Ball Tracking in Association Football - Leveraging Player Detections to Locate the Ball Using Graph Attention Networks

Gustav Nirvin

LUND UNIVERSITY

Department of Mathematics

Abstract

When automating data analysis or broadcasts in association football, a key piece of information is the position of the football. Locating the ball is often easier said than done since there are many scenarios where the ball is hard or even impossible to detect accurately using object detection methods. This thesis explores a different approach to determining the ball's position. Since the players on the pitch are much easier to detect than the football, those detections are leveraged to predict the position of the ball. By using a Graph Neural Network (GNN) architecture with attention layers the thesis explores with what accuracy it is possible to predict the ball's position in different scenarios of a game of football, as well as what features are most important for making a good prediction. The results show that it is possible to predict the ball's position with a total mean prediction error of 13.0 meters, with kick-offs having the lowest mean error, and penalties the highest. Besides the positions of the players, their velocities are important pieces of information that help make a good prediction in many, but not all, scenarios of a game. Furthermore, adding too much history of the players' positions and velocities leads to worse predictions but further investigation is necessary to determine if the model architecture is at fault or if the data has no useful underlying structure. In conclusion, GNNs show promise for predicting the ball's position since they improve significantly compared to a baseline, especially if put in place to operate in tandem with an object detection network. More features, such as what team each player is on and their body pose, could possibly improve the network's performance but were not explored in this project.

Keywords: machine learning, graph neural network, graph attention network, association football, ball tracking

Populärvetenskaplig sammanfattning

Går det att prediktera en fotbolls position utifrån var spelarna befinner sig på planen? Detta arbete utreder om det är möjligt att använda information om fotbollsspelares positioner och rörelser för att avgöra var bollen befinner sig i olika skeenden av en fotbollsmatch.

Med framfarten av artificiell intelligens och maskininlärning har antalet användningsområden skjutit i höjden. Inom fotboll så används tekniken idag bland annat för att automatisera dataanalys av matcher och träningar samt för att livesända matcher, utan behov av en kameraman. För båda dessa användningsområden är det absolut nödvändigt att veta var fotbollen befinner sig. På grund av bollens storlek och det faktum att den ofta rör sig väldigt snabbt så har vanliga metoder av objektdetektering ofta svårt att följa bollen med precision. Eftersom spelarna på planen är betydligt lättare att detektera ämnar detta arbete att utreda med vilken precision det går att använda dessa detektioner för att avgöra bollens nuvarande position. Därtill så tittas det närmre på olika scenarion som uppstår i en fotbollsmatch, så som straff och frispark, och hur prediktionens precision förändras i dessa olika scenarion.

En gren inom maskininlärning som blivit populär de senaste åren är grafnätverk. Varje spelare på planen representeras som en nod i en graf med attribut som motsvarar deras position och hastighet. Mellan varje nod finns en kant som representerar hur spelarna påverkas av och förhåller sig till varandra. Förhoppningen är att datormodellen ska lära sig vilka noder (spelare) och vilka kanter som är viktiga för att bestämma bollens position och därefter kunna göra en prediktion som är nära bollens faktiska position. Resultaten visar på att det är möjligt att bestämma bollens position med ett snittfel på 13 meter. Att inte bara veta spelarnas nuvarande positioner, utan också hur de rört sig precis innan är avgörande för att modellen ska göra en bra prediktion. Det visar sig däremot att för mycket historisk information inte behöver vara fördelaktigt utan förvirrar istället modellen. Arbetet visar på potential för användning av grafnätverk. Kan en modell användas tillsammans med ett detektionsnätverk så finns möjligheten att skapa en robust arkitektur för bollföljning inom fotboll. Framtida arbete inom detta område bör utreda om fler attribut kan användas för att minska snittfelet, till exempel spelarnas lagtillhörigheter och hur deras kroppar är orienterade.

Acknowledgements

I want to thank Spiideo AB for not just giving me the chance to write my thesis in collaboration with them, but also for providing me with excellent conditions to do well with my thesis.
I want to thank Håkan and Daniel specifically for the support you have offered throughout, guiding me through every step of this thesis.
I also want to thank Ivar and Ludvig for your supervision. Your technical insights nudged me in the right direction when necessary and your administrative help allowed me to focus entirely on the project.

The computations were enabled by the Alvis resource at C3SE, Chalmers University of Technology, provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS), partially funded by the Swedish Research Council through grant agreement no. 2022-06725, as well as the Berzelius resource provided by the Knut and Alice Wallenberg Foundation at the National Supercomputer Centre (NSC).

Contents

List of Abbreviations

ANN	**A**rtificial **N**eural **N**etwork
GAT	**G**raph **At**tention Network
GNN	**G**raph **N**eural **N**etwork
MAE	**M**ean **A**bsolute **E**rror
ML	**M**achine **L**earning
MSE	**M**ean **S**quared **E**rror
OD	**O**bject **D**etection
VOD	**V**ideo **O**bject **D**etection

1 Introduction

One of the most popular and fastest advancing technologies in recent years has been Machine Learning (ML) and Artificial Intelligence (AI) and with its advancements, the number of use cases has increased. It is therefore no surprise that ML and data analysis have seeped their way into the multi-billion dollar industry that is professional sports and therefore also association football, or soccer, the world's most popular sport. Today, ML is used in many aspects of the sport.

One example of a use case of ML in football is automated broadcasts. Automating broadcasts reduces the cost of live-streaming games by removing the need for a camera operator, thereby making it accessible to more clubs and supporters. A fixed camera setup may be sufficient in some cases, but to enhance the viewer experience the camera needs to be able to pan and zoom to where the play is currently taking place. To enable the camera to do so a computer needs to be taught what is of interest and for that, ML is perfectly suited.

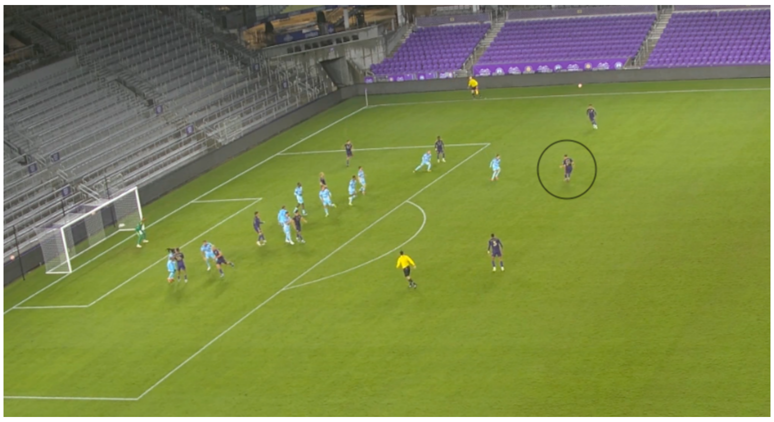
Another use case is automated data analysis. The modern football fan has likely become familiar with terms such as "possession" and "xG" (expected goals), both common metrics for analyzing games and teams. While it is possible to manually use for example a stopwatch to keep track of the amount of time each team has spent in possession of the ball, it is costly, error-prone, and labour-consuming. It is also possible for a football expert to evaluate each shot taken and determine the probability that it finds the back of the net but it is highly inaccurate and once again very costly. Adding to the fact that these are just two of many metrics that are of interest to track, one can easily understand how these processes benefit from being automated. By using ML it is possible to analyze games faster and more accurately than any set of humans could.

For many applications of ML in football, the football itself is integral and its position on the pitch is a key piece of information. Therefore, this thesis will investigate how to automatically determine its position.

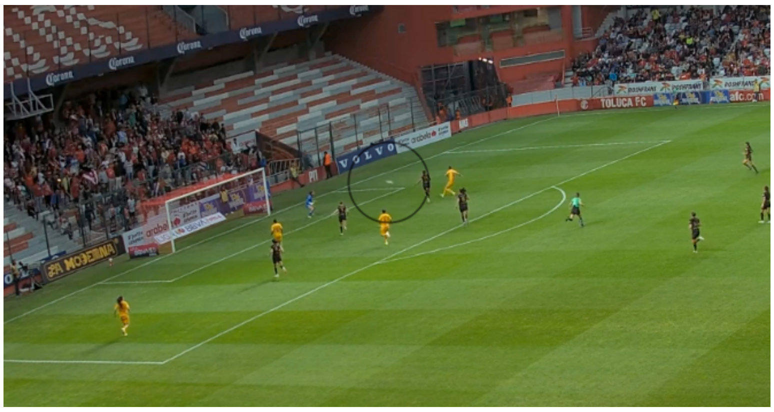
1.1 Problem Statement

It is established that the ball's position is an important piece of information for ML use cases in football and so the question then becomes how is it acquired? Extracting information about the position of an object in an image is known as Object Detection (OD), a task relating to the computer vision branch of machine learning [36]. Methods for completing such a task have been well established and can for simple cases be considered near trivial. Detecting objects in videos is known as Video Object Detection (VOD) and is often a slightly harder task. Difficult cases are when the object appears small to the camera and therefore only makes up a few pixels in the image, when the object is subject to motion blur due to fast movements, when the object is, fully or partially, occluded by another object, or when lighting conditions makes the object harder to identify. In all these cases, many methods of VOD fail and the problem must be approached differently. Two examples can be seen in Figure 1.1.

This is the purpose of this master's thesis: to investigate if it is possible to determine the ball's position on a football pitch with a single fixed camera setup, without the use of VOD. Instead, objects that are easier to detect will be used as contextual information from which the ball's position will be predicted. This mainly refers to the players on the pitch since they are not as often subject to the sce-



(a) An example of the ball being occluded by a player (circled)



(b) An example of the ball being subject to motion blur (circled)

Figure 1.1 Examples of situations where object detection might not work

narios where VOD methods may fail. Players often appear much larger to the camera, they do not move as fast, and are rarely fully occluded and can therefore be detected more reliably than the ball itself. Instead of VOD methods, Graph Attention Networks (GATs) will be used to predict the ball's position. Its suitability for this task will be investigated in this thesis by answering the research questions posed in Section 1.2.

1.2 Research Questions

The questions this thesis aims to answer are:

- With what accuracy is it possible to predict the location of the ball in different scenarios or sequences of a game of football using a GAT architecture? For example corner kicks, shots, or general play.
- What features play an important role in predicting the ball's position when using a GAT architecture? For example players' current positions, players' previous positions, or players' velocities.

1.3 Related Work

As previously mentioned there are a few different scenarios that often pose a problem when trying to detect objects and research has been carried out trying to solve these problems with varying degrees of success. In this section, some of this research is summarised as well as another paper that relates to this thesis.

The task of detecting objects being distorted due to motion blur is probably the problem on which most progress has been made over the last decade. In 2017, a first-of-its-kind dataset was introduced containing fast-moving objects and with it, the first detection method was proposed [22]. After that came proposals such as MANet [28] and FGFA [37] which utilize optical flow derived from a 2015 paper called FlowNet [6] and aggregate features of nearby frames. By doing this, more information than what is available in the current frame is used which makes for a better OD method. More recently TransVOD [12] utilizes both previously developed spatial transformer as well as their own temporal transformer to detect objects in videos.

For handling occlusions in OD, a paper from 2021 reviews the work done in the area [23]. In short, the ability to detect occluded objects has yet to be solved: "detection of occluded objects remains an open problem" but when it comes to handling partial occlusions, there has been some progress. CompNet [15] and CA-CompNet [27] have been proposed recently and the latter proposes a robust method of detecting occluded objects by utilizing the context of the image depending on the degree of occlusion.

Video tiny-object detection is the task of detecting objects that only make up a few pixels of an image. This is often a difficult task and even more so when the background of the image is complex. A paper from 2023 proposes a method in which spatial-temporal motion strength maps are used to locate small objects, specifically UAVs, such as drones, against complex backgrounds [32].

In a paper released in 2024 by Google DeepMind in cooperation with football club Liverpool FC an artificial tactical assistant called TacticAI is proposed [29]. In the paper, a GNN is used to learn different tasks relating to corner kicks. These tasks are predicting the ball recipient, predicting whether or not a shot will be taken as well as giving tactical advice regarding the players' positions, both for the defending and attacking team. The work carried out in this paper has inspired the approach taken in this thesis and will therefore be discussed later in this report.

1.4 Ethical Considerations

Training neural networks requires large amounts of computational power which in turn means that a lot of energy is consumed when working with projects such as this one. This is not without significance since energy consumption is often closely correlated with climate impact and has therefore been taken into consideration during this project.

The data used in this project consists of videos of football matches and therefore also of humans. When dealing with videos of people one should take certain caution and ensure that the collection of such footage is done with consent. The organizations from which the videos are collected have been made aware by Spiideo that the data may be used for research like that carried out in this thesis. Beyond that, it is the organization's responsibility to ensure that anyone who could appear on video, by entering the pitch or the arena at large, has been made aware of this.

1.5 Report Outline

To ensure that the reader will be able to understand the contents of this thesis, a chapter covering the most vital theory behind the thesis will kick off this report. Then follows a breakdown of how the thesis has been carried out and after that the results acquired are recounted. Those results are then discussed in depth and finally conclusions are drawn from that discussion.

2

Theory

This chapter will give an overview of the terms and subjects that are necessary to grasp to understand the chapters that follow. The reader is assumed to have a pre-existing understanding of linear algebra.

2.1 Foundations of Machine Learning

A popular definition of the term *machine learning* goes as follows:

> A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E. [16]

Since this is a very broad definition, it is necessary to explain this further and specifically how it relates to the problem posed in this thesis. The task T as stated in Chapter 1 is the task of predicting where the ball is located on a football pitch. The experience E will henceforth be referred to as *data* and consists of all the information at the disposal of the computer program to solve the task T. The performance measure P is how it is determined what is a good and what is a bad prediction of the ball's position. A natural performance measure could be the Euclidean distance, i.e. the physical distance between the prediction and the ball's actual position, the so-called *ground truth*. The choice of performance measures is discussed further in Section 2.1, Loss Function.

Artificial Neural Network

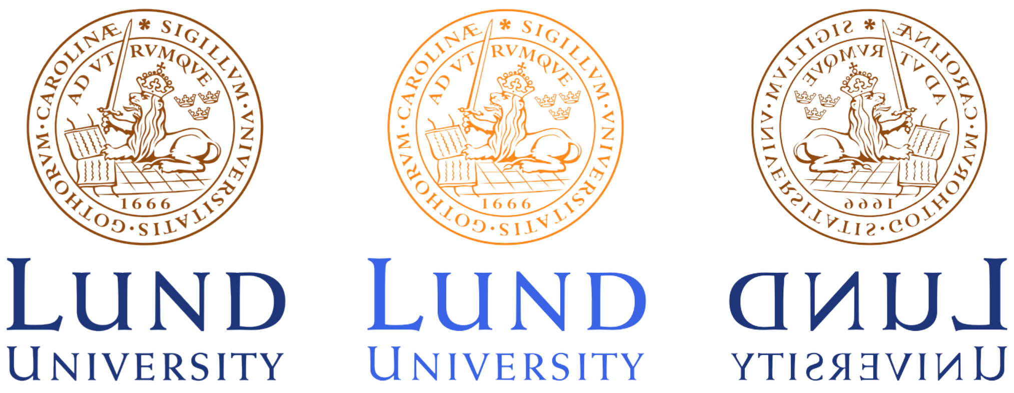
An Artificial Neural Network (ANN) is a common model architecture used for solving machine learning tasks [2]. An ANN consists of *layers* and each layer consists of *nodes*. Depending on how the network is defined, these nodes will be able to perform different mathematical operations. Information is input into the network and passed through the layers (and nodes) via many *edges*, connecting nodes of different layers. All of these edges contain adjustable *weights*, or *parameters*. The final layer of the network is known as the *output layer*. This thesis aims to predict where the football is located on a football pitch and thus the output will be two real numbers, representing the x- and the y-coordinate of the ball respectively.

During training, all of the *training data* available is passed through the network multiple times and after each time the network adjusts the weights on the edges. This is done with the hope that the next time information is passed through the network, the output will be closer to the ground truth, making it a better prediction. How the weights are adjusted may vary but in this project, a method called *back-propagation* is used. This method is discussed further in Section 2.1, Optimization.

Through an iterative process of adjusting edge weights and comparing the network output to the ground truth, the network learns what weights give the smallest loss and thus what makes a good model.

Data

In machine learning, data is the input into the model for it to learn from. Commonly all data that is available is divided into three different sets: *training*, *validation*, and *testing*. The training set is the largest and this is what the model is allowed to see when adjusting its weights. The validation set is used as an initial test to see how well the model performs on data that it has not seen before, i.e. how well it *generalizes*. If the model performs poorly on the validation data, the model architecture is not able to



(a) Original image (b) Increased brightness (c) Vertically flipped

Figure 2.1 Examples of data augmentations to an image

learn the underlying structures of the data and needs to be modified. The test set is used to make a final evaluation of the model and to compare different models and, unlike the validation set, no changes are made to the model depending on the performance on the test set. If the data has been divided with even distributions between the sets and the model generalizes well, there will be small differences between the performance on the validation and the test set. If however outliers or very rare data occur in one of the two sets, the performance between them may differ quite a bit.

Loss Function

The *loss function* is what tells the model if its output is good or bad. The choice of loss function may vary depending on the task. For regression tasks, like the one presented in this thesis, two common loss functions are Mean Absolute Error (MAE) and Mean Squared Error (MSE).

MAE, or L1 loss, is defined as in Equation 2.1 [4] and MSE is defined as in Equation 2.2 [30].

$$MAE = \frac{\sum_{n=1}^{N} |\hat{y}_n - y_n|}{N} \tag{2.1}$$

$$MSE = \frac{\sum_{n=1}^{N} (\hat{y}_n - y_n)^2}{N} \tag{2.2}$$

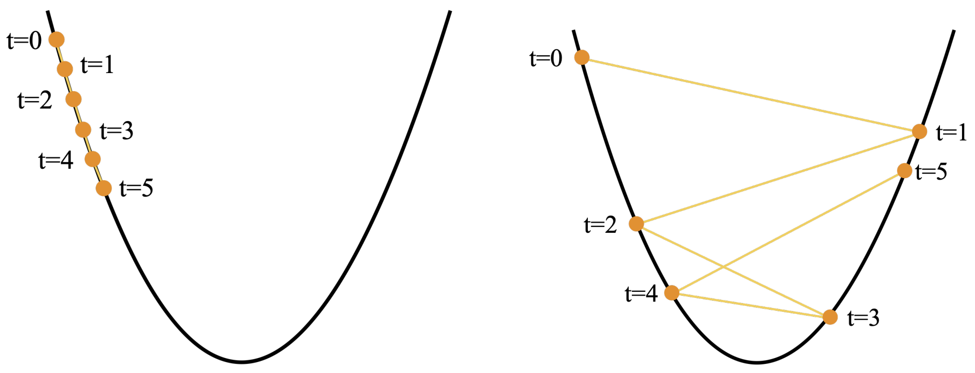
In both equations, y_n is the ground truth for input indexed n, $\hat{y}_n$ is the model output for the same input index and N is the total number of input data. Thus both functions are a way to represent the average model error. The difference between the two is that MSE squares the error and thus penalizes larger differences between the output and the ground truth, more so than MAE.

Data Augmentation and Overfitting

In ML, *data augmentation* is a technique used for expanding the data set [35]. By modifying existing data points the data set can be made significantly larger and more diverse. This in turn is hoped to lead to a better and more robust model as well as to reduce the risk of *overfitting*. Overfitting is when a model is trained "too well" on the training data set but unable to capture the underlying structure of the data and therefore not able to generalize to the validation and test sets. Overfitting can often be recognized when the loss is small for data points from the training set, but not on the validation and test sets. By taking existing data points from the data set and making small changes to them and then putting this modified data point back into the data set, the data set can be made much bigger than it originally was. Some common types of data augmentations are found in ML applications where images make up the data set. Examples of augmentations could be rotating an image slightly, changing the brightness of the image, or mirroring the image, either vertically or horizontally. Two examples can be seen in Figure 2.1.

Dropout

One or more *dropout* layers can be included in a model architecture to reduce the risk of overfitting [13]. Dropout means randomly omitting some nodes during training. This reduces the risk of co-adaptation



(a) A too low learning rate (b) A too high learning rate

Figure 2.2 Examples of problems which can occur when a suitable learning rate is not chosen

where nodes depend on each other since any node could at one point be omitted. The *dropout rate* is the probability that a node in that layer is omitted.

Optimization

When training a model, the parameters need to be updated with respect to the loss. This is known as optimization: adjusting model parameters to optimize the model output and therefore also minimize the loss. Most common methods of optimizing the model parameters use *backpropagation*. Backpropagation refers to moving backward through the model layers, from the output to the input layer, and calculating the gradient of the loss function with respect to the model parameters. By moving through the model architecture backward the number of necessary calculations is heavily reduced, making for a much faster optimization algorithm than the naive approach of forward propagation [10].

One way of utilizing backpropagation in optimization is Adam [14]. Adam (**Ada**ptive **m**oment estimation) is a stochastic gradient-based optimization method that has been shown to outperform other algorithms, such as Stochastic Gradient Descent (SGD) and AdaGrad (**Ada**ptive **Grad**ient), in terms of rate of convergence [14]. The method requires only first-order gradients and computes adaptive learning rates for different parameters through estimates of their first and second-order moments.

Learning Rate Parameters that are chosen beside the learning of a network are known as *hyperparameters*; they are not learned by the model in training but instead predetermined and affect the learning of the model. The stepsize or *learning rate* α is particularly important and should be chosen carefully. If the learning rate is too small the solution that minimizes the loss may not be found within a reasonable number of iterations. If it is too large however, there is a risk of over-shooting and missing the minimum. These potential problems are illustrated in Figure 2.2 where five steps are being taken with both a too low and a too high learning rate.

The learning rate is commonly applied together with a learning rate scheduler to allow for a larger learning rate early in the training, to then decrease the size of the learning rate as the model goes through more and more iterations [21]. The idea behind this is to allow for large changes in the parameter weights when the model is unsure of what they should be and to then make smaller changes when it should be getting more confident in the optimal parameter values.

Object Detection

Object Detection (OD) is the task of detecting objects in images and Video Object Detection (VOD) is the corresponding task for videos [36]. A well-known application is in self-driving vehicles which need to be able to detect for example humans, cars, and stop lights. Object detection is often a combination of two types of ML tasks, *classification* and *regression*. The classification part assigns a label to each detection, such as "car", "human" or "dog". The regression part relates to the position of the detection which often takes one of two shapes: as a single point in the image or more commonly as a *bounding box*. A bounding box is made up of the coordinates of one of the corners of a rectangle along with a height and width. The goal of the bounding box is to completely contain the detected object but to also

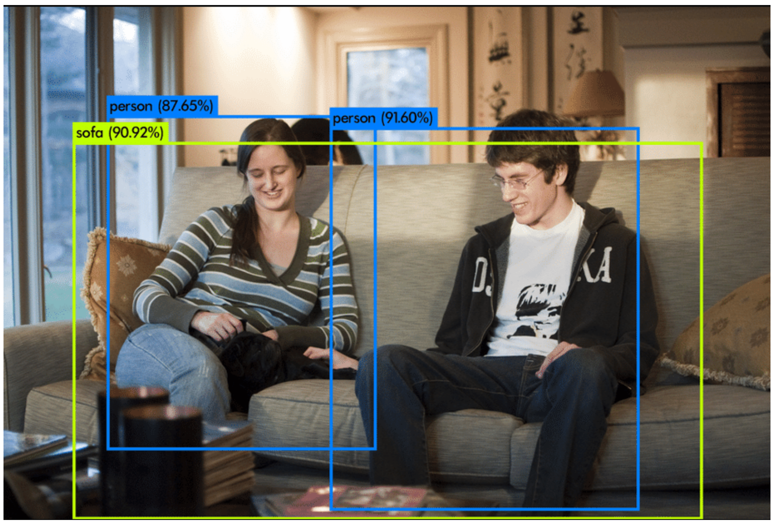


Figure 2.3 An example of bounding boxes along with classes and confidence levels [19]

be as small as possible to not contain any unnecessary parts of the image. Examples of bounding boxes can be seen in Figure 2.3.

One-hot Encoding

One-hot encoding is an encoding method often used to encode non-numerical values as numbers [34]. It is commonly used when classes, such as "person" or "sofa" in Figure 2.3, are used either as input or output of a model. When working with many classes, one might jump to the conclusion that assigning each class a number would solve this problem. However, this would lead to problems when implementing a loss function. If the classes "sofa", "person", and "dog" are assigned numbers 1, 2, and 3 respectively. If the model outputs a 1 (sofa) both when the ground truth is 2 (person) and 3 (dog), since 1 is closer to 2 than 3, the model will think that its prediction was better in the case of the ground truth being a person than when it is a dog. However, it is hard to define a measure of "closeness" when dealing with classes, it is either right or wrong.

Instead, a one-hot encoding means creating a vector consisting of only zeros and a 1 in only one position. The position of the 1 is what distinguishes classes from each other. In the example above, a one-hot encoding could look as follows:

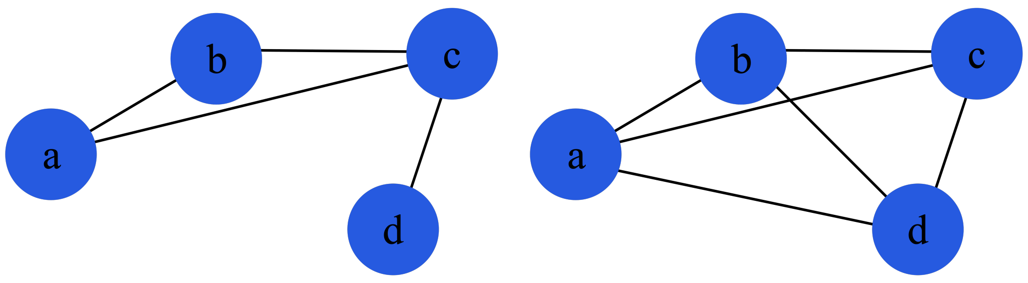
Sofa: $[1,0,0]$
Person: $[0,1,0]$
Dog: $[0,0,1]$

The problem with the relations of numbers explained earlier is now removed since the model does not make any difference between how far apart the ones are in a vector.

2.2 Graphs

A *graph* is a mathematical structure commonly used when modeling objects consisting of points and connections [11]. Formally a graph, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, consists of *nodes*, $\mathcal{V} = \{1, ..., n\}$, and *edges*, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$, where $(i, j) \in \mathcal{E}$ denotes the edge between node i and node j. A *neighborhood* of a node i is the sub-graph consisting of node i itself, all nodes connected to node i by an edge, and all edges connecting those nodes. An edge may be *directed* or *undirected* and edges may have attributes known as *weights* associated with them [11].

For example, people on Facebook may be represented as nodes and friendships between people as edges. Since person A being friends with person B is the same thing as person B being friends with person A, this would be an undirected edge. An edge may have the attribute of the length of this friendship. A graph where all edges are undirected is an undirected graph. Another example is a network of



(a) Incomplete but connected graph (b) Complete graph

Figure 2.4 Examples of graphs

roads and intersections in a country which can be used for route planning in GPSs. Each intersection is represented as a node and each road as an edge. In this case, it would be modeled as a directed graph since one-way roads may occur and an edge could carry the weight of how long the distance is between two intersections.

Graphs may also be *connected* or not and either *complete* or not [11]. A connected graph has the property that there exists a path between any pair of nodes. In a complete graph, every node must have an edge to *every* other node in the graph. If a graph is complete, it must also be connected. If a graph is not connected it cannot be complete.

Once again using the example of Facebook, the graph consisting of all people and friendships would likely not be connected or complete. It is probably not connected since it is not necessarily true that you can start at any person and reach all other people by traversing friendships. It is not complete since all people are not friends with all other people. In the example of roads and intersections, this would be a connected graph since all intersections can be reached, no intersection can exist on its own. It would however not be complete since not all intersections are connected to all other intersections.

Figure 2.4(a) depicts a graph with four nodes and four edges. The graph is connected but not complete. It is connected since there exists a path between any pair of nodes. The neighborhood of node a is the sub-graph consisting of nodes a, b, and d together with their connecting edges. The neighborhood of node c is the sub-graph of nodes c and b and the edge between them.

In Figure 2.4(b) two edges have been added and the graph is now complete. The neighborhood of any of the nodes is now the entire graph, a statement that holds for all complete graphs.

2.3 Graph Neural Network

A Graph Neural Network (GNN) is a class of ANNs, designed for handling data that can be represented as graphs [24]. In its most simple form, a GNN takes a graph as an input and gives a graph as an output. By adding a final layer, or *head*, the graph which is output may however be aggregated in order to better suit a specific task. A popular application for GNNs is in studying citation networks where graphs describe directed citations between a collection of documents. One of the advantages of using a GNN is that the size of the input may vary. In a Convolutional Neural Network (CNN), input images always need to be of the same size which means lowering the resolution is a common pre-processing step. When using GNNs however, the number of input nodes and edges do not need to be the same over all data points, making for a more versatile architecture [33].

Message Passing

A key feature of GNNs is *message passing*. In GNNs, message passing refers to the ability of nodes to share information between each other via edges [9]. For those familiar with CNNs, this is similar to how the convolution operator works on an image, in which case pixels work as nodes with connecting edges to adjacent pixels. In the simplest cases, a linear operator such as addition may be performed on the information passed around. Recently however, more advanced techniques have been developed which are able to model complex data in a better way.

Graph Attention Networks

Introduced in 2018, Graph Attention Networks (GATs) [26] build off of the additive *attention* mechanism by Bahdanau et al. [1] and the *multi-head attention* used in the Transformer [25]. Attention is a mechanism used to allow a network to only focus on, or attend to, the most important parts of the data. Popularized by the introduction of the Transformer and especially common in the field of Large Language Models (LLMs) used in for example ChatGPT (the T in GPT stands for Transformer). When dealing with inputs in the form of text it is important for the network to be able to learn which words, or perhaps even sub-words, are of importance and which are not, which should it attend to, and which can be ignored.

GATs work similarly and allow the network to attend differently to different parts of a graph and can assign different importance to nodes in the same neighborhood. By doing this the network is able to learn which nodes and edges play an important role in solving the task at hand and can therefore complete the task more efficiently and more accurately. By using multiple heads, many latent attention features can be computed and are then concatenated to give the final output feature representation.

GATv2

GATv2 [3] is a proposed improvement to GAT. By modifying the order of operations, GATv2 computes a more expressive form of attention referred to as *dynamic* by the authors, in contrast to the *static* attention computed by GATs. GATv2 is shown to outperform GAT on multiple open graph benchmarks while having the same time complexity. When combining the Edge-Featured GAT (EGAT) [31] and GATv2, the attention coefficients are computed as per Equation 2.3 and Equation 2.4. This is how GATv2 is implemented in the PyTorch Geometric library [8].

$$\alpha_{i,j} = \text{softmax}_j(e(\mathbf{h}_i, \mathbf{h}_j)) = \frac{\exp(e(\mathbf{h}_i, \mathbf{h}_j, \mathbf{e}_{i,j}))}{\sum\limits_{j' \in \mathcal{N}(i)} \exp(e(\mathbf{h}_i, \mathbf{h}_{j'}, \mathbf{e}_{i,j'}))} \tag{2.3}$$

$$e(\mathbf{h}_i, \mathbf{h}_j, \mathbf{e}_{i,j}) = \mathbf{a}^\top \text{LeakyReLU}(\mathbf{W}[\mathbf{h}_i || \mathbf{h}_j || \mathbf{e}_{i,j}]) \tag{2.4}$$

The new node features are then computed as per Equation 2.5.

$$\mathbf{h}'_i = \sum_{j \in \mathcal{N}(i)} \alpha_{i,j} \mathbf{W} \mathbf{h}_j \tag{2.5}$$

In Equations 2.3, 2.4 and 2.5 the following notations are used:

$\alpha_{k,l}$ - attention coefficient for the edge between nodes k and l

$\mathbf{h}_k$ - current node feature vector for node k

$\mathbf{h}'_k$ - updated node feature vector for node k

$\mathbf{e}_{k,l}$ - edge attribute vector for edge between nodes k and l

$\mathcal{N}(k)$ - the set of nodes in the neighborhood of node k

$\mathbf{a}^\top$ is a learned parameter vector and $\mathbf{W}$ is a learned parameter matrix

LeakyReLU - **Leaky Re**ctified **L**inear **U**nit, $\text{LeakyReLU}(x) = \begin{cases} x, & \text{if } x \geq 0. \\ \text{negative\_slope} \cdot x, & \text{otherwise} \end{cases}$

$||$ denotes vector concatenation

The equations can be summarized as follows. For node i, the function e computes a value for each edge (j,i), where j is a neighbor of i or i itself, representing the importance of the features of j to i, or how much j attends to i. These attention values are then normalized over all neighbors, $j \in \mathcal{N}(i)$, using the softmax function, defined as in Equation 2.3, to give an attention coefficient for each edge in the neighborhood of i. Each attention coefficient will be a number between 0 and 1, $\alpha_{i,j} \in [0,1]$, and can be thought of as a percentage of the total effect of node j to node i. Each node's feature vector is then updated w.r.t. each of its neighbors' feature vectors and attention coefficients.

When using multiple heads, the attention coefficients are computed in parallel for each head and then concatenated to give one final new node feature vector. When edge attributes are not present, Equation 2.4 is simply tweaked to not include the term $\mathbf{e}_{i,j}$.

3 Methodology

In this chapter, an account of the working process of the thesis is given.

3.1 Data

The foundation of the data used in this thesis is short clips of football games. These clips vary in length, the shortest are around 6 seconds and the longest are 30 seconds. The clips are recorded by a camera placed in alignment with the middle of the pitch along one of the long sides and is most often positioned around 5-20 meters above the ground. The clips show games played by men, women, and children. Each clip is accompanied by a file containing information about the recording, such as the camera's position relative to the pitch, the pitch size, and a tag label. The tag label describes what is happening in the clip and three examples of these tag labels are "General-Play", "Kick-Off", and "Free-Kick".

These clips are processed to extract the information that will be used to train a model. This processing consists of two things: detection of players and annotation of the ball. The players are detected by a CNN which has been trained to detect and classify people playing sports. For each frame in each clip, the detected people in the frame are classified as either players or spectators, but only those classified as players will be used as data. The players' detections are represented by a bounding box and a unique ID. This is the basis of the input data to the model. The ball's position has been manually annotated every three frames and linearly interpolated between the annotations. The ball's position consists of a coordinate in the image space only and will be used as ground truth. An example of a frame from a clip with the tag "General-Play", with detected people and the annotated ball circled, can be seen in Figure 3.1. This example also shows that the detections of players are not perfect. The two players that are closest to the ball are being detected only as one, since there is only one bounding box around them. As discussed in Section 2.3 the model needs to be able to handle these cases where the size of the input varies, and that is one of the reasons why a GNN architecture should be well suited for the task.

Pre-processing

Before this data is used however, some final pre-processing is done. The coordinates of the players and the ball are projected onto the pitch to give the positions in world space instead of image space. This is made possible through a number of known parameters such as camera position, orientation, settings, the locations of pitch lines in the image, and the pitch size. Through matrix operations not necessary to explain for the purpose of this thesis, the real-world positions are extracted from the image space coordinates. This step is included to make the data invariant to which directions objects are moving. Since the camera is not recording from straight above the pitch, but instead at an angle, vertical and horizontal movements on the pitch will yield different changes in pixel values in the image space. Moving 1 meter vertically (long side to long side) will show less change in pixel values than if the movement was 1 meter horizontally (short side to short side). Reasonably, movements should carry the same weight no matter their direction, and by projecting all positions onto the pitch this is achieved.

Through the first steps of working with this data, it became clear that a model would not be able to learn the structure of the data because of one significant problem. When the ball is traveling through the air, projection onto a two-dimensional pitch is not possible. Since the ball is only annotated as a single point and not a bounding box, the distance from the camera to the ball is not known and therefore assumptions made when projecting the ball onto the pitch will not hold when the ball is not on the

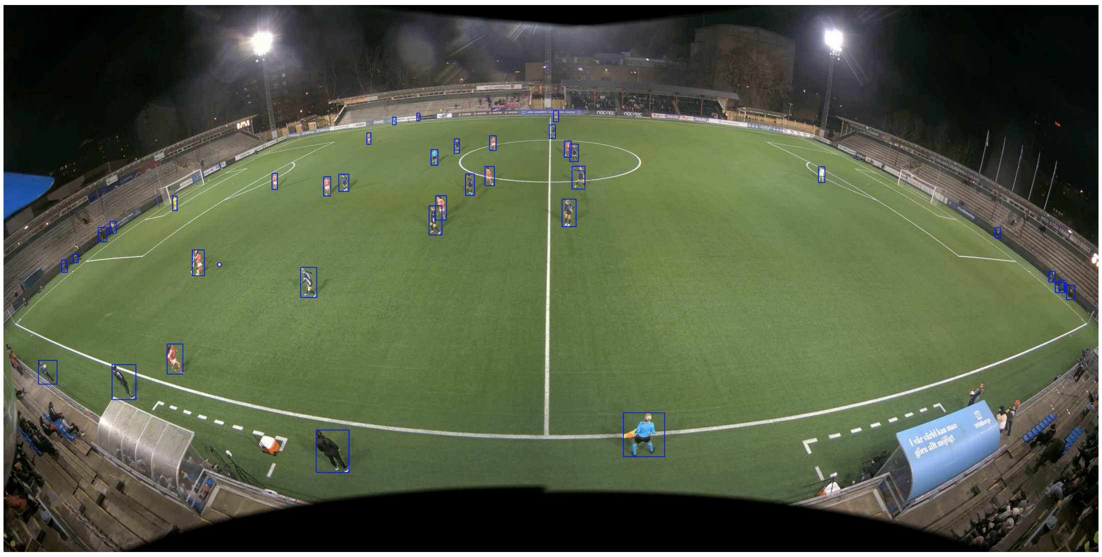


Figure 3.1 Example of a frame from the data set. The clip has tag "General-Play", the detected people are surrounded by their respective bounding boxes, and the annotated ball has been circled.

ground. Had the ball been annotated as a bounding box, then the size of that box could have been used to determine the ball's position in three-dimensional space and more of the total data could have been used.

Having established that occurrences of unwanted data exist in the data set, a way of handling this must be determined. It was decided that removing all data where the ball was not on the ground was the best way to do so and therefore also restrict the thesis to only working with two dimensions. Since manually going through the clips to find and remove frames where the ball is in the air would be a rather time-consuming task, a simpler and significantly faster way of doing so was thought of. Figure 3.2 shows an example of three different views of a ball's trajectory when it is traveling through the air. In (a) is shown how the ball moves over the image as seen by the camera. Note the warped view of the pitch due to the camera's position. In (b) is shown how the ball moves as seen from above. With these two images, it is possible to deduce that the ball is moving horizontally along the long side closest to the camera as well as up into the air. However, when projected into world coordinates the image in (c) is the result, a trajectory where the ball appears to move in a parabola from one long side to the other and then back again.

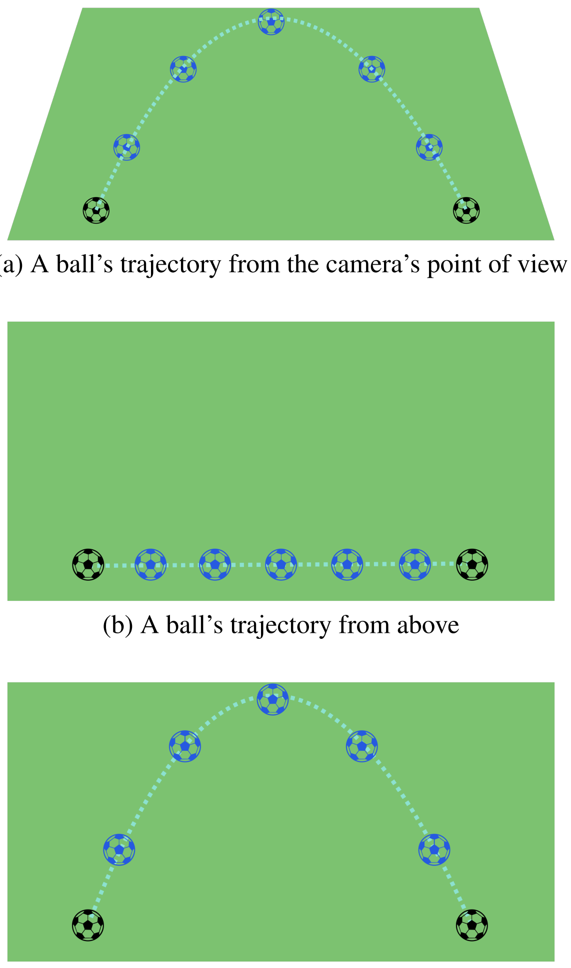
Since both (b) and (c) take place during the same amount of time but the "ball" will have covered a much larger distance in (c), the velocity of the ball will also be much higher. So to automatically find and remove frames where the ball is not on the ground it may be sufficient to look for when the projected ball has an abnormally high velocity. By making a quick Google search it became obvious that it is unlikely for the ball to move faster than 180 km/h, or 50 m/s, and so the threshold was set at that velocity. Any frames in which the projected ball moved faster than that were discarded and not used.

Once the pre-processing was done the final distribution of the data tags was as in Table 3.1. The total number of frames available for training after pre-processing was 60767.

Augmentation

To expand the data set and make for a more robust model, data augmentation is used. Reflection symmetry is assumed meaning that vertical and/or horizontal reflections do not impact the validity of the data. Using the dihedral group $D_2 = \{id, \leftrightarrow, \updownarrow, \leftrightarrow\updownarrow\}$ as the set of augmentations the data set can be quadrupled in size. Here, *id* represents the original data, $\leftrightarrow$ is the horizontal flip, and $\updownarrow$ is the vertical flip. These augmentations are applied once the data has been projected onto the pitch. An example of a frame with its D_2 augmentations is shown in Figure 3.3. [*]

[*]Though in this illustration the people appear upside down, their positions are all that is of interest, not their orientations. Therefore the augmentations are considered valid and can be used as new data points.


(a) A ball's trajectory from the camera's point of view

(b) A ball's trajectory from above

(c) A ball's trajectory from above, according to the projection algorithm

Figure 3.2 Example of different views on a ball in the air

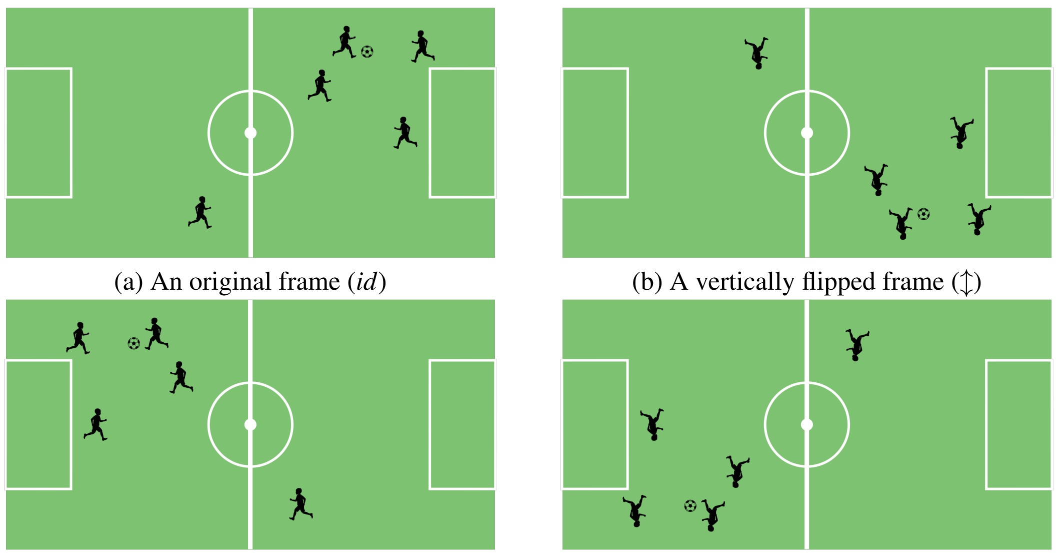
Category	Tag	Percentage of data set
Open Play	General-Play	38%
	Shot	5%
	Goalkeeper-Throw	4%
	Cross	4%
	Goalkeeper-Punt	3%
Set Piece	Free-Kick	17%
	Kick-Off	17%
	Throw-In	5%
	Corner	5%
	Penalty	1%

Table 3.1 Distribution of data per tag

Data split

The clips are split into training, validation, and test sets with roughly a 70/20/10 split between them. To ensure that the test set is not randomly created to contain only one or a few kinds of clips, the split is done in such a way that the distribution of tag labels is similar in all three sets. This way it is made sure that the sets the model is evaluated on are representative of the data as a whole and so that no tag label is left out from any set.

It is also ensured that no clip can have its frames distributed in more than one set. If one portion of the frames from a clip appears in the training set and the other part in the test set, the test set may



(a) An original frame (*id*)

(b) A vertically flipped frame ($\updownarrow$)

(c) A horizontally flipped frame ($\leftrightarrow$)

(d) A horizontally and vertically flipped frame ($\leftrightarrow\updownarrow$)

Figure 3.3 Examples of augmentations to a frame

not fulfill its purpose. The test set is there to evaluate the model on data that has not been seen by the model before to best simulate how it would be applied in the future. This criterion will always hold strictly since no single frame will appear in more than one set. However, due to the high sample rate of the data, if a clip is allowed to have its frames spread out over multiple sets, many data points in different sets will be very similar. Since the data is sampled at 25 frames per second, two following frames are captured $1/25 = 0.04$ seconds apart. If one frame is allowed to exist in the training set and its following frame in the test set, this could be regarded as "cheating". The model is likely to perform better on the test set than what is actually representative of how it would perform in the future where it will encounter frames that will not be as similar to the training data. For this reason, all clips can only have their frames occur in one of the three sets.

The distribution of tags in each set is similar to that of the entire set (Table 3.1) No augmented data is used in the test set as it is important to keep this set true to real-world scenarios and similar to what data the model will see when being applied and implemented.

3.2 Model Architecture and Feature Representation

As mentioned in Section 1.3, Related Work, TacticAI [29] has acted as an inspiration for the model base used in this thesis. The model is a GNN consisting of nine GATv2 layers each with eight attention heads, each of those computing 20 latent features. A dropout layer is included after three and six GATv2 layers respectively. Finally, a linear layer is implemented to aggregate the features into an output of two dimensions, one for each coordinate of the predicted location of the ball.

Once the model architecture was chosen, little focus was spent trying to optimize neither the structure, such as the number of layers or attention heads per layer, nor the hyperparameters used for training. Instead, the main focus went towards how to represent the data in such a way that it is most useful to the model. This can be done in a number of different ways. Features can be placed on nodes or edges, edges can be included or left out and information about velocities can be extracted from the player's historical positions. The pre-processed data will be represented in a number of different ways to gain information about what the model finds useful and to hopefully in the end have been able to train a model to accurately predict the position of the ball. The different data representations are explained below.

Position

The simplest form of the data is to leave it as is and to only input to the model the current positions of all players. In that case, the data is represented as a complete graph where every player makes up a node with two features: the x- and y-coordinate at that time. The edges were not given any attributes with hopes of the model being able to learn the spatial relationships between players on its own.

Position and Velocity

A slightly more advanced representation is to also input the current velocities of the players. This representation is similar to when only inputting the positions, the only difference being that each node in this case has four features: x- and y-coordinates as before and velocities in x and y direction respectively. To calculate the velocity of a player, its unique detection ID is matched with the same ID in an earlier frame. The velocities were then calculated as the difference between the current position and the earlier position and then divided by the time difference to give the velocity at that frame in m/s. For the first frame in a clip, the player's velocities are set to zero since there is no previous frame to compare with. For the following frames the velocities in x and y direction for a player at frame i are calculated respectively as

$$v_{i,x} = \frac{x_i - x_{i-d}}{d/\text{framerate}}, \qquad v_{i,y} = \frac{y_i - y_{i-d}}{d/\text{framerate}}, \qquad \text{where } d = \min\begin{cases} i \\ 10 \end{cases}.$$

Since the players are detected and not manually annotated there will be frames where one or more detection is missing. In such a case the tracking of a player is broken up and when it is detected again it will have a new ID. In such a case, when the ID of a player does not exist in frame $i-d$, the velocity is also set to zero in both directions.

Position and Previous Position

To investigate whether the model was able to calculate or represent the velocities on its own, perhaps in a more useful way, another representation of data was used. In this case, the data consisted of the positions in the current frame as before, along with the positions in an earlier frame. This could be done in a few different ways. It was chosen to do this by including the previous players as separate nodes with their positions as features. Then all players are connected to every other player in the current frame and their previous selves. Every previous player is in their turn connected to every other previous player. In Figure 3.4 is an example of what this could look like, $t = 0$ being the current frame and $t = -10$ being the previous frame. Every edge is now also equipped with an attribute indicating how many frames apart the nodes are. If the nodes that make up the endpoints of an edge are from the same frame then that edges attribute would be 0. If they are not, the attribute is the number of frames they differ. From the example in Figure 3.4 one such edge would have 10 as its attribute. The previous frame that is included is chosen in the same way as when calculating velocities. For a frame with index i, the previous frame is chosen as

$$\text{Previous frame} = \begin{cases} \text{None}, & \text{if } i = 0 \\ i - d, & \text{otherwise, where } d = \min\begin{cases} i \\ 10 \end{cases}. \end{cases} \tag{3.1}$$

Sequence of Frames

To utilize the frequent sampling of the data, a sequence of frames can be used as input. This is a combination of using previous positions as well as velocities. Each node will have the four features that are made up of position and velocity and each node will be connected to every other node from the same frame, as well as all of its previous selves. Figure 3.5 shows an example with three previous frames. This can be done with many different configurations, both in terms of how many frames to include and how closely spaced the frames used should be. It is likely not very useful to include frames directly adjacent since only 1/25 of a second has passed between them and little is likely to have changed in terms of positions and velocities. By spacing the frames used, the hope is to still capture as much information as possible but to do so in a more computationally efficient way. The model which showed the

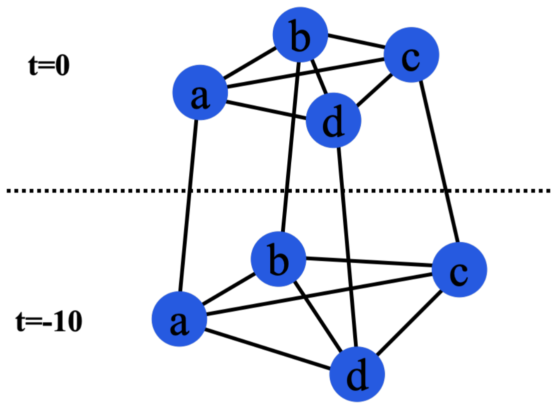


Figure 3.4 Example of how data can be represented with two frames as input. Every node is connected by and edge to every other node in the same frame, as well as their previous selves.

best results from experimenting with sequence length and spacing is presented in the following chapter and uses every fifth frame up to 2 seconds back in time. Again each edge is equipped with an attribute indicating the number of frames between the two nodes. For two nodes in the same frame, this would be 0 but for edges that link players between frames, this number would then be any multiple of 5 up to 50 (2 seconds $\cdot$ 25 frames per second). The motivation behind structuring the data as described is that hopefully the model is able to learn more advanced and useful trajectories of the players, in terms of both direction and speed since knowing previous velocities allows acceleration to be calculated.

Sequence of Frames and Previous Ball Position

To further investigate the network's suitability for the task at hand and if the network could work alongside an OD network, the ball's previous position was also included as input data. The idea is to simulate the outputs of an OD network acting as input into the GAT. When doing so there will be cases when the OD network is able to detect the ball and then that information should be of use for the GAT. In practice, this meant adding the ball's annotated previous position as another node in the input data. Since the number of features for each node has to be the same, its features were also its position and velocity. Beyond this, every node (player and ball) got an additional fifth feature. This feature was a one-hot-encoding for the network to distinguish the ball from the players. Every player's feature got the value 0 and the ball got the value 1. If the ball's previous position is useful the network should learn to attend more to the node having a 1 as that fifth feature. Since adding the ball's position only one frame earlier would likely make the problem too easy for the network, a gap between the current frame we are predicting and the frame from which the ball is gathered is probably a good idea. This gap was set to 2 seconds which should provide the network with information that is useful yet not giving the current position away.

3.3 Training

The networks were trained using the Adam [14] optimizer, MAE, 2.1, as the loss function, and a learning rate scheduler, decreasing the learning rate for each epoch of training. All training was done using the PyTorch library [20] and the data was structured as graphs using PyTorch Geometric [8]. All models were trained on GPUs provided either by the Berzelius resource at the NSC or the Alvis resource at Chalmers University of Technology. The exact hardware and training parameters specific for each model can be found in Appendix A. Hyperparameters were tested and set up when initially working with a small sample of the entire dataset when first setting up the pipeline. After that, little tuning was done to these as the focus remained on the representation of the data.

3.4 Establishing a Baseline

When evaluating the model results, absolute numbers are not always very useful. In order to properly determine if the model is useful it must be compared to a simple estimate of where the ball might be.

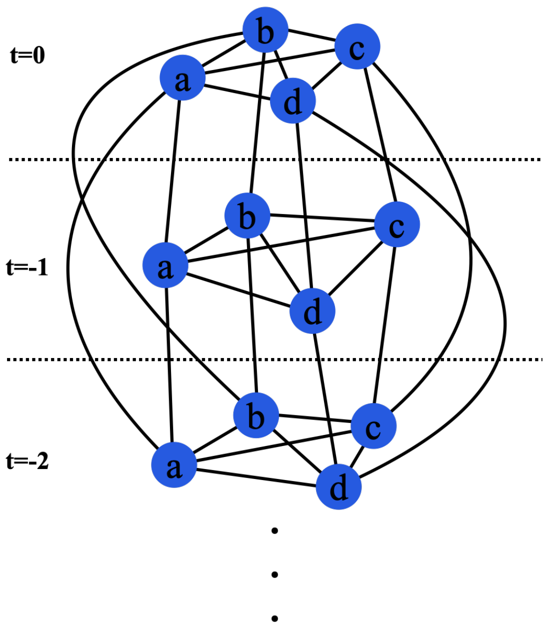


Figure 3.5 Example of how data can be represented with multiple frames as input. Every node is connected by an edge to every other node in the same frame, as well as all of their previous selves.

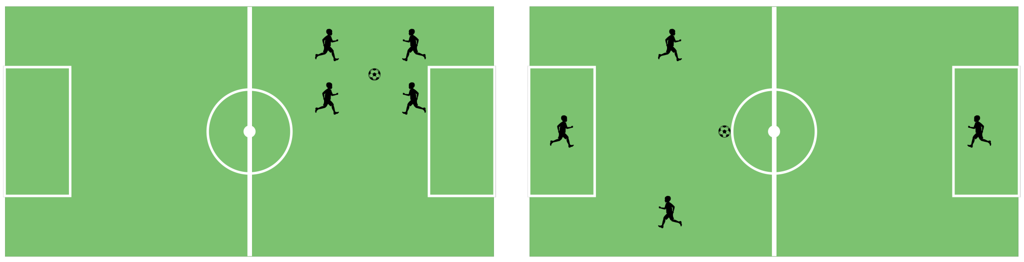


Figure 3.6 Examples of approximate baselines. The baseline prediction is represented by a football in the diagram and is calculated as the average position of all detected players on the pitch.

This is known as the *baseline* and in this thesis, it was chosen to be the average or middle position of all the players. Since most players most of the time want to be close to the ball this should work as a naive guess in a lot of cases, though it is apparent that this will not always be a very good guess. For example, when corners are being taken, most players are inside one of the penalty areas but the ball is actually by the corner flag. This is fine however since the baseline should be a very simple guess to compare the results to. If a model yields worse predictions than the baseline then it is clear that that model is unable to learn the underlying structures of the data and can be deemed not useful. Two examples can be seen in Figure 3.6 where the ball in this case illustrates approximately where the baseline prediction would be.

3.5 Evaluation

When evaluating the model predictions, a couple of different metrics were chosen. Most intuitive is the mean prediction error, i.e. on average how far away is the model's prediction from the ground truth position of the ball. The median prediction error was also calculated. By comparing the median and the mean it is possible to draw conclusions about the distribution of the prediction errors. The percentage of times that the prediction was better than the baseline prediction was also calculated. In addition to these three metrics, the histograms of the prediction errors were plotted to get a sense of the distribution of the prediction errors. Finally, to determine if the model is of use for automated broadcasts and automated data analysis, the percentage of times the prediction was within two thresholds was calculated. For automated broadcasts, this threshold was set to 10 meters and for data analysis, it was 1 meter.

3.6 Ablation Study

To not only evaluate the structure of the data or the features used, an ablation study was performed on the pre-processing of the data. To ensure that the pre-processing steps were motivated and helped get a better model in the end, a model was trained without using the augmented data and without having removed balls traveling through the air. By analyzing the results from these trained models, conclusions about the usefulness of the two pre-processing steps can be drawn.

4 Results

In this chapter, the results gathered in this thesis are presented along with an ablation study of some of the pre-processing of the data.

4.1 Results

The mean prediction errors in meters for the different models are presented in Table 4.1. An explanation of the model names is given below.

Category	Tag	Baseline	Pos	Velo	Prev Frame	Seq
Open Play	General-Play	20.7	15.7	**13.1**	13.5	17.6
	Shot	19.8	16.3	**12.8**	17.2	23.9
	Goalkeeper-Throw	27.3	**11.9**	16.0	19.2	21.5
	Cross	26.9	12.5	12.9	**10.9**	15.8
	Goalkeeper-Punt	34.1	22.2	26.4	**17.0**	24.7
Set Piece	Free-Kick	20.3	**12.6**	13.3	15.6	16.0
	Kick-Off	**6.0**	7.8	6.8	9.5	10.3
	Throw-In	17.6	12.5	**7.0**	10.0	9.8
	Corner	24.0	20.5	**16.1**	22.7	29.4
	Penalty	12.9	17.3	28.9	**11.8**	25.1
	Total	19.6	14.5	**13.0**	14.2	17.3

Table 4.1 Mean prediction errors (m) for each tag for the four models and the baseline prediction. The lowest mean error for each tag is typed in boldface font. The velocity model performs best in total and on four out of ten tags.

Pos is the architecture that only takes the players' current positions on the pitch as input. **Velo** takes the players' current positions and velocities as input. **Prev Frame** takes the current position and the positions up to 10 frames earlier as input. **Seq** takes a sequence of players' positions and velocities as input. The model presented in Table 4.1 and Table 4.3 used 10 previous frames, spaced 5 frames apart.

In Table 4.2 further performance metrics for the different models are presented.

	Baseline	Pos	Velo	Prev Frame	Seq
Better than baseline	-	68.2%	**74.8%**	71.4%	61.0%
Within 10m	15.2%	33.2%	**42.6%**	39.6%	26.1%
Within 1m	0.2%	0.4%	0.5%	**1.0%**	0.3%
Mean inference time	**0.03ms**	7.0ms	6.9ms	6.9ms	9.3ms

Table 4.2 Further performance metrics for the four models and the baseline. The best scores are typed in boldface font. The velocity model is closer than the baseline prediction in the highest percentage of frames. The baseline model has the shortest inference time.

Table 4.3 presents results for the sequence model trained with and without the ball's previous position as part of the input data. Both models were trained without augmented data, hence the difference in performance to Table 4.1. The model which takes the previous position of the ball as input data has access to the ball's annotated position 2 seconds earlier.

Category	Tag	Seq	Seq w/ prev. ball
Open Play	General-Play	25.0	**16.5**
	Shot	**19.4**	24.5
	Goalkeeper-Throw	**15.6**	28.0
	Cross	18.5	**14.3**
	Goalkeeper-Punt	**27.5**	30.7
Set Piece	Free-Kick	**18.8**	20.6
	Kick-Off	15.4	**8.6**
	Throw-in	**11.7**	20.1
	Corner	**15.3**	39.4
	Penalty	**12.6**	23.6
	Total	21.7	**18.1**

Table 4.3 Mean prediction errors (m) for the sequence model, with and without the ball's previous position as input. The lowest mean error for each tag is typed in boldface font. With the ball's previous position as input the mean prediction error is reduced in total and especially on "General-Play".

4.2 Ablation Study

In Table 4.4 results of the ablation study are presented. The best-performing model was trained without augmented data and without the removal of frames where the ball is in the air.

Category	Tag	Velo	No augmentation	With ball in air
Open Play	General-Play	**13.1**	14.8	15.7
	Shot	**12.8**	14.2	19.8
	Goalkeeper-Throw	16.0	15.8	**15.0**
	Cross	12.9	**9.7**	23.3
	Goalkeeper-Punt	26.4	**16.0**	20.5
Set Piece	Free-Kick	**13.3**	13.6	16.4
	Kick-Off	**6.8**	8.2	10.4
	Throw-In	**7.0**	13.1	13.8
	Corner	**16.1**	30.7	24.8
	Penalty	28.9	29.7	**27.4**
	Total	**13.2**	14.8	16.3

Table 4.4 Mean prediction errors (m) for the velocity model trained on different data sets: with augmentation and having removed frames where the ball is in the air, without augmentation, and with frames where the ball is in the air. The lowest mean error for each tag is typed in boldface font. The model scores the lowest mean prediction error having done both pre-processing steps.

4.3 Example Prediction

An example of a prediction for a frame with tag "General-Play" can be seen in Figure 4.1, the same frame as in Figure 3.1. The annotated ball has been circled in blue, the prediction (by the velocity model) is the green dot and the baseline prediction is the red dot.

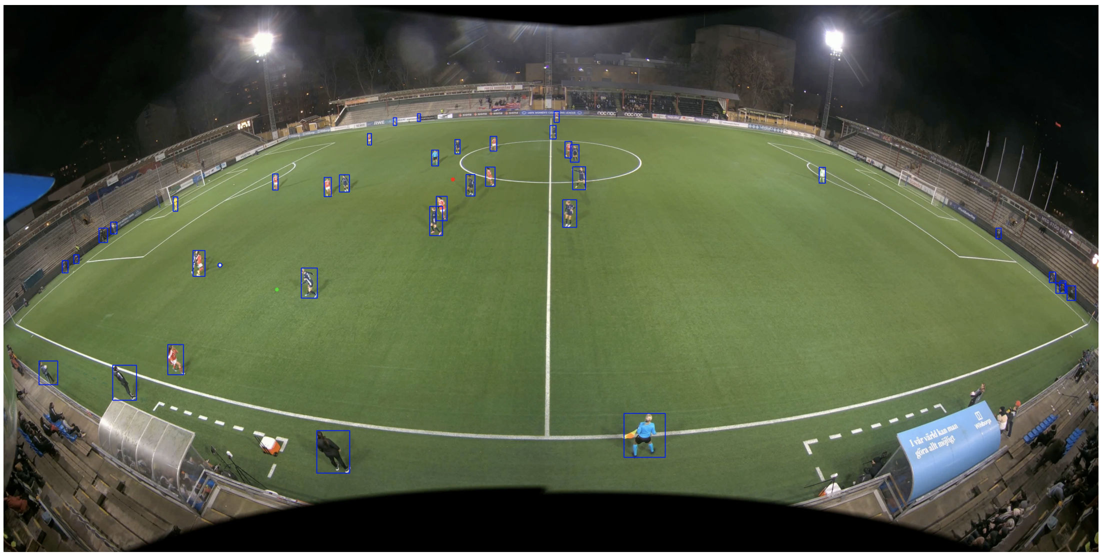


Figure 4.1 Example of a prediction on "General-Play". The ball is annotated in blue, the prediction is made by the velocity model in green and the baseline prediction in red. People detected in the frame are surrounded by bounding boxes.

5

Discussion

This chapter contains a discussion about the choices made throughout the thesis along with an analysis centered around the results presented in the previous chapter.

5.1 Pre-Processing

Removal of Ball in Air

Removing frames where the ball is in the air was not done from the start but it became clear early on that handling those frames would be necessary. There were quite a few instances of very high prediction errors at first and upon further investigation it was evident that these errors occurred when the ball was not in contact with the pitch. This problem was discussed with the supervisors and the agreed-upon solution was to try and remove the problematic frames. A few different ways of doing this were discussed and the discussion resulted in calculating the speed of the ball and removing those frames where the ball is moving faster than what is physically probable. The speed of the ball was calculated by dividing the distance traveled by the time elapsed between the current frame and 10 frames earlier. This assumes that the ball has moved in a straight line between now and 0.4 seconds earlier which will not always be true. If if is not true however, the actual speed of the ball will be even greater and should therefore be removed anyway. A more advanced approach would be to estimate the ball's motion as a function of higher order. This was however deemed not necessary for the purpose of removing frames from the data set and would likely not have made a big impact on what data was removed and what was kept. The ball's trajectory over such a short period of time can likely be estimated as a straight line without it being too big of an over-simplification.

A problem with removing frames as described is that frames where the ball is on the pitch but moving very quickly run a risk of being removed. This leads to simplifying the problem since data that would likely be hard to predict is discarded. Setting an appropriate threshold for the ball's velocity is key to not over-simplify the problem and by setting it to 50m/s, it is highly unlikely that any data where the ball is on the pitch has been removed. Though the risk exists, it was deemed worth taking in order to clean up the data set so that the results acquired were meaningful.

Another way of detecting when the ball is in the air is to calculate the acceleration of the ball and how far away the closest player is. When the ball is kicked into the air it may have changes in its acceleration despite no player touching the ball, or even being close to it. This way, those frames could be discarded since the change in acceleration must have been due to gravity and not a player being in contact with the ball. This method reduces the likelihood of discarding frames where the ball is on the pitch however, it would likely not catch all scenarios when the ball is in the air. Since the ball can sometimes appear to be close to players despite it being several meters up in the air, as illustrated in Figure 3.2, many frames would not be discarded despite the ball being in the air. Therefore this method was not implemented.

5.2 Model Architecture and Hyperparameters

The model architecture and its hyperparameters were experimented with mainly in the early stages of the thesis. When for example the number of layers, number of attention heads per layer, learning rate, and dropout rate gave useful results for a smaller sample of the data, it was then used on the entire data set as well. By tuning these parameters it is likely that it is possible to get better results than those acquired in this thesis.

A sacrifice in model complexity was made when the input data was structured as a sequence of frames. Every player node is connected to every other player in the same frame as well as all their previous selves. It is possible that information goes to waste since this graph is not complete since the players are not connected to other players in previous frames. The gain of making the graph complete is believed to be very small however and since it would make the input data much larger and therefore computationally heavy to handle, those extra edges were left out. For example, if 22 players are detected in each frame and 10 frames are used, a 7.3x increase in the number of edges is required to make the graph complete. Beyond this, the number of edge attributes would need to increase by the same amount. The computational limitations will be discussed further in Section 5.4.

5.3 Data

Player Velocities

As discussed earlier, there are more advanced ways of calculating velocities than the method used in this thesis. By approximating the player's velocities as linear it is possible that complex information about the player's movements gets lost. Once again however this loss is deemed small and likely not significant since the velocities are calculated over a short span of time (0.4 seconds) and motions of players can likely be estimated as straight lines without much information being lost. Therefore the method used is motivated. It is also possible to change the gap between the frames from which the velocities were calculated. A gap of 10 frames was used but perhaps fewer would make for a more accurate approximation.

Placement of Features

As mentioned in Section 3.2, there are a few different ways of structuring the data as a graph. In this thesis, most of the features were placed on the nodes, such as the player's positions and velocities, and edge-level features were only used when using previous frames. Placing features on the edges could have been done for other structures as well. For example, every edge could have the distance between the two nodes as a feature. Since the players' positions exist as node-level features the hope is that the model is able to extract this information on its own (if it is useful), but perhaps it is not able to do so. Then calculating this extra piece of information for the model could end up affecting the model's performance.

5.4 Computational Limitations

To an extent, this thesis was limited by the computational power available. A limited amount of GPU hours allocated from the GPU cluster resources meant that what models to train and on what kind of hardware had to be chosen with some level of care. There were also times when the hardware was not able to handle the size of the data points constructed. For example, when constructing a sequence of frames as input the number of frames that could be included was limited due to the GPU memory available. Perhaps a more advanced GPU would be able to handle bigger data points and therefore longer sequences of data could have been tested.

It is also possible that the PyTorch Geometric library is not optimal for structuring the data. Perhaps another library, such as TensorFlow-GNN [7], is more efficient in storing data and could therefore take up less memory while containing the same amount of information.

5.5 Results

Model Performance Metrics

The lowest total mean prediction error was achieved by the model taking only current positions and velocities as input at 13.0 meters. This model also acquires the lowest mean prediction error on four out of ten tags and outperforms the baseline on eight out of ten tags. In total this model reduces the prediction error by 34% compared to the baseline and predicts closer to the ground truth 74.8% of the time. The model's prediction is within 10 meters of the ground truth in 42.6% of the frames and within 1 meter 0.5% of the time. The mean inference time on an NVIDIA T4 [18] GPU is 6.9ms and can therefore process roughly 145 frames per second. The poor results on the tags "Goalkeeper-Punt" and "Penalty" are likely due to the small amount of data available for these scenarios.

Interestingly the model taking only the current and one previous position as input performs better on crosses, goalkeeper punts, and penalties. Though these make up a small part of the test set and consist only of a few examples each, it might indicate that for those scenarios the players' previous positions are more useful to know than their velocities when predicting the ball's position.

Another interesting result is that the model only using the current positions is best at predicting the ball's position on goalkeeper throws and free kicks, and outperforms the previous frame model on shots, kick-offs, and corners as well. It is hard to say with certainty why this is but it points to the fact that the players' movements are not as important in these scenarios as their current positions, and in fact knowing the past positions only makes the predicting more difficult. Perhaps the structure behind the players' movements in these scenarios is too complex for the model architecture to capture or they simply do not carry any importance in predicting where the ball is. Since more features increase the risk of overfitting to the training data, it is useful to also look at how the models perform when evaluated on the training data. If the previous frame model performs better on the training data for the scenarios mentioned, the results are likely a consequence of overtraining. This would point to more data being necessary before any definitive conclusions could be drawn. When evaluating on the training data however, the same pattern is seen: the position model is able to make better predictions for the aforementioned scenarios than the previous frame model.

The baseline model performing best on kick-offs is not surprising since the two teams must have all players on their half of the pitch when the kick-off is taken. Therefore the average of all players' positions will always be somewhat close to the middle of the pitch, where the kick-off happens. The inference time being the lowest for the baseline model is also not surprising since it consists of few and simple mathematical operations compared to the other models.

The results of the sequence model are most surprising. This model has by far the most information available but performs worse than the rest, only performing better than the baseline. There are two reasons which are most probable to be the cause of this. Either the extra information fed into the model has no underlying structure and the importance of knowing players' positions and velocities further back in time has been exaggerated, or the model is unable to capture such an underlying structure. Most likely, a combination of the two is what yields the poor results. Perhaps the element of randomness to the players' movements becomes too large when looking too far back in time and including it as input has no meaning. Perhaps the model architecture is at fault and needs to be expanded or at least modified to be better suited to handle data of such a form. There is also the possibility that perhaps a GAT model is not the best choice for this data structure. There may be other layers more useful or perhaps even the idea of using a GNN must be scrapped in order to make use of this data. It is difficult to say with certainty why the model performs so poorly with so much information available and further testing beyond this thesis is necessary if definitive conclusions are going to be drawn. Since a model with more features is more likely to overtrain, it can be good to once again check that the model has not overfitted to the training data. When evaluating the model on the training data it can be concluded that the poor results are not because of overtraining since the prediction errors are not significantly better on the training data, neither in absolute numbers nor compared to the other models.

For most models and most of the tags, the median prediction error, presented in Table B.1, is lower than the mean prediction error. This indicates that many predictions have an error that is lower than the mean error in Table 4.1 and that a few number of occurrences with very large prediction errors raises the mean. When looking at the histograms in Appendix B this becomes even more clear. In Figure B.11 it can be seen that there are more occurrences of prediction errors greater than 40 meters for the velocity model than for the baseline. Most of these seem to come from the tags "General-Play" and "Goalkeeper-Punt" and the latter has the largest percentage of its frames being predicted more than 40 meters from the ground truth. By looking at the shape of the distribution for this tag and comparing it to others, it seems as if "Goalkeeper-Punt" really stands out. Perhaps this scenario is much harder to learn the underlying structure of, and that it differs a lot from other scenarios, or the lack of data for this tag means that the model has not been able to learn it.

With Ball's Previous Position

In Table 4.3 the difference in performance of the sequence model with and without the ball's previous position becomes clear. Knowing the ball's position 2 seconds earlier reduces the prediction error by a lot in scenarios when the ball generally does not move very fast, i.e. general play and kick-offs. In other scenarios when the ball generally covers a lot of ground during the clips, such as corners, shots, and goalkeeper punts, knowing the ball's previous positions only makes the prediction more difficult. Since Table 4.3 shows the performance of models that were not trained on the augmented data it is not certain how well the trends seen there would translate if they were trained on the augmented data. If the improvement in general play (34% reduction in mean prediction error) would be the same when training with augmented data, this model would set a new lowest mean prediction error for that scenario at 11.6 meters. Similarly, the total mean prediction error could be reduced to 14.4 meters.

These results show more promise. When implementing a GAT model in reality it would likely work in tandem with an OD network. The output from the OD network, i.e. detected players and a detected ball from time to time, would be input into the GAT model. Depending on how long ago a ball was last detected this information could be used as an input node to the GAT model. For general play, which makes up the majority of a game of football, this piece of information is clearly very valuable, at least when the position of the ball is 2 seconds old.

The problem, however, is that this model's predictions are not better than a naive approach would be with the same information. The prediction of the sequence model is in fact much worse than the naive prediction of guessing that the ball's current position is the same as it was 2 seconds ago, 9 meters worse in the mean prediction error on the test set. Once again this result points to the fact that the model is unable to capture any structure to the players' movements when looking further back in time.

Ablation Study

From Table 4.4 it is clear to see that both augmentation and removing frames where the ball is in the air have a positive effect on the mean prediction error and lower it significantly. This does not hold for all tags though but the reason for this can likely be attributed to the few number of clips that make up these frames in the test set. In conclusion, both pre-processing steps were of help when training the models and are therefore motivated.

6 Conclusions and Future Work

This section will wrap up the thesis and summarize the results and discussions. Additionally, some topics for future work are brought up.

6.1 Conclusions

The model using only current positions and velocities of the players on the pitch is able to improve on the baseline by 34% in the mean prediction error and makes a better prediction than the baseline 74.8% of the time. The model is best at predicting the ball's position on kick-offs and throw-ins and worst at goalkeeper-punts and penalties, though the poor performance on penalties is likely heavily influenced by the lack of data with this tag. In total, the model achieves a mean prediction error of 13.0 meters. For the model to be useful when automating broadcasts, the prediction should preferably be within 10 meters of the ball's actual position. This is achieved 42.6% of the time with this model. Thus the model could be used for automating broadcasts but one would expect it to not always work as smoothly as one might want. For automating data analysis the same threshold was set to 1 meter which is achieved 0.5% of the time. Therefore it is safe to say that the model can not be used for automating data analysis, it simply is not able to predict the ball's position well enough.

It is evident that using more information than just the players' current positions helps make accurate predictions in most cases. This can be done either by adding players' previous positions or their current velocities, which are calculated using their previous positions.

It seems though that there is a limit to how much history the architecture is able to make use of. When feeding the model information about the players' positions and velocities up to 2 seconds previously, the model makes worse predictions than with less history as input. This could be because the information has too much randomness to it for the model to make sense of the information, or it could be because the model architecture needs to be modified.

6.2 Future Work

To further try to answer the questions posed in this thesis, there are a few points that need to be investigated.

The first would be to modify the model architecture. It is very possible that this has held back the performance of the models which had sequences of frames as input but it is impossible to know for sure until it has been thoroughly tested.

Another would be to take inspiration from group equivariant convolutional neural networks [5] when augmenting data. By allowing a greater degree of weight-sharing, reflection symmetries can be utilized further than in this thesis. To further improve model robustness, another augmentation step could be implemented. By randomly removing an existing player or adding a fake player to the input graph the model should become more robust and less variable due to the randomness of players' positions and movements. Similar to how dropout works it should reduce the risk of overfitting and co-adaptation between nodes.

Implementing an update scheme for the predictions is a way to make the model more robust. By feeding the previous prediction into the model, it should make the predictions less likely to differ very much from one frame to another, making for a more useful model in practice. The sacrifice is that if the previous prediction was not very good, the model may need more time to correct this mistake and make closer predictions. If a model were to be implemented for automated broadcasts for example, this tradeoff might be worthwhile to avoid the camera making to large and fast changes to where it is directed and what it is recording.

More features than those used in this thesis could possibly be added. Either by applying a clustering algorithm to the input clips or by manual annotation, what team each player belongs to could be an additional feature that could be of use for a model. Human pose estimation is another example. By estimating the orientation of the limbs of the players it is possible to give a more advanced representation of players' movements. Though players often are oriented in the direction they are moving, this is not always going to be the case. Specifically, the orientation of players heads, i.e. where they are looking, should be valuable information.

Bibliography

[1] D. Bahdanau, K. Cho, and Y. Bengio. “Neural machine translation by jointly learning to align and translate”. In: Y. Bengio et al. (Eds.). *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. 2015. URL: `http://arxiv.org/abs/1409.0473`.

[2] I. A. Basheer and M. Hajmeer. “Artificial neural networks: fundamentals, computing, design, and application”. *Journal of microbiological methods* **43**:1 (2000), pp. 3–31.

[3] S. Brody, U. Alon, and E. Yahav. “How attentive are graph attention networks?” In: *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL: `https://openreview.net/forum?id=F72ximsx7C1`.

[4] T. Chai and R. R. Draxler. “Root mean square error (rmse) or mean absolute error (mae)?– arguments against avoiding rmse in the literature”. *Geoscientific model development* **7**:3 (2014), pp. 1247–1250.

[5] T. Cohen and M. Welling. “Group equivariant convolutional networks”. In: *International conference on machine learning*. PMLR. 2016, pp. 2990–2999.

[6] A. Dosovitskiy, P. Fischer, E. Ilg, P. Häusser, C. Hazirbas, V. Golkov, P. v. d. Smagt, D. Cremers, and T. Brox. “Flownet: learning optical flow with convolutional networks”. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 2758–2766. DOI: `10.1109/ICCV.2015.316`.

[7] O. Ferludin, A. Eigenwillig, M. Blais, D. Zelle, J. Pfeifer, A. Sanchez-Gonzalez, W. L. S. Li, S. Abu-El-Haija, P. Battaglia, N. Bulut, J. Halcrow, F. M. G. de Almeida, P. Gonnet, L. Jiang, P. Kothari, S. Lattanzi, A. Linhares, B. Mayer, V. Mirrokni, J. Palowitch, M. Paradkar, J. She, A. Tsitsulin, K. Villela, L. Wang, D. Wong, and B. Perozzi. “TF-GNN: graph neural networks in tensorflow”. *CoRR* **abs/2207.03522** (2023). URL: `http://arxiv.org/abs/2207.03522`.

[8] M. Fey and J. E. Lenssen. “Fast graph representation learning with PyTorch Geometric”. In: *ICLR 2019 Workshop on Representation Learning on Graphs and Manifolds*. New Orleans, USA, 2019. URL: `https://arxiv.org/abs/1903.02428`.

[9] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. “Neural message passing for quantum chemistry”. In: *International conference on machine learning*. PMLR. 2017, pp. 1263–1272.

[10] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[11] J. L. Gross, J. Yellen, and P. Zhang. *Handbook of Graph Theory, Second Edition*. 2nd. Chapman & Hall/CRC, 2013. ISBN: 1439880182.

[12] L. He, Q. Zhou, X. Li, L. Niu, G. Cheng, X. Li, W. Liu, Y. Tong, L. Ma, and L. Zhang. “End-to-end video object detection with spatial-temporal transformers”. In: *Proceedings of the 29th ACM International Conference on Multimedia*. MM ’21. Association for Computing Machinery, Virtual Event, China, 2021, pp. 1507–1516. ISBN: 9781450386517. DOI: `10.1145/3474085.3475285`. URL: `https://doi.org/10.1145/3474085.3475285`.

[13] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. “Improving neural networks by preventing co-adaptation of feature detectors”. *arXiv preprint arXiv:1207.0580* (2012).

[14] D. Kingma and J. Ba. "Adam: a method for stochastic optimization". *International Conference on Learning Representations* (2014).

[15] A. Kortylewski, J. He, Q. Liu, and A. L. Yuille. "Compositional convolutional neural networks: a deep architecture with innate robustness to partial occlusion". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).* IEEE Computer Society, Los Alamitos, CA, USA, 2020, pp. 8937–8946. DOI: `10.1109/CVPR42600.2020.00896`. URL: `https://doi.ieeecomputersociety.org/10.1109/CVPR42600.2020.00896`.

[16] T. Mitchell. *Machine Learning*. McGraw-Hill International Editions. McGraw-Hill, 1997. ISBN: 9780071154673. URL: `https://books.google.se/books?id=EoYBngEACAAJ`.

[17] NVIDIA. *A100 tensor core datasheet.* `https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet-us-nvidia-1758950-r4-web.pdf`. Accessed: 2024-07-29.

[18] NVIDIA. *T4 tensor core datasheet.* `https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-t4/t4-tensor-core-datasheet-951643.pdf`. Accessed: 2024-07-29.

[19] R. Padilla, S. L. Netto, and E. A. Da Silva. "A survey on performance metrics for object-detection algorithms". In: *2020 international conference on systems, signals and image processing (IWSSIP).* IEEE. 2020, pp. 237–242.

[20] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. "Pytorch: an imperative style, high-performance deep learning library". In: H. Wallach et al. (Eds.). *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc., 2019. URL: `https://proceedings.neurips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf`.

[21] J. Patterson and A. Gibson. *Deep learning: A practitioner's approach.* " O'Reilly Media, Inc.", 2017.

[22] D. Rozumnyi, J. Kotera, F. Sroubek, L. Novotny, and J. Matas. "The world of fast moving objects". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* IEEE, 2017. DOI: `10.1109/cvpr.2017.514`. URL: `http://dx.doi.org/10.1109/CVPR.2017.514`.

[23] K. Saleh, S. Szenasi, and Z. Vamossy. "Occlusion handling in generic object detection: a review". In: *2021 IEEE 19th World Symposium on Applied Machine Intelligence and Informatics (SAMI).* IEEE, 2021. DOI: `10.1109/sami50585.2021.9378657`. URL: `http://dx.doi.org/10.1109/SAMI50585.2021.9378657`.

[24] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. "The graph neural network model". *IEEE transactions on neural networks* **20**:1 (2008), pp. 61–80.

[25] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. "Attention is all you need". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Curran Associates Inc., Long Beach, California, USA, 2017, pp. 6000–6010. ISBN: 9781510860964.

[26] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. "Graph attention networks". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL: `https://openreview.net/forum?id=rJXMpikCZ`.

[27] A. Wang, Y. Sun, A. Kortylewski, and A. Yuille. "Robust object detection under occlusion with context-aware compositionalnets". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).* 2020, pp. 12642–12651. DOI: `10.1109/CVPR42600.2020.01266`.

[28] S. Wang, Y. Zhou, J. Yan, and Z. Deng. "Fully motion-aware network for video object detection". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018.

[29] Z. Wang, P. Veličković, D. Hennes, N. Tomašev, L. Prince, M. Kaisers, Y. Bachrach, R. Elie, L. K. Wenliang, F. Piccinini, and et al. "Tacticai: an ai assistant for football tactics". *Nature Communications* **15**:1 (2024). DOI: `10.1038/s41467-024-45965-x`.

[30] Z. Wang and A. C. Bovik. “Mean squared error: love it or leave it? a new look at signal fidelity measures”. *IEEE signal processing magazine* **26**:1 (2009), pp. 98–117.

[31] Z. Wang, J. Chen, and H. Chen. “Egat: edge-featured graph attention network”. In: I. Farkaš et al. (Eds.). *Artificial Neural Networks and Machine Learning – ICANN 2021*. Springer International Publishing, Cham, 2021, pp. 253–264. ISBN: 978-3-030-86362-3.

[32] X. Yang, G. Wang, W. Hu, J. Gao, S. Lin, L. Li, K. Gao, and Y. Wang. “Video tiny-object detection guided by the spatial-temporal motion information”. In: *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2023, pp. 3054–3063. DOI: `10.1109/CVPRW59228.2023.00307`.

[33] G. Yehudai, E. Fetaya, E. Meirom, G. Chechik, and H. Maron. “From local structures to size generalization in graph neural networks”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 11975–11986.

[34] L. Yu, R. Zhou, R. Chen, and K. K. Lai. “Missing data preprocessing in credit classification: one-hot encoding or imputation?” *Emerging Markets Finance and Trade* **58**:2 (2022), pp. 472–482.

[35] Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang. “Random erasing data augmentation”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 34. 07. 2020, pp. 13001–13008.

[36] H. Zhu, H. Wei, B. Li, X. Yuan, and N. Kehtarnavaz. “A review of video object detection: datasets, metrics and methods”. *Applied Sciences* **10**:21 (2020), p. 7834.

[37] X. Zhu, Y. Wang, J. Dai, L. Yuan, and Y. Wei. “Flow-guided feature aggregation for video object detection”. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 408–417. DOI: `10.1109/ICCV.2017.52`.

A

Training Details

This Appendix features some of the training details for the models presented in Chapter 4.

	Pos	**Velo**	**Prev Frame**	**Seq**
Batch Size	150	100	100	10
Learning Rate	0.001	0.0001	0.0001	0.0001
Dropout Rate	0.2	0.2	0.2	0.2
Training GPU	NVIDIA T4 [18]	NVIDIA T4	NVIDIA T4	NVIDIA A100 [17]
Training Time	5h 18min	8h 53min	7h 40min	12h 17min

Table A.1 Training details for the models presented in Chapter 4.

B

Additional Performance Metrics

This Appendix features additional performance metrics for the models presented in Chapter 4. Firstly, a table presenting the median prediction errors for the models and different scenarios and then prediction error histograms for the velocity model for the different scenarios. The prediction error histograms for the baseline model is also presented in these figures.

Category	Tag	Baseline	Pos	Velo	Prev Frame	Seq
Open Play	General-Play	21.9	13.9	12.1	**10.8**	17.3
	Shot	22.3	16.0	**12.3**	13.8	19.0
	Goalkeeper-Throw	26.4	**11.6**	14.4	19.0	20.7
	Cross	28.3	12.1	11.5	**10.3**	16.7
	Goalkeeper-Punt	34.5	22.8	24.5	**14.9**	23.4
Set Piece	Free-Kick	19.8	**13.0**	13.7	14.1	14.1
	Kick-Off	**3.3**	6.7	6.0	8.7	9.6
	Throw-In	17.2	12.8	**6.3**	8.9	8.8
	Corner	30.2	18.0	**17.0**	21.0	28.4
	Penalty	12.9	16.5	28.6	**12.0**	23.1
	Total	20.6	13.2	**11.7**	12.1	16.2

Table B.1 Median prediction errors (m) for each tag for the four models and the baseline prediction. The lowest mean error for each tag is typed in boldface font.

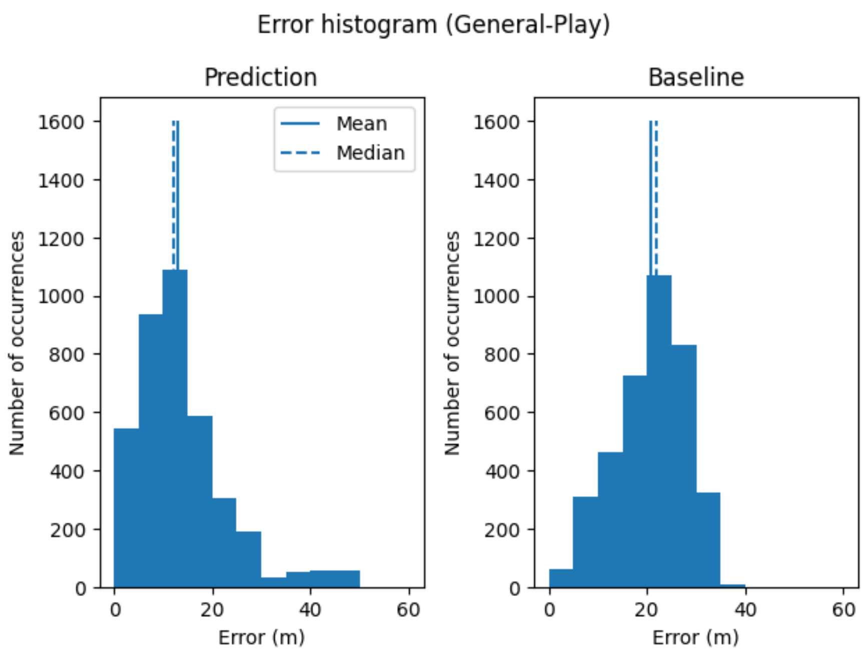


Figure B.1 Prediction error histogram for "Velo" model (General-Play)

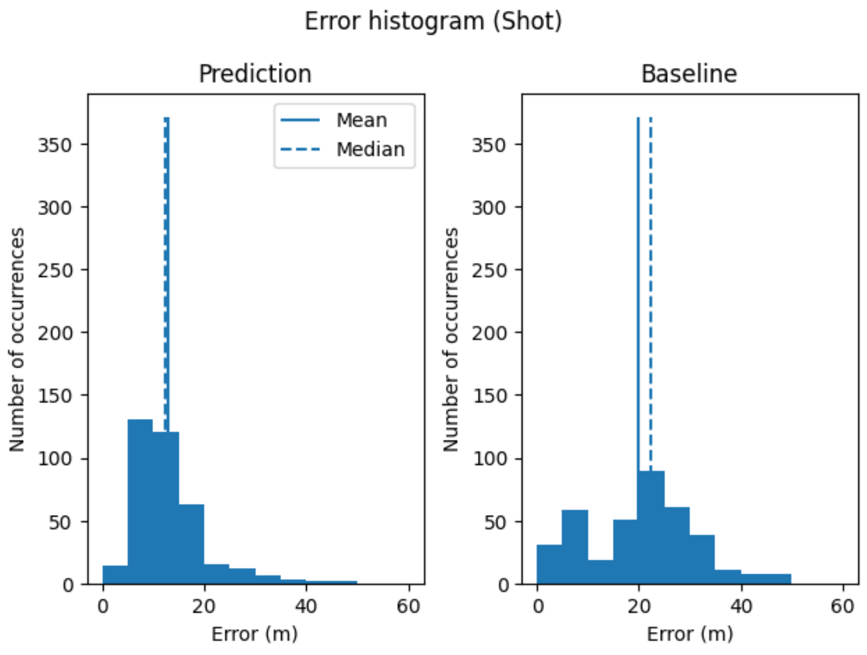


Figure B.2 Prediction error histogram for "Velo" model (Shot)

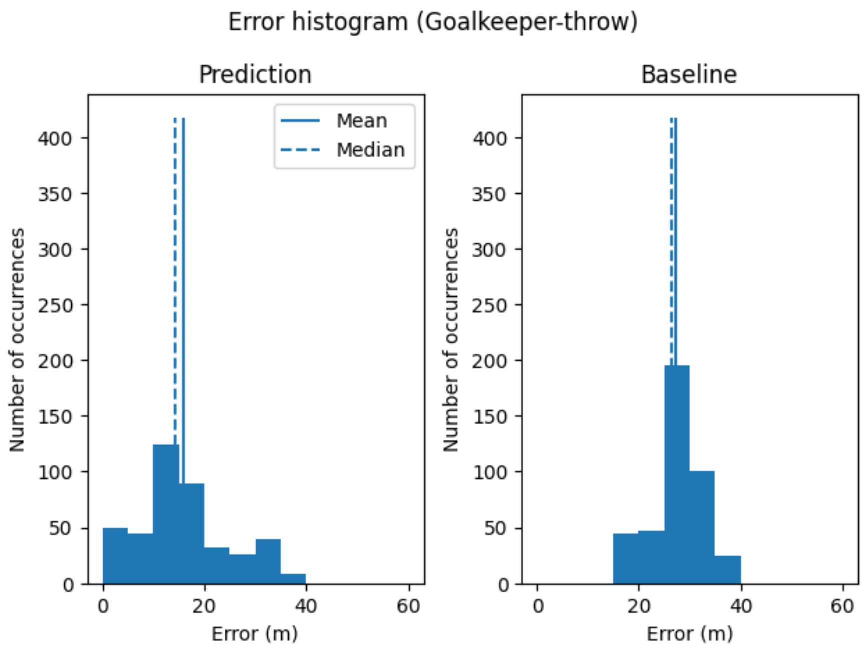


Figure B.3 Prediction error histogram for "Velo" model (Goalkeeper-Throw)

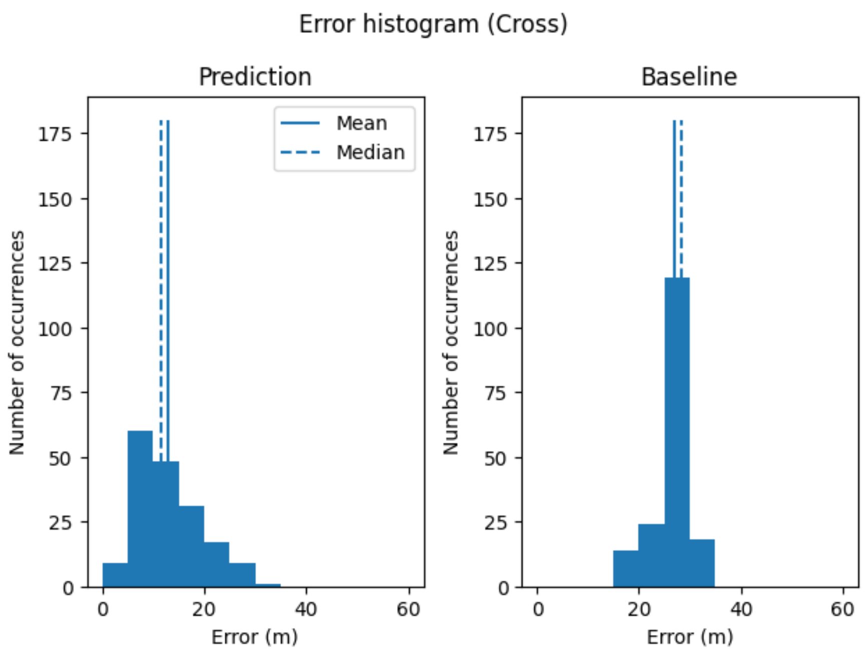


Figure B.4 Prediction error histogram for "Velo" model (Cross)

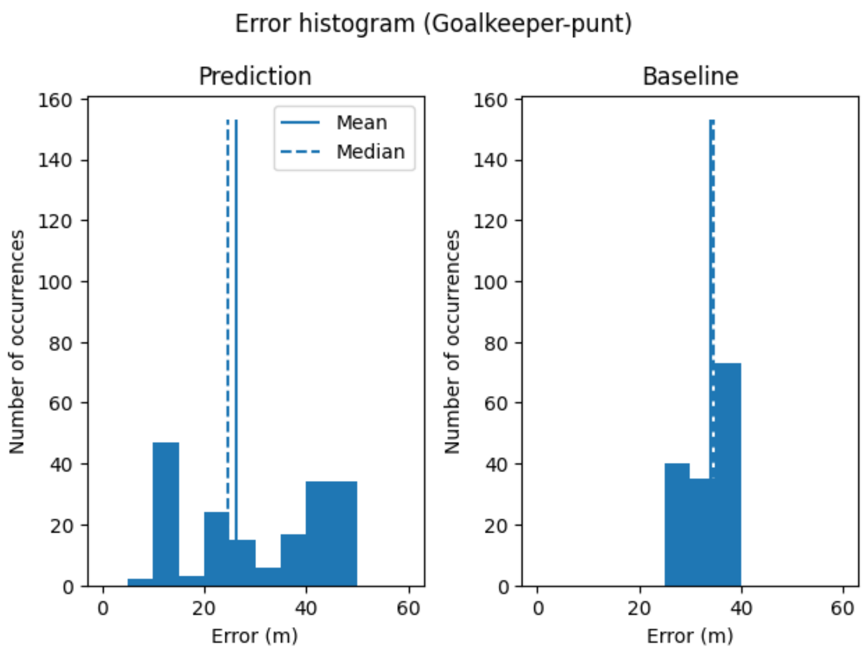


Figure B.5 Prediction error histogram for "Velo" model (Goalkeeper-Punt)

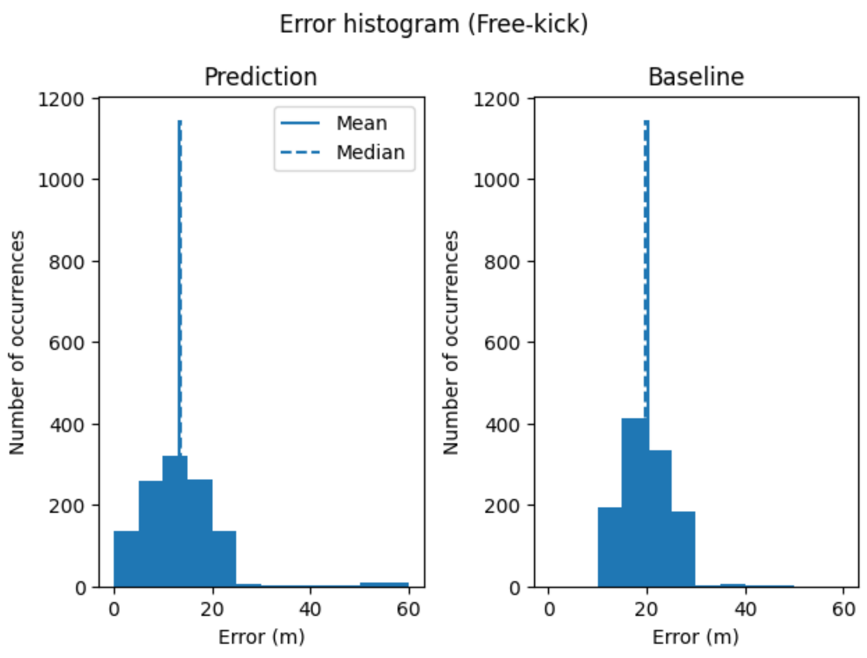


Figure B.6 Prediction error histogram for "Velo" model (Free-Kick)

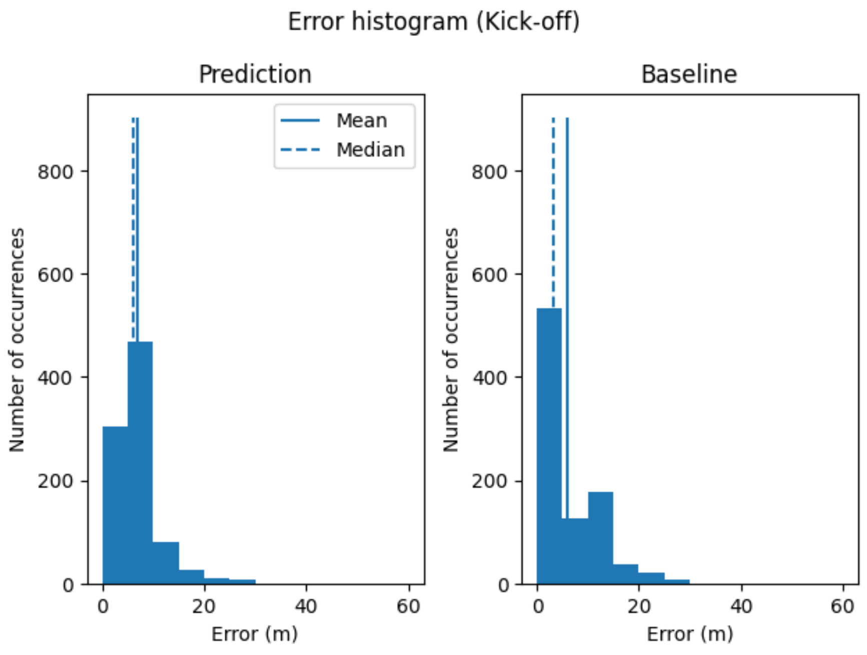


Figure B.7 Prediction error histogram for "Velo" model (Kick-Off)

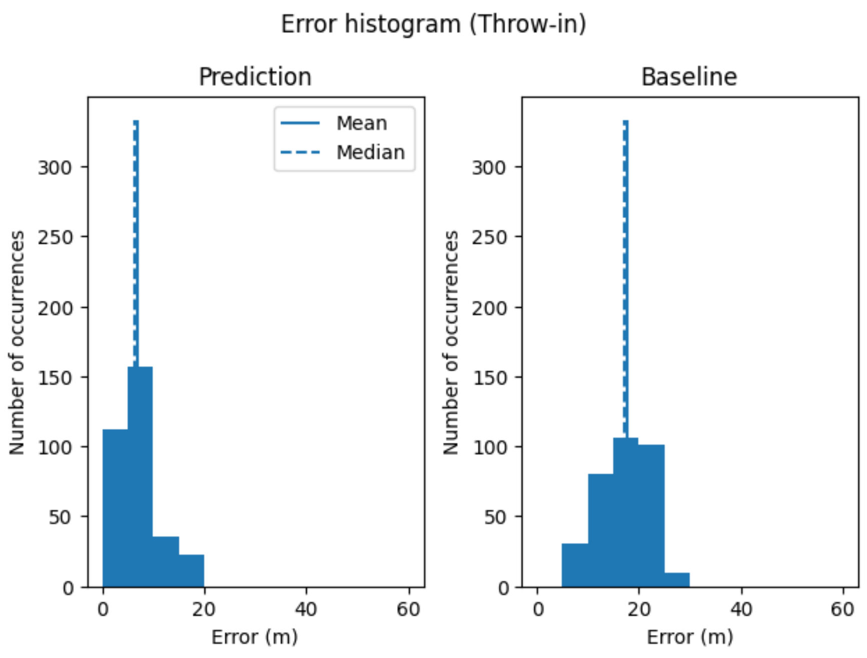


Figure B.8 Prediction error histogram for "Velo" model (Throw-In)

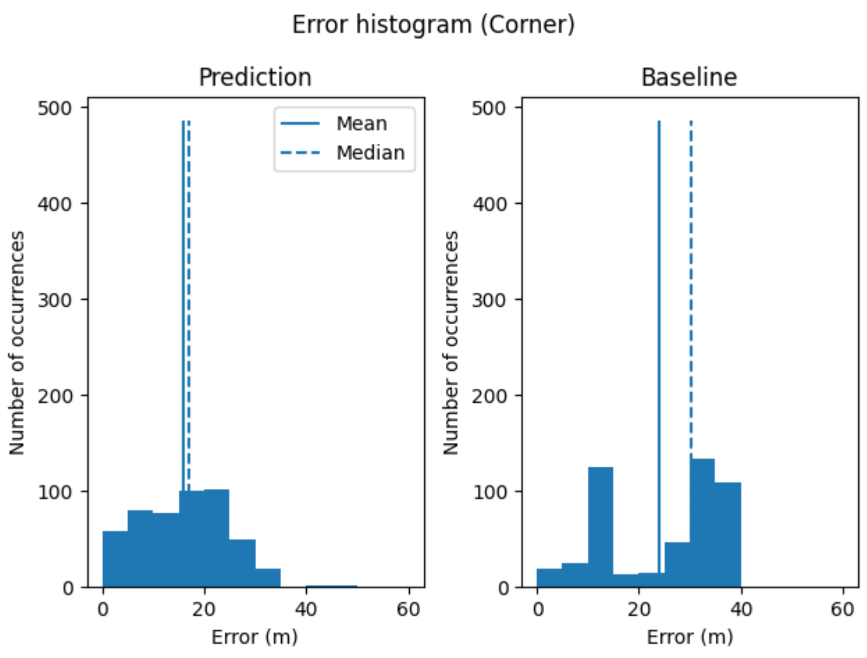


Figure B.9 Prediction error histogram for "Velo" model (Corner)

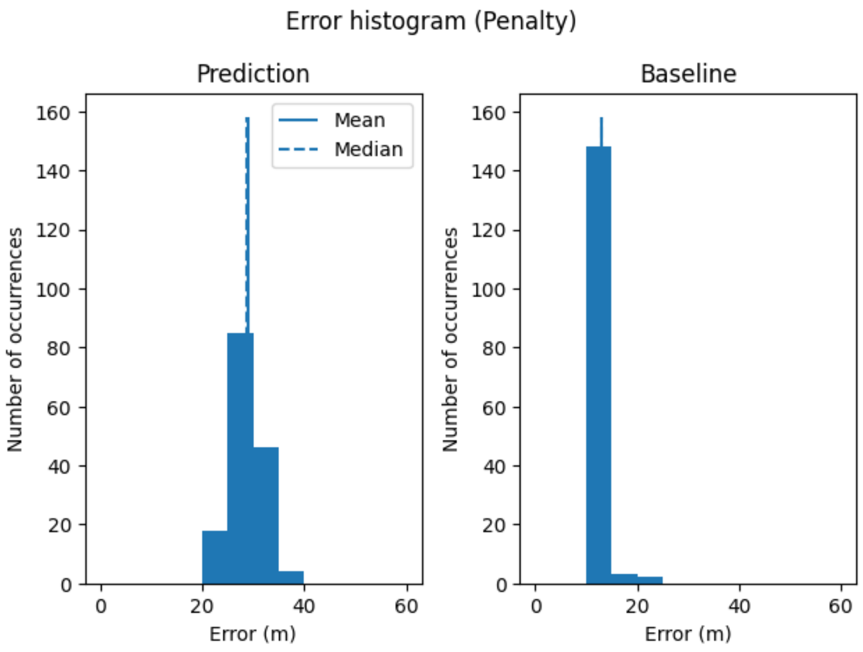


Figure B.10 Prediction error histogram for "Velo" model (Penalty)

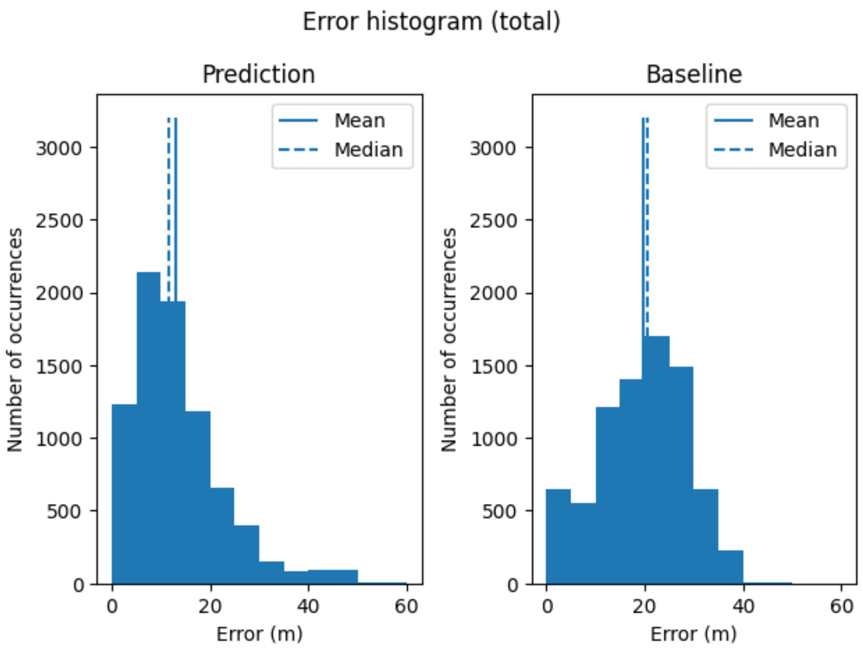


Figure B.11 Prediction error histogram for "Velo" model (Total)