Using Temporal Convolutional Networks to estimate ball possession in soccer games

# Using Temporal Convolutional Networks to Estimate Ball Possession in Soccer Games

Matteo Borghesi[a], Lorenzo Dusty Costa[b], Lia Morra[a], Fabrizio Lamberti[a]

[a] *Department of Control and Computer Engineering, Politecnico di Torino, Corso Duca degli Abruzzi 24, Torino, 10129, Italy*
[b] *Math&Sport, Via Privata Deruta 20, Milano, 20132, Italy*

*Email addresses:* `s268199@studenti.polito.it` (Matteo Borghesi), `lorenzo.costa@mathandsport.com` (Lorenzo Dusty Costa), `lia.morra@polito.it` (Lia Morra), `fabrizio.lamberti@polito.it` (Fabrizio Lamberti)

## Abstract

The use of tracking data in the field of sport analytics has increased in the last years as a starting point for in-depth tactical analyses. This work investigates the use of Temporal Convolutional Networks (TCNs), a powerful architecture for sequential data analysis, to extract ball possession information from tracking data. This task is a crucial step for many tactical analysis and is nowadays carried out manually by a human operator in the stadium, which is costly, difficult to implement, and prone to errors. In this work, several classification approaches are explored to classify the game state as dead, ball owned by the home team, or by the away team: as a single-branch, ternary prediction, or as two binary predictions, first detecting whether the game is dead or alive and then which team owns the ball. TCNs are exploited to create independent trajectory embeddings from tracking data of each object; since there is no semantic ordering among the tracked objects, we investigate different permutation-invariant layers to combine the embeddings, namely, an element-wise sum over the embeddings, a self-attention module, and the use of 2D convolutions. Performance evaluation on tracking data from professional soccer games shows that the proposed method outperforms state-of-the-art rule-based methods, achieving 86.2% accuracy in possession estimation (+7.3% compared to the state of the art) and 89.2% accuracy in dead-alive classification (+33.2% compared to the state of the art). Extensive ablation studies were conducted to investigate how different input data concur to the final prediction.

*Keywords:* Sport analytics, Deep learning, Tracking data, Ball possession, Game state, Temporal Convolutional Networks

## 1. Introduction

In recent years, the field of sport analytics has received increasingly attention, as it has been realized that the systematical analysis of the big amount of data produced by sports daily can help to develop strategies capable of increasing the chances of winning a match.

In this paper, we focus on the automatic extraction of ball possession information from a soccer game from spatio-temporal tracking data. In typical soccer analytics pipelines, estimating ball possession and game state is the first step in understanding the events that occur in a game and their relationship. Without this information, only physical quantities, e.g., on covered distance and speeds, can be measured and aggregated. In turn, the availability of a ball possession estimation component opens to a wider range of analyses: besides computing simple game state statistics, it becomes possible to analyze every single pass, to split the game in actions, to classify them and to study players and team behaviors in attack and defense phases, etc.

The need to develop this component stems from considering that raw tracking data about players and ball positions are today commonly extracted by specialized companies and made available for the world's top leagues such as Premier League, Bundesliga, LaLiga and Serie A, but these companies still rely on human interventions for the provision of game state information. In particular, for ball possession the soccer industry uses a definition which considers the amount of time that a team spends controlling the ball and, to extract this information, has long relied on a human operator watching the game armed with a three-button timer. The buttons are used to record the beginning of a new game phase, which can be either the home team having possession, the away team having possession, or a stoppage (because the ball is outside the pitch, or the referee has interrupted the game, e.g., after a foul).

The reliance on a human operator is motivated by the fact that there are a number of situations where it can be extremely difficult to define clearly which team owns the ball (Bialik, 2014). The need to rely on human operators watching and annotating each game, however, clearly has economic and logistic impacts, which are detrimental to the implementation of an automatic data analytics pipeline. Furthermore, it has also been observed that these annotations are prone to errors, which negatively impact the subsequent data analysis steps (Richly et al., 2017). Another important reason for making the extraction of ball possession automatic is that it would allow to add this information to tracking datasets regarding past matches where it was originally missing, making them accessible for further analyses that otherwise would be extremely time consuming.

Scientific literature regarding the automatic estimation of game state is not particularly rich (especially from tracking data), mainly due to the scarcity of public datasets. The few approaches proposed so far generally relied on a different definition of ball possession based on the number of passes completed by a team (Glasser, 2014; Sarkar et al., 2019) that is not the commonly adopted one, or leveraged handcrafted rules (Link and Hoernig, 2017; Morra et al., 2020; Khaustov and Mozgovoy, 2020) that can hardly capture the discrimination abilities of human operators.

Hence, in the present paper, we propose to:

- use deep learning to make a computer learn how to automatically estimate the state of a soccer game starting from spatio-temporal data about players and ball positions, without resorting to rules defined based on domain knowledge about soccer;
- output this information in the same format of the standard, three-button timer mechanism.

In particular, we investigate the use of Temporal Convolutional Networks (TCNs), which in recent years proved to be particularly effective for dealing with classification tasks on sequential data. In this work, TCNs are used to create independent trajectory embeddings from tracking data. We then experimented with three architectures that combine the embeddings in different ways and compared the obtained results with those achieved by state-of-the-art methods leveraging predefined rules.

The remaining of the paper is organized as follows. Section 2 reviews relevant literature pertaining sport analytics. Section 3 introduces the proposed method, whereas Section 4 presents the protocol that has been set up to evaluate it. Section 5 reports on experimental results. Finally, Section 7 provides conclusions and suggests possible directions for future research in this field.

## 2. Related work

Over the years, sport analytics and, particularly, event detection in soccer games have been addressed in different ways. The existing literature can be roughly classified on the basis of the type of input used, which can be represented by either visual or tracking data (or a combination of them).

### *2.1. Visual data*

Videos indeed represent the most common source of input in the context of sport analytics. Unsurprisingly, in the last several years, most of the research explored the use of deep learning models. Deep learning has been proved successful for many purposes, from players tracking (Kukleva et al., 2019), (Xu et al., 2018), to video summarization (Rockson et al., 2019), (Gao et al., 2020) and the generation of high-level game statistics (Fernández et al., 2019), (Memmert and Rein, 2018), (Theagarajan et al., 2018).

Among the possible applications, two tasks that are particularly relevant for the goal of this paper are action recognition and event recognition. For instance, Hong et al. (2018) focused on the possibility to use transfer learning with state-of-the-art Convolutional Neural Network (CNN) models to detect events like corner, free-kick, penalty and goal plus different types of camera shots from soccer videos. Other authors, like Xu and Tasaka (2020), focused on improving the accuracy and speeding up the identification of particular events in 4K multi-view videos of soccer games extending well-known CNN-based object detection and pose estimation methods (such as YOLO (Farhadi, 2016) and OpenPose (Cao et al., 2021)).

The most common approach to address the above task on video data is known as Convolutional Recurrent Neural Network (RNN): this approach extends the architecture used in previously cited works since, first, features are extracted from each frame in the video using a CNN, then they are passed to a RNN which produces the output.

An example of this setting can be found in Sorano et al. (2020), which aims at producing a graph of passes that occur in a soccer game. In the proposed architecture, video frames are processed both by a convolutional object detection network (YOLO, in this case) and by a feature extraction network (ResNet18 (He et al., 2016)). For each frame, the feature extraction module produces a vector describing the whole scene; the object detector, in turn, is responsible for detecting the players as well as the ball, and returns a vector describing the position of the ball and the players closest to it. The two vectors are then concatenated. By processing all the frames in the video, it is possible to produce a sequence of feature vectors that are fed into the sequence classification module, which consists of a bidirectional LSTM (Long Short-Term Memory), a model commonly used in this context. This module outputs a pass vector that indicates, for each frame of the original sequence, whether it is part of a pass sequence or not. Another work exploiting this methodology to detect a larger set of events is represented by Jiang et al. (2016). Here, play-break segments are first obtained. Then, semantic features are extracted from them using a CNN. Finally, four event types are classified (namely, corner, goal, goal attempt and card) using a RNN.

The above approach is also adopted in Roy Tora et al. (2017). In this case, the focus is on ice hockey, but the task is closer to that addressed in the present work, as the authors' goal is to recognize puck possession events. Like in the above works, the frames are processed in parallel by two CNNs which extract frame-related features and, based on the output of an object detector, individual player-related features. The features are then concatenated and passed to an LSTM, which processes them sequentially and produces the output.

The main drawback of the methods reviewed so far lays in the fact that they rely on recurrent architectures, which have been proven to be characterized by a performance bottleneck due to the use of sequential computations. A way to cope with the above limitation when dealing with sequential data is represented by TCNs. These networks rely on convolutional layers, whose operations can be easily parallelized, thus benefiting of continuous advancement in computing technology. Bai et al. (2018) and Guirguis et al. (2021), who focused on comparing recurrent architectures with their convolutional counterparts on a variety of tasks, showed the largely higher performance of the latter models in terms of accuracy, as well as of training and inference time.

In the context of sport analytics, TCNs have been largely applied to action recognition (the domain they actually stemmed from). An example is provided by Martin et al. (2018), where a Siamese spatio-temporal CNN is used to simultaneously process color images and optical flow data associated with table tennis games to this purpose.

These models have also been widely used for event detection, which can be considered as a particular case of action recognition. Some examples in this field are represented, e.g., by Liu et al. (2017), Lee et al. (2018), Yu et al. (2019) and Khan et al. (2018b).

The approach of Khan et al. (2018b) is particularly interesting since it uses C3D (Tran et al., 2015), which is basically the three-dimensional (spatio-temporal) counterpart of the well-known two-dimensional VGG network (Simonyan and Zisserman, 2014) and leverages many of the state-of-the-art characteristics for image classification (such as a high number of layers and small kernels); moreover, the authors showed for the first time how to use such an architecture not only for classifying a video, but also for creating effective descriptors of it, which could be used in a transfer learning pipeline for further analyses.

It is worth observing that the problems addressed by the above works are different than that tackled by the present paper. In fact, as stated in Section 1, the task of estimating the game state has not been dealt with in depth by the research community yet.

Besides some research done in the field of game state description (addressed in the context of summarization), one of the few works that considered ball possession is represented by Khan et al. (2018a). The authors do not use deep learning end-to-end, since they propose a framework in which the frames of soccer videos are first processed by a Single Shot MultiBox Detector (SSD)-based object detection module (Liu et al., 2016). The output is then passed to a rule-based system that uses temporal and logical operators, which starts by detecting the so-called “simple” events and assembles them to recognize the “complex” events.

### *2.2. Tracking data*

As shown by the last work reviewed in the previous section, an alternative way to deal with the problem of interest for this paper and, in general, with sport analytics tasks consists of exploiting tracking data. With the improvements in tracking technology, sport researchers are using them for ever more complex tasks, from event detection, to statistics generation, tactic effectiveness quantification, etc.

As for video data, the most recent works in this field rely on deep learning, and leverage spatio-temporal convolutions directly on raw data to create low-dimensional representations that summarize the motion of objects of interest in space over time periods. In the literature, these representations are referred to as trajectory embeddings.

An important milestone in the use of trajectory embeddings in sport analytics has been set by the work reported in Horton (2020). The author's goal is to learn an internal feature representation of the movements of all players in a soccer game. To this purpose, a network is designed that takes as input raw trajectory data and learns an internal representation of the individual and coordinated movements of all players. The trajectory of a single player is represented as a sequence of time-stamped frames, and each frame is a vector containing the $x$ and $y$ coordinates for the player at that time, possibly with additional information such as his or her orientation and speed. The main contribution of this work stems from the consideration that most machine learning methods require a predefined structure in the input format that also comprehends an ordering within each input element, such as in the case of an image. However, in the case of tracking data, it is often impossible to define a predefined shape of the input due to the naturally variable duration of a game play, and there is no intrinsic ordering of players in a given interval of play that persists throughout the game or from game to game (in some sports number of player can even change, e.g., due to red cards). Previously, both the variable duration problem and player-ordering problem had been circumvented by introducing a preprocessing step in which raw tracking data are transformed into structured feature representations designed ad hoc for the task at hand. The method adopted by Horton (2020) avoids the limitations typically associated with feature engineering, and addresses the first problem by means of 1D convolutions and adaptive pooling mechanisms; it then deals with the second problem by using a set-based architecture (Zaheer et al., 2017) that treats the input as an unordered set, devising a model that learns the feature representation directly from raw data. The authors used the proposed method to create two models for making predictions about passes (probability of completion, length, and reception location) and tackles (probability for a player to be the first to attempt it, distance covered, and location).

Other ways that have been explored to achieve set-based learning in sport analytics consist in leveraging roles rather than identities for players (e.g., when the task is to study the behavior of an entire adversarial team, like in Lucey et al. (2013)), or in identifying an object, like a player or a ball, that can be used as "anchor" and define an ordering relative to it. An example of this latter approach is given in Mehrasa et al. (2018). Like in Horton (2020), trajectory embeddings are created by 1D convolutions. Then, a permutation-invariant sorting scheme is defined based on the distance of a candidate object (a player, in this case) to the anchor, with the trajectory of the anchor being placed always in the first position, the closest object next to it, and the farthest object appended to the end. The authors applied this technique to two different tasks, i.e., event recognition in ice hockey (with six events considered, and the player carrying the puck acting as the anchor), and team classification in basketball (with the ball selected as the anchor). It is worth observing that the devised approach based on trajectory data was found to outperform the C3D model that uses video as input, and to be capable of achieving even better performance when used in combination with video.

A work that is particularly interesting considering the focus of the present paper is represented by Sanford et al. (2020). The authors address the detection of atomic actions in a soccer game (pass, shot, and reception), and focus on analyzing the performance of vision-based and trajectory-based models. The authors considered four vision-based models. All of them rely on an inflated 3D CNN (I3D) (Carreira and Zisserman, 2017). In the first model, the 3D convolution is applied to the whole image frame. In the other cases, it is applied to players' "tubelets", i.e., sequences of bounding boxes containing a single player; the features extracted from the tubelets are then processed in three different ways: via max-aggregation, a Graph Convolutional Network (GCN), and a transformer. The best performance was achieved by the model that processes the whole frame, without using the tubelets. For trajectory-based detection, they considered three models, namely a TCN (named Wavenet (van den Oord et al., 2016)), a transformer, and a TCN followed by a transformer. In all three cases, these blocks were followed by a fully connected layer to predict the actual player's activity. Experiments were run both using only the ball trajectory and using the ball along with the K-nearest players: the model containing both the TCN and the transformer and using the (five) K-nearest players proved be the one providing the best performance on all three atomic activities. When comparing the two approaches, vision- and trajectory-based models were found to provide, on average, comparable results. The findings of the latter work are particularly interesting since they confirm the effectiveness of the trajectory-based method for dealing with sports analytics tasks. They also pinpoint TCNs as the best candidate to address the detection of game events.

Works reported above are all relevant to the present paper, since they provide concrete architectures that can be used to perform classification and action detection tasks on tracking data. Notwithstanding, their goal still differs from that considered here, since they cannot be directly used for ball possession estimation (although this limitation mainly concerns the last layers of the network, which are described by the authors themselves as task-specific).

Like for visual data, the amount of works that focused on ball possession by leveraging tracking data is still quite limited. An example is represented by Link and Hoernig (2017). This work uses a rule-based system to segment the game into possession phases, which are further subdivided into actions and void phases (e.g., when the ball is in the air). A possession phase begins when a new player starts to interact with the ball. Interactions are detected looking at the spikes in the ball acceleration; when a local maximum greater than $4\text{ms}^{-2}$ is found, the possession is assigned to the player closest to the ball. The idea of looking at the derivatives of the ball position comes from the fact that tracking data often do not include the $z$ coordinate; therefore, it is necessary to prevent accidental changes in possession during phases in which the ball crosses intermediate players.

A similar idea is exploited in Morra et al. (2020). Here, the temporal and logical operators that were originally used in Khan et al. (2018a) on the output of a visual data processing stage are extended and applied to spatio-temporal data obtained through a soccer game simulator for the detection of game events, including player's ball possession. In this case, the ball speed is considered, based on the consideration that while a player controls the ball, the latter should move relatively slowly. Furthermore, for a possession to be valid, the distance between the player and the ball should be low for a certain amount of time, and the distance between the opponents and the ball should be above a given threshold.

A more recent example of a rule-based system that predicts ball possession from spatio-temporal data is given in Khaustov and Mozgovoy (2020). As in the previous cases, when a possession change occurs, the ball is assigned to the closest player. Possession changes are detected in three ways: through changes in ball speed, changes in ball direction, and prolonged proximity to the ball.

In the present work, we address the problem of estimating ball possession from tracking data following a different approach. First, as in the works focusing on action detection reported at the beginning of this section, our aim is to remove the need to rely on handcrafted rules. The objective is to devise models capable to learn directly from data, without resorting to domain knowledge about soccer, which humans use to explain the (possible ambiguous) concept of possession. Second, our expected outcome also slightly deviates from that of the latter works reviewed above. In fact, they actually addressed the ball possession problem in a more fine-grained way, as they estimate the possession on an individual level, telling which player, not only which team, owns the ball. Indeed, this fact is directly connected with the nature of rule-based systems, which follow a bottom-up approach that allows to extract semantic knowledge from the intermediate results. However, we intentionally faced the problem from the point of view of the team rather than of the player, since, as said, the actual mechanism to collect ball owner information is based on the three-button timer used by a manual operator. Data collected through this mechanism only describe the state of the game (home team controlling the ball, away team controlling the ball, game stopped), without providing information about the single player who is owning the ball. Thereby, it is convenient to start with a less fine-grained approach, and only afterwards add information about the single player on top of the data obtained for the team.

## 3. Proposed method

This section illustrates the main principles that underlie our methodology. First, the general formulation of the problem statement is presented in Section 3.1. The three architectures evaluated in this study are then introduced in Section 3.2. Finally, the loss function, TCN design, and aggregation functions are discussed in detail in Sections 3.3, 3.4, and 3.5, respectively.

### *3.1. Problem statement*

As anticipated in Section 1, our goal is to propose a network architecture able to classify the game state for a given time window by leveraging tracking information.

Let us define the proposed network as a function $\mathcal{H} : \mathbf{x} \in \mathbb{R}^{n_f \times n_o \times n_c} \mapsto y \in \mathcal{Y}$, where $\mathbf{x}$ is the input tensor and $\mathcal{Y} = \{\textsc{dead}, \textsc{home}, \textsc{away}\}$ is the three-class output. The classifier is trained in a supervised fashion from a labeled dataset $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^{N}$.

We assume the input tensor to be of size $n_f \times n_o \times n_c$, where $n_f$ is the number of frames in the observed time window, $n_o$ is the number of tracked objects (e.g., players, ball, referee, etc.) and $n_c$ is the number of feature channels associated to each object. For simplicity, and without loss of generality, we assume that the feature channels include at least the position of the tracked objects with respect to the pitch and the team; however, as exemplified later in Section 4.1, the feature vector can be extended to include other features such as the player id, the object velocity, visual features, etc.

To simplify the explanation of the learning procedure, we decompose the network $\mathcal{H}$ as a combination of three functions

$$\mathcal{H}(\mathbf{x}) = f_c\big(\Lambda\big(f_{tcn}(\mathbf{x})\big)\big) \tag{1}$$

each implemented by one or more layers.

The *embedding* function $f_{tcn} : \mathbb{R}^{n_f \times n_o \times n_c} \to \mathbb{R}^{n_o \times l_{traj}}$ maps the trajectories of each individual object to an embedding vector of length $l_{traj}$. As detailed later, this component is based on TCNs, hence the subscript.

The *aggregation* function $\Lambda$ combines the embeddings associated with different objects into a single feature vector, which is then given as input to the actual classifier $f_c$ (last function). The order of the players within the input data is based solely on the jersey number of each player within the team; hence, this order does not carry any semantic meaning and needs therefore to be abstracted. It is crucial that, given the same position of the objects on the pitch, the network predicts the same result if two players are swapped, i.e., that the output does not vary in case of a permutation in the input data: hence the need to define a *permutation-invariant* function. An alternative strategy, detailed in Section 3.5.3, is to order the objects according to a predefined criterion, which bypasses the need to use a permutation-invariant function.

Finally, the last *classification* function $f_c$ is a simple feed-forward network (FFN) that computes the output class $c$. Alternatively, it is possible to redefine the output space as a combination of two binary labels $(y_{DA}, y_{POSS})$, where $y_{DA} \in \{\text{DEAD}, \text{ALIVE}\}$ and $y_{POSS} \in \{\text{HOME}, \text{AWAY}\}$. The peculiarity of this formulation, as discussed in Section 3.3, is that $y_{POSS}$ is not defined when $y_{DA} = \text{DEAD}$.

### *3.2. High-level architecture*

This section explores in detail several variants for each of these three components and their combinations, and introduces the three high-level architectures that were experimentally compared in this work.

All architectures exploit TCNs as the *embedding* function. As discussed in Section 2, according to the recent deep learning literature, TCNs applied to tracking data have proven to work well in different tasks, such as event detection, team classification, etc. They also compared favorably with respect to recurrent architectures (reported, e.g., in Bai et al. (2018) and Guirguis et al. (2021)).

In particular, the proposed architectures are based on $k \times 1$ convolutional filters in order to produce separate trajectory embeddings for each object on the field (in our case, as said, the players, the ball, and the referee).

The first proposed architecture, denoted in the following as the *single-branch* model, frames the problem as a ternary classification. The network, depicted in Fig. 1a consists of three blocks that implement the functions introduced in Section 3.1. In this formulation, $f_c(\cdot)$ is a FFN with three output classes, which takes as input the trajectory embeddings obtained through spatio-temporal convolution as detailed in the problem statement. It is important to stress that the aggregation function $\Lambda$ must be invariant to permutation; different architectural choices that satisfy this property are illustrated in Section 3.5.

The second class of architectures requires producing two binary classifications: one telling if the game state is active, the other one telling which team owns the ball in an active game phase. This leads to an architecture with two parallel output layers, each responsible for one classification. At the network level, it is possible to achieve these goals in two ways, outlined respectively in Fig. 1b and Fig. 1c.

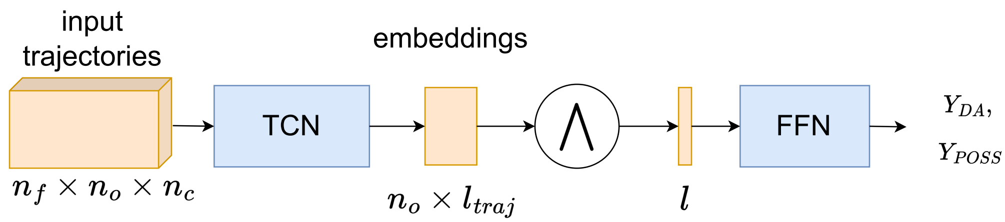
The first variant, illustrated in Fig. 1b and denoted in the following as the *two-branch* network, computes the trajectory embeddings once and uses them to predict both output variables. The TCN output is passed to two different $\Lambda$ layers, which in turn pass their output to two separate FFNs, the Dead-Alive (DA) branch, and the Possession (POSS) branch. Each FFN produces a scalar output, representing respectively $P(Y_{DA} = \textsc{dead} \mid X)$ and $P(Y_{POSS} = \textsc{home} \mid X, Y_{DA} = \textsc{alive})$. Alternatively, it is also possible to share both the TCN and the $\Lambda$ layers, splitting only the FFN network or part of it. The choice clearly represents a trade-off between computational needs and flexibility; here, we preferred to keep the $\Lambda$ layers separated, since we expect that having distinct representations may be useful to optimize each classification. It is important to note that both branches are trained in parallel, i.e., a single backpropagation is performed, and hence the TCN is trained to jointly optimize both tasks. Parallel training can be achieved using a combined loss function (discussed in Section 3.3) that produces a single scalar value resulting from both branches.

Alternatively, it is possible to perform the classifications by two separate networks, a Dead-Alive (DA) network and a Possession (POSS) network, as shown in Fig. 1c. This variant will be denoted in the following as the *two-networks* configuration. In this case, each network computes its trajectory embeddings that are then passed to the $\Lambda$ layers and finally to the FFNs for the binary classification. Computing separate embeddings allows the TCNs to capture those aspects of the tracking data that may be more relevant for the specific task, rather than producing a set of general-purpose feature vectors that are expected to solve both tasks at the same time. The two networks are trained separately end-to-end, with the possibility of adapting them to specific task needs, which include using different sets of hyperparameters. The drawback of this alternative is that training two networks requires roughly twice as much computational resources; this choice is viable only if it brings about a boost in performance that justifies such investment. Furthermore, the use of separate trajectory embeddings goes against the concept of embedding as a general descriptor that effectively summarizes the data and can be used in a wide range of applications, as described in Khan et al. (2018b).

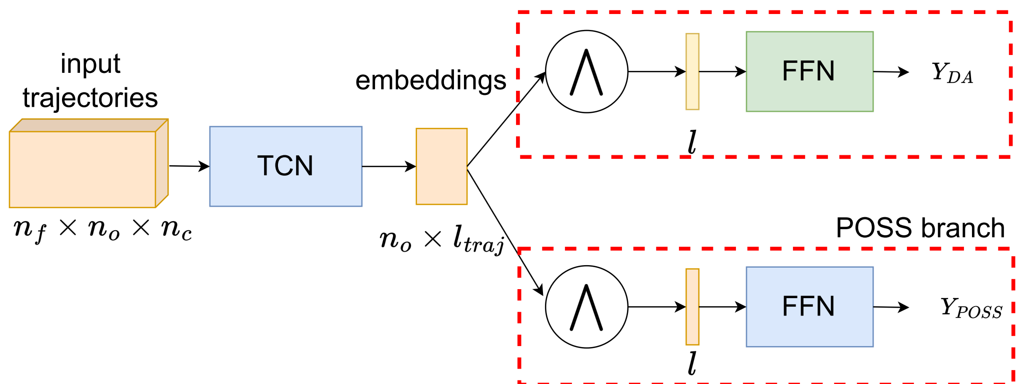
### *3.3. Loss functions*

The single-branch, multi-class model is trained using a standard cross-entropy loss written as

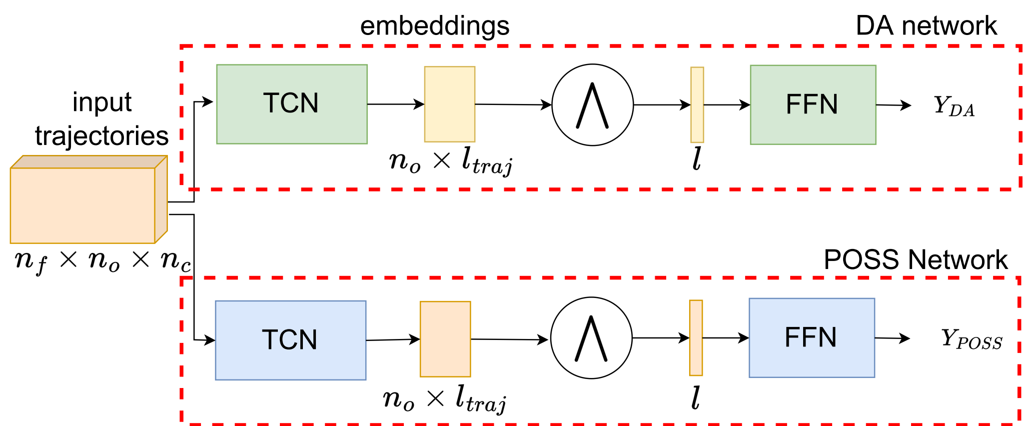
$$L(\hat{y}) = -\sum_i y_i \cdot log(\hat{y}_i) \tag{2}$$




(a) Single-branch




(b) Two-branches




(c) Two-networks

Figure 1 *(previous page)*: Comparison of the three proposed architectures. All take as input a multi-dimensional array of size $n_f \times n_o \times n_c$, where $n_f$ is the number of frames, $n_o$ is the number of objects (including all players, the ball and the referees), and $n_c$ is the number of channels (i.e., features) associated with each object (including, e.g., the position, velocity, team, etc.). All architectures output two scalars representing the probabilities $P(Y_{DA} = \textsc{dead} \mid X)$ and $P(Y_{POSS} = \textsc{home} \mid X, Y_{DA} = \textsc{alive})$. Each architecture is composed by one or more TCN computing the embeddings (one of each object), a permutation-invariant aggregation function $\Lambda$ that combines the trajectories of all objects, and finally one or more FFNs $f_c$ that computes the output probabilities. While the single-branch architecture computes both output probabilities using a single TCN and FFN (a), in the two-branch architecture two separate FFN layers are defined on top of a single shared embedding TCN (b). In the two-networks architecture, DA state and ball possession are estimated using separate embedding and classification functions (c).

Using a one-hot encoding, $y_i$ is zero for all classes but the correct one; hence, the cross-entropy loss turns out to be $-log(\hat{y}_K)$, whereby $K$ is the true class.

For the two-networks model, the DA network does not differ substantially from the previous one, except that it performs a binary classification: however, this can be considered as a special case of multiclass classification, which allows to use a slightly different cross-entropy function that accounts for the fact that the network outputs a scalar value instead of a vector. Since the network predicts the conditional probability of $Y_{DA} = \textsc{dead}$, the true label $y_{(DA)}$ should be a scalar with value 1 if the game state is DEAD and 0 otherwise. With these modifications, the binary loss function of the DA network can be expressed as:

$$L_{DA}(\hat{y}_{(DA)}) = -(y_{(DA)} \cdot log(\hat{y}_{(DA)}) + (1 - y_{(DA)}) \cdot log(1 - \hat{y}_{(DA)})) \qquad (3)$$

The issue is more complex when considering the POSS network. The classification here does not only depend on the input data $X$, but also on the value of $Y_{DA}$: $Y_{POSS}$ is meaningful only as long as the game is active, otherwise it is useless to estimate which team owns the ball. During training, this means that the network should not update its parameters if it is faced with a sample where the true label is DEAD. To obtain this result, it is possible to define the loss function as follows:

$$L_{POSS}(\hat{y}_{(POSS)}) = \begin{cases} BCE(\hat{y}_{(POSS)}), & y_{(DA)} = 0 \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

where $BCE$ is the binary cross entropy:

$$\begin{aligned} BCE(\hat{y}_{(POSS)}) = -(y_{(POSS)} \cdot log(\hat{y}_{(POSS)}) + \\ + (1 - y_{(POSS)}) \cdot log(1 - \hat{y}_{(POSS)})) \end{aligned} \tag{5}$$

Finally, the two-branch model is trained using a multi-tasking loss function defined as

$$L(\hat{y}_{(DA)}, \hat{y}_{(POSS)}) = \alpha \cdot L_{DA}(\hat{y}_{(DA)}) + (1 - \alpha) \cdot L_{POSS}(\hat{y}_{(POSS)}) \tag{6}$$

i.e., as an average of the two loss functions described above with an additional weight parameter $\alpha$. During backpropagation, the derivative of $L$ with respect to an arbitrary parameter $x$ is given by the formula

$$\nabla_x L = \alpha \cdot \nabla_x L_{DA} + (1 - \alpha) \cdot \nabla_x L_{POSS} \tag{7}$$

For the parameters located in the branches, this means that the update process is the same as in the two-networks model: parameters lying in one branch do not impact the loss function of the other branch; thus, one of the two members of the derivative above will be zero. With respect to the parameters in the TCN, the update will depend on both losses, according to the weight factor $\alpha$. In particular, it can be noticed that if a sample belongs to a segment of inactive game, the function $L_{POSS}$ and its derivatives will be zero, which means that the parameters in the TCN are updated based only on the output of the DA branch.

### *3.4. TCN design*

In the proposed architectures, the TCN is responsible for producing trajectory embeddings, i.e., fixed-size representations of the movements on the pitch of every relevant object. This is achieved by stacking several layers of temporal convolutions, which gradually incorporate information from different points in time into a single vector. The structure of the layers defines *a priori* the size of the receptive field, i.e., the number of elements in the sequence that concur to the final prediction. As a result, the receptive field determines how many frames are needed to form an input sample, a parameter that has already been introduced as $n_f$. In this choice, there should be a trade-off (which has to be made at design time) between two factors: on the one hand, larger sequences allow to consider a larger portion of the game when producing an output; on the other hand, they require a deeper network, which in turn needs more time and more data to be trained.

It is important to note that the target frame, i.e., the frame for which we want to predict the game state, can be located anywhere within the sample: in case the input sequence only includes past frames, the convolution is said to be *causal*, otherwise it is called *acausal*. The choice between these alternatives depends on how fast the ball possession prediction has to be made; however, it is important to consider that seeing how the action goes on after the target frame can help to enhance the model performance. For example, using only the tracking data, it is difficult to recognize immediately whether a foul was called: in this case, it can be helpful to consider also some frames afterwards, based on the consideration that if a foul is called, the players will probably stop running or move towards the referee. For this reason, unless the system has strict time constraints, it seems appropriate to opt for acausal convolutions.

With respect to these two concepts, it can be useful to point out two aspects pertaining TCNs. First, it is clear that the input slices in two adjacent forward passes have almost the same elements; yet, since they are in different positions, it is not possible to reuse the results of the convolutions from one pass to the other. Second, as shown in Fig. 2, the temporal convolutions are computed on all elements in the sequence, applying dilations and padding when necessary. However, only a small part of the computations (which are shown in the figure with continuous lines) effectively contribute to the output results (the orange circle in the top right). The other computations are needless, since their results are gradually discarded by the following layers.

In order to design the internal structure of the TCN, it is important to recall from the problem statement that it should apply a function $f_{tcn}$ to the input data, such that

$$f_{tcn} : \mathbb{R}^{n_f \times n_o \times n_c} \rightarrow \mathbb{R}^{n_o \times l_{traj}} \tag{8}$$

However, since the trajectory embeddings should be computed separately for each object, $f_{tcn}$ is equivalent to applying on $n_o$ inputs a function $f^{'}_{tcn}$,

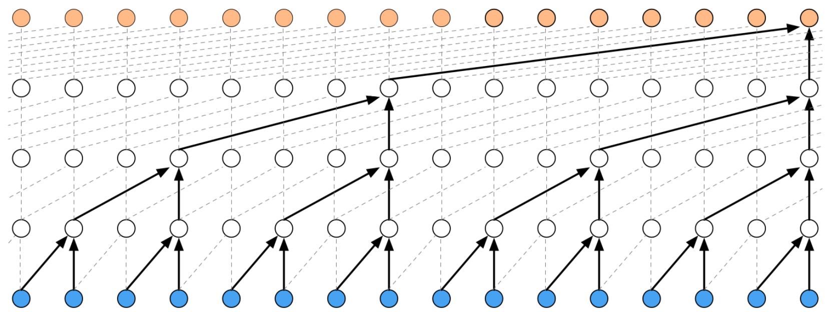


Figure 2: Scheme of dilated convolution (van den Oord et al., 2016): black lines show the convolutions that actually contribute to the result.

such that

$$f'_{tcn} : \mathbb{R}^{n_f \times n_c} \rightarrow \mathbb{R}^{l_{traj}} \tag{9}$$

A function with these characteristics can be achieved using 1D convolutions, i.e., convolutions with a filter of size $k$ and not $k_1 \times k_2$, as in the more common 2D convolutions. At implementation time, it should be considered that filters always have one additional dimension, since they are applied over many channels at the same time; however, this aspect is usually disregarded in the definitions, which explains why they are referred to as 1D convolutions even though the input is two-dimensional. In order to apply $f'_{tcn}$ in parallel on all objects, the most straightforward way is to arrange the operation as a 2D convolution with a $k \times 1$ filter on the whole input, which has size $n_f \times n_o \times n_c$. This technique, proposed by Horton (2020), allows at each step the filter to be convolved with a portion of the input tensor, containing $k$ frames related to only one object. The result of the 2D convolutional layer is a matrix of size $n_o \times l_{traj}$, whose columns correspond to the output of the 1D convolution applied to the respective object. In other words, by means of a $k \times 1$ filter it is possible to compute the function $f_{tcn}$ on the whole input tensor in a single pass.

The final structure of the TCN is given in Table 1. The first layer is a $1\times1$ convolution, in order to adapt the third dimension of the input to the size of the final embeddings, which is $l_{traj}$. Next, a batch normalization layer is applied, as proposed in Ioffe and Szegedy (2015). After that, the architecture features a block containing three layers: the first one is a convolutional layer with a $k \times 1$ filter, which constitutes the most relevant part of the function $f_{tcn}$. Then, there is a dropout layer and another $1\times1$ convolution. The block is repeated multiple times (the exact number $n\_blocks$ is a hyperparameter of the network) with an exponentially growing dilation rate: as said at the beginning of this section, the number of blocks in the network determines the receptive field of the TCN and, hence, the length of the subsequence considered at each forward pass. Finally, after having applied dropout and batch normalization once again, the last sequence element is selected since, as said, this element captures the whole receptive field, thus offering a summarized representation of the whole temporal sequence.

### *3.5. Permutation invariance*

In the architectures presented in Section 3.2, a major role is played by the aggregation layer $\Lambda$, which transforms the individual trajectory embeddings into a global representation of the game sequence, which in turn can then be classified by an FFN. It has already been pointed out that $\Lambda$ should be permutation-invariant, i.e., the result should be independent of the players' order in the input tensor. In this section, three possible ways to achieve this goal are analyzed, with different characteristics and complexities.

| **Layer type** | | **Output size** | **Parameters** |
|---|---|---|---|
| input | | $n_f \times n_o \times n_c$ | - |
| conv | | $n_f \times n_o \times l_{traj}$ | filter $1 \times 1$ |
| batch_norm | | '' | - |
| conv | $\left.\right\}$ | '' | filter $k \times 1$ |
| dropout | $\left.\right\} \times n\_blocks$ | '' | - |
| conv | $\left.\right\}$ | '' | filter $1 \times 1$ |
| dropout | | '' | - |
| batch_norm | | '' | - |
| slice | | $n_o \times l_{traj}$ | - |

Table 1: Architecture of the TCN module.

#### *3.5.1. Reduce by sum*

Considering an input matrix $A$, a simple invariant operation with respect to column permutation is the multiplication $A \cdot \mathbf{1}$, where $\mathbf{1}$ is a vector of all ones. This operation is equivalent to computing a vector whose $i$-th value is the sum of all the values in the $i$-the row of $A$. It is evident that if two columns in the input matrix are swapped, the sum of the values across each row remains unchanged; therefore, the function

$$f_{reduce\_sum} : \mathbb{R}^{m \times n} \to \mathbb{R}^m, A \mapsto A \cdot \mathbf{1} \tag{10}$$

is permutation-invariant. In the current case, the TCN outputs a tensor $A_{tcn}$ of size $n_o \times l_{traj}$. Since the position of the ball and the referee is already fixed by the fact that their tracking data are placed in the first two columns of the dataset, in order to make the $\Lambda$ layer permutation invariant, it is sufficient to consider the two submatrices $A_{tcn}^{(home)}$ and $A_{tcn}^{(away)}$ containing the embeddings of the home and away players. The submatrices have size $(11+6) \times l_{traj}$, since each soccer team has 11 starting players and up to six substitutions [1], and can be passed to $f_{reduce\_sum}$ obtaining two vectors $v_{home}$ and $v_{away}$ of size $l_{traj}$ that act as trajectory embeddings of one team each.

It is therefore possible to construct a permutation-invariant $\Lambda$ layer through the linear transformation

$$f_\Lambda : \mathbb{R}^{n_o \times l_{traj}} \to \mathbb{R}^{4 \times l_{traj}}, A_{tcn} \mapsto \begin{pmatrix} | & | & | & | \\ v_{ball} & v_{ref} & v_{home} & v_{away} \\ | & | & | & | \end{pmatrix} \tag{11}$$

where $v_{ball}$ and $v_{ref}$ are the trajectory embeddings of the ball and the referee. It is possible to use this result as input to a FFN by flattening the matrix into a one-dimensional vector of size $n_o \cdot l_{traj}$, as is commonly done in CNNs designed for image classification.

### *3.5.2. Self-attention*

A second possibility in the construction of $\Lambda$ is to take advantage of recent advances in the field of attention models. In particular, the self-attention module introduced by Vaswani et al. (2017) allows to create embeddings of the original elements that not only take into consideration other elements in the tensor, but are linear combinations of those elements (or more precisely, of their *values*). Notably, in the original self-attention model, the tensor consists of different elements of a sequence, but at this point there are no

[1] One additional player is encoded as an extra substitution to account for possible tracking errors

sequential data to work with since the temporal information is flattened by the TCN into the trajectory embeddings. Thus, in this case, the self-attention model is not used to process different elements within a temporal sequence, but rather elements that are part of an unordered set, such as the submatrices $A_{tcn}^{(home)}$ and $A_{tcn}^{(away)}$ introduced above.

Since the biggest part of the possession estimation is related to one single object, namely the ball, it seems reasonable to think that if the self-attention module is able to grasp all the interactions where the ball is involved, it is possible to make a reliable prediction without considering the interactions among the other objects. At the same time, since self-attention is specifically designed to output a weighted representation of the interactions between the input columns, extracting such a representation of the ball trajectory should provide enough information for a successful classification. Based on these considerations, if the ball trajectory has to be enriched by all the other objects, it is evident that the self-attention layer should receive the whole tensor produced by the TCN: it will be then the self-attention task to recognize which objects have a role in determining the possession and which objects are irrelevant, such as bench players. In this sense, it is relevant to note that the value of a given column in the output of a self-attention layer is independent of the ordering of the other columns. This means that the computation of the column related to the ball is permutation-invariant with respect to the columns related to the players.

If we denote by $S \in \mathbb{R}^{n_o \times n_{att}}$ the matrix produced by the self-attention layer and by $s_i$ its column vectors, the operation of the $\Lambda$ layer can be described by the function

$$f_\Lambda : \mathbb{R}^{n_o \times l_{traj}} \to \mathbb{R}^{n_{att}}, A_{tcn} \mapsto s_1 \tag{12}$$

where $n_{att}$ indicates the size of the *query*, *key* and *value* vectors as defined in Vaswani et al. (2017).

*3.5.3. 2D Convolutions*

A way that is often used to achieve permutation invariance is to impose an ordering based on an anchor object. In this case, the game state is estimated based predominantly on the ball: it is possible therefore to order the players according to their distance to this object, so that the network can operate on the data independently of how the players are arranged in the input tensor. Although this idea can also be applied to the options presented above, e.g., by limiting the reduce or the self-attention operations to the players close to the ball, its most powerful consequence is that it allows one to structure the input tensor in a way that avoids the need to create isolated embeddings for the objects.

The input data can be arranged across two dimensions, which should be flattened in order to obtain a single prediction of the game state. The two dimensions are the temporal axis and the different objects, and their sizes are $n_f$ and $n_o$, respectively. Structuring the input data allows to operate on them as commonly done for video streams, where the temporal and the spatial dimensions are processed in parallel. In other terms, the input data can be interpreted as two-dimensional, rather than one-dimensional.

Taking this fact into account, it is possible to apply a temporal convolution to the input by using a 2D convolutional layer. However, while in

Section 3.4 2D convolution was performed by means of a $k \times 1$ filter to separate each object, here $k_1 \times k_2$ filters are used in order to fuse together the information along both axes. This third proposal to obtain a permutation-invariant representation of the global features therefore does not foresee any $\Lambda$ layer: instead, permutation invariance is a by-product of the TCN design, after introducing the additional constraint that players in the input data are pre-ordered according to their distance from the ball.

## 4. Experiments

### *4.1. Dataset*

The dataset at our disposal consists of tracking data taken from 35 games during the 2019-20 season of a top professional European league. The data are collected at an average rate of 16 frames per second (fps) and for each frame the following information is provided:

- a *frame number*, an incremental id of the frame starting from 1;
- a *game state* label, as described in the problem statement: this is the target variable of the system;
- a *timestamp*, indicating at which moment the data was collected;
- a *half* flag, indicating whether the frame was collected in the first or in the second half of the game;
- the $x$ and $y$ coordinates of the ball;
- the $x$ and $y$ coordinates of the referee;

- for each player, the $x$ and $y$ coordinates, and a flag to distinguish goalkeepers.

Ball and players coordinates are provided from a third part company specialized in real time tracking technologies for the sport sector, through a system of ad-hoc cameras installed directly in each venue. The coordinates spaces is a rectangle corresponding to the football pitch and the coordinates system is centered in the center of the pitch (kickoff point) with x-positive axis pointing to the right and y-positive axis pointing up. Considering the standard dimensions of a football pitch (105 $\times$ 68 meters), the range is [–52.5, 52.5] for x–axis and [–34,34] for y–axis both for ball and players. If the tracking system could not locate an object or if the object was not on the pitch (e.g., in the case of a player sitting on the bench or being expelled), the corresponding $x$ and $y$ fields are empty.

Target labels DEAD, HOME and AWAY were manually assigned in real time by a human operator as part of a series of services provided by a third company to the league organization. The target labels in the dataset are distributed as follows: about 40% of the samples belong to the class DEAD, the rest of the samples are quite evenly distributed between the classes HOME (29.6%) and AWAY (30.3%).

Since the model takes as input a tensor of size $n_f \times n_o \times n_c$, the whole game is split in sequences of length $n_f$. Clearly, it is also possible for samples to overlap with each other, as the game sequence is transformed into samples following a sliding window approach with a stride of 1 (i.e., adjacent samples differ only by one frame).

Datasets acquired during real games often have highly variable quality. A simple yet effective metric to assess data quality is the percentage of samples in which the ball coordinates are missing. A slightly more informative version of this metric can be obtained by considering only the samples in which the game is active, i.e., the game state is not DEAD. The eight games with the lowest percentage of missing ball coordinates, measured according to the latter metric, were included in this study. These games were then split in three subsets of respectively four, two, and two games. From the first subset, 100K samples were randomly chosen to create the training set; from the second subset, 5K samples were randomly chosen to create the validation set; from the third subset 5K samples were randomly chosen to create the test set. By using different games in each subset, we aimed to have statistically independent data across the phases. Furthermore, the data in the three subsets were acquired in different stadiums and with different teams involved to ensure that the model generalizes well in other contexts.

### *4.2. Implementation*

Each of the alternatives presented in Section 3 is defined by two independent factors, namely, the high-level architecture (i.e., single-branch, two-branch or two-networks configuration) and the permutation-invariant layers (i.e., reduce by sum, self-attention or 2D convolution). Both factors can be varied as desired even within the same architecture, e.g., it is possible to define a two-networks model in which one network uses self-attention and the other one uses 2D convolution. The only constraint in this sense is that in a two-branch model, it is not possible to combine a 2D convolution with another permutation-invariant function: in fact, when using 2D convolutions, the TCN outputs a single vector that is passed to both branches. Therefore, while it is possible in a two-branch network to use a sum layer in one branch and a self-attention layer in the other one, in the case of 2D convolution the choice affects necessarily both branches since the TCN is shared by both.

As said, each data sample is structured in a three-dimensional tensor of size $n_f \times n_o \times n_c$. In this work, we set $n_f = 64$, based on experimental evidence and domain knowledge. The target frame is the 48th element within the sequence, which means that the model is acausal.

The total number of objects $n_o$ is equal to 1 ball+1 referee+22 starting players + 12 substitutions = 36. Padding columns with empty values are added when the teams did not exploit all possible substitutions. Finally, the $n_c = 11$ channels are defined for each object with the following information:

- the $x$ and $y$ coordinates;
- the velocity in the $x$ and $y$ directions, computed by subtracting the coordinate vector in two adjacent time points and dividing it by the frame period;
- three channels encoding the role of the object, i.e., whether it is a ball, a referee or a goalkeeper;
- two channels encoding whether the object belongs to the home team or to the away team;
- a flag telling whether the object is located outside of the pitch;
- a flag telling whether the object is missing.

Before training, the data are *preprocessed* to ensure training convergence and reduce the effect of noise. The ball coordinates are first interpolated using the Akima spline (Akima, 1970). Then, each $x$ and $y$ coordinates are separately rescaled using min-max scaling so that they fall into the interval $[-1, 1]$. Missing data are assigned the value $-2$, and values far outside of the game field are truncated before scaling in order to provide more stability to normalization.

The network architecture has been described in detail in Section 3. The FFN consists of two fully connected layers, with 64 and 32 units, respectively. The TCN and the self-attention module are initialized according to the Xavier normalized algorithm, while the FFN initialization follows the Xavier uniform algorithm (Glorot and Bengio, 2010). All layers have the ReLU activation function, except for the FFN layers which use an ELU activation. All networks were implemented in Python based on Keras v2.4.3 e Tensorflow v2.3.1. For training, the Adam (Kingma and Ba, 2014) optimizer was used, with an initial learning rate of $10^{-5}$ and a decay rate of 0.7 after each epoch.

### *4.3. Performance assessment*

The models were evaluated on the basis of three accuracy metrics. First, *global accuracy* is considered, i.e., the percentage of correct predictions among all predictions made within the ternary classification setting presented in the problem statement. Global accuracy can be thus expressed as

$$acc_{global} = \frac{\#\text{correct predictions}}{\#\text{all predictions}} \tag{13}$$

Especially for multi-branch and multi-network models, it is also interesting to consider two additional metrics, namely *dead-alive accuracy*, $acc_{DA}$, and *possession accuracy*, $acc_{POSS}$. These measures represent the ability of a model to solve one of the two sub-tasks into which the problem can be decomposed. In particular, the dead-alive accuracy represents the percentage of samples for which the model correctly identifies whether the game is active or not and is computed as

$$acc_{DA} = \frac{\#tp_{DA} + \#tn_{DA}}{\#tp_{DA} + \#tn_{DA} + \#fp_{DA} + \#fn_{DA}} \tag{14}$$

where $tp_{DA}$ are the samples for which both the true and predicted labels are not DEAD, $tn_{DA}$ are the samples for which both the true and predicted labels are DEAD, $fp_{DA}$ are the samples for which the true label is DEAD while their predicted label is not DEAD, and $fn_{DA}$ is the opposite case. On the contrary, the possession accuracy represents the percentage of samples for which the game is active, and the model correctly identifies which team owns the ball. It is computed as

$$acc_{POSS} = \frac{\#tp_{POSS} + \#tn_{POSS}}{\#tp_{POSS} + \#tn_{POSS} + \#fp_{POSS} + \#fn_{POSS}} \tag{15}$$

where $tp_{POSS}$ are the samples for which both the true and the predicted label are HOME, $tn_{POSS}$ are the samples for which both the true and the predicted label are AWAY, $fp_{POSS}$ are the samples for which the true label is AWAY while their predicted label is HOME, and $fn_{POSS}$ is the opposite case. It is thus important to note that $acc_{POSS}$ only considers those samples for which true label is not DEAD, i.e., those frames where the game is active.

For the selected architectures, the inference time (mean and standard deviation) needed to process one batch was calculated. Execution time was measured on a PC equipped with an NVIDIA 1080Ti GPU with 11Gb VRAM, 32G RAM and Intel i7-7700 CPU @ 3.60GHz.

## 5. Results

The goal of this section is to provide an evaluation of the presented methods. Thus, in Section 5.1, different design alternatives are compared in order to identify the best model to solve the problem statement. This model is then compared in Section 5.2 with other methods taken from the existing literature on related topics. Finally, in Section 5.3, some ablation studies are conducted in order to identify which parts of the model contribute most to the final outcome.

### *5.1. Comparison of design alternatives*

The results obtained by comparing different design alternatives are shown in Table 2, where each row represents a different combination of architecture and aggregation function. Overall, most of the results are within a small range: the mean accuracy ($\pm$ standard deviation) for all models is 82.93% $\pm$ 1.71%. On average, the accuracy achieved with single-branch (83.11% $\pm$ 1.39%) and two-branch architectures (83.79% $\pm$ 1.85%) is higher than the two-networks solution (82.40% $\pm$ 1.82%).

We also measured inference time for the best performing network, i.e., the two-branch network with self-attention aggregation layers. The average time ($\pm$ standard deviation) needed to process one batch is equal to 37.18 ms $\pm$ 4.66 ms for a batch size of 1, 57.47 ms $\pm$ 5.94 ms for a batch size of 16, and 52.39 ms $\pm$ 1.26 ms for a batch size of 32. Given that the data is sampled at 16 frames/s, processing times are comparable with real-time inference even on relatively low-performance, consumer-grade GPU.

| **Architecture** | **Aggregation function** | $acc_{global}$ |
|---|---|---|
| Single-branch | sum | 83.42 % |
| | self-att. | 84.32 % |
| | 2D-conv. | 81.6 % |
| Two-branch | sum + sum | 82.74 % |
| | self-att. + sum | 84.55 % |
| | sum + self-att. | 83.78 % |
| | self-att. + self-att. | 86.39 % |
| | 2D-conv. | 81.49 % |
| Two-networks | sum + sum | 82.78 % |
| | self-att. + sum | 84.44 % |
| | 2D-conv. + sum | 82.38 % |
| | sum + self-att. | 82.86 % |
| | self-att. + self-att. | 84.32 % |
| | 2D-conv + self-att. | 79.42 % |
| | sum + 2D-conv. | 79.62 % |
| | self-att. + 2D-conv. | 82.16 % |
| | 2D-conv + 2D-conv. | 83.64 % |

Table 2: Performance (global accuracy) of different design alternatives. The first column refers to the high-level architectures, whereas the second column reports the permutation-invariant aggregation function $\Lambda$. For multi-branch/multi-network architectures, the first aggregation function refers to the DA branch/network, whereas the second one refers to its POSS counterpart.

### 5.2. Comparison with the state of the art

In order to fully assess the contribution of this work, it is important to provide a quantitative analysis with respect to the state of the art. Since there are no works that address the overall problem of estimating the game state, the comparison will be made separately with respect to the two subtasks of estimating the densities $P(Y_{DA} \mid X)$ and $P(Y_{POSS} \mid X, Y_{DA} = \text{ALIVE})$.

First of all, the classification between active and inactive game phases is considered, comparing the model presented in this work with the one from Wei et al. (2013), which uses a decision tree trained with the ball coordinates only. Each model is tested on 20K samples randomly selected from two games, chosen among those that were not used to train the neural network. As shown in Table 3 (upper part), the network greatly outperforms the baseline model, which in turn performs only 6% better than a random classifier (since it is a binary classification and the classes are relatively balanced, a random classifier has a 50% chance of guessing the correct label). In Section 5.3 it will be also shown that, even if the network is provided only with the ball coordinates (thus holding out the players and the referee), it is still able to achieve 83% accuracy in the dead-alive problem, which is 27% better than the decision tree.

Regarding possession, the current work is compared with three methods, taken respectively from Link and Hoernig (2017), Morra et al. (2020) and Khaustov and Mozgovoy (2020). These works propose rule-based systems, in which possession is estimated starting from considerations drawn from domain knowledge, regarding, e.g., the closest player to the ball, the speed and acceleration of the ball, etc. Again, each model is tested on 20K samples randomly selected from two games; the test set is also pruned of those samples where the game is inactive, since the baseline models are designed for estimating ball possession only.

All competing models were reimplemented based on the available information from the original papers. Specifically, in Link and Hoernig (2017), the ball acceleration was computed as a finite difference starting from the ball coordinates. The threshold on the ball acceleration was set to $4\text{ms}^{-2}$, as proposed by the authors. The minimum distance $T_P$ between the player and the ball, used to discriminate if the player is interacting with the ball, is not provided in the work and was set through validation to 1.5m. In Morra et al. (2020), ball possession is estimated based on the distance from the closest player, the movement of the player and the ball speed, each controlled by a separate threshold. Hyperparameters were taken from the code released by the authors and set to 1.09m, 1.19m, and $8.6\text{ms}^{-1}$, respectively. As concerns Khaustov and Mozgovoy (2020), the algorithm as well as its hyperparameters are thoroughly listed in the paper and were kept unchanged.

The obtained results are listed in the lower part of Table 3, and show that the best performance is achieved by the neural network, with a margin of 7% in accuracy with respect to the best rule-based model, which is the one from Morra et al. (2020).

### *5.3. Ablation studies*

The goal of this section is to analyze which parts of the input data concur to the final result, in order to understand what aspects are deemed as more important by the network to produce the output, and what is ignored. In particular, ablation studies are performed on two axes: on the one hand,

| **Solution** | $acc_{DA}$ | $acc_{POSS}$ |
|---|---|---|
| Ours | 89.2% | 86.2% |
| (Wei et al., 2013) | 56.0% | - |
| (Link and Hoernig, 2017) | - | 64.5% |
| (Morra et al., 2020) | - | 79.1% |
| (Khaustov and Mozgovoy, 2020) | - | 75.4% |

Table 3: Comparison of our best model (two-branch network with self-attention aggregation layers) with the state of the art on the task of dead-alive classification ($acc_{DA}$) and possession classification ($acc_{POSS}$).

we evaluate what happens when we remove *objects*, in particular players; on the other hand, we investigate the role of individual *channels*, i.e., of the information related to each object. The two directions are followed separately in an orthogonal way, i.e., when objects are removed, all the channels are considered, and vice versa.

Ablation studies report all three different metrics introduced in Section 4.3. In fact, some objects – or some channels – may be important to determine only one of the two aspects, i.e., only if the game is active or which team owns the ball. The ultimate goal of this analysis is therefore to understand *which parts* of the input are important to produce *which parts* of the output. This is particularly relevant since, as it has been shown above, it is possible to build a model using two separate branches or even two separate networks, each of which performs a binary classification. Knowing which parts of the input data are more important for each prediction enables us to finetune separately the training of each branch/network.

Ablation studies are performed on an extended test set which includes 20 games, encompassing a larger variety, in terms of acquisition settings and data quality, with respect to the games included in the training set. The two-branch model with self-attention, which achieves the highest global accuracy as reported in Table 2, is selected as baseline.

#### *5.3.1. Ablation of objects*

The object ablation study progressively removes some of the objects from the input data. The input data consist of a tensor of size $n_f \times n_o \times n_c$, where $n_o$ amounts to 36, since it includes the ball, the referee and 17 players from each team (11 starting players and 6 possible incoming players). Performing an ablation study on the objects thus means to cut away a slice of the input on the second axis, passing to the network a tensor of size $n_f \times n'_o \times n_c$, where $n'_o$ is the number of objects that are kept.

The ablation is performed in two steps. First, the players far from the ball are removed. The distance can be computed in different ways: here, the Euclidean distance is considered at the frame in which the game state should be estimated. This approach, based on the idea of the K-nearest neighbors (KNN) algorithm, is rather common and can be found in several works from the literature (Sanford et al., 2020) (Mehrasa et al., 2018). In the second step, a more aggressive ablation is performed, and only the ball is retained: the intuition behind this choice is that the ball trajectory, by itself, carries a considerable part of the information.

The results in terms of global accuracy are shown in Fig. 3. The blue dots in the figure represent the baseline, which achieves a mean accuracy of 81.59% on the test set, as shown in Table 4. The yellow dots refer to the

| **Ablation** | $acc_{global}$ | $acc_{DA}$ | $acc_{POSS}$ |
|---|---|---|---|
| baseline | 81.59 % | 88.25 % | 84.95 % |
| 5NN | 79.41 % | 85.8 % | 84.98 % |
| ball only | 58.35 % | 83.62 % | 50.54 % |
| $(x, y)$ + roles | 67.25 % | 74.89 % | 82.09 % |
| $(x, y)$ only | 56.16 % | 75.81 % | 51.89 % |

Table 4: Mean accuracies of different ablation models.

model trained using the ball and its five nearest players (5NN) and performs about 2% worse than the baseline. Finally, the red dots show the performance when the model is trained using only the tracking data of the ball: this leads to a considerable drop in the accuracy, since only 58% of the samples are classified correctly on average.

Table 4 compares models also with respect to the additional metrics $acc_{DA}$ and $acc_{POSS}$. The latter presents a similar trend as the global accuracy: the 5NN model performs on par with the baseline, while the ball-only model performs significantly worse. On the contrary, in order to estimate if the game is active, it is useful to include all players, since there is a 2.5% difference in $acc_{DA}$ between the 5NN model and the baseline (which ultimately causes the difference in global accuracy). Most interestingly, it can be noticed that the ball trajectory alone is able to achieve a good 83.62% mean $acc_{DA}$.

#### *5.3.2. Ablation of channels*

Channel ablation studies aims to explain which part of the information about each object are important to produce the output result. In the original

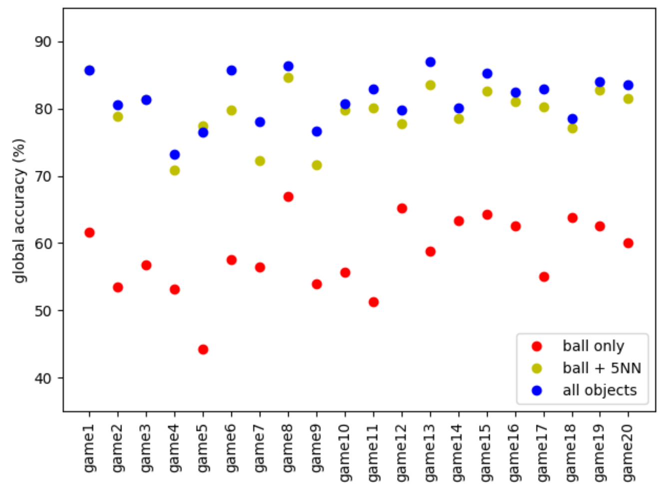



Figure 3: Results of the object ablation study. Each dot represents the global accuracy of the baseline (blue), ball + 5NN (yellow) and ball only (red) models, calculated separately on each game in the extended test set.

input data, 11 channels are passed to the network, including some hand-crafted features, such as velocity, pre-computed in the data pre-processing phase. The goal of this section is therefore to identify which information can be considered as redundant, and whether the designed architecture is capable of automatically encoding or compute features from the raw spatial coordinates, if they are indeed relevant for the classification. Since one of the most relevant characteristics of neural networks is their ability to recognize hidden patterns, which avoids the need to hand engineer the input features as it was typical of the earlier machine learning techniques, we aim to measure up to which point the network is able to fulfill this expectation, and conversely when it is better to provide some explicit information in order to improve the performance.

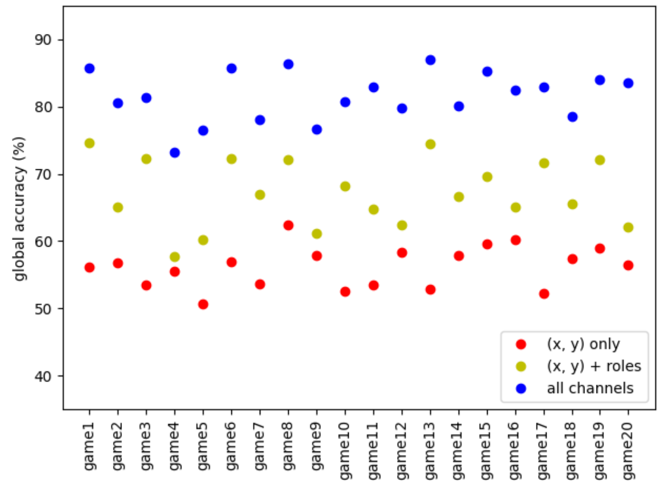



Figure 4: Results of the feature ablation study. Each dot represents the global accuracy of the baseline (blue), $(x, y)$ coordinates + roles (yellow) and $(x, y)$ coordinates only (red) models, calculated separately on each game in the extended test set.

As in the previous section, the ablation is done in two steps: in the first step, information about the coordinates, the roles and the team is kept (in total seven channels), whereas in the second step only the two coordinates are used. The detailed results in terms of global accuracy are shown in Fig. 4, whereas the mean accuracy across the 20 games in the test set are reported in Table 4. The results show a clear difference between the three models: the baselines achieves 81.59% mean accuracy, the model using only the roles has 67.25%, whereas the model that uses only the spatial coordinates has 56.16%. This means that, from a general point of view, all groups of channels make a significant contribution to the output.

When considering separately the accuracy on the two binary classification settings, however, it is possible to note some differences. In fact, in terms of the dead-alive accuracy, the role model (i.e., the first ablation model) has nearly the same performance as the coordinate model (the second ablation model), which indeed performs a slight 1% better. On the contrary, with respect to the possession accuracy, the role model has a performance less than 3% worse than the baseline, whereas the coordinate model achieves as little as 51.9% accuracy.

## 6. Discussion

In this paper, we have investigated different TCN architectures to estimate the state of a soccer game starting from spatio-temporal data about players and ball positions. All proposed architectures are based on common principles: first, TCNs are employed to map trajectories into an embedding space, and second, the architecture is designed to be permutation-invariant with respect to the orders of the players. However, they differ with respect to other design choices, such as the number of branches, the choice of the permutation-invariant aggregation function, and the loss, which were experimentally compared in this paper.

With respect to the *global architecture*, the two-networks architecture, in which dead-alive classification and possession estimation are predicted by two separate networks, performs on average worse than those based on a single network. A possible interpretation is that in order to build effective trajectory embeddings, training simultaneously on samples from both active and inactive game phases is more beneficial than having a more flexible network with a higher number of parameters. When training on related tasks, multi-task learning can improve performance by promoting implicit regularization and more robust feature representation (Ruder, 2017), (Vandenhende et al., 2020). In addition, models consisting of two separate networks may need significantly more resources for both training and inference.

Taking into account the trade-off between training time and performance, as well as between memory and performance, the single-branch models achieve results that are often similar or even better than more complex variants. For example, when using 2D convolution, a single classification branch does not perform worse than its two-branch counterpart. From a computational perspective, the processing time of the dual-branch architecture with self-attention is low and even compatible with real-time use. However, it must be stressed that the processing time to extract players and ball tracking data from sensors and/or videos was not considered in the present work. At the same time, many sports analytics pipeline do not require real-time processing capabilities, but rather high accuracy.

The choice of the *aggregation function* $\Lambda$ has a moderate impact on the overall performance. Most of the information can be captured by simple functions, such as summing over all trajectory embeddings. Yet, the best overall performance (86.39% global accuracy) is achieved by the two-branch model using self-attention in both branches: self-attention is the most elaborated of the three aggregation functions, and allows to capture task-specific features that cannot be recognized otherwise.

*Another important aspect to consider is how different input features affect the overall performance.* In this case, the input is composed by multiple objects (i.e., the players and the ball), each further characterized by several features (or channels), including the ($x$, $y$) coordinates, additional features related to the position (the velocity in the $x$ and $y$ directions, whether the object is located inside or outside the pitch, and whether it is missing), the role played by each object, and the team. Both aspects were studied through extensive ablation studies. In order to globally classify the game state, it is not possible to consider only the position of the ball, as the accuracy drops slightly above chance level ($acc_{global}$=58.35%). However, our ablation studies show that, on average, it is sufficient to consider the five players closest to the ball at the beginning of the sequence ($acc_{global}$=79.41%). It should be noticed that, because the distance is computed only at one point in the sequence, samples in which the ball is kicked at the beginning of the sequence could be misclassified (e.g., in the case of a long pass to an empty area of the pitch, in which the possession does not change even if the passing player is very far from the ball at the moment of the evaluation).

However, the input information required for each specific task is different. To determine whether the game state is active or not, the trajectory of the ball alone achieves a strong performance ($acc_{DA}$ = 83.62%), quite close to the baseline ($acc_{DA}$ = 88.25%): hence, ball tracking information accounts for 94% of the information captured by the network that allows to determine whether the game state is active or not. Removing information about all but the closest players also reduces the performance by 2.5% ($acc_{DA}$ = 85.8%). On the contrary, in order to estimate ball possession, it is sufficient to include the five nearest players ($acc_{POSS}$ = 84.98%) to achieve comparable results to the baseline ($acc_{POSS}$ = 84.95%), whereas ball tracking information alone cannot reach accuracy above chance level.

Similar considerations apply for the features (channels) associated with each object. In terms of dead-alive classification, removing velocities and position with respect to the external line has a large impact on accuracy. In fact, accuracy when using only ($x$, $y$) coordinates drops significantly ($acc_{DA} = 75.81\%$), and adding role information even slightly degrades performance ($acc_{DA} = 74.89\%$). On the other hand, with respect to possession accuracy, role information is crucial, whereas velocities and other features play a minor role: in fact, a model that takes as input only position and role of each object achieves accuracy comparable to the baseline ($acc_{POSS} = 82.09\%$ vs. $acc_{POSS} = 84.89\%$). Both these insights are in line with intuition: in order to tell if the game is active, it is important to know the velocity of the objects (e.g., to know if the ball is moving) and if they are inside the pitch, whereas to assign the ball possession it is essential to correctly assign each object to the proper team.

The results of the ablation studies are consistent with those of the comparison of different architectures. In fact, a two-branch model that uses self-attention in both branches would be able to automatically select the most relevant features for each task. On the other hand, if a two-networks architecture is selected, it would be advisable to tailor the input data passed to each network in order to maximize the performance of the system. Likewise, in a two-branch model, since the trajectories are computed separately for each object, it is possible to pass only a subset of the embeddings to each branch, based on which objects are most important for the classification. For example, if only the nearest players are needed to determine $Y_{POSS}$, it would be reasonable to prune the input of the POSS branch in Fig. 1b, selecting only the trajectory embeddings related to the objects needed.

Finally, *the proposed model outperforms previously published solutions* on both possession accuracy (+7%) (Link and Hoernig, 2017; Morra et al., 2020; Khaustov and Mozgovoy, 2020) and game state classification (+27%) (Wei et al., 2013). The most recent competing methods (Morra et al., 2020; Khaustov and Mozgovoy, 2020) are based on rules or temporal logic techniques; these methods do not require training, but may include provisions to tune rule-specific hyper-parameters (Morra et al., 2020). It is worth noticing that all previous techniques were reimplemented and tested on the same dataset to ensure a fair comparison; however, hyper-parameters were kept to the original values proposed by the authors, and were thus tuned on different datasets, at least in one case leveraging synthetic datasets (Morra et al., 2020). The comparison offers an interesting insight about the trade-offs present in rule-based and deep learning models. On the one hand, handcrafted rules allow to build hierarchical models, which can be expanded more easily (e.g., to perform event detection) and often have nice by-products, such as the fact that the possession estimation is already done at individual level. However, this may come at a price in terms of performance, since neural networks often present a greater flexibility that allows them to learn more difficult mappings. In this case, it is particularly reasonable to opt for a deep learning system because the dataset is quite big, which allows to train larger and more powerful networks with little impact on generalization.

## 7. Conclusions and future work

This study aimed to devise a deep learning system capable of estimating the state of a soccer game on a frame-by-frame basis given a set of spatio-temporal tracking data. The best performing architecture is a two-branch architecture which exploits a TCN backbone to extract trajectory embeddings for each object/player, and self-attention modules to aggregate embeddings in a permutation-invariant way. Extensive experimental analysis on tracking data from a professional soccer league show that the proposed method outperforms, by a large amount, state-of-the-art rule-based systems in both dead-alive classification and ball possession classification.

The present study can be considered as a stepping stone towards automating a task that presently requires constant human input and supervision. At the same time, it represents an important contribution to the state of the art, which currently lacks methods to simultaneously and reliably estimate ball possession and game state. From a technical point of view, this study proved that techniques and network architectures that have been successfully developed in similar fields, such as event detection, can be applied in the context of ball possession as well. This work also systematically compares different techniques for achieving permutation invariance on set-based data, which may be of interest for other applications based on the analysis of tracking data.

Ample directions for future research emerge from the results of the present study. For instance, the dataset used in this work is based on cameras providing only $x$ and $y$ coordinates: improvements in the accuracy of the model could be achieved by leveraging more advanced systems that provides a very accurate tracking of the ball, including the $z$ coordinate. Regarding the methodology, an interesting alternative to the approach adopted here could be to estimate the game state from a set of events *by subtraction*, i.e., by detecting all the events that determine a change in the game state, and segmenting the game accordingly. In this way, it would be possible to exploit the large body of existing literature in the field of event detection, as well as to take one more leap in the direction of an end-to-end deep learning system capable of analyzing spatio-temporal data. Clearly, this would also require the availability of a more fine-grained annotated dataset, including information on the individual players as well as the team in the classification.

## Acknowledgements

We would like to thank Ms. Beatrice Gaviraghi, former Math&Sport project manager, for her consistent support, assistance and organization especially during the initial phases of this research work.

## References

Akima, H. (1970). A new method of interpolation and smooth curve fitting based on local procedures. *Journal of the ACM (JACM)*, 17(4):589–602.

Bai, S., Kolter, J. Z., and Koltun, V. (2018). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*.

Bialik, C. (2014). The people tracking every touch, pass and tackle in the world cup. `https://fivethirtyeight.com/features/the-people-tracking-every-touch-pass-and-tackle-in-the-world-cup/`. Accessed: 2021-06-25.

Cao, Z., Hidalgo, G., Simon, T., Wei, S.-E., and Sheikh, Y. (2021). Openpose: Realtime multi-person 2D pose estimation using part affinity fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(1):172–186.

Carreira, J. and Zisserman, A. (2017). Quo vadis, action recognition? A new model and the kinetics dataset. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 6299—6308.

Farhadi, J. R. S. D. R. G. A. (2016). You Only Look Once: Unified, real-time object detection. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 779–788.

Fernández, J., Bornn, L., and Cervone, D. (2019). Decomposing the immeasurable sport: A deep learning expected possession value framework for soccer. In *Proc. 13 th Annual MIT Sloan Sports Analytics Conference*.

Gao, X., Liu, X., Yang, T., Deng, G., Peng, H., Zhang, Q., Li, H., and Liu, J. (2020). Automatic key moment extraction and highlights generation based on comprehensive soccer video understanding. In *Proc. IEEE International Conference on Multimedia and Expo Workshops*, pages 1—-6.

Glasser, H. (2014). The problem with possession: The inside story of soccers most controversial stat. `https://slate.com/culture/2014/06/soccer-possession-the-inside-story-of-the-games-most-controversial-stat.html`. Accessed: 2021-06-25.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth inter-*

*national conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings.

Guirguis, K., Schorn, C., Guntoro, A., Abdulatif, S., and Yang, B. (2021). SELD-TCN: sound event localization & detection via temporal convolutional networks. In *2020 28th European Signal Processing Conference (EUSIPCO)*, pages 16–20. IEEE.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proc. IEEE conference on computer vision and pattern recognition*, pages 770—-778.

Hong, Y., Ling, C., and Ye, Z. (2018). End-to-end soccer video scene and event classification with deep transfer learning. In *Proc. 2018 International Conference on Intelligent Systems and Computer Vision*, pages 1—-4.

Horton, M. (2020). Learning feature representations from football tracking. In *Proc. MIT Sloan Sports Analytics Conference*.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

Jiang, H., Lu, Y., and Xue, J. (2016). Automatic soccer video event detection based on a deep neural network combined CNN and RNN. In *Proc. IEEE 28th International Conference on Tools with Artificial Intelligence*, pages 490—-494.

Khan, A., Lazzerini, B., Calabrese, G., and Serafini, L. (2018a). Soccer event detection. In *Proc. 4th International Conference on Image Processing and Pattern Recognition*, pages 119—-129.

Khan, M. Z., Saleem, S., Hassan, M. A., and Khan, M. U. G. (2018b). Learning deep C3D features for soccer video event detection. In *2018 14th International Conference on Emerging Technologies (ICET)*, pages 1–6. IEEE.

Khaustov, V. and Mozgovoy, M. (2020). Recognizing events in spatiotemporal soccer data. *Applied Sciences*, 10(22):8046.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kukleva, A., Asif, M., Hafez Farazi, K., and Behnke, S. (2019). Utilizing temporal information in deep convolutional network for efficient soccer ball detection and tracking. In *Robot World Cup*, pages 112—-125.

Lee, J., Kim, Y., Jeong, M., Kim, C., Nam, D.-W., Lee, J., Moon, S., and Yoo, W. (2018). 3D convolutional neural networks for soccer object motion recognition. In *Proc. 20th International Conference on Advanced Communication Technology*, pages 354—-358.

Link, D. and Hoernig, M. (2017). Individual ball possession in soccer. *PLoS One*, 12(7).

Liu, T., Lu, Y., Lei, X., Zhang, L., Wang, H., Huang, W., and Wang, Z. (2017). Soccer video event detection using 3d convolutional networks and shot boundary detection via deep feature distance. In *Proc. International Conference on Neural Information Processing*, pages 440—-449.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Scott, R., Fu, C.-Y., and Berg, A. C. (2016). SSD: Single Shot MultiBox Detector. In *Proc. 14th European Conference on Computer Vision*, pages 21–37.

Lucey, P., Bialkowski, A., Carr, P., Morgan, S., Matthews, I., and Sheikh, Y. (2013). Representing and discovering adversarial team behaviors using player roles. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 2706—2713.

Martin, P.-E., Benois-Pineau, J., Péteri, R., and Morlier, J. (2018). Sport action recognition with siamese spatio-temporal CNNs: Application to table tennis. In *Proc. International Conference on Content-Based Multimedia Indexing*, pages 1—-6.

Mehrasa, N., Zhong, Y., Tung, F., Bornn, L., and Mori, G. (2018). Deep learning of player trajectory representations for team activity analysis. In *11th MIT Sloan Sports Analytics Conference*, volume 2, page 3.

Memmert, D. and Rein, R. (2018). Match analysis, big data and tactics: Current trends in elite soccer. *German Journal of Sports Medicine*, 69(3):65–71.

Morra, L., Manigrasso, F., Canto, G., Gianfrate, C., Guarino, E., and Lamberti, F. (2020). Slicing and dicing soccer: Automatic detection of complex events from spatio-temporal data. In *Proc. International Conference on Image Analysis and Recognition*, pages 107—-121.

Richly, K., Moritz, F., and Schwarz, C. (2017). Utilizing artificial neural networks to detect compound events in spatio-temporal soccer data. In

*Proc. 3rd SIGKDD Workshop on Mining and Learning from Time Series*, pages 13—-17.

Rockson, A., Rafiq, M., and Gyu Sang, C. (2019). Soccer video summarization using deep learning. In *Proc. IEEE Conference on Multimedia Information Processing and Retrieval*, pages 270—-273.

Roy Tora, M., Chen, J., and Little, J. J. (2017). Classification of puck possession events in ice hockey. In *IEEE Conference on Computer Vision and Pattern Pecognition Workshops*, page 147–154.

Ruder, S. (2017). An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*.

Sanford, R., Gorji, S., Hafemann, L. G., Pourbabaee, B., and Javan, M. (2020). Group activity detection from trajectory and video data in soccer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 898–899.

Sarkar, S., Chakrabarti, A., and Mukherjee, D. P. (2019). Generation of ball possession statistics in soccer using minimum-cost flow network. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 2515–2523.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Sorano, D., Carrara, F., Cintia, P., Falchi, F., and Pappalardo, L. (2020). Automatic pass annotation from soccer video streams based on object detection and LSTM. *arXiv preprint arXiv:2007.06475*.

Theagarajan, R., Pala, F., Zhang, X., and Bhanu, B. (2018). Soccer: Who has the ball? Generating visual analytics and player statistics. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition Workshops*, page 1749–1757.

Tran, D., Bourdev, L., Fergus, R., Torresani, L., and Paluri, M. (2015). Learning spatiotemporal features with 3D convolutional networks. In *Proc. IEEE international conference on computer vision. 2015*, pages 4489––4497.

van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. In *9th ISCA Speech Synthesis Workshop*, pages 125–125.

Vandenhende, S., Georgoulis, S., Proesmans, M., Dai, D., and Van Gool, L. (2020). Revisiting multi-task learning in the deep learning era. *arXiv preprint arXiv:2004.13379*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30:5998–6008.

Wei, X., Sha, L., Lucey, P., Morgan, S., and Sridharan, S. (2013). Large-scale analysis of formations in soccer. In *2013 international conference on digital image computing: techniques and applications (DICTA)*, pages 1–8. IEEE.

Xu, J., Kanokphan, L., and Tasaka, K. (2018). Fast and accurate object detection using image cropping/resizing in multi-view 4k sports videos. In *Proc. 1st International Workshop on Multimedia Content Analysis in Sports*, pages 97—-103.

Xu, J. and Tasaka, K. (2020). Keep your eye on the ball: Detection of kicking motions in multi-view 4K soccer videos. *ITE Transactions on Media Technology and Applications*, 8(2):81—-88.

Yu, J., Lei, A., and Hu, Y. (2019). Soccer video event detection based on deep learning. In *Proc. International Conference on Multimedia Modeling*, page pp. 377–389.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Póczos, B., R., S., and J., S. A. (2017). Deep Sets. In *Proc. Annual Conference on Neural Information Processing Systems*.