Predicting Goal-Scoring Opportunities in Soccer by Using Deep Convolutional Neural Networks

Martijn Wagenaar

16 November 2016

Master's Thesis

Department of Artificial Intelligence,
University of Groningen,
The Netherlands

Internal supervisor
Dr. M.A. Wiering,
Artificial Intelligence & Cognitive Engineering,
University of Groningen

External supervisor
Dr. W.G.P. Frencken,
Football Club Groningen;
Center of Human Movement Sciences,
University of Groningen

university of groningen

faculty of mathematics and natural sciences

Abstract

Over the past decades, soccer has encountered an enormous increase in professionalism. Clubs keep track of their players' mental condition, physical condition, performances on the pitch, and so on. New technology allows for the automation of some of these processes. For instance, position data captured from soccer training and matches can be used to keep track of player fitness. There are opportunities for the use of position data in artificial intelligence as well. In the current research, position data gathered from matches played by a German Bundesliga team has been used to predict goal-scoring opportunities. The problem was approached as one of classification: given a snapshot of position data, is it more likely that a goal-scoring opportunity will be created or that ball possession will be lost? Snapshots of position data were taken and transformed to 256×256 images, which were used as input for machine learning algorithms. The performance of two deep convolutional neural networks was compared: an instance of GoogLeNet and a less complex 3-layered net. GoogLeNet came out as the best performing network with an average accuracy of 67%. Although the final performance was not spectacular, there are some promising indicators for future research and possible practical uses.

Contents

1 Introduction

Over the past decades, soccer has encountered an enormous increase in professionalism. The major clubs spend millions on transfers and salaries. Performances on the pitch are not only reflected in the standings, but also have substantial financial consequences. When a team does not do well for a number of matches, the coach is often quickly replaced.

The increase in professionalism includes extensive monitoring of players. Clubs keep track of their players' mental condition, physical condition, performances on the pitch, and so on. New technology allows clubs to partially automate some of these processes. For example, position data of players can help in analyzing the physical condition of players, as it can be used to calculate the distance a player has covered during a match, what their average speed was, how fast the player did accelerate, etcetera.

The large amount of available data offers opportunities in the field of computer science and artificial intelligence. When every position of every object on the field at any time is known, one could try to use these data in order to predict certain match events right before they occur. Particularly interesting is the occurrence and prediction of goal-scoring opportunities. In the end, scoring goals is what clubs, coaches and players want to achieve during a soccer match.

The prediction of goal-scoring opportunities can be approached as a classification problem. The occurrence of a goal-scoring opportunity then takes the role of a positive example, while the occurrence of the opposite (the non-occurrence of the aforementioned event) fulfills the role of a negative example. When labels are present, supervised machine learning methods can be applied in order to classify snapshots of a soccer match as either possible goal-scoring opportunities or as less promising states.

In the current research, goal-scoring opportunities were detected by analyzing soccer match position data. Abstract image representations of the intervals around goal-scoring opportunities (and non-opportunities) were used as input to convolutional neural networks [37]. Convolutional neural networks are particularly interesting because they might be able to detect higher-order tactical patterns in soccer by repeated convolutions. The convolutional neural networks could then be used to classify soccer snapshots into a class indicating that a goal-scoring opportunity might arise and a class which indicates the opposite scenario.

1.1 Research question

The main research question of this research is as follows:

Research question Can convolutional neural networks be used to classify and predict goal-scoring opportunities when presented with position data from soccer matches?

1.2 Thesis structure

This thesis mainly focuses on machine learning methods. After all, these are the methods that were used to predict goal-scoring opportunities. In chapter 2, however, some background about soccer game dynamics from a sports science perspective will be given. Only a small subset of all research on this topic used machine learning methods on position data.

Chapter 3 is about neural networks and machine learning. Basic descriptions of several types of neural networks are given. At the end of the chapter, an outline of convolutional neural networks will be given. The chapter concludes with describing several state of the art convolutional neural networks, which have shown spectacular performance in the Imagenet challenge [9].

Chapter 4 is all about the dataset. The chapter zooms in on several characteristics of the data set and how they were dealt with. Definitions of goal-scoring opportunities and ball possession are given, because of their importance for extracting training, validation and test data for machine learning. The last section of the chapter is about data exploration: some traditional and less traditional visualizations are applied to the data set in order to examine its contents.

Chapter 5 lists the experiments that were run and the results. While chapter 4 zooms in on initial preprocessing of the data, this chapter describes the process of taking processed data and transforming it to image datasets for machine learning. A total of three experiments with sub-experiments are described whose results are presented at the end of the chapter.

In the final chapter of this thesis, chapter 6, the results of the experiments are discussed and a conclusion is formulated. Finally, suggestions for future research are given.

Capturing the dynamics of soccer 2

The current standard to assess tactical performance is by game observation. Human observers rely on their knowledge about the game to extract certain tactical performance indicators, often guided by a set of rules, or handbook, which highlight factors of attention [27]. As a consequence, this type of analysis is subjective and slow. An automated system for tactical performance assessment would eliminate these major downsides of game observation by humans.

The main problem with building an automated system is to determine which indicators of performance it should take into account. A traditional method to capture soccer match dynamics is to keep track of the frequency of occurrences of specified events. These events can range from events which are relatively easy to detect automatically, such as the total number of passes, the percentage of successful passes and the number of shots on goal to more explicitly defined events such as the number of key passes of a specific player. Frequencies of event occurrences have successfully been used to predict the outcome of soccer matches [38] and have been shown to be able to discriminate between successful and unsuccessful teams [7].

Frequencies of event occurrences of specified events do not tell all. One could miss out on 'complex series of interrelationships between a wide variety of performance variables' when not looking beyond frequencies [4]. Among factors that are missed out on are higher-order tactical patterns which emerge during soccer. The following sections go into further detail on attempts that have been made to extract these higher-order tactical patterns from soccer position data.

2.1 Machine learning: object trajectories and player formations

There has been some research on soccer position data focusing on trajectories of objects and player formations involving machine learning techniques.

Knauf et. al. proposed a class of spatio-temporal convolution kernels to capture similarities between trajectories of objects [22]. The clustering method has been applied successfully to soccer data. Knauf et. al. distinguished two different *trajectory* *sequences*: game initiations and scoring opportunities. Game initiations began with a pass from the goal keeper and ended when possession was lost or the ball was carried to the attacking third of the field (or the start of a new game initiation). Scoring opportunities marked the event of carrying the ball to a 'predefined zone of danger' in the attacking area of the field. After clustering on a particular set of trajectory sequences, the clusters represented tactical patterns which could ultimately be used for classification.

A similar approach involving a variant of self-organizing maps [23] focused on detecting formations rather than trajectories [13]. Two interactions between players were investigated: the interaction between the four attacking players of the French team, the four defending players of the Italian team and the ball during FIFA World Cup 2006, and vice versa. The tactical patterns that were fed to the network were short game initiations and long game initiations. Both tactics started when the ball was won from the opposing team by either the goalkeeper or the defense players. When the pass following this gain of ball possession exceeded 30m, the tactical pattern was classified as a long game initiation. Otherwise, the tactical pattern was called a short game initiation. After training and manual expert labeling of the self-organizing map's output layer, the self-organizing map successfully detected 84% of all game initiations.

Memmert et al. outline other opportunities in the field of self-organizing maps [28]. A special self-organizing map was trained with examples of defensive and offensive patterns from the UEFA Champions League quarterfinal of FC Bayern Munich against FC Barcelona from the 2008/2009 season. The research illustrates the use of self-organizing maps in evaluating tactical formations. The neural network that they used to train on formations found that the most frequent defensive pattern of FC Bayern Munich (formation 3) led to obtaining ball possession for 40% of the total occurrences against the most frequently used offensive formation of FC Barcelona (formation 2, see figure 2.1).

2.2 Team centroids and surface area

The previous section focused on raw position data, where algorithms were used to extract the relevant information. Sometimes it is beneficial to guide an algorithm in the right direction, by pointing out which derived variables from position data could possibly be of use. Two of these derived variables are team centroids (the mean positions of the team's players) and the surface area of teams.

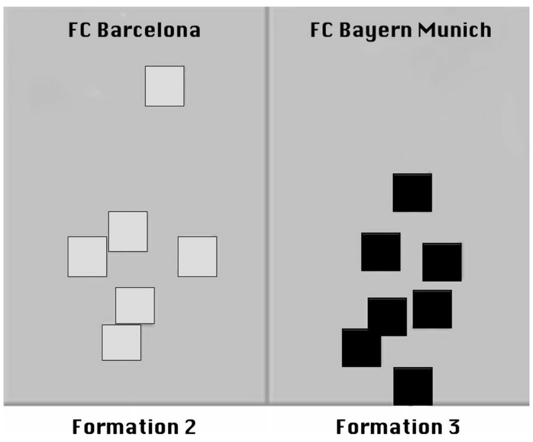


Fig. 2.1.: Most frequently used offensive formation from FC Barcelona (formation 2) and most frequently used defensive formation from FC Bayern Munich (formation 3) during the UEFA Champions League quarterfinal during the 2008/2009 season. Image copied from [28].

Frencken et al. found that the centroid position of a soccer team can provide valuable information about the 'coordinated flow of attacking and defending' in 5 versus 5 soccer matches [12]. A positive linear relation was found between the two teams' centroids: when the centroid of a team moved in a specific direction, the other teams' centroid did as well. This positive linear relation was present for both the y-direction (length) and x-direction (width). More interesting, deviations from the above described pattern occurred during the build-up of goals. For 10 out of 19 scored goals, a crossing of the centroids occurred, deviating from the positive linear relation.

While interesting findings, they do not directly extrapolate to 11 versus 11 soccer matches. For 11 versus 11 matches, the team centroids showed low variability. The interaction between each player and his position-specific centroid, however, has shown more potential to capture player movement behavior [36]. Two methods have been particularly successful in capturing player movement dynamics with respect to their team defending, midfield or forward centroids. An approximate entropy method evaluated the time series of individual player distances with respect to their accompanying centroids, and classified players into classes of predictability (low, medium and high predictability) [36, 34, 33]. A second method examined the relative phase of position-specific centroids, by analyzing two centroids as oscillators which could either be in-phase or not [25, 10].

Frencken et al. also examined possible linear relations between the surface areas of opposing teams, but none were found for 5 versus 5 matches. Other research has shown that the surface area of a team depends on the strength of the opposition [6]. When attacking, the surface area of a team was larger against weaker teams, while when defending, the surface area was larger against stronger opponents.

2.3 Clustering algorithms

Voronoi analysis of electronic soccer games has given some insights into soccer game dynamics [21]. Because it was, at the time of publication, hard to get data from real-world games, the video game FIFA Soccer 2003 (developed by EA Sports) was used to obtain a data set. It was argued that FIFA Soccer 2003 resembles soccer well and is similar to the actual game. Kim [21] used the positions of the players on the field as the point set to construct Voronoi diagrams. Kim argues that, when the total area of Voronoi segments of a team is larger than the area of the opponent's segments, the first team dominates the latter. Fonseca et al. found a similar result by analyzing games of futsal with Voronoi diagrams [11]. It was found that the area of the Voronoi segments - the total area of the 'dominant regions' of the players - was greater for the attacking team and smaller for the defending team.

Neural networks 3

This chapter gives a theoretical background of several types of neural networks. These types of neural networks will later be used to predict goal-scoring opportunities in soccer by using position data as the neural network's input.

3.1 Perceptron

The smallest building block of modern day's neural networks is the perceptron [35]. A perceptron maps its input vector **x** to a binary output $f(\mathbf{x})$ by computing the dot product between the input and a vector of weights **w** (see equation 3.1). If the outcome of this calculation is above a certain threshold value, the output of the perceptron is 1. If otherwise, the output is 0. In modern artificial neural networks, the threshold value is incorporated into the network by a bias term b. Figure 3.1 shows a schematic example of a perceptron.

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

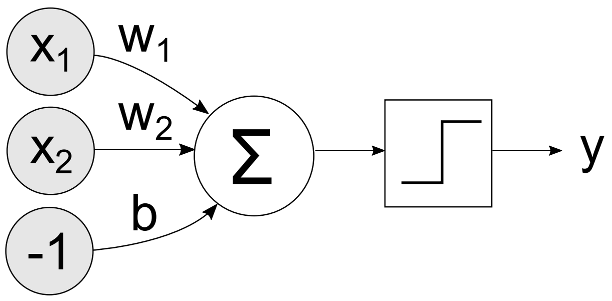


Fig. 3.1.: A perceptron with two inputs.

A single-layer perceptron is a linear classifier: the parameters of the network (weight vector and bias term) can be set to approximate a linear function. The parameters of a perceptron are determined by a learning process. In order to train a perceptron, a training set is required. In the training example below, $D = \{(\mathbf{x_1}, d_1), \ldots, (\mathbf{x_s}, d_s)\}$ is the training set where s is the number of training examples, $\mathbf{x_j}$ is the input vector of example j and d_j is the label (1 or 0) of example j.

A simple learning algorithm to train a perceptron is as follows:

1. Initialize the weight vector **w** to either zeros or random small values;
2. For each example j in the training set D:
    a) Calculate the actual output:
    $$y_j(t) = f[\mathbf{w}(t) \cdot \mathbf{x}_j] = f[w_0(t)x_{j,0} + w_1(t)x_{j,1} + \cdots + w_n(t)x_{j,n}] \quad (3.2)$$
    b) Update the weights:
    $$w_i(t+1) = w_i(t) + (d_j - y_j(t))x_{j,i} \quad (3.3)$$
3. Repeat step 2 until
    a) iteration error on time t is less than a user-specified threshold; or
    b) the algorithm has run for a predetermined maximum number of iterations.

3.2 Multi-layered perceptron and backpropagation

An extension of the perceptron is the multi-layered perceptron (MLP). An MLP consists of multiple neurons, where the perceptron only contained a single neuron, and can distinguish data that are not linearly separable. It has been shown that MLPs are capable of approximating any measurable function to any desired degree of accuracy [17].

MLPs consist of at least three layers: an input layer, one or more *hidden* layers and an output layer. An MLP is fully-connected: every neuron of layer k is connected to every neuron in layer $k+1$ by corresponding weights. To guarantee that an MLP is able to learn non-linear functions, non-linear *activation functions* are applied to the summed output of the hidden neurons. The outcome of the activation functions is then passed through to the neurons in the next layer. For MLPs, sigmoid functions are often used as activation functions, such as the hyperbolic tangent:

$$f(v_i) = tanh(v_i) \quad (3.4)$$

or the logistic function:

$$f(v_i) = (1 + e^{-v_i})^{-1} \tag{3.5}$$

where v_i is the weighted sum of the inputs connected to the ith neuron in a certain layer, and $f(v_i)$ is the output of that neuron.

Training an MLP differs a lot from training a perceptron, especially because of added non-linearities and an additional hidden layer. First, we would have to define a cost function which we would like to minimize in order to decrease the error. A frequently used cost function is the mean squared error:

$$E(n) = \frac{1}{2}\sum_j e_j^2(n) \tag{3.6}$$

where $e_j(n) = d_j(n) - y_j(n)$ is the error of output neuron j for the nth training example, $d_j(n)$ is the target value and $y_j(n)$ is the computed value by the MLP. The main problem is now to determine to which extent specific neurons have contributed to the error value. This is also known as the *credit assignment problem* in artificial neural networks: how much should a particular weight be adapted in order to minimize the error [14]?

The most common way is to propagate the error value through the network, starting from the output layer. This method is called *backpropagation* of the error [43]. Using gradient descent, we find the following general update rule for a weight $w_{ij}(n)$ where i reflects the node in the layer which is closest to the input.

$$\Delta w_{ij}(n) = -\eta \frac{\partial E(n)}{\partial v_j(n)} y_i(n) \tag{3.7}$$

where η is the learning rate and y_i the output of the previous neuron. Note that the weights are adapted in the direction of the negative gradient, hence the term gradient descent. The derivative term in equation 3.7 varies for hidden nodes and output nodes. For output neurons:

$$-\frac{\partial E(n)}{\partial v_j(n)} = e_j(n) f'(v_j(n)) \tag{3.8}$$

and for hidden neurons:

$$-\frac{\partial E(n)}{\partial v_j(n)} = f'(v_j(n)) \sum_k -\frac{\partial E(n)}{\partial v_k(n)} w_{jk}(n) \tag{3.9}$$

where f' is the derivative of the activation function used in the neural network. The sum term in equation 3.9, which sums over all output neurons k to which hidden neuron j is connected, shows that the hidden weight updates rely on computing the cost function derivative with respect to the output weights first: the error is *back*propagated. Although the above example is for a perceptron with only one hidden layer, deeper neural networks with a much more complex structure can still be trained by backpropagation. The weight update for a neuron then relies on all neurons that are located between this particular neuron and the neural network's output.

3.3 Guiding the learning process

Neural networks can contain thousands or millions of parameters which all have to be set to appropriate values to approximate a certain function. It is not hard to imagine that the process of learning is not straightforward. Over the past decades several methods have been developed that guide the learning process by making changes to the weight update equations for the neural network's parameters. The following sections will be about some of these methods, particularly the ones that are often used in deep neural networks.

3.3.1 Weight decay

It has been shown that penalizing large weights in a neural network can aid generalization [29]. The easiest way to achieve this is by adding a penalty term for large weights to the neural network's cost function:

$$E(\mathbf{w}) = E_0(\mathbf{w}) + \frac{1}{2}\lambda \sum_i w_i^2 \tag{3.10}$$

where E_0 is the original error function (such as sum of squared errors as in equation 3.6), $\mathbf{w}$ the weight vector containing all free parameters in the network and λ a hyper-parameter which determines to what extent large weights are penalized. When using gradient descent, the weight update function becomes as follows:

$$w_i(t) = w_i(t-1) - \eta \frac{\partial E}{\partial w_i(t-1)} - \lambda \eta w_i(t-1) \tag{3.11}$$

By penalizing large weights, it is attempted to push learning in a direction where all connections between neurons participate in producing the neural network's outcome. When weights are not penalized for becoming larger and larger, there is a possibility that only a few of the network's parameters are of importance and the majority is neglected. In such a scenario, using a much smaller network would then be able to achieve similar performance: the bigger network does not live up to its full potential.

3.3.2 Momentum

When adapting the weights in the direction of the negative gradient based on single examples (or single batches of examples), the weight adaptations tend to fluctuate. The weights are changed too much based on single examples, which blocks the path for the weights to find their optimal values. A simple method to counter this issue, is to introduce some sort of momentum to the network's weight update function [41, 19]. When certain weights are changed in the same direction for consecutive iterations, their momentum grows and the weights tend to be adapted in the same direction for the next iteration. Momentum can be implemented by altering the weight update function as follows:

$$w_i(t) = w_i(t-1) - \eta \frac{\partial E}{\partial w_i(t-1)} + \alpha \Delta w_i(t-1) \tag{3.12}$$

where α is the momentum parameter which decides how much the previous weight adaptation weighs.

3.3.3 Dropout

A recently developed technique which helps to prevent over-fitting in neural networks is *dropout* [39, 40]. When training data is limited, noise in the input data can cause the network to train on noise as if it were features of the input patterns. This ultimately leads to worse generalization and a lower test accuracy. Dropout tackles this issue by disabling neurons with a certain, pre-specified probability during the training phase. During one training iteration, each neuron is dropped with probability p. When $p = 0.5$, roughly half of the neurons are used for the forward and backward pass. Because fewer neurons are given the task to generate the desired output, they have to be more flexible and learn more general features. While this generally results in a higher training error, the test error is often greatly reduced, as has been shown with different sets of data [40].

Dropout can also be considered a form of model combination. With neural networks, it is often desirable to combine the output of multiple independently trained neural networks to obtain a better prediction [5, 2]. With very deep neural networks, for instance the convolutional neural networks used in the Imagenet competition, it is practically impossible to train multiple neural networks. The training time is simply too long. With dropout, for every iteration a unique model is used, due to the unique combination of available neurons.

3.4 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a type of neural networks which excessively make use of mathematical convolution to learn a function which is non-linear in nature. CNNs are particularly suited for image data because of convolution kernels which are shifted over an image. A fully-connected multi-layered perceptron would have too many connections and therefore focus too much on single pixels instead of patterns spanning multiple pixels.

The first CNN appeared in literature in 1990. LeCun et. al. used a small neural network incorporating two convolutional layers to recognize handwritten digits [26]. Over the last few years, CNNs and deep learning have experienced an enormous boost in popularity. This can mainly be ascribed to their recent successes in the Imagenet competition, which will be described in more detail in section 3.5.

The typical elements of CNNs are listed in the following subsections. For training deep neural networks, the training algorithm, or solver, differs slightly from the ones described before. The last part of this section lists the main differences between the training processes.

From now on, it is assumed that the convolutional neural networks are used for classification. The outputs of these networks are class-conditional probabilities: the networks output k numbers which indicate the probability that a presented input matrix belongs to a specific class.

3.4.1 Typical elements

A typical CNN consists of multiple layers. The first few layers of a CNN are often convolutional layers or pooling layers. The convolutional layers typically contain added nonlinearities (ReLUs) which process the output activations of the neurons. After the convolutional layers, mostly one or more fully-connected feedforward layers are used to obtain activation scores for every class. A softmax on the final output layer yields final class-conditional probabilities.

Convolutions

Theoretically, convolutional layers in a CNN can be described as being constraint fully-connected layers (where the constraints are that severe that the layers are not fully-connected anymore). The nature of the constraints, such as shared weights amongst subsets of neurons (the neurons belonging to a specific filter or kernel) and sparse connectivity (not every hidden or output neuron is connected to every input neuron), make that we can consider convolutional layers in a more practical way by addressing the principle of mathematical convolution. This section will primarily make use of the latter description.

The first convolutional layer in a CNN convolves an input representation, often in the form of multiple arrays, with a set of filters. Every filter, or kernel, in a convolutional neural network detects distinct features. A filter has its own set of weights which can be trained to extract specific features. For example, the weights of a 5×5 filter can be set to 1 around the edges and to 0 in the center. When an input is convolved with this filter, the output activity is high for areas containing 5×5 rectangles, and low for other regions.

Filters in a CNN operate through the full depth of the input representation. In case of an RGB image, the input is represented as three stacked 2D arrays: one array for every color channel. The kernels thus operate on all three layers, taking every color into account. The convolution operations, however, are still two-dimensional of nature: convolutions are applied to independent two-dimensional slices (e.g. the red-colored channel of a kernel acts on the red input channel).

The number of filters in a convolutional layer determines the depth of the output volume of that particular layer. Let us consider an arbitrary activation on position (x, y, z) in an output volume of a convolutional layer. x and y tell us something about the spatial location where the filter was applied to the input. The height of z however, tells us the *filter number* or *depth slice*. For every layer, the number of filters, or *depth*, can be set manually: it functions as a hyperparameter for the CNN.

There are a couple of other hyperparameters which can be set for every layer. The *stride* of a kernel determines how much a filter slides when it is applied repeatedly. It determines the spacing between applied filters. With a stride of 1, a filter slides 1 'pixel' (when dealing with an input image real image pixels are meant; in higher layers the term 'pixel' refers to 'convolutional pixels') to the side after applying the filter. When the stride is lower than the dimension of the kernels, there is a certain overlap between the filters. Another hyperparameter is *zero-padding*. When the dimensions of the output volume need to be controlled, one could add zeros to the sides of the input volume.

A single convolutional layer is not very effective in detecting higher-order patterns that are suitable for image classification. The strength of CNNs lays in stacking convolutional layers. The first layers will detect primary image features, such as edges. The latter layers, which take the output of the former layers as their input, will be able to detect higher-order features. In other words, the latter layers will detect patterns of patterns. When using kernels that are large enough and a sufficient amount of convolutional layers, the higher-order patterns can be very global when compared to the input image.

Rectified Linear Units

In order for a CNN to learn a non-linear function, nonlinear activation functions have to be added to the net. Traditional neural networks often use a hyperbolic tangent $f(x) = tanh(x)$ or a logistic sigmoid function $f(x) = (1 + e^{-x})^{-1}$ for added nonlinearities (see section 3.2).

Most modern, deep CNNs, however, use Rectified Linear Units (ReLUs) to implement nonlinearities. A ReLU outputs its input if it is above zero, and zero otherwise:

$$f(x) = max(0, x) \tag{3.13}$$

One of the most important advantages or ReLUs is that it has been shown that neural networks with ReLUs tend to converge faster than networks with traditional activation functions [30].

Pooling

Pooling layers effectively subsample an input volume. Pooling layers shift a small filter, often 2x2, over an input volume, every time selecting the maximum value of the 4 numbers. Other forms of pooling exist, such as average and fractional pooling. In this thesis both max pooling and average pooling were used for the GoogLeNet architecture, and only max pooling for a self-constructed 3-layered convolutional neural network.

$$\begin{array}{|c|c|} \hline x_1 & x_2 \\ \hline x_3 & x_4 \\ \hline \end{array} \Rightarrow \text{MAX}(x_1, x_2, x_3, x_4)$$

The hyperparameter *stride* functions the same as for kernels: it determines how much the pooling filter is shifted every time it is applied. For pooling, the stride is often equal to the filter dimension (e.g. a stride of 2 for 2×2 filters), although there are exceptions such as 3x3 filters with stride 2 (e.g. AlexNet [24]).

Besides reducing the amount of computation in a network, pooling layers also prevent overfitting and therefore aid generalization. Because for every 2x2 region the maximum value is selected, a kind of translation invariance is introduced. For example, when an object is moved one pixel up on an input image, the pooling operation ensures that the final outcome is still similar to the non-translated version of the image.

Fully-connected layers

In a typical CNN, multiple convolutional and pooling layers are followed by one or more fully-connected layers. These fully-connected layers take the output of the last convolutional layer and yield output activities for every single class.

Softmax regression

The output of the last fully-connected layers is k-dimensional, where k represents the number of classes in the training data. As a final step, the output activations have to be transformed to k numbers which reflect the probability that a certain input belongs to a certain class. These probabilities will then be used for calculating the loss or error for training and test examples. For the transformation to class-conditional probabilities, a softmax function is often used. Softmax regression is a form of logistic regression and expands it to a multi-class scenario:

$$h_\theta(x^{(i)}) = \begin{bmatrix} p(y^{(i)} = 1|x^{(i)};\theta) \\ p(y^{(i)} = 2|x^{(i)};\theta) \\ \vdots \\ p(y^{(i)} = k|x^{(i)};\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{k} e^{\theta_j^T x^{(i)}}} \begin{bmatrix} e^{\theta_1^T x^{(i)}} \\ e^{\theta_2^T x^{(i)}} \\ \vdots \\ e^{\theta_k^T x^{(i)}} \end{bmatrix} \tag{3.14}$$

where $x^{(i)}$ is a presented input, $p(y^{(i)} = n|x^{(i)};\theta)$ the probability that the model output $y^{(i)} = n$ given $x^{(i)}$ and all model parameters θ. $\theta_n^T x^{(i)}$ is the intermediate output for class n by passing input $x^{(i)}$ through the for class n related parameters in the model. The softmax function is then computed over all output activations.

3.4.2 Training a deep convolutional neural network

The most common way to train a deep convolutional neural network is by applying variants of gradient descent. The solver method used in this research is Nesterov's accelerated gradient, which uses gradient descent as a basis but also includes momentum in its gradient computation:

$$V_{t+1} = \mu V_t - \eta \nabla L(W_t + \mu V_t) \tag{3.15}$$

$$W_{t+1} = W_t + V_{t+1} \tag{3.16}$$

where μ is the momentum value, η the learning rate and ∇L the gradient.

Deep neural networks are often not trained in an online fashion where the input matrices are presented sequentially. Instead, the input matrices are presented in mini-batches. For every mini-batch, containing a pre-defined number of examples (typical values are 8-64, depending on the computational power of the available GPUs), dot products with the model's parameters are computed for the full mini-batch to generate the network's output. Usually, the larger the size of the mini-batch, the better the generalization, as the weight adaptations of the model depend less on single examples.

Another characteristic of deep neural networks is that, when dealing with a lot of model parameters, the models tend to overfit quite easily when not enough training examples are presented. To overcome this issue, forms of data augmentation are used to artificially enlarge the training data (e.g. [24]). Methods include extracting patches from the original images and altering color channel intensities.

3.5 State of the art

The Imagenet Large Scale Visual Recognition Challenge (ILSVRC) is a yearly recurring competition for object detection and object localization algorithms [9]. The Imagenet challenge consists of multiple sub-challenges, of which the image classification task has probably received most attention. For the image classification task, the classes (of a total of 1000 classes) of objects depicted on images have to be predicted by an image classification algorithm. The decisive performance measure is the top-5 error rate which has to be as low as possible. A top-5 error occurs when the actual class of an image is not among the top-5 predicted classes by the algorithm.

During the last couple of years, the image classification challenge has been dominated by deep convolutional neural networks [24, 42, 16]. Even among convolutional neural networks the accuracy has drastically improved over the past years. Only from 2012 to 2015 the top-5 error has decreased from 15.3 (AlexNet) to 4.49 (ResNet). The increase in accuracy is partially an effect from new types of deep convolutional architectures, but can be ascribed to the availability of more computational power as well.

The next sections describe three ILSVRC winners from the past years: AlexNet (2012), GoogLeNet (2014) and ResNet (2015). These convolutional neural networks all have a very different structure, which makes it interesting to list all of them and not only the best performing one.

3.5.1 AlexNet

AlexNet is a deep convolutional neural network which was entered in the ILSVRC2012 competition by Alex Krizhevsky et al. [24]. AlexNet achieved top-1 and top-5 error rates of 40.7 and 18.2 when using a single CNN, and error rates of 36.7 and 15.3 when averaging predictions of 7 similar CNNs.

The overall structure of AlexNet is depicted in figure 3.2. The first five layers of AlexNet are convolutional of nature. Convolutions with 96 kernels of size $11 \times 11 \times 3$ and stride 4 act on the 3-dimensional input data. The second convolutional layer has 256 kernels of size $5 \times 5 \times 48$ which act on the normalized and max-pooled output of the first convolutional layer. Note that the depth of the kernels in the second convolutional layer is half of the number of kernels in the previous layer: this is due to parallel processing on 2 GPUs. The third convolutional layer contains 384 kernels of size $3 \times 3 \times 256$ which are connected to the normalized, max-pooled output of the second convolutional layer. The last two convolutional layers also use 3×3

convolutions: 384 kernels of size $3 \times 3 \times 192$ and 256 kernels of size $3 \times 3 \times 192$ for the fourth and fifth layer, respectively.

The five convolutional layers are followed by two fully-connected layers. These layers both contain 4096 neurons. The output of the last fully-connected layer is used for 1000-way softmax classification. Rectified Linear Units (ReLUs) were used as the activation function throughout the network, for fast computation and convergence.

AlexNet is a large CNN with around 60 million parameters. Krizhevsky et al. use several methods to improve the training process and reduce overfitting. Dropout [39, 40] (see section 3.3.3) was used in the fully-connected layers to prevent the network from overfitting. Two forms of data augmentation are used with the same purpose in mind. From the original ImageNet data, which contained 256×256 images, patches of 224×224 were extracted as well as mirrored versions, effectively increasing the amount of data by a factor 2048. These patches were used during training[1]. The other form of data augmentation encompassed altering the RGB channels in training images by adding a small quantity to every pixel related to the principal components of the image.

A small amount of weight decay (0.0005) and momentum (0.9) was used to facilitate learning.

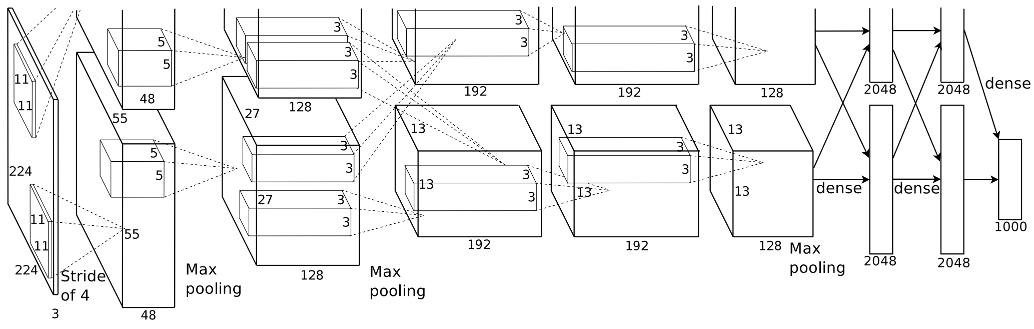


Fig. 3.2.: The full AlexNet architecture. Parallel computations are done on two GPUs, which is illustrated in the above figure by two separate pathways. Image copied from [24].

3.5.2 GoogLeNet

GoogLeNet [42], the winning convolutional neural network in the ILSVRC2014 classification challenge, is very different compared to the previously described AlexNet. Where AlexNet used relatively few convolution kernels which acted on

[1]During testing, five 224×224 patches (and mirrored versions) were extracted from an original 256×256 image: the four corner patches and a center patch. The final classification was then the result of averaging softmax predictions of ten distinct patches.

big volumes of data, GoogLeNet introduced so-called Inception modules, where convolutions with differently sized kernels are applied in parallel. The outputs of the multiple convolutional layers within a module were concatenated and passed to the next layers. Figure 3.3 shows an illustration of a single Inception module. In the full CNN, Inception modules were stacked on top of each other, where the output of the previous module functioned as the input for the next.

Deep convolutional neural networks have the undesired property that the volumes of data, due to repeated convolutions, quickly become too large to be handled by current computer hardware. Some networks attempt to tackle this issue by using subsampling methods such as average or maximum pooling. In GoogLeNet, every time the computational requirements would increase too much to be handled by the hardware, the dimension of convolutional volumes is reduced. This is achieved by using max pooling or average pooling and by applying 1×1 convolutions. This is clearly visible in figure 3.3: before 3×3 and 5×5 convolutions, the input is convolved with small 1×1 kernels. 1×1 filters can be used to reduce the dimension of convolutional volumes by using less filters than the depth of the input volume: the depth of the convolutional volumes is reduced.

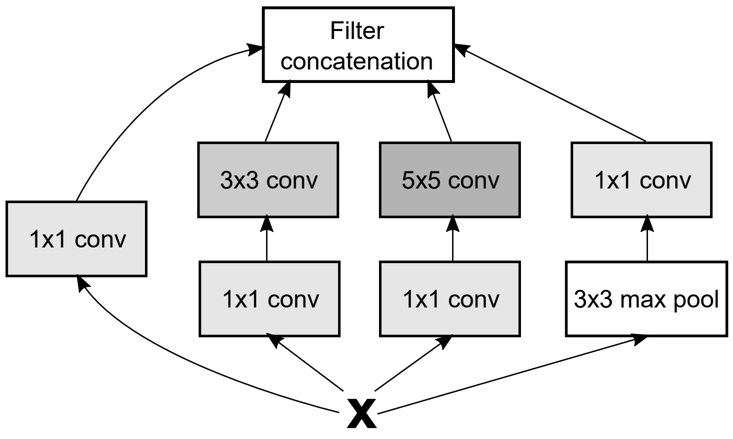


Fig. 3.3.: A single Inception module. Image based on [42].

Because GoogLeNet is a very deep network with 22 layers with parameters (excluding pooling layers which do not have parameters/weights), it can be hard to correctly adapt the weights using backpropagation. There is a problem of *vanishing gradients*: the error vanishes when it is propagated back into the network, leading to insufficient weight changes in the neurons near the input [3]. GoogLeNet deals with this problem by adding two auxiliary classifiers to the network, which are connected to intermediate layers. The output of these layers was taken into account for backpropagation during training: the error of the auxiliary classifiers was weighted with a factor 0.3 (opposed to 1.0 for the final, ‘third’ output). In this way, the error did not vanish as much as it would had there only been one output, as the intermediate classifiers were closer to the input than the final classifier. The auxiliary classifiers were not taken into account during test and validation time.

The network was trained on the Imagenet dataset by using a momentum of 0.9 and by decreasing the learning rate by 4% every 8 epochs. Dropout was used only in the fully-connected layers, with a value of 0.7 for the branches used for intermediate classification and with a value of 0.4 in the main classification branch. The designers of GoogLeNet [42] do not give any further details on the training process, and mention that it is hard to give a definitive guidance to the most effective way to train the network.

For testing, seven independently trained versions of the same GoogLeNet model were used. These models were used for ensemble prediction. The only differences between the training processes of these models was in sampling methodologies and the randomized input image order. Aggressive cropping was applied to the test data, leading to 144 crops per image. The softmax probabilities were then averaged over multiple crops for all individually trained networks, leading to a final classification.

The performance of GoogLeNet in terms of top-5 error was very good: a top-5 error of 6.67% was achieved, which is significantly better than the error rate of AlexNet (15.3%) and ILSVRC2013 winner Clarifai (11.2%) [44].

The full layout of GoogLeNet can be found in appendix A.

3.5.3 ResNet

Convolutional neural networks with a very high number of layers have the potential to learn more complex functions than networks which are shallower. Deeper networks do, however, not always perform better in terms of reducing the training error [15]. This is called the *degradation problem*. Note that this can not be ascribed to overfitting, because then the test error would be higher and not the training error. The winners of the ILSVRC2015 challenge, MSRA, think that the phenomenon occurs due to an inability of the deeper network to learn identity mappings when necessary [16]. The additional layers of the deeper network fail to map the identity function (note that when all additional layers map the identity function, performance would be identical to the more shallow network).

MSRA introduced a deep residual framework that deals with this issue by letting one or more successive convolutional layers learn residual functions rather than full mappings. Let $H(\mathbf{x})$ be the desired mapping of a few stacked convolutional layers. Normally, this set of layers would learn the direct mapping $F(\mathbf{x}) = H(\mathbf{x})$.

Instead, the input $\mathbf{x}$ is not only passed to the convolutional layers, but also passed through the identity function after which it is added to the convolutional output $F(\mathbf{x})$: $F(\mathbf{x}) + \mathbf{x}$. Instead of the direct mapping $H(\mathbf{x})$, now the residual mapping $F(\mathbf{x}) := H(\mathbf{x}) - \mathbf{x}$ is learned. An illustration of a building block for deep residual learning can be found in figure 3.4.

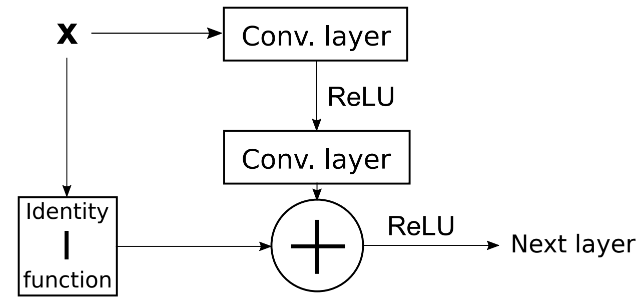


Fig. 3.4.: Residual learning building block. Image based on [16].

The best performing full residual network consisted of building blocks which are a bit more complex than depicted in figure 3.4. Instead of two-layer blocks of 3×3 convolutions, three convolutional layers were present within a block. The first convolutional layer, with 64 kernels of dimensions 1×1, was presented an input with depth 256. After ReLUs were applied to the output of the first layer, a second convolutional layer (3×3, 64 kernels) processed the data. ReLUs were applied to the intermediate data volume, followed by the third and last convolutional layer (1×1, 256 kernels). The convolved result was added to the input, after which the data was passed through another ReLU.

The best accuracy on the Imagenet dataset was reported with a network consisting of a total of 152 convolutional layers. The full network consisted solely of residual learning building blocks, apart from initial 7×7 convolutions, and two pooling operations at the beginning and at the end of the net. With some momentum and weight decay but without dropout, training went fluently and the degradation problem that deep networks often show was not present. The final top-5 accuracy of a 152-layered ResNet on the Imagenet validation set was as low as 4.49: significantly better than previous year entries in the ILSVRC competition.

4 Dataset definitions and exploration

This chapter is about the dataset that was used for the experiments. Pre-processing methods are described, important definitions are given and the dataset is explored with visualization methods.

4.1 Dataset contents

The dataset consists of two-dimensional position data of a selection of matches played by a German Bundesliga team during the 2008/2009 and 2009/2010 season. The positions for every player on the pitch were captured by the Amisco® multiple-camera system. The Amisco® system consists of multiple cameras placed around the stadium and tracks all moving players on the soccer field at a sampling frequency of 25 Hz [1]. Computer vision techniques are used to track objects and estimate their positions.

Because the ball position was originally not tracked by the system, it was manually added to the data. Therefore, the position of the ball is not as precise as the player movement. When the ball was passed or shot, only its start position and end position were marked. As a result, the ball always moved in straight lines, even in cases of curved shots or passes. When a player had ball possession and dribbled with the ball, the x- and y-coordinates from the player were copied and used as ball position.

As a follow-up step the data were imported into soccer analysis software developed by Inmotio [18]. Consecutively, the raw export function of the Inmotio software was used to export match data at a downsampled frequency of 10 Hz.

4.2 Preprocessing

4.2.1 Data filtering

Three matches were deleted from the original dataset. The position data of these matches were incomplete: in all three cases data from the opposing team's players were missing. The actual dataset therefore included a total number of 29 full-length matches. Only one of these matches was an away game.

4.2.2 Data cleansing

When dealing with continuous data measured in a dynamic environment, it is no surprise that parts of the data contain errors. In the used dataset, there existed time intervals for which the coordinates of a certain player were not measured correctly. In the dataset, such measurements were indicated by the coordinates adopting very high values (in the order of $1.0 \cdot 10^5$ meters).

When inspecting the data, it was apparent that there were two main reasons which caused the errors. The first reason is because of players being not between the lines of the soccer field. For this category a distinction can be made between players which were not between the lines for only a limited amount of time, and players which had left the pitch indefinitely. For the first category, coordinates were linearly interpolated for the interval that contained erroneous data. For the second category, erroneous data rows were deleted. Note that this did not negatively impact the continuity of the dataset: the players had left the soccer field indefinitely, and did not return to the pitch.

The second cause for erroneous data is the computer vision algorithm not being able to correctly capture the position of a player. This effect seemed to be present most during corner kicks, during which the players were standing very near to each other, causing tougher extraction of individual players. In these cases the player position was linearly interpolated as well.

Figure 4.1 shows the distribution of the duration of noisy intervals: time intervals during which the coordinates were not measured correctly for at least one player. The figure shows that the erroneous intervals do not make up a large part of the data. This indicates that linear interpolation would not affect the data too much in a negative way.

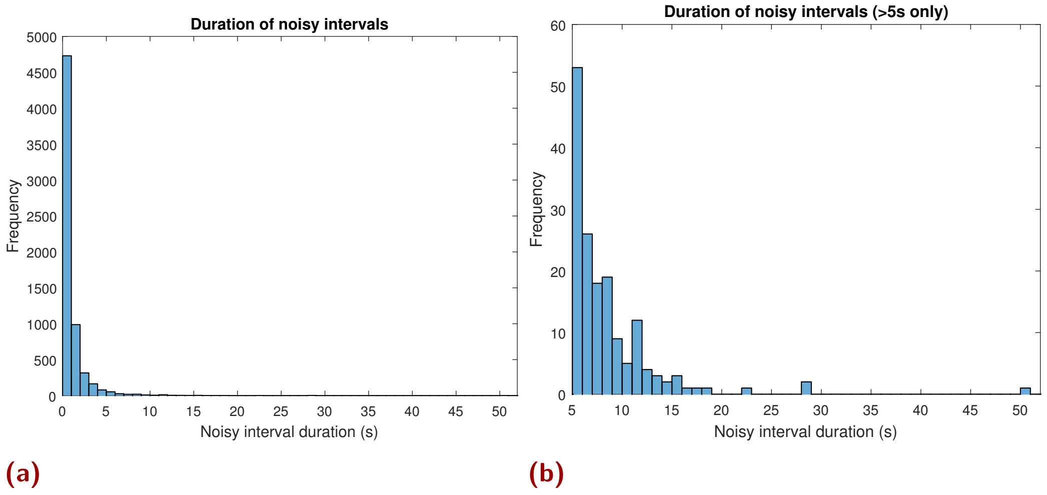


Fig. 4.1.: (a, b) Duration of noisy intervals.

4.2.3 Addition of derived variables and quantities

The raw dataset lacked information in certain aspects, as it did only contain a timestamp, two-dimensional coordinates, object velocity, object id, object name and shirt number for all players and ball. Missing essential parts of the data were added manually. Every player had to be assigned to a team, which was either the Bundesliga team or the opposing team. This was done manually. The playing direction of the teams was extracted by looking at the position of players at the beginning of the matches. When most players belonging to one of the teams were located on a particular side of the field, their playing direction was set to the other side, and vice versa. Finally, a variable representing the direction of movement for players and ball was added. Coordinates of two adjacent samples were taken into account for determining the direction of movement.

4.3 Definition: goal-scoring opportunities

There are many ways to define a goal-scoring opportunity in soccer. One could state that possession of the ball in a certain area of the soccer field, for instance the penalty area, is a goal-scoring opportunity. Frequency statistics about soccer matches often include the total number of shots and the number of shots on goal, which could both also be considered goal-scoring opportunities.

Due to the two-dimensional nature of the data, it is impossible to distinguish between shots which were on goal and shots which went over the bar. Taking these limitations into account, goal-scoring opportunities have been defined as shots which (almost) crossed the end line near the goal. A shot which is a little wide would still be classified as a goal-scoring opportunity, as will a shot which crosses the goal on the upper side. A movement of the ball was considered a shot when:

1. the ball had moved in a more or less straight line towards the goal;
2. the velocity of the ball was above a certain threshold all the time;
3. before the velocity of the ball passed this threshold, a player belonging to the attacking team was near the ball.

Algorithm 4.3 shows pseudocode of the algorithm which was used for finding goal-scoring opportunities. Start index, end index and the side of the field where the goal-scoring opportunity occurred were returned for each opportunity. The algorithm relies on several constants that affect the threshold settings for detecting goal-scoring opportunities, which are listed below. Between brackets are the actual values that were used for extracting opportunities.

min-velocity [set to: 20 km/h]
The minimum velocity for the shot. Velocity had to be above this threshold for a movement of the ball to be considered a shot.

max-p-distance [set to: 1.5 m]
Maximum distance from attacking player to ball at the start of a goal-scoring opportunity.

max-angle-change [set to: 20 degrees]
Maximum change of direction of the ball during a shot. In section 4.1 it was stated that, due to the later addition of the ball to the dataset, the ball always moved in a straight line when it was shot. This variable may seem unnecessary at first sight, but imagine the case of a shot which is touched by another player halfway through. It would not be appropriate to consider this as a single shot.

max-dist-to-goal [set to: 1.0 m]
When the ball was directed towards goal in the previous sample, but is not anymore in the current sample, this parameter determines the maximum distance from the ball to the end line for the shot to be considered a goal-scoring opportunity. By setting this threshold to a value higher than zero, shots which do not directly pass the end line can still be detected as opportunities. A shot stopped by the goal keeper before the goal line will then still be classified

as an opportunity. Another advantage of assigning a slightly positive value to the threshold is that it helps to tackle noisy measurements.

min-shot-time [set to: 0.5 s]
: Minimum duration for a shot.

opp-margin [set to: 5.0 m]
: Goal-scoring opportunity margin on both sides of goals. This parameter determines how far a shot can go wide while still being classified as a goal-scoring opportunity. Note: for clarity, this parameter is not listed in algorithm 4.3.

Algorithm 4.3 Algorithm to find goal-scoring opportunities

```
1:  function FIND–OPPORTUNITIES($data$)
2:      $opportunities \leftarrow$ empty list
3:      for each $sample \in data$ do
4:          if $sample.ball\text{-}velocity < min\text{-}velocity$ then
5:              continue
6:          if $sample.ball\text{-}direction$ is towards one of the goals then
7:              $near \leftarrow$ list of players whose distance to ball $\leq max\text{-}p\text{-}distance$
8:              $team\text{-}att \leftarrow$ team attacking the goal to which the ball is directed
9:              if there is a $player \in near$ belonging to $team\text{-}att$ then
10:                 for $end\text{-}sample \in [sample + 1, sample + 2 \ldots sample + n]$ do
11:                     $dir\text{-}diff \leftarrow$ diff in direction between $end\text{-}sample$ and $end\text{-}sample - 1$
12:                     if $dir\text{-}diff > max\text{-}angle\text{-}change$ then
13:                         break
14:                     if ball at $end\text{-}sample$ is not anymore directed towards goal then
15:                         if distance at $end\text{-}sample$ to goal $\leq max\text{-}dist\text{-}to\text{-}goal$ then
16:                             if time between $sample$ and $end\text{-}sample \geq min\text{-}shot\text{-}time$ then
17:                                 Add $(sample - 1, end\text{-}sample)$ to $opportunities$
18:                                 Outer for loop: jump to sample $end\text{-}sample + 1$
19:                         else
20:                             break
21:                     else
22:                         break
23:     return $opportunities$
```

4.4 Definition: ball possession

The ball possession was assigned to the team whose player was closest to the ball. Some extra parameters were added to avoid the algorithm from switching ball possession when the ball passed a player closely, but did not undergo a change in direction or a significant decrease in velocity.

Algorithm 4.4 shows pseudocode of the algorithm used for ball possession detection. For every sample in the data, an integer indicating possession for the German

Bundesliga team (1), the opponent (2) or none (0) was returned. The parameters which could be set are as follows:

max-p-distance [set to: 1.5 m]
: The nearest player at a specific time had to be closer than `max-p-distance` to the ball to have ball possession.

max-ball-velocity [set to: 10 km/h]
: When a player was close enough to the ball to be considered for ball possession, the ball velocity had to be under `max-ball-velocity` for actually getting possession assigned. There is one exception, which involves the last constant.

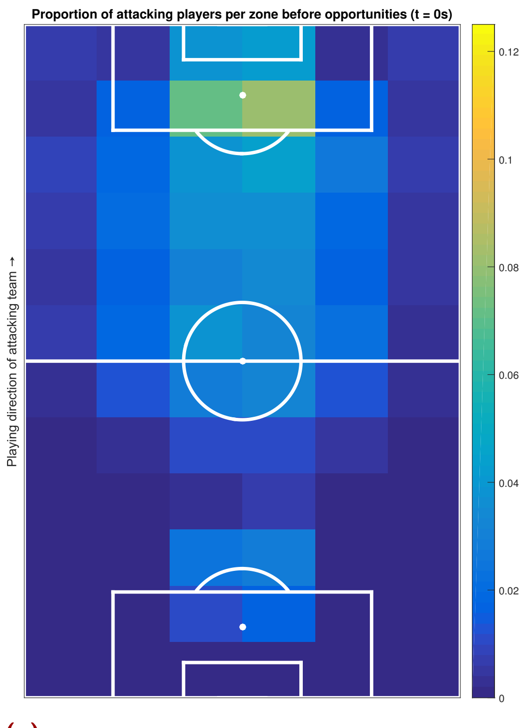
min-ang-chg [set to: 20 degrees]
: When the angular change of the ball was very high while it was close to a player, it was implied that the player had caused the change of direction.

Algorithm 4.4 Ball possession algorithm

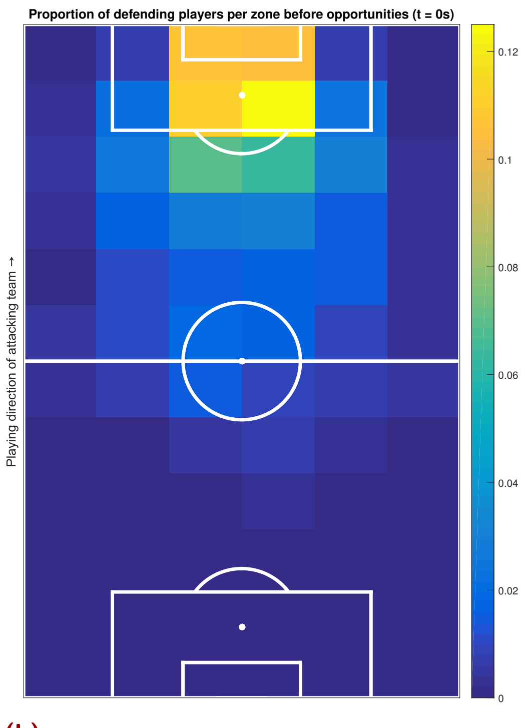
```
1:  function GET–BALL–POSSESSION($data$)
2:      $possession \leftarrow$ empty list
3:      Add 0 to $possession$
4:      for each match $sample \in data$ except the first do
5:          $nearest \leftarrow$ nearest player to ball at $sample$
6:          if $distance(nearest, ball)$ at $sample \leq max\text{-}p\text{-}distance$ then
7:              $ang\text{-}chg \leftarrow$ change of ball direction between $sample - 1$ and $sample$
8:              if $sample.ball\text{-}velocity \leq max\text{-}ball\text{-}velocity$ or $ang\text{-}chg \geq min\text{-}ang\text{-}chg$ then
9:                  $team \leftarrow$ team to which $nearest\text{-}player$ belongs
10:                 $possession(sample.idx) \leftarrow team$
11:             else
12:                 $possession(sample.idx) \leftarrow possession(sample.idx - 1)$
13:         else
14:             $possession(sample.idx) \leftarrow possession(sample.idx - 1)$
15:     return $possession$
```

4.5 Dataset exploration

This section is about dataset exploration. The main focus will be around the 349 goal-scoring opportunities that were found: what was the distribution from attacking and defending players' positions in the built-up to goal-scoring opportunities, how did ball possession fluctuate, how did both team centroids correlate, etcetera.



(a)



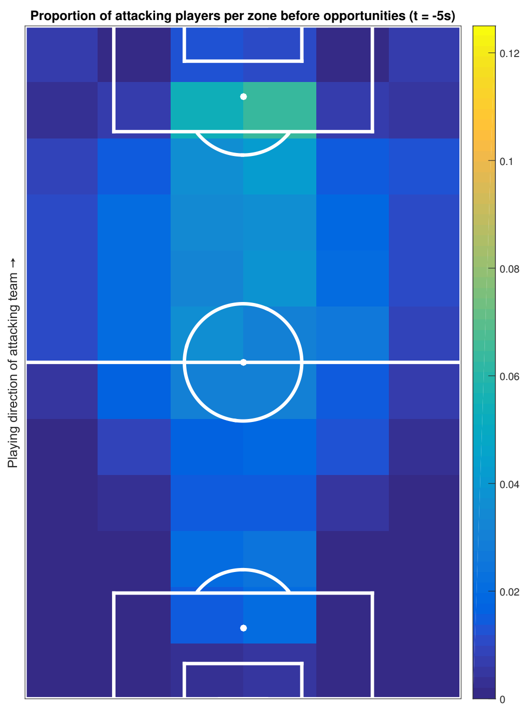
(b)

Fig. 4.2.: Average proportion of attacking and defending players per zone before goal-scoring opportunities. $t = 0$ indicates the exact moment of a goal-scoring opportunity: the instant the ball was shot.

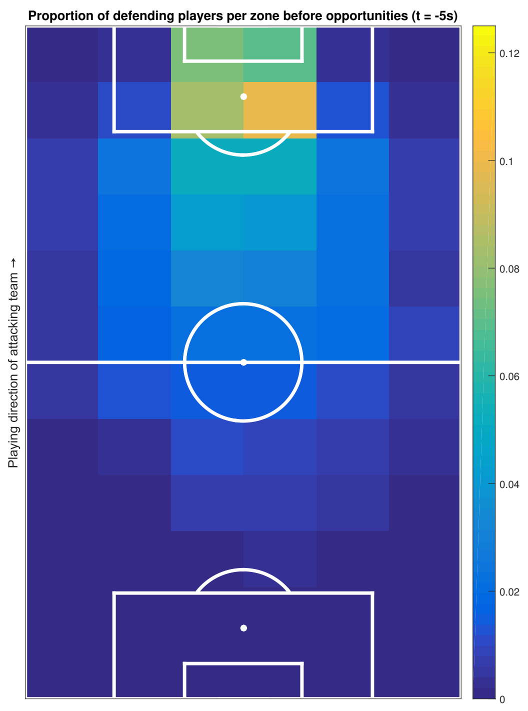
4.5.1 Player positions

Studying player positions before goal-scoring opportunities can give insights into the processes that led to the creation of the opportunities. Figure 4.2 shows the average division of players over the soccer field at the exact instant of a goal-scoring opportunity. Figure 4.2a shows the distribution of attacking players, figure 4.2b shows the distribution of defending players. The proportions for each figure add up to 1. The attacking players seemed to be rather equally divided over the attacking half of the field, with a peak presence in the zones located around the penalty spot. The defending players were predominantly located around their team's penalty area. Overall, the positions of the defending team's players were much more concentrated around one spot than the attacking players.

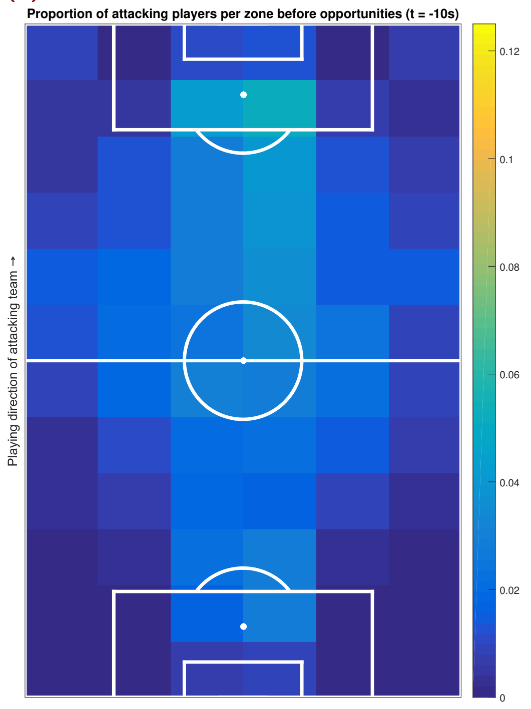
Note that the goal of this study is to predict goal-scoring opportunities. It might be more interesting to zoom in on player location distributions moments *before* goal-scoring opportunities were created. Figure 4.3 shows average divisions of attacking and defending players 5 seconds (figure 4.3a and 4.3b) and 10 seconds (figure 4.3c and 4.3d) before opportunities. The defending players were already predominantly located around their penalty area, although a bit more divided over the field. Interestingly, the attacking players were almost equally divided around the entire field 10 seconds before goal-scoring opportunities. This could indicate that regarding the use of player positions for goal-scoring opportunity predictions, only a short time interval before opportunities is of importance.



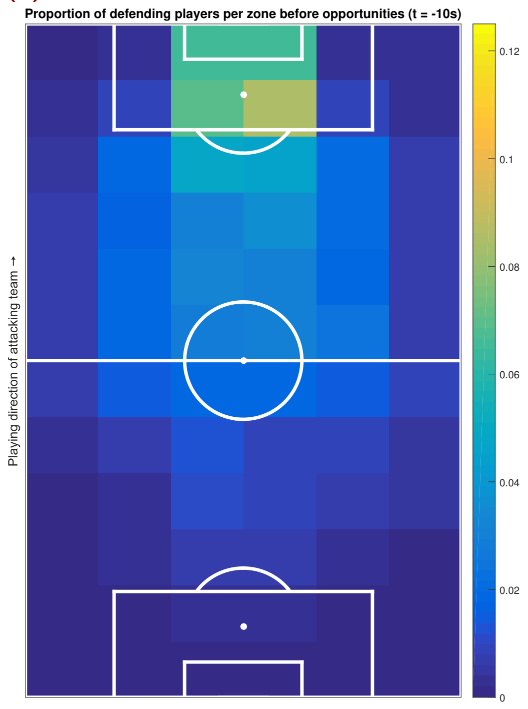
(a)



(b)



(c)



(d)

Fig. 4.3.: Average proportion of attacking and defending players per zone 5 and 10 seconds before goal-scoring opportunities.

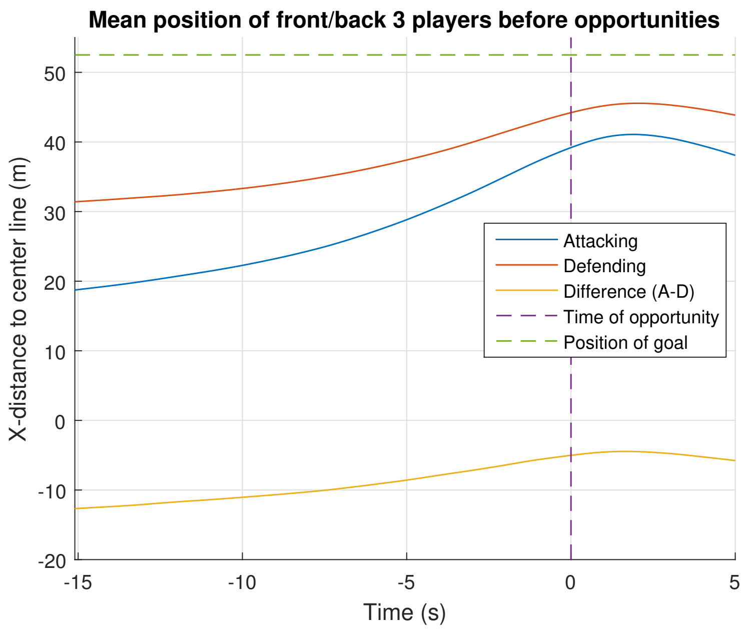


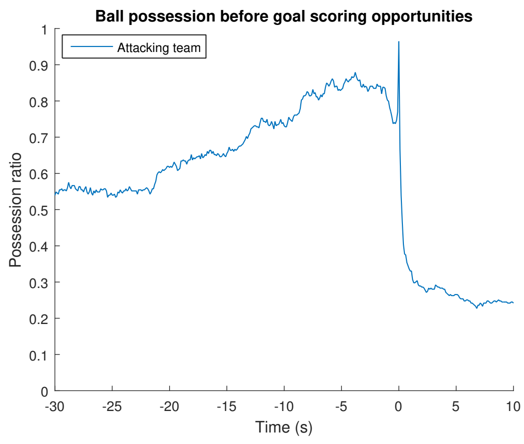
Fig. 4.4.: Mean distance to half-way line of front (attacking team)/back (defending team) 3 players during goal-scoring opportunities.

4.5.2 Team centroids

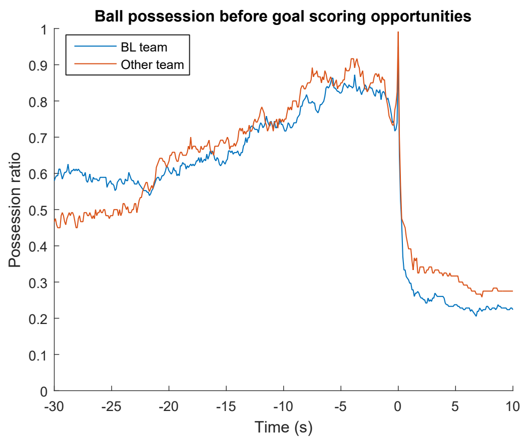
Frencken et al. found that the centroid position of a soccer team can provide valuable information about the 'coordinated flow of attacking and defending' in 5 versus 5 soccer matches [12]. Among other findings, they found that team centroids crossed for 10 out of 19 scored goals in the length direction of the field.

Figure 4.4 shows centroid positions before goal-scoring opportunities for the forward 3 (attacking team) and back 3 (defending team) players of each team. Goalkeepers are excluded. Only the centroid coordinates from the axis over the length of the field are plotted. The difference between the attacking and defending team is plotted as well. The attacking team's centroid seemed to approach the defending team's centroid during the built-up of goal-scoring opportunities. Over a time period of 15 seconds before opportunities, the difference in distance between the centroids was decreased by approximately 8 meters. A crossing of centroids as described by Frencken et. al., however, was not present.

The same effect seemed to be present for centroids of the back and front 4/5 players. For 7 and 10 players the effect disappeared.



(a)



(b)

Fig. 4.5.: Ratio of ball possession before goal-scoring opportunities for (a) all attacking teams and (b) the Bundesliga team and other teams, when attacking.

4.5.3 Ball possession

Figure 4.5a shows the average ball possession for the attacking team during goal-scoring opportunities. $t = 0$ marks the instant the ball was shot. The y-axis (possession ratio) reflects the number of times the attacking team had ball possession divided by the number of times the defending team possessed the ball. The ball possession was determined as described in section 4.4. When inspecting figure 4.5a, there was a clear increase in ball possession starting at 20 seconds before goal-scoring opportunities.

Figure 4.5b shows ball possession for attacking teams as well, but this time a distinction is made between opportunities created by the same Bundesliga team (blue line) and opposing teams (red line). The possession ratio seemed to be alike for both the same Bundesliga team and opposing teams, except for the beginning.

The peak in the graphs towards a possession ratio of 1 at $t = 0s$ was an effect of the definitions of ball possession (section 4.4) and goal-scoring opportunities (section 4.3) that were followed. Goal-scoring opportunities were only able to occur when a player of the attacking team had ball possession at the time of shooting the ball. The drop in average ball possession just before the opportunities could possibly be ascribed to opportunities created by capturing the ball near the opponent team's goal, quickly resulting in a shot on goal.

4.5.4 Backline crossings

Figure 4.6 shows the distribution of y-coordinates where the ball crossed the backline. The ball mostly crossed the backline near the goal. Note that the backline crossings near the goal are considered goal-scoring opportunities, with a certain margin.

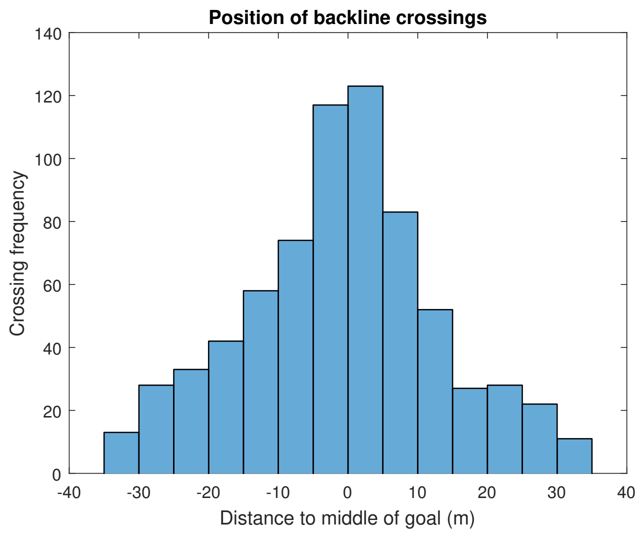


Fig. 4.6.: Distance from shots crossing the backline to the center of goal. A negative distance reflects a shot crossing the backline on the left side of the goal.

4.5.5 Passing

Figure 4.7 shows the start and destination zones of passes before goal-scoring opportunities. Most passes seemed to be short passes which stayed within their zones. The center zones were best represented. The right side of the field (zones 12, 15 and 18) seemed to be equally important as the left side (zones 10, 13 and 16) for opportunity creation. This is not a surprise at the highest level of soccer: when a player is not sufficiently efficient, he is often quickly replaced.

A simple algorithm was used to extract passes from the data. A pass started when the ball was near a player during the previous samples but not anymore during the current sample. When ball velocity dropped afterwards or the ball changed direction, the ending of the pass was marked. Passes with a duration under 0.2 seconds were discarded.

The direction of the passes can be an interesting variable for analyzing team tactics. Figure 4.8a shows the ratio of passes towards goal compared to all passes that originated from that zone. It should be noted that passes did not have to reach its intended destination. This would have been too difficult with the current dataset, especially due to the manually added ball positions. A shot on goal would therefore still be considered a pass. This is one of the main reasons why the passes towards goal ratio was very high for the zones in the penalty area.

Figure 4.8b shows the ratio of passes with a forward direction compared to all passes sent from that zone.

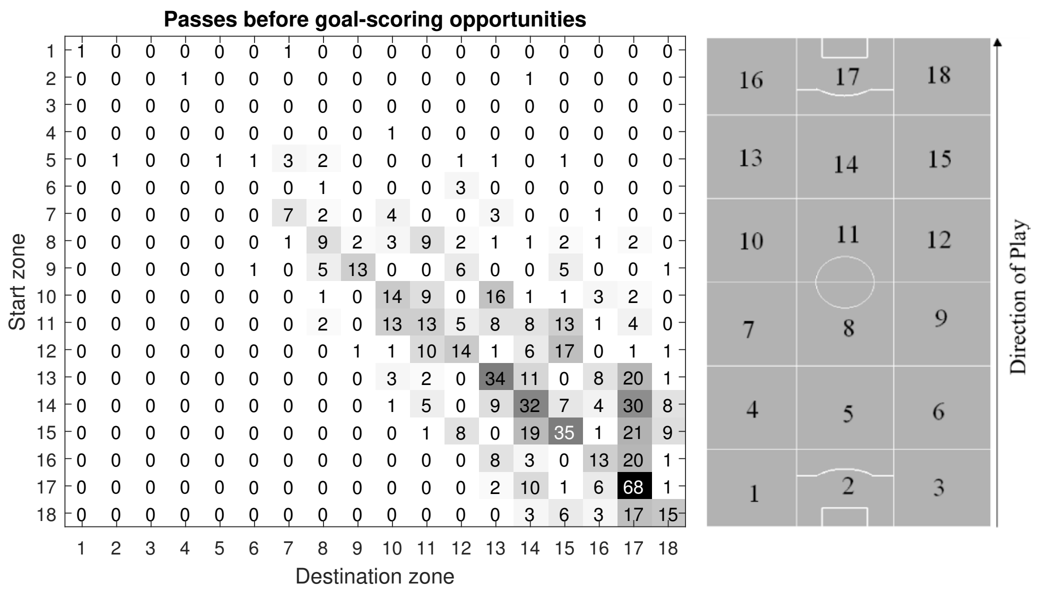


Fig. 4.7.: Start and destination zones for passes during a 10-second interval before goal-scoring opportunities. Right image from [4].

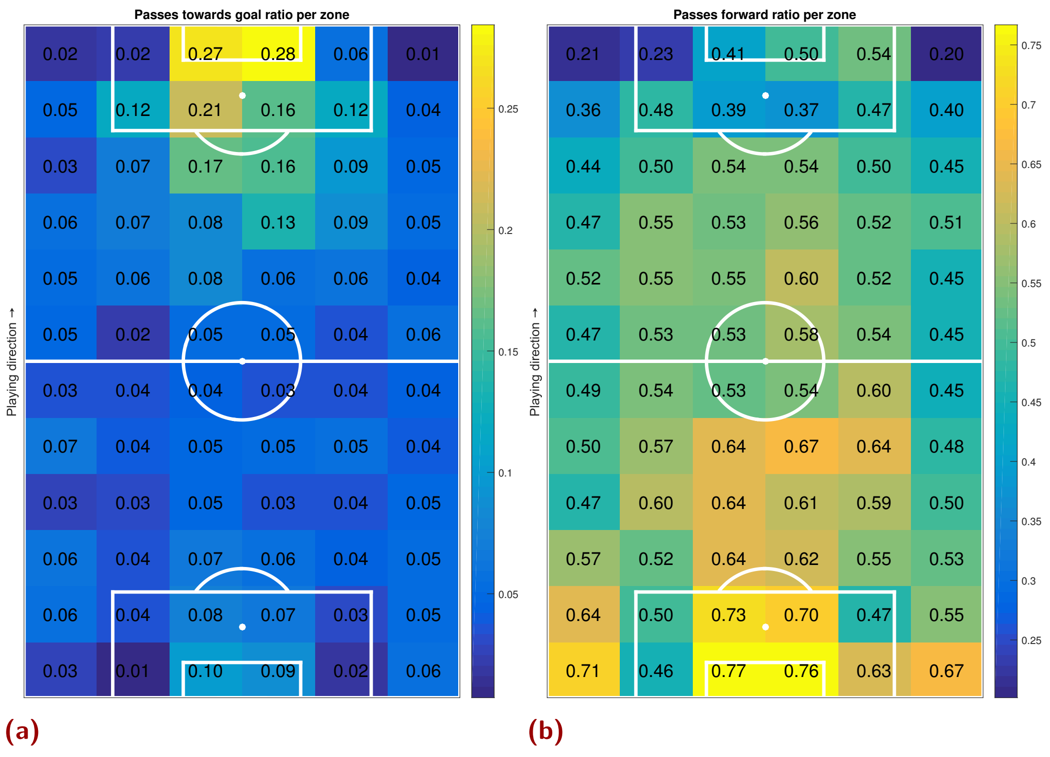


Fig. 4.8.: Ratio of (a) passes towards goal and (b) forward passes per zone.

5 Methods, experiments and results

5.1 Classification problem

The underlying problem of all experiments in this thesis is one of classification. Given a specific combination of player positions and ball position, what is the probability that a goal-scoring opportunity will emerge within reasonable amount of time? When this estimated probability is above 0.5, the specific instance will be classified as a goal-scoring opportunity. Otherwise, the instance will be classified as the non-occurence of a goal-scoring opportunity.

It can be quite a challenge to classify a specific situation on the pitch into one of the above categories. This challenge becomes even harder when the window around goal-scoring opportunities is enlarged: when not only the exact instant of the shot on goal is considered a goal-scoring opportunity, but also the 5 or 10 seconds before the event. While more of a challenge, a bigger window is beneficial for the predictive power of the classifier. It enables more practical uses as well: with an extended window, one could possibly use the classifier for predicting goal-scoring opportunities in the next couple of seconds. In this thesis, a goal-scoring opportunity window of 10 seconds is used. Explicitly, this means that the 10 seconds before goal-scoring opportunities are considered goal-scoring opportunities, while the remainder of the matches is not (for more information about goal-scoring opportunities, see section 4.3).

Convolutional neural networks are used to classify extracted samples from the position data. Convolutional neural networks are suited for this task because of their capability of extracting higher-order patterns in higher-dimensional data. As input to the machine learning algorithms, RGB color images are used. Images are suitable for visualizing player positions because the dataset consists of two-dimensional coordinates of the objects on the field: the height of the objects was not captured. The players and ball can therefore be represented by blobs on the images, such as rectangles. Different colors can be used to distinguish between teams. The next section describes the process from raw data to images in more detail.

5.2 Constructing images for learning

The following steps sketch the process from raw data to images in a dataset. Multiple datasets were created with different distributions for the train, validation and test sets within the dataset. Every dataset consisted of approximately 7000 positive examples and 7000 negative examples.

1. For every match out of 29 matches, goal-scoring opportunities were detected using the algorithm described in section 4.3.

2. From a total of 349 detected goal-scoring opportunities

    a) 45% were used for the training set;

    b) 10% were used for the validation set;

    c) 45% were used for the test set.

3. For every goal-scoring opportunity

    a) 11 images were constructed from the 10-second window before the opportunity with a spacing of 1 second ($t_{opp} - 10$, $t_{opp} - 9$, $\ldots$, t_{opp}) and added as positive examples to the appropriate set. Mirrored versions of the images along the length axis of the field were added to the dataset as well.

    b) 11 images were constructed from the 10-second window before semi-randomly selected timestamps marking negative examples. The negative examples were extracted from the same match in which the currently considered goal-scoring opportunity occurred, and the same team had to be attacking (further details in section 5.2.1). The images, including mirrored versions, were added to either the training, validation or test set.

5.2.1 Negative examples

The process of extracting non-occurrences of goal-scoring opportunities, or selecting negative examples, is not straightforward. There is a trade-off between the performance of the classifier and practical applicability. When the negative examples would have been selected entirely randomly, performance of the classifier would be best. The positive examples would differ a lot from the negative examples, making it easy for the classifier to distinguish between them.

The question one should ask is how such a classifier would perform in practice: can it be used to predict upcoming goal-scoring opportunities and is this prediction reliable? This would probably not be the case. There is a danger of the classifier extracting very simple differences between the positive and negative examples. The mean ball position of the randomly selected examples will probably be around the center of the field, while the mean position of the positive examples will be shifted more towards the penalty area. The classifier would probably perform best by learning a simple rule taking into account only ball position. Because the dataset is not correctly balanced, the classifier can learn simple rules like this and ‘get away with it’.

For the problems dealt with in this thesis, a more evenly balanced dataset has been constructed by making positive and negative examples as similar as possible, while still maintaining the main differences between the two classes.

We finally defined negative examples as follows: for every positive example, a negative example was added from the same match half. Negative examples were selected by looking at loss of ball possession from the team which was attacking in the positive example. Possession loss had to occur on the attacking side of the field. From all instances that met the above requirements, one was randomly selected and added to the dataset. In this way the two classes could still be considered opposites (loss of ball possession vs. a shot towards goal), but are not as different as the example sketched above.

Note that this problem only exists when the classes are not strictly defined. When objects from the real world have to be classified, the class definitions and border are much clearer and the problem is close to non-existent.

5.2.2 The images

Figure 5.1a shows how player and ball positions were displayed on 256×256 images. To facilitate learning, the same Bundesliga team was always playing from right to left on the images, and marked as red (players) and yellow squares (goalkeeper). The opposing team’s players’ positions were displayed as blue (players) and light blue (goalkeeper) squares. The size of the squares for players was 7×7 pixels, while the ball was of size 11×11 pixels. The ball was always printed on the background and was colored green. When two players’ positions did partly overlap, the colors of these players were mixed, resulting in a purple color for a red and blue player.

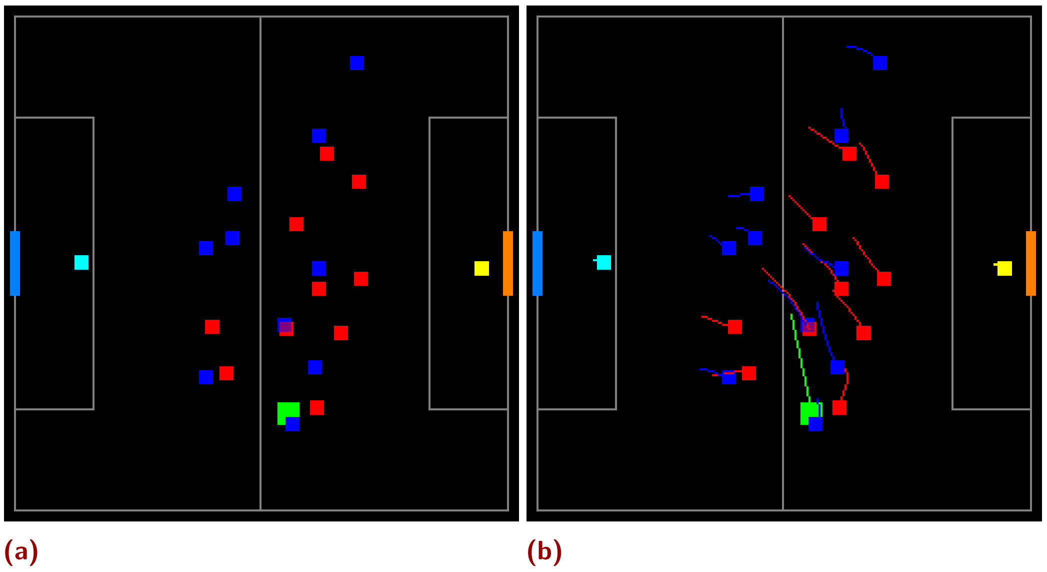


Fig. 5.1.: Example of images used as input for the convolutional neural networks. (a) shows an example of a default image, (b) an enhanced image.

Some experiments were done for varying sizes of squares, but this did not affect the performance of the classifiers much. Therefore the size for the players and ball was kept as small as possible, to allow maximum detail.

Apart from moving objects, the goals were colored as well. The most important lines of the field, such as the borders, middle line and penalty area were printed as thin gray lines. By adding lines to the images, certain formation patterns can become easier to detect. Ideally, one would not want to unnecessarily aid a learning algorithm too much, because it can cause the classifier to neglect more complex, deeper patterns. Because of the relative small amount of data, it was however necessary to add at least some lines to the images. Some tests were done with completely removing lines from the images and with a grid-like background pattern, but they were unsuccessful, probably due to an insufficient amount of data.

To illustrate how samples belonging to the different classes were transformed to images, figure 5.2 shows 4 randomly selected positive examples and 4 randomly selected negative examples. Enhanced versions of the images were used, which will be described in the next section. The top row images (a-d) belong to the positive, or goal-scoring opportunity, class. Within 10 seconds of the depicted scenarios, goal-scoring opportunities were created. The same applies to the bottom row images (e-h) which show negative examples. Within 10 seconds of the shown scenarios, ball possession was lost by the attacking team. Note that mirrored images along the length of the field are used in the dataset as well.

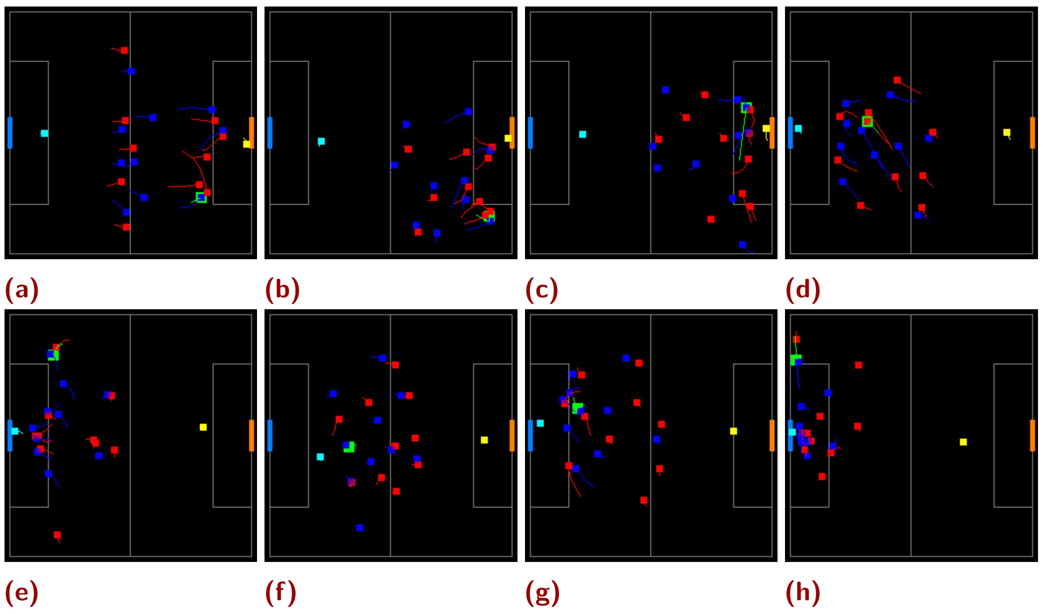


Fig. 5.2.: 8 enhanced images extracted from the intervals before different match events. The top row images (a-d) belong to the positive, *opportunity* class; the bottom row images (e-h) to the negative, *loss of ball possession* class.

5.2.3 Enhanced images

The default images were enhanced with information about the previous positions of objects. This was based on the idea that when there is more information available about how a specific situation arose, the easier it is for the algorithms to extract dynamics from the images.

Information about the past was added in the form of trails for the players and ball, which indicated what their positions were during the 2 seconds before the depicted moment. The trails were 1 pixel wide and colored the same as the accompanying object. Figure 5.1b shows an example of an enhanced image.

5.3 Experiments

Three experiments were carried out: GoogLeNet and a 3-layered convolutional neural network were trained on soccer images, while k-nearest neighbors was trained on ball positions. GoogLeNet is the most complex neural network featuring in our experiments. Although GoogLeNet (and the other networks described in section 3.5) achieved good performance on the Imagenet dataset, it should also be suitable for images constructed from soccer data. There might be too many layers because of the abstract data representation with relatively few features, but the intermediate outputs of GoogLeNet could be sufficient measures to handle this.

The convolutional neural networks in this thesis were implemented in Caffe: a deep learning framework which efficiently makes use of modern day's graphical processing units (GPUs) for convolutional calculations [20]. The implementation of GoogLeNet is distributed with Caffe[1]. This was one of the main reasons why GoogLeNet was used instead of even more recent, better performing convolutional nets.

The NVIDIA Deep Learning GPU Training System (DIGITS) was used as an interface to Caffe. DIGITS offers tools which make it easy to queue tasks and quickly offer insight into the training process [32]. It can be hard to get the hyperparameters right for deep neural networks. DIGITS can visualize training and validation errors for different timestamps during the learning process, making it easy to fine-tune the values of hyperparameters and show their direct effects on learning.

All experiments described in this section were conducted 10 times on differently constructed datasets. Every single dataset contained all of the detected goal-scoring opportunities, although they were randomly distributed along the train, validation and test sets. There was less overlap regarding the negative examples: these were randomly selected for every dataset using the process described in section 5.2.1.

5.3.1 Experiment 1: GoogLeNet

GoogLeNet was both trained starting from scratch and from a pre-trained model which was used for classification on the Imagenet dataset (which was distributed with Caffe). Because the Imagenet dataset contained 1000 classes and in this thesis we are dealing with binary classes, the last pre-trained layers from GoogLeNet could not be re-used.

Training and testing was done with both default datasets and enhanced datasets. This leads to a total of 4 experiments:

Experiment 1a GoogLeNet trained from scratch with default data;

Experiment 1b Pre-trained GoogLeNet, trained with default data;

Experiment 1c GoogLeNet trained from scratch with enhanced data;

Experiment 1d Pre-trained GoogLeNet, trained with enhanced data.

[1]https://github.com/BVLC/caffe/tree/master/models/bvlc_googlenet

The GoogLeNet models were trained for 8 epochs with a learning rate of 0.001 which was decreased by a factor 10 every 33% of the epochs (2.67 epochs). A momentum of 0.9 was used and a weight decay with $\lambda = 0.0005$. Nesterov's accelerated gradient was used as a solver for the network [31], which is a variant of stochastic gradient descent with the difference that the gradient is taken on the current weights with added momentum, as opposed to stochastic gradient descent which only takes the current weights into account for computing the gradient (see section 3.4.2). The mini-batch size for training was set to 8.

5.3.2 Experiment 2: 3-layered CNN

GoogLeNet may be too complex and deep for our soccer data, because of our abstract image representation. Another, self-constructed convolutional neural network was used to test this hypothesis. Its structure is straightforward: convolutions are computed sequentially on big volumes of intermediate data. Figure 5.3 shows the neural net's structure.

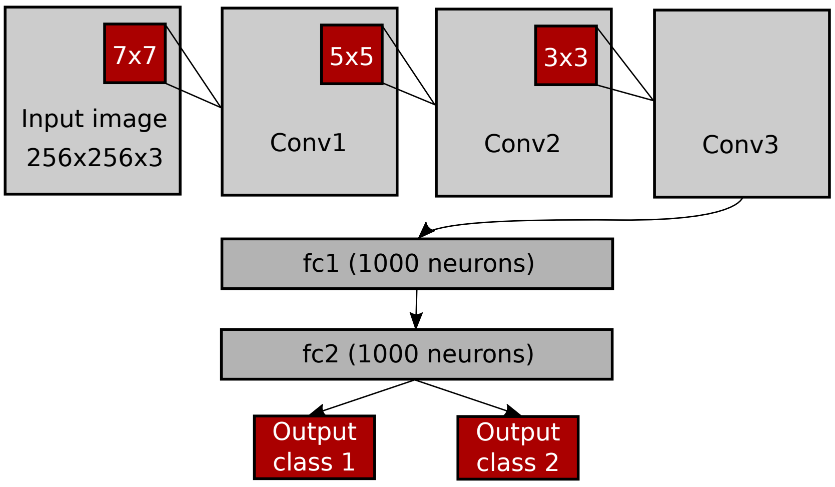


Fig. 5.3.: The 3-layered convolutional neural network used for experiment 2.

The first convolutional layer takes the input and convolves it with 48 kernels of size 7×7 with a stride of 2. The number of kernels in this layer was intentionally left low because of the very simple shapes that were used to construct the images in the datasets. The convolutional layer was followed by a ReLU unit and a max pooling layer with size 3×3 and stride 2 (similar to AlexNet [24]). The second convolutional layer consisted of 128 kernels of size 5×5, and was followed by ReLU units and a max pooling layer with the same hyper-parameters as described previously. The third and last convolutional layer housed 192 kernels of size 3×3 whose output was passed through a ReLU and max pooling layer.

Two fully-connected layers with 1000 neurons each formed the final part of the neural network. ReLUs were applied to the activations of every neuron. To aid generalization, dropout was added to both fully-connected layers as well, with a dropout rate of 0.5. The final 2 output neurons were passed through a softmax function to yield probabilities for both classes.

The 3-layered convolutional neural network, where the number '3' points to the number of *convolutional* layers in the net, was trained for 2 epochs on every dataset. The learning rate was set to 0.00005 and decreased with a factor 10 every 33% of the epochs (0.67 epochs). A momentum of 0.9 was used and a weight decay with $\lambda = 0.0005$. Nesterov's accelerated gradient was used for training the 3-layered convolutional neural network. The mini-batch size for training was set to 8.

For the smaller CNN, two experiments were conducted: one with the original data, and another with enhanced data.

Experiment 2a 3-layered CNN with default data;

Experiment 2b 3-layered CNN with enhanced data.

5.3.3 Experiment 3: KNN baseline

A final k-nearest neighbors experiment functioned as a baseline [8]. K-nearest neighbours was trained on *ball positions* of 50 percent of a dataset. For testing, the other half of the data was sequentially presented. To every testing example, the class of the majority of the $k = 25$ nearest training examples was assigned[2]. The euclidean distance was used as distance measure. The accuracy was then calculated by dividing the number of correctly classified examples by the total number of test examples.

Experiment 3 K-nearest neighbor classification

[2]Multiple values of k were tested. The best results were achieved with $k = 25$.

Experiment	Accuracy (10 runs)
1a: GoogLeNet scratch, original images	$64.8 \pm 2.4\%$
1b: GoogLeNet pre-trained, original images	$63.3 \pm 3.0\%$
1c: GoogLeNet scratch, enhanced images	**67.1** $\pm 2.7\%$
1d: GoogLeNet pre-trained, enhanced images	$65.4 \pm 2.3\%$
2a: 3-layered net, original images	$62.4 \pm 2.1\%$
2b: 3-layered net, enhanced images	$62.8 \pm 1.7\%$
3: K-nearest neighbors	$57.3 \pm 1.2\%$

Tab. 5.1.: Average classification results for all experiments. The final accuracy is the result of averaging 10 individual testing runs for a given experiment.

5.4 Results

The final classification results are shown in table 5.1. The accuracy is averaged over 10 runs with different datasets. GoogLeNet trained from scratch with enhanced images (experiment 1c) performed best and achieved an accuracy of 67%. This is almost 10 percent higher than the outcome of the k-nearest neighbors baseline experiment (accuracy: 57%, experiment 3).

The convolutional neural networks in experiment 1 were trained for 8 epochs on datasets with a total of approximately 14000 images (6300 training images, 1400 validation images and 6300 test images). The 3-layered net in experiment 2 was trained for 2 epochs, with the same amount and distribution of images as in experiment 1. Figure 5.4 shows the training progress of a single run from experiment 1c (GoogLeNet trained from scratch with enhanced images). Figure 5.5 shows an example of the training progress for a single run of experiment 2b (3-layered convolutional neural network with enhanced images).

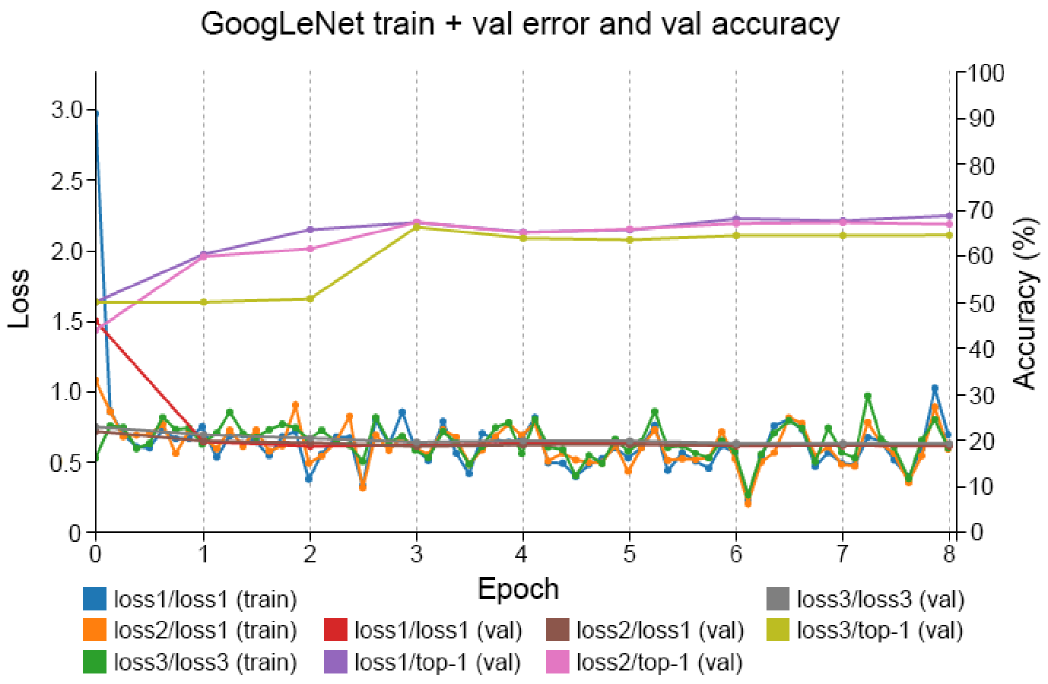


Fig. 5.4.: Training process for a single GoogLeNet run with enhanced images.

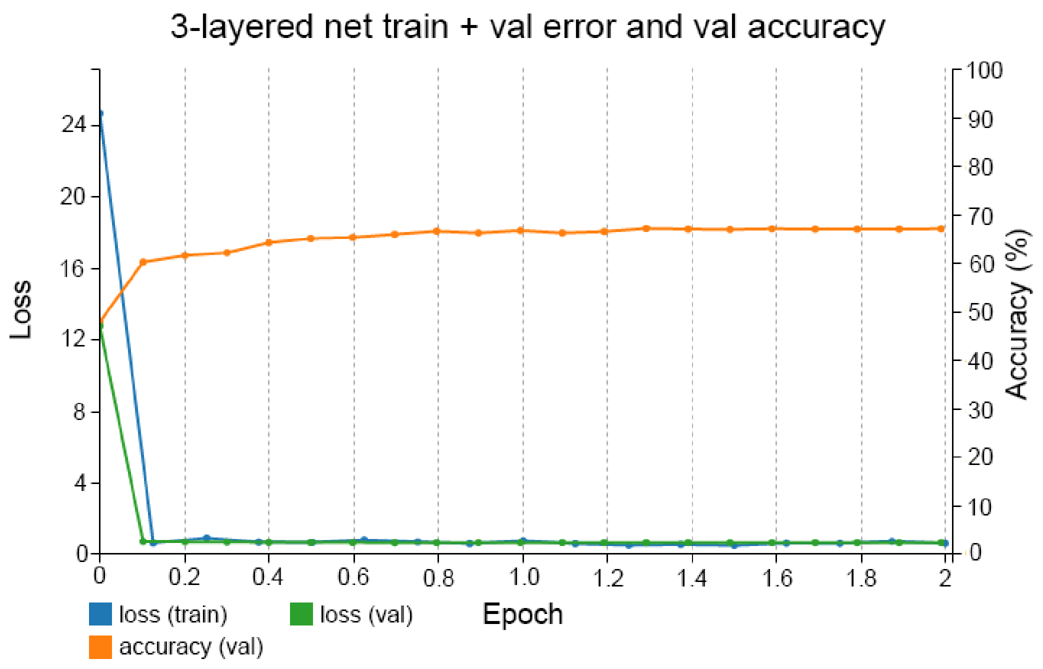


Fig. 5.5.: Training process for a single 3-layered CNN run with enhanced images.

6 Discussion, conclusion and future research

The most complex network used in the experiments, GoogLeNet, obtained the top performance with an accuracy of 67%. This is not more than a reasonable result. Possible improvements can be found later in this chapter, in section 6.2.

When we take a look at the training process, we see that the training error is greatly reduced during the first epoch and does not decrease that much from epoch 2 to 8. The same applies to the validation accuracy: sometimes the accuracy already peaks after 1 epoch. The only reason why training was run for 8 epochs is that in some cases the validation accuracy was very low in the beginning, possibly due to unlucky initialization or an unfavoured sequence of presented inputs, so it did need some iterations to get to a reasonable level.

The same phenomenon of early convergence of model parameters was visible for the less complex 3-layered convolutional neural network. This network only showed a drop in training error at the very beginning which did not decrease much afterwards. The accuracy did not improve when it was trained for more than 2 epochs. It is therefore hard to believe that the network successfully accomplished to learn all higher-order patterns of soccer which are important for creating opportunities. The network did probably only manage to learn a set of simple rules, which led to an average accuracy of a little less than 63%.

The difference in performance and training between GoogLeNet and the 3-layered convolutional neural network is interesting. The top-performing variant of GoogLeNet (trained from scratch, enhanced images) achieved an accuracy more than 4% higher than the 3-layered net. Although this is not a very big difference, it seems that GoogLeNet is able to capture more complex tactical patterns than the other models.

The average accuracy increase of approximately 2% for enhanced images over default images is promising as well. The added information in the form of object trails seems to be picked up by the models. The images could possibly be enriched with even more bells and whistles, which might be interesting to research in the future. The absence of this difference for the 3-layered network suggests that GoogLeNet is indeed able to catch some higher-order patterns, in which the 3-layered net does not succeed.

Then remains the question how bad an accuracy of 67% actually is. Of course it is significantly higher than the 57% accuracy of the k-nearest neighbors baseline. But what would the performance of the best performing human or artificial classifier be for the current problem? Would an accuracy of, say, 90% be realistic? This is probably not the case. Samples are extracted already 10 seconds before goal-scoring opportunities. A lot can happen in the 10 second interval to the actual opportunity. Even at the time of the shot itself, it can be hard to predict whether there is going to be a shot on goal from position data only. The ball does not even move in the direction of the goal yet, making it even harder. This illustrates that the problem is not a straightforward classification problem.

6.1 Conclusion

Research question Can convolutional neural networks be used to classify and predict goal-scoring opportunities when presented with position data from soccer matches?

The research question above can be answered positively: the results indicate that convolutional neural networks are capable of predicting goal-scoring opportunities to a certain extent. The performance of the methods used in this thesis may not be sufficient yet, but there are enough indicators of possible uses, which will be discussed in more detail in the following section.

6.2 Future research

The results suggest that convolutional neural networks are capable of predicting goal-scoring opportunities to a certain extent. The top performing net, however, achieved an accuracy of only 67%. There are a couple of ways how the performance of the convolutional neural networks could be improved.

6.2.1 More data

For this study 29 soccer matches were used, leading to a total of 349 detected goal-scoring opportunities. These 349 opportunities differ a lot: they emerged from corner kicks, counter attacks, carefully built-up attacking play, etcetera. Due to this variability within examples and the low number of examples, it is difficult for a convolutional neural network to detect repeated patterns. Superficial patterns will still be learned, but deeper, higher-order patterns may never end up in the neural net's parameters during training.

Another difficulty lies in samples extracted from the 10 second interval before goal-scoring opportunities which were still fed to the classifiers as opportunities. The game of soccer can only be controlled to a certain extent: sometimes opportunities are created due to pure luck or coincidence. The other way around holds as well: perfectly executed attacking play can be finished by an attacker missing the ball completely. A lot can happen during the 10-second interval before a possible opportunity. This problem could be eliminated by shortening the 10-second interval, but then the predictive power of the neural network is affected (and there would be even less examples to learn from).

6.2.2 Higher number of classes

Two classes were distinguished for this research: goal-scoring opportunities and loss of ball possession. The class distribution could be enriched by detecting other events in the data and using these for classification. The ultimate neural network could estimate from every single soccer game snapshot in which state it was in. This network could not only be used for prediction of opportunities, but also for other purposes such as automated match highlights detection.

Another way to alter the approach in this thesis is to handle samples extracted around goal-scoring opportunities differently. Now, every sample before goal-scoring opportunities has the same weight: a sample extracted 10 seconds before an opportunity receives the same label as an extracted sample from the exact instance of the shot on goal. An extension could be to introduce weights for samples around opportunities. A sample at the exact occurrence of the shot could be weighted with factor 1.0, while earlier samples are weighted less. The final output of the network can either be a class or a number approximating the probability that a goal-scoring opportunity is going to happen given a specific snapshot.

6.2.3 Using a more specialized neural network

The top-performing network on the soccer data was an instance of GoogLeNet, trained from scratch on enhanced images. GoogLeNet, however, is a very deep convolutional neural network originally designed to classify images from the Imagenet dataset into a spectrum of 1000 distinct classes. These images are from the real world, while the constructed soccer snapshots were abstract representations of soccer formations. These fundamental differences between the data suggest that a more soccer-focused convolutional neural network could achieve better accuracy.

Another cue which indicates that GoogLeNet might not be perfectly suitable for the soccer task is the equal performance of all three outputs. The first auxiliary classifier shows similar performance to the final classifier, while there are a lot of layers and inception modules in between. Similar results could have probably be achieved with a stripped-down version of GoogLeNet, using less than half of the layers.

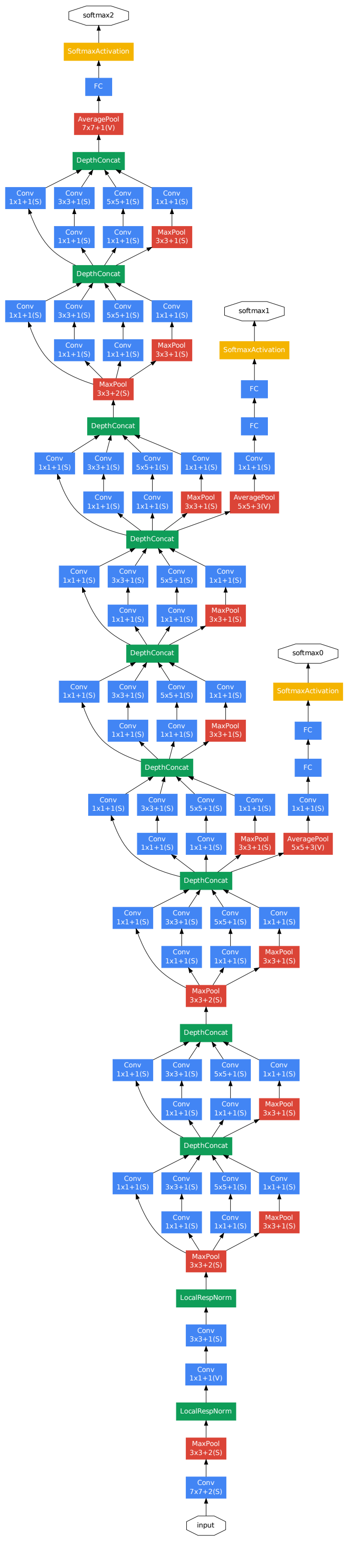
6.2.4 An ensemble of classifiers

Training a convolutional neural network is a form of supervised learning: given a set of input data and corresponding labels, the algorithm used for training sets all parameters/weights in the network appropriately. It would be interesting to add other methods which are grounded in sports science to form an ensemble of classifiers of different types. Additional classifiers could take centroids into account, surfaces of teams, focus on object trajectories, and so on.

As the previous sections indicate, there are still a lot of roads unexplored in the field of applying deep neural networks to soccer position data. This research could hopefully be a starting point for more research in this interesting field.

Appendices

A GoogLeNet


softmax2
SoftmaxActivation
FC
AveragePool 7x7+1(V)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
softmax1
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
SoftmaxActivation
MaxPool 3x3+2(S)
FC
DepthConcat
FC
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
AveragePool 5x5+3(V)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
DepthConcat
softmax0
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
SoftmaxActivation
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
FC
DepthConcat
FC
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
AveragePool 5x5+3(V)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
MaxPool 3x3+2(S)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
MaxPool 3x3+2(S)
LocalRespNorm
Conv 3x3+1(S)
Conv 1x1+1(V)
LocalRespNorm
MaxPool 3x3+2(S)
Conv 7x7+2(S)
input

Bibliography

[1] Sian Barris and Chris Button. „A review of vision-based motion analysis in sport“. In: *Sports Medicine* 38.12 (2008), pp. 1025–1043 (cit. on p. 25).

[2] Robert M Bell and Yehuda Koren. „Lessons from the Netflix prize challenge“. In: *ACM SIGKDD Explorations Newsletter* 9.2 (2007), pp. 75–79 (cit. on p. 14).

[3] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. „Learning long-term dependencies with gradient descent is difficult“. In: *IEEE transactions on neural networks* 5.2 (1994), pp. 157–166 (cit. on p. 21).

[4] Andrew Borrie, Gudberg K Jonsson, and Magnus S Magnusson. „Temporal pattern analysis and its applicability in sport: an explanation and exemplar data“. In: *Journal of sports sciences* 20.10 (2002), pp. 845–852 (cit. on pp. 5, 36).

[5] Leo Breiman. „Random forests“. In: *Machine learning* 45.1 (2001), pp. 5–32 (cit. on p. 14).

[6] Julen Castellano, Bruno Figueira, Diogo Coutinho, Jaime Sampaio, et al. „Identifying the effects from the quality of opposition in a Football team positioning strategy“. In: *International Journal of Performance Analysis in Sport* 13.3 (2013), pp. 822–832 (cit. on p. 7).

[7] Julen Castellano, David Casamichana, and Carlos Lago. „The use of match statistics that discriminate between successful and unsuccessful soccer teams“. In: *Journal of human kinetics* 31 (2012), pp. 137–147 (cit. on p. 5).

[8] Thomas Cover and Peter Hart. „Nearest neighbor pattern classification“. In: *IEEE transactions on information theory* 13.1 (1967), pp. 21–27 (cit. on p. 44).

[9] Jia Deng, Wei Dong, Richard Socher, et al. „Imagenet: A large-scale hierarchical image database“. In: *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2009, pp. 248–255 (cit. on pp. 2, 19).

[10] Hugo Folgado, Ricardo Duarte, Orlando Fernandes, and Jaime Sampaio. „Competing with lower level opponents decreases intra-team movement synchronization and time-motion demands during pre-season soccer matches“. In: *PloS one* 9.5 (2014), e97145 (cit. on p. 7).

[11] Sofia Fonseca, João Milho, Bruno Travassos, and Duarte Araújo. „Spatial dynamics of team sports exposed by Voronoi diagrams“. In: *Human movement science* 31.6 (2012), pp. 1652–1659 (cit. on p. 8).

[12] Wouter Frencken, Koen Lemmink, Nico Delleman, and Chris Visscher. „Oscillations of centroid position and surface area of soccer teams in small-sided games“. In: *European Journal of Sport Science* 11.4 (2011), pp. 215–223 (cit. on pp. 7, 33).

[13] Andreas Grunz, Daniel Memmert, and Jürgen Perl. „Tactical pattern recognition in soccer games by means of special self-organizing maps“. In: *Human movement science* 31.2 (2012), pp. 334–343 (cit. on p. 6).

[14] Kevin Gurney. *An introduction to neural networks*. CRC press, 1997 (cit. on p. 11).

[15] Kaiming He and Jian Sun. „Convolutional neural networks at constrained time cost“. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 5353–5360 (cit. on p. 22).

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. „Deep residual learning for image recognition“. In: *arXiv preprint arXiv:1512.03385* (2015) (cit. on pp. 19, 22, 23).

[17] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. „Multilayer feedforward networks are universal approximators“. In: *Neural networks* 2.5 (1989), pp. 359–366 (cit. on p. 10).

[18] Inmotio. *Inmotio company website*. [Online; accessed 6-February-2016]. 2015 (cit. on p. 25).

[19] Robert A Jacobs. „Increased rates of convergence through learning rate adaptation“. In: *Neural networks* 1.4 (1988), pp. 295–307 (cit. on p. 13).

[20] Yangqing Jia, Evan Shelhamer, Jeff Donahue, et al. „Caffe: Convolutional architecture for fast feature embedding“. In: *Proceedings of the 22nd ACM international conference on Multimedia*. 2014, pp. 675–678 (cit. on p. 42).

[21] S Kim. „Voronoi analysis of a soccer game“. In: *Nonlinear Analysis: Modelling and Control* 9.3 (2004), pp. 233–240 (cit. on p. 8).

[22] Konstantin Knauf, Daniel Memmert, and Ulf Brefeld. „Spatio-temporal convolution kernels“. In: *Machine Learning* 102.2 (2016), pp. 247–273 (cit. on p. 5).

[23] Teuvo Kohonen and Panu Somervuo. „Self-organizing maps of symbol strings“. In: *Neurocomputing* 21.1 (1998), pp. 19–30 (cit. on p. 6).

[24] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. „Imagenet classification with deep convolutional neural networks“. In: *Advances in neural information processing systems*. 2012, pp. 1097–1105 (cit. on pp. 17–20, 43).

[25] MJ Kurz and Nicholas Stergiou. „Applied dynamic systems theory for the analysis of movement“. In: *Innovative analyses of human movement* (2004), pp. 93–117 (cit. on p. 7).

[26] Y LeCun, B Boser, JS Denker, et al. „Handwritten digit recognition with a backpropagation network“. In: *Advances in neural information processing systems 2*. Morgan Kaufmann Publishers Inc. 1990, pp. 396–404 (cit. on p. 14).

[27] Tim McGarry, Peter O'Donoghue, and Jaime Sampaio. *Routledge handbook of sports performance analysis*. Routledge, 2013 (cit. on p. 5).

[28] Daniel Memmert, Koen APM Lemmink, and Jaime Sampaio. „Current Approaches to Tactical Performance Analyses in Soccer Using Position Data". In: *Sports Medicine* (2016), pp. 1–10 (cit. on pp. 6, 7).

[29] J Moody, S Hanson, Anders Krogh, and John A Hertz. „A simple weight decay can improve generalization". In: *Advances in neural information processing systems* 4 (1995), pp. 950–957 (cit. on p. 12).

[30] Vinod Nair and Geoffrey E Hinton. „Rectified linear units improve restricted Boltzmann machines". In: *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*. 2010, pp. 807–814 (cit. on p. 16).

[31] Yurii Nesterov. „A method of solving a convex programming problem with convergence rate O (1/k2)". In: *Soviet Mathematics Doklady*. Vol. 27. 2. 1983, pp. 372–376 (cit. on p. 43).

[32] NVIDIA. *NVIDIA Deep Learning GPU Training System (DIGITS)*. [Online; accessed 5-September-2016]. 2016 (cit. on p. 42).

[33] Steven M Pincus. „Approximate entropy as a measure of system complexity." In: *Proceedings of the National Academy of Sciences* 88.6 (1991), pp. 2297–2301 (cit. on p. 7).

[34] Joshua S Richman and J Randall Moorman. „Physiological time-series analysis using approximate entropy and sample entropy". In: *American Journal of Physiology-Heart and Circulatory Physiology* 278.6 (2000), H2039–H2049 (cit. on p. 7).

[35] Frank Rosenblatt. „The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386 (cit. on p. 9).

[36] J Sampaio and V Maçãs. „Measuring tactical behaviour in football". In: *International Journal of Sports Medicine* 33.05 (2012), pp. 395–401 (cit. on p. 7).

[37] Jürgen Schmidhuber. „Deep learning in neural networks: An overview". In: *Neural Networks* 61 (2015), pp. 85–117 (cit. on p. 1).

[38] Jeffrey Snyder. „What Actually Wins Soccer Matches: Prediction of the 2011-2012 Premier League for Fun and Profit". In: (2013) (cit. on p. 5).

[39] Nitish Srivastava. „Improving neural networks with dropout". PhD thesis. University of Toronto, 2013 (cit. on pp. 13, 20).

[40] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. „Dropout: a simple way to prevent neural networks from overfitting." In: *Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958 (cit. on pp. 13, 14, 20).

[41] Ilya Sutskever, James Martens, George E Dahl, and Geoffrey E Hinton. „On the importance of initialization and momentum in deep learning." In: *ICML (3)* 28 (2013), pp. 1139–1147 (cit. on p. 13).

[42] Christian Szegedy, Wei Liu, Yangqing Jia, et al. „Going deeper with convolutions“. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 1–9 (cit. on pp. 19–22).

[43] Paul J Werbos. „Backpropagation through time: what it does and how to do it“. In: *Proceedings of the IEEE* 78.10 (1990), pp. 1550–1560 (cit. on p. 11).

[44] Matthew D Zeiler and Rob Fergus. „Visualizing and understanding convolutional networks“. In: *European Conference on Computer Vision*. Springer. 2014, pp. 818–833 (cit. on p. 22).