose from a pool of available ID tags instead of consistently deleting and assigning new IDs. Implementing such simple rules may significantly increase tracking performance in the soccer domain and is consequently highly encouraged to investigate.

Improving Sports Analytics Data Extraction

Regarding extracting data for sports analytics purposes, two main concepts could significantly improve the data. Firstly, one could improve team association by implementing a deep learning-based method for separating teams, goalkeepers, and referees. It can either be accomplished by training a completely new classifier or training our existing detectors by adding a new head, or changing the output to support multi-class labeling. Secondly, one could implement a deep keypoint detection method to automatically perform the perspective transformation. With keypoint detection, it would not be necessary to have all the critical areas of the pitch visible, which is often the case using single-camera setups. The keypoints could then continuously be adjusted to account for a moving camera. Regarding non-deep CV methods, performing template matching could be an effective option. Additionally, using a form of ellipse detection for detecting the center circle, which is skewed by the camera angle, would also be an enticing next step.

Bibliography

[1] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine, V. Vasudevan, W. Han, J. Ngiam, H. Zhao, A. Timofeev, S. Ettinger, M. Krivokon, A. Gao, A. Joshi, Y. Zhang, J. Shlens, Z. Chen, and D. Anguelov, "Scalability in perception for autonomous driving: Waymo open dataset," 2019.

[2] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," *CoRR*, vol. abs/1505.04597, 2015. [Online]. Available: http://arxiv.org/abs/1505.04597

[3] "Cvsports — 6th international workshop on computer vision in sports (cvsports) at cvpr 2020." [Online]. Available: https://vap.aau.dk/cvsports/

[4] "Sport analytics," 2021. [Online]. Available: https://en.wikipedia.org/wiki/Sports_analytics

[5] C. Custance, "Analytics give blackhawks an edge," 2015. [Online]. Available: https://www.espn.com/nhl/story/_/id/12370826/analytics-give-blackhawks-advantage

[6] "Astros' world series win may be remembered as the moment analytics conquered mlb for good," 2017. [Online]. Available: https://www.washingtonpost.com/sports/astros-world-series-win-may-be-remembered-as-the-moment-analytics-conquered-mlb-for-good/2017/11/02/ac62abaa-bfec-11e7-97d9-bdab5a0ab381_story.html

[7] "The evolution and future of analytics in sport," 2015. [Online]. Available: https://www.linkedin.com/pulse/evolution-future-analytics-sport-sugato-ray

[8] "The great analytics rankings," 2016. [Online]. Available: http://www.espn.com/espn/feature/story/_/id/12331388/the-great-analytics-rankings#mlb-hou

[9] "Piero sports graphics analysis — cg graphics systems," Ross Video. [Online]. Available: https://www.rossvideo.com/products-services/acquisition-production/cg-graphics-systems/piero-sports-graphics-analysis/

[10] G. Thomas, R. Gade, T. Moeslund, P. Carr, and A. Hilton, "Computer vision for sports: Current applications and research topics," *Computer Vision and Image Understanding*, vol. 159, 04 2017.

[11] "Sportvu," Stats Perform. [Online]. Available: https://www.statsperform.com/team-performance/football/optical-tracking/

[12] T. Wolverton, "Big data meets big-time basketball," The Mercury News, 05 2014. [Online]. Available: https://www.mercurynews.com/2014/05/17/big-data-meets-big-time-basketball/

[13] "LaLiga's mediacoach offers real-time performance data in the cloud." [Online]. Available: https://www.sportskeeda.com/football/laliga-s-mediacoach-offers-real-time-performance-data-in-the-cloud

[14] "Tracab technologies." [Online]. Available: https://tracab.com/products/tracab-technologies/

[15] "Four tracking systems given quality certificates by fifa." [Online]. Available: https://trainingground.guru/articles/four-tracking-systems-given-quality-certificates-by-fifa

[16] "DFL X TRACAB." [Online]. Available: https://tracab.com/article/dfl-and-chyron-sign-new-tracab-contract-for-bundesliga-and-bundesliga-2/

[17] S. Ellens, D. Hodges, S. McCullagh, J. Malone, and M. Varley, "Interchangeability of player movement variables from different athlete tracking systems in professional soccer," *Science and Medicine in Football*, 01 2021.

[18] "Second spectrum," Second Spectrum. [Online]. Available: https://www.secondspectrum.com/ourwork/teams-leagues.html

[19] “MLS data deal adds stadium cameras for player tracking and prop bets.” [Online]. Available: https://frontofficesports.com/mls-second-spectrum-2/

[20] “Kameraer som ser alt forandrer sporten: Hvordan skal løsningen tilpasses lag med svakere økonomi?” 2021. [Online]. Available: https://www.digi.no/artikler/datasyn-erobrer-sporten-neste-store-skritt-blir-a-fa-det-til-a-fungere-i-serie-2/493815?utm_source=newsletter-tudaily&utm_medium=email&utm_campaign=newsletter-2020-08-06&key=OxTjaRD4

[21] A. P. A. Gudnason, Dan Oved, “Estimating 3d poses of athletes at live sporting events — the new york times - research development,” rd.nytimes.com, 06 2020. [Online]. Available: https://rd.nytimes.com/projects/estimating-3d-poses-of-athletes-at-live-sporting-events

[22] J. Gilligan, “How gps technology is changing rugby,” www.sporttechie.com, 2014. [Online]. Available: https://www.sporttechie.com/how-gps-technology-is-changing-rugby/

[23] X. Zhou, D. Wang, and P. Krähenbühl, “Objects as points,” 2019.

[24] R. E. Kalman, “A new approach to linear filtering and prediction problems,” *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[25] “Hough transform.” [Online]. Available: https://en.wikipedia.org/wiki/Hough_transform

[26] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” 2016.

[27] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” 2016.

[28] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015. [Online]. Available: https://doi.org/10.1038/nature14539

[29] W. Luo, J. Xing, A. Milan, X. Zhang, W. Liu, and T.-K. Kim, “Multiple object tracking: A literature review,” *Artificial Intelligence*, vol. 293, p. 103448, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370220301958

[30] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, “Simple online and realtime tracking,” *2016 IEEE International Conference on Image Processing (ICIP)*, Sep 2016. [Online]. Available: http://dx.doi.org/10.1109/ICIP.2016.7533003

[31] N. Wojke, A. Bewley, and D. Paulus, “Simple online and realtime tracking with a deep association metric,” 2017.

[32] Y. Zhang, C. Wang, X. Wang, W. Zeng, and W. Liu, “Fairmot: On the fairness of detection and re-identification in multiple object tracking,” 2020.

[33] S. Norvig, *Artificial Intelligence : A Modern Approach.* Pearson, 2018.

[34] M. A. Nielsen, “Neural networks and deep learning,” Neuralnetworksanddeeplearning.com, 2018. [Online]. Available: http://neuralnetworksanddeeplearning.com/

[35] L. Bottou, “Large-scale machine learning with stochastic gradient descent,” pp. 177–186, 2010.

[36] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2017.

[37] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, “Backpropagation applied to handwritten zip code recognition,” *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[38] “An intuitive guide to convolutional neural networks,” 3 2018. [Online]. Available: https://www.freecodecamp.org/news/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050/

[39] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei, "Deformable convolutional networks," 2017.

[40] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[41] F. Yu, D. Wang, E. Shelhamer, and T. Darrell, "Deep layer aggregation," *CVPR*, p. 2403–2412, 2018.

[42] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," 2017.

[43] J. R. Uijlings, K. E. Sande, T. Gevers, and A. W. Smeulders, "Selective search for object recognition," *Int. J. Comput. Vision*, vol. 104, no. 2, p. 154–171, Sep. 2013. [Online]. Available: https://doi.org/10.1007/s11263-013-0620-5

[44] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," 2014.

[45] "Imagenet." [Online]. Available: https://www.image-net.org/

[46] M. Everingham, L. Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, pp. 303–338, 2009.

[47] M. A. Sadeghi and D. Forsyth, "30hz object detection with dpm v5," in *Computer Vision – ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds. Cham: Springer International Publishing, 2014, pp. 65–79.

[48] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," 2016.

[49] ——, "Yolov3: An incremental improvement," 2018.

[50] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," 2020.

[51] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft coco: Common objects in context," 2015.

[52] G. Jocher, "ultralytics/yolov5: v5.0 - YOLOv5-P6 1280 models, AWS, Supervise.ly and YouTube integrations," Apr. 2021. [Online]. Available: https://doi.org/10.5281/zenodo.4679653

[53] J. Nelson and J. Solawetz, "Yolov5 is here," 2020. [Online]. Available: https://blog.roboflow.com/yolov5-is-here/

[54] C.-Y. Wang, H.-Y. M. Liao, I.-H. Yeh, Y.-H. Wu, P.-Y. Chen, and J.-W. Hsieh, "Cspnet: A new backbone that can enhance learning capability of cnn," 2019.

[55] G. Jocher, "Csp backbone in yolov5," 2020. [Online]. Available: https://github.com/ultralytics/yolov5/issues/548

[56] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, "Path aggregation network for instance segmentation," 2018.

[57] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "Scaled-yolov4: Scaling cross stage partial network," 2021.

[58] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," 2018.

[59] "Affinity matrix." [Online]. Available: https://deepai.org/machine-learning-glossary-and-terms/affinity-matrix

[60] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, pp. 83–97, 1955. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800020109

[61] Leal-Taixé, L. Milan, A. Reid, I. Roth, and S. Schindler, "MOTChallenge 2015: Towards a benchmark for multi-target tracking."

[62] A. Rosebrock, "Intersection over Union (IoU) for object detection." [Online]. Available: https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

[63] J. Hui, "map (mean average precision) for object detection." [Online]. Available: https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173/

[64] K. Bernardin and R. Stiefelhagen, "Evaluating multiple object tracking performance: the clear mot metrics," *EURASIP Journal on Image and Video Processing*, vol. 2008, pp. 1–10, 2008.

[65] E. Ristani, F. Solera, R. Zou, R. Cucchiara, and C. Tomasi, "Performance measures and a data set for multi-target, multi-camera tracking," *ECCV. Springer*, vol. 2008, p. 17–35, 2016.

[66] R. J. Tan, "Breaking down mean average precision (map)." [Online]. Available: https://towardsdatascience.com/breaking-down-mean-average-precision-map-ae462f623a52

[67] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick, "Detectron2," 2019. [Online]. Available: https://github.com/facebookresearch/detectron2

[68] "PyTorch." [Online]. Available: https://pytorch.org/features/

[69] "detectron2model_zoo package detectron2 0.3 documentation," detectron2.readthedocs.io. [Online]. Available: https://detectron2.readthedocs.io/modules/model_zoo.html

[70] "Computer Vision Annotation Tool (CVAT)." [Online]. Available: https://github.com/openvinotoolkit/cvat

[71] "What is a container?" 2013. [Online]. Available: https://www.docker.com/resources/what-container

[72] E. Monier Ibrahim, P. Wilhelm, and U. Rückert, "A computer vision based tracking system for indoor team sports," 01 2009.

[73] S. Hurault, C. Ballester, and G. Haro, "Self-supervised small soccer player detection and tracking," in *Proceedings of the 3rd International Workshop on Multimedia Content Analysis in Sports*, ser. MMSports '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 9–18. [Online]. Available: https://doi.org/10.1145/3422844.3423054

[74] J. Komorowski, G. Kurzejamski, and G. Sarwas, "Footandball: Integrated player and ball detector," 01 2020, pp. 47–56.

[75] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," *Lecture Notes in Computer Science*, p. 21–37, 2016. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-46448-0_2

[76] T. D'Orazio, M. Leo, N. Mosca, P. Spagnolo, and P. L. Mazzeo, "A semi-automatic system for ground truth generation of soccer video sequences," *2009 Sixth IEEE International Conference on Advanced Video and Signal Based Surveillance*, pp. 559–564, 2009.

[77] K. Lu, J. Chen, J. J. Little, and H. He, "Light cascaded convolutional neural networks for accurate player detection," 2017.

[78] R. Theagarajan, F. Pala, X. Zhang, and B. Bhanu, "Soccer: Who has the ball? generating visual analytics and player statistics," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2018, pp. 1830–18 308.

[79] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[80] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," 2016.

[81] S. Giancola, M. Amine, T. Dghaily, and B. Ghanem, "Soccernet: A scalable dataset for action spotting in soccer videos," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, Jun 2018. [Online]. Available: http://dx.doi.org/10.1109/CVPRW.2018.00223

[82] R. Arandjelović and A. Zisserman, "All About VLAD," *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1578–1585, 2013.

[83] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," 2018.

[84] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," 2018.

[85] "Remote Development using SSH," Microsoft. [Online]. Available: https://code.visualstudio.com/docs/remote/ssh

[86] A. Cioppa, A. Deliege, N. U. Huda, R. Gade, M. Van Droogenbroeck, and T. B. Moeslund, "Multimodal and multiview distillation for real-time player detection on a football field," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2020.

[87] S. A. Pettersen and D. Johansen, "Soccer video and player position dataset," *Proceedings of the International Conference on Multimedia Systems (MMSys)*, vol. Singapore, pp. 18–23, 2014.

[88] "Panofield panorama," 2016. [Online]. Available: https://www.youtube.com/watch?v=E_JUIIIehDE

[89] "Panofield autozoom," 2015. [Online]. Available: https://www.youtube.com/watch?v=E0GVCdDktZA&t=311s

[90] "Proviosport panorama," 2017. [Online]. Available: https://www.youtube.com/watch?v=dBoYHTWFPmw

[91] Z. Wang, L. Zheng, Y. Liu, and S. Wang, "Towards real-time multi-object tracking."

[92] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," p. 3, 2018.

[93] "MOT15 Results." [Online]. Available: https://motchallenge.net/results/MOT15/?det=Private

[94] M. Brostrom, "Yolov5 deepsort repo." [Online]. Available: https://github.com/mikel-brostrom/Yolov5_DeepSort_Pytorch

[95] G. Jocher, "Yolov5 newest release and weights," 2021. [Online]. Available: https://github.com/ultralytics/yolov5/releases/tag/v5.0

[96] "MOT20 Results." [Online]. Available: https://motchallenge.net/results/MOT20/?det=Private

[97] "Overview map." [Online]. Available: https://all-free-download.com/free-vector/download/soccer-field-football-pitch-clip-art_15819.html

[98] "Cielab color space," 2021. [Online]. Available: https://en.wikipedia.org/wiki/CIELAB_color_space

[99] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, vol. 60, no. 2, pp. 91–110, Nov. 2004. [Online]. Available: http://dx.doi.org/10.1023/B:VISI.0000029664.99615.94

[100] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International Conference on Computer Vision*, 2011, pp. 2564–2571.

[101] "Experiment results: Video demos." [Online]. Available: https://drive.google.com/drive/folders/14Jc-xJWcw7NTSN1xlTRai5Jv4-LAYdbt?usp=sharing

[102] P. Dendorfer, H. Rezatofighi, A. Milan, J. Shi, D. Cremers, I. Reid, S. Roth, K. Schindler, and L. Leal-Taixé, "MOT20: A benchmark for multi object tracking in crowded scenes," *arXiv:2003.09003[cs]*, Mar. 2020, arXiv: 2003.09003. [Online]. Available: http://arxiv.org/abs/1906.04567

[103] Y. Zhang, "ifzhang/fairmot," GitHub, 12 2020. [Online]. Available: https://github.com/ifzhang/FairMOT

[104] A. Bewley, "SORT," Available at https://github.com/abewley/sort.

[105] V. Bushaev, "Stochastic gradient descent with momentum." [Online]. Available: https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d

Appendix

A Architecture Hyperparameters

A.1 Faster R-CNN + SORT

Faster R-CNN and SORT Hyperparameters

Detection architecture	Faster R-CNN
Tracking architecture	SORT
Detection architecture implementation	Detectron2 [67]
Tracking Architecture implementation	[104]
Backbone	ResNet50 with FPN
Weight initialisation	Detectron2's COCO benchmark weights [69]
Optimizer	SGD
Learning rate	10k its. @ 1e-2 + 40k its. @ 1e-3 + 26K @ 1e-4
Momentum	0.9
Gamma	0.1
Weight decay	1e-4
Data Augmentations	RandomFlip
Hardware	1 x Tesla V100 PCIe 32GB
Training time	30 hours
Iterations	76000
Total batch size	6 x 256 + 6 x 512
Frozen layers	All 5 ResNet layers
Network input size	1080 x 1920
Image batch size	6
Image normalisation mean	(103.530, 116.280, 123.675)
Image normalisation STD	(1, 1, 1)
Down sampling ratio (Stride)	4, 8, 16, 32, 64 (FPN)
Backbone output channels	256
RPN batch size per image	256
RPN NMS IoU treshold	0.7
RPN input strides	4, 8, 16, 32, 64
RPN Pre-NMS top-K (train) (per FPN lvl)	2000
RPN Pre-NMS top-K (test) (per FPN lvl)	1000
RPN Post-NMS top-K (train)	2000
RPN Post-NMS top-K (test)	1000
RPN anchor sizes (per feature map)	(32, 64, 128)
RPN anchor scales	(0.5, 1.0, 2.0)
RPN positive IoU tresh	0.7
RPN negative IoU tresh	0.3
RPN posetive examples fraction	0.5
RPN loss function	Smooth L1
RPN loss weight	1.0
Train IoU threshold	0.5
Box head input strides	4, 8, 16, 32
Box head NMS threshold (train)	0.5 Box head NMS threshold (test)
0.7 Box head positive examples fraction	0.25
Confidence threshold	0.5
Box head loss function	Smooth L1
Box head loss weight	1.0
Box head batch size per image	512
FC dimension	1024
RoI pool resolution	7
Number of classes	1
Detections per image	40
Optimizer loss	Sum, weighted
SORT IoU threshold	0.3

Table 19: The complete list of hyperparameters for Faster R-CNN and SORT used in the Experiment from Section 4.2.1.

Table 19 lists all the hyperparameters used for the experiment conducted with Faster R-CNN and SORT (See Section 4.2.1). The parameters correspond to tuneable values that are offered by the Detectron2 library [67]. The vast number of parameters to adjust showcases the flexibility of the library. Indeed, the Faster R-CNN paper [26] did not in fact intend for many of these to be adjustable, but Detectron2 successfully generalized the architecture, making it possible carefully tailor the network to most domains. Below are some notes on the specific parameters and the rationale for their values.

A.2 YOLOv5 + DeepSORT

YOLOv5 and DeepSORT Hyperparameters

Detection architecture	YOLOv5
Tracking architecture	DeepSORT
Detection architecture implementation	[95]
Tracking architecture implementation	[94]
Backbone	Modified CSPNet [55]
Weight initialisation	YOLOv5's COCO benchmark weights [95]
Optimizer	SGD
Momentum	0.937 [105]
Weight decay	0.0005
Learning rate	0.01
Data Augmentations	Scaling, color space adjustments, mosaic augmentation, flipping & translation
Hardware	2 x Tesla V100 PCIe 32GB
Training time	60 hours
Iterations	128000
Network input size	1920 x 1920
Image batch size	4
Down sampling ratio (Stride)	8, 16, 32
Backbone output channels	1024
Class loss	Binary Cross-Entropy With Logits Loss
Object loss	Binary Cross-Entropy With Logits Loss
Box loss	Generalized Intersect over Union (GIoU)
Optimizer loss	Scaled sum of Box, Object & Class Loss
Confidence threshold	0.4
NMS IoU threshold	0.5
Number of classes	1
ID head loss	Cross entropy
Re-ID embedding dim.	512
DeepSORT crop size	64 x 128
DeepSORT max distance	0.2
DeepSORT confidence thresh.	0.3
DeepSORT NMS Thresh	0.5
DeepSORT max IoU distance	0.7
DeepSORT Iterations	31360
DeepSORT batch size	512
DeepSORT training time	12 hours
Max track age	70
Tracking buffer	100

Table 20: The complete list of hyperparameters for YOLOv5 and DeepSORT used in the Experiment from Section 4.4.1. Note that DeepSORT crops from the original image, and not the (possibly) resized network input image.

A.3 FairMOT

FairMOT Hyperparameters

Architecture	FairMOT
Architecture implementation	FairMOT'S official repo [103]
Backbone	DLA34 (See Section 2.12)
Weight initialisation	FairMOT's MOT benchmark weights [103]
Optimizer	Adam
Learning rate	2K its. @ 1e-3 + 20K its. @ 1e-4 + 8K its. @ 1e-5
Data Augmentations	Random left-right flip, affine transform,& hsv color augmentation.
Iterations	30000
Image batch size	6
Hardware	2 x Tesla V100 PCIe 32GB
Training time	33 Hours
Network input size	1056x1920
Down sampling ratio (Stride)	4
Backbone output channels	256
Re-ID embedding dim.	128
Heatmap box loss	Pixel-wise log. loss with focal loss
Offset head loss	L1
Width/Height head loss	L1
Optimizer loss	Mean with learnable weights
ID head loss	Cross entropy
Confidence threshold	0.3
NMS IoU threshold	0.4
Tracking buffer	30
Number of classes	1

Table 21: The complete list of hyperparameters for FairMOT used in the Experiment from Section 4.4.1.

B Additional Results

B.1 Faster R-CNN + SORT

Faster R-CNN + SORT Performance Metrics

Metrics	**Alfheim**		**ISSIA**		**Ranheim**		**Overall**	
Pre/post train	Pre	Post	Pre	Post	Pre	Post	Pre	Post
MOTA ↑	43.5	**84.7**	-12.2	**98.6**	12.7	**89.3**	01.8	**94.6**
MOTP ↑	29.4	20.1	14.6	12.2	27.9	25.4	18.7	15.6
IDF_1 ↑	60.6	87.3	42.4	71.1	46.9	71.6	45.6	74.1
IDP ↑	55.6	87.3	31.3	71.4	41.0	73.7	35.4	74.7
IDR ↑	66.6	87.3	65.4	70.8	54.8	69.6	64.0	73.6
Number of tracks	30	30	24	24	20	20	74	74
ID switches	42	12	60	28	53	35	155	**75**
Mostly tracked ↑	16	22	23	23	8	19	47	64
Partially tracked ↓	8	2	0	0	12	1	20	3
Mostly lost ↓	6	6	1	1	0	0	7	7
Transfers ↓	6	4	2	3	20	19	28	26
Fragmentations ↓	114	28	85	35	115	73	314	**136**
AP ↑	22.8	**51.7**	69.8	**83.1**	23.1	**42.4**	55.2	**72.8**
AP50 ↑	62.8	**90.2**	91.7	98.6	57.7	**91.8**	82.4	96.4
AP75 ↑	09.1	**55.1**	86.6	96.2	12.7	30.2	62.4	82.1
APsmall ↑	00.1	**16.7**	-	-	17.5	40.7	14.0	**39.6**
APmedium ↑	25.1	**52.5**	70.0	83.0	33.8	49.5	59.4	76.5
APlarge ↑	19.1	**50.4**	74.7	84.9	-	-	55.1	72.6
Recall ↑	82.1	0.925	98.4	98.9	73.7	92.3	91.6	96.7
Precision ↑	68.5	0.924	47.1	**99.8**	55.1	**97.6**	50.6	98.1
False positives ↓	2208	445	23452	33	2997	113	28657	**591**
False negatives ↓	1047	439	333	228	1316	387	2696	**1054**

Detection FPS: 12 **Association FPS**: 167 **Total FPS**: 11

Table 22: The full list of validation performance metrics for Faster R-CNN and SORT, before and after optimisation.

Faster R-CNN Good Detection Demo

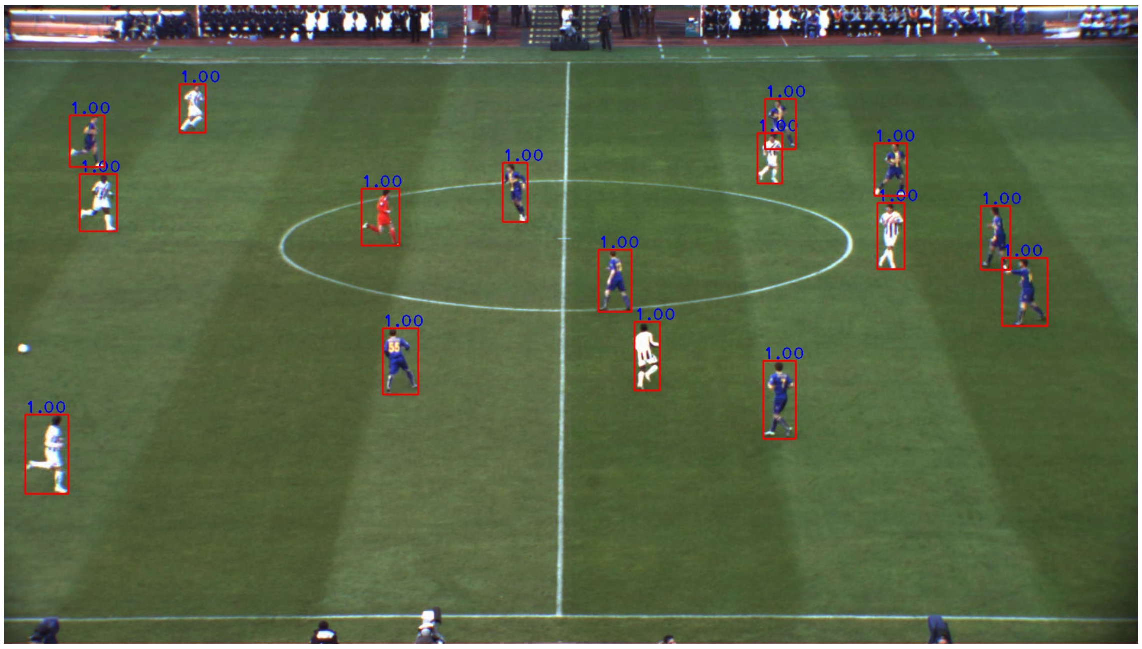


Figure 69: Examples of good detections by the finetuned Faster R-CNN model. Faster R-CNN localizes players with tight bounding boxes when they are not significantly occluded.

Faster R-CNN Bad Detection Demo

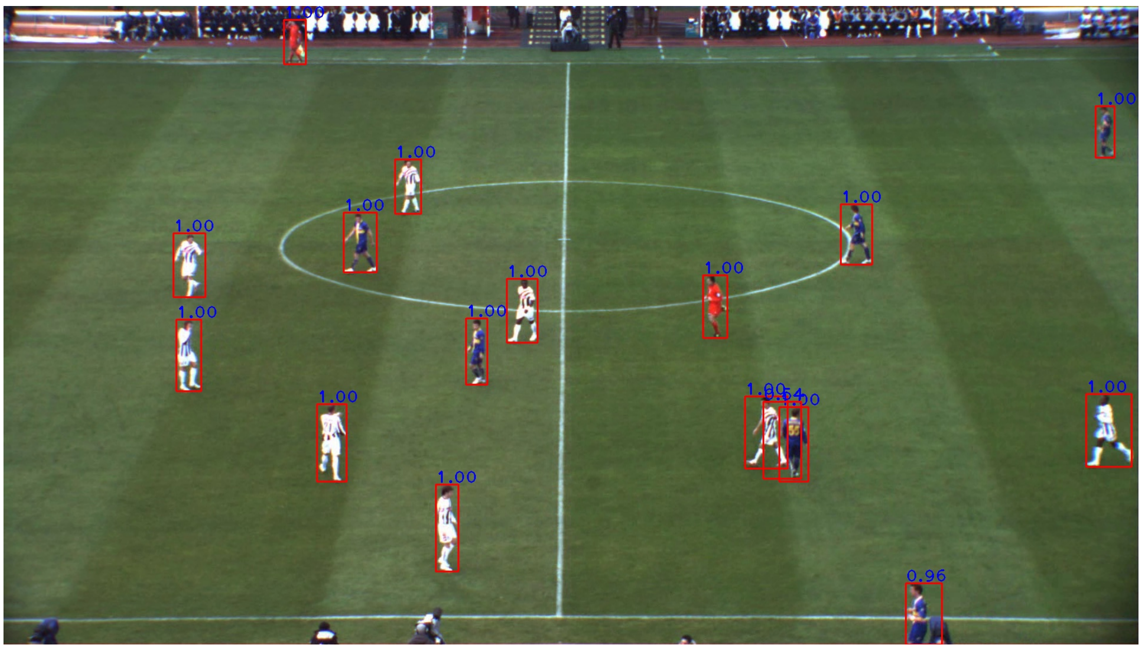
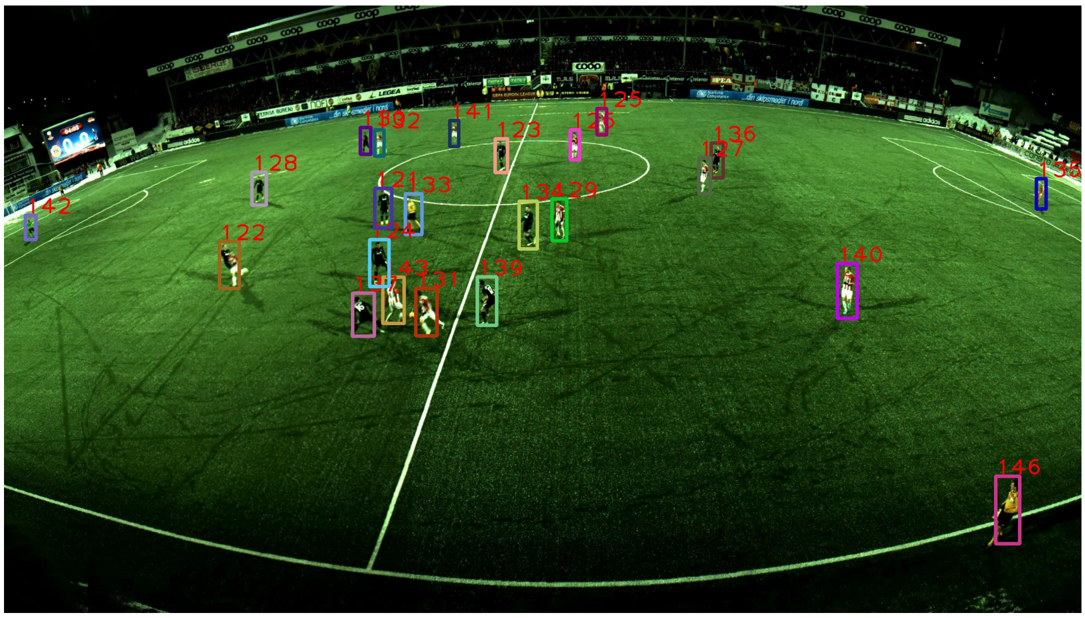


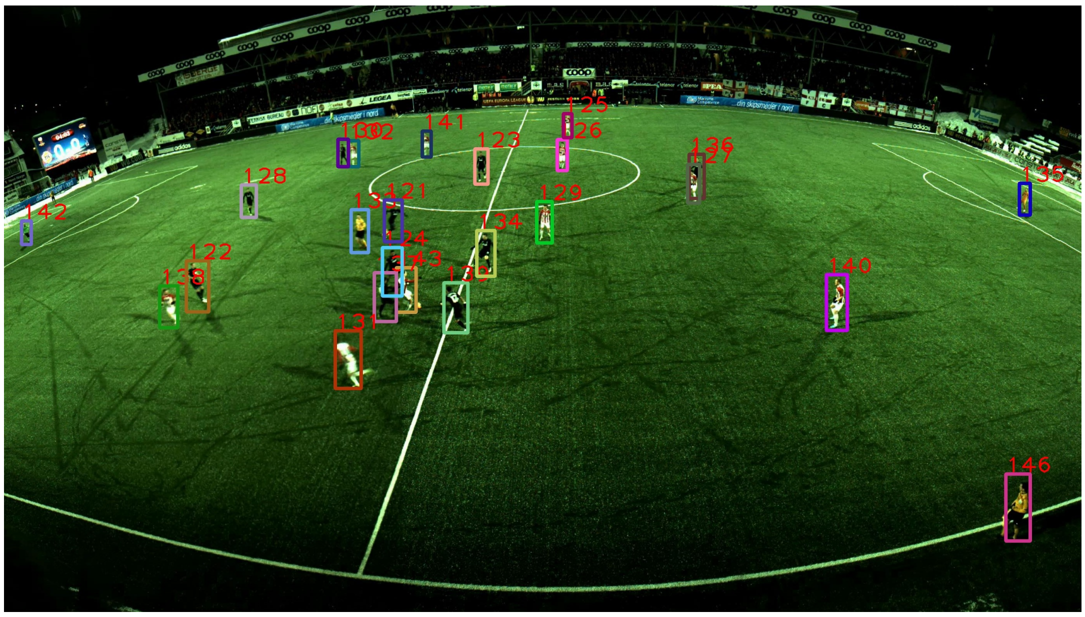
Figure 70: Examples of bad detections by the finetuned Faster R-CNN model. Notice the two players closest to one another confuse the model to output a third wrongly placed bounding box. Although hard to see, that third box has lower confidence than the neighboring ones.

Faster R-CNN + SORT Tracking Demo 2



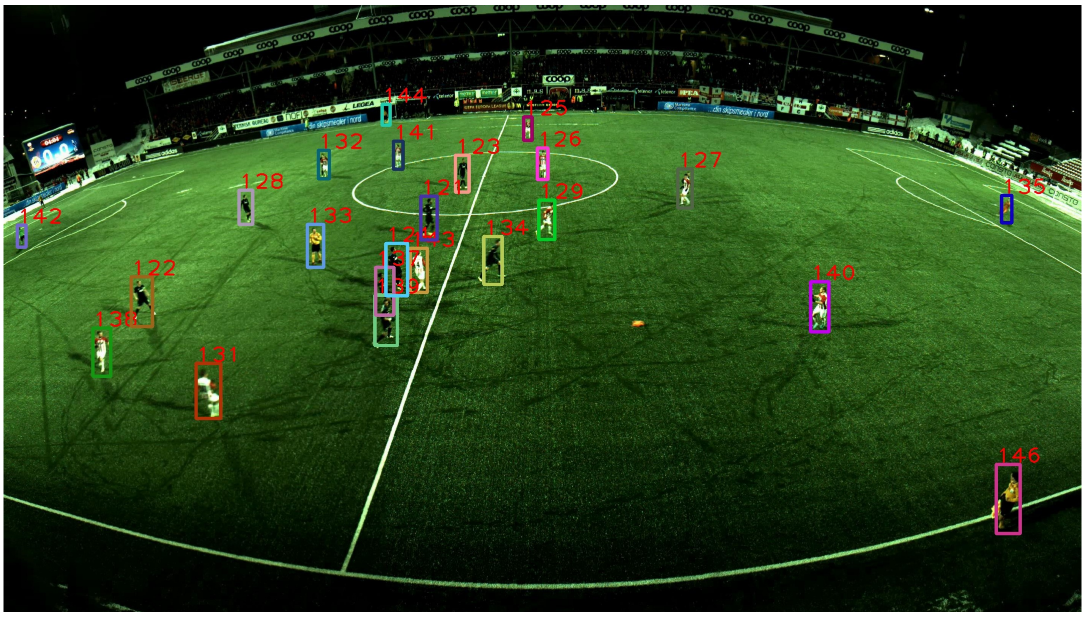
t=1

In The first frame, all objects are detected by Faster R-CNN, and assigned IDs by SORT

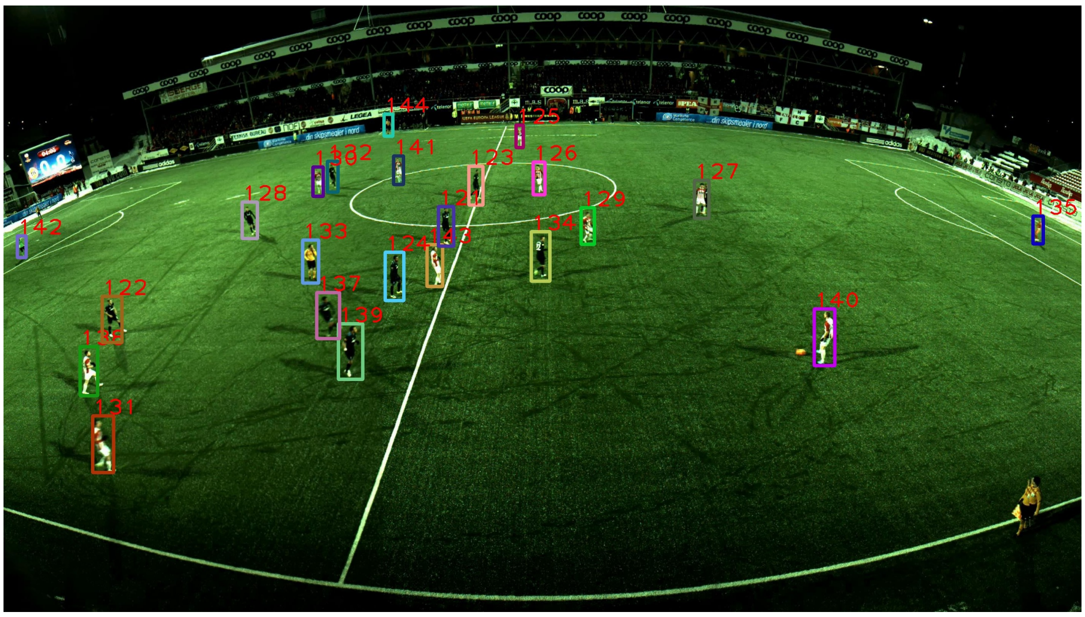


t=2

Track IDs are successfully ”carried” over between frames. Note the player with ID 138 was previously occluded, and is now being tracked.


t=3

Note that the tight cluster of players in the middle are still being correctly tracked.


t=4

SORT is able to consistently maintain tracks across the sequence shown.

Figure 72: Faster R-CNN and SORT tracking demo on the Alfheim validation sequence. The whole demo can be viewed `here`, and here [101].

Figure 72 together with Table 22's MOT metrics for the Alfheim sequence, demonstrates that tracking is perfectly realistic on unseen footage from the Alfheim dataset. In the time frame shown, there are no ID switches even though there is challenging cluster of players in the middle of the field. In fact, Table 22 reveals that there are only a total of 12 ID switches in the entire validation sequence.

B.2 YOLOV5 + DeepSORT

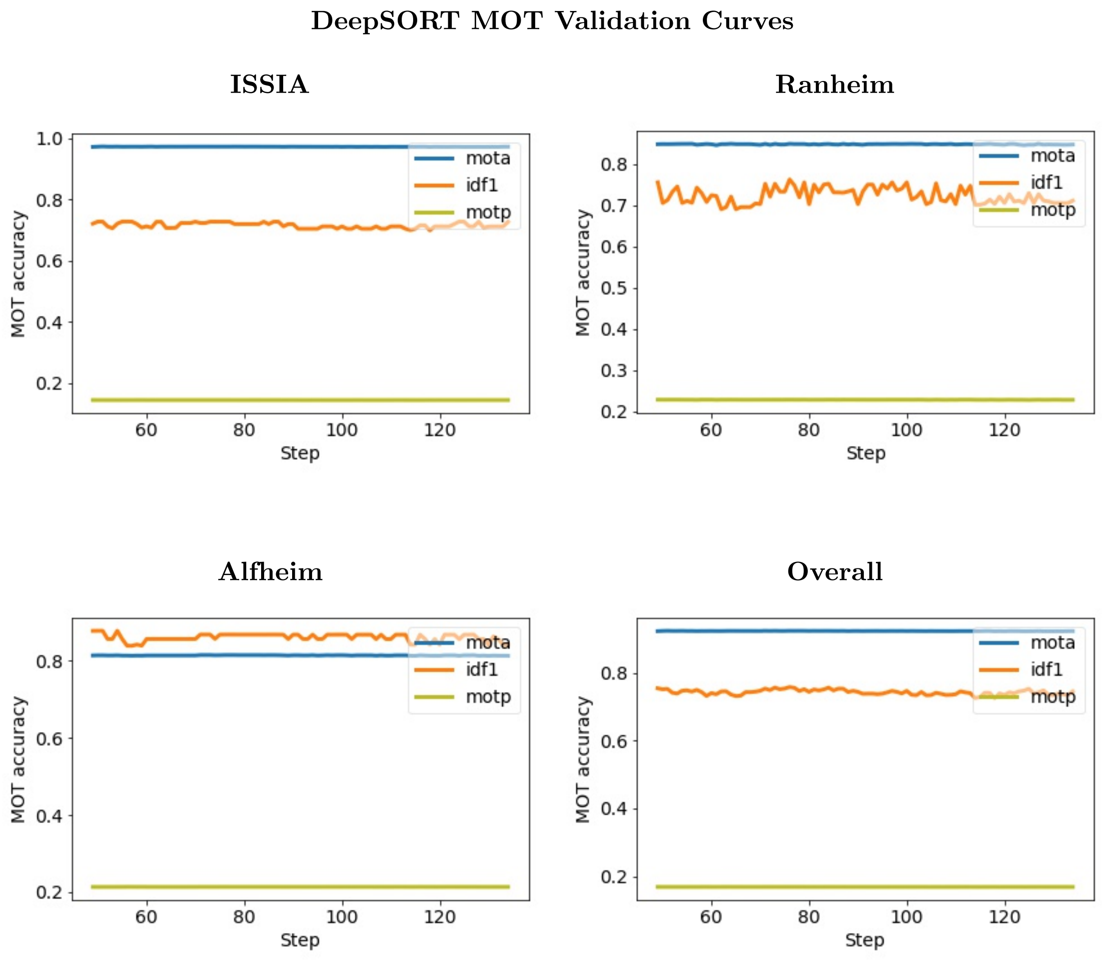


Figure 73: MOT accuracy metrics obtained during training of DeepSORT by regularly evaluating on the validation set. Note that DeepSORT is trained independently of the already optimized YOLOv5 model. During MOT evaluation, it uses the optimized YOLOv5 models outputs as inputs. Refer to Section 3.5 for a description of the complete procedure.

YOLOv5 + DeepSORT Performance Metrics

Metrics	**Alfheim**		**ISSIA**		**Ranheim**		**Overall**	
Pre/post train	Pre	Post	Pre	Post	Pre	Post	Pre	Post
MOTA ↑	66.7	80.7	92.7	97.1	58.3	**91.2**	82.7	**93.2**
MOTP ↑	27.7	21.0	16.1	14.1	27.4	23.4	19.2	16.7
IDF_1 ↑	73.8	88.4	68.9	72.7	61.8	77.6	68.9	**76.4**
IDP ↑	78.5	87.0	68.7	72.8	81.5	80.8	71.7	76.6
IDR ↑	69.6	89.9	69.1	72.6	49.8	74.7	66.2	76.1
Number of tracks	30	30	24	24	20	20	74	74
ID switches. ↓	21	6	26	26	23	26	70	58
Mostly tracked ↑	15	22	23	23	7	19	45	64
Partially tracked ↓	7	2	1	0	11	1	19	3
Mostly lost ↓	8	6	0	1	2	0	10	7
Transfers ↓	7	3	6	5	21	25	34	33
Fragmentations ↓	97	37	137	60	127	64	361	**161**
AP ↑	25.3	**49.9**	75.3	85.5	21.0	**47.6**	62.3	**74.9**
AP50 ↑	67.7	88.1	94.7	98.8	48.5	92.8	87.5	96.5
AP75 ↑	9.9	53.5	90.9	96.6	10.4	37.3	69.83	83.3
APsmall ↑	0.	27.9	-	na .	6.3	47.0	6.1	**46.2**
APmedium ↑	27.1	59.0	75.1	88.9	36.6	61.1	65.7	81.4
APlarge ↑	22.7	43.7	79.8	86.1	-	a .	61.9	70.2
Recall ↑	77.9	92.1	96.7	98.5	59.9	92.1	87.6	96.3
Precision ↑	87.9	89.1	96.2	98.7	98.1	99.6	94.9	97.0
False positives ↓	626	657	819	**266**	58	20	1503	943
False negatives ↓	1288	464	697	322	2005	393	3990	**1179**

Detection FPS: 25 **Association FPS**: 13 **Total FPS**: 9

Table 23: The full list of validation performance metrics for YOLOv5l6 and DeepSORT, before and after optimisation.

YOLOv5x + DeepSORT Performance Metrics

Metrics	Alfheim		ISSIA		Ranheim		Overall	
Pre/post train	Pre	Post	Pre	Post	Pre	Post	Pre	Post
MOTA ↑	61.8	79.8	90.6	97.3	50.2	94.1	79.0	**93.6**
MOTP ↑	27.7	20.6	16.3	14.1	28.4	22.4	19.3	16.5
IDF_1 ↑	69.4	85.5	63.3	73.4	57.1	83.3	63.6	**77.2**
IDP ↑	76.8	84.0	64.8	73.6	80.6	85.5	68.4	77.3
IDR ↑	63.3	87.0	61.9	73.3	44.2	81.2	59.4	77.0
Number of tracks	30	30	24	24	20	20	74	74
ID switches ↓	28	8	42	31	23	22	93	61
Mostly tracked ↑	12	22	23	23	6	20	41	65
Partially tracked ↓	9	1	1	0	10	0	20	1
Mostly lost ↓	9	7	0	1	4	0	13	8
Transfers ↓	12	6	16	11	23	19	51	36
Fragmentations ↓	118	25	167	55	103	41	388	121
AP ↑	23.7	49.8	70.4	85.6	20.5	51.3	58.9	**75.3**
AP50 ↑	63.2	87.0	90.4	98.7	47.3	95.7	82.7	96.4
AP75 ↑	10.7	53.0	86.6	96.4	8.6	44.8	67.4	83.7
APsmall ↑	0	6.4	-	-	17.7	48.9	17.2	**48.1**
APmedium ↑	25.8	59.3	69.9	88.9	34.2	61.9	61.7	81.6
APlarge ↑	21.3	46.1	78.2	87.6	-	a .	60.1	73.1
Recall ↑	72.4	91.8	93.1	98.6	52.7	94.8	83.0	96.7
Precision ↑	87.8	89.0	97.5	98.9	96.2	99.7	95.7	97.1
False positives ↓	588	693	502	233	103	13	1193	939
False negatives ↓	1614	477	1466	308	2365	261	5445	1047

Detection FPS: 15 **Association FPS**: 13 **Total FPS**: 7

Table 24: Performance metrics for YOLOv5x + DeepSORT. YOLOv5x was tested in 3.3.2, when investigating how the different YOLOv5 architectures would perform on the soccer domain.

YOLOv5 Good Detection Demo

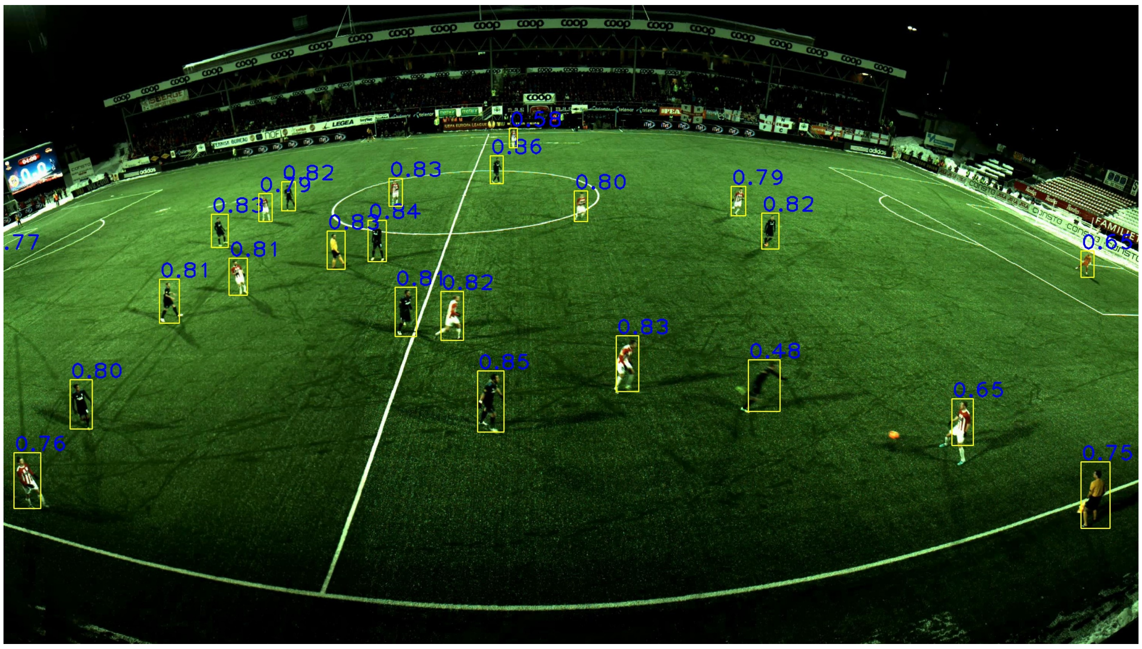


Figure 74: Examples of good detections by the finetuned YOLOv5 model. All players are localized with high confidence scores on an image from the Alfheim validation sequence.

YOLOv5 Bad Detection Demo

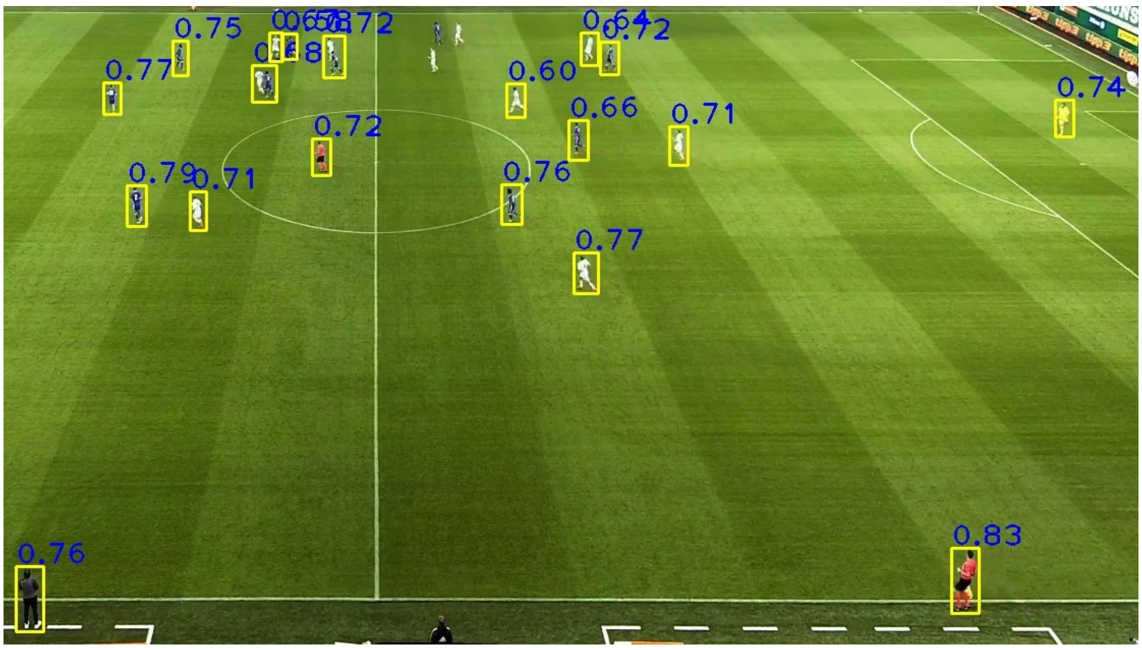
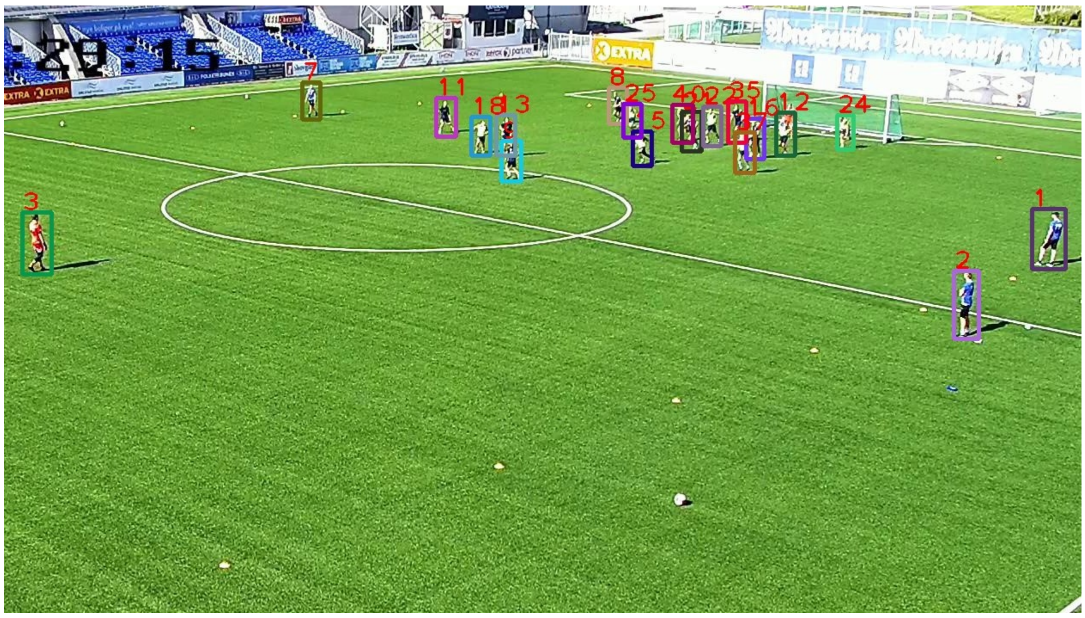
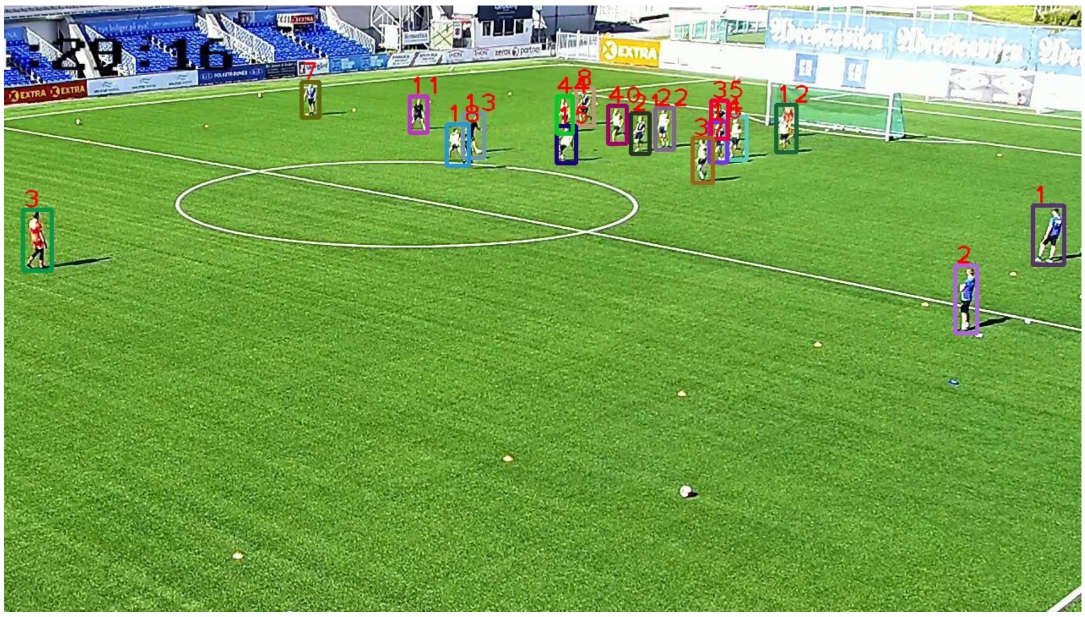


Figure 75: Examples of bad detections by the finetuned YOLOv5 model. The input image is from the additional unlabeled test set. On the far end of the field, there are three missed detections. Also, two players are incorrectly mistaken to be one object.

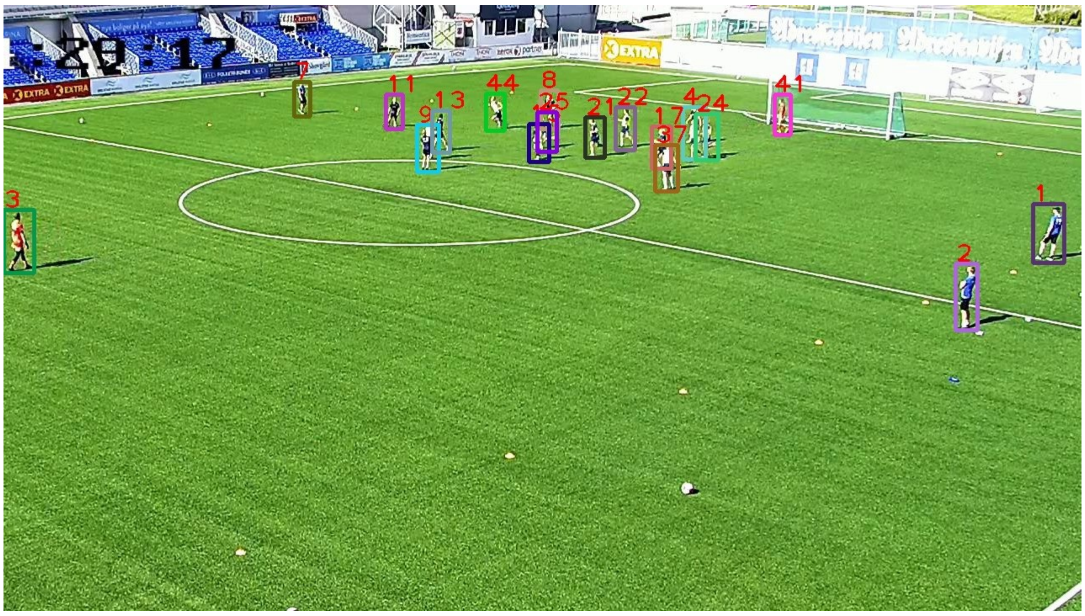
YOLOv5 + DeepSORT Tracking Demo 2

t=1 

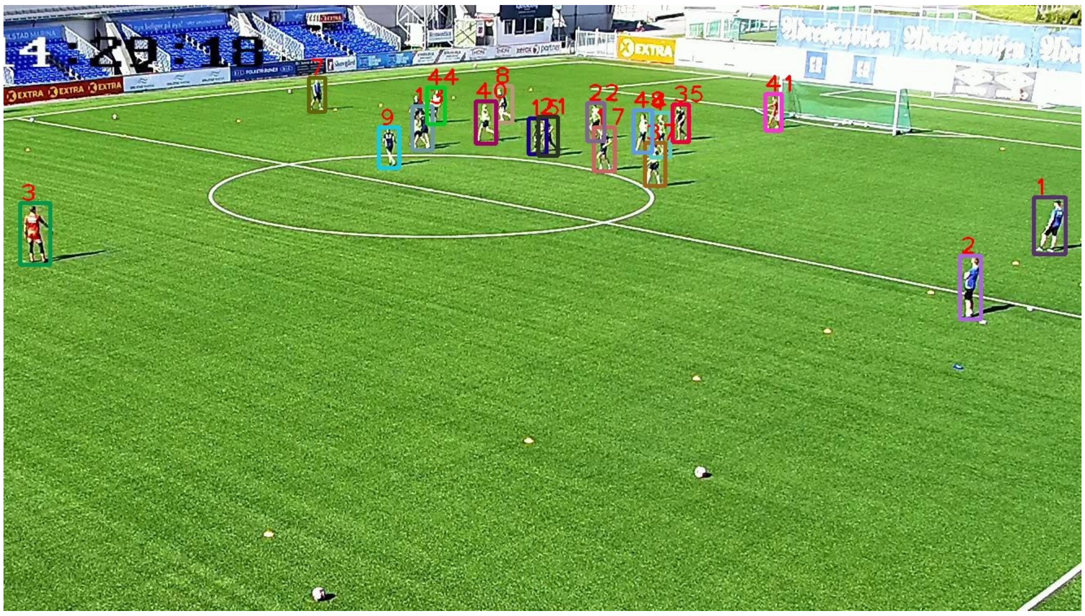
All players are detected and localized by YOLOv5 in the first frame.

t=2 

Most players IDs are retained a second later. Player 9 (light blue), 24 (light green), and 25 (purple) visible in t=1 is now in occlusion. Player 44 (green) has emerged from occlusion.

t=3 

Player 9, 24 and 25 is re-identified after occlusion. Player 41 (pink) has emerged from occlusion.

t=4 

Player 11, 15, 43, and 25 is in occlusion. Player 35 (red) has been re-identifed after occlusion.

Figure 77: YOLOv5 + DeepSORT tracking demo on the Ranheim validation sequence. The sequence is very similar to data the model has trained on. The tracking is fragmented due to missed detections. A video demo can be viewed `here`, and here [101].

Figure 77 depicts an example where YOLOv5 handles detecting a tight cluster of players well, demonstrating that it can handle crowded situations even though it may not be one of its strengths. We can also observe DeepSORT re-identifying almost entirely occluded players. Although DeepSORT's deep feature functionality appears to struggle in the football domain, it is still a highly functioning tracker, as we can observe from the example above.

B.3 FairMOT

FairMOT Performance Metrics

Metrics	**Alfheim**		**ISSIA**		**Ranheim**		**Overall**	
Pre/post train	Pre	Post	Pre	Post	Pre	Post	Pre	Post
MOTA ↑	18.6	**83.0**	-8.7	**97.1**	65.4	79.2	7.8	**91.8**
MOTP ↑	26.7	0.231	22.3	18.7	28.9	24.9	23.5	20.3
IDF$_1$ ↑	37.7	**89.7**	47.4	73.2	62.5	78.8	48.0	**77.0**
IDP ↑	68.7	0.902	35.4	**73.0**	71.7	85.6	39.7	**77.8**
IDR ↑	26.0	**89.1**	71.6	73.5	55.4	73.0	60.8	76.2
Number of tracks	30	30	24	24	20	20	74	74
ID switches. ↓	29	12	23	24	39	71	91	107
Mostly tracked ↑	1	**21**	24	24	7	13	32	58
Partially tracked ↓	11	3	0	0	12	7	23	10
Mostly lost ↓	18	**6**	0	0	1	0	19	6
Transfers ↓	13	**4**	1	2	19	17	33	23
Fragmentations ↓	213	64	225	58	295	206	733	**733**
AP ↑	08.7	**42.7**	34.2	58.8	22.2	36.0	31.3	**58.9**
AP50 ↑	17.4	**85.5**	63.7	**97.4**	57.5	**74.0**	60.1	**92.9**
AP75 ↑	8.0	**37.5**	32.4	66.7	7.8	**29.1**	27.9	57.7
APsmall ↑	0.	10.2	-	-	20.0	41.4	0.	**40.7**
APmedium ↑	8.9	**53.4**	34.5	66.7	27.2	55.2	32.4	63.5
APlarge ↑	19.8	**57.7**	39.3	64.6	-	-	33.3	**61.8**
Recall ↑	28.5	**91.0**	96.9	99.0	71.7	82.9	80.5	95.0
Precision ↑	75.3	92.1	47.9	**98.3**	92.8	97.3	52.7	**97.0**
False positives ↓	545	453	22410	**366**	279	115	23234	**934**
False negatives ↓	4177	**527**	662	219	1413	855	6252	**1601**

Total FPS: 12

Table 25: Validation performance metrics for FairMOT, before and after training.

FairMOT Bad Detection Demo

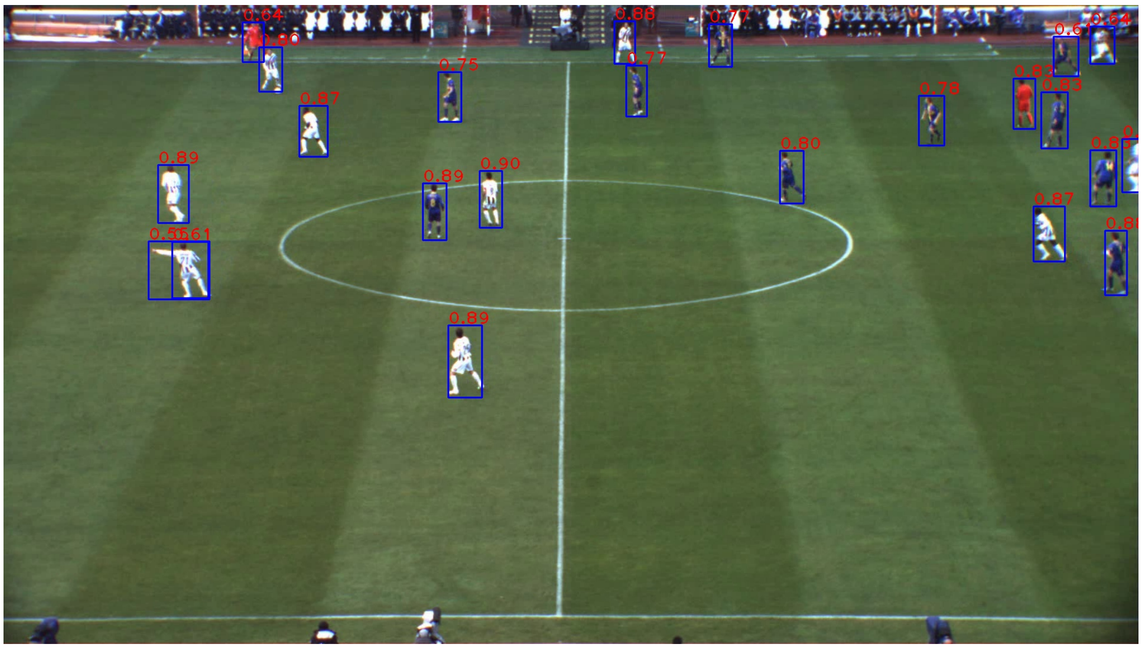


Figure 78: Examples of bad detections by the finetuned FairMOT model. The leftmost player is incorrectly labeled as two different objects. The smaller (incorrect) box has a large overlap with the correct one.

FairMOT Good Detection Demo

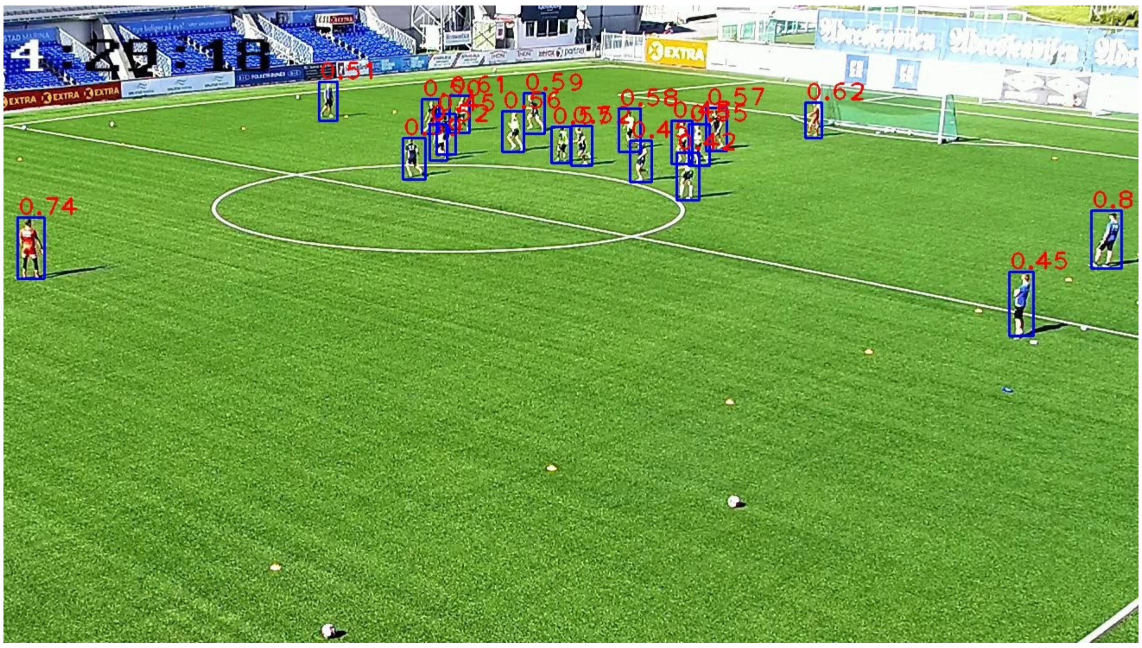
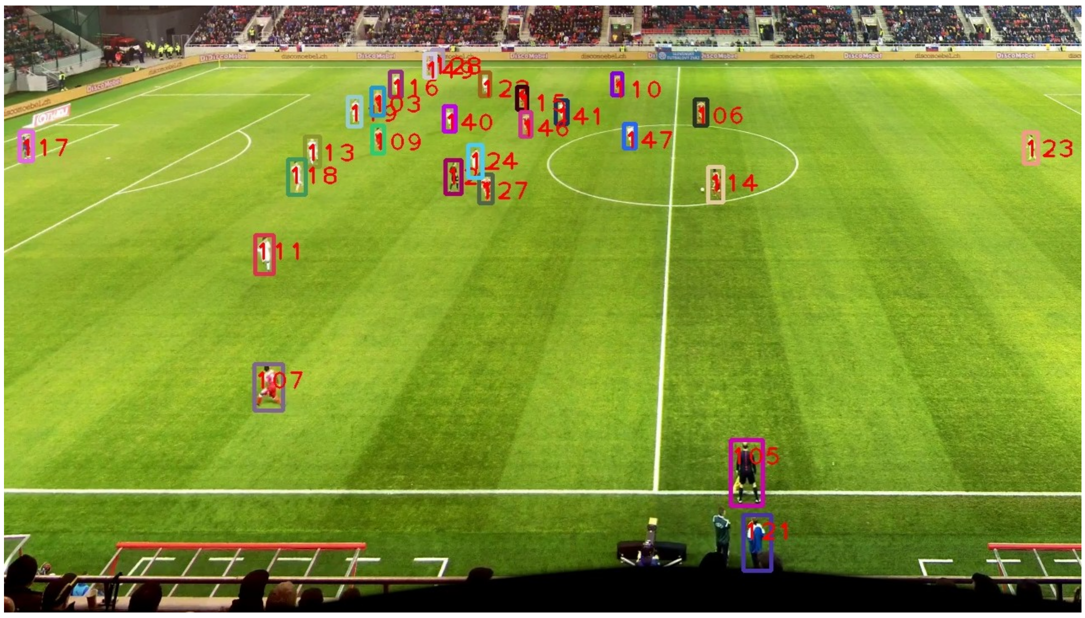
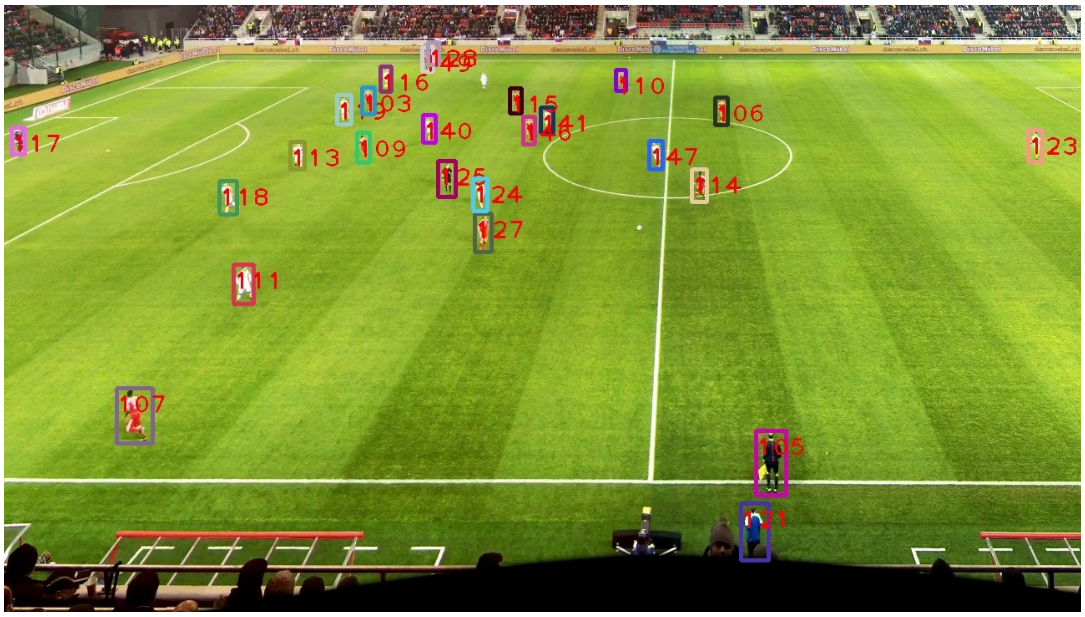


Figure 79: Examples of good detections by the finetuned FairMOT model. Although hard to see due to the many overlapping bounding boxes, all objects are correctly detected.

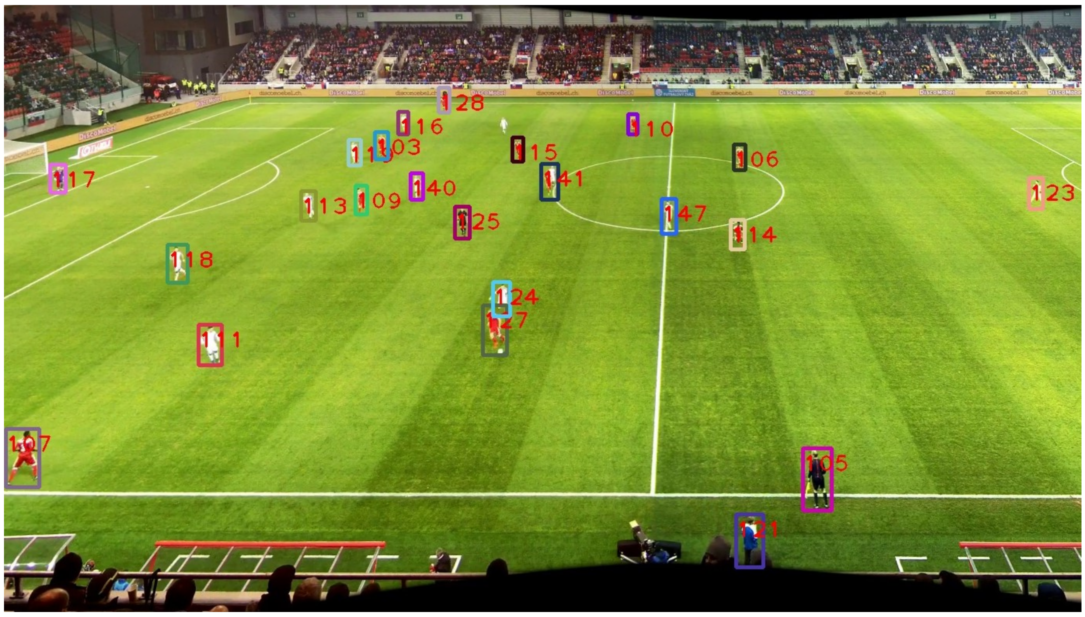
FairMOT Tracking Demo 2

t=1 

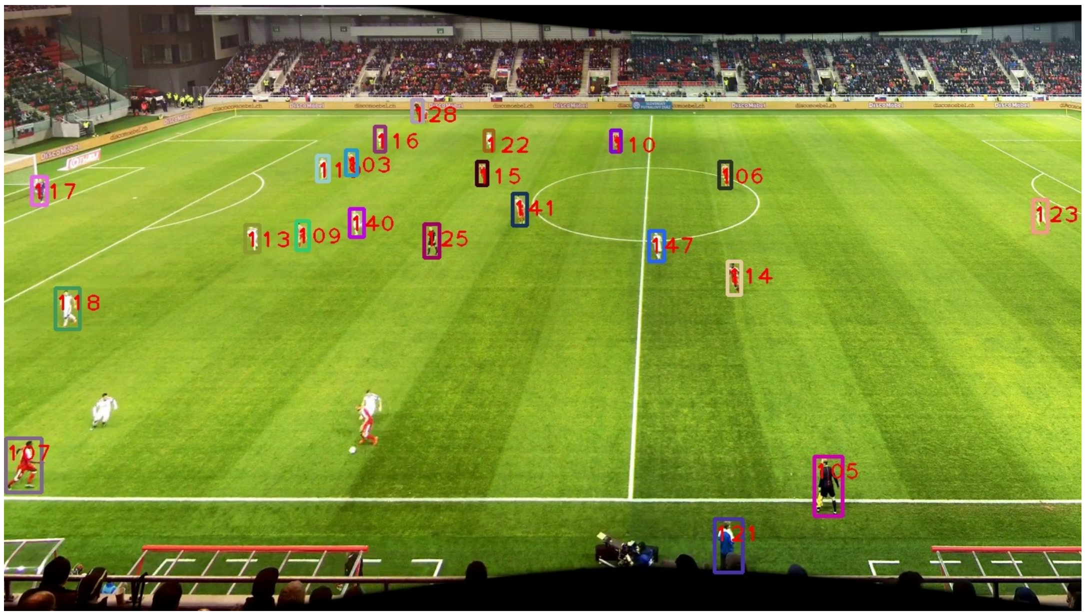
All players are correctly identified in the first frame.

t=2 

About a second later, a single object was failed to be identified.

t=3 

Player 122 is still not detected. Also, player 146 is not detected.

t=4 

Three players are not detected. Player 122 was re-identified upon detection.

Figure 81: FairMOT tracking demo on one of the additional test sequences, described in Section 3.2. Objects are small compared to the ones from the validation sequences. The proposal nearest the camera is a false positive, and is tracked during the entire sequence shown. A video demo can be viewed `here`, and here [101].

In Figure 81, the second tracking demo is shown. There, FairMOT demonstrates is capabilities to generalize beyond the dataset that were annotated in this thesis. While not perfect, most objects are tracked, which is impressive considering the very small sizes of the players. There are occasional detection misses, which confirms the number of false negatives and fragmentation in Table 25. However, they mostly introduce fragmentations, and not as much ID switches. The latter is arguably more severe in the domain of soccer.

Scott Gullaksen & Eirik Lie Morken

Automatic Player Tracking in Single-Camera Soccer Videos

NTNU

Kunnskap for en bedre verden