Connecting the unseen dots

Combining machine learning models to better predict off-screen events in soccer broadcasts

Håkon Hernes & Brynjard Buvarp Misvær

Thesis submitted for the degree of
Master in Programming and System Architecture
60 credits

Department for Informatics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Spring 2022

Connecting the unseen dots

Combining machine learning models to better predict off-screen events in soccer broadcasts

Håkon Hernes & Brynjard Buvarp Misvær

Connecting the unseen dots

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Abstract

Soccer games are rich with events that form the basis for creating highlights, game analysis, player and team statistics, game commentary, and more. Such events needs to be annotated. At present, stakeholders spend a large amount of time and money to do this manually. In recent years, machine learning models have shown promise in this regard, potentially being able to fully automate this task in the future, without human intervention. However, most models have a difficulty spotting events that are happening off-screen, i.e., interesting events that are not shown in the video broadcast, because of some choice made by the broadcast producers. An example of this can be a *Kick-off* not being shown after a *Goal* because of replays.

In this thesis, we present a new idea combining three machine learning models to better spot off-screen events in soccer broadcasts. We train the different machine learning models on what we define as three different dimensions of context: A *Past* dimension, where a model can use an event happening in the present to inform what happened in the past; a *Present* dimension, where a model is able to recognise the event happening in the present; and a *Future* dimension, where a model can use the event happening in the present to inform what will happen in the near future. To do this, we will leverage the rules of soccer to define relationships between events, alter the dataset that we are evaluating our models on, and finally combine these models in a data fusion layer.

We evaluate these models on the SoccerNet-v2 dataset, and compare it to an existing action spotting model, to see how our model compares when it comes to the task of action spotting, focusing specifically on spotting off-screen events. We also define two real-life use cases where such an action spotting model might be applied, and compare the results to the existing action spotting model.

Our results show that our best performing model improves the off-screen spotting for some types of events, and that our model is able to produce a higher recall than the existing model, when applied to a use case of generating statistics from soccer games.

Acknowledgements

We would like to thank our supervisors, Pål and Michael, for the feedback and support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background and motivation

In sports, soccer leagues are among the most valuable in the world. The European soccer market had an estimated total revenue of 28.9 billion euros in 2018/2019 [31], while the 20 largest clubs generated a combined revenue of 8.2 billion euros in the 2019/2020 season [1].

Soccer games contains events that both professional actors and fans have an interest in. Events form the basis for creating highlights, analysis, and game and player statistics. Extracting information from the games is both time-consuming and expensive, since it is a manual operation performed by humans.

In recent years, efforts have been made in trying to automate the annotation of sport videos using event detection. A well working system for this task would greatly decrease both the time and cost of producing such annotations.

In 2018, SoccerNet [18] was released, which offered a highly scalable dataset with 764 hours of video, and 6,637 annotated events distributed across the three classes *Goals*, *Cards*, and *Substitutions*. The authors of this paper presented two tasks to further stimulate research in the field; classifying events in one-minute trimmed videos, and spotting events throughout a game. SoccerNet also defined a generalised evaluation metric, called the Average-mAP, which produces a single number to represent the performance of a machine learning model evaluated on the dataset. SoccerNet provided a baseline for the two tasks, with an Average-mAP score of 49.7% for event spotting, and an Average-mAP of 67.8% for classification.

Following this, a convolutional neural network (CNN) was presented by Rongved et al. [35, 38], that increased the Average-mAP for classification on trimmed video clips to 88.4%. Improvements were also made to the baseline for event spotting. A context-aware loss-function proposed by Cioppa et al. [12], managed to increase the Average-mAP

to 62.5%. This model leveraged the temporal distance between a potential spotting to its ground truth when penalising the model during training. Efforts have also been made to combine audio and video streams [49] to improve upon the baselines set by SoccerNet.

In late 2020, a new version of SoccerNet, SoccerNet-v2 [11], was released. SoccerNet-v2 extended the amount of annotations to ~300,000, expanding upon the event spotting classes as well as introducing classes related to video production. Giancola & Ghanem [19] developed a feature pooling method, NetVLAD++, that improved upon the baseline of the event spotting (called action spotting in SoccerNet-v2) task by 12.7 percentage points, to reach an Average-mAP of 53.4%. The winner of the action spotting contest [16], Baidu research [54], managed to achieve an Average-mAP of 74.84%, by fine-tuning feature extractors on soccer clips.

In soccer broadcasts, events can happen off-screen, as a result of replays or other choices made by the broadcast producers. These off-screen events happen in the game, but are not shown in the video stream broadcast to the viewer. An example may be a *Kick-off* event after a *Goal*, which has not been shown due to replays of the *Goal* itself. Common for most efforts working with SoccerNet-v2, is that the models perform worse when spotting off-screen events. Our observation is also that very little research focuses explicitly on off-screen action spotting. For most real-life scenarios where we need to annotate events in soccer, it is important to capture these off-screen events. One could argue that the most important events in soccer are most often shown on-screen, since it is in the interest of the broadcast producers to show these events. For a statistical system however, it is also important to capture the off-screen events. We suspect that the challenge of predicting off-screen events can be generalised to many other computer vision applications, where the events that needs to be captured require a deeper understanding than the visual features alone can provide. An exciting prospect is therefore to consider how one can improve a machine learning models ability to spot off-screen events in soccer broadcasts.

1.2 Problem statement

Today, machine learning models focused on annotating soccer events perform worse when annotating events that happen off-screen, and little research is focused on off-screen action spotting. This thesis aims to explore one possible way for a machine learning model to better be able to predict off-screen events.

Recent advances in deep-learning based action recognition has shown that we are able to develop models that can correctly classify

events that are happening on-screen quite well. In soccer, it is easy for humans to understand what has happened off-screen, because we have a strong sense of context throughout the game. This seems to be more challenging for machine learning models, as context is not necessarily informed by visual features alone. This thesis aims to explore the following question:

Can we leverage the strictly defined rules of soccer to develop a machine learning model that better understands the context of the game and perform better when predicting off-screen events?

Based on this question, we identify the following objectives for our thesis:

Objective 1 Research and develop an approach that leverages the rules of soccer to better spot off-screen events in soccer broadcasts.

Objective 2 Implement a model that follows the selected approach.

Objective 3 Analyse the performance of our model, and compare it to an existing SoccerNet-v2 action spotting model.

Objective 4 Analyse the applicability of our model to practical use cases, and compare the results to that of an existing action spotting model.

1.3 Scope and limitations

Within the domain of action recognition in soccer broadcasts, it is challenging for machine learning models to have the same sense of context as humans have. Training a machine learning model also requires large amounts of training data. In this thesis we have limited our dataset to SoccerNet-v2, with its 764 hours of soccer clips. We have also limited the events we are considering to the 17 different classes of events annotated for the action spotting task, which are detailed further in Section 3.2. We have based all our implemented models on NetVLAD++ [19], which is available through the SoccerNet-v2 development kit (devkit). For our experiments, our models needs to convert the raw video clips into features. We have used the pre-extracted features made available through the SoccerNet-v2 devkit. In this thesis, we only consider the task of action spotting, defined in SoccerNet-v2 [11] as the task of temporally grounding an event in a soccer clip, and classifying which event it is.

During the development of our models, we have used the Average-mAP evaluation function provided by the SoccerNet-v2 devkit. This evaluation function provides a score for the performance overall (spotting both visible and off-screen events), as well as visible only, and off-screen only. Later on, we compare our best performing model to the baseline using the F1 score, precision and recall evaluation metrics. Here we will also consider the three categories of visibility: all, visible and off-screen. Although we discuss all three categories in our work, the most important is the off-screen metric, which reflects how the models perform when spotting off-screen actions.

In this thesis we only consider the action spotting performance of machine learning models. We have therefore not done any system benchmarks, or made any significant efforts to optimise hardware utilisation.

1.4 Research methods

In this thesis, we use the research methodology proposed by ACM [14], which include three paradigms when it comes to computing as a research discipline. Each paradigm has a set of steps that should be followed when conducting research, and the steps should be iterated when necessary. The individual paradigms might also include elements of the other paradigms, and are therefore not completely independent of each other.

Theory The theory paradigm is rooted in mathematics and follows four steps; definition, theorem, proof and interpretation of results. The definition is the characterisation of the object of study, and the theorem step includes hypothesising possible relationships between these objects. In the proof step, the relationships are determined to be true or false and finally the results are interpreted.

Abstraction Abstraction, or modelling, is rooted in the experimental scientific method and contains four stages. The first is to form a hypothesis, then create a model in order to make a prediction. The next stage is to design an experiment and collect data, and finally analyse the results.

Design The design paradigm is rooted in engineering and has four steps for setting up a system to solve a problem. First, the requirements should be stated, then the specifications should be stated. The next step is to design and implement the system and before finally testing the system.

Our thesis uses a combination of these paradigms. We develop experimental prototypes of a system and conduct our experiments in an iterative fashion. Each experiment begins with forming a hypothesis in accordance with the abstraction paradigm. Then, we design and implement the system before testing it, in accordance with the design paradigm. Finally, we analyse the results, conforming again to the abstraction paradigm. At the next step, when we are forming our next hypothesis, we try to include insights gained from the previous iterations. We will build upon hypothesis' and models which yield good results and seem interesting for further research. Elements of the theory paradigm such as definition are also applied, when describing theoretical concepts from machine learning, and when formally defining phenomena in the dataset and our machine learning models.

1.5 Main contributions

As we discussed in Section 1.2, we will leverage the rules of soccer to create a machine learning model that better predicts off-screen events in soccer broadcasts. Throughout the work of this thesis, we present the following main contributions:

- We define an intuition for how we can introduce a sense of context into a machine learning model, for it to better be able to spot off-screen events in a soccer broadcast. We identify relationships between events in soccer, and describe how these can be used as the basis for creating three dimensions of context in soccer. From this, we discuss how the dataset can be manipulated to train three separate models that reflect the different dimensions of context. We design a new model architecture that trains and combines a *Past*, *Present* and *Future* model to better predict off-screen events in soccer.

- We iteratively use the existing SoccerNet-v2 labels to create new labels that are used to train the *Past* and *Future* models. Furthermore, we create a data fusion layer that combines the output of the aforementioned models to create one final model output. In order to combine these models in the best way, we experiment with data fusion algorithms, and implement sliding window algorithms with different window sizes.

- We evaluate our model using the existing Average-mAP evaluation function, and compare the results to a set baseline model. Our best performing model achieves a higher performance when

spotting off-screen events for four of the 17 classes in SoccerNet-v2.

- We define two practical use cases for action spotting models within the soccer domain, and use precision, recall and F1 score as evaluation metrics. When used for statistics, our model achieves a higher recall score. When used for game commentary, we only considered action spotting on off-screen events. Our results showed that our model achieved a higher precision for two classes, and a higher recall for three classes compared to the baseline.

These contributions address the question and objectives in our problem statement, and presents new ideas for how one can better spot off-screen events in soccer broadcasts. Our results show promise, hopefully facilitating further development of our combined machine learning model architecture. Furthermore, we think that our work explores an exciting challenge in computer vision as a whole, which is understanding how a machine learning model can understand the context within the domain it is applied to. Our code is available here: `https://github.com/Brynjard/Masterthesis`

1.6 Thesis outline

Chapter 2 - Background In this chapter, we introduce theoretical concepts and definitions required to understand our work throughout the thesis. We present related works within the field of activity recognition and action spotting, and provide a context for the work that has contributed to the current state-of-the-art models within the field. We divide these into two sections, focusing first on action spotting in general, and then on action spotting off-screen events. Towards the end of this chapter we provide an overview of datasets that have facilitated progression in the field.

Chapter 3 - Methodology In our methodology chapter, we first introduce the SoccerNet-v2 dataset. Then we introduce our intuition behind how context in soccer is formed. We also define relationships between interesting events in soccer, and present our ideas for how these relationships can be leveraged to introduce context into our models. We present our baseline model, and choices we have made in terms of hyperparameters, and data fusion techniques. Finally, we detail how we have evaluated our models.

Chapter 4 - Experiments & Results In this chapter, we present our

experiments and results. We compare our results to the baseline using two different evaluation methods, and compare them in light of their applicability to real-life use cases. We also analyse our results, and draw possible conclusions in terms of the behaviour of our models. Finally, we reflect on our work and discuss some shortcomings and choices we have made throughout this thesis.

Chapter 5 - Conclusions In the conclusions chapter, we summarise our work, outline our main contributions, and discuss ideas for future work building on this thesis.

Chapter 2

Background

2.1 Introduction

In this chapter, we will define relevant theory, related literature and terminology used as background knowledge for this thesis. We start by defining machine learning and the two most relevant paradigms for this thesis. We will also define classification as a general machine learning problem. Furthermore, we define neural networks and other technical concepts relevant for video processing in machine learning. As our topic is related to action spotting in soccer videos, we define relevant terms such as events, action spotting and event visibility. We also present a definition for what context in soccer is. We present current related work in the field of event detection and action recognition, and finally provide an overview of datasets for video understanding in general, and for the soccer domain. Towards the end of this chapter, we address ethical considerations related to our work.

2.2 Machine Learning

Machine learning can be defined as computer systems and algorithms that learn and improve from experience. Mohri et al. [34] defines experience in this context, as past information available to the system. A key concept in machine learning is that the systems learning process is not explicitly programmed by the designer of the system. Recent years have seen new applications for machine learning surface at an increasing rate. A major contribution to this revolution in machine learning applications is the increase in, and availability of computing power. The growing amount of data available in the world [32] has also helped research and the success of machine learning applications. A machine learning system can often be categorised under one of three paradigms: Supervised learning, unsupervised learning, and reinforcement learning. In the field of video event detection, most

research effort has been focused on supervised and unsupervised learning.

2.2.1 Supervised Learning

Supervised learning is a learning paradigm in which the model is provided with a set of observations, and each observations correct target. The system then tries to generalise on this data to provide the correct target for new, unseen data. In other words, given a set of unseen data-points $\{x_1,\ldots,x_n\}$ and a set of targets $\{y_1,\ldots,y_m\}$ the goal of the model is to correctly map each x_i to their correct target, y_i.

2.2.2 Unsupervised Learning

In unsupervised learning there are no targets provided with the observations. It is purely up to the system to cluster or categorise the data in different groups, based on common traits and patterns found in the data by the model.

2.3 Classification

Classification is a set of problems in which the main goal is to correctly map a set of observations to their correct class or classes. An important aspect of a classification problem is that it is discrete. The complete set of possible classes define all possibilities of the solution, and each input observation evaluates to one class or a combination of classes [32]. A classical classification problem is that of image recognition. In image recognition, an image is provided as an input, and the goal of the model is to correctly map this image to one, or several classes depending on the task. The model would initially be fed with a (preferably) large amount of images with their correct classes attached. After a certain training period, the model should then be able to map (with a certain error-margin) new images to their correct class. One attempt from the model to map an input to a class is usually called a prediction.

2.4 Neural networks

In the context of this thesis, we limit the scope of neural networks and deep learning to supervised learning. In a neural network, each data-point (input) is represented as a set of features $\{x_1,\ldots,x_n\}$. For any given task, a pre-processing step is required, in which the input data is transformed into features. For image classification, this might be to represent each pixel in the image as a real number. In Figure 2.1, each

left-most node represents a feature. Each feature is a real-numbered value representing some attribute of the input. The goal of the model, is to correctly predict a correct target, t based on a combination of features. A neural network consists of three or more layers of connected nodes, where the first layer, called the input layer, are the features representing the data-point. The last layer is the target-space, which can be one node, or several, depending on the problem we are trying to solve. In the case of Figure 2.1, each prediction will be one of two possibilities. In between these layers are one or more layers of

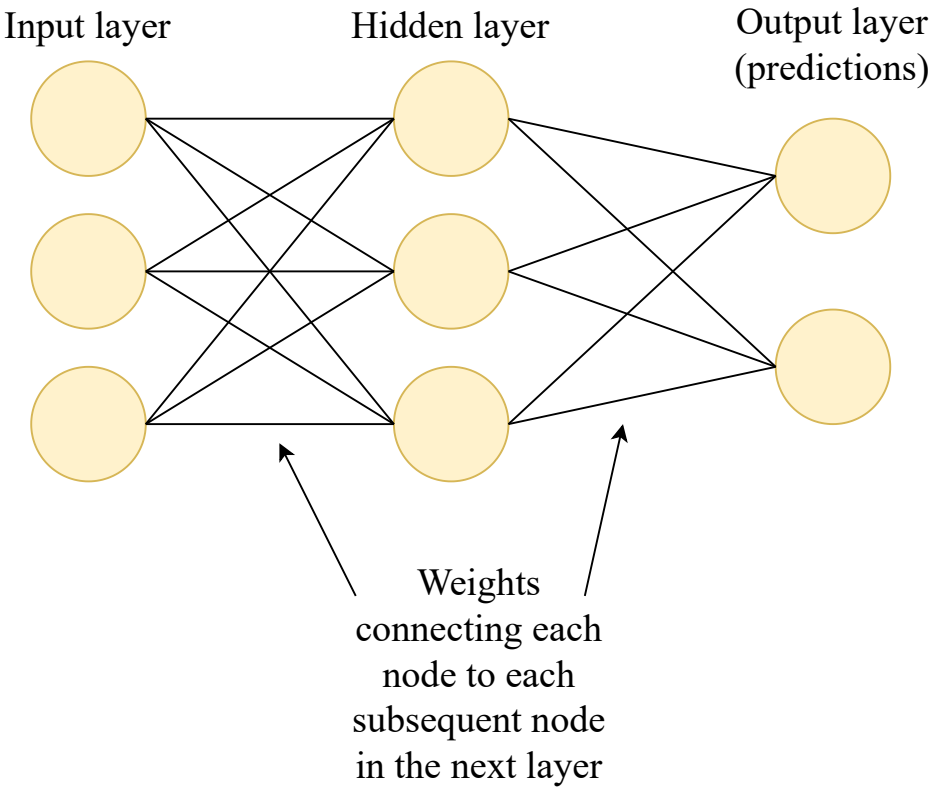


Figure 2.1: A simple neural network architecture for a classification problem. The network consists of three input nodes representing three features for one datapoint, three nodes in a hidden layer, and two nodes in the output layer.

hidden nodes. Each layer starting with the input nodes, are fully connected to the next layer through weights, which are real numbers that are initialised at the start of the training phase. In Figure 2.1, each node in the hidden layer has three weights connecting it to all nodes in the previous layer, and two weights connecting it to the next layer. The network *feeds forward* the values from one layer to the next, through a dot product function of each preceding node and the corresponding weight, to the current node. For a given node in the hidden layer in Figure 2.1, this fed forward value is the sum of three

products. In this figure, each feed-forward step is a left-to-right process over each consecutive layer. For each node in each layer, the forward fed value from the previous layer is processed through an activation function, before moving on to the next layer. This continues until the last layer, and we get the prediction from our model. An error-function then calculates the cost of our prediction (how wrong we were) and propagates that error backwards, by updating the weights of each layer. This continues through a number of iterations or epochs, or until some other exit criteria is met. This is how the neural network learns.

Consider the following neural network based on Figure 2.1:
The left-most input layer with features $\{x_1,\ldots,x_n\}$, one hidden layer of nodes $\{z_1,\ldots,z_m\}$, and one layer of output nodes, $\{y_1,\ldots,y_o\}$. The weights connecting the input layer and the hidden layer is a matrix, v, of size $n \times m$. The weights connecting the hidden layer to the output layer, w, is a matrix of size $m \times o$. In the feed-forward phase of the algorithm, we calculate the *activation* for each node in the hidden layer, as well as for the output-layer. For any given node z_j in the hidden layer, we calculate the dot product of each node in the preceding layer with the corresponding weights as such:

$$z_j = \sum_{i=1}^{n=3} x_i v_{i_j}$$

This value is then processed by an activation function g, to give the activation of z_j: $g(z_j)$. This process is repeated between the hidden layer and the output layer, resulting in a vector of activations $\{y_1,\ldots,y_o\}$, which represent the predictions made by our models. This vector can take many forms, depending on the problem we are trying to solve. In this case, our vector has a length of two.

In the backwards-phase of the algorithm, our predictions y, is first compared to a target-vector $\{t_1,\ldots,t_o\}$. Depending on the problem at hand, an error function, $L(y,t)$ calculates the error-rate of our predictions, which sets the basis for how our weights are adjusted. Marshland [32] defines one such error function, the sum-of-squares function, as such:

$$L(y,t) = \frac{1}{2}\sum_{k=1}^{N}(y_k - t_k)^2$$

The backwards-phase of the algorithm, consists of computing the gradients of the error in respect to the weights, so that we can adjust the weights as to minimise the error.

2.5 Deep learning

Deep learning describes a subset of machine learning algorithms that utilises several hidden layers in a model (like a neural network) that can be fine-tuned to optimise its problem-solving capabilities. For each layer in the deep learning model, the input from the previous layers build upon each other and become more and more sophisticated [3].

2.6 Convolutional neural networks

In recent years, a subset of neural networks, called Convolutional Neural Networks (CNN) has been used with much success. In the realm of image and video detection, CNNs have showed huge improvements compared to other models [2]. A CNN can be implemented through various architectures. Features can either be extracted by so-called feature engineering, or they be can be learned by a module of the model architecture.

One integral part of the CNN are the convolutional layers. As seen in Figure 2.2, a 2 x 2 filter, K, slides over the input, X, from left to right, top to bottom. Each step outputs the dot-product between K and the submatrix of X, which is one value in the featured matrix, M. The purpose of this layer is to extract the relevant features from the input. CNNs also utilise parameter sharing, which decreases the number of calculations in the convolutional layer. With parameter-sharing, each node in the convolutional layer uses the same weights. In Figure 2.2, K is shared amongst all nodes in M.
At different layers in the CNN, a non-linearity function is normally applied to the featured matrix. This is often implemented as the ReLU-function:

$$ReLU(x) = \begin{cases} 0, & \text{if } x < 0 \\ x, & \text{otherwise} \end{cases}$$

Pooling is another important part of the CNN architecture. As can be seen in Figure 2.3, pooling reduces the amount of features in the matrix. In this case, a 2 x 2 filter slides over the matrix, disregarding all values but the largest in its current scope, effectively reducing four values to one. This is called max-pooling. Pooling reduces the complexity and amount of calculations that our CNN needs to process. There are other variations of pooling as well, such average-pooling, which averages the values in its current scope.

The final building block of the CNN is the fully connected layer. This layer is fully connected to each activation in both the previous and the next layer, much like in a traditional neural network. Different CNN architectures comprise of different combinations of these layers.

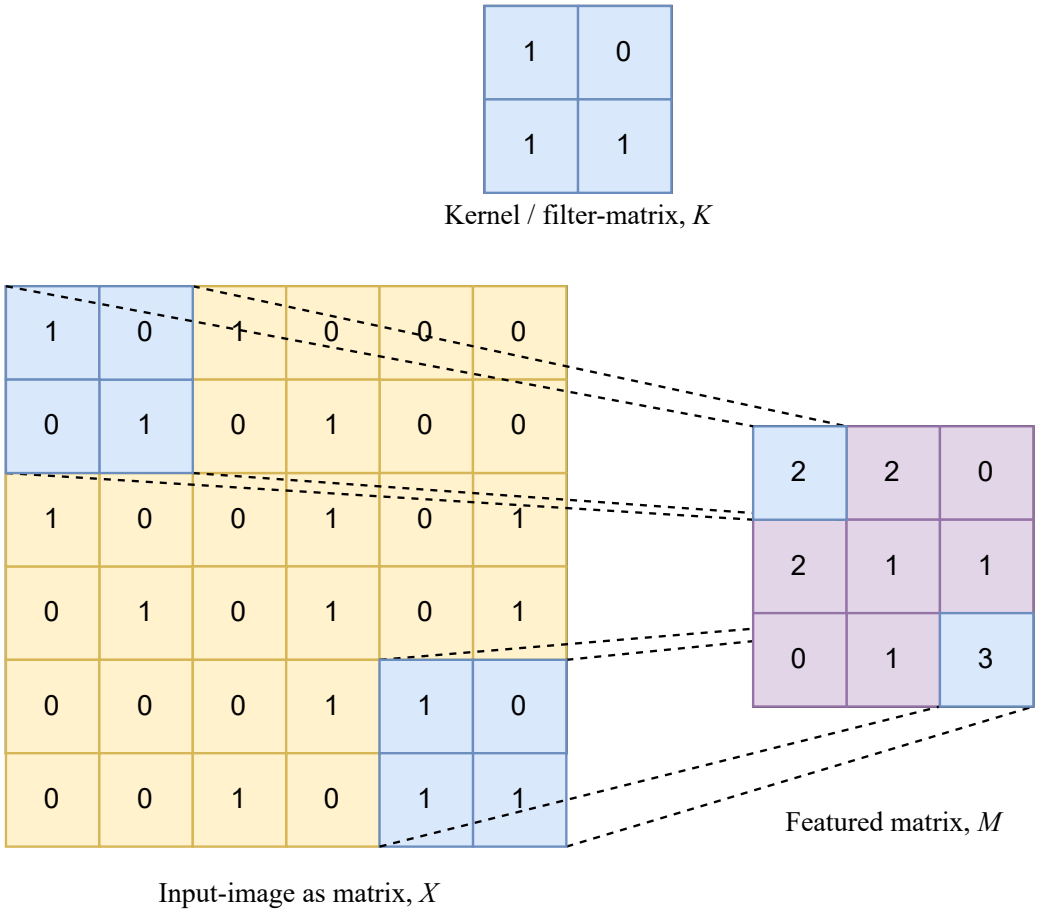
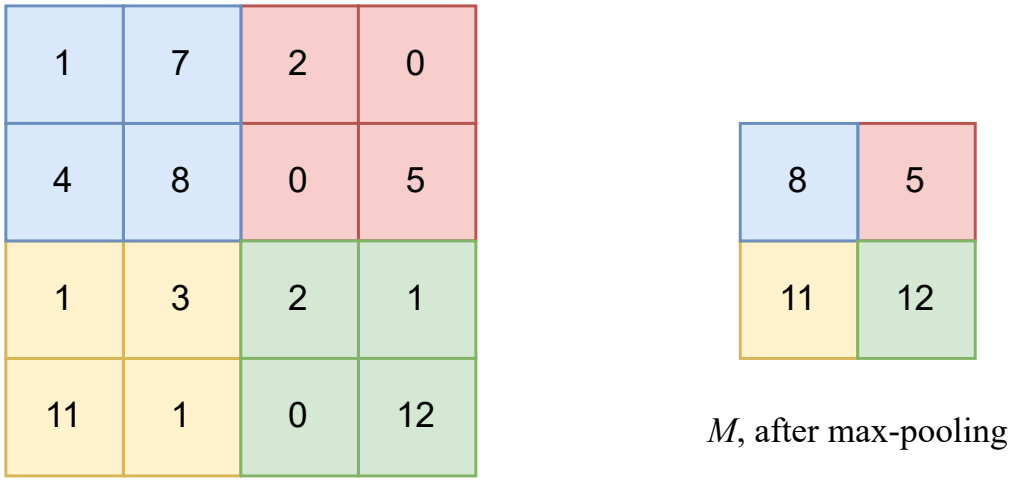


Figure 2.2: Illustrating how a 2 x 2 filter, K, *convolves* over an input-matrix X in a CNN, resulting in the featured matrix, M. For simplicity's sake, this convolutional layer uses a stride of 2, although a stride of 1 is more normal.

2.7 Principal Component Analysis

Most applications and experiments with machine learning models require large amounts of data. In the context of video analysis, the features we are working with are often large in number of dimensions, resulting in a large amount of data needing to be processed and calculated when training a model. The curse of dimensionality [32] also states that as the number of dimensions grow, the size of the training data also needs to grow. The result of this is that the machine learning model needs a lot of time and space in terms of computing power to undergo sufficient training. Principal component analysis (PCA) is an algorithm that transforms the data into a lower dimensional representation, while retaining as much information in the data as possible [32].



Featured matrix, M

M, after max-pooling

Figure 2.3: Max-pooling on a 4 x 4 feature matrix. With a filter-size of 2x2 and a stride of 2, this results in a 2 x 2 matrix.

2.8 Computer vision

Computer vision is related to making computer models that are able to see and gain an understanding of the visual world, by processing sensor data from images and videos. In the early days of artificial intelligence research, tasks involved with object recognition in images were seen as trivial, compared to tasks related to higher reasoning and planning [46]. These tasks have later proven to be quite challenging, with a large portion of the related problems not yet solved with satisfactory results. In soccer, computer vision has multiple applications, from object recognition to game understanding. Use of object recognition in soccer broadcasts includes player localisation and tracking, ball tracking and field lines localisation [47, 21].

2.9 Non-maximum suppression

In computer vision tasks in general, and for object detection or event recognition specifically, a common challenge is that the machine learning model outputs predictions and bounding boxes that are overlapping. In the context of object detection, this overlap can be defined as the intersect-over-union between bounding boxes, and in event recognition it can be defined as several predictions of the same class being within a set time window. Non-maximum suppression is an algorithm which filters these predictions/bounding boxes, and outputs the ones that are of most likely to be correct. This algorithm works

under the assumption that predictions or bounding boxes that are overlapping, are most likely predicting the same observation.

2.10 Event definition

In an attempt to suitably define an action in the context of computer vision, Chen et al. [10] argues that an action contains four aspects: an agent, an intention, a bodily movement, and one or more side-effects. Defining the start and end of an action is not completely unproblematic - Sigurdsson et al. [42] showed how temporally localising activities is an ambiguous task. By re-annotating the Charades [41] and MultiTHUMOS [52] datasets, they found that their agreement with the ground truth were only 72.5% and 58.7% *timeIoU* respectively. Conveniently, soccer has a strict and defined rule-set for events such as *Goal*, which is the exact moment when the ball crosses the goal-line. This lets us adhere to SoccerNet's [18] definition of an event, as an action temporally anchored with a single timestamp.

2.11 Action spotting

Introduced with the release of SoccerNet [18], *event spotting* is the task of temporally anchoring an event in an untrimmed video. Given a temporal tolerance, δ, a candidate for spotting is considered correct, if it is within the δ of the ground truth. Event spotting is therefore the task of classifying what event is taking place, as well as when. In SoccerNet-v2 [11], this task is referred to as *action spotting*. In this thesis, we will use action spotting, but use the terms action and event interchangeably when talking about events as defined in Section 2.10. We will also use the terms action spotting and spotting interchangeably.

2.12 Event visibility

In soccer broadcasts, events of interest can happen off-screen, as a result of the choices made by the broadcast producers. Usually, this is the result of replays. After a *Goal* is scored, the producers might choose to show several replays, before the broadcast returns back to showing the game in real-time. After these replays, the game might have started. Between the *Goal* and the end of the replays, a *Kick-off* might have occurred that was not shown in the broadcast. The *Kick-off* in this instance, has happened off-screen. Throughout a soccer game, several events will happen off-screen as a consequence of the broadcast producers choices.

In the SoccerNet-v2 dataset, each annotated event has an attribute called *visibility*. This is set to either *visible* or *not shown*. *Visible* events describe events that are shown in the broadcast, while *not shown* events are not shown in the broadcast. We prefer to refer to use the term off-screen events when referring to *not shown* events. In this thesis we will therefore use off-screen to describe *not shown* events, except for places where we are directly referring to the value of the *visibility* attribute of an annotation in the dataset.

2.13 Context in soccer

Context is a relative term. We feel that it is important to define what we mean by context in soccer, since this is mentioned throughout this thesis. For a machine learning model to better be able to spot off-screen events, we believe that the model needs to have an understanding of the context of the game. We define context in this sense, as an understanding of how events in soccer are influenced by what happened in the past, and how events influence what happens in the future, within a single game.

2.14 Related work

2.14.1 Activity recognition & action spotting

The previous decade saw large advancements in the field of action recognition and event detection. At the start of the previous decade, research was beginning to make the available datasets for action recognition obsolete. The datasets were too limited in their classes to represent the vast variety of human actions in the real world, and the video-clips themselves were too controlled in terms of video quality, camera movement, and human actions [45]. UCF101 [45] and HMDB [29] sought to change this, by introducing datasets with a higher number of classes, video quality variety, and sheer volume of content. These datasets included a wide range of classes, including sports, body-motion, inter-human interaction, and human-object interaction. Following the success of CNNs applied to image classification [28], Karpathy et al. [26] provided an extensive empirical evaluation of using CNNs on a large-scale video dataset. They achieved significantly better results than contemporary models, and showed that CNNs were able to generalise to other datasets as well. By applying their model to the UCF101[45] dataset, they were able to improve the baseline by 19.4 percentage points.

In the field of general human activity understanding, ActivityNet [7] presented a large scale dataset with a total number of 849 untrimmed hours of video, containing 203 activity classes. Three benchmark tasks were introduced with the dataset: predicting activities in untrimmed videos, where each video might contain several activities, predicting the correct label of trimmed videos containing one activity, and detecting all activities in untrimmed videos, with temporal start and end times for each activity. Comparing their model to other state-of-the-art models [17, 26, 45, 29], ActivityNet had a significantly lower benchmark, reflecting the difficulty of the novel challenges introduced with this dataset.

A common challenge when working on computer vision problems, is to procure enough labelled data [25]. To sufficiently train a model, a significant amount of training data is required. Labelling data requires a human effort, which makes it an expensive procedure. A semi-supervised algorithm has been proposed [25] to reduce the amount of annotated data needed for training. This model can, with a limited amount of labelled test data, produce results matching those of the state-of-the-art models that fully utilises supervised learning. By training a CNN with the 3D ResNet 18 architecture on the UCF101 [45], Kinetics [27], and HMDB51 [30] datasets, the algorithm can outperform other state-of-the-art supervised models. Another solution to the problem of limited labelled data was proposed by Brattoli et al. [5] which uses Zero-shot (ZSH) learning, that is able to generalise well to other problems where the classes are not known. The model was trained once on the Kinetic dataset [27] and then tested on different datasets with different classes. The authors used a trainable 3D CNN network to learn visual features, as opposed to other ZSH methods, that uses pretrained feature extraction. Using a training set with a high diversity of classes, this model proved to outperform other state-of-the-art models where training data and test data overlaps.

In 2018, the SoccerNet dataset was introduced in a paper by Giancola et al. [18]. Along with the dataset, the authors provided a baseline for event recognition and action spotting on the classes of *Goals*, *Substitution* and *Cards*. They achieved an Average-mAP of 49.7% for the action spotting task. This dataset, and its novel tasks, facilitated further research into video understanding in the soccer domain.

Cioppa et al. [12] proposed a context-aware loss-function that increased this baseline from 49.7% to 62.5% for the action spotting task. This loss-function utilises the temporal distance of a given frame from the closest ground truth action frame for each class. The basic idea is that frames that are temporally close to the ground truth action, contain more relevant information than the frames further away. Because of this, frames with a high temporal distance to the

ground truth that predict an action, incur a high loss. Frames that are temporally close to the ground truth and that predicts an action, incur a low loss. Furthermore, frames that are close to the ground truth that do not predict an action, also incur a high loss. While this loss-function improved upon the previous baseline, the loss-function saw a reduction in Average-mAP minutes before and after the half-time break. As the authors stated, the network requires padding at the beginning and end of the halves, which may contribute to the reduction in performance. The model also saw a reduction in performance when actions are in close proximity. This was speculated to be a result of a reduced number of visual cues such as replays, or in the case of two consecutive actions, where the replay of the first action is shown after the second action has happened. Vats et al. [51] achieved comparable results using a multi-tower temporal convolutional network. Since different events can occur over different time spans, the towers have different receptive fields to account for this. The model was tested on both ice-hockey and soccer datasets, but the results were better on SoccerNet, where the model achieved an Average-mAP of 60.1% for action spotting. Vanderplaetse & Dupont [49] proposed using both audio and video for action spotting in soccer videos. Most work has been done using only data from videos. The authors trained the video feed and audio stream separately and then used merge methods to fuse the models. Video-only performed better than audio-only, except for classifying *Goals*, while the combined multimodal approach outperformed both. Rongved et al. [36] used multimodal approaches as well. They experimented with different models and both late and early model fusion. Using a multimodal approach with audio and video features improved the performance for *Goal* events, but not for *Cards* and *Substitutions*.

Efforts has also been concentrated on developing models that perform real-time spotting. Rongved et al. [35] experimented with a CNN with end-to-end training on SoccerNet. Although this model had a lower detection accuracy than the state-of-the-art, it performed well when tolerance for time estimation was low. Their model also had a significantly lower delay than the current state-of-the-art, showing promising results towards the goal of reaching real-time action spotting and event recognition in soccer. Tomei et al. [48] devised RMS-net, a network for action spotting that achieved an Average-mAP 67.8%. They built on the work by Cioppa et al. [12], and the idea that the most relevant visual cues occur just after an event. During training, the network is paired with a masking strategy to focus on the relevant portions of the input data.

Minoura et al. [33] achieved an Average-mAP of 81.6% using a Transformer model [50]. The transformer model was adapted to capture context over longer time periods and similar scenes in soccer videos.

SoccerNet-v2, an extension to the SoccerNet dataset, was released in 2020 [11]. Together with the dataset, Deliége et al. [11] set a baseline for the task of action spotting using the context aware loss-function from Cioppa et al. [12]. The Average-mAP achieved was 40.7%.

Giancola & Ghanem created a pooling method named NetVLAD++ [19], based on NetVLAD [4], to improve the results on action spotting. The proposed pooling method, compared to for example max pooling, is trainable. The pooling layer uses k-means to learn clusters from the features in the training data, extracting low-level semantical information from the frames it is trained on. Furthermore, for each window of frames that the model processes, NetVLAD++ aims to extract meaningful information from the window as a whole, by aggregating the features with respect to the average distance of each feature to its closest cluster. NetVLAD++ also considers the order of the frames, with the idea that temporal context before and after an event contains information related to the event. Two different events may have similar frames just before the event took place, while the frames just after differs, and vice versa. At the time of publication, Giancola & Ghanem achieved a state-of-the-art Average-mAP of 53.4% for the action spotting task.

Zhou et al. from Baidu research achieved impressive results [54] on the SoccerNet-v2 dataset using features extracted from a feature-extractor trained on soccer clips. Rather than training a model on the features extracted by ResNet [20], pretrained on ImageNet [13], the Baidu research team trained several action recognition models on the clips of SoccerNet videos, before extracting the features. Combined with a Transformer [50] model, the research team won the SoccerNet-v2 competition for action spotting with an Average-mAP score of 79.28% and 47.8% for shown and off-screen events respectively, and a total Average-mAP score of 73.77% [16].

2.14.2 Off-screen event detection

In SoccerNet-v2, the binary *visibility* attribute was introduced to the annotated events. The authors ran 4 action spotting models [18, 49, 12] on the new dataset, and reported the Average-mAP results for both visible and off-screen events. All models had lower performance for off-screen, compared to visible events. The new benchmark, set using the CALF model [12], achieved an Average-mAP score of 42.1% and 29.0% for spotting visible and off-screen events respectively. Since these method were previously developed for SoccerNet, which did not include the visibility of events, none of the methods were created with any special considerations for off-screen events.

At the time of publishing, Giancola & Ghanem presented a new state-of-the-art model [19], achieving an Average-mAP score of 53.4%,

59.4% and 34.8% when spotting all, visible, and off-screen events respectively. They argued that off-screen actions are challenging to learn from. In the future, they encouraged researching models that reach a higher level of understanding of the soccer broadcast [19].

Since we started writing this thesis, several models have been developed that achieved a higher Average-mAP than NetVLAD++ when spotting off-screen events [16]. To the best of our knowledge, we have not been able to find any sources in the literature that explicitly focus on improving off-screen action spotting. The authors of SoccerNet-v2 argue that it is challenging to spot off-screen events, since this requires analysis beyond a frame-by-frame basis, and requires the model to consider the context before and after the actions occur. We therefore think that this is an important and intriguing area of the field to research further.

2.15 Datasets

Deep-learning models have shown to be highly scalable with large datasets. The last two decades have seen demand rising for more and more complex datasets for video understanding with deep-learning. Available datasets in the 2000s were limited by the fact that they were recorded in controlled conditions (fixed camera angles, limited background noise, etc) and did not represent the real world [8]. In the early 2010s, several datasets were introduced that had non-controlled backgrounds and illumination [45, 29]. Advancement in the field of action spotting were also facilitated, with the release of THUMOS14 [17], a dataset with a total of 24 hours of temporally untrimmed videos, with 6,000 temporally anchored activities, distributed over 20 classes. ActivityNet [7] introduced a large-scale dataset for human activity understanding in videos. Totalling 849 hours of video, where 68.8% of those contain general human activities distributed among 203 activity classes. ActivityNet utilised YouTube to gather video material, and hired manual workers to label the untrimmed videos, as well as annotating temporal boundaries for activities in the videos. Yu et al. released a dataset with 222 soccer videos from the World Cup, Asian Cup, European Championship and the English Premier League [19]. The videos each consist of one half of a soccer game and are approximately 45 minutes in length. The videos have three types of annotations: video shots, event detection and player tracking. The video shot annotations consist of start and ending frame number for the shot, type of video shot, and shot transition type. The annotations for event detection consist of a label for the event type and a start and end point. There are eleven different event types in the dataset and a total of 6,850 event annotations. Annotations for player tracking consist of 1,908 frames

with a bounding box for players.

SoccerNet introduced the first large-scale soccer focused dataset for activity detection. SoccerNet consists of a total of 500 games, with 764 hours of video and 6,637 action annotations, distributed among three classes, *Goals*, *Yellow/red card* and *Substitutions*. The 500 games are collected from the 6 biggest European soccer competitions, over the course of three seasons. Each game consists of two untrimmed videos, one for each half of the game. Games are collected from online sources, and event annotations with a 1-minute margin of error are collected from parsing match reports from the respective leagues websites. This annotation process makes the dataset highly scalable. For a more accurate annotation, manual labour provided annotations down to the second of the event occurring, according to the rules defined in soccer. SoccerDB [24] built upon this dataset by expanding it with 7 additional classes and 76 new games, while using only half of the games from SoccerNet. The dataset does not provide full videos of the games, but smaller video segments with annotations. In 2019, Pappalardo et al. [37] released a dataset of soccer logs describing match events from seven large soccer competitions. The dataset does not contain video, but has annotations for seven event types that can be used for event detection.

The recently released SoccerNet-v2 [11] expanded upon the SoccerNet dataset. Although the quantitative amount of videos remains unchanged from the first version, the amount of annotations were significantly increased, from ~6,000 to ~300,000 annotations. SoccerNet-v2 expanded the categories of annotations with camera shots and replays. The number of classes for action spotting was also increased from 3 to 17.

Compared to other soccer-related databases [24, 53, 37], SoccerNet-v2 provides the most comprehensive dataset available for work related to action spotting and event detection in soccer broadcasts.

2.16 Ethical Considerations

In this section, we discuss the relevant ethical considerations that need to be addressed when researching and developing a machine learning model for activity recognition in sports.

When we are working on a machine learning model that is based on imaging data, we must consider the possibility that the model is not able to generalise well to unseen data. This is especially true if the training data is not representative of the population [39]. In the context of our work, we consider this a realistic scenario. The SoccerNet-v2 dataset, which has been exclusively used to train our model, only contains data from the top leagues in Europe [11]. The

geographical limitation of this might lead to certain ethnicities being under-represented in the dataset. The result might be that the machine learning model works less well when applied to other leagues and geographically situated soccer teams. As such, other leagues than the top European leagues might not be able to reap the benefits of such a system. In a hypothetical scenario in which an action spotting model is used to gather statistics for a team to analyse, the disadvantage might become quite significant.

Following this, it is important to consider the availability of action spotting systems applied to real-life scenarios. One could argue that applying, and taking full advantage of machine learning models in soccer, requires resources and technical proficiency not available to all soccer teams. This could therefore lead to an unfairness in which only the most resourceful leagues and teams are able to use such models, which would increase an already existing performance advantage.

Throughout the work in this thesis, we have aimed to be as transparent in our research as possible. This includes the following:

- Presenting our results objectively, without leaving out key figures and numbers.
- Explaining our reasoning behind the analyses of these results.
- To the best of our knowledge, discuss the shortcomings of our results, as well as our work and analysis, and how these shortcomings might affect the results.

2.17 Summary

In this chapter, we have presented relevant paradigms in machine learning for this thesis, as supervised learning and unsupervised learning. We have introduced classification tasks, as they are a common problem within the domain of supervised learning. We have learned that computer vision tasks are related to machines understanding the visual world. We have presented the general ideas behind neural networks and CNNs, which are important for understanding how many modern algorithms for video and image understanding work. Deep learning, a sub-set of machine learning algorithms that utilises a deep-layered architecture to better solve problems, has shown great promise and practical applicability in recent years.

We defined an event in soccer as an action happening at an exact point in time. We defined action spotting as the task of classifying which event takes place, and when, in a soccer clip. We have detailed

what event visibility in SoccerNet-v2 is, and presented a definition for what context in soccer means.

Automating action spotting in videos has seen great improvements in recent years with new datasets and methods. During the last two decades, relevant datasets in activity understanding in videos has gone from the non-realistic [8], to the comprehensive [7], and into the domain specific [11]. With the release of SoccerNet and SoccerNet-v2, several papers have contributed to the task of action spotting [12, 51, 49, 19, 54]. Still, most of these machine learning models perform worse when spotting off-screen events, and none focus explicitly on these events. By the end of this chapter, we discussed the relevant ethical considerations for this thesis. In the next chapter, we will describe our ideas for a machine learning model that can better spot off-screen events.

Chapter 3

Methodology

3.1 Introduction

Today, events in soccer videos are annotated manually by human operators. This is a time-consuming and costly process. Using machine learning to automate this, could greatly decrease these costs. Furthermore, researching this field could potentially generalise to other domains of computer vision and bring new insights into machine learning as a whole. One of the biggest factors that contribute to furthering computer vision research is the increasing sophistication, size, and data quality of available datasets for model training. Since the release of SoccerNet [18], we have seen large improvements in terms of action spotting in soccer videos [12, 48, 54]. In 2021, SoccerNet-v2 [11], an extension to the SoccerNet dataset was released. SoccerNet-v2 increased the total number of annotations for action spotting from ~6,600 to ~110,000, and increased the target-space from 3 classes, to 17 classes. Following the release of this dataset, a new state-of-the-art model was presented [54] that increased the Average-mAP score from 67.8% to 74.8%.

When spotting actions, our observation is that most research efforts are focused on either using visual data on a frame-by-frame basis, or using multimodal approaches combining audio and video data [49]. An exception to this is Cioppa et al. [12], which uses the temporal distance from a frame to its closest ground truth action to penalise its model during training. These efforts does not put much emphasis on the understanding of the larger context in the game, or the relationship between different types of events in soccer. Furthermore, little effort is focused on explicitly spotting off-screen events. We believe that most existing models perform worse when spotting off-screen events as a result of this. We want to study how we can train machine learning models to learn the relationship between different events in soccer. We want to see how this impacts the performance of action spotting

in general, and specifically how this impacts spotting off-screen events in the dataset.

In this chapter, we will present our new idea for better spotting off-screen events in soccer. We propose a model architecture combining three individual models, each learning their own separate dimension of context in a soccer game. We will first describe the dataset, with its limitations and specifications. We will also describe the intuition behind our ideas. Then, we will discuss how we can use event relationships in soccer to support this intuition, and present the *Past*, *Present* and *Future* contextual dimensions that together form a contextual understanding of what is happening in a soccer game. We will present how we have manipulated the dataset to facilitate training three different models on these contextual dimensions. Following this, we will detail how we can combine these models to create a model architecture with enhanced off-screen action spotting capabilities. We will also describe the existing model we have used as a baseline for our research. We will present how we have combined our different machine learning models in terms of data fusion, and the choices that were made in regards to hyper-parameters. We will finally present how most models using the SoccerNet dataset are evaluated, how we evaluate our models, and provide a summary at the end of this chapter.

3.2 The SoccerNet-v2 dataset

SoccerNet-v2 is a dataset containing 500 soccer games, as they were broadcast, totalling 764 hours, with ~300,000 timestamped annotations [11]. As in SoccerNet, the games span three different seasons (14/15, 15/16 and 16/17), the five biggest national leagues in Europe (La Liga in Spain, English Premier League (EPL) in England, Ligue 1 in France, Serie A in Italy, and Bundesliga in Germany) as well as the annual UEFA Champions League competition. Table 3.1 shows the distribution of games per season and league. SoccerNet-v2 is an extension of SoccerNet [18], which contained the same set of soccer games, but only included ~6,600 annotations. Compared to the first version of SoccerNet, SoccerNet-v2 has expanded the target-space in action spotting from 3 to 17 classes. These 17 classes represent some of the most important events in soccer. Each event is annotated with a single timestamp, and the timestamps are defined in the supplementary material of SoccerNet-v2 [11] as:

- *Ball out of play*: Moment when the ball crosses one of the outer field lines.
- *Throw-in*: Moment when the player throws the ball.

Game distributions				
League	Season			**Total**
	14/15	15/16	16/17	
EN - English Premier League	6	49	40	**95**
ES - La Liga	18	36	63	**117**
FR - Ligue 1	1	3	34	**38**
DE - Bundesliga	8	18	27	**53**
IT - Serie A	11	9	76	**96**
EU - Champions League	37	45	19	**101**
Total	**81**	**160**	**259**	**500**

Table 3.1: Distribution of games across different leagues and seasons in the SoccerNet-v2 dataset.

- *Foul*: Moment when the foul is committed.
- *Indirect free-kick*: Moment when the player shoots, to resume the game after a foul, with no intention to score.
- *Clearance* (goal kick): Moment when the goalkeeper shoots.
- *Shots on target*: Moment when the player shoots, with the intention to score, and the ball goes in the direction of the goal frame.
- *Shots off target*: Moment when the player shoots, with the intention to score, but the ball does not go in the direction of the goal frame.
- *Corner*: Moment when the player shoots the corner.
- *Substitution*: Moment when the replaced player crosses one of the outer field lines.
- *Kick-off*: Moment when, at the beginning of a half-time or after a goal, the two players in the central circle make the first pass.
- *Yellow card*: Moment when the referee shows the player the yellow card.
- *Offside*: Moment when the side referee raises his flag.
- *Direct free-kick*: Moment when the player shoots, to resume the game after a foul, with the intention to score or if the other team forms a wall.

- *Goal*: Moment when the ball crosses the line.
- *Penalty*: Moment when the player shoots the penalty.
- *Yellow then red card*: Moment when the referee shows the player the red card.
- *Red card*: Moment when the referee shows the player the red card.

At this point it is important to note that we do not feel that these classes represent every interesting event in soccer, and as such it can be seen as a limitation of the dataset. For example, players can be awarded a yellow or red card without committing a foul, by excessively yelling at the referee or other players. These events are often controversial and interesting to viewers. In this case, SoccerNet-v2 will only have annotated the yellow card. We nevertheless adhere to the aforementioned 17 classes in this thesis.

Some of the event names used may cause some confusion, since they have a different meaning in other contexts. The term *Ball out of play* has a looser definition in the rules of the game, article 9.1 [23], where it says that the ball is out of play when the play has been stopped by the referee as well as when the ball crosses the outer field line. A *Clearance* is often used to describe the act a player performs when the player kicks the ball away from the goal they are defending. The *Clearance* event in SoccerNet-v2 is called a *Goal kick* in the laws of the game, article 16 [22]. There may also be a difference between the free-kicks, *Indirect free-kick* and *Direct free-kick*, annotated in SoccerNet-v2 and their definition according to the rules of the game. In SoccerNet-v2, the difference between *Indirect free-kick* and *Direct free-kick* is the intention to score. In the rules of the game, the difference between the free kicks is the fouls for which they were awarded. According to soccer rules, a player is not allowed to score directly from an *Indirect free-kick*. Because of this, a *Direct free-kick* following the rules of soccer, may be annotated as an *Indirect free-kick* if it takes place far from the opponents goal and the player chooses to pass to a teammate. In order to avoid confusion, we will refer to the SoccerNet-v2 definitions when using the names in the list above throughout this thesis, unless explicitly stated otherwise.

In the SoccerNet-v2 dataset, there are 110,458 annotated actions, averaging 221 actions per game, or one action every 25 seconds. There are also two other categories of annotations: Camera shots and replays. Camera-annotations represent a change in camera, and replay-annotations are timestamps of replays in the broadcast. Since we are only considering action spotting, camera and replay annotations are outside the scope of this thesis, and are ignored in our work.

Each game is divided into two videos, one for each half, which leads to a total of 764 hours of video. Each annotated action also contains a visibility-attribute (*visible/not shown*), indicating if the action is shown in the broadcast or not. For example, if a *Goal* triggers one or more replays, the following *Kick-off* might happen off-screen. The *Kick-off* event after the goal is off-screen in the broadcast and is therefore tagged as *not shown*. Another example might be that the production of the broadcast chooses to show a close-up video of a player, as a *Throw-in* is happening. Figure 3.1 shows the distribution of all the different classes, and how these are distributed between visible and off-screen. This is also presented with raw numbers in Table 3.2.

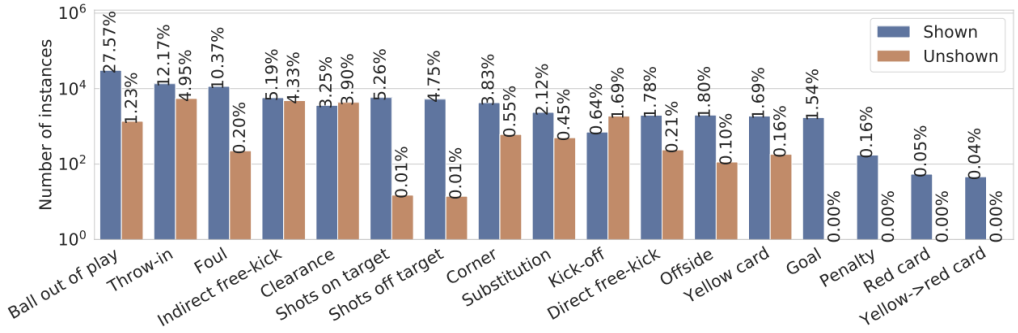


Figure 3.1: Figure showing the log-scale distribution for visible (shown) and off-screen (unshown) actions in the SoccerNet-v2 dataset. Reprinted from [11].

.

The dataset folders are structured after league first, season second, and game third. Each game has its own folder where labels and features are stored. Labels are stored in the JSON-format, and there is one JSON-file for each game. Each label-file contains some metadata about the game itself, such as the relative path to the game from the dataset root-folder, a YouTube-URL (if there is any), and a list of annotations formatted as JSON-objects. Figure 3.2 shows an example of such an annotation. The *gameTime* attribute contains information about which half the action occurred, and at which time (*mm:ss*). The *label* attribute indicates which class the action belongs to, and the *position* attribute is the time in milliseconds from the start of the half where the action occurred. The *team* attribute indicates which team performs the action, and lastly, the *visibility* attribute says whether the action is visible or off-screen in the broadcast. The *team* attribute is not considered in our work for this thesis. SoccerNet [18] provides high- and low-quality video of all games in the dataset. The SoccerNet-v2 devkit also provides extracted features for all the games in the dataset. The SoccerNet-v2 devkit offers a framework of code and

Class	Visible	Off-screen	Total
Ball out of play	30,450	1,360	31,810
Throw-in	13,448	5,470	18,918
Kick-off	703	1,863	2,566
Indirect free-kick	5,734	4,787	10,521
Clearance	3,590	4,306	7,896
Foul	11,450	224	11,674
Corner	4,229	607	4,836
Substitution	2,341	498	2,839
Offside	1,985	113	2,098
Direct free-kick	1,963	237	2,200
Yellow card	1,866	181	2,047
Shots on target	5,805	15	5,820
Shots off target	5,242	14	5,256
Goal	1,702	1	1,703
Red card	54	1	55
Penalty	173	0	173
Yellow then red card	46	0	46
Total	**90,781**	**19,677**	**110,458**
Percentage	**82.2**	**17.8**	**100**

Table 3.2: The distribution of events in the SoccerNet-v2 dataset. The events are divided into 17 classes and are either visible or off-screen.


```
{
    "gameTime": "1 - 02:29",
    "label": "Throw-in",
    "position": "149168",
    "team": "away",
    "visibility": "visible"
},
```


Figure 3.2: An example of an annotation in the SoccerNet-v2 dataset.

APIs, so that one can easily experiment with the dataset and some of its models out-of-the-box. The features in the devkit were extracted with a ResNet-152 [20] model, pretrained on ImageNet [13]. This ResNet-152 model extracts features at 2 fps. Furthermore, the devkit offers functionality to reduce the dimensions of the features to 512 dimensions, with PCA. However, Giancola and Ghanem [19] argue that dimensionality reduction with a linear layer is better suited to learn a linear combination of the frame features. For our work in this thesis,

we have experimented with both.

3.3 Understanding the context of soccer

As can be seen in Table 3.3, existing models evaluated on the SoccerNet-v2 dataset perform worse when spotting off-screen events in comparison to shown events. The difference in performance

Model	Total Average-mAP	Average-mAP *visible*	Average-mAP *off-screen*
Baidu Research	73.77	79.28	47.84
AlmageLab-RMS	63.49	68.88	38.02
NetVLAD++	53.40	59.41	34.97

Table 3.3: Performance of three existing models on the test-split of the SoccerNet-v2 dataset. The columns show model performance when spotting all visibilities, visible events only, and off-screen events only.

between visible and off-screen events seems reasonable. For example, NetVLAD++ uses only visual features and it therefore comes as no surprise that this model performs worse when spotting off-screen actions. A human counterpart on the other hand, does not rely solely on the visual features of the game. As an example, consider a *Goal* that is followed by several replays. After the replays, when the broadcast returns to the live game, the players have already started playing. In this case, we have an off-screen *Kick-off* event, that occurred during the replays. For humans it is easy to understand that a *Kick-off* happened at some point during the replays, because we have an understanding of the rules, and the causality between events in the game. For our model to better be able to spot off-screen events, we believe that our model has to have some sort of understanding of the context of the game.

Soccer has clearly defined rules, and for most events in the game, what happened in the past (the event before the current one) or in the future (the event after the current one) can be derived from the event currently happening.

3.4 Event relationships in soccer

We believe that current machine learning models perform worse when spotting off-screen events because they have no sense of context of the game. We also believe that context in soccer is in large part informed by the causal relationships between the events of the game. Most events have a connection to other events according to the rules of the game. An

example is the *Ball out of play* event and the *Throw-in* event. A *Throw-in* is awarded when the ball passes over the touchline, which means that a *Throw-in* is always preceded by a *Ball out of play*. A related example is the *Goal* event, which is always followed by a *Kick-off*. We will utilise these relationships between different events, in order to better spot off-screen events in soccer videos.

Expanding on these examples, we identify which events may precede or follow each of the other events from the 17 classes, using the rules of the game. These relationships will allow us to spot an off-screen action in the soccer broadcast using the event before or after the action we are spotting.

More formally, we can define two binary relations over the set of events from Section 3.2. The first relation is detailed in Figure 3.3. This relation aRb from event a to event b means that event *a leads to* event b. The second relation, xRy, shown in Figure 3.4, means that event x is a *result of* event y.

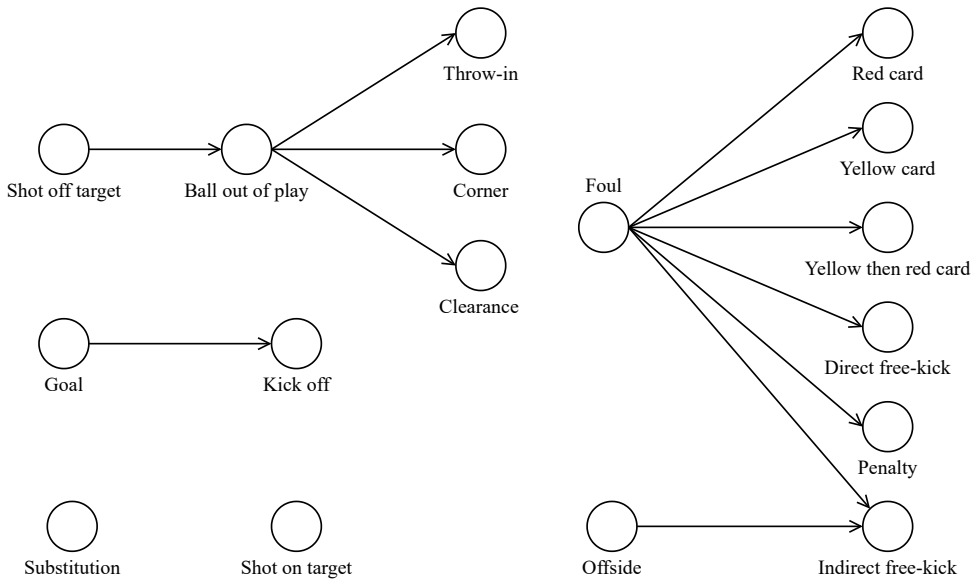


Figure 3.3: Relation aRb from event a to event b, where event a leads to event b.

As we can see from Figure 3.3, the *Ball out of play* event leads to a *Throw-in*, *Clearance* or *Corner*. The position on the field where the ball crosses the boundary lines and which player last touched the ball is the deciding factors of which of the events is going to happen. An *Offside* leads to an *Indirect free-kick*, a *Shot off target* event leads to *Ball out of play*, and a *Goal* event leads to *Kick-off*. A *Foul* can lead to *Indirect free-kick*, *Direct free-kick*, *Penalty*, *Yellow card*, *Red card*, and/or *Yellow then red card*. An overview of events in this group can be seen in Table 3.4.

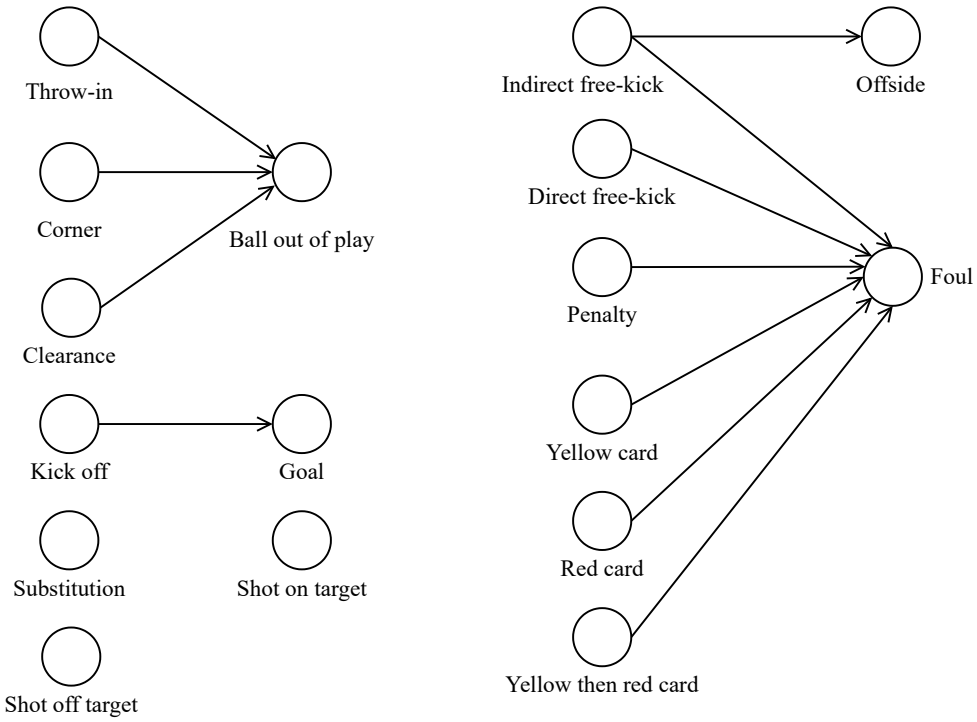


Figure 3.4: Relation xRy from event x to event y, where event x is a result of event y.

Event	Followed by
Ball out of play	Throw-in, Clearance, Corner
Foul	Indirect free-kick, direct free-kick, yellow card, red card, penalty
Offside	Indirect free-kick
Shot off target	Ball out of play
Goal	Kick off

Table 3.4: Soccer events that are followed by other events

In Figure 3.3, we can see that the *Foul* event has many possible outcomes and not all are mutually exclusive. Of the possible outcomes, the three events *Indirect free-kick*, *Direct free-kick* and *Penalty* are mutually exclusive, and the three events *Yellow card*, *Red card* and *Yellow then red card* are mutually exclusive events after a *Foul*. This means that the action in a single *Foul* event can cause both a *Yellow card* and a *Penalty*.

Looking at the second relation shown in Figure 3.4, we can see that the *Throw-in* event is a result of a *Ball out of play* event, if the ball

crosses the touchline. A *Kick-off* is a restart of play and happens after a *Goal* event and at the start of each the half. An *Indirect free-kick* is the result of a *Foul* or an *Offside*. A *Clearance* is the result of a *Ball out of play* event if the ball crosses the goal line and the last player to touch the ball was of the attacking team, while a *Corner* is the result of a *Ball out of play* event if the ball crosses the goal line and the last player to touch the ball was of the defending team.

The remaining events in this group; *Direct free-kick*, *Yellow card*, *Red card*, *Penalty* and *Yellow then red card* are possible results from the *Foul* event. An overview of events in this group can be seen in Table 3.5.

Event	Preceded by
Throw-in	Ball out of play
Kick-off	Goal
Indirect free-kick	Foul, Offside
Clearance	Ball out of play
Corner	Ball out of play
Direct free-kick	Foul
Yellow card	Foul
Red card	Foul
Penalty	Foul
Yellow then red card	Foul

Table 3.5: Soccer events that are preceded by other events

The events might not happen immediately after one another. There may be one or more events happening in the time between the two events that has an identified relationship. An example of such an event could be a *Substitution*. A substitution only occurs during a stoppage in play, and this is typically after a *Ball out of play* event, but before the following *Throw-in* event. In the dataset, we might therefore find examples of *Substitution* and other events happening between the events we have identified as having a relationship.

As mentioned earlier, a *Throw-in* event is always preceded by a *Ball out of play*, but it does not follow that a *Ball out of play* event is always followed by a *Throw-in*. A *Ball out of play* is followed by either a *Throw-in*, *Clearance* or *Corner*, depending on where the ball went of of play and which team the player who last touched the ball belongs to. Since the labels from the dataset does not contain information about these relationships, we believe that information about the position of the ball on the field and the player is captured in the video, and that our model will use this information to be able to predict the correct event after e.g a *Ball out of play*.

Not all the events in the 17 classes have an identified relationship. These are *Substitution* and *Shot on target*. We will not be able to predict these events based on the previous or following event. We also mentioned that a *Kick-off* event is preceded by the start of each the half in addition to the *Goal* event. The start of each half is not a labelled event, and we will have to consider the edge-cases where our model predicts a *Goal* based on a start of the half *Kick-off*. In our dataset we may have some exceptions to these rules. As mentioned, cards have been given without a foul committed and there might be events missed due to human error.

3.5 Dataset annotation modification

Following the previous section and the event relationships we discovered, we define three dimensions of context. A *Present* context, which is an understanding of what is happening in the current frame. A *Past* context, which is an understanding of what has happened previously, based on what is happening in the current frame. A *Future* context, which is an understanding of what event will happen next or in the near future, based on what is happening in the current frame. We theorise that it is easier to recognise and predict an off-screen event by combining these dimensions to form a contextual understanding of what is happening in the game.

We want to train three separate models that learns these three contextual dimensions to better predict off-screen events. A natural progression of this, is therefore to consider the training data. In our dataset, there are ~500 files with annotations, one for each soccer game. These files follows the file structure described in Section 3.2. Our idea is to use the original annotations provided by the SoccerNet-v2 devkit to train the *Present* model, and derive the training data for the *Past* and *Future* models from the original annotations. Figure 3.5 shows a basic label-shifting scheme, where we modify the original annotations so we are able to train our *Past* and *Future* models on their respective contextual dimension.

The green objects in Figure 3.5 represents a list with the original annotations for a game, including its *label*, which we will denote $A_{Present}$. The annotations are sorted by the *position* attribute, so that the first annotation in the list represents the first annotated event in the game. For the *Future* model, we first make copies of the annotations, A_{Future}, shown as blue objects in the figure. We then iterate through the A_{Future} annotations, using an iterator, i, and for each annotation, we shift the *label* such that: $A_{Future}[i][''label''] = A_{present}[i+1][''label'']$. This is done for all A_{Future} annotations, except the last one, which represents the last annotated event. This annotation

remains unchanged, since there are no more following events. We repeat the process for the annotations used by the *Past* model, A_{Past}, shown as red objects, except that the labels are shifted such that: $A_{Past}[i]["label"] = A_{present}[i-1]["label"]$. The first annotation in A_{Past} is not changed, since there are no events preceding it.

At first glance this method is quite naive. From Figure 3.5 we see that this label-shifting scheme will replace the *Kick-off label* in Annotation 2 (*Labels-future.json*) with *Shots on target*, since these two events follow each other in this particular game. There is no semantical connection between these events in the rules of the game. This is by design from our part - we want to start with a very naive method, in the hopes that we will learn from each model, and improve the label-shifting as we go along with our experiments.

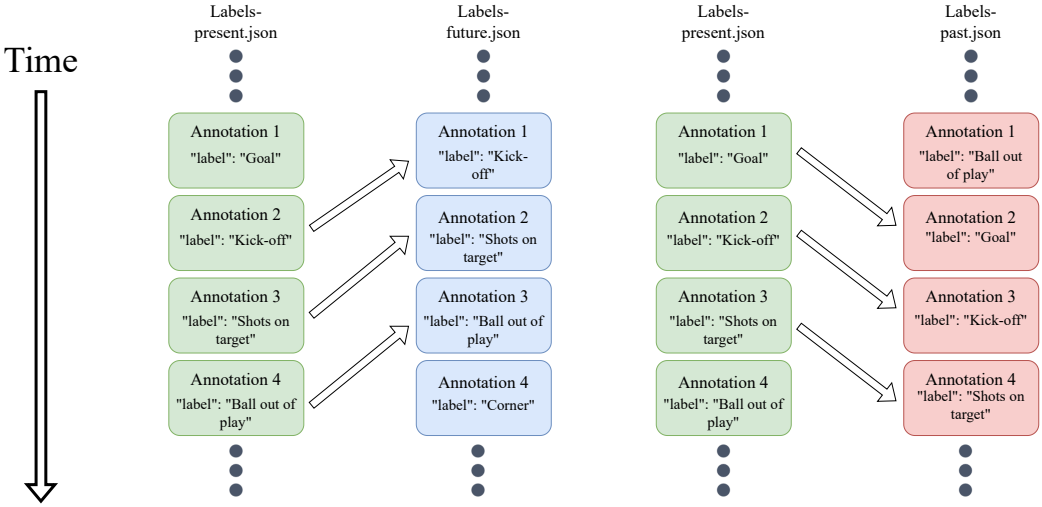


Figure 3.5: Basic label-shifting scheme, showing how we shift the *label* of an annotation in the *Labels-past.json*/*Labels-future.json* files with the *label* of the previous/following annotation, to create training data that reflects the *Past* and *Future* contextual dimensions.

3.6 Combining the past, present and future to understand context

In the previous section we presented a basic label-shifting scheme as to create training data for the *Past* and *Future* models. Our idea is that each model (*Past*, *Present*, *Future*) is trained to become an expert on its contextual dimension. To create an action spotting model that has a better understanding of the context of the game, we need to combine these three models. Figure 3.6 shows the architecture of these combined models. The leftmost NetVLAD++ implementation, named

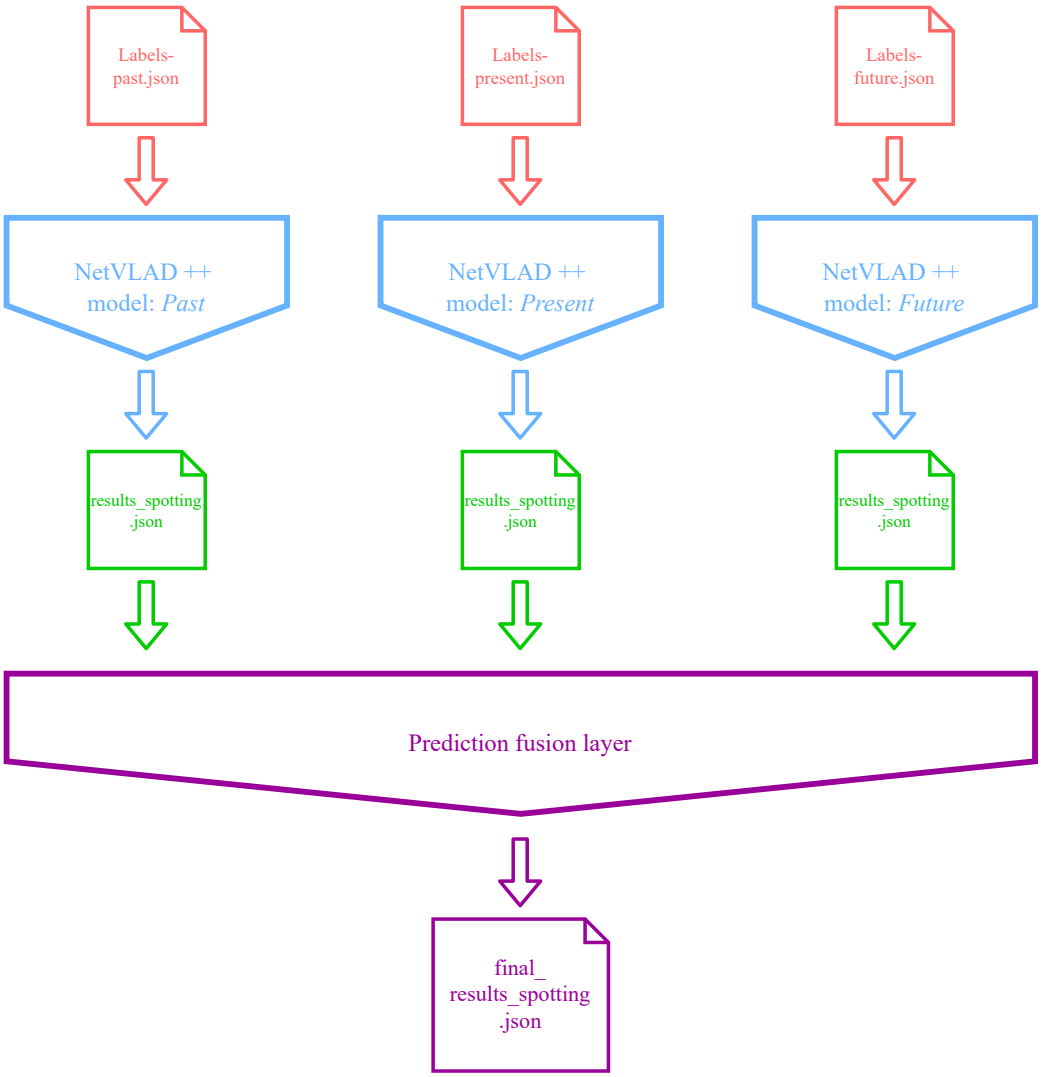


Figure 3.6: Architecture for our three machine learning models. We train a *Past*, *Present* and *Future* model on their respective labels, before combining the output from the three models in a prediction fusion layer, that outputs a final list of combined predictions.

the *Past* model, is trained to learn what event precedes the one that is currently shown in the frame of the broadcast, trained on the *Past* contextual dimension. This model should predict a frame showing a *Kick-off* as a *Goal*, since we know from the rules of soccer that a *Kick-off* is always preceded by a *Goal*, unless it is the start of the half, which we describe in more detail in Section 4.6. It is trained on its own set of labels derived from the labels in the SoccerNet-v2 devkit, described in Section 3.5, named *Labels-past.json*. Our *Present* model, shown in the middle, will be the *eyes* of our model, and spot the actions that are currently happening in the broadcast based on visual features alone. This model is also a NetVLAD++

implementation, and its labels, *Labels-present.json*, are therefore the same labels as were provided by the SoccerNet-v2 devkit. The *Future* model to the right, aims to learn what action follows the one currently in the frame. Given a frame of a *Goal*, this model should output a label of *Kick-off*. Like the *Past* model, the *Future* model also needs its own, modified set of labels to learn from, *Labels-future.json*. For each game these models process, each model outputs their individual file of predictions, named *results_spotting.json*. We define a final prediction fusion layer (fusion layer), that combines the input of these three models, and uses some form of data fusion to output one prediction file (*final_results_spotting.json*), that serves as the combined output from the three input models. Since we use NetVLAD++ to form the basis of all of our models, our main challenge lies in configuring the labels for each model and designing our fusion layer.

3.7 Model selection

We wish to derive our experiments from a model that is as close to, or the actual state-of-the-art in regards to the action spotting task of SoccerNet-v2. This model will also be used as our baseline model, as we want to see if we can improve upon its results. At the time of writing, the state-of-the-art model, and the winner of the action spotting task of the SoccerNet-v2 competition [16], is a solution proposed by Baidu research [54]. Originally, their model used a Transformer [50], but according to their tech-report, a NetVLAD++ model could achieve similar results. In this thesis, our initial plan was to derive our experiments from a NetVLAD++-model with Baidu's soccer embeddings, and use this model as our baseline. We did however have issues combining NetVLAD++ with Baidu's features. Instead, we will use a NetVLAD++ model with the features provided by the SoccerNet-v2 devkit as our baseline, and we will also derive our models from NetVLAD++. The reason for doing this is for the most part practical. The NetVLAD++ model is included in the SoccerNet-v2 devkit, and is thoroughly documented.

With NetVLAD++, Giancola & Ghanem proposed a novel temporally-aware pooling-method for action spotting in SoccerNet-v2. Compared to more traditional pooling methods, such as max pooling (see Section 2.6) or average pooling, this new pooling method is trainable. From a set of features, NetVLAD++ learns a set of clusters. Using k-means, NetVLAD++ will use all the features from the dataset, and learn K clusters, where these clusters will represent a semantic vocabulary over the dataset. As seen in Figure 3.7, this vocabulary might include Field camera, Pass, Goalkeeper, Shooting, etc. For a window of N frames, with N features of dimension D, this pooling method aims

to extract meaningful features from the whole window, by comparing the features to their closest cluster-center. The resulting output is a feature of dimension KxD. Furthermore, the contextual information in the frames before or after an action, is different in the game of soccer. A *Goal* action might share some of the same contextual features as a *Shot on target* before the goal takes place, but not after, since the *Goal* is usually followed by celebration from the scoring team, while the *Shot on goal* is not. Considering this, NetVLAD++ has two pooling-modules, one which pools features *before* the action takes place, and one which pools the features *after* the action takes place. In Figure 3.7, we see that the frames, and therefore features, before and after the *Goal*-event are pooled differently, separated by the temporal context in orange and green. The size of the vocabulary is not adapted for neither pooling module, and the clusters are not shared between them, therefore enforcing that each modules cluster size is equal.

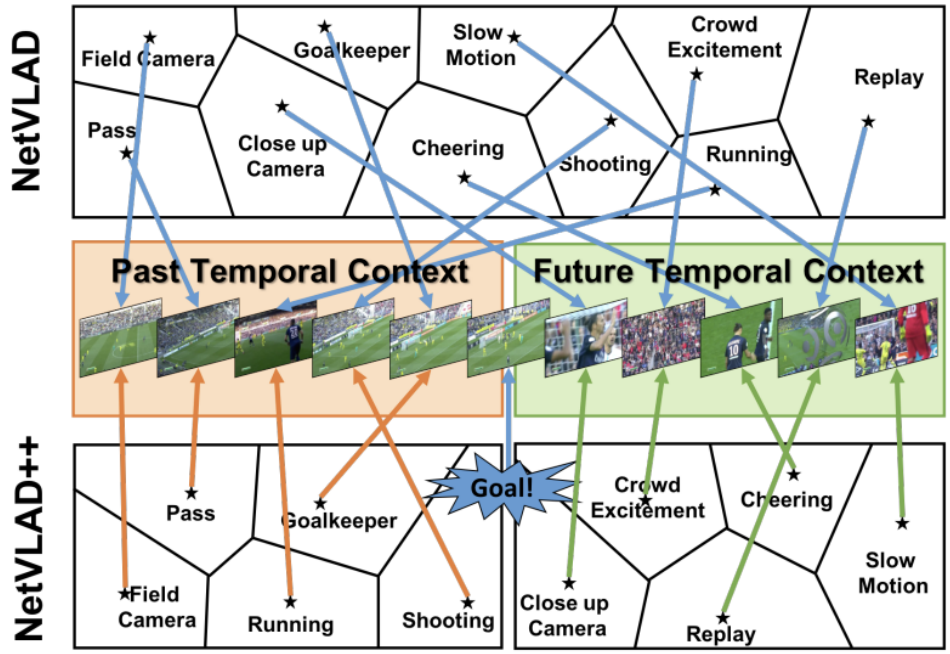


Figure 3.7: Figure showing how the NetVLAD++ pooling-layer extracts semantical meaning from a window of frames, using clustering. Reprinted from Giancola & Ghanem [19]

Giancola & Ghanem argue that this pooling technique provides more insightful output than more traditional non-trainable pooling modules. Additionally, how the frames in the soccer clip are ordered are important for the semantic meaning of the clip. If we reverse the order of frames in a clip of a *Ball out of play*, the semantical meaning of the clip would be changed. Max and average pooling does not consider this,

and the output from one of these pooling techniques would be same whether or not the frames were reversed. NetVLAD++ on the other hand, considers the temporal order in which the frames are processed. With regards to dimensionality reduction, NetVLAD++ proposes a learnable linear layer as a better performing alternative to using the PCA-algorithm. Table 3.6 shows the improvement in performance using the learnable layer instead of PCA for dimensionality reduction. In our experiments we will experiment with both.

Pooling method	NetVLAD++	NetVLAD++
Encoder	PCA	Linear layer
I3D	38.1 $\pm$0.1	41.5 $\pm$0.1
C3D	47.2 $\pm$0.2	48.6 $\pm$0.8
ResNet	50.7$\pm$0.2	53.3 $\pm$0.2

Table 3.6: Action spotting performance using I3D, C3D and ResNET video-encoders, averaged over 5 runs with different techniques for dimensionality reduction.

Figure 3.8 shows the architecture of the NetVLAD++ model. Using a sliding window approach with a temporal stride of 1 second, the same pre-trained ResNet-152 features that was presented in SoccerNet [18] are used. After this, dimensionality reduction is applied. The features are then processed by the pooling modules. A per-frame multi-label classifier is then applied to provide actionness for each frame. This classifier comprises of a single neural layer with sigmoid activation, and optimised with a cross-entropy loss function. Non-maximum suppression is then applied, to provide the action spotting.

3.8 Data fusion & ensemble learning

Data fusion can be defined as combining different sources of information to improve the quality of the information [9]. Within the scope of this thesis, our work will to a large degree be focused on combining the predictions of three different machine learning models in the hopes of achieving a greater result than any of the models can achieve individually. To do so, we must consider some form of data fusion.

3.8.1 Late data fusion

Late data fusion is the concept of combining two or machine learning models after they have been trained on the training data [43]. As can be seen in Figure 3.9, N-number of models are trained individually

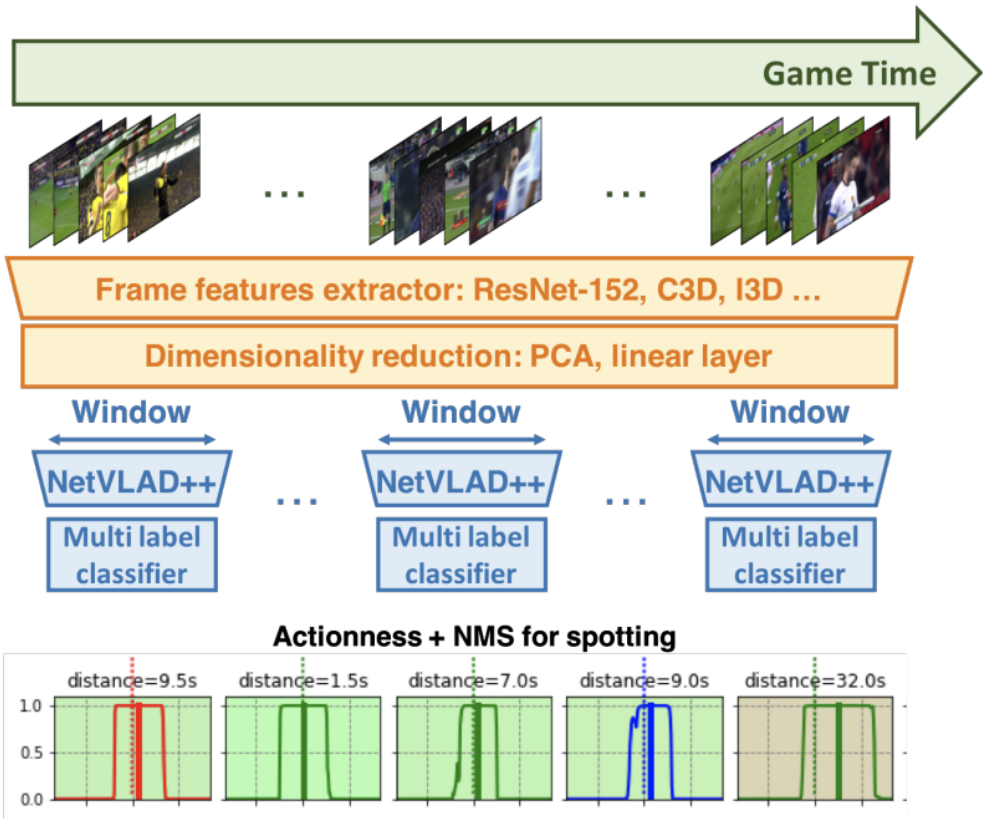


Figure 3.8: Figure showing the architecture of the NetVLAD++ model vertically, with the SoccerNet-v2 frames as input at the top. Reprinted from [19].

on some set of features extracted from a dataset. These models are then combined in a final layer, which fuses the output of the individual models together. In this thesis, the features are the same for all individual models, but the labels on which they train, are not. Finally, we combine these models in our fusion layer to produce our final output. Considering the fact that the features are the same for all of our models, and that it is the training of the models that are different, we use late data fusion to combine our models.

3.8.2 Ensemble learning

Ensemble learning combines multiple machine learning models to make a decision (or in our case, set of predictions) [40]. Typically applied to supervised learning problems, ensemble learning hopes to achieve a better result than any one model can achieve, by assuming that the errors or shortcomings of every individual model is compensated for, by the other models. In our case, we realise that there is a fine line between ensemble learning and late data fusion, since

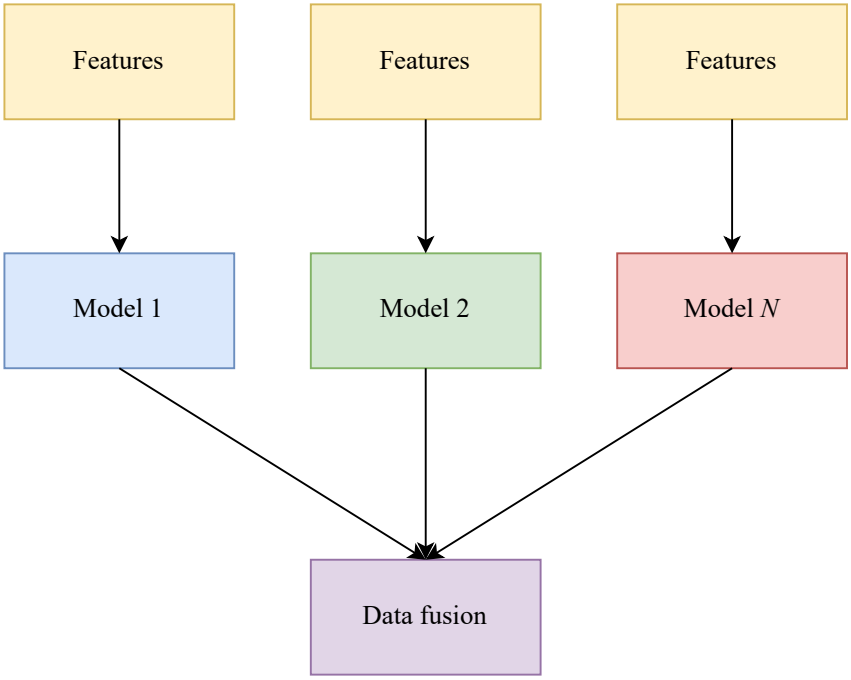


Figure 3.9: Late data fusion. Models 1 - N are trained separately, and processes new data individually. Each models output serves as input in the data fusion layer, which combines the different models outputs by some algorithm.

we use the same set of features for training the models we combine. Following our presentation of ensemble learning and late data fusion concepts, one can argue that the biggest difference between them is that data fusion defines *when* the data is combined and ensemble learning defines *how* it is combined. We have chosen to mention both concepts as we feel they that they are somewhat overlapping in our case, and that our implementations build upon a combination of ideas from both concepts.

3.8.3 Implementation

As mentioned in Section 3.6, we need to use data fusion. We use late data fusion and combine the output of our *Past*, *Present* and *Future* models, by using ensemble learning in the form of output fusion [40]. This seems reasonable, since we have a single data source and manipulate the labels, rather than the features.

We combine our *Past*, *Present* and *Future* models in a fusion layer that merges our predictions. These three models may predict the same event. The only two classes which have informed predictions from only one model is the *Shots on target* and *Substitution* classes, which are only predicted by our *Present* model. For the other classes, we may have

multiple predictions for the same event and we wish to remove these duplicate prediction to reduce the number of false positive predictions.

First, we have to change the value of the *position* attribute of the predictions from our *Past* and *Future* models. The predictions from these models have a *position* value corresponding to the related event and not the event we are predicting. We therefore have to decrease the *position* value for the predictions from the *Past* model, and increase the *position* value for the predictions from the *Future* model. The *position* value was changed with a value T so that the new *position* value is $position \pm T$. Both the T and *position* values are in milliseconds. After we have adjusted the *position* of the predictions from the *Past* and *Future* models, we continue merging the predictions. This is explained in further detail in Section 3.9.2.

3.9 Hyperparameters

Hyper-parameter selection is vital when working with machine learning models. These choices will affect how well our model is able to learn and generalise over our chosen problem. In this section we will discuss which choices were made in regards to hyper-parameters.

3.9.1 NetVLAD++

We want to derive all models from our baseline NetVLAD++ model, to see if our models and ideas are able to improve upon this model. For most hyper-parameters, we have therefore chosen to use the values that resulted in the best performance, according to Giancola & Ghanem [19]. For the pooling-modules, we use the same amount of parameters as a traditional NetVLAD [4] pooling layer. As in NetVLAD++, we have used a window-size of 15 seconds, and K = 64 clusters for the k-means clustering since this resulted in the best performance. Additionally we have used a centered window of 30 seconds for the non-maximum suppression. An Adam optimiser with default parameters from PyTorch is used for learning. This optimiser has a starting learning rate of 10^{-3}, which decays from a factor of 10 after the validation loss does not improve for 10 epochs. Training is stopped once the learning rate decays below 10^{-8}. For training data, we have used a training/validation/test split of 300/100/100 for the 500 soccer games. With regards to dimensionality reduction, we have used both the learnable layer introduces with NetVLAD++, and the PCA dimensionality reduction supplied in the SoccerNet-v2 devkit.

3.9.2 Fusion layer

Since we decided to use the NetVLAD++ model *as is*, most of our experimentation with hyper-parameters will be done in the fusion layer. One prominent challenge with this layer is concerning how we should combine the predictions from our *Past*, *Present* and *Future* models. For some experiments, we consider a time-window (Tw) before and after each prediction in the *Present* model, formally defined as such: For each prediction in the *Present* model p, with a timestamp t, and a label c, if there is a prediction p', in the *Past* or *Future* model with a timestamp t' such that $t - (Tw/2) <= t' <= t + (Tw/2)$, with a label c' such that $c' \neq c$, we include p' in the list of final predictions. We therefore use our *Present* model as the basis for our final predictions, and include the predictions from the *Past* and *Future* models that are not already included in the *Present* model. To do this, we must find the optimal Tw. In the experiments where we used this fusion algorithm, we experimented with values of Tw, and used the value that yielded the best performance.

Throughout our experiments, a persisting challenge was that our final output after data fusion had duplicate predictions. What we mean by this is that within a short time span of seconds, there might be several predictions of the same class. In soccer, it is not likely that three goals occur within five seconds of the game. For some of our experiments, we therefore filtered our predictions on confidence score. To do this, we must find the confidence thresholds that gives us the best results. As explained in Section 3.5, our initial idea was to shift the labels with the event before or after for our *Past* and *Future* models (Figure 3.5) when training our models. In the fusion layer, we need to change the timestamp of the *Past* and *Future* predictions, so that it happens in the future or past. As an example for the *Future* model, if we have predicted a *Kick-off* in a frame where a *Goal* originally occurs, we have to change the timestamp to reflect that the *Kick-off* happens some time after the goal. For our early experiments, we used the average time between events (~25 seconds) for this. For the above example, our logic would be the following: *When our Future model has predicted a Kick-off on a Goal frame, that Kick-off will occur 25 seconds in the future*. As our experiments progressed, we wanted to experiment with this time-shifting parameter as well. We calculated some temporal statistics between relevant events for the *Past* and *Future* models, which can be seen in Table 3.7. Here we can see the average, median, minimum and maximum time between different events in the SoccerNet-v2 dataset. When a prediction from the *Past* or *Future* model was added to the final output of our model, we shifted the time corresponding to this table. For our *Future* model, this would mean that if we predicted a *Kick-off* we would add 55.7 or 41.5 seconds

Event relationships	Avg	Median	Min	Max
Future model relationships				
Ball out of play → Throw in, Clearance, Corner	17.4s	14.7s	0s	582.8s
Foul → Indirect free-kick, Direct free-kick, Yellow card, Red card, Penalty event	22.9s	17.1s	0s	302s
Offside → Indirect free-kick	21s	20s	0s	156s
Goal → Kick-off	55.7s	55.4s	17s	195s
Past model relationships				
Ball out of play ← Throw-in	35.1s	24.2s	0s	310s
Goal ← Kick-off	55.7s	41.5s	7.3s	205s
Foul, Offside ← Indirect free-kick	34.6s	24.3s	0s	323s
Ball out of play ← Clearance	37.3s	26.3s	0s	475s
Ball out of play ← Corner	25.4s	12.1s	0s	287s
Foul ← Direct free-kick	33.7s	20.2s	0s	246s
Foul ← Yellow card	63.5s	55.2s	2.7s	251s
Foul ← Red card	63.5s	108.2s	66s	147s

Table 3.7: Table showing temporal statistics between events that are related, in accordance to Figure 3.3 and Figure 3.4. These statistics are calculated by using the time in seconds from the timestamp of an event, until the timestamp of the next related event in the SoccerNet-v2 dataset.

to the timestamp depending on whether we were using the average or median time-shifting for that experiment. We also experimented with using a weighted median (80%, 90%, 110%) for some experiments. We also experimented with Tw (from Section 3.9.2) in an overlapping or non-overlapping sliding window algorithm, which is further explained in Section 4.6.

3.10 Evaluation metrics

To be able to correctly evaluate the performance achieved by our models and compare it to the baseline, we need to apply an evaluation

function that results in numerical values that are unambiguous and clear.

For the purposes of evaluating our model, we can redefine a multiclass classification problem (such as ours) to a binary one vs. all classification problem for each of our 17 classes. In classification, a true positive (TP) is when a model correctly predicts the presence of a class in an observation. A false positive (FP) is when a model incorrectly predicts the presence of a class in an observation. A true negative (TN) is when a model correctly predicts the absence of a class in an observation, and a false negative (FN) is when a model incorrectly predicts the absence of a class in an observation. In the context of evaluating on the SoccerNet-v2 dataset, we define a true positive as a correctly predicted class that is predicted with a timestamp within a tolerance δ, of the ground truth event. A false positive is a prediction that falsely predicts a class outside of a ground truth event (also padded with δ), a true negative correctly does not predict an event when there is none. A false negative is an instance where our model fails to predict an event of the class within the δ of where a ground truth event is happening.

It is often beneficial to use more than one evaluation metric to get a full and comprehensive picture of how our model is performing. For example, using the well-known evaluation metric accuracy (correct predictions / total number of predictions) in isolation, does not necessarily give a correct picture of our performance. Given a distribution of 90 positive samples and 10 negative samples, our model will get an accuracy of 90% even if it is only able to predict positive samples. This will not reflect the models inability to predict negative samples correctly. The precision of a machine learning model answers the question *How confident can we be that a positive prediction is correct?* and can be defined as such:

$$Precision = TP/(TP + FP)$$

Recall is another evaluation metric, that we can formulate as *How many of the positive observations is our model able to capture?* and can be defined as such:

$$Recall = TP/(TP + FN)$$

Recall and precision is often combined to form an F1 score, formally defined as:

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

We will first evaluate our models using a function introduced in SoccerNet, the Average-mAP function, which uses both precision and recall to measure the performance of the models. We have chosen to

use Average-mAP, as it is the function all models evaluated on the SoccerNet dataset are evaluated with, and it therefore makes it easy to compare our models to the performance of the baseline model.

3.10.1 Average-mAP

For action spotting, we have to correctly classify which event is taking place, as well as when the event is happening. This added dimension of time needs to be considered when evaluating our model. Since it is challenging (for both human and machine) to pick the exact frame where the event in the soccer video is happening, a tolerance δ is introduced, which ranges from 5 - 120 seconds. If a prediction falls within this tolerance of 2.5 - 60 seconds before or after the ground truth event, the prediction is considered a true positive. Average-mAP uses a greedy approach, where each ground truth observation is paired to its closest prediction. To evaluate the performance of our model, and calculate the Average-mAP, we do the following:

For a given class, we iterate over 200 confidence thresholds in the range 0 - 1. For each threshold, we calculate precision and recall. Furthermore, we calculate the average precision:

$$\textit{Average-Precision}(AP) = \sum_{t=1}^{T}(R_t - R_{t-1})P_t$$

Where R_t denotes the recall for a given threshold t, and P_t denotes the precision for a given threshold. This function calculates the space under the precision-recall curve from calculating precision and recall for all thresholds, by an 11 point approximation. This reduces the precision-recall curve to a single numerical value. We calculate the AP for all values of δ. Once we have calculated all AP values, we can calculate the Average-AP, which is a trapezoid approximation of the area under the curve of all AP values. The Average-mAP is then calculated by averaging the Average-AP across all 17 classes, resulting in a single numerical value to measure the general action spotting capabilities of a model across different tolerances of δ:

$$\textit{Average-mAP} = \frac{\sum_{c=1}^{C} \textit{Average-AP}_c}{C}$$

Figure 3.10 shows how different values of δ in the Average-mAP function results in different values for true positives, false positives, and false negatives. In this example, we can see that a greater δ yields better results.

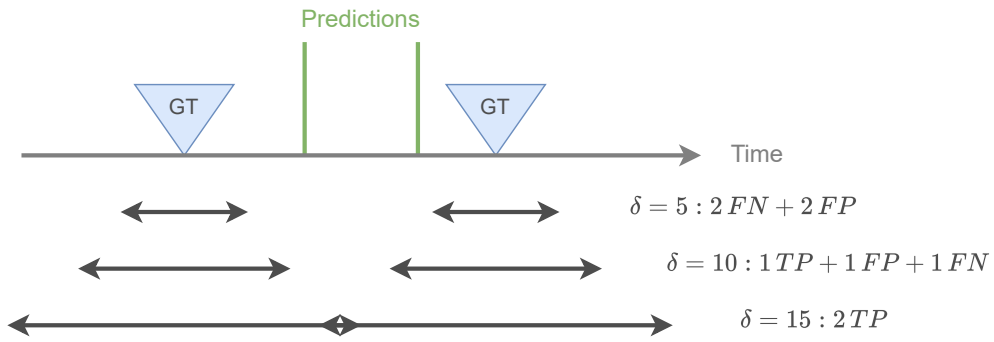


Figure 3.10: Figure showing how the Average-mAP evaluation function calculates true positives (TP), false positives (FP), and false negatives (FN) with different δ values. In this example, there are two annotations (GT), and two predictions. Recreated from [44].

3.10.2 Reflection on alternative evaluation methods

We feel bound to using the Average-mAP evaluation to be able to compare our experiments to previous work in this domain. We do however think that it is worth discussing alternative methods of evaluation, especially when considering real-life applications of these types of machine learning models. In a real-life scenario within the domain of soccer, there are several problems we can optimise a machine learning model for. One such problem is to classify all events throughout a game, with little or no regard to how accurate the predictions are in terms of timestamps. For statistics, it is not important that our model spots when the events occur, only that they do occur. For automatic highlight production on the other hand, it is important that the model is able to accurately spot both when and what action occurs. For most real-life applications, there will be a trade-off between some dimensions of performance that we need to consider: Temporal precision, classification accuracy, processing time, etc. For this reason, we do not think that the Average-mAP method accurately reflects performance in a real-life scenario. The Average-mAP considers all δ values from 5-60 seconds, and considers 200 different confidence thresholds between 0 and 1. After conducting the main parts of our experiments in Chapter 4, we will therefore evaluate our best model using precision, recall and F1 score for different confidence thresholds, and compare the results to the baseline model in the context of practical use cases. We will define two real-life scenarios in the soccer domain, for which we can apply an action spotting model, and reflect on the suitability of our model in these scenarios. For these evaluations and discussions, we will also consider the performance when spotting visible events where we deem it appropriate. For

some real-life applications, such as statistics (as mentioned above) the visibility of the event is not relevant. Also, it seems reasonable to reflect on how our work might improve the performance of an action spotting model as a whole.

3.10.3 Baseline model

As mentioned in Section 3.7, we will use the NetVLAD++ model [19] as the baseline. The overall Average-mAP scores achieved by this model can be seen in Table 3.8, while the classwise scores can be seen in Table 3.9. These Average-mAP scores are included in the SoccerNet-v2 devkit and will be used for comparisons when evaluating our models in Sections 4.2 - 4.6.

Model	Total Average-mAP	Average-mAP *visible*	Average-mAP *off-screen*
Baseline	53.23	59.24	34.37

Table 3.8: Overall performance of the baseline using the Average-mAP score. The performance is calculated for all actions, as well as visible and off-screen actions separately.

	Ball out of play	Throw-in	Foul	Ind. free-kick	Clearance	Shots on tar.	Shots off tar.	Corner	Substitution	Kick-off	Yellow card	Offside	Dir. free-kick	Goal	Penalty	Yel. to Red	Red card
All	79.1	61.2	72.8	69.1	37.7	38.5	40.6	56.8	70.3	68.8	63.8	44.4	56.4	79.9	56.4	3.2	6.0
Visible	86.1	73.1	73.5	76.2	40.2	38.7	40.9	69.3	70.6	72.1	64.6	39.6	65.1	80.6	60.8	25.1	30.7
Off-screen	0.0	55.3	0.0	28.7	11.5	4.4	0.4	44.7	64.8	60.7	35.9	51.7	3.0	73.6	12.4	0.0	0.0

Table 3.9: Average-mAP scores per class for NetVLAD++, which we will use as the baseline. The scores are split between classes and grouped by all, visible and off-screen events.

3.11 Summary

In this chapter, we have presented the SoccerNet-v2 dataset in detail with its limitations. We have established an intuition for how we believe a machine learning model can better understand the context

of a soccer game. We discovered the relationships between events in soccer, and from that defined three dimension of context in a soccer game. We have also presented how we can manipulate the SoccerNet-v2 dataset in order to train three different action spotting models on each of these contextual dimensions. Following this, we presented our new model architecture that combines a *Past*, *Present* and *Future* model to better predict off-screen events in soccer. We have established a baseline model from which we will compare our work to, and established that our experiments will be derived from this baseline model. We have also explained some of the theory behind how one can combine the output of several machine learning models. We have also presented methods for evaluating our models, so we can compare them to existing work within this domain. In the next chapter, we will present our models and results, experimenting with the proposed combined model architecture in the context of an action spotting model as a whole, but specifically in the context of spotting off-screen events. Towards the end of the next chapter we will reflect on our work and discuss possible shortcomings in our process.

Chapter 4

Experiments & Results

4.1 Introduction

We propose a new model architecture combining a *Past*, *Present* and *Future* model in order to better spot off-screen events in soccer. In Chapter 3, we presented a general idea for how an action spotting model can better understand the context of a soccer game. We discovered semantic relations between events in soccer, and defined three dimensions of context from these relations. From this, we presented a general label-shifting scheme, so that we can train three machine learning models on these dimensions of context. We also discussed a model architecture that combines these three models.

In this chapter, we will present our experiments and results. We have worked in an iterative way, and started with experiments at a point where we did not have as deep a knowledge about the domain and the dataset as we have now. As time went on, we tried to learn from each experiment, and improve our result in the next iteration. Our experiments will therefore be presented chronologically, where the findings from one will form the basis for the next. For each experiment, we will first explain our intuition and basis for the experiment. Then, we will present the overall results using the Average-mAP evaluation function and compare these results to the NetVLAD++ model we established as a baseline in Section 3.10.3. The results will also be considered on a class-by-class basis where we see fit to do so. We will present our results where our models were trained on the training/validation/test split (300/100/100 games), where the performance on the test split is the final value we use to compare the different models. We will analyse our results from the output of the Average-mAP function, as well as with visualisation tools to try to understand better why our models perform the way they do. We will also test our model in more practical use cases, where we evaluate the models using precision, recall and F1 score. We will analyse these

metrics individually and on a class-by-class basis where we deem it interesting to do so. We will also discuss the importance of precision, recall and F1 score and if it is reasonable to prioritise one over the other, in light of the use cases. Towards the end of this chapter, we will discuss some of the choices we have made throughout our experiments, and reflect on the performance of our models. Finally, we will do a process review and reflect on future work.

4.2 Model 1 - Baseline model as part of expert ensemble

As expected, we have seen that state-of-the-art models evaluated on the SoccerNet-v2 dataset perform worse when spotting off-screen versus visible events (Table 3.3). Our general idea, as detailed in Section 3.3, was to combine three expert models to improve this (Figure 3.6). For a given frame showing an event, one model would learn what event preceded it (*Past* model), one would learn what event is currently present in the frame (*Present* model) and one model would learn what event will follow in the future (*Future* model). Part of this idea is to see if each of these models can learn their specific contextual dimension (*Past*, *Present* or *Future*) better than our baseline. For our first experiment, we wanted to check this hypothesis by implementing a model that became an expert on spotting actions in the present.

For this model, we hypothesised that the *Present* model could achieve better results spotting visible events if we trained it using visible events only, ignoring the off-screen events. Figure 4.1 shows the difference between a *Throw-in* that is annotated as visible and a *Throw-in* annotated as off-screen. In Figure 4.1, we believe that teaching our model that a frame of one of the teams coaches (top image) is a *Throw-in*, will diminish its understanding of what a *Throw-in* looks like when it happens on-screen (bottom image). Furthermore, we believe that the difference between what is happening in the frame when an event occurs off-screen, and the typical visual traits of the actual event is so great, that there is no learnable connection between the off-screen event, and what is shown on the frame.

For this experiment, we modified a NetVLAD++ model to simply ignore all off-screen events during training. This was done by removing every annotation with the *visibility* attribute set to "not shown" from the training data. This model was trained with the standard settings that provided the best results for a NetVLAD++ model, as explained in Section 3.9.1, using NetVLAD++ pooling modules, a learning rate of 10^{-3} and a learned linear layer for dimensionality reduction. The learning rate plateaued after 81 epochs, stopping the training phase of

Figure 4.1: Two frames from a game in the SoccerNet-v2 dataset. Top frame shows the frame at the timestamp of a *Throw-in* happening off-screen and the bottom frame shows a *Throw-in* happening on-screen.

the model.

Table 4.1 shows the overall results of this model (Model 1) compared to our baseline. With regards to overall performance, Model 1 performs marginally better than the baseline on visible events only, and has a drop in performance for the total Average-mAP score compared to the baseline. Model 1 naturally performed worse than the baseline on off-screen events, since the model was not trained on any off-screen events. Figure 4.2 shows the classwise performance of Model 1 compared to our baseline when spotting visible actions. For *Throw-*

Model	Total Average-mAP	Average-mAP *visible*	Average-mAP *off-screen*
Baseline	53.23	59.24	34.37
Model 1	51.88	59.40	28.96

Table 4.1: Overall performance of Model 1 compared to the baseline using the Average-mAP score. The performance is calculated for all actions, as well as visible and off-screen actions separately.

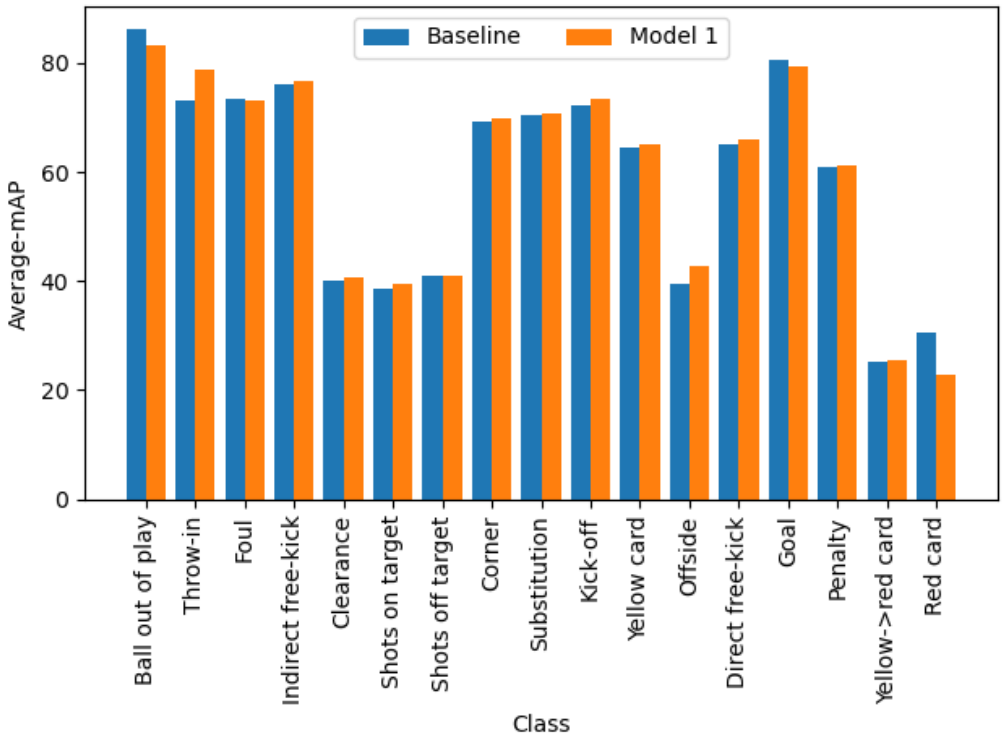


Figure 4.2: Classwise Average-mAP scores for Model 1 compared to the baseline. The Average-mAP scores are for visible events only. The raw numbers can be found in Table A.1.

in, which is the class with second most annotations in the dataset, we achieved a higher score than the baseline. Looking back at Table 3.2, we see that this class has a high number of off-screen events. Based on this, one might argue that our aforementioned theory has some merit. Using visible and off-screen events in training might confuse the model when it is learning how to spot visible actions, if there are a high number of annotations for off-screen events in the training data. For the classes *Foul*, *Indirect free-kick*, *Clearance*, *Shots on target*, *Shots off target*, *Corner*, *Substitution*, *Kick-off*, *Yellow card*, *Direct free-kick*, *Goal*, *Penalty* and *Yellow then red card* the difference in performance

is marginal. *Ball out of play* is interesting, since our model sees a performance decrease of about ~3 percentage points compared to the baseline. From Table 3.2, we see that this is the most frequent event-type, and that there are 1360 off-screen annotations. Considering our observation with the *Throw-in* class, one would think that our model would produce better results for this class as well, when we removed the off-screen annotations. For *Red card*, our model has a ~8 percentage points drop in performance. On first glance this might seem interesting, considering there is only one off-screen annotation of this class. Upon further inspection though, we do not feel that this result is very surprising. There are a total of 55 *Red card* annotations in the dataset, which makes for an inadequate amount of data to train a reliable classifier on in this context.

For our initial hypothesis to hold true overall, the classes with a high number of off-screen events should see an increase in performance, and the classes with no or a low number of off-screen events should stay relatively the same. This holds true for some classes, but not for others. In Figure 3.3 and Figure 3.4, we can see that events of classes *Shots on target* and *Substitution* cannot be inferred from either the *Past* or *Future* model. This means that if we were to implement this first model as our *Present* model, we couldn't utilise the expertise of any one of our models to predict off-screen events of these classes, since our *Present* model is not trained on off-screen events for these classes. Nevertheless, we chose to use this model as our *Present* model for the time being. We want to see how this model works together with the *Past* and *Future* models, which will be introduced in the next iteration.

4.3 Model 2 - First multimodal approach

As discussed in Section 3.4, we want to utilise the event relationships in soccer to spot off-screen events. More specifically, we want to see if an event currently happening in a broadcast, can be informed from the event happening before or after it. In order to train the *Past* and *Future* models, we created two new sets of annotations. For the *Past* model, we moved the *label* attribute from one annotation to the next annotation in the list. This is visualised in Figure 3.5. For the *Future* model, the *label* attribute was moved to the previous annotation in the list. The labels were shifted without any regard for which event is annotated and does not use the relationships identified in Section 3.4. The *Past* and *Future* models were trained using all the annotations in the training data. We did not remove off-screen annotations for either our *Past* or *Future* model. After training, we run the models on the test dataset, to get three separate prediction files for each game. The prediction files needed to be combined in order to evaluate the

performance of a combined model. Before combining the predictions, we had to change the timestamps of the predictions created by the *Past* and *Future* models. As stated in Section 3.9.2, we calculated the average time between all events in the dataset to be 25 seconds. We used this number to subtract 25 seconds from the *position* attribute of the predictions made by the *Past* model, and add 25 seconds to the *position* attribute of the predictions made by the *Future* model.

When merging the predictions in the fusion layer, we use the predictions made by the *Present* model as a foundation. For each prediction in the *Past* or *Future* model, we include that prediction if there is no prediction of the same class within a given time window T_w in the *Present* model. The predictions made by the *Present* model are used as a foundation since we believe these have a more exact timestamp. The timestamps of the predictions made by the *Past* and *Future* models are after all approximated and may not be as close to the ground truth of the actual event.

More detailed, the merging is done as follows. We start with the *Past* model and look at each prediction. We use the *position* attribute and the time window T_w, to create an interval that ranges from $position - T_w/2$ to $position + T_w/2$. Then, we check if there is a prediction from the *Present* model of the same class as the prediction we are looking at, and has a *position* attribute value that falls within the interval we created. If this is the case, we say that the predicted event from the *Past* model already exists in our model and look at the next prediction from the *Past* model. If no prediction from our *Present* model of the same class falls within the interval, we include the prediction from the *Past* model in our final output. These steps are repeated for the predictions made by the *Future* model. Algorithm 1 details this further.

We ran evaluation with T_w set to the values $[5, 10, 15, 20, 25, 30, 35, 40]$. Our overall results are shown in Table 4.2. The overall result rises as T_w gets larger, but are worse than the baseline when looking at total Average-mAP. Overall, our best result is when $T_w = 40$, where the total Average-mAP is 51.72%. This is 1.51 percentage points lower than the baseline model. For visible events only, the Average-mAP is higher than our baseline, but only by 0.06 percentage points. For off-screen events, our best model has an Average-mAP of 28.77%, which is 5.6 percentage points lower than the baseline score of 34.37%. Overall, our best model performs worse because of the off-screen events in the broadcast, which are the events that our *Past* and *Future* models are trying to predict.

Looking closer at the classwise results in Figure 4.3 for the off-screen events, we can see a couple of classes that Model 2 is much worse at predicting than the baseline. For the classes *Throw-in*, *Corner*, *Kick-off* and *Offside* the baseline outperforms our model by double digits.

Algorithm 1 Algorithm for including predictions made by the *Past* and *Future* models in our final fusion layer

```
for p in Past/Future model do
    position ← p["position"]
    interval ← [position − T_w/2, position + T_w/2]
    current_label ← p["label"]
    is_duplicate ← False
    for p2 in Present model do
        timestamp ← p2["position"]
        label ← p2["label"]
        if (label == current_label) AND timestamp in interval then
            is_duplicate ← True
            break
        end if
    end for
    if (is_duplicate == False) then
        Add p to final model
    end if
end for
```

Model	Total Average-mAP	Average-mAP *visible*	Average-mAP *off-screen*
$T_w = 5s$	46.46	53.40	26.02
$T_w = 10s$	48.57	55.98	26.44
$T_w = 15s$	50.77	58.36	27.79
$T_w = 20s$	51.48	59.10	28.45
$T_w = 25s$	51.60	59.22	28.60
$T_w = 30s$	51.63	59.25	28.61
$T_w = 35s$	51.68	59.28	28.67
$T_w = 40s$	51.72	59.30	28.77
Baseline	53.23	59.24	34.37

Table 4.2: Overall performance of Model 2 with different values for T_w, using the Average-mAP score. The baseline scores are added for comparison. The performance is calculated for all actions, as well as visible and off-screen actions separately.

First, we have *Throw-in*, where our model scores 12.2 percentage points (p.p.) lower than the baseline for off-screen events. The score for *Corner* is 15.89 p.p. lower, the score for *Kick-off* is 11.96 p.p. lower, and the score for *Offside* is 15.8 p.p. lower than the baseline. Our model does not outperform the baseline in any of the classes when considering off-screen events only. With Model 2, we tried to see if the

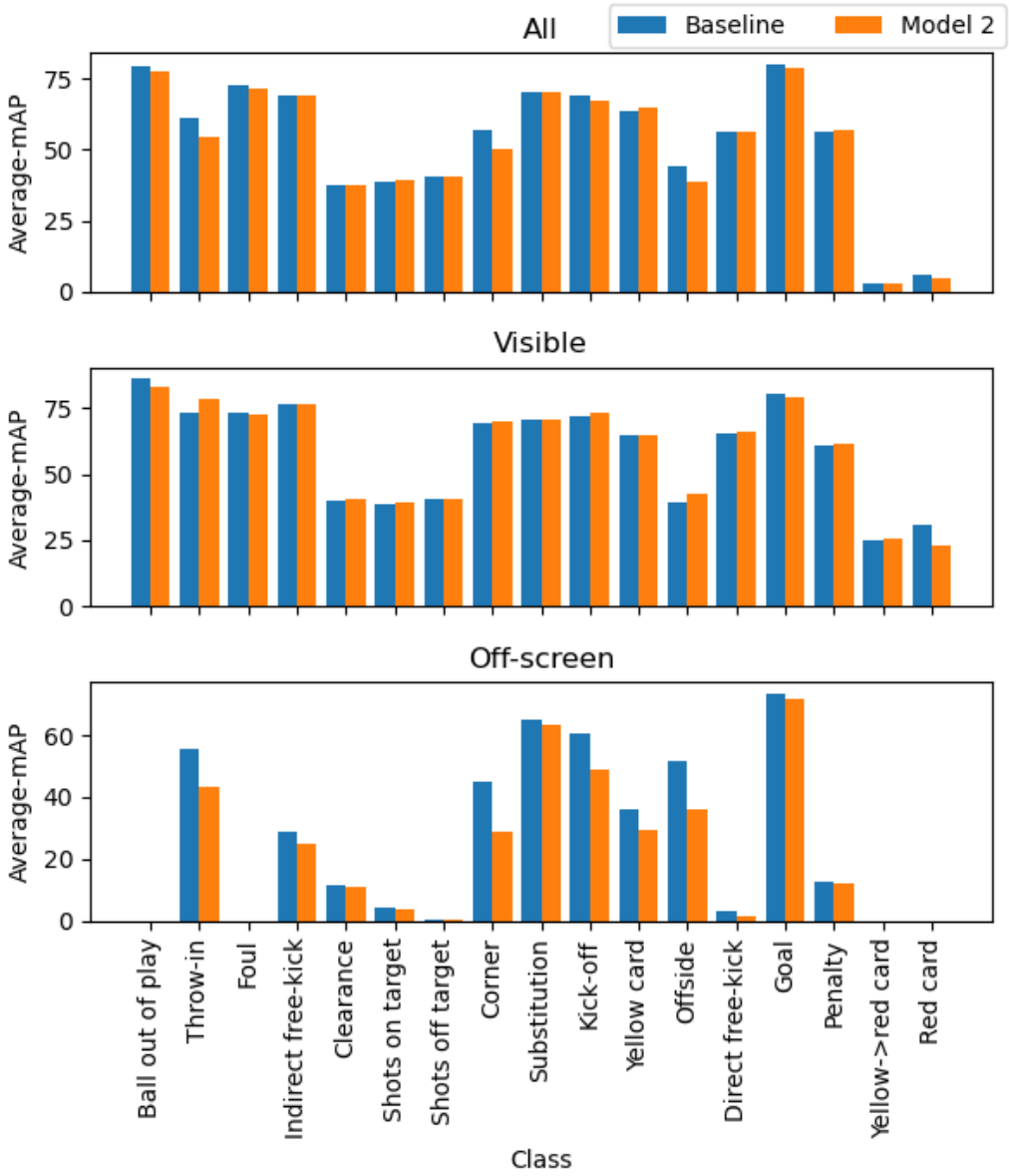


Figure 4.3: Classwise Average-mAP scores for Model 2 where $T_W = 40$, compared to the baseline. The figure is separated in three subplots for different visibility of the events. The classes on the x-axis are the same for all three subplots. The raw numbers can be found in Table B.1.

events before and after an event may be used to predict the event, but these results are not promising. A reason may be that our approach for shifting labels was too basic, since we just shifted the label with the nearest events label. The different events may not have any connection and might also be too far apart in time. Two succeeding events in our dataset may be a *Substitution* and then a *Foul* three minutes later. These events do not have any connection with each other.

Our idea for merging the predictions made by the different models using the time window, was that it would prevent the same event from being predicted twice. When looking at the final prediction list, we can see which model the predictions came from. When counting the predictions from the run with T_w set to 40, it turned out that 99.12% of the predictions came from the *Present* model. The *Past* and *Future* models contributed with only 0.5% and 0.38%, respectively. The numbers are shown in Table 4.3, which also shows that there were a total of 480,526 predictions from the combined model. Looking closer at the predictions, there are many predictions in close proximity time wise, from different classes and with low confidence scores. Since we used the *Present* model as a foundation, and only include predictions not already found within a time frame, few predictions will be included from the *Past* and *Future* models. As an example, a *Goal* prediction from the *Present* model with a confidence score of 0.001 would be included over a *Goal* prediction from the *Past* model with a confidence score of 0.9.

Model	***n* predictions**	%
Past	2,408	0.50
Present	476,314	99.12
Future	1,804	0.38
Total	*480.526*	*100.0*

Table 4.3: Contributions from the *Past*, *Present* and *Future* models to the final output of Model 2, with T_w = 40. 476,314 predictions, which is 99.12% of the total number of output predictions in Model 2, comes from the *Present* model.

Model 2 served as our first multimodal approach, where we had to create an algorithm for fusing our three models outputs together. We have seen that our results are quite poor compared to the baseline. With the current data fusion algorithm, we also see that the *Future* and *Past* model only contributes with 0.88% of the total final predictions of our model. We believe there are two main reasons for this poor performance. First off, this poor performance can be attributed to the way we shifted the labels for our *Past* and *Future* models. In the course of a game of soccer, not all events following each other will adhere to the relationships defined in Section 3.4. Secondly, we have used the average time between all events in all games in the SoccerNet-v2 dataset to shift the timestamp of events predicted by the *Past* and *Future* models. This will naturally create errors, since the time elapsed between different events in soccer varies by a great degree.

4.4 Model 3 - Refining time-shifting for predictions produced by the *Past* and *Future* models

Our second model yielded poor results, but gave us familiarity with how we could approach combining the *Past*, *Present* and *Future* models from a technical perspective, and started the process of uncovering some of the challenges and pain-points going forward. After experimenting with Model 2, it became evident for us that that there are especially two areas of improvement where we have an intuitive idea for how we can potentially progress.

First off, shifting every label with the one preceding/following it for the *Past* and *Future* models, did not work. As mentioned in Section 3.4, there are specific rules as to what event follows another, and for some events, we cannot inform the previous or following event from the current one at all. More importantly, we want the *Past* and *Future* models to learn and predict two different sets of classes. A *Throw-in* event is valuable for our *Past* model, since a *Throw-in* is always preceded by a *Ball out of play*. On the other hand, a *Throw-in* is not valuable for our *Future* model, since it is impossible to say what follows a *Throw-in* in open play. We therefore need to fine-tune these models, so they are not trained to learn events that are not relevant for their contextual dimension. Using our labels from Model 2, we filtered the labels for our *Past* and *Future* models. For our *Past* model, we only included annotations with labels of the following classes: *Throw-in*, *Kick-off*, *Indirect free-kick*, *Clearance*, *Corner*, *Direct free-kick*, *Yellow card*, *Red card*, *Penalty*, and *Yellow then red card*. For the *Future* model, we only included annotations with these labels: *Ball out of play*, *Foul*, *Offside*, *Goal*, *Shots off target*. As in our previous experiments, we trained these models with the standard hyper-parameters from NetVLAD++ [19]. The learning rate for both our *Past* and *Future* models plateaued after 71 epochs, where training was stopped. After training, we realised that even though we have strictly defined which annotations our models can train on, they will still try to spot actions that are of an *invalid* class for their respective model. Therefore, we needed to filter the predictions for our *Past* and *Future* models in our fusion layer. For our *Future* model, this means removing all predictions of classes *Ball out of play*, *Foul*, *Substitution*, *Shots on target*, etc. As discussed, our *Future* model cannot with certainty predict a *Ball out of play* from the events preceding it.

Our second challenge is related to time shifting. Our method for shifting the timestamp of the events is based on the average time between events in soccer (~25 seconds) and will need to be fine-tuned. Recalling Table 3.7, we see that the average and median time between

different events are quite different. Calculated over all annotations in the training set, we hypothesise that these values will more accurately reflect the time between different events. In the fusion layer, we therefore shift the timestamp of our prediction according to this table. For each prediction in our *Past* model, we map the class of that prediction to the corresponding average or median time between an event of that class and preceding events, and subtract this value from the *timestamp* of our predictions. For our *Future* model, we add this value to the *timestamp* of our predictions. We experimented with using both the average, and median time between different events, and found there was little difference between them in terms of performance. Our intuition is that median time is more suited, since a lot of unexpected and irregular things can happen between events in soccer, such as injuries, arguing between players, crowds rushing the field etc. Our thinking is that these things can influence the average time between events severely, and therefore using the median time can give a more general representation of how much time passes between two given events. In Table 3.7, there is also a large difference between the shortest and longest time between the events. The presence of outliers might also skew the average time, and not reflect the most suitable time between events. For these reasons, we decided to use the median time in our fusion layer.

With Model 2, we discovered that when we were fusing the outputs from our three models (*Past*, *Present*, *Future*) as in Algorithm 1, the predictions from our *Past* and *Future* models were mostly ignored. With Model 3, we used a much simpler method for combining our input streams. We simply concatenated all three models predictions, without any filtering algorithm. For this reason, we did not need to tune our time window parameter Tw, as discussed in Section 3.9.2. The overall results were significantly poorer than that of our baseline, as can be seen in Table 4.4.

Model	Total Average-mAP	Average-mAP *visible*	Average-mAP *off-screen*
Baseline	53.23	59.24	34.37
Model 3	46.14	52.53	24.89

Table 4.4: Overall performance of Model 3 compared to the baseline using the Average-mAP score. The performance is calculated for all events, visible events, and off-screen events separately.

Figure 4.4 shows that our model has the biggest drop in performance compared to the baseline when spotting *Throw-in* for all visibility metrics. When looking at the *Throw-in* predictions in Figure 4.5,

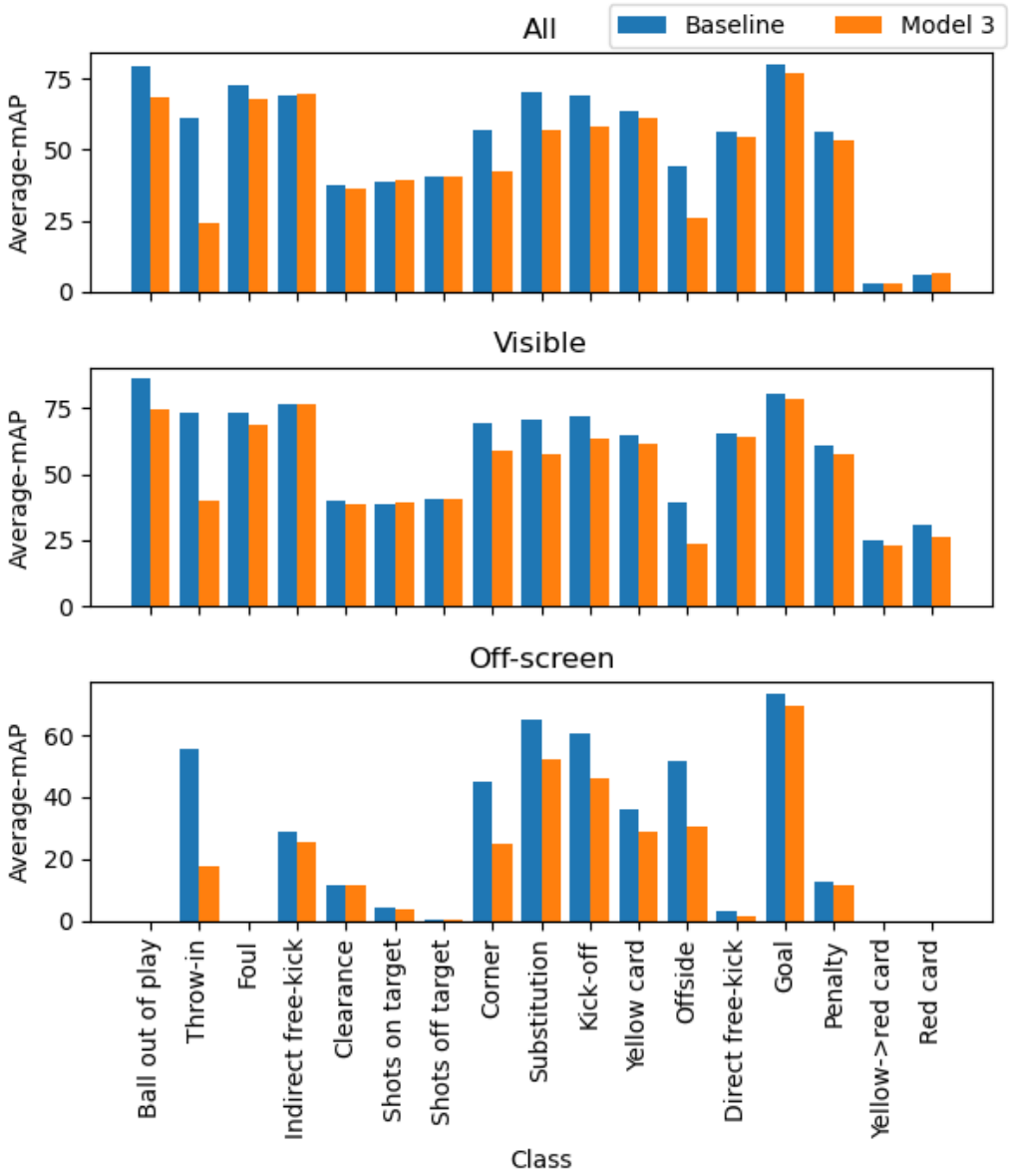


Figure 4.4: Classwise Average-mAP scores for Model 3, compared to the baseline. The figure is separated in three subplots for different visibility of the events. The classes on the x-axis is the same for all three subplots. The raw numbers can be found in Table C.1.

we can see that Model 3 contains quite a few more predictions than our baseline model. Considering the Average-mAP evaluation method, Model 3 will consequentially produce more false positives. We can also see that Model 3, for the most part, is more imprecise than the baseline.

In Figure 4.5, we have a visible *Throw-in* event temporally annotated just before the 35:00 timestamp, indicated by a blue vertical bar. In the subplot for the baseline predictions, we can see that a

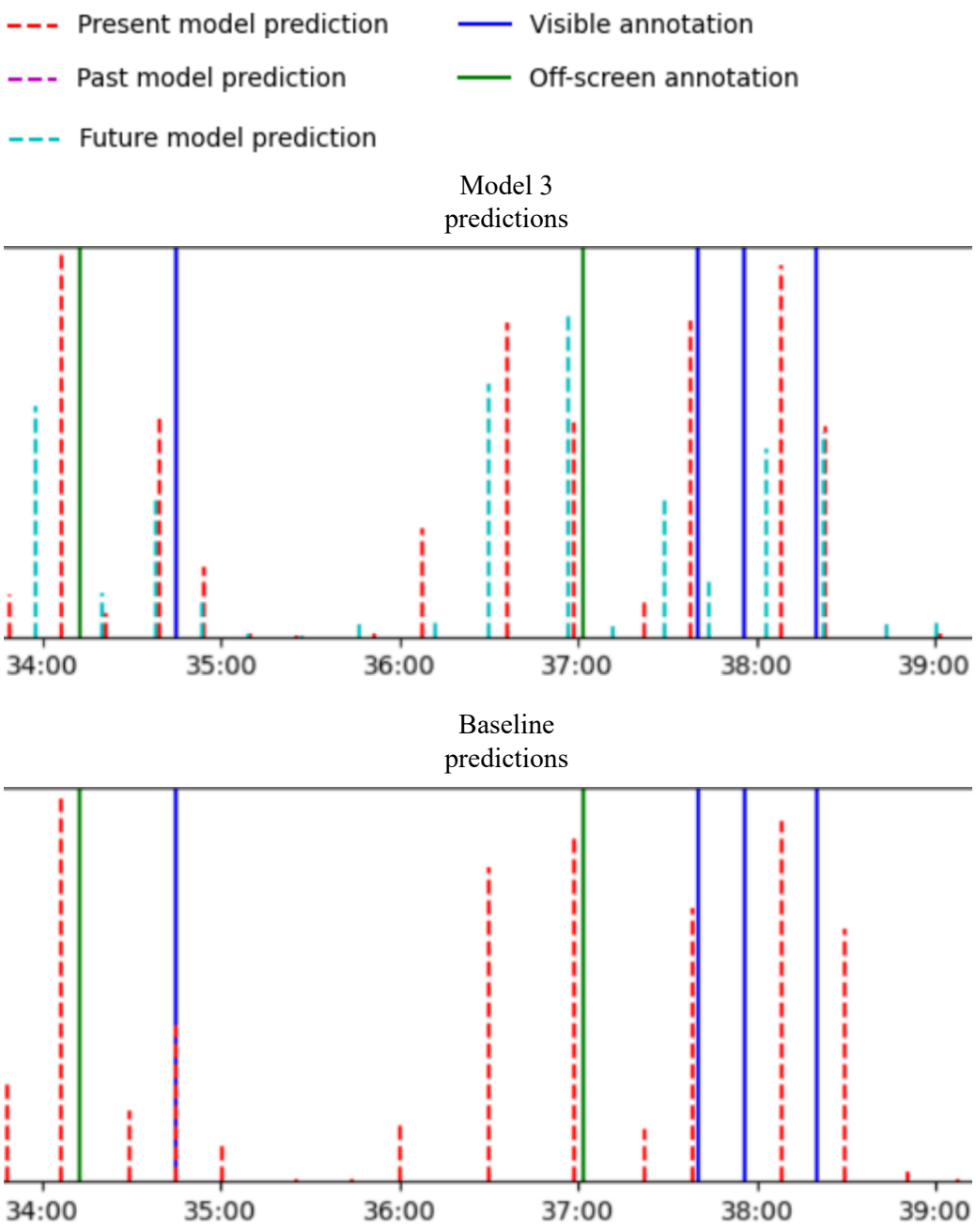


Figure 4.5: Visualising *Throw-in* predictions from Model 3 and the baseline for a game in the SoccerNet-v2 dataset, during a 5-minute period. The labels on the x-axis are timestamps from the game (*mm:ss*). The height of the dotted bars are determined by the confidence score of the prediction. The top subplot shows the predictions of Model 3, with predictions from the *Past*, *Present* and *Future* models, while the bottom subplot shows predictions from the baseline.

prediction for that *Throw-in* event is temporally located at the correct timestamp. In the subplot for Model 3, a prediction for the event is

close to the annotation, but not as precise as the prediction from the baseline.

We do see some promise in our work, since our *Future* model (teal dotted line, top subplot) has a prediction that is not too far off from the off-screen ground truth event temporally located around the 37:00 timestamp. Overall, it seems that Model 3 produces an abundance of false positives when spotting actions. During a period of four minutes of open play in the same game as in Figure 4.5, the baseline model predicted three *Corner* events, where there were none, while Model 3 predicted seven *Corner* events.

In this experiment, we changed our label-shifting scheme before training to account for the relationships presented in Figures 3.3 and 3.4. This was done so that the *Past* and *Future* models are only trained on events that are relevant for their specific contextual dimension. We also disregarded our fusion algorithm, and simply concatenated the predictions made by all of our three models. We also shifted the timestamp of the predictions from our *Past* and *Future* models with the median time between events. The result of this was a poorer performance than Model 2, and a higher number of predictions than our baseline model produces. We suspected that these changes created an abundance of false positives in the predictions made by our model.

4.5 Models 4.1, 4.2, 4.3 - Modifying label-shifting scheme

Following our experiment with Model 3, there were several areas fit for improvement. In Figure 3.3 and Figure 3.4, we established the relationships between different classes of events in soccer. As suspected in Section 3.4, after investigating the annotations in the training data, we observed that the events in these relationships do not always follow each other directly. For example, a certain type of *Ball out of play* will always lead to a *Throw-in*, but there can be events happening in-between. A *Ball out of play* event can be followed by a *Red card*, a *Yellow card* and a *Substitution* before we finally get a *Throw-in*. This is perfectly possible in the game of soccer, since there might be an argument between players and the referee in the time between the *Ball out of play* and *Throw-in*, leading to the cards shown. This also prolongs the time between the *Ball out of play* and *Throw-in* events, even though they are still related. If we were to use the label-shifting scheme introduced in Figure 3.5 on the aforementioned example, a *Substitution* event would be mapped to a *Throw-in* label in our *Future* model. In this experiment, we introduced new logic for shifting our labels. For our respective *Past* and *Future* models,

we iterated through all the original annotations. If the label L is appropriate for the model (e.g. *Ball out of play* for the *Future* model, or *Kick-off* for the *Past* model), we checked the *two* following or preceding annotations (depending on if we are working with the *Future* or *Past* model), and see if we found the annotation with the label connected to L as defined in Section 3.4. In the preceding example for our *Future* model, this would be finding a *Throw-in*, *Clearance* or *Corner* label in one of the two annotations following the *Ball out of play* label L, or a *Goal* label within the two annotations before a *Kick-off* for our *Past* model. If this condition was upheld, we replaced L with the label of the appropriate annotation. If the condition is not upheld, we ignored the annotation with the label L. There can of course be more than two events happening in between the two related events. We assumed that this situation is less common, and since we were using the median time between events (as presented in Table 3.7), we believed that the time elapsed between the related events in these situation exceed the median time, so we discarded them from the training set. Another adjustment that was done for this model, was that we wanted to train our *Past* and *Future* models on visible events only. The reasoning behind this, is the following: If we want our *Future* model to use a *Ball out of play* frame to learn that the next event is for example a *Throw-in*, that *Ball out of play* event needs to be visible. This follows our intuition from Model 1. We therefore discarded all off-screen annotations from the training data used by our new *Past* and *Future* models.

Figure 4.6 shows the performance of this model (Model 4.1) after training. As in our previous experiments, all other NetVLAD++ hyperparameters remained unchanged. As can be seen, the results are for the most part, comparable, or worse to those of Model 3. Looking at Figure 4.7, it is challenging to draw any conclusions. Comparing the predictions produced by the models for a 5-minute period shows that predictions from both models are fairly equal.

We wanted to make some further modifications to this model. Recalling that our *Present* model has a lower performance than our baseline (Table 4.1), we exchanged our *Present* model for an out-of-the-box NetVLAD++ model, trained on both visible and off-screen annotations. Table 4.5 shows the performance of this model (Model 4.2), compared to Model 3. The performance has increased considerably for off-screen events, indicating that our *Present* model has a considerable influence on our performance when spotting off-screen events.

In Figure 4.8, a classwise comparison with Model 3 shows some interesting results. When we looked at the predictions for the classes with the most improvement in performance (*Throw-in*, *Offside*, *Corner* etc.) we got surprisingly little visual cues as to how this improvement in performance took place. Like with Model 4.1, the predictions look

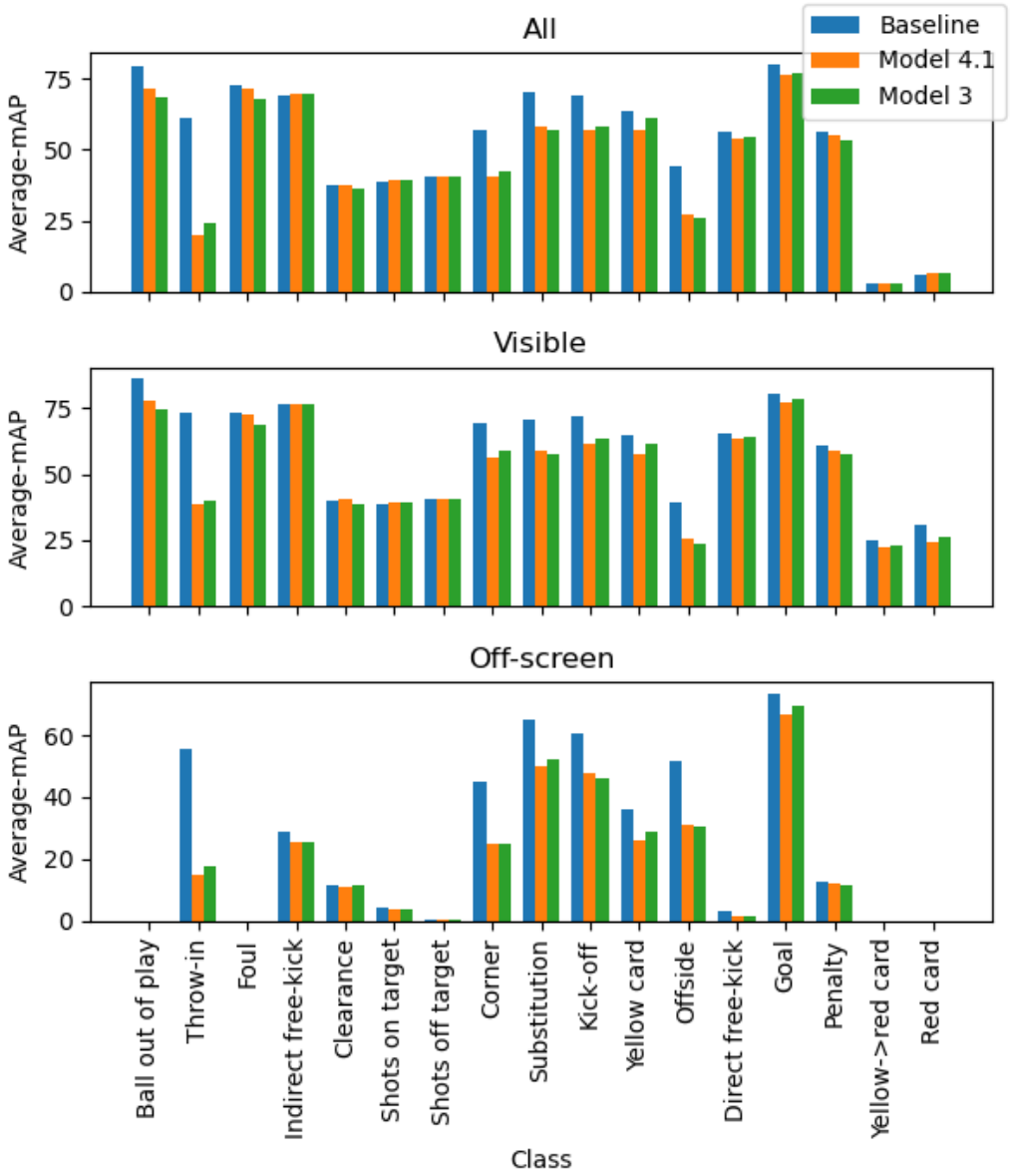


Figure 4.6: Classwise Average-mAP scores for Model 4.1, compared to Model 3 and the baseline. The figure is separated in three subplots for different visibility of the events. The classes on the x-axis are the same for all three subplots. The raw numbers can be found in Table D.1.

very similar between Model 4.2 and Model 3.

Figure 4.9 shows how Model 4.2 compares to Model 3 when predicting *Kick-off* events during a 5-minute period. An off-screen *Kick-off* (green line) event is temporally located around the 03:00 minute mark. The *Present* model in Model 4.2 (red dotted line) is able to spot this more precisely than Model 3. This seems reasonable as the *Present* model in Model 4.2 is now an out-of-the-box NetVLAD++

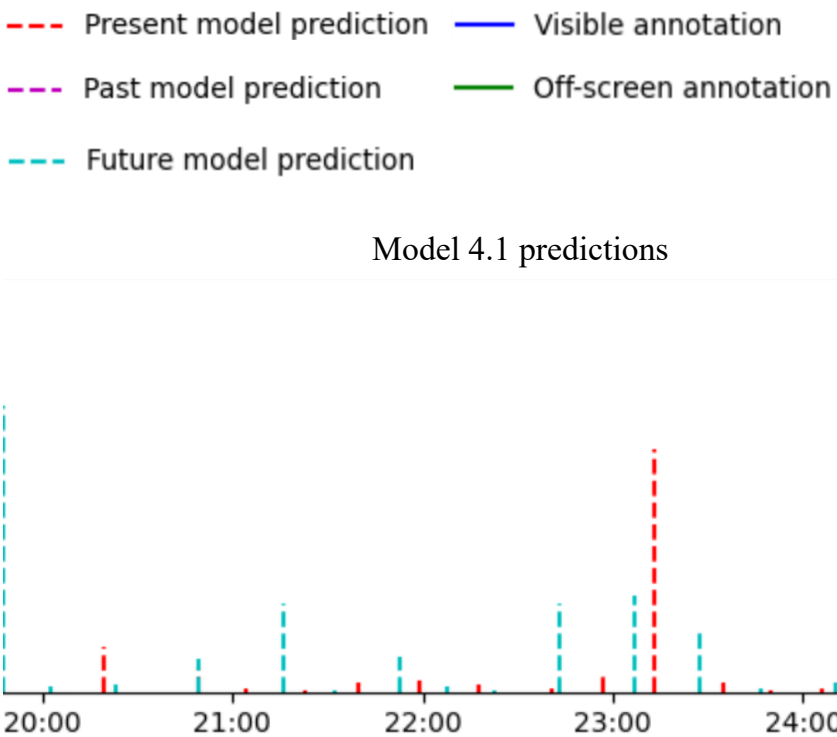


Figure 4.7: Visualising *Throw-in* predictions from Model 4.1 and Model 3 for a game in the SoccerNet-v2 dataset, during a 5-minute period. The labels on the x-axis are timestamps from the game (*mm:ss*). The height of the dotted bars are determined by the confidence score of the prediction. The top subplot shows the predictions from Model 4.1, with predictions from the *Present* and *Future* models, while the bottom subplot shows predictions from Model 3, with predictions from the *Present* and *Future* models.

Model	Total Average-mAP	Average-mAP *visible*	Average-mAP *off-screen*
Model 4.2	47.9	53.7	29.7
Model 3	46.1	52.5	24.9
Baseline	53.2	59.2	34.4

Table 4.5: Overall performance of Model 4.2 compared to Model 3 and the baseline using the Average-mAP score. The performance is calculated for all actions, as well as visible and off-screen actions separately.

model, trained on both visible and off-screen events. It performs better than the *Present* model in Model 3, which is worse at spotting off-screen events, since it has not been trained on off-screen annotations. This is, of course a single example, but one might think that this is the case for a larger number of events. When comparing Table 4.6 to Table 4.3, our *Past* and *Future* models now contributes more to the overall number of predictions in Model 4.2 after data fusion. However, concatenating all our predictions made by the *Past* and *Future* models does reduce our overall performance compared to the baseline, and the way we fuse these predictions together is a potential area of improvement.

Model	***n* predictions**	*%*
Past	111,924	12.9
Present	476,314	54.9
Future	279,310	32.2
Total	*867,548*	*100.0*

Table 4.6: Contributions from the *Past*, *Present* and *Future* models to the final output of Model 4.2. 54.9% of the total number of predictions in the output of Model 4.2 comes from the *Present* model.

For Model 4.1 and Model 4.2, we made two assumptions: The first being that our *Past*/*Future* models should only be trained on visible annotations. This makes sense from a human perspective, but the results from Section 4.2, and replacing our *Present* model for an out-of-the-box NetVLAD++ model in Model 4.2, suggests otherwise. The second assumption we made, was that related events more often than not have a maximum of two other events between them. For our last iteration of this experiment (Model 4.3), we challenged these assumptions. Before training our *Past* and *Future* models, we adjusted our label-shifting scheme so that we included off-screen annotations,

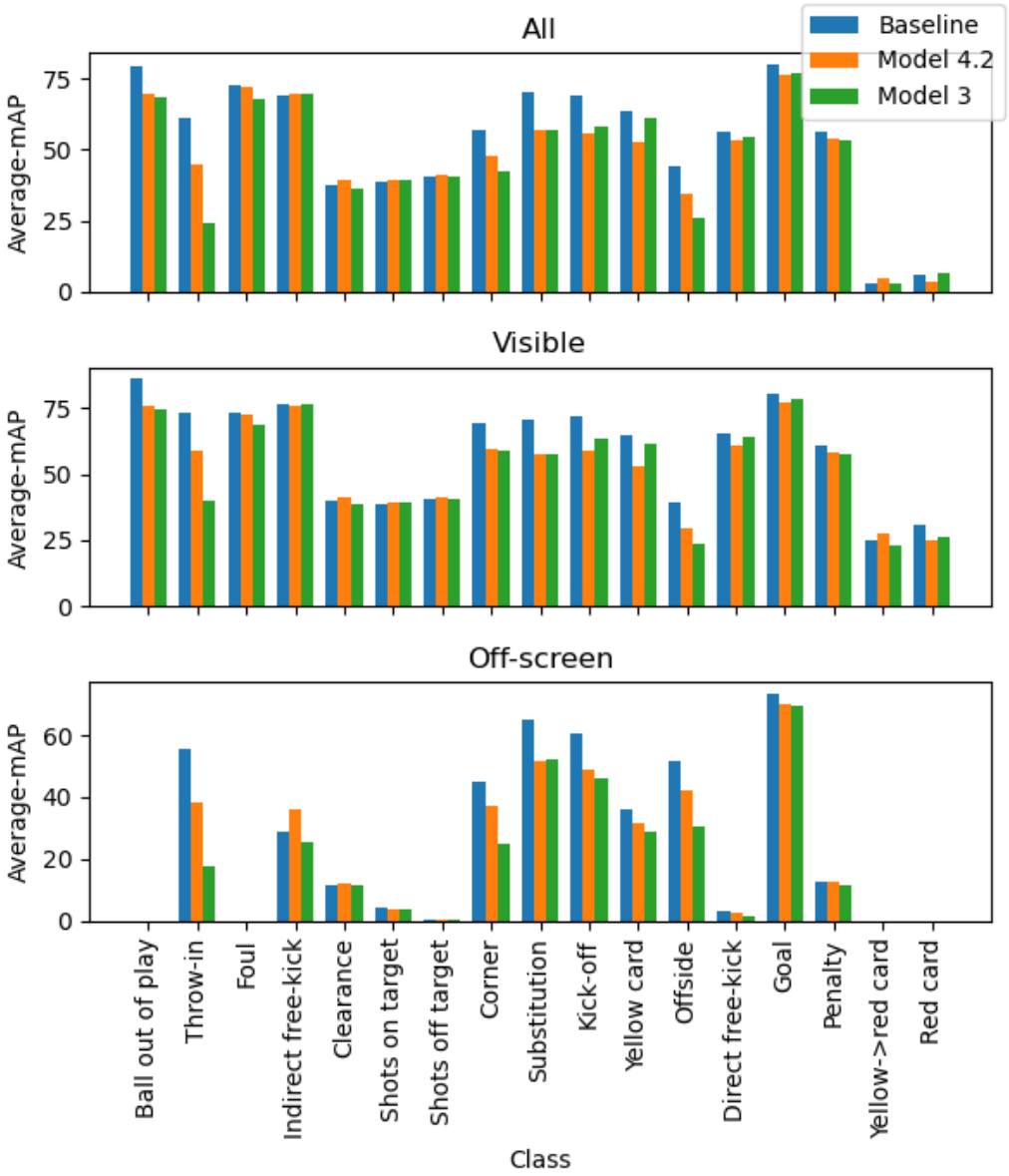


Figure 4.8: Classwise Average-mAP scores for Model 4.2, compared to Model 3 and the baseline. The figure is separated in three subplots for different visibility of the events. The classes on the x-axis are the same for all three subplots. The raw numbers can be found in Table E.1.

and increased the number of events we look at before and after our current event to 10. This means that we consider the situations where there are 3, 4, ... 10 events between the two related events. Table 4.7 shows a slight improvement in performance, compared to Model 4.2. Figure 4.10 shows that *Throw-in*, *Yellow card* and *Offside* are the classes with the biggest increase in performance when spotting off-screen events. These are all classes with a relatively high frequency of

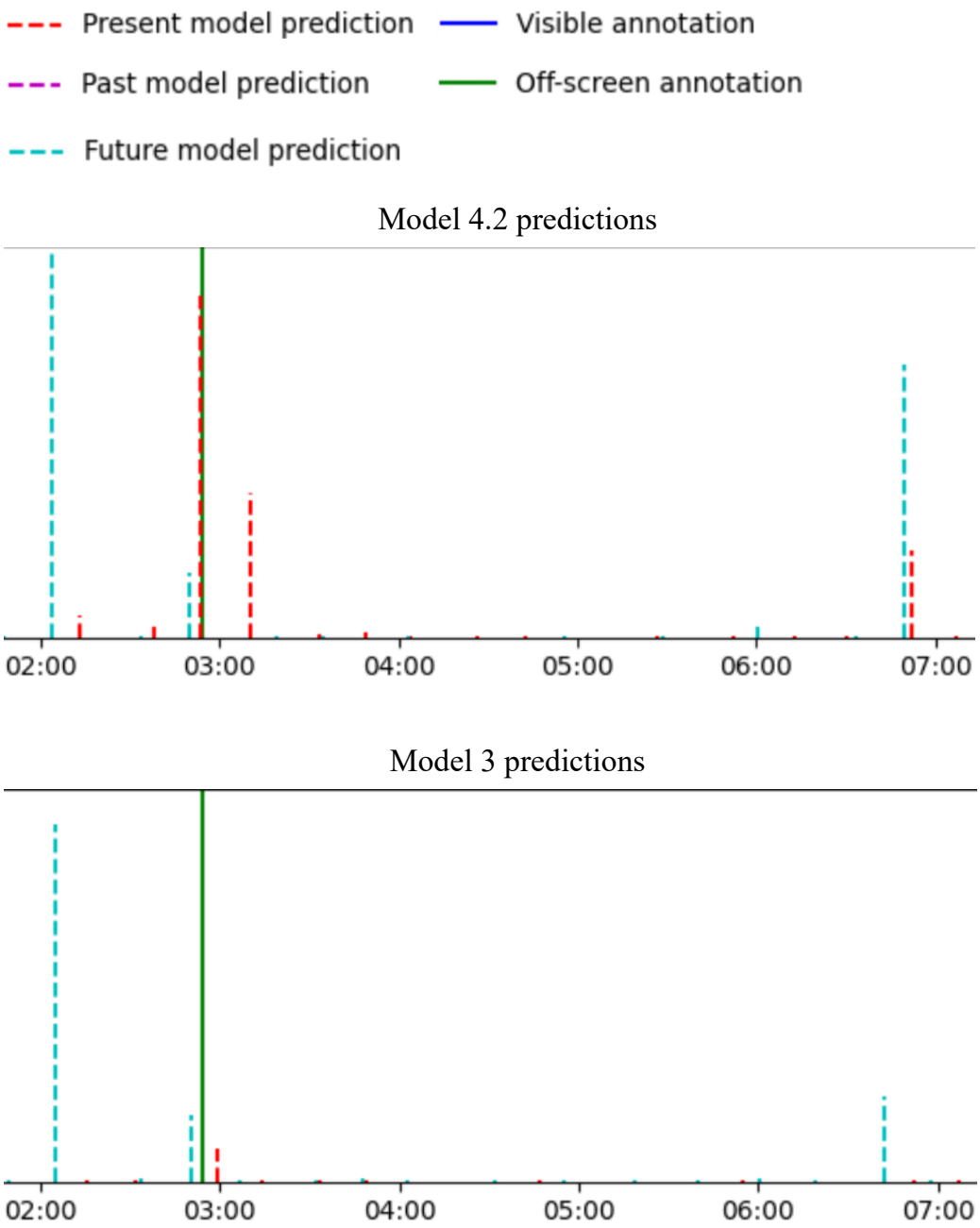


Figure 4.9: Visualising *Kick-off* predictions from Model 4.2 and Model 3 for a game in the SoccerNet-v2 dataset, during a 5-minute period. The labels on the x-axis are timestamps from the game (*mm:ss*). The height of the dotted bars are determined by the confidence score of the prediction. The top subplot shows the predictions from Model 4.2, with predictions from the *Present* and *Future* models, while the bottom subplot shows predictions from Model 3, with predictions from the *Present* and *Future* models.

Model	Total Average-mAP	Average-mAP *visible*	Average-mAP *off-screen*
Model 4.3	48.5	54.3	30.3
Model 4.2	47.9	53.7	29.7
Baseline	53.2	59.2	34.4

Table 4.7: Overall performance of Model 4.3 compared to Model 4.2 and the baseline using the Average-mAP score. The performance is calculated for all actions, as well as visible and off-screen actions separately.

off-screen events. Model 4.3 has the biggest drop in performance when spotting *Goal*, which is a class with only one off-screen event in the dataset.

Table 4.8 shows that the contributions from each of the *Past*, *Present* and *Future* models in Model 4.3 has the same distribution as in Model 4.2, although there are a few more predictions made by the *Past* and *Future* models. We used Model 4.3 as the basis for our next experiments.

Model	***n* predictions**	%
Past	112,079	12.9
Present	476,161	54.9
Future	279,642	32.2
Total	*867,882*	*100.0*

Table 4.8: Contributions from the *Past*, *Present* and *Future* models to the final output of Model 4.3. 54.9% of the total number of predictions in the output of Model 4.3 comes from the *Present* model.

For Models 4.1, 4.2 and 4.3, we were able to get further confirmation that training a model on visible events only, does not improve its ability to spot the events that it is trained on. Rather, it seems to decrease the performance. By replacing our *Present* model from Section 4.2 with an out-of-the-box NetVLAD++ model, we also learned that a *Present* model trained on both visible and off-screen data increases overall performance of our model. When shifting the labels for our *Past* and *Future* models, we have learned that including the relevant events that have a connection to the 10 previous or following events gives us a small improvement in performance, compared to when we only look at the two previous or following events. Model 4.3 still produces too many predictions, since we are concatenating the output from our

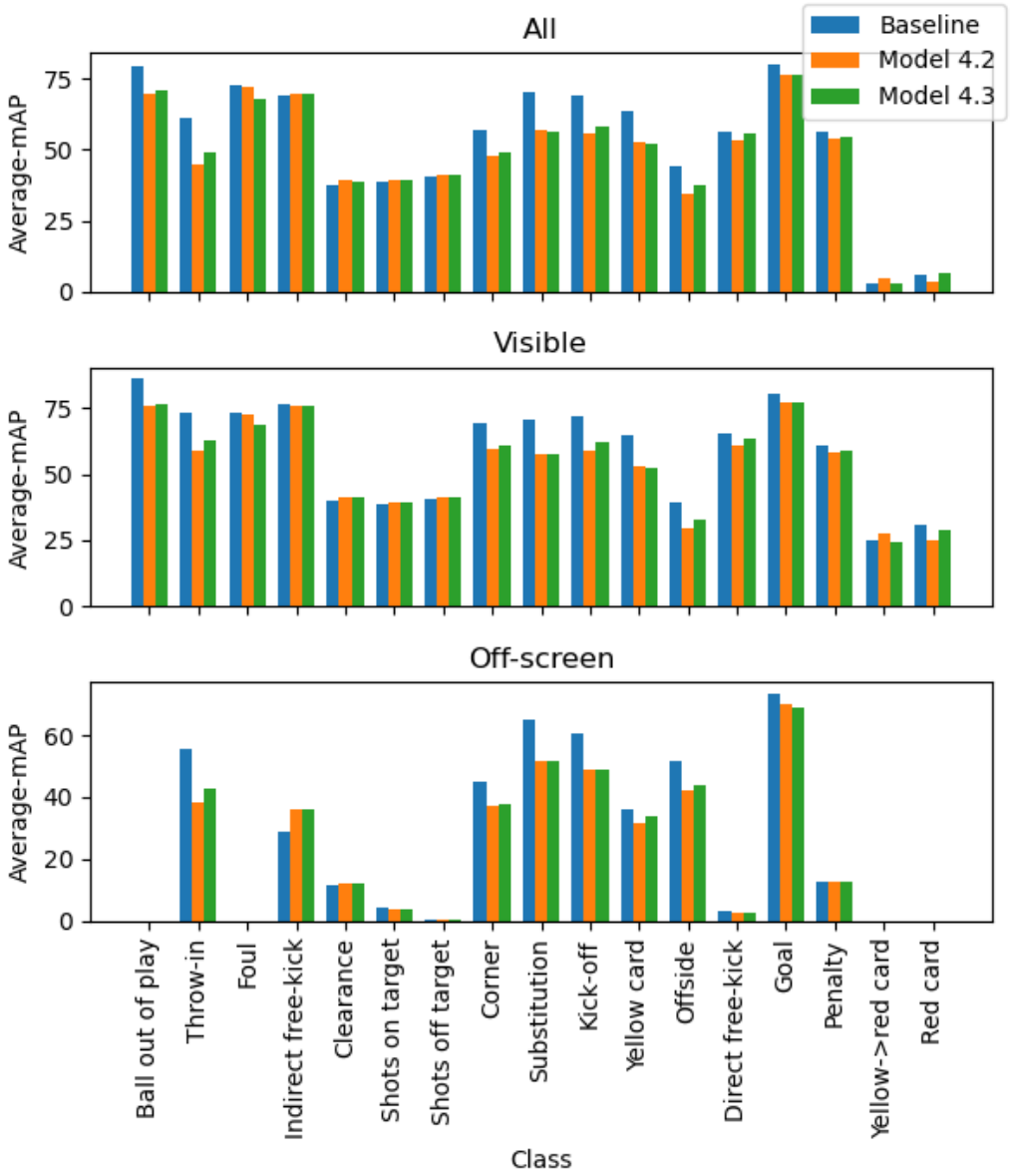


Figure 4.10: Classwise Average-mAP scores for Model 4.3, compared to Model 4.2 and the baseline. The figure is separated in three subplots for different visibility of the events. The classes on the x-axis are the same for all three subplots. The raw numbers can be found in Table F.1.

Past, *Present* and *Future* models. In the next experiment, we will try to create a data fusion algorithm which is improved from the one in Section 4.3, so that we do not include too many predictions, but at the same time utilise the predictions made by our *Past* and *Future* models.

4.6 Models 5.1 & 5.2 - Improving the data fusion algorithm

As we started working on a better data fusion algorithm, we discovered that our *Past* model exhibited some peculiar behaviour. When inspecting the predictions (after training, but before our fusion layer), we discovered that the *Past* model had predictions with a negative *position* value. In practice, this means that the model is predicting events temporally located before the game has started. We believe that this comes as a result of our *Past* model learning to predict a *Goal* event in the past, when it sees a *Kick-off* event. Since every half of every game starts with a *Kick-off*, a plausible consequence might be that our *Past* model predicts *Goal* events that happen before the first *Kick-off* has been performed. We filtered out all predictions that had a negative *position* value. This reduced the total number of predictions made by the *Past* model after data fusion with ~1300 predictions, and increased our action spotting results, as can be seen from Model 5.1 in Table 4.9.

Model	Total Average-mAP	Average-mAP *visible*	Average-mAP *off-screen*
Model 5.1	50.8	56.4	33.7
Model 4.3	48.5	54.3	30.3
Baseline	53.2	59.2	34.4

Table 4.9: Overall performance of Model 5.1 compared to Model 4.3 and the baseline using the Average-mAP score. In Model 5.1, predictions with a negative timestamp value was removed. Predictions with a negative value for the timestamp are predictions of events taking place before the game has started, which is not possible. The performance is calculated for all actions, as well as visible and off-screen actions separately.

With this issue resolved, we continued working on improving the fusion layer. We theorised that simply concatenating our three models predictions creates too many false positives. Within a given time frame, we wanted to keep one prediction of each class that had the highest confidence score, as to rely on the expertise of the individual *Past*, *Present* and *Future* models. We implemented this using a non-overlapping (Figure 4.11) and overlapping (Figure 4.12) sliding window algorithm with a variable window-size of t.

These algorithms slides a window of size = t across the temporal dimension of the game, and filters out all predictions for each class that does not have the highest confidence score of the predictions within the

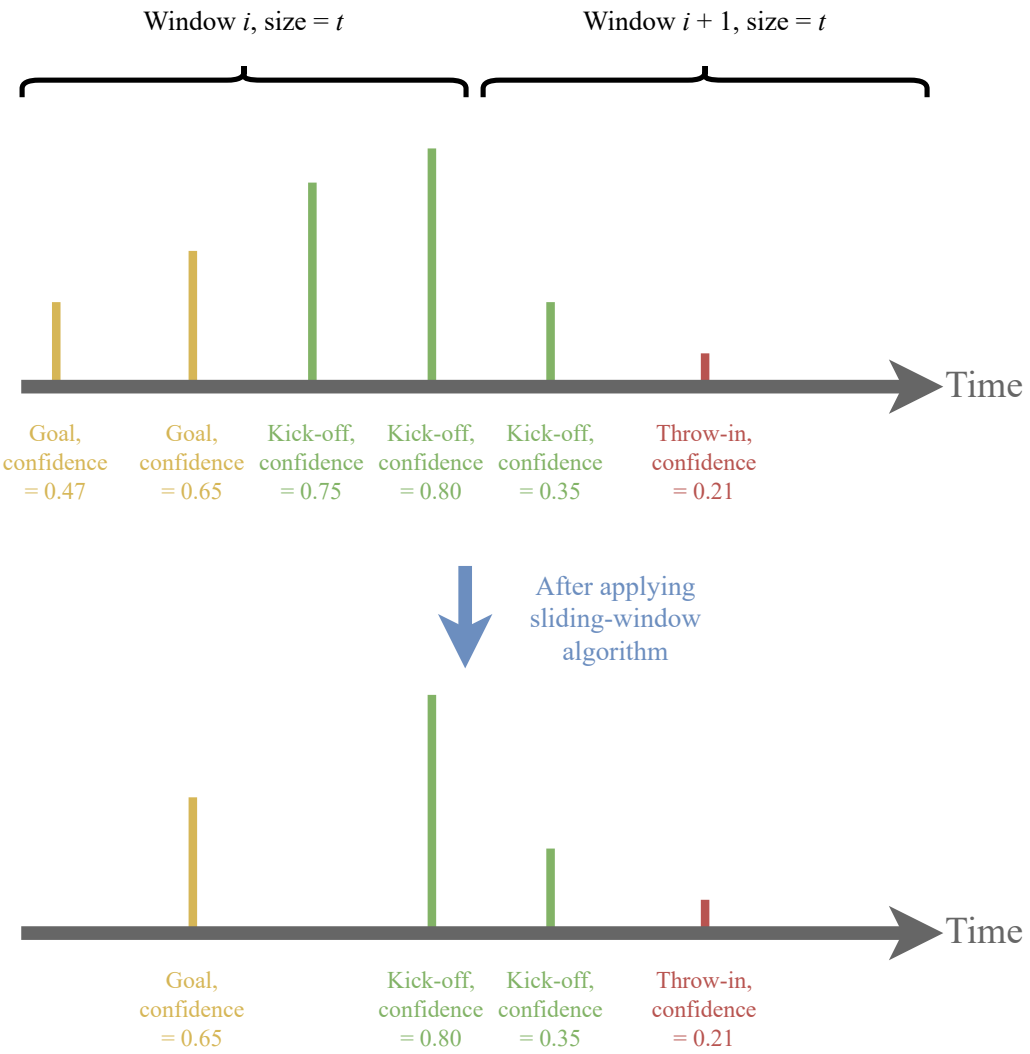


Figure 4.11: Non-overlapping sliding window fusion algorithm. The topmost figure shows *Goal*, *Kick-off* and *Throw-in* predictions before applying the algorithm, the bottom figure below the blue arrow shows the same predictions after filtering.

window. The difference between them is that the start and end of each window is configured differently. For the non-overlapping approach in Figure 4.11, the start of the window is right after the preceding window ends. For the overlapping approach, the start and end of each window, is defined as $position \pm t/2$ for each prediction, effectively creating one window per prediction in the model. In terms of efficiency, the non-overlapping methods requires far fewer calculations and windows, and is therefore more efficient. The reason for implementing a non-overlapping and overlapping approach, is because we were concerned that two predictions of the same class at the end of one window, and at the start of the next, would not be filtered against each other in the

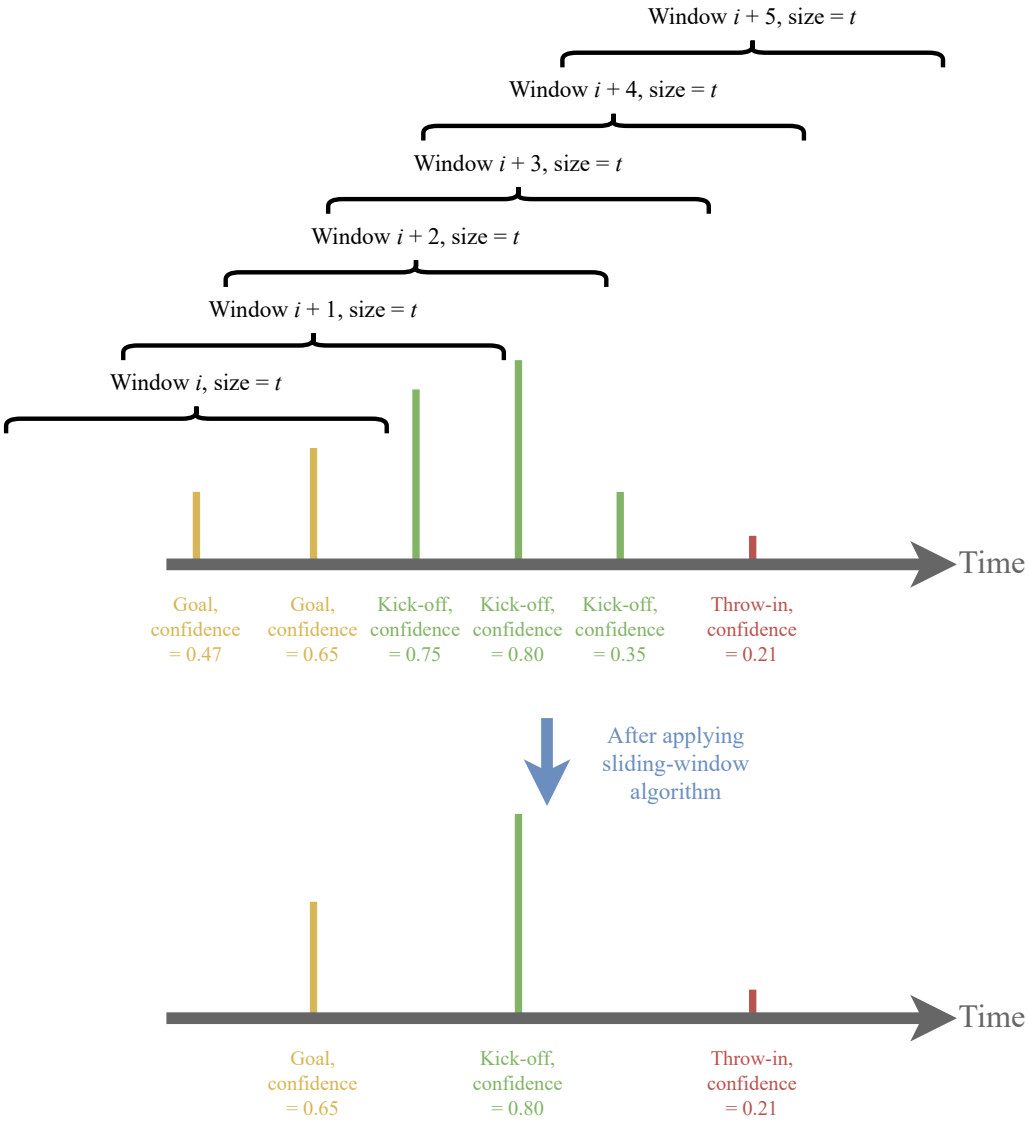


Figure 4.12: Overlapping sliding window fusion algorithm. The topmost figure shows *Goal*, *Kick-off* and *Throw-in* predictions before applying the algorithm, the bottom figure below the blue arrow shows the same predictions after filtering.

non-overlapping approach. In Figure 4.11, we can see that two of the three *Kick-off* predictions remain after the filtering. Realistically, two *Kick-off* events will not happen that frequently, but since each window does not overlap, this might happen. In Figure 4.12, we can see that the same three *Kick-off* events are filtered down to one which is intuitively more correct. We experimented with different values of t, ranging from 1 second to 40 seconds, naming this Model 5.2. Table 4.10 shows an overview of the best performing results. Despite our suspicions, our non-overlapping algorithm yielded the best results with t = 15 seconds.

Model	Total Average-mAP	Average-mAP *visible*	Average-mAP *off-screen*
Overlap, t=1s	50.4	56.1	33.1
Overlap, t=5s	50.3	55.8	33.1
Overlap, t=15s	49.9	55.2	32.5
No overlap, t=15s	51.2	56.8	34.3

Table 4.10: Overall performance of Model 5.2 using the Average-mAP score. The rows reflects using our overlapping or non-overlapping fusion algorithm, as well as different values of t. The performance is calculated for all actions, as well as visible and off-screen actions separately.

Figure 4.13 shows a classwise comparison to the baseline. Our model performs worse than the baseline overall, and especially when spotting visible events. For the off-screen events, there are improvements to some classes. In Figure 4.14, we can see that the *Future* model from Model 5.2 (teal dotted line, top subplot) is able to more precisely spot an off-screen *Clearance* between the 41:00 and 42:00 minute marks. In this case, the prediction from our *Future* model is more accurate, and has a higher confidence score than the baseline model, which is the red dotted line, in the lower subplot.

Our belief is that the drop in performance for the visible actions comes as a result of our *Past* and *Future* models making predictions on visible actions, where our *Present* model is more accurate. Since we also filter our predictions on confidence score, it is likely that our *Past* and *Future* models *wins out* over the *Present* model when spotting some events that are visible. Ideally, our *Past* and *Future* models should supplement our *Present* model when spotting off-screen events only. A challenge with this is that our models does not distinguish between visibility when predicting an event, making it difficult for us to programmatically enforce this. Table 4.11 shows the contributions of the different models after the fusion layer. We can see that the *Present* and *Future* models has a higher relative contribution than the *Past* model, compared to Model 4.3 (Table 4.8). We theorised that this is an effect of how our fusion algorithms are designed. It seems reasonable that the *Present* model has a higher number of predictions with a high confidence score, since it is trained on more data and more classes than our *Past* and *Future* models. It also makes sense that the *Past* models contribution is diminished, given that we filtered out ~1300 predictions made by this model when we removed predictions with a negative *position* value.

While conducting the experiments in this section, we discovered

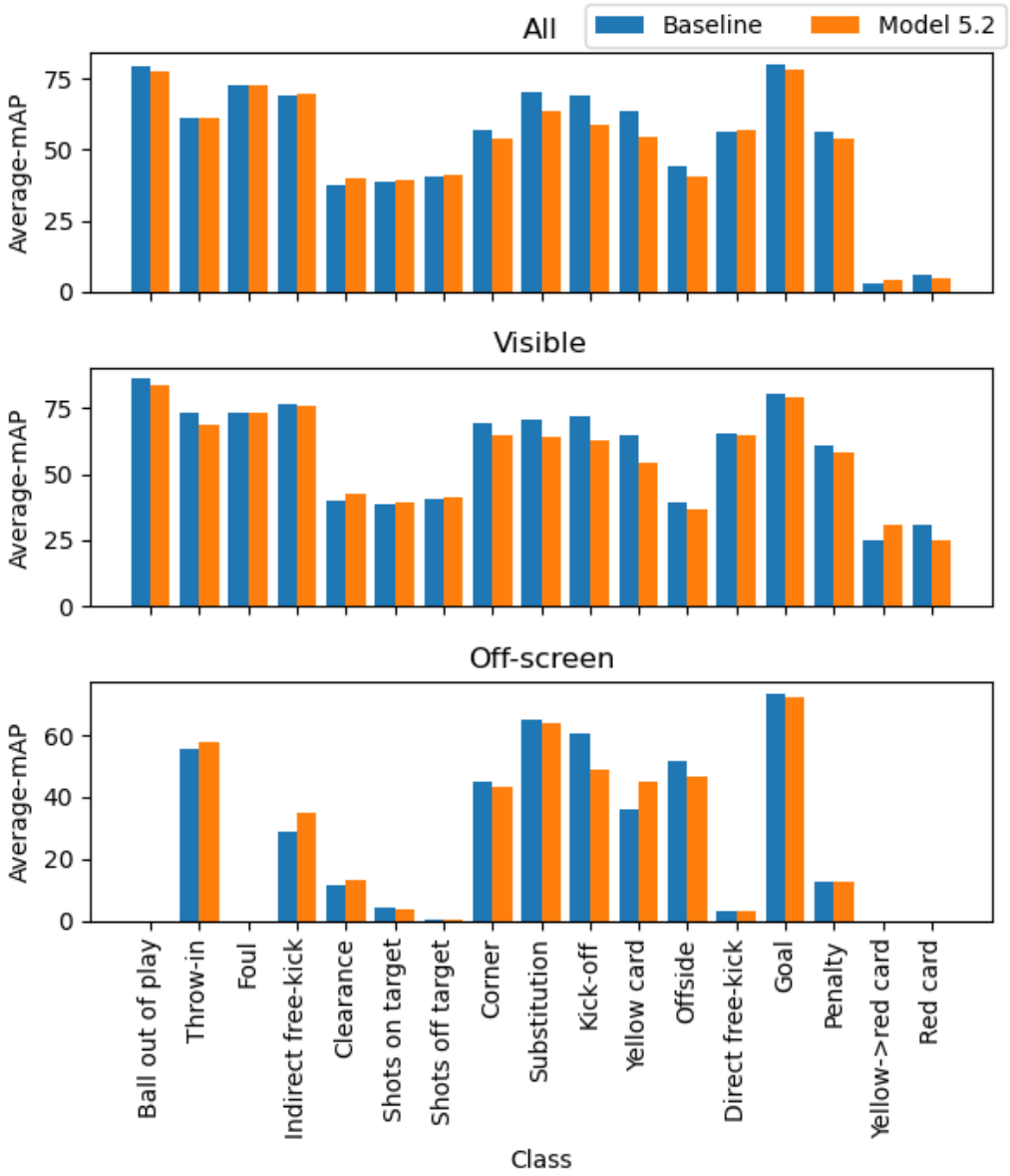


Figure 4.13: Classwise Average-mAP scores for Model 5.2, compared to the baseline. The figure is separated in three subplots for different visibility of the events. The classes on the x-axis are the same for all three subplots. The raw numbers can be found in Table G.1.

an anomaly in our *Past* model, where it had a series of predictions with a negative timestamp. Removing these predictions improved the performance of our model. We were also able to create two versions of a sliding window algorithm for our fusion layer. The best results were obtained when using the non-overlapping algorithm, and a window-size of 15 seconds. These two improvements gave us our best performing model, Model 5.2. This model achieved a higher Average-

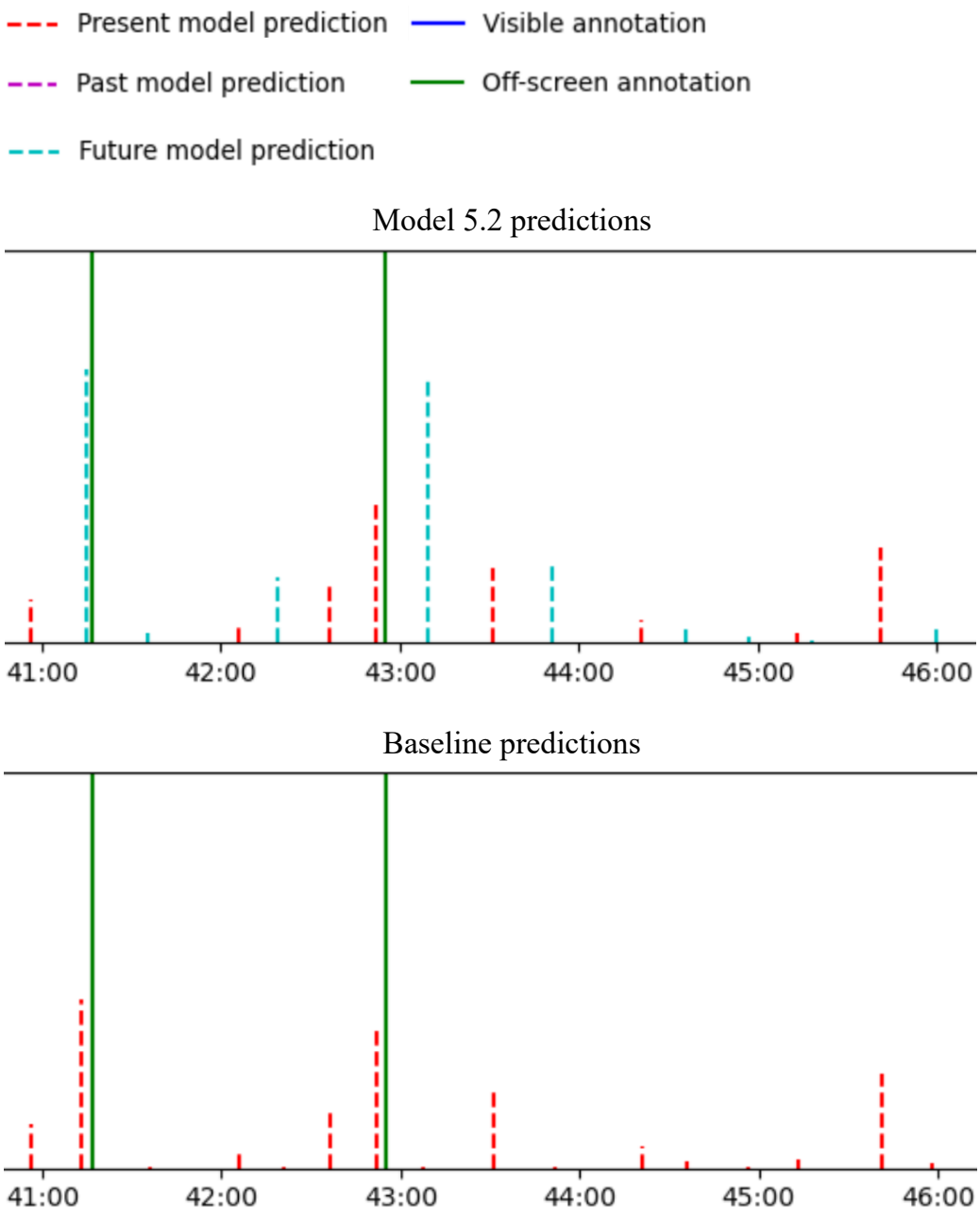


Figure 4.14: Visualising *Clearance* predictions from Model 5.2 and the baseline for a game in the SoccerNet-v2 dataset, during a 5-minute period. The labels on the x-axis are timestamps from the game (*mm:ss*). The height of the dotted bars are determined by the confidence score of the prediction. The top subplot shows the predictions from Model 5.2, with predictions from the *Present* and *Future* models, while the bottom subplot shows predictions from the baseline.

mAP than the baseline when spotting off-screen *Throw-in*, *Indirect free-kick*, and *Yellow card* events, and when spotting visible *Clearance*

Model	*n* predictions	%
Past	62,081	10.8
Present	321,523	56.1
Future	189,867	33.1
Total	*573,471*	*100.0*

Table 4.11: Contributions from the *Past*, *Present* and *Future* models to the final output of Model 5.2, where we used the non-overlapping sliding window algorithm in the fusion layer with $t = 15s$. 56.1 % of the total number of output predictions in Model 5.2 comes from the *Present* model

and *Yellow then red card* events.

4.7 Alternative evaluation for real-life applications

As mentioned in Section 3.10.2, we will evaluate our best performing model, Model 5.2, using precision, recall and F1 score, and reflect on its suitability to real-life scenarios.

4.7.1 Use case: Generating statistics

In soccer, there is a long-standing tradition of recording statistics from games. These statistics have value for bookmakers, athletes, managers and fans alike. In this scenario, we defined statistics as a record of all events happening throughout a game, limited to the 17 classes of SoccerNet-v2, and disregarding the timestamp of the events. In other words, a record of the events in a game without any information on when the events happened. Throughout all our experiments, we have found it challenging to correctly change the timestamps of the predictions made by the *Past* and *Future* models. This evaluation is therefore a way to see if our model is competitive in a context where we eliminate this challenge. For each game, and for each class, with a number of annotated events, X, and a number of predictions made by our model, Y, we defined the true positive, false positive, and false negative values of the models output as such:

$$TP = \begin{cases} Y, & \text{if } X \geq Y \\ X, & \text{otherwise} \end{cases}$$

$$FP = Y - TP$$

$$FN = X - TP$$

In the context of this use case, calculating TP, FP and FN is trivial. Since we disregarded the timestamps of the events, every prediction of a class is a TP, as long as the number of predictions does not exceed the number of annotations for that class. Every prediction beyond that, is a false positive. Lets consider an example, where we have a game in the dataset with two *Goal* annotations, and our model outputs three *Goal* predictions. Since we did not consider the temporal tolerance δ (described in Section 3.10.1), and the timestamps of the events to calculate true positives, we only considered the number of predictions and the number of annotations for each class. In this example, we have two TPs, and one FP. Shortcomings of this calculation is further discussed in Section 4.8.4. We did not define true negative, as this is every frame in a game where our model correctly does not spot an action.

If we recall Section 3.10, we defined precision as answering the question *How confident can we be that a positive prediction is correct?* and recall as *How many of the positive observations is our model able to capture?* In the context of this use case, one could argue that both metrics are of equal importance. We need to capture all events of all classes, and at the same time not capture too many. This seems like a challenging proposition, given the precision-recall trade-off that states that as one increases, the other decreases [6]. For this reason, we think that the F1 score is the most suitable metric to measure which models performs best for the given use case, given that it is the harmonic mean between precision and recall. We will therefore discuss all three metrics mentioned above, but use the F1 score to determine which model is best suited for the use case.

4.7.2 Results

As we saw in Table 4.11, Model 5.2 makes a total of 573,471 predictions for the 100 games in the test set, which is higher than the 22,551 annotations for the same games. Our model includes all predictions made, regardless of the confidence score of the predictions. When only including predictions at different confidence threshold, the number of relevant predictions drops by a large margin from confidence threshold 0 to 0.1, as seen in Figure 4.15. Between threshold 0.1 to 0.9, the decline is more gentle. From Table 4.12, we can see that the number of predictions made by our model at a confidence threshold between 0.6 and 0.7 is close to 22,551, the actual number of events in the test set.

When choosing a confidence threshold, we are making a trade-off between precision and recall, and therefore we start by looking at the

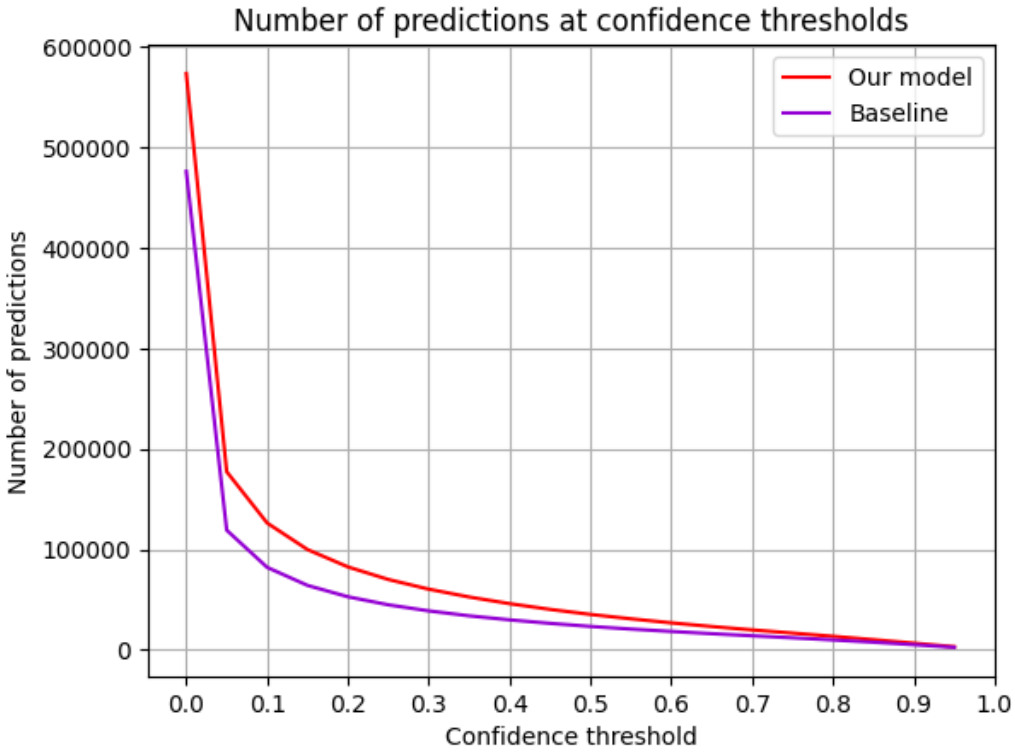


Figure 4.15: The figure shows how the number of predictions are reduced as the confidence threshold increases. The rapid decrease in number of predictions from confidence threshold 0.0 to 0.1, for both our model and the baseline, shows that there are a lot of predictions with low confidence score below 0.1.

scores at different thresholds.

In Figure 4.16, we have plotted the precision score for our model, against confidence thresholds from 0 to 0.95. We included the baseline model as well. The baseline has a higher precision score at all confidence thresholds, though the gap is closing at a confidence threshold of 0.9. At this threshold, both models makes few predictions and the predictions made are likely to be true positives. If we have fewer predictions than the number of annotations for a class in a given game, there will be zero false positive predictions for that class and the precision score will be 1. From Table 4.12, we see that the total number of predictions at confidence threshold 0.9 is 6,799, lower than the number of annotations.

The gap between our precision scores might be explained by the fact that our model makes more predictions than the baseline, as seen in Figure 4.15 and have more false positive predictions. In our fusion layer, where we merge the *Past*, *Present* and *Future* model, we tried to remove duplicate predictions, but the algorithm is not perfect.

Confidence threshold	Number of predictions	
	Model 5.2	Baseline
0.0	573,471	476,161
0.1	126,360	82,225
0.2	82,651	52,944
0.3	60,503	38,902
0.4	46,107	29,987
0.5	35,399	23,460
0.6	27,006	18,454
0.7	20,003	14,166
0.8	13,590	10,002
0.9	6,799	5,309

Table 4.12: Comparing the total number of predictions made by Model 5.2 and the baseline model, when filtering the predictions on different confidence thresholds.

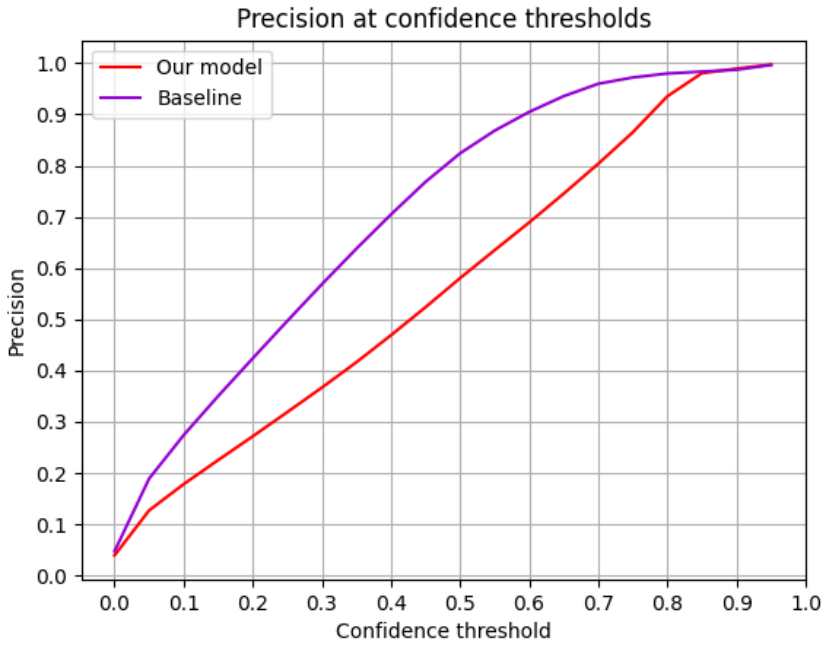


Figure 4.16: Comparing the precision scores for our model and the baseline, when filtering the predictions on different confidence thresholds.

In Figure 4.17, we can see that all classes except *Yellow then red card* and *Red card* have the highest precision at confidence threshold 0.9 as well. For *Yellow then red* and *Red card*, the precision scores

are not applicable beyond a confidence threshold of 0.2 and 0.1 respectively. This is because there are no predictions for these classes with a confidence score above 0.2, and the precision score cannot be calculated. We feel confident that this is because there are few training instances of these classes, which leads to low-confident predictions made by our model.

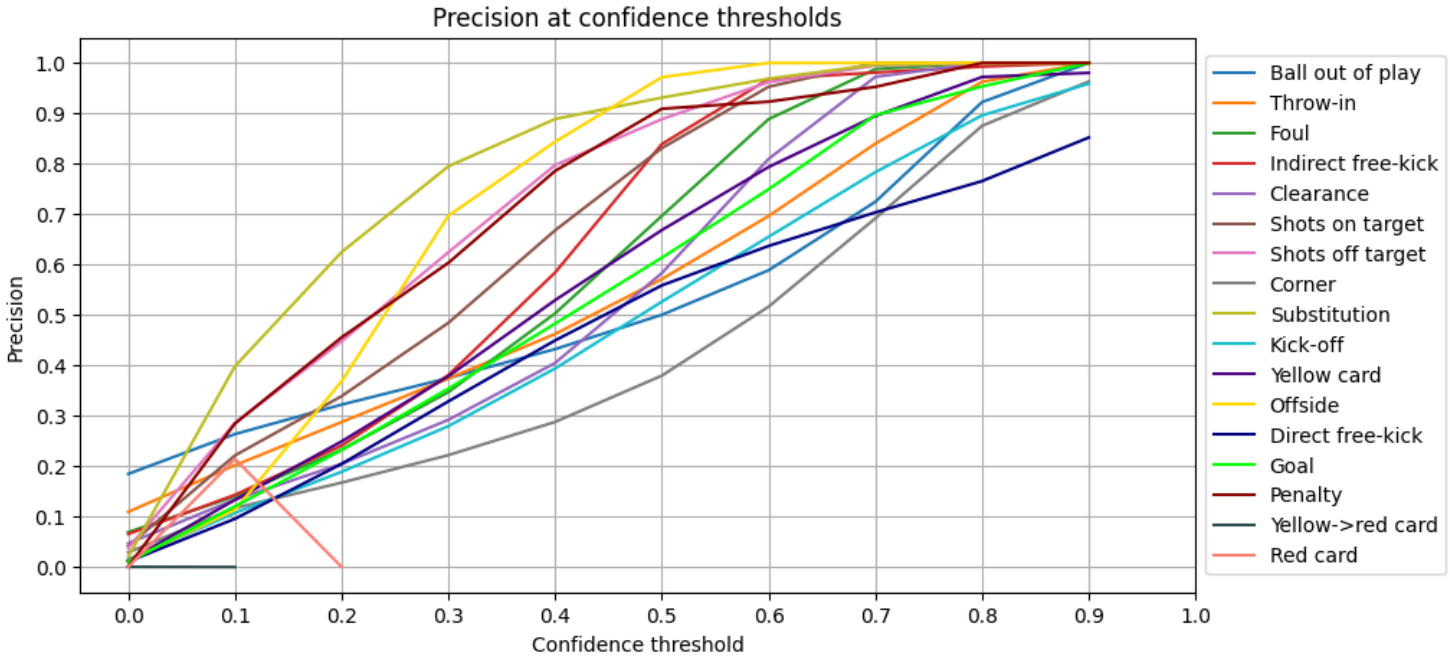


Figure 4.17: Classwise precision scores for our model, when filtering our predictions on different confidence thresholds. The raw numbers can be found in Table H.1.

Figure 4.18 shows the recall score at the same thresholds as Figure 4.16. Our model has the same recall score as the baseline up to a confidence threshold of about 0.25, but at higher thresholds our model has higher recall-values than the baseline. This is expected, as our model is designed to spot more events by including the off-screen predictions from our *Past* and *Future* models. This means that we will have less false negatives, or missed predictions, and this increases the recall score.

On a classwise basis, we can see from Figure 4.19 that the optimal confidence threshold is at 0.0 for all classes. At 0.0, our model is able to capture all events from all classes. Because of the precision-recall trade-off [6], a confidence threshold of 0.0 would greatly decrease the precision of our model. We can also see that the performance of *Yellow then red card* and *Red card* is not applicable with a confidence threshold of above 0.1 and 0.2 respectively. *Ball out of play* is one of the classes with the highest recall across confidence thresholds. *Ball*

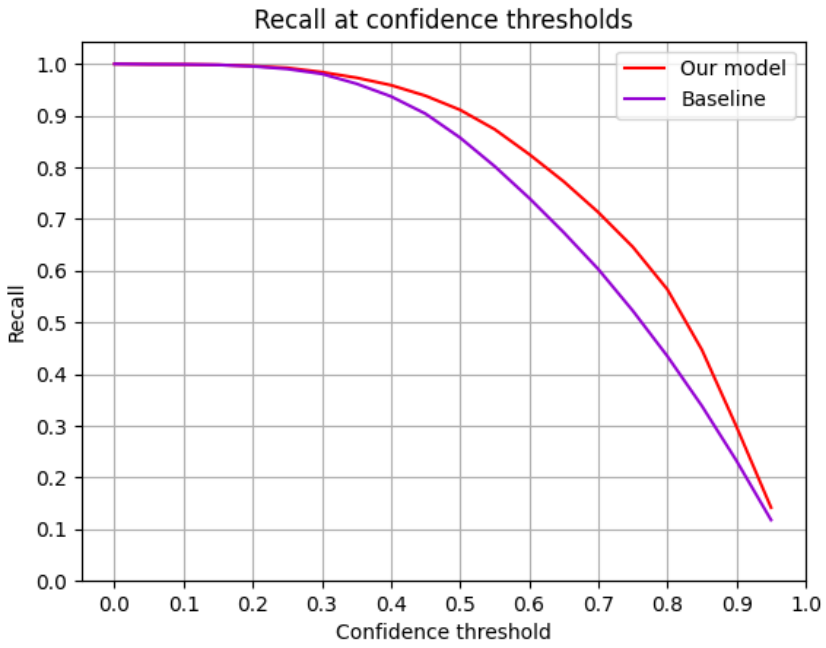


Figure 4.18: Comparing the recall scores for our model and the baseline, when filtering the predictions on different confidence thresholds.

out of play is the class with the most annotations by far, and it seems reasonable that this contributes to the ability of our model to be able to learn to spot more events from that class than most others. For confidence thresholds between 0.8 and 0.9, we can see that most classes has a significant drop in recall. We observe that *Ball out of play* has a large drop in recall score from 0.8 to 0.9. From Figure 4.15 we can see that the number of predictions made by our model is less than the number of annotations in the test set (22,551) with confidence thresholds above 0.6. This will increase the amount of false negatives. We believe that since *Ball out of play* is the most frequent class in the test set, the recall for this class will be penalised more than the other classes, as the confidence thresholds increase beyond 0.6. When the number of true positive predictions start to diminish, the number of false negative predictions will increase proportionate to the relative distribution of the class in the test set. This could also explain why the *Throw-in* class has an almost identical drop in recall from confidence threshold 0.8 to 0.9, considering that it is the class with the second largest amount of annotations in the test set.

Figure 4.20 shows the F1 score at different thresholds. Our model outperforms the baseline when the confidence threshold is higher than about 0.67, but the highest F1 score is achieved by the baseline model.

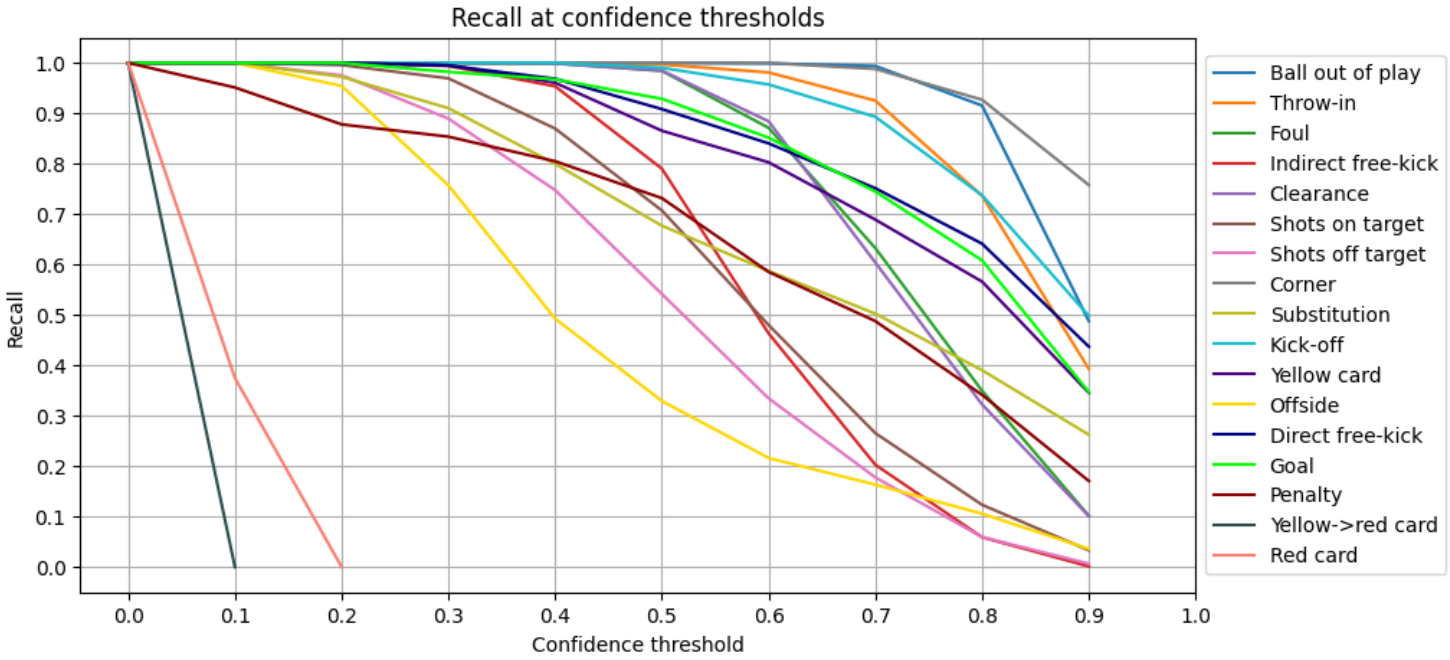


Figure 4.19: Classwise recall score for our model, when filtering the predictions on different confidence thresholds. The raw numbers can be found in Table H.2.

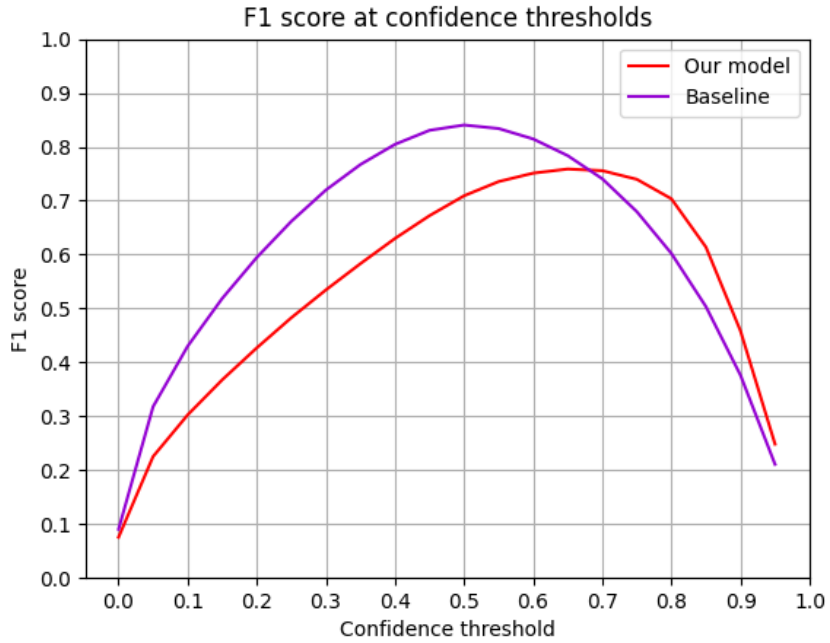


Figure 4.20: Comparing F1 score between our model and the baseline, when filtering predictions on different confidence thresholds.

The gap in score between our model and the baseline is larger when we are looking at the precision scores in Figure 4.16 compared to the recall scores in Figure 4.18. Because of this, the baseline has a higher F1 score than our model at the maximum on the F1-curve. Our model and the baseline also reaches their maximum F1 score at different confidence thresholds since the intersection of the precision and recall curves differ for our model and the baseline. When looking at the

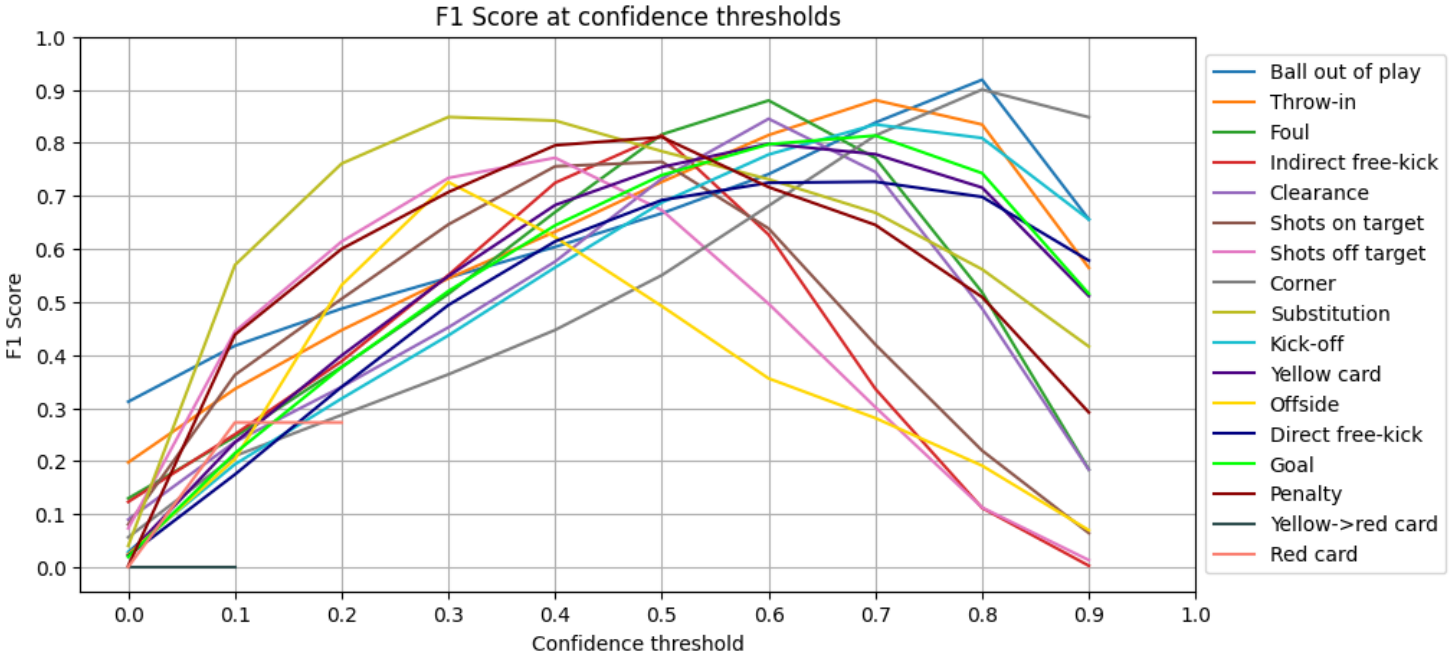


Figure 4.21: Classwise F1 score for our model, when filtering the predictions at different confidence thresholds. The raw numbers can be found in Table H.3.

classwise F1 scores for our model in Figure 4.21 and the baseline in Figure 4.22, we can see that there is a disparity between the optimal confidence threshold for most classes. Since our objective for this use case is to maximise the F1 score, we filtered the predictions of each class based on different confidence thresholds. The optimal confidence threshold for each class was chosen by finding the confidence threshold with the highest F1 score (for visible and off-screen events combined) from 10 intervals between 0 and 1. This was done independently for both our model and the baseline model. The results of these adjustments can be seen in Table 4.13. The baseline model achieves an F1 Score of 87.2%, which is slightly higher than 86.2%, the F1 score achieved by our model. The baseline also achieves a 2.4 p.p higher precision at 88%, compared to 85.6% for our model. In terms of recall, our model scores 0.4 p.p higher at 86.7%, while the baseline achieved 87.2%. The improvement in recall comes as a result of our model

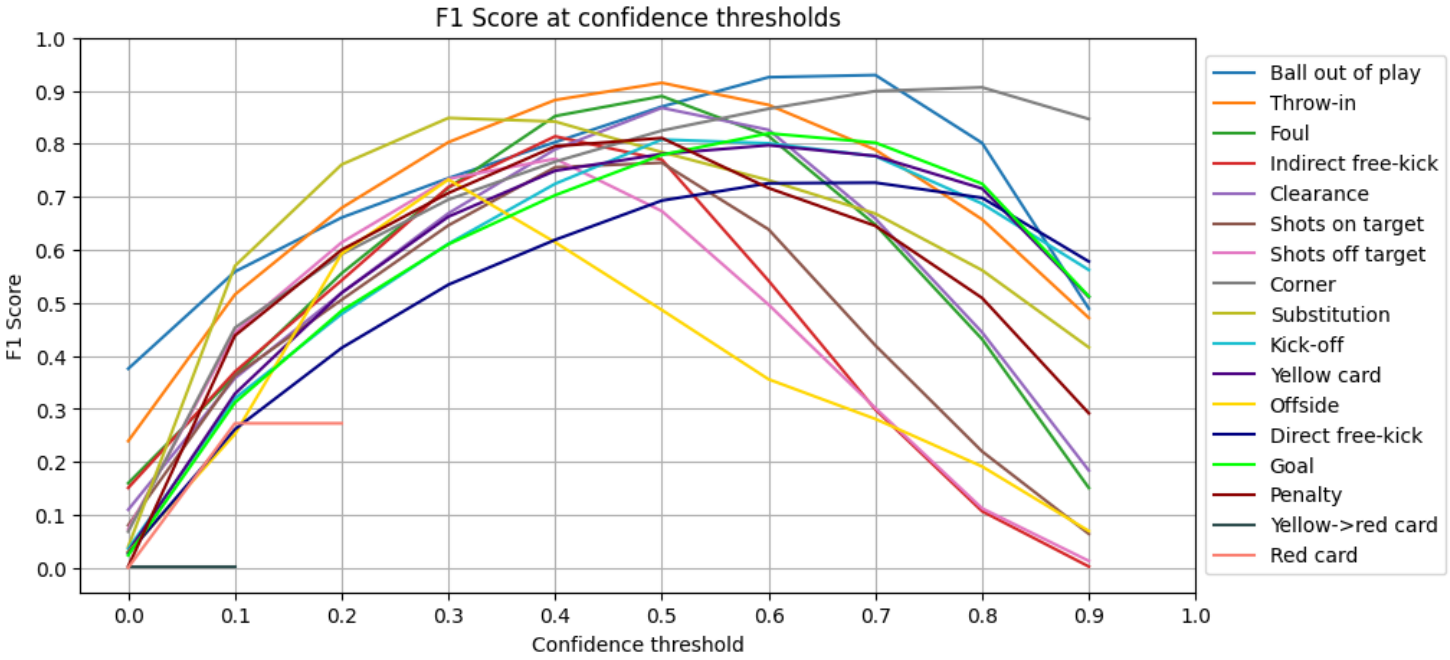


Figure 4.22: Classwise F1 score for the baseline, when filtering the predictions at different confidence thresholds. The raw numbers can be found in Table H.4.

having a higher number of true positives than the baseline. In this use case, this is to be expected, since our model outputs more predictions than the baseline. The difference in precision can be explained by the fact that the baseline model has 635 fewer false positives than our model.

In this evaluation we have evaluated our model against the baseline using precision, recall, and F1 score. We have also filtered the models (our model and the baseline) predictions on the confidence thresholds that give the highest F1 score for each individual class. This reduces the amount of false positives in both models, and gives a number of predictions closer to the number of annotations in the test dataset. After filtering the predictions on the optimal confidence thresholds, we saw that the baseline achieved the highest precision and F1 score, while our model had the highest recall. We have stated that the preferred model for this use case is the one with the highest F1 score, and must therefore conclude that the baseline model is the preferred model in this use case.

4.7.3 Use case: Game commentary

We want to formulate a use case for an action spotting model where we have to consider the timestamp of the annotations. For many real-

	Model 5.2	Baseline
True positives	19,552	19,459
False positives	3,278	2,643
False negatives	2,999	3,092
Precision	0.856	**0.880**
Recall	**0.867**	0.863
F1 Score	0.862	**0.872**

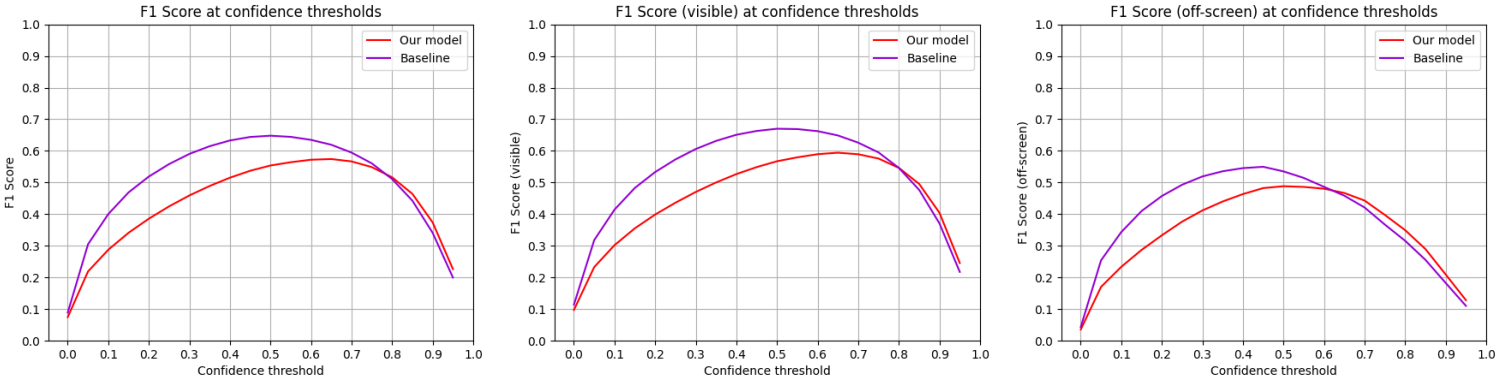
Table 4.13: Comparing outcomes and evaluation metrics between Model 5.2 and the baseline model. The number of true positive, false positive and false negative predictions are compared, as well as the precision, recall and F1 score evaluation metrics.

life applications, this is important, and it lets us also evaluate the capabilities our model has to correctly spot off-screen events.

Soccer broadcasters like ESPN often offer information about events in a soccer match in the form of textual match commentaries [15]. These commentaries have information about match events, often with a one minute granularity. The event information contains what event has happened, which player(s) are involved in the event, which teams are involved, and at what time the event occurred. We therefore want to evaluate Model 5.2s performance on a similar, albeit simplified use case. In our simplified use case, we do not consider which team, or which player performs an action, and constrain ourselves to the 17 classes in SoccerNet-v2. In short, the models output a list of predicted events with a timestamp. We define true positives, false positives, and false negatives as we did for SoccerNet-v2 evaluation in Section 3.10, and set a fixed temporal tolerance of δ = 60 seconds. From these numbers, we can calculate precision, recall and F1 score overall, and for each class. Since each ground truth observation is matched with the closest prediction, as explained in Section 3.10.1, we can see how our model performs on off-screen events only. We will focus on off-screen events and compare our results with the baseline model, as this is the primary focus of this thesis. As in our previous use case evaluation, we will filter the predictions of each class on the confidence threshold that provide the best F1 score (all visibilities) for that particular class.

4.7.4 Results

Figure 4.23 shows the F1 scores over different confidence thresholds and grouped by visibility. The baseline has the overall highest F1 score, but our model does outperform the baseline at higher confidence thresholds. Our model performs best when spotting off-screen events



(a) F1 score for all events (b) F1 score for visible events (c) F1 score for off-screen events

Figure 4.23: Comparing F1 score between our model and the baseline, when filtering predictions on different confidence thresholds. Each figure shows the F1 score for different visibility metrics (all, visible, and off-screen from left to right).

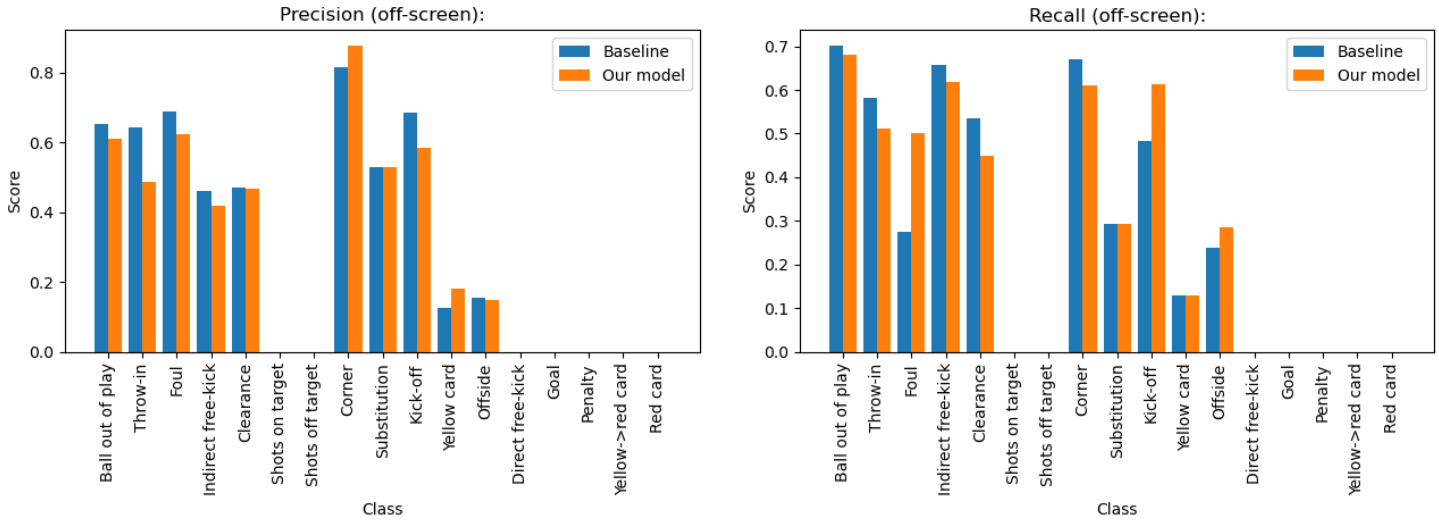
only. We see in Figure 4.23c that our model has a higher F1 score than the baseline for more confidence thresholds than in Figure 4.23a and 4.23b. In Figure 4.23c, our model starts performing better than the baseline when the confidence threshold is above ~0.6, while for figures 4.23a and 4.23b those confidence thresholds are ~0.76 and ~0.8 respectively. As we did in Section 4.7.1, we find the confidence threshold which gives the highest F1 score (all visibilities) per class. For this use case, we calculated the F1 score for 200 confidence thresholds for each class. The confidence threshold which resulted in the highest F1 score per class is used from this point onward. The confidence thresholds are independent for each model and can be seen in Table 4.14.

Threshold	Ball out of play	Throw-in	Foul	Ind. free-kick	Clearance	Shots on tar.	Shots off tar.	Corner	Substitution	Kick-off	Yellow card	Offside	Dir. free-kick	Goal	Penalty	Yel. to Red	Red card
Baseline	0.60	0.48	0.51	0.36	0.44	0.35	0.32	0.78	0.46	0.70	0.55	0.38	0.70	0.67	0.46	0.03	0.08
Model 5.2	0.76	0.68	0.54	0.42	0.60	0.35	0.32	0.84	0.46	0.72	0.66	0.36	0.70	0.68	0.46	0.03	0.08

Table 4.14: Comparing the confidence thresholds that gives the highest F1 score on a classwise basis, for Model 5.2 and the baseline.

Looking at precision and recall, which contributes to the F1 score, we see that there are differences between the classes. In Figure 4.24, the precision and recall for predictions for off-screen events is plotted

per class for both the baseline and our model. Seven of the classes, namely *Shots on target*, *Shots off target*, *Direct free-kick*, *Goal*, *Penalty*, *Yellow then red card* and *Red card* has a score of 0 for both precision and recall. For *Goal*, *Penalty*, *Yellow then red card* and *Red card* this is because there are no off-screen events for these classes in the test set. Table 4.15 show that the sum of true positive and false negative predictions for these classes are zero. For *Shots on target*, *Shots off target* and *Direct free-kick*, neither our model, nor the baseline manages to correctly produce any true positive predictions.



(a) Precision for off-screen events

(b) Recall for off-screen events

Figure 4.24: Comparing precision and recall between our model and the baseline, when spotting off-screen events. For this figure, the predictions of each class are filtered on the confidence thresholds that yielded the highest F1 score for that class.

Our model has a higher precision score than the baseline for *Corner* and *Yellow card*, as can be seen in Figure 4.24a. The baseline has a higher score for multiple classes, but the difference is greatest for *Throw-in*. From Figure 4.25, we can see that for the *Corner* class, our model has about 80% less false positive predictions than the baseline. Our model has less true positive and more false negative predictions as well, which does have a negative impact on the score, but the difference for TP and FN are much lower than the false positives.

For *Yellow card*, there is no difference between the number of true positive and false negative predictions, but our model has about 50% less false positives. In Table 4.14, we see that the confidence threshold used by the baseline model for the *Yellow card* class is 0.11 lower than our model. More predictions with lower confidence score may be included from the baseline model and this can explain the additional

Metric	Ball out of play	Throw-in	Foul	Ind. free-kick	Clearance	Shots on tar.	Shots off tar.	Corner	Substitution	Kick-off	Yellow card	Offside	Dir. free-kick	Goal	Penalty	Yel. to Red	Red card
True Positive																	
Model 5.2	192	547	20	628	383	0	0	72	27	219	4	6	0	0	0	0	0
Baseline	198	621	11	669	456	0	0	79	27	172	4	5	0	0	0	0	0
False Positive																	
Model 5.2	122	574	12	867	435	14	10	10	24	155	18	34	56	8	3	44	18
Baseline	105	347	5	783	512	14	10	18	24	79	28	27	56	8	3	44	18
False Negative																	
Model 5.2	90	521	20	389	470	1	2	46	65	138	27	15	17	0	0	0	0
Baseline	84	447	29	348	397	1	2	39	65	185	27	16	17	0	0	0	0
n predictions																	
Model 5.2	404	1642	52	1884	1288	15	12	128	116	512	49	55	73	8	3	44	18
Baseline	387	1415	45	1800	1365	15	12	136	116	436	59	48	73	8	3	44	18

Table 4.15: Comparing outcomes and predictions between Model 5.2 and the baseline. The numbers are split between classes and grouped by true positives, false positives, false negatives outcomes, as well as the number of predictions made.

false positive predictions.

From Figure 4.25, we can see that our model performs worse for all the outcomes when predicting *Throw-in*. Our model has less true positive predictions, more false positive, and false negative predictions. The largest difference is for false positive predictions, where our model has ~40% more than the baseline, and this number does contribute to the precision score. In Section 3.9.2, we discussed how our *Future* and *Past* model shift the timestamp of the actions that are predicted in the past/future. We suspect that this timestamp-shifting scheme is imprecise, leading to false positive predictions. In Figure 4.26, we see one example of where the *Future* model (teal dotted line, top subplot) predicts a false positive *Throw-in* event, right after the 37:15 mark. Our model uses a sliding window approach to filter out all but the most confident prediction for each class in 15 second windows. From Figure 4.26, we see that the aforementioned FP prediction from our *Future* model right after the 37:15 mark, and the TP prediction from our *Present* model (red dotted line, top subplot) right before the 37:00 mark, are too far apart to be filtered against each other. It seems reasonable that a larger window-size than 15 seconds could reduce the amount of FP predictions for this class. We tuned this window-size to achieve the best possible Average-mAP score - given more time, we

would have liked to experiment more with our data fusion algorithm to optimise for these new evaluation metrics.

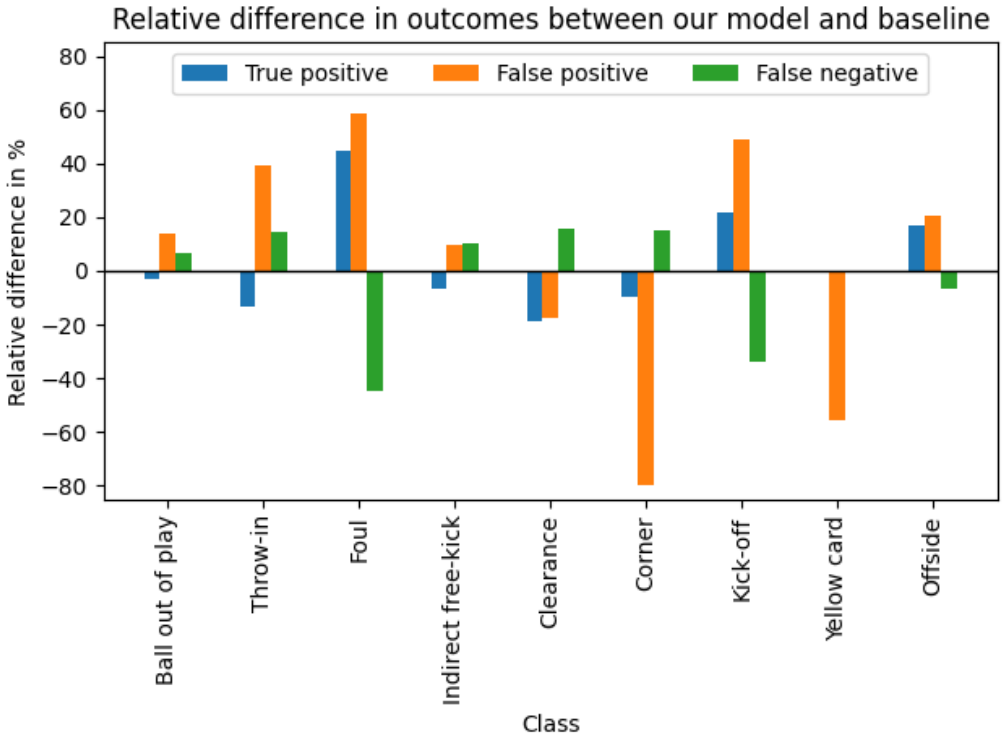


Figure 4.25: Relative difference for TP, FP and FN between the baseline and our model, with respect to the baseline. Only classes where there is a difference in at least one of the possible prediction outcomes are included. A positive percentage indicates that our model has a higher number of instances of that metric than the baseline.

For the recall score in Figure 4.24b, our model achieves a higher score for the *Foul*, *Kick-off* and *Offside* classes. The baseline outperforms our Model 5.2 of the classes *Ball out of play*, *Throw-in*, *Indirect free-kick*, *Clearance* and *Corner*. The recall score for *Foul* is almost twice as high for our model compared to the baseline. In Figure 4.25, we can see that our model have over 40% more true positive predictions and over 40% less false negative predictions. Both of these numbers contribute to the large difference in recall score. Our model does have about 60% more false positive predictions as well, but this does not affect the recall score. As depicted in Figure 3.4, there are 6 types of events (*Indirect free-kick*, *Direct free-kick*, *Penalty*, *Yellow card*, *Red card*, *Yellow then red card*) that are preceded by a *Foul*. This leads us to believe that our *Past* model will produce more predictions of this class (both FP and TP) than the baseline model. This is especially beneficial when spotting off-screen events, since our *Past* model can

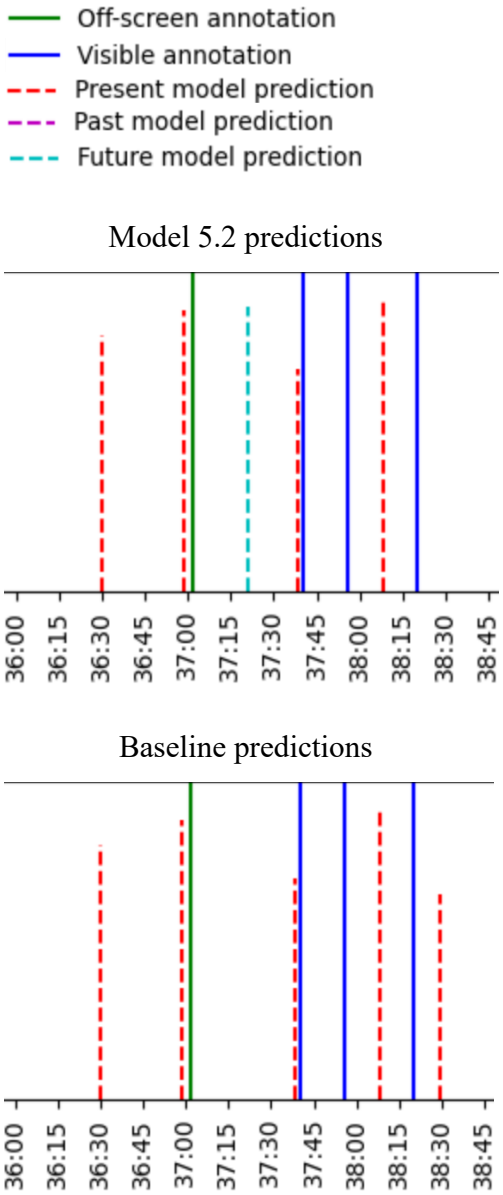


Figure 4.26: Visualising *Throw-in* predictions from Model 5.2 and the baseline for a game in the SoccerNet-v2 dataset, during a ~3-minute period. The labels on the x-axis are timestamps from the game (*mm:ss*). The height of the dotted bars are determined by the confidence score of the prediction. The top subplot shows the predictions from Model 5.2, with predictions from the *Present* and *Future* models, while the bottom subplot shows predictions from the baseline.

leverage 6 types of events that are mostly visible (Table 3.2), to predict an off-screen event in the past.

The predictions for *Kick-off* follows the same pattern as those for *Foul*. Our model has more true positives and false positives, and less

false negatives. Although the pattern is the same, the differences are relatively smaller than for the *Foul* class. The number of predictions for *Kick-off* is higher than the number of *Foul* predictions, which means that each number of *Kick-off* predictions have less impact on the relative difference. As we saw in Figure 3.1, *Kick-off* is a class with more off-screen than visible events. This is often due to the broadcast production showing replays of a goal after it is scored. On the other hand, there are almost no off-screen *Goal* events. Our *Future* model should predict a *Kick-off* after every predicted *Goal* event. Our conclusion from this is that our *Future* model utilises visible events (*Goal*) to predict off-screen *Kick-off* events. Intuitively, this would make our model predict more *Kick-off* events than the baseline, which only uses the visual features of the off-screen *Kick-off*. In Section 4.2, we theorised that the disparity between what is shown in the frames and the event that is happening off-screen is quite large, which would make it more challenging to correctly classify an off-screen event based on the visual features alone. Figure 4.27 shows one instance where the *Future* model in Model 5.2 (teal dotted line, top subplot) is able to provide a true positive for an off-screen *Kick-off* event, which the baseline is not able to spot.

For the *Throw-in* class, the recall score is lower for our model, as were the case for the precision score. We have about 10% less true positives and 10% more false negative predictions. Looking at total number of predictions, our model has more than the baseline, even though our model uses a 0.2 higher confidence threshold. We believe the higher confidence threshold reduces the amount of true positives compared to the baseline, since potential true positive predictions with lower confidence score gets removed. Usually, a higher confidence threshold should also reduce the amount of false positive predictions, but this is not the case for our model. We believe there are two reasons for this: the changing of the timestamp of the predictions made by our *Future* model, and the size of the sliding window used in the fusion layer. In Section 3.8.3, we described how we need to change the timestamp of the predictions made by our *Past* and *Future* models. We added the median time between all *Ball out of play* and *Throw-in* events to the timestamp of our *Throw-in* predictions made by the *Future* model. *Throw-in* is an event that may have multiple occurrences in a short span of time. The amount of time passing from the *Ball out of play* until the *Throw-in* is taken may differ greatly each time as we can see in Table 3.7. Because of this difference, the use of median time to change the timestamp may not place the event at the correct timestamp. This may lead to imprecise predictions made by our *Future* model with regards to the timestamp. Our fusion layer is designed to remove these imprecise predictions, but may fail at this due to the size of the sliding window. The sliding window for removing

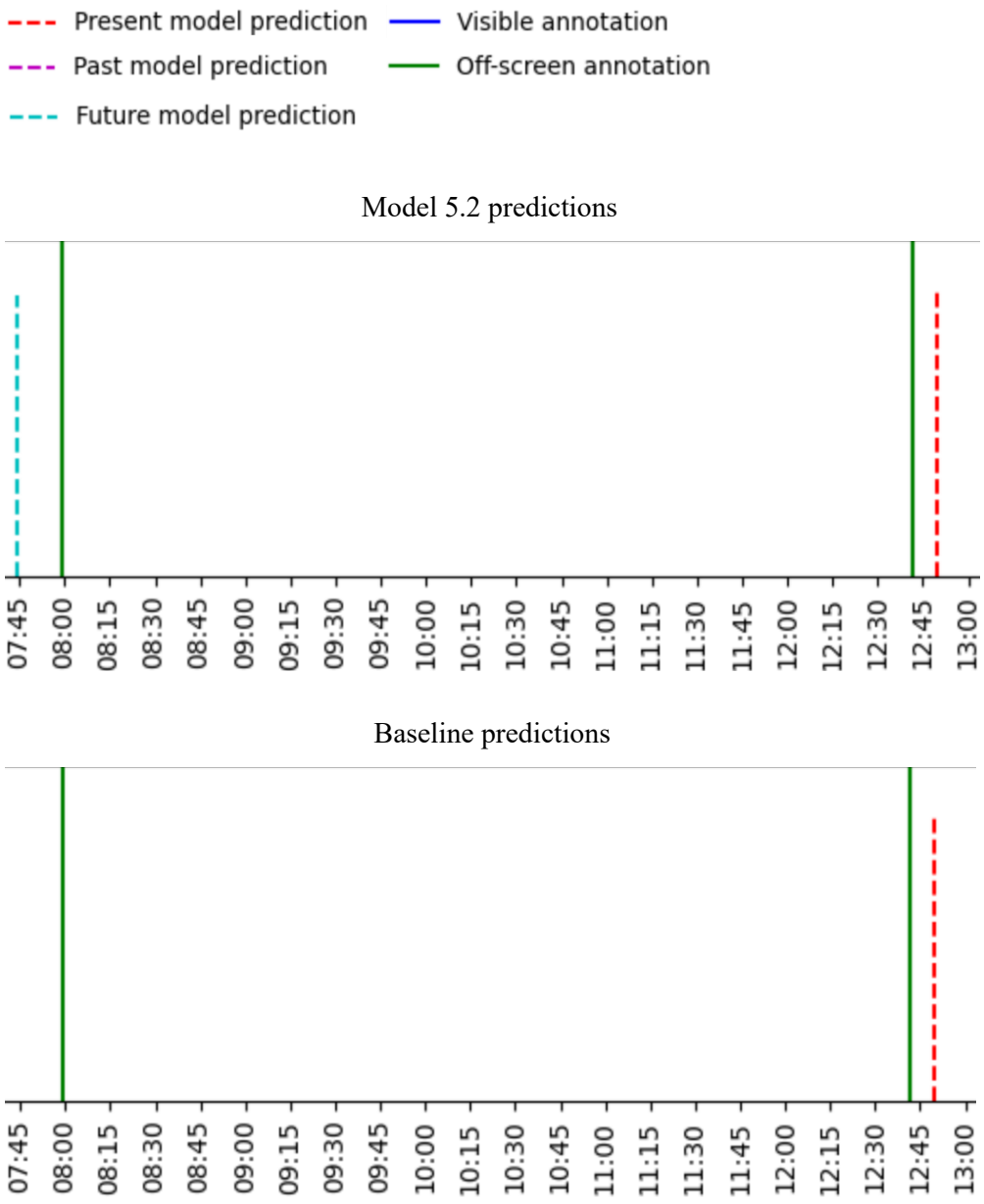


Figure 4.27: Visualising *Kick-off* predictions from Model 5.2 and the baseline for a game in the SoccerNet-v2 dataset, during a ~5-minute period. The labels on the x-axis are timestamps from the game (*mm:ss*). The height of the dotted bars are determined by the confidence score of the prediction. The top subplot shows the predictions from Model 5.2, with predictions from the *Present* and *Future* models, while the bottom subplot shows predictions from the baseline.

duplicate events is only 15 seconds, which may lead to the fusion layer failing to filter out duplicate events from our *Present* and *Future* model, if the timestamp is off. This can explain the high number of false positive predictions.

We can see from Figure 4.25 that our model predicts ~20% more false negatives, and ~20% less false positives and true positives than the baseline for the *Clearance* class. The lower recall score for our model can be explained by the distribution of false negatives and true positives. We can see from Figure 4.28 that when we filter the predictions for Model 5.2 on the confidence threshold (top right subplot), our model fails to predict the off-screen *Clearance* around the 09:45 mark. If we do not filter the predictions from Model 5.2 on the confidence threshold (bottom right subplot), we see that our model is able to predict the same *Clearance*. When looking at the predictions between the 09:45 mark and the 13:00 mark in the bottom right subplot, and the lack of predictions in the same timespan in the top right subplot, we speculate that filtering the predictions on confidence thresholds reduces the amount of false positives in our model. When measuring recall, this is irrelevant, as false positives does not impact the score. For *Clearance*, the confidence threshold for our model is 0.60, and for the baseline it is 0.44. This might explain why the baseline (top left subplot) does not filter out its true positive prediction for the *Clearance* at around the 09:45-minute mark, while Model 5.2 (top right subplot) does. In general, we believe that filtering the *Clearance* predictions on these confidence thresholds leads to our model having less true positives and more false negatives than the baseline, thus reducing the *Clearance* recall score for Model 5.2.

We can see from Figure 4.25 that our model predicts less false negatives and more true positives than the baseline, leading to our model achieving a higher recall for *Offside*. For the *Corner* class, the opposite is true - the baseline predicts more true positives and false positives and less false negatives compared to our model. This also leads to a better recall for the baseline model. In this use case evaluation, we have re-introduced the timestamp dimension for our predictions. This lets us pair each annotation to its closest prediction, and look at the performance achieved by our model on all, visible and off-screen events. We saw that in terms of F1 score, our model performs best when spotting off-screen events. We focused our analysis on spotting off-screen events only. Since we have trained our *Past* and *Future* models on labels according to Figure 3.3 and Figure 3.4, we speculate that Model 5.2 is better able to spot off-screen events for classes like *Foul* and *Kick-off*, since the *Past* and *Future* models in Model 5.2 leverages other, mostly visible events, when spotting these classes.

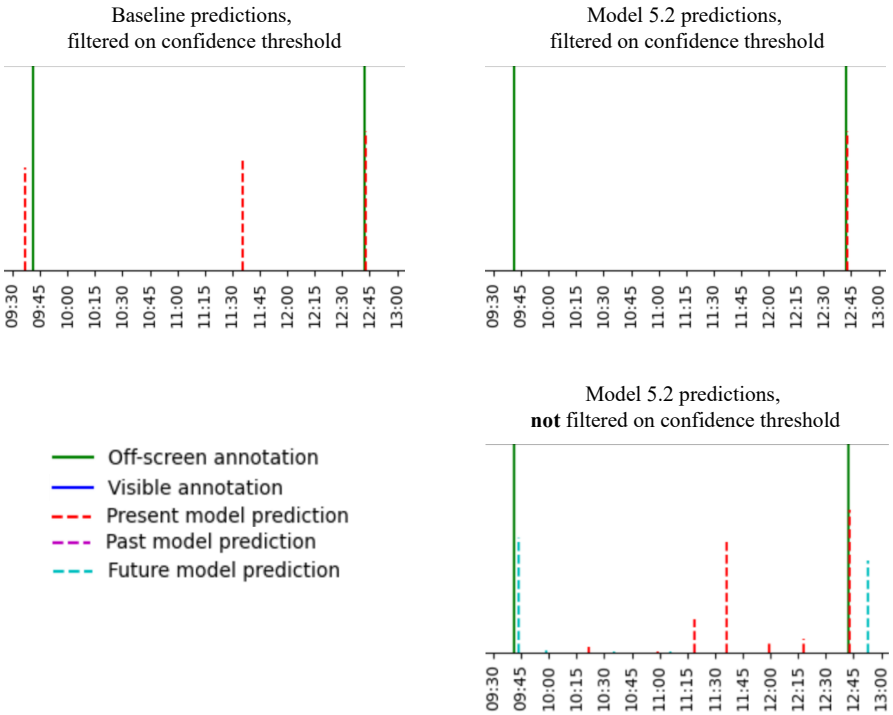


Figure 4.28: Visualising *Clearance* predictions from Model 5.2 and the baseline for a game in the SoccerNet-v2 dataset, during a 3.5-minute period. The labels on the x-axis are timestamps from the game (*mm:ss*). The height of the dotted bars are determined by the confidence score of the prediction. The top-left subplot shows the predictions of the baseline, when filtered on the confidence threshold found in Table 4.14. The top-right subplot shows the predictions of Model 5.2, with predictions from the *Present* model, after they have been filtered on the confidence threshold. The bottom-right subplot shows the predictions of Model 5.2, with predictions from the *Present* and *Future* models, without filtering the predictions on the confidence threshold.

4.8 Discussion

We have analysed our results continually throughout our experiments. In this section, we will discuss our findings, as well as some of the shortcomings and main challenges in our work.

4.8.1 Model implementation

As mentioned in Section 1.3, we did not perform any system benchmarks. For our experiments and implementations, we used the CPUs of an Nvidia DGX-2, which contains dual Intel Xeon Platinum 8168, 2.7 GHz 24-core CPUs.

4.8.2 Baseline models

We have defined our baseline as an out-of-the-box run of the NetVLAD++ model, first presented in [19]. The SoccerNet-v2 devkit provides the results of 6 runs of this model. For our initial models in Sections 4.2 - 4.6, our baseline model were the results of one of these runs (Baseline A). For Section 4.7.1 and 4.7.3 we used our *Present* model, as presented in Section 4.5, as our baseline model (Baseline B). This was an out-of-the-box NetVLAD++ model, trained with the hyper-parameters that gave Giancola & Ghanem the best performance. The reason for doing this, was that the 6 NetVLAD++ runs provided by the SoccerNet-v2 devkit only contained the results of the runs, and not the actual prediction-files. We therefore had to use a different baseline for Sections 4.7.1 and 4.7.3, since we needed these predictions in order to calculate true positives, false positives and false negatives. Table 4.16 shows a comparison of the results of these two baseline models.

Model	Total Average-mAP	Average-mAP *visible*	Average-mAP *off-screen*
Baseline A	53.23	59.24	34.37
Baseline B	53.40	59.27	35.26

Table 4.16: Comparing the performance of the two baselines we have used during our experiments, split by results when spotting all, visible, and off-screen events.

4.8.3 Model evaluation

We have evaluated our models for two purposes. First, we have measured the performance achieved by our models against the NetVLAD++ model, using the Average-mAP function to analyse how our efforts compares to existing models evaluated on the SoccerNet-v2 dataset. Secondly, we have used F1 score, precision and recall to see how our model would perform in two simplified real-life scenarios. This provides insight as to how our model would perform in a practical application. Using Average-mAP as an evaluation metric is challenging because of its generality. It is challenging to draw conclusions on what the model is specifically doing, since Average-mAP considers a range of confidence intervals and variants of the temporal tolerance, δ. Using precision, recall and F1 score for each visibility, and on a classwise basis, facilitates more detailed analysis on what the strengths and weaknesses of our models are. In Section 4.7.1, we discussed how both precision and recall are import metrics to evaluate a machine learning model. Because of the precision-recall

trade off, we argued that the F1 score is the most suitable metric to evaluate performance achieved by our models against the baseline, since it considers both precision and recall. The reason for this is that we considered precision and recall to be equally important for the use cases we defined. In both use cases, we deemed it as equally important to both capture all instances (recall), and at the same time not capture too many (precision). On reflection, we argue that a reduction in precision is acceptable for a higher recall. Until a perfect action spotting system is developed, a human operator will always be required to review and edit the results of the model when applied to a real-life scenario. We think that it is more important then, that the model is able to spot all actions that are of interest in a game, than it is to provide less false positives. If a model has a recall of ~1 and a low precision, the human task is to remove false positives from the models output. If a model has a higher precision and a lower recall, the human would have go through the whole game manually, double-checking that the model did not miss any interesting actions. In that case, a machine learning model is superfluous, and it is as efficient to have a human operator go through the game from start to finish and annotate the relevant events. It is important to note that we consider this to be true for this specific scenario. Given another domain, or even use case, another evaluation metric might be more favourable.

As can be seen in Table 4.12, one of the challenges when evaluating our model in Section 4.7, was that the number of predictions made by the model were far greater than the annotations in the dataset, especially with lower confidence scores. Our solution to this was to filter out the predictions on a classwise basis based on the confidence threshold that gave the optimal F1 score for that particular class. Looking back at Figures 4.23 and 4.20, the general observation is that our model has a higher F1 score than the baseline with high confidence thresholds. If we were to set a fixed confidence threshold of lets say 0.8, our model would perform better than the baseline. However, we wanted to see how much we could optimise F1, precision and recall. To do this, we filtered each class individually (for our model and the baseline) on the confidence threshold that would give the optimal F1 score for that particular class, as per Table 4.14. This increases the overall F1 score of both our model and the baseline, and did not make our model perform better than the baseline.

4.8.4 Limitations in use cases

In our use case for generating statistics, we disregarded the timestamp of the events. This may lead to unintended results, where two wrong predictions make a right one. We can illustrate this with an example. We have a game where there is only one *Goal* annotation. This goal

is scored in the second half, after 60 minutes. Our model has a *Goal* prediction 10 minutes into the first half, and no goal predictions in the second half. Usually this would count as a false positive prediction in the first half, and a false negative prediction in the second half. Instead, we will have a true positive prediction for the *Goal* class in this game and no false positives or false negatives.

In an actual real-life application of generating statistics, the team that performs the action would naturally be an important part. If we predict 4 goals in a game, we would want to know how many goals were scores by each team. This is needed in order to determine the score of a game, if the game ended 2-2, 4-0 or 3-1. For our model and the baseline to be used within this scenario, we had to simplify this approach and ignore which team performs the action.

In Section 4.7.3 we evaluated our model in terms of a serving as a game commentary service. Since we considered the timestamps of the events, we were able to measure F1 score, precision and recall on all, visible and off-screen events. This allowed us to perform a more detailed analysis than in the previous use case. Realistically, when measuring the performance of a model for this use case, we would not discern between visible and off-screen performance, since both are equally important for the system. Since our thesis is mostly focused on improving the off-screen action spotting performance, and since we had time constraints to consider, we chose to focus our efforts on analysing how our model performed on off-screen events in this use case.

4.8.5 Duplicate events

During the evaluation for our last use case in Section 4.7.3, we discovered some anomalies in the dataset. Throughout this thesis we have calculated that there are a total of 22,551 annotations in the test dataset. During our last evaluation, we found that 41 of these are duplicates, and that in reality there are 22,510 unique annotations in the test dataset. We derived our evaluation function from the Average-mAP function provided in the SoccerNet-v2 devkit. We discovered that during the processing of this function, a total of 41 annotations were filtered out from the set of labels. Upon further inspection, we found that the Average-mAP-function uses the *gameTime* attribute of the annotations instead of the *position* attribute. Since there can only be one event of any class happening in one frame, we found that the Average-mAP filtered 41 annotations that had the same *gameTime* values, but different *position* values.

Table 4.17 shows how these duplicates are distributed across the 17 classes in the dataset. For most classes, the amount of duplicates are quite small, except for *Goal*, where 11 of our original 337 annotations are duplicates. We sampled some of the games where we found a

Class	Original annotations	Without duplicates	Number of duplicates
Kick-off	514	513	1
Goal	337	326	11
Substitution	579	564	15
Shots on target	1175	1171	4
Shots off target	1058	1057	1
Clearance	1631	1630	1
Ball out of play	6460	6459	1
Throw-in	3809	3808	1
Foul	2414	2413	1
Yellow card	431	426	5

Table 4.17: Overview of the number of duplicate annotations for each class in SoccerNet-v2 test dataset.

Figure 4.29: Two duplicate *Goal* annotations from a game between Manchester City and Barcelona, from the 2014-2015 Champions League cup.

difference between the annotations in the *Goal* class, and found that the original annotations had more *Goal* annotations than the final score of the game. Figure 4.29 shows one such example. We can see that there are two *Goal* annotations that share the same *gameTime*,

team and *visibility* attributes. We do not think that this has had an effect on the performance achieved by our models.

4.8.6 *Present* model in Models 2 and 3

For our first model (Section 4.2) we introduced the idea of training the *Present* model on visible data only. In Sections 4.3 and 4.4 we continued to use our first model as our *Present* model, despite the poor results in Model 1. This was done in part because we wanted to see if our ideas in Section 4.2 held some merit when we introduced a *Past* and *Present* model. After Section 4.4 we saw that the results were seemingly held back by this poor *Present* model, and decided to change it.

4.8.7 Repeatability of results

As mentioned in Section 4.8.2, the SoccerNet-v2 devkit provided us with five runs (training and evaluation on the test-set) of the NetVLAD++ model. When looking at this model, we see that there are some variations in the Average-mAP score for some of the classes between the different runs. Since we have evaluated the performance achieved by our models on a class-by-class basis, we would ideally have liked to average the class-by-class results of our models over several runs. Unfortunately, we did not have the time or resources to do so, and therefore our models are trained once.

4.8.8 Final reflections on model performance

In Sections 4.2 - 4.6 we have evaluated our models using the Average-mAP method. This lets us compare our efforts in these sections to that of the baseline, as well as other models evaluated on the SoccerNet-v2 dataset. Looking at Table 4.18 we can see that our best performing model, Model 5.2, has a comparative performance to the baseline when spotting off-screen events, but that it is worse in total, and when spotting visible events only. Looking at the classwise comparison between these models in Figure 4.13, we see that Model 5.2 has a comparative or worse performance for all classes except *Clearance* for the overall performance. When spotting visible events only, Model 5.2 is for the most part worse than the baseline, except for *Clearance*. When spotting off-screen events, Model 5.2 has a higher score than the baseline on classes *Throw-in*, *Indirect free-kick*, *Clearance*, and *Yellow card*. When looking at the leader-board for the competition presented in SoccerNet-v2 [16], we can see that our baseline stands at number four for the action spotting task, evaluated on the test dataset. The best performing model [54] achieved an Average-mAP of 73.77, 79.48

and 47.84 for total, visible and off-screen respectively. From this we can safely conclude that our best performing model is not competitive to the current state-of-the-art models when evaluating with the Average-mAP method.

Model	Total Average-mAP	Average-mAP *visible*	Average-mAP *off-screen*
Baseline	53.23	59.24	34.37
Model 5.2	51.17	56.85	34.31

Table 4.18: Overall performance of Model 5.2 compared to the baseline, split by spotting all events, visible events only, and off-screen events only.

In Section 4.7.1 we evaluated our model and the baseline in terms of F1 score, precision and recall and concluded that F1 score was the final metric used to evaluate which model performed best. From Figure 4.20 we can see that the baseline achieved the highest total F1 score of ~0.84, but that our model has a higher F1 score when confidence thresholds are above ~0.67. Choosing a confidence threshold for a use case such as the one mentioned above is difficult. If it is too low, a humans job of filtering out all the false positives will be tedious considering that there are ~500,000 predictions spread across 100 games. If the confidence threshold is too high, we risk losing true positives. In Section 4.7.1, we filtered our predictions on the confidence thresholds that yielded the best F1 score on a classwise basis. The results of this can be seen in Table 4.13. Although the differences are marginal, we can see that our model has a higher recall, and that the baseline has a higher precision and F1 score. As we discussed in Section 4.8.3, we believe that a lower precision can be accepted when it yields a higher recall. To conclude the analysis of the use case in Section 4.7.1, we therefore argue that our model is most suitable.

For our evaluation in Section 4.7.3, we only considered off-screen events in-depth. It is therefore more challenging to draw any conclusions as to what model is overall the most suited to that particular use case. We see from Table 4.15 that for 5 classes, the baseline is able to capture more true positives than our model. Our model is able to capture more true positives for 3 of the classes. Again, we think that an overall high recall (being able to capture true positives) is the most important aspect if one were to apply an imperfect machine learning model in these use cases, given our arguments in Section 4.8.3. If we were to assume that the baseline is also able to capture more true positives of the visible events, we think it is safe to say that the baseline model is preferable to our model in

this use case.

4.8.9 Generalising our idea to other applications

For our evaluation in Section 4.7.1, our model was a able to achieve a higher recall than the baseline. From our evaluation in Section 4.7.3 we saw that our model was able to achieve a higher recall (Figure 4.24) when spotting off-screen events for *Foul*, *Kick-off*, and *Offside*. By comparing Model 5.2 to the baseline in Figure 4.13, we also saw that our model was able to achieve a higher Average-mAP score for *Throw-in*, *Indirect free-kick*, *Clearance*, and *Yellow card* when spotting off-screen events. We believe there is some merit to our idea of combining machine learning models trained on three different contextual dimensions, and we believe that this idea can be generalised to other domains besides soccer. We believe that our architecture can add value to a system if the activity adheres to the following general description:

- The actions that takes place, and the order in which they take place are defined by a strict set of rules.
- The video recording or camera capture of the activity is produced so that some events happen off-screen.

It is important to note that if the video of the activity is able to capture all interesting events, we deem it more fitting to focus efforts on improving the neural networks and its more traditional action recognition capabilities. As our experiments have shown, it is challenging to combine the *Past*, *Present* and *Future* models, and it is challenging to create the correct timestamps for the events produced by the *Past* or *Future* models.

4.9 Retrospect of process

During the initial research-phase of this thesis, we spent a lot of effort getting a firm grasp of the theoretical principles behind neural networks, CNNs and other algorithms, such as PCA and Non-maximum suppression. Therefore, our knowledge and familiarity with the SoccerNet-v2 dataset and the NetVLAD++ model, came in large part as a result of our experiments. This, together with our iterative process of experimenting on each model, has in large part made our work a situation of learning on the job, so to speak. It was quite late in our working process that we actually learned the details of the Average-mAP evaluation function.

After reviewing the results in Sections 4.2 through 4.6, we would have liked to spend more time iterating and experimenting with our models using the evaluation metrics used in Sections 4.7.1 and 4.7.3, namely precision, recall and F1 score. These evaluations arguably provided more opportunities in terms of analysis, compared to the more generalised Average-mAP method. Our experience was that it was easier to draw conclusions in terms of how our model achieved their results using these metrics. One side-effect of using Average-mAP evaluation during development of our models, was that our ideas for improvement between each model in Sections 4.2 through 4.6 were in large part based on intuition and our understanding of soccer, and were not as data-driven as we might have preferred.

In the beginning of our practical part of this thesis, we had some challenges combining the NetVLAD++ model with the features provided by Baidu [54]. In retrospect, we would have liked to have spent some more time on this, since the result would have given us a baseline model closer to the current state-of-the-art.

4.10 Summary

In this chapter, we have presented our experiments and results for action spotting on the SoccerNet-v2 dataset, with a focus on spotting off-screen events. In our initial experiments, we have evaluated the performance achieved by our models against the performance of our baseline, the NetVLAD++ model, using the Average-mAP function. These experiments had varying results, but shows promise, since our model achieved an improved score when spotting off-screen events for some classes.

In the second part of this chapter, we have evaluated our best performing model and the baseline using precision, recall and F1 score, in the context of two simplified real-world applications for an action spotting model. Overall, we showed that the baseline model achieved a higher F1 score, but that our model shows promise, as it achieves a higher recall for our statistics use case, and a higher recall on some specific classes for our game commentary use case.

In the last part of this chapter, we discussed some of the shortcomings, limitations and choices we have made in our work, as well as making some concluding reflections on the performance achieved by our models. We discussed the pros and cons of evaluating with the Average-mAP method, or using precision, recall and F1 score. From this, we also concluded that recall is the most important evaluation metric for our two simplified use cases. Following this, we argued that our best performing model is the most appropriate one for the statistics use case. We also briefly presented an anomaly we found in the

SoccerNet-v2 dataset. Towards the end of this chapter, we discussed how our new idea of combining three models can be applied to other domains. Lastly, we reflected on our process throughout this thesis, and how we would have liked to have done things differently.

Chapter 5

Conclusions

5.1 Summary

Today, the performance of models spotting soccer events are worse when spotting off-screen events, compared to event that are visible in the broadcast. Also, little research is explicitly focused on spotting off-screen events in soccer.

In this thesis, we have presented an intuition for what an understanding of context is in soccer. We have identified relationships between soccer events and used these relationships to create three models that, combined, could make better predictions for off-screen events. We conducted a number of experiments, and made changes in order to improve the performance of our models. We also compared the performance achieved by our model, against an existing action spotting model evaluated on the SoccerNet-v2 dataset.

After our initial experiments we evaluated best performing model in the context of practical use cases and compared our model to the baseline. For our statistics use case, we found that our model overall do predict more correct events, but this comes at the cost of more incorrect predictions as well. In our second use case, game commentary, we compared the performance achieved by our model to the performance of the baseline when spotting off-screen events. Although the baseline was able to capture more true positives for most classes, we saw promise in the fact that our model managed to outperform the baseline for some classes when spotting off-screen events. Towards the end of this thesis we have discussed some of the shortcomings and limitations of our work, as well as some of the choices we have made throughout our work. We also discussed how our new model architecture can be applied to other activities with strict rules, where interesting events are happening off-screen. Finally, we reflected on our working process throughout this thesis.

5.2 Main contributions

As described in Section 1.2, we endeavoured to develop a machine learning model that was able to leverage the strict rules of soccer to better spot off-screen events. In order to solve this problem, we divided the task into objectives, which we can use to illustrate our contributions:

- We defined an intuition for how we can introduce a sense of context into a machine learning model, for it to better be able to spot off-screen events in a soccer broadcast. We identified relationships between events in soccer, and described how these can be used as the basis for creating three dimensions of context in soccer. From this, we discussed how the dataset can be manipulated to train three separate models that reflect the different dimensions of context. We designed a new model architecture that trains and combines a *Past*, *Present* and *Future* model to better predict off-screen events in soccer.

- We iteratively used the existing SoccerNet-v2 labels to create new labels that are used to train the *Past* and *Future* models. Furthermore, we created a data fusion layer that combines the output of the aforementioned models to create one final model output. In order to combine these models in the best way, we experimented with data fusion algorithms, and implemented sliding window algorithms with different window sizes. From this, we found that it is challenging to combine these three models.

- We evaluated our model using the existing Average-mAP evaluation function, and compared the results to our baseline. Our best performing model achieved a higher performance when spotting off-screen events for *Throw-in*, *Indirect free-kick*, *Clearance*, and *Yellow card*.

- We defined two practical use cases, statistics, and game commentary for action spotting models within the soccer domain, and used precision, recall and F1 score as evaluation metrics. Upon reflection, we argued that a higher recall is preferable until a perfect machine learning model is developed. When used for statistics, our model achieved a higher recall score compared to the baseline. When used for game commentary, we only considered spotting off-screen events. Our results showed that our model achieved a higher precision for *Corner* and *Yellow card*, and a higher recall for *Foul*, *Kick-off* and *Offside* compared to the baseline.

These contributions serve to address the question and objectives in our problem statement. In this thesis, we have presented new ideas for how one can improve action spotting on off-screen events. Our results show promise, hopefully facilitating further development of our combined machine learning model architecture. Furthermore, we think that our work explores an exciting challenge in computer vision as a whole, which is understanding how a machine learning model can understand the context within the domain it is applied to.

5.3 Future work

For future work, the time-shifting for our *Past* and *Future* models predictions can be improved. We believe more accurate timestamps would make those models perform better. One exciting prospect would be to introduce the time-shifting as an element when training the models. Including the time between events in the training of the model, might make the model learn what time period should be used for the time-shifting.

It would also be interesting to use a different model than NetVLAD++ [19] to train the *Past*, *Present* and *Future* model. As stated in Section 3.7, NetVLAD++ is not state-of-the-art anymore, and using a Transformer model like Baidu research [54] might achieve better results.

As mentioned in Section 4.8.7, we were not able to train and evaluate our models more than once. In the future, it would be interesting to see the results of our model averaged over several runs.

We also suspect that the fusion algorithm in the fusion layer can be improved. Introducing a weighting system to decide which predictions are used from which model could be interesting. When there are competing predictions between the three models (*Past*, *Present*, *Future*), the one who is included in the final predictions would come from the model that can make the most informed decision by some metrics.

For practical uses, making use of the team-attribute from the SoccerNet-v2 dataset could bring a new dimension to the statistics and game commentary use cases. As we discussed in Section 4.8.4, knowing which team scored a goal is needed to obtain the final result of a game.

Appendix A

Model 1 - additional results

As mentioned in Section 4.2, we hypothesised that a model could achieve better results for spotting visible events if we trained the model using visible events only, ignoring the off-screen events. In order to test this, we removed all off-screen events from the training data and trained a new model using visible events only. The classwise results for this model (Model 1) can be seen in Table A.1.

	Ball out of play	Throw-in	Foul	Ind. free-kick	Clearance	Shots on tar.	Shots off tar.	Corner	Substitution	Kick-off	Yellow card	Offside	Dir. free-kick	Goal	Penalty	Yel. to Red	Red card
Model 1	83.3	78.8	73.0	76.7	40.8	39.5	40.9	69.8	70.7	73.5	65.0	42.7	66.1	79.5	61.2	25.6	22.8
Baseline	86.1	73.1	73.5	76.2	40.2	38.7	40.9	69.3	70.6	72.1	64.6	39.6	65.1	80.6	60.8	25.1	30.7
Difference	-2.8	5.8	-0.6	0.5	0.6	0.8	0.1	0.5	0.1	1.4	0.4	3.1	0.9	-1.1	0.4	0.5	-8.0

Table A.1: Comparing Average-mAP scores per class between Model 1 and the baseline. The Average-mAP scores are for visible events only. A positive difference means that Model 1 performed better, a negative difference means the baseline performed better.

Appendix B

Model 2 - additional results

In Section 4.3, we experimented with our first multimodal approach. We created an algorithm, detailed in Algorithm 1, that combines predictions from our *Past*, *Future* and *Present* models. We experimented with different values for T_w, defined in Section 3.9.2, where $T_w = 40$ achieved the highest overall Average-mAP score. Table B.1 shows the Average-mAP scores for Model 2 with $T_w = 40$ and the baseline.

	Ball out of play	Throw-in	Foul	Ind. free-kick	Clearance	Shots on tar.	Shots off tar.	Corner	Substitution	Kick-off	Yellow card	Offside	Dir. free-kick	Goal	Penalty	Yel. to Red	Red card
All																	
Model 2	77.2	54.4	71.3	68.9	37.5	39.3	40.5	50.4	70.3	67.0	64.5	38.9	56.2	78.6	56.6	2.7	4.8
Baseline	79.1	61.2	72.8	69.1	37.7	38.5	40.6	56.8	70.3	68.8	63.8	44.4	56.4	79.9	56.4	3.2	6.0
Difference	-1.9	-6.7	-1.5	-0.2	-0.2	0.8	0.0	-6.4	0.0	-1.7	0.6	-5.4	-0.2	-1.3	0.2	-0.5	-1.2
Visible																	
Model 2	83.3	78.7	72.3	76.4	40.6	39.5	40.9	69.8	70.6	73.5	65.0	42.5	66.1	79.4	61.2	25.6	22.8
Baseline	86.1	73.1	73.5	76.2	40.2	38.7	40.9	69.3	70.6	72.1	64.6	39.6	65.1	80.6	60.8	25.1	30.7
Difference	-2.8	5.6	-1.2	0.2	0.4	0.8	0.0	0.5	0.1	1.3	0.4	2.9	0.9	-1.1	0.4	0.5	-8.0
Off-screen																	
Model 2	0.0	43.1	0.0	25.1	10.8	3.5	0.2	28.8	63.2	48.8	29.3	35.9	1.3	71.9	12.3	0.0	0.0
Baseline	0.0	55.3	0.0	28.7	11.5	4.4	0.4	44.7	64.8	60.7	35.9	51.7	3.0	73.6	12.4	0.0	0.0
Difference	0.0	-12.2	0.0	-3.6	-0.6	-0.9	-0.2	-15.9	-1.6	-12	-6.7	-15.8	-1.7	-1.7	-0.2	0.0	0.0

Table B.1: Classwise Average-mAP results for Model 2 where $T_W = 40$, compared to the baseline. The results are split between classes and grouped by all, visible and off-screen events. A positive difference means that Model 2 performed better, a negative difference means the baseline performed better.

Appendix C

Model 3 - additional results

Model 3 was presented in Section 4.4, where we changed the label-shifting scheme for the training data. The new label-shifting scheme utilises the event relationships discussed in Section 3.4, which means that the *Past* and *Future* models were only trained using relevant events. The predictions from all three models where concatenated, without use of any data fusion algorithm. The Average-mAP scores for Model 3 and the baseline is presented in Table C.1.

	Ball out of play	Throw-in	Foul	Ind. free-kick	Clearance	Shots on tar.	Shots off tar.	Corner	Substitution	Kick-off	Yellow card	Offside	Dir. free-kick	Goal	Penalty	Yel. to Red	Red card
All																	
Model 3	68.4	24.0	67.6	69.3	36.2	39.3	40.6	42.4	57.1	57.9	61.0	25.9	54.6	77.1	53.5	2.9	6.7
Baseline	79.1	61.2	72.8	69.1	37.7	38.5	40.6	56.8	70.3	68.8	63.8	44.4	56.4	79.9	56.4	3.2	6.0
Difference	-10.7	-37.2	-5.3	0.2	-1.5	0.8	0.0	-14.3	-13.2	-10.8	-2.8	-18.5	-1.8	-2.8	-3.0	-0.3	0.7
Visible																	
Model 3	74.7	40.0	68.4	76.7	38.9	39.5	40.9	58.7	57.8	63.3	61.6	23.8	64.1	78.2	57.7	22.8	26.1
Baseline	86.1	73.1	73.5	76.2	40.2	38.7	40.9	69.3	70.6	72.1	64.6	39.6	65.1	80.6	60.8	25.1	30.7
Difference	-11.4	-33.1	-5.1	0.5	-1.3	0.8	0.0	-10.7	-12.8	-8.9	-2.9	-15.8	-1.0	-2.4	-3.1	-2.3	-4.6
Off-screen																	
Model 3	0.0	17.9	0.0	25.6	11.7	3.5	0.2	25.1	52.2	46.1	28.6	30.4	1.4	69.3	11.6	0.0	0.0
Baseline	0.0	55.3	0.0	28.7	11.5	4.4	0.4	44.7	64.8	60.7	35.9	51.7	3.0	73.6	12.4	0.0	0.0
Difference	0.0	-37.4	0.0	-3.1	0.3	-0.9	-0.2	-19.5	-12.6	-14.6	-7.4	-21.3	-1.6	-4.3	-0.8	0.0	0.0

Table C.1: Classwise Average-mAP results for Model 3, compared to the baseline. The results are split between classes and grouped by all, visible and off-screen events. A positive difference means that Model 3 performed better, a negative difference means the baseline performed better.

Appendix D

Model 4.1 - additional results

With Model 4.1, presented in Section 4.5, we made some changes to the training data for the *Past* and *Future* models. We modified our label-shifting scheme to check the *two* preceding/following annotations for an annotation of a related event, instead of just the first preceding/following annotation. This was done to account for the fact that related events does not always directly follow/precede each other in a game, but might have other unrelated events happening in between. We also trained our *Past* and *Future* models on visible events only, by discarding all off-screen annotations from the training data used by those models. The Average-mAP scores for Model 4.1 and Model 3 are presented in Table D.1.

	Ball out of play	Throw-in	Foul	Ind. free-kick	Clearance	Shots on tar.	Shots off tar.	Corner	Substitution	Kick-off	Yellow card	Offside	Dir. free-kick	Goal	Penalty	Yel. to Red	Red card
All																	
Model 4.1	71.1	20.1	71.3	69.3	37.6	39.3	40.6	40.7	57.8	57.1	56.7	27.2	53.8	76.1	54.9	2.9	6.4
Model 3	68.4	24.0	67.6	69.3	36.2	39.3	40.6	42.4	57.1	57.9	61.0	25.9	54.6	77.1	53.5	2.9	6.7
Baseline	79.1	61.2	72.8	69.1	37.7	38.5	40.6	56.8	70.3	68.8	63.8	44.4	56.4	79.9	56.4	3.2	6.0
Visible																	
Model 4.1	77.8	38.5	72.3	76.7	40.7	39.5	40.9	56.4	58.7	61.7	57.3	25.4	63.2	77.3	59.2	22.4	24.2
Model 3	74.7	40.0	68.4	76.7	38.9	39.5	40.9	58.7	57.8	63.3	61.6	23.8	64.1	78.2	57.7	22.8	26.1
Baseline	86.1	73.1	73.5	76.2	40.2	38.7	40.9	69.3	70.6	72.1	64.6	39.6	65.1	80.6	60.8	25.1	30.7
Off-screen																	
Model 4.1	0.0	14.7	0.0	25.6	10.9	3.5	0.2	24.9	50.2	47.9	26.1	30.9	1.7	66.8	11.8	0.0	0.0
Model 3	0.0	17.9	0.0	25.6	11.7	3.5	0.2	25.1	52.2	46.1	28.6	30.4	1.4	69.3	11.6	0.0	0.0
Baseline	0.0	55.3	0.0	28.7	11.5	4.4	0.4	44.7	64.8	60.7	35.9	51.7	3.0	73.6	12.4	0.0	0.0

Table D.1: Classwise Average-mAP results for Model 4.1, compared to Model 3 and the baseline. The results are split between classes and grouped by all, visible and off-screen events.

Appendix E

Model 4.2 - additional results

In Model 4.2, we replaced the *Present* model used in Model 4.1 with the NetVLAD++ model, trained using all events in the dataset. The classwise Average-mAP scores achieved by this modification, and the classwise Average-mAP scores of the baseline, can be seen in Table E.1.

	Ball out of play	Throw-in	Foul	Ind. free-kick	Clearance	Shots on tar.	Shots off tar.	Corner	Substitution	Kick-off	Yellow card	Offside	Dir. free-kick	Goal	Penalty	Yel. to Red	Red card
All																	
Model 4.2	69.7	44.6	71.7	69.8	39.0	39.1	40.9	47.9	57.0	55.6	52.6	34.5	53.5	76.4	54.1	4.7	3.7
Model 3	68.4	24.0	67.6	69.3	36.2	39.3	40.6	42.4	57.1	57.9	61.0	25.9	54.6	77.1	53.5	2.9	6.7
Baseline	79.1	61.2	72.8	69.1	37.7	38.5	40.6	56.8	70.3	68.8	63.8	44.4	56.4	79.9	56.4	3.2	6.0
Visible																	
Model 4.2	75.7	59.1	72.5	76.1	41.5	39.2	41.4	59.4	57.9	59.0	53.0	29.7	60.9	77.3	58.3	27.7	24.8
Model 3	74.7	40.0	68.4	76.7	38.9	39.5	40.9	58.7	57.8	63.3	61.6	23.8	64.1	78.2	57.7	22.8	26.1
Baseline	86.1	73.1	73.5	76.2	40.2	38.7	40.9	69.3	70.6	72.1	64.6	39.6	65.1	80.6	60.8	25.1	30.7
Off-screen																	
Model 4.2	0.0	38.0	0.0	35.8	11.9	3.9	0.2	37.1	51.4	48.8	31.3	41.9	2.8	70.1	12.6	0.0	0.0
Model 3	0.0	17.9	0.0	25.6	11.7	3.5	0.2	25.1	52.2	46.1	28.6	30.4	1.4	69.3	11.6	0.0	0.0
Baseline	0.0	55.3	0.0	28.7	11.5	4.4	0.4	44.7	64.8	60.7	35.9	51.7	3.0	73.6	12.4	0.0	0.0

Table E.1: Classwise Average-mAP results for Model 4.2, compared to Model 3 and the baseline. The results are split between classes and grouped by all, visible and off-screen events.

Appendix F

Model 4.3 - additional results

For Model 4.3, we challenged some assumptions made when creating Models 4.1 and 4.2. First, we trained the *Past* and *Future* models on all events, both *visible* and *off-screen*, as this gave better results for the *Present* model. Secondly, we increased the number of preceding/following annotations to check for an annotation of a related event. This number was increased from 2 to 10 and the process is more detailed in Section 4.5. Table F.1 shows the classwise Average-mAP scores for Model 4.3.

	Ball out of play	Throw-in	Foul	Ind. free-kick	Clearance	Shots on tar.	Shots off tar.	Corner	Substitution	Kick-off	Yellow card	Offside	Dir. free-kick	Goal	Penalty	Yel. to Red	Red card
All																	
Model 4.3	70.9	49.2	67.7	69.8	38.7	39.1	40.9	49.2	56.5	57.9	52.0	37.4	55.7	76.0	54.7	2.9	6.3
Model 4.2	69.7	44.6	71.7	69.8	39.0	39.1	40.9	47.9	57.0	55.6	52.6	34.5	53.5	76.4	54.1	4.7	3.7
Baseline	79.1	61.2	72.8	69.1	37.7	38.5	40.6	56.8	70.3	68.8	63.8	44.4	56.4	79.9	56.4	3.2	6.0
Visible																	
Model 4.3	76.8	62.6	68.5	76.1	41.0	39.2	41.4	60.7	57.4	62.0	52.1	32.8	63.7	77.0	59.1	24.1	28.6
Model 4.2	75.7	59.1	72.5	76.1	41.5	39.2	41.4	59.4	57.9	59.0	53.0	29.7	60.9	77.3	58.3	27.7	24.8
Baseline	86.1	73.1	73.5	76.2	40.2	38.7	40.9	69.3	70.6	72.1	64.6	39.6	65.1	80.6	60.8	25.1	30.7
Off-screen																	
Model 4.3	0.0	42.5	0.0	35.8	12.1	3.9	0.2	37.9	51.9	48.7	33.6	43.6	2.7	68.7	12.4	0.0	0.0
Model 4.2	0.0	38.0	0.0	35.8	11.9	3.9	0.2	37.1	51.4	48.8	31.3	41.9	2.8	70.1	12.6	0.0	0.0
Baseline	0.0	55.3	0.0	28.7	11.5	4.4	0.4	44.7	64.8	60.7	35.9	51.7	3.0	73.6	12.4	0.0	0.0

Table F.1: Classwise Average-mAP results for Model 4.3, compared to Model 4.2 and the baseline. The results are split between classes and grouped by all, visible and off-screen events.

Appendix G

Model 5.2 - additional results

In Section 4.6, we discovered predictions with a negative timestamp, which were removed in Model 5.2. We also experimented with new fusion algorithms, using both overlapping and non-overlapping sliding windows, as well as different values for the window size t. In Table G.1, the classwise Average-mAP scores are presented for Model 5.2 with a non-overlapping fusion algorithm and t = 15 seconds.

	Ball out of play	Throw-in	Foul	Ind. free-kick	Clearance	Shots on tar.	Shots off tar.	Corner	Substitution	Kick-off	Yellow card	Offside	Dir. free-kick	Goal	Penalty	Yel. to Red	Red card
All																	
Model 5.2	77.7	61.3	72.4	69.7	40.1	39.1	41.0	53.9	63.7	58.7	54.2	40.6	56.9	78.3	54.1	3.9	4.5
Baseline	79.1	61.2	72.8	69.1	37.7	38.5	40.6	56.8	70.3	68.8	63.8	44.4	56.4	79.9	56.4	3.2	6.0
Difference	-1.4	0.1	-0.4	0.6	2.4	0.6	0.4	-2.9	-6.6	-10.1	-9.6	-3.8	0.5	-1.5	-2.3	0.7	-1.5
Visible																	
Model 5.2	84.0	68.5	73.3	76.1	42.5	39.3	41.4	65.0	63.8	63.1	54.4	36.8	65.0	79.3	58.5	30.8	24.7
Baseline	86.1	73.1	73.5	76.2	40.2	38.7	40.9	69.3	70.6	72.1	64.6	39.6	65.1	80.6	60.8	25.1	30.7
Difference	-2.1	-4.5	-0.3	-0.1	2.3	0.6	0.5	-4.4	-6.8	-9.0	-10.1	-2.8	-0.1	-1.2	-2.3	5.7	-6.0
Off-screen																	
Model 5.2	0.0	57.8	0.0	35.0	13.3	3.9	0.2	43.4	63.9	48.9	45.1	46.6	2.9	72.4	12.6	0.0	0.0
Baseline	0.0	55.3	0.0	28.7	11.5	4.4	0.4	44.7	64.8	60.7	35.9	51.7	3.0	73.6	12.4	0.0	0.0
Difference	0.0	2.6	0.0	6.3	1.9	-0.4	-0.1	-1.3	-0.9	-11.9	9.2	-5.1	-0.1	-1.1	0.2	0.0	0.0

Table G.1: Classwise Average-mAP results for Model 5.2 compared to our baseline. Model 5.2 uses a non-overlapping fusing algorithm and $t = 15s$. The results are split between classes and grouped by all, visible and off-screen events. A positive difference means that Model 5.2 performed better, a negative difference means the baseline performed better.

Appendix H

Precision, recall and F1 score per class for statistics use case

In Section 4.7.1, we defined a use case for evaluating Model 5.2 in a more practical context. We defined predictions as true positive, false positive or false negative in order to calculate precision, recall and F1 score. In Table H.1, precision scores are calculated per class for Model 5.2 at different confidence thresholds. Table H.2 contains recall scores per class for Model 5.2 at different confidence thresholds. In Table H.3, F1 scores for Model 5.2 are calculated per class for different confidence thresholds, while Table H.4 has the same F1 scores calculated for the NetVLAD++ baseline.

Confidence $\geq$	Ball out of play	Throw-in	Foul	Ind. free-kick	Clearance	Shots on tar.	Shots off tar.	Corner	Substitution	Kick-off	Yellow card	Offside	Dir. free-kick	Goal	Penalty	Yel. to Red	Red card
0.0	0.18	0.11	0.07	0.07	0.05	0.04	0.04	0.03	0.02	0.01	0.01	0.01	0.01	0.01	0.00	0.00	0.00
0.1	0.26	0.20	0.14	0.14	0.13	0.22	0.29	0.12	0.40	0.11	0.13	0.11	0.10	0.12	0.28	0.00	0.21
0.2	0.32	0.29	0.23	0.24	0.20	0.34	0.45	0.17	0.62	0.19	0.25	0.37	0.20	0.23	0.46	*NaN*	0.00
0.3	0.38	0.37	0.35	0.38	0.29	0.48	0.62	0.22	0.79	0.28	0.38	0.70	0.33	0.35	0.60	*NaN*	*NaN*
0.4	0.43	0.46	0.50	0.58	0.40	0.67	0.80	0.29	0.89	0.39	0.53	0.84	0.45	0.48	0.79	*NaN*	*NaN*
0.5	0.50	0.57	0.70	0.84	0.58	0.83	0.89	0.38	0.93	0.53	0.67	0.97	0.56	0.61	0.91	*NaN*	*NaN*
0.6	0.59	0.70	0.89	0.97	0.81	0.95	0.96	0.52	0.97	0.66	0.79	1.00	0.64	0.75	0.92	*NaN*	*NaN*
0.7	0.72	0.84	0.99	0.98	0.97	1.00	0.99	0.69	1.00	0.78	0.89	1.00	0.70	0.90	0.95	*NaN*	*NaN*
0.8	0.92	0.96	1.00	0.99	1.00	1.00	1.00	0.88	1.00	0.90	0.97	1.00	0.77	0.95	1.00	*NaN*	*NaN*
0.9	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.96	1.00	0.96	0.98	1.00	0.85	1.00	1.00	*NaN*	*NaN*

Table H.1: Classwise precision scores for Model 5.2, when filtering the predictions on different confidence thresholds. Our model does not have any *Yellow then red card* or *Red card* predictions with confidence scores above or equal to 0.2 and 0.3 respectively, giving the value *NaN*.

Confidence $\geq$	Ball out of play	Throw-in	Foul	Ind. free-kick	Clearance	Shots on tar.	Shots off tar.	Corner	Substitution	Kick-off	Yellow card	Offside	Dir. free-kick	Goal	Penalty	Yel. to Red	Red card
0.0	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
0.1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.95	0.00	0.38
0.2	1.00	1.00	1.00	1.00	1.00	1.00	0.98	1.00	0.97	1.00	1.00	0.95	1.00	1.00	0.88	*NaN*	0.00
0.3	1.00	1.00	1.00	0.99	1.00	0.97	0.89	1.00	0.91	1.00	1.00	0.76	0.99	0.98	0.85	*NaN*	*NaN*
0.4	1.00	1.00	1.00	0.95	1.00	0.87	0.75	1.00	0.80	1.00	0.96	0.49	0.97	0.97	0.80	*NaN*	*NaN*
0.5	1.00	1.00	0.98	0.79	0.98	0.71	0.54	1.00	0.68	0.99	0.87	0.33	0.91	0.93	0.73	*NaN*	*NaN*
0.6	1.00	0.98	0.87	0.46	0.88	0.48	0.33	1.00	0.59	0.96	0.80	0.22	0.84	0.85	0.59	*NaN*	*NaN*
0.7	0.99	0.93	0.63	0.20	0.60	0.27	0.18	0.99	0.50	0.89	0.69	0.16	0.75	0.74	0.49	*NaN*	*NaN*
0.8	0.92	0.74	0.35	0.06	0.32	0.12	0.06	0.93	0.39	0.74	0.57	0.11	0.64	0.61	0.34	*NaN*	*NaN*
0.9	0.49	0.39	0.10	0.00	0.10	0.03	0.01	0.76	0.26	0.50	0.35	0.04	0.44	0.35	0.17	*NaN*	*NaN*

Table H.2: Classwise recall scores for Model 5.2, when filtering the predictions on different confidence thresholds. Our model does not have any *Yellow then red card* or *Red card* predictions with confidence scores above or equal to 0.2 and 0.3 respectively, giving the value *NaN*.

Confidence >=	Ball out of play	Throw-in	Foul	Ind. free-kick	Clearance	Shots on tar.	Shots off tar.	Corner	Substitution	Kick-off	Yellow card	Offside	Dir. free-kick	Goal	Penalty	Yel. to Red	Red card
0.0	0.31	0.20	0.13	0.12	0.09	0.08	0.07	0.06	0.04	0.03	0.02	0.02	0.02	0.02	0.00	0.00	0.00
0.1	0.42	0.34	0.25	0.25	0.24	0.36	0.44	0.21	0.57	0.19	0.24	0.20	0.17	0.21	0.44	0.00	0.27
0.2	0.49	0.45	0.38	0.39	0.34	0.51	0.61	0.29	0.76	0.32	0.40	0.53	0.34	0.38	0.60	*NaN*	0.27
0.3	0.55	0.54	0.52	0.55	0.45	0.65	0.73	0.36	0.85	0.44	0.55	0.73	0.49	0.52	0.71	*NaN*	*NaN*
0.4	0.60	0.63	0.67	0.72	0.58	0.76	0.77	0.45	0.84	0.57	0.68	0.62	0.61	0.64	0.80	*NaN*	*NaN*
0.5	0.67	0.73	0.82	0.81	0.73	0.76	0.67	0.55	0.78	0.69	0.75	0.49	0.69	0.74	0.81	*NaN*	*NaN*
0.6	0.74	0.81	0.88	0.63	0.85	0.64	0.50	0.68	0.73	0.78	0.80	0.36	0.72	0.80	0.72	*NaN*	*NaN*
0.7	0.84	0.88	0.77	0.34	0.75	0.42	0.30	0.81	0.67	0.83	0.78	0.28	0.73	0.81	0.65	*NaN*	*NaN*
0.8	0.92	0.83	0.52	0.11	0.49	0.22	0.11	0.90	0.56	0.81	0.72	0.19	0.70	0.74	0.51	*NaN*	*NaN*
0.9	0.66	0.56	0.18	0.00	0.18	0.06	0.01	0.85	0.42	0.66	0.51	0.07	0.58	0.52	0.29	*NaN*	*NaN*

Table H.3: Classwise F1 scores for Model 5.2, when filtering the predictions on different confidence thresholds. Our model does not have any *Yellow then red card* or *Red card* predictions with confidence scores above or equal to 0.2 and 0.3 respectively, giving the value *NaN*.

Confidence >=	Ball out of play	Throw-in	Foul	Ind. free-kick	Clearance	Shots on tar.	Shots off tar.	Corner	Substitution	Kick-off	Yellow card	Offside	Dir. free-kick	Goal	Penalty	Yel. to Red	Red card
0.0	0.38	0.24	0.16	0.15	0.11	0.08	0.07	0.07	0.04	0.04	0.03	0.03	0.03	0.02	0.00	0.00	0.00
0.1	0.56	0.52	0.36	0.37	0.36	0.36	0.44	0.45	0.57	0.32	0.33	0.25	0.26	0.31	0.44	0.00	0.27
0.2	0.66	0.68	0.56	0.54	0.52	0.51	0.61	0.59	0.76	0.48	0.52	0.59	0.41	0.49	0.60	*NaN*	0.27
0.3	0.74	0.80	0.72	0.72	0.67	0.65	0.73	0.69	0.85	0.61	0.66	0.73	0.53	0.61	0.71	*NaN*	*NaN*
0.4	0.80	0.88	0.85	0.81	0.79	0.76	0.77	0.77	0.84	0.72	0.75	0.61	0.62	0.70	0.80	*NaN*	*NaN*
0.5	0.87	0.91	0.89	0.77	0.87	0.76	0.67	0.82	0.78	0.81	0.78	0.49	0.69	0.78	0.81	*NaN*	*NaN*
0.6	0.93	0.87	0.81	0.54	0.83	0.64	0.50	0.87	0.73	0.80	0.80	0.36	0.73	0.82	0.72	*NaN*	*NaN*
0.7	0.93	0.79	0.65	0.30	0.66	0.42	0.30	0.90	0.67	0.78	0.78	0.28	0.73	0.80	0.65	*NaN*	*NaN*
0.8	0.80	0.66	0.43	0.11	0.44	0.22	0.11	0.91	0.56	0.69	0.72	0.19	0.70	0.72	0.51	*NaN*	*NaN*
0.9	0.49	0.47	0.15	0.00	0.18	0.06	0.01	0.85	0.42	0.56	0.51	0.07	0.58	0.51	0.29	*NaN*	*NaN*

Table H.4: Classwise F1 scores for our baseline, when filtering the predictions on different confidence thresholds. The baseline does not have any *Yellow then red card* or *Red card* predictions with confidence scores above or equal to 0.2 and 0.3 respectively, giving the value *NaN*.

Bibliography

[1] Theo Ajadi, Tim Bridge, Tom Hammond, Chris Hanson and Zal Udwadia. *Testing times: Football Money League*. Tech. rep. Manchester, UK: Deloitte Sports Business Group, Jan. 2021.

[2] Saad Albawi, Tareq A. Mohammed and Saad Al-Zawi. 'Understanding of a Convolutional Neural Network'. In: *CET 2017 : The International Conference on Engineering & Technology 2017*. Aug. 2017.

[3] Md Zahangir Alom, Vijayan K. Asari, Abdul A S. Awwal, Brian C Van Esesn, Mst Shamima Nasrin, Paheding Sidike, Tarek M. Taha, Stefan Westberg and Christopher Yakopcic. *The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches*. Accessed: 27.04.2021. 2018. arXiv: 1803.01164 [cs.CV].

[4] Relja Arandjelović, Petr Gronat, Akihiko Torii, Tomas Pajdla and Josef Sivic. *NetVLAD: CNN architecture for weakly supervised place recognition*. 2016. arXiv: 1511.07247 [cs.CV].

[5] Biagio Brattoli, Krzysztof Chalupka, Pietro Perona, Joseph Tighe and Fedor Zhdanov. 'Rethinking Zero-Shot Video Classification: End-to-End Training for Realistic Applications'. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020.

[6] Michael Buckland and Fredric Gey. 'The relationship between recall and precision'. In: *Journal of the American society for information science* 45.1 (1994), pp. 12–19.

[7] Fabian Caba Heilbron, Victor Escorcia, Bernard Ghanem and Juan Carlos Niebles. 'ActivityNet: A Large-Scale Video Benchmark for Human Activity Understanding'. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2015.

[8] Enrique J. Carmona, Jose M. Chaquet and Antonio Fernández-Caballero. 'A survey of video datasets for human action and activity recognition'. In: *Computer Vision and Image Understanding* 117.6 (2013), pp. 633–659. ISSN: 1077-3142. DOI: https://

doi.org/10.1016/j.cviu.2013.01.013. URL: https://www.sciencedirect.com/science/article/pii/S1077314213000295.

[9] Federico Castanedo. 'A review of Data Fusion Techniques'. In: 2013 (2013). DOI: 10.1155/2013/704504. URL: https://doi.org/10.1155/2013/704504.

[10] Wei Chen, Jason J. Corso, Caiming Xiong and Ran Xu. 'Actionness Ranking with Lattice Conditional Ordinal Random Fields'. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2014.

[11] Anthony Cioppa, Adrien Deliege, Marc Van Droogenbroeck, Jacob V. Dueholm, Bernard Ghanem, Silvio Giancola, Thomas B. Moeslund, Kamal Nasrollahi and Meisam J. Seikavandi. 'SoccerNet-v2: A Dataset and Benchmarks for Holistic Understanding of Broadcast Soccer Videos'. arXiv:2011.13367v1. 2020.

[12] Anthony Cioppa, Adrien Deliege, Marc Van Droogenbroeck, Rikke Gade, Bernard Ghanem, Silvio Giancola and Thomas B. Moeslund. 'A Context-Aware Loss Function for Action Spotting in Soccer Videos'. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020.

[13] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li and Li Fei-Fei. 'ImageNet: A large-scale hierarchical image database'. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[14] P.J. Denning, D.E. Comer, D. Gries, M.C. Mulder, A. Tucker, A.J. Turner and P.R. Young. 'Computing as a discipline'. In: *Computer* 22.2 (1989), pp. 63–70. DOI: 10.1109/2.19833.

[15] ESPN. *ESPN soccer match commentary*. Mar. 2022. URL: https://www.espn.com/soccer/commentary?gameId=605754.

[16] EvalAi. *SoccerNet-v2 competition leaderboard*. Apr. 2022. URL: https://eval.ai/web/challenges/challenge-page/761/leaderboard/2072.

[17] Yu-Gang. Jiang, Jingen Liu, Amir Roshan Zamir, George Toderici, Ivan Laptev, Mubarak Shah and Rahul Suk-thankar. *THUMOS Challenge 2014*. http://crcv.ucf.edu/THUMOS14/index.html. Accessed: 2021-05-05. 2014.

[18] Silvio Giancola, Mohieddine Amine, Tarek Dghaily and Bernard Ghanem. 'SoccerNet: A Scalable Dataset for Action Spotting in Soccer Videos'. In: *CoRR* abs/1804.04527 (2018). arXiv: 1804.04527. URL: http://arxiv.org/abs/1804.04527.

[19] Silvio Giancola and Bernard Ghanem. *Temporally-Aware Feature Pooling for Action Spotting in Soccer Broadcasts*. 2021. arXiv: 2104.06779 [cs.CV].

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun. 'Deep Residual Learning for Image Recognition'. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016.

[21] Namdar Homayounfar, Sanja Fidler and Raquel Urtasun. 'Sports Field Localization via Deep Structured Models'. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 4012–4020. DOI: 10.1109/CVPR.2017.427.

[22] The IFAB. *LAW 16 The GOAL KICK*. Feb. 2022. URL: https://www.theifab.com/laws/latest/the-goal-kick/.

[23] The IFAB. *LAW 9 THE BALL IN AND OUT OF PLAY*. Feb. 2022. URL: https://www.theifab.com/laws/latest/the-ball-in-and-out-of-play/.

[24] Yudong Jiang, Kaixu Cui, Leilei Chen, Canjin Wang and Changliang Xu. 'SoccerDB'. In: *Proceedings of the 3rd International Workshop on Multimedia Content Analysis in Sports* (Oct. 2020). DOI: 10.1145/3422844.3423051. URL: http://dx.doi.org/10.1145/3422844.3423051.

[25] Longlong Jing, Toufiq Parag, Zhe Wu, Yingli Tian and Hongcheng Wang. 'VideoSSL: Semi-Supervised Learning for Video Classification'. arXiv:2003.00197. 2021.

[26] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar and Li Fei-Fei. 'Large-scale Video Classification with Convolutional Neural Networks'. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2014.

[27] Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, Mustafa Suleyman and Andrew Zisserman. *The Kinetics Human Action Video Dataset*. 2017. arXiv: 1705.06950 [cs.CV].

[28] Alex Krizhevsky, Ilya Sutskever and Geoffrey E Hinton. 'ImageNet Classification with Deep Convolutional Neural Networks'. In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou and K. Q. Weinberger. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[29] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio and T. Serre. 'HMDB: A large video database for human motion recognition'. In: *2011 International Conference on Computer Vision*. 2011, pp. 2556–2563. DOI: 10.1109/ICCV.2011.6126543.

[30] Hilde Kuehne, Hueihan Jhuang, Estibaliz Garrote, Tomaso Poggio and Thomas Serre. 'HMDB51: A Large Video Database for Human Motion Recognition'. In: Nov. 2011, pp. 2556–2563. ISBN: 978-3-642-33373-6. DOI: 10.1109/ICCV.2011.6126543.

[31] David Lange. *Market size of the European professional football market from 2006/07 to 2018/19*. https://www.statista.com/statistics/261223/european-soccer-market-total-revenue/. Accessed: 2021-04-27. 2020.

[32] Stephen Marshland. *Machine Learning: An Algorithmic perspective*. 2nd ed. Boca Raton, Florida: Chapman & Hall/CRC, 2015.

[33] Hiroaki Minoura, Tsubasa Hirakawa, Takayoshi Yamashita, Hironobu Fujiyoshi, Mitsuru Nakazawa, Yeongnam Chae and Björn Stenger. 'Action Spotting and Temporal Attention Analysis in Soccer Videos'. In: *2021 17th International Conference on Machine Vision and Applications (MVA)*. 2021, pp. 1–6. DOI: 10.23919/MVA51890.2021.9511342.

[34] Mehryar Mohri, Afshin Rostamizadeh and Ameet Talwalkar. *Foundations of Machine Learning*. 2nd ed. Cambridge, Massachusetts: The MIT Press, 2018.

[35] Olav A. Nergård Rongved, Steven A. Hicks, Vajira Thambawita, Håkon K. Stensland, Evi Zouganeli, Dag Johansen, Michael A. Riegler and Pål Halvorsen. 'Real-Time Detection of Events in Soccer Videos using 3D Convolutional Neural Networks'. In: *2020 IEEE International Symposium on Multimedia (ISM)*. 2020, pp. 135–144. DOI: 10.1109/ISM.2020.00030.

[36] Olav Andre Nergård Rongved, Markus Stige, Steven Alexander Hicks, Vajira Lasantha Thambawita, Cise Midoglu, Evi Zouganeli, Dag Johansen, Michael Alexander Riegler and Pål Halvorsen. 'Automated Event Detection and Classification in Soccer: The Potential of Using Multiple Modalities'. In: *Machine Learning and Knowledge Extraction* 3.4 (2021), pp. 1030–1054. ISSN: 2504-4990. DOI: 10.3390/make3040051. URL: https://www.mdpi.com/2504-4990/3/4/51.

[37] Luca Pappalardo, Paolo Cintia, Alessio Rossi, Emanuele Massucco, Paolo Ferragina, Dino Pedreschi and Fosca Giannotti. 'A public data set of spatio-temporal match events in soccer competitions'. In: *Scientific Data* 6.236 (2019).

[38] Olav A. Nergård Rongved, Steven A. Hicks, Vajira Thambawita, Håkon K. Stensland, Evi Zouganeli, Dag Johansen, Cise Midoglu, Michael A. Riegler and Pål Halvorsen. 'Using 3D Convolutional Neural Networks for Real-time Detection of Soccer Events'. In: *International Journal of Semantic Computing* 15.02 (2021), pp. 161–187. DOI: 10.1142/S1793351X2140002X. eprint: https://doi.org/10.1142/S1793351X2140002X. URL: https://doi.org/10.1142/S1793351X2140002X.

[39] Nabile M. Safdar, John D. Banja and Carolyn C. Meltzer. 'Ethical considerations in artificial intelligence'. In: *European Journal of Radiology* 122 (2020), p. 108768. ISSN: 0720-048X. DOI: https://doi.org/10.1016/j.ejrad.2019.108768. URL: https://www.sciencedirect.com/science/article/pii/S0720048X19304188.

[40] Omer Sagi and Lior Rokach. 'Ensemble learning: A survey'. In: *WIREs Data Mining and Knowledge Discovery* 8.4 (2018), e1249. DOI: https://doi.org/10.1002/widm.1249. eprint: https://wires.onlinelibrary.wiley.com/doi/pdf/10.1002/widm.1249. URL: https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/widm.1249.

[41] Gunnar A Sigurdsson, Gul Varol, Xiaolong Wang, Ali Farhadi, Ivan Laptev and Abhinav Gupta. 'Hollywood in Homes: Crowdsourcing Data Collection for Activity Understanding'. In: *Computer Vision - ECCV 2016*. Cham: Springer International Publishing, 2016, pp. 510–526. ISBN: 978-3-319-46448-0.

[42] Gunnar A. Sigurdsson, Olga Russakovsky and Abhinav Gupta. 'What Actions Are Needed for Understanding Human Actions in Videos?' In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. Oct. 2017.

[43] Cees G. M. Snoek, Marcel Worring and Arnold W. M. Smeulders. 'Early versus Late Fusion in Semantic Video Analysis'. In: *Proceedings of the 13th Annual ACM International Conference on Multimedia*. MULTIMEDIA '05. Hilton, Singapore: Association for Computing Machinery, 2005, pp. 399–402. ISBN: 1595930442. DOI: 10.1145/1101149.1101236. URL: https://doi.org/10.1145/1101149.1101236.

[44] *SoccerNet-v2 challenge - Tutorial #1 (live session)*. URL: https://www.youtube.com/watch?v=n-fWUr2SKqQ.

[45] Khurram Soomro, Amir Roshan Zamir and Mubarak Shah. *UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild*. 2012. arXiv: 1212.0402 [cs.CV].

[46] Richard Szeliski. *Computer Vision: Algorithms And Applications*. London: Springer, 2011.

[47] Graham Thomas, Rikke Gade, Thomas Moeslund, Peter Carr and Adrian Hilton. 'Computer vision for sports: Current applications and research topics'. In: *Computer Vision and Image Understanding* 159 (Apr. 2017). DOI: 10.1016/j.cviu.2017.04.011.

[48] Matteo Tomei, Lorenzo Baraldi, Simone Calderara, Simone Bronzin and Rita Cucchiara. *RMS-Net: Regression and Masking for Soccer Event Spotting*. 2021. arXiv: 2102.07624 [cs.CV].

[49] Bastien Vanderplaetse and Stephane Dupont. 'Improved Soccer Action Spotting Using Both Audio and Video Streams'. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. June 2020.

[50] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser and Illia Polosukhin. 'Attention is all you need'. In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.

[51] Kanav Vats, Mehrnaz Fani, Pascale Walters, David A. Clausi and John Zelek. *Event detection in coarsely annotated sports videos via parallel multi receptive field 1D convolutions*. 2020. arXiv: 2004.06172 [cs.CV].

[52] S. Yeung, O. Russakovsky, N. Jin, M. Andriluka, G. Mori and L. Fei-Fei. 'Every Moment Counts: Dense Detailed Labeling of Actions in Complex Videos'. In: *Scientific Data* 126 (2018), pp. 375–389.

[53] Junqing Yu, Aiping Lei, Zikai Song, Tingting Wang, Hengyou Cai and Na Feng. 'Comprehensive Dataset of Broadcast Soccer Videos'. In: *2018 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*. 2018, pp. 418–423. DOI: 10.1109/MIPR.2018.00090.

[54] Xin Zhou, Le Kang, Zhiyu Cheng, Bo He and Jingyu Xin. 'Feature Combination Meets Attention: Baidu Soccer Embeddings and Transformer based Temporal Detection'. In: *CoRR* abs/2106.14447 (2021). arXiv: 2106.14447. URL: https://arxiv.org/abs/2106.14447.