Football activity recognition

Improving and testing football activity recognition based on signal data using deep learning.

Ricardo Tebbens

TU Delft

Football activity recognition

Improving and testing football activity recognition based on signal data using deep learning.

by

Ricardo Tebbens

to obtain the degree of
Master of Science
in Applied Mathematics
specialization in Stochastics
at the Delft University of Technology
to be defended publicly on March 27, 2023

Abstract

There is a raising demand for player statistics in the world of football. With the developments over the last years in wearable sensors, Human Activity Recognition (HAR) based on wearable IMU sensors can be used to tackle this problem. This thesis builds upon an earlier research done for this topic, where an end-to-end pipeline based on deep learning was created that is able to be trained and used for football activity recognition. The goal is to test and improve said pipeline. This was done by adding change of directions (COD's) to the classifiable activities. Furthermore, run velocities were build as a spectrum with several categories depending on the speed. A combination of convolutional and recurrent layers resulted in test accuracies up to 88.9%.

Afterwards, the pipeline was used to evaluate larger datasets containing football drill and a football physiotherapy training. For this a sliding window evaluation procedure was proposed. These evaluations gave promising results. Many actions and football related activities could be recognized, however many smaller, shorter actions were missed. This can be seen as lack in trainingdata. In this data, little activities with the ball were present. Hence the deep learning models could not be trained accordingly. Later, it was researched if additional training of activities with ball increased the evaluation. This was indeed confirmed, since the evaluations showed more detailed and realistic results. Including even more additional trainingdata, could result in the pipeline performing reliably in real-life football scenario's.

Acknowledgements

Over a year ago, I started to orientate myself for a master thesis. Since the start of master program Applied Mathematics, I developed a big interest in machine learning and deep learning, and wanted to find a thesis in this field. On the other hand, my passion since I was young was football. Since I did not have the talent to make it as a player, it would be great to use my interest in mathematics for that field. During interesting conversations with Dr. Jakob Sohl and Professor Dr. ir. Geurt Jongbloed, they mentioned this project. This made it the ideal project for me from the start. During this project, I have learned many things. Knowledge about Deep learning and programming, but also about how to be part of a bigger project, knowing when to communicate with people, who to contact, how to present updates of an ongoing project, how to plan a project over a long period of time, and much more.

Now I would like to take a moment to thank some people who helped me through this project. First of all, my deepest gratitude to Professor Dr. Ir. Kaspar Jansen and PhD candidate Michał Ciszewski. Their daily supervision was of vital importance for this thesis. They were always available for questions, suggestions, ideas etcetera. We have had some long and difficult discussions together, but these all contributed to the thesis as it is now. Second, are Dr. Jakob Sohl and Professor Dr. ir. Geurt Jongbloed. As mentioned, they helped me find this project. Furthermore they helped with me with administration problems around the thesis and accepted to be on my committee, for which I am grateful. Next is Rafael Cuperman Coifman. You will find his name or a citation to his article multiple times in this report. He was my direct predecessor in this project and helped me to get started. The work he did for this project was of a very high level and it was a very interesting challenge to continue his work. Also a big thank you to PhD Candidates Erik Wilmes and Bram Bastiaansen. They provided the data used in this thesis, and were always open to help and answer questions to get a better understanding of what the mathematical results tell us in practice. Bastiaansen also let me look over his shoulder several times, to see how the data was collected. This was very fun and interesting to see and provided more background for the data. And finally, obviously a very big thank you to my parents, sisters, friends and girlfriend. The last months were not easy, I faced a lot of challenges along the way. And I could not have done it without them.

Contents

1 Introduction

Analyzing data and statistics becomes of increasing importance in professional sports. When watching the 2022 World Cup football in Qatar, every few minutes player statistics such as passes and distance covered is displayed on the screen. To a general spectator, these statistics are unimportant. But for teams, coaches and players they become more and more crucial. Statistics tell teams and players about performance, fitness and possible predictions for future games, which can be vital in professional sports. For this reason, it is necessary to capture and measure these statistics. Nowadays, this is usually done high-quality camera's, or by people manually writing down every action of a certain player or team. Since these camera's tend to very expensive and difficult to place and manually keeping track of each action is time consuming, it is beneficial to use Inertial Measurement Unit Sensors attached to several parts of the body. These contain accelerometers and gyroscopes which tell the precise accelerations and rotations performed by those body parts. Once that data is available, the goal is to let a computer recognize which activity is performed based on that data. This is called Human Activity Recognition. Moreover, these sensors are especially useful since they are low-cost, especially compared to high-quality camera's and manual labeling. The sensors also became cheaper and smaller over the last decade, which makes it possible for them to be implemented in sport clothing, such that the wearer is not limited in his movements.

The goal of this thesis is to use the data from these sensors to let the computer classify several different football related activities such as passing, sprinting and jumping. This project is a continuation of an earlier project conducted by Cuperman Coifman in 2021 [2]. He created an end-to-end pipeline, which uses deep learning to create and train models that are trained to classify 5 different football activities: jogging, sprinting, passing, shooting and jumping. Once a model is trained, a technique called the sliding window evaluation method is used to classify larger datasets. This project will expend on this. Firstly, change of directions are added to the list of activities. These actions are very common in football and put a load of pressure on the joints and muscles of players. Secondly, the jogging and sprinting are replaced for different run intensities. This is done since jogging can be seen as slow running and sprinting as fast running, but everything in between is ignored. Moreover, what would classify as sprinting for one untrained, can be seen as jogging for a professional sportsman. Therefore, these categories are replaced by intensity levels based on running speed. Once deep learning models are trained with the additional activities, they are used to analyze large datasets. These datasets consist of a football drill and a full physiotherapy training. The aim is to see how the end-to-end pipeline performs for realistic football scenarios.

This document will be structured as follows. First a literature review will be presented in section 2. Here former research into Human Activity Recognition using deep learning and Human Activity Recognition used in football is explored, and finally a short introduction to [2] will be given. In section 3, an explanation of the used sensors and used datasets will be given. Afterwards the fundamentals of deep learning techniques used in this thesis will be explained in section 4. In section 5, the whole end-to-end pipeline will be presented. This includes training of the deep learning models and the sliding window evaluation method. Afterwards, the results from the training and the evaluation of larger datasets are presented in section 6. Finally in section 7 conclusions will be drawn and discussed, with recommendations for future work.

2 Literature review

With the rise of computer power in the last 40 years, the possibilities of computer applications has also increased. One of these applications is recognising and identifying human activities with the use of forms of Artificial Intelligence (AI). The most applicable forms of AI to this day are machine learning models and deep learning models. The general idea of these forms are similar, namely the user should feed it data that it needs to learn to recognize. Once a model has trained itself, it can be used to let a computer recognize certain features, such as objects, shapes, environments and movements. The difference between these models lies in the way they are trained, and will be explained in more detail later. Since these types of models of models become increasingly popular, also in Human Activity Recognition (HAR), there is much work to build upon.

In this section, firstly multiple results of different scientific researches about HAR will be presented. These results were obtained by using sensor data coming from wearable inertial sensors. In these articles, the results were computed with the use of deep learning models. Afterwards, there will be looked at research results around football activity recognition. Finally, there will be a broader look at the results of Cuperman Coifman from 2021. This thesis is a direct continuation of that report, which makes it important to explain several results in more detail.

2.1. HAR using deep learning

As mentioned, deep learning is a fast growing industry, also in HAR. This comes mostly from the fact that it is both faster and more accurate that the traditional techniques used in machine learning [3]. It also states that usage of deep learning in Human Activity Recognition tasks is recent and promising but not yet fully explored. This is also notable when doing research in the usage of football. Except [2], which will be analysed in more detail in section 2.3 and used throughout the rest of this thesis, there is no research done using sensor data in football activity recognition.

Numerous studies have been conducted on Human Daily Activity Recognition using algorithms similar to the ones proposed in this thesis. Therefore, the methodology and results of these studies were considered. Table 2.1 presents a summary of the reviewed articles. The metrics and results in the table cannot be compared among themselves, as each paper worked with different datasets. However, they provide illustrative evidence of the potential of these approaches. Convolutional Neural Networks are a popular choice due to their ability to extract and leverage latent feature representations in time series with high tolerance of time translation [5]. Many deep architectures, from consecutive convolutional layers to more complex and modern approaches like inception and residual CNNs, were reviewed. However, “ (...) CNN lacks the capability to capture temporal dependency in timeseries sensory data. RNN is designed to model time series data, and it is suitable for discovering relationships in temporal dimension” [13]. Which is why other authors evaluated the usage of Recurrent Neural Networks, mainly using LSTM units. A noteworthy approach is the combination of CNN’s and RNN’s to build a larger and better-performing network.

Reference	Activity type	Layer type	Accuracy
[4]	Volley-ball	CNN	83.2%
[5]	Golf	CNN	95.1-97.7%
[6]	Multiple sports	CNN	99.9%
[7]	Ballet	CNN	98.2%
[8]	Daily activities	CNN	97.4-98.3%
[9]	Daily activities	CNN	91.3-91.9%
[10]	Daily activities	CNN	0.883 (F1 score)
		CNN+LSTM	0.915 (F1 score)
[11]	Daily activities	CNN	0.894 (F1 score)
		LSTM	0.912 (F1 score)
		bLSTM	0.927 (F1 score)
[12]	Daily activities	LSTM	96.7-97.8%
		bLSTM	92.5%
[13]	Daily activities	CNN+LSTM	85%

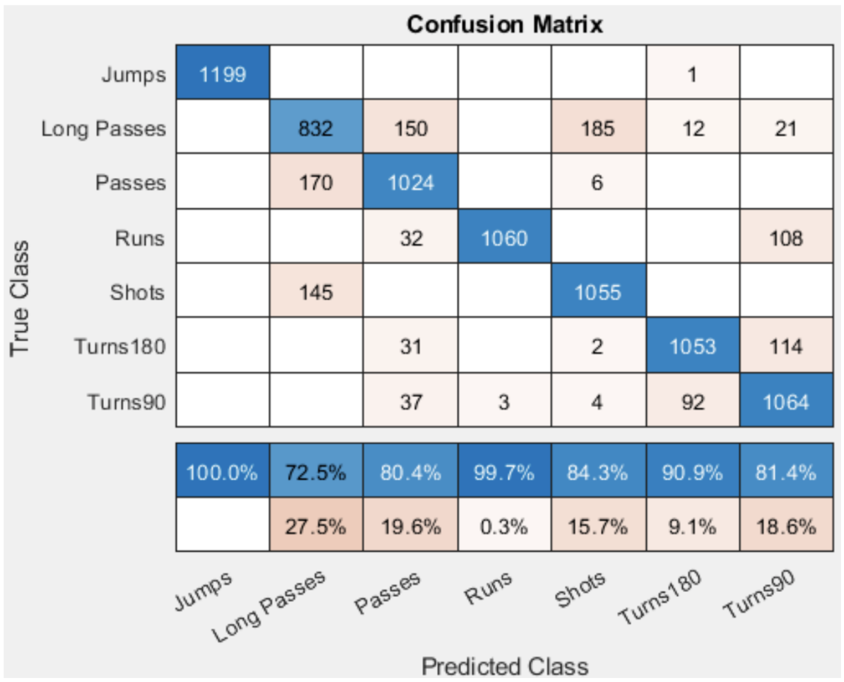
Table 2.1: Results of HAR using deep learning with accuracies as stated in the respective paper.

2.2. HAR used in football

As stated in section 2.1, not many football activity recognition research papers can be found. Especially when using deep learning, except [2], only one other article is worthy of mentioning. That is [14]. This article uses a technique called Actor Relation Graph (ARG) to detect and classify both individual and group football activities. This is a special form of deep learning which learns from different camera frames. With this technique, the writer of the article is able to detect many different actions, such as dribbles, intercepting, tackling, but also also group activities such as which team is attacking. It can test accuracies up to 94% for individual activities and 70% for team activities. Clear downside of this method is that the football match or training has to be recorded, which is not the case in many instances.

Football activity recognition using sensor data, has previously only been done using machine learning by [15] and [16]. They made use of several machine learning methods, in particular supervised learning, whereas this thesis will use deep learning methods. The two methods are closely related, so much so that deep learning is seen as a subset of machine learning. Machine learning is a subset of artificial intelligence, and refers to the study of computer systems that learn and adapt automatically from experience, without being explicitly programmed. Hence, when it is fed data with a specific outcome of that data, a machine learning model trains itself to predict the outcome of any newly given data. The difference between machine and deep learning lies in the way it trains itself. Machine Learning needs mostly human intervention to train, which is done based on fixed algorithms. Deep learning trains on its on own, purely based on its environment and past mistakes. The concept of deep learning will be explained further in section 4.

A system was developed by [15] that distinguishes between pass and shot using a peak detection algorithm and trained Support Vector Machines, Decision Trees, and Naive Bayes classifiers. The linear SVM approach yielded the best results, achieving up to 88.6% accuracy. [16] evaluated several classifiers, including Naive Bayes, k-Nearest Neighbor, Support Vector Machines, Discriminant Analysis, and Decision Trees, to classify between 4 or 7 football activities. The influence of including or excluding frequency domain features and the extraction of features from raw signals or from the euclidean norm of the signals from X, Y, and Z axis of each sensor were also evaluated. The SVM approach produced the best accuracy of 92% in his dataset. Additionally, the results for recognizing turns were interesting, with 90.9% accuracy for 180 degree COD's and 81.4% accuracy for 90 degree turns, which will be a new factor explored in this thesis. The results are shown in Figure 2.1.



Confusion Matrix

True Class \ Predicted Class	Jumps	Long Passes	Passes	Runs	Shots	Turns180	Turns90
Jumps	1199					1	
Long Passes		832	150		185	12	21
Passes		170	1024		6		
Runs			32	1060			108
Shots		145			1055		
Turns180			31		2	1053	114
Turns90			37	3	4	92	1064
	100.0%	72.5%	80.4%	99.7%	84.3%	90.9%	81.4%
		27.5%	19.6%	0.3%	15.7%	9.1%	18.6%

Figure 2.1: Confusion matrix of the analysis performed by [16].

2.3. Cuperman Coifman 2021

As mentioned, this thesis is the direct continuation of a similar project performed by Cuperman Coifman in 2021 [2]. Therefore it is important to highlight several results from that report in greater detail. Important is to note that from Section 5, a large portion is also present in the paper of Cuperman Coifman. This is included separately since it consists of vital information in how the models used in this thesis are build, trained and used. The results described in this section are main results, that will later be used without further explanation.

2.3.1. Explosive/Periodic activities

The first task in classifying football related activities, was to identify whether is was a periodic activity, such as running, or an explosive activity, such as shots and jumps. The difference between these two groups is rather intuitive. Periodic activities are activities where a certain pattern of movements is repeated over a period of time. An explosive activity is an activity where the movement is only performed once without a notable pattern. [2] has shown that these two groups can best be identified with using Interquartile Range (IQR) as a metric. The IQR is calculated as the difference between the 75th and 25th percentiles and it measures the statistical dispersion of a signal or set of values. To classify an activity as periodic or explosive, the euclidean norm of all the accelerometer signals of the recording was taken, then this resulting signal was normalized between 0 and 1, and finally the IQR was calculated. If this value exceeded a certain threshold, the recording was considered as a periodic movement or, otherwise, as an explosive movement. The best threshold found for the IQR to make this distinction can be set to 10 for unnormalized signals and 0.12 for normalized ones. In Figure 2.2 it can be seen that this method reached very high accuracies.

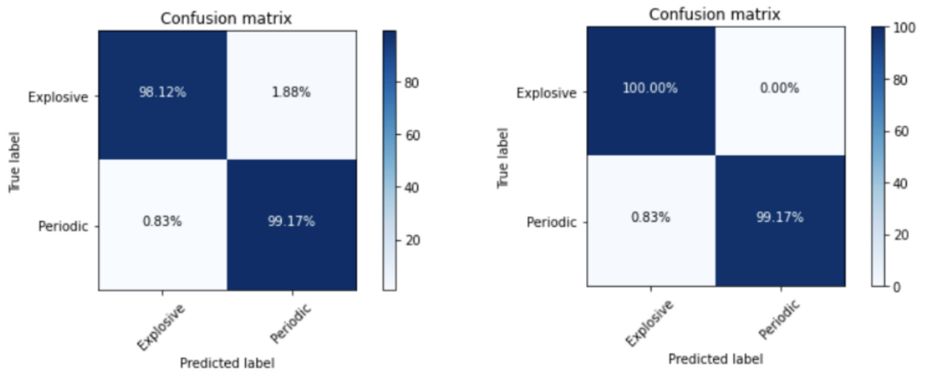


Figure 2.2: Confusion matrices for explosive vs periodic movements classifier based on IQR. Unnormalized signals with threshold 10 on the left and normalized signals with threshold 0.12 on the right [2]

2.3.2. Activity detection

As will be explained in section 3.2, the used to train the deep learning models contains isolated activities. Hence, before and after each performed activity, there exists a moment of standing still or slowly walking by the participant. This we call moments of low activity. This causes that all activity files start and end with several milliseconds of low activity. The data shown here is the accelerator data of all sensors of a participant performing a shot at goal. It can clearly be seen that the data starts with a low activity, followed by the shot, and ending again with low activity.

The goal is to isolate the data where the actual activity takes place and to scrub the periods of low activity. This comes from the fact that low activity data contains no relevant information, which can only mess up the model. Therefore, [2] gives an algorithms to isolate the actual activity. This goes as follows:

1. Take only the accelerometer data from the 5 locations: pelvis, left thigh, left shank, right thigh, and right shank.
2. Take the euclidean norm of all the accelerometer signals. This results in one norm signal.
   (a) Calculate the IQR of the signal.
   (b) If the IQR is less than or equal to 0.12, consider the recording as an explosive movement. Otherwise, consider it as a periodic movement.
3. Take the euclidean norm of X, Y and Z axis of each sensor location. This results in 5 norm signals.
4. For each one of the 5 signals:
   (a) Calculate the mean value.
   (b) Set the threshold to be the mean if the recording was classified as a periodic activity. If the recording was classified as an explosive activity, set the threshold as 1.5 times the mean
   (c) Start of activity: find the first timestep where the signal and the following 50 timesteps are larger than the previously defined threshold. Take that timestep as the start of the activity.
   (d) End of activity: find the last timestep where the signal and the previous 50 timesteps are larger than the previously defined threshold. Take that timestep as the end of the activity.
5. Find the overall start and end of the activity.
   (a) Overall start of activity: take the minimum among the starts of activity from the previous step. Subtract 250 timesteps (if possible).
   (b) Overall end of activity: take the maximum among the ends of activity from the previous step. Add 250 timesteps (if possible).

Applying this procedure results in Figures 6.11 and 2.4. Figure 6.11 represents a short pass, which we consider to be an explosive activity, whereas Figure 2.4 represents a sprint which is a periodic activity.

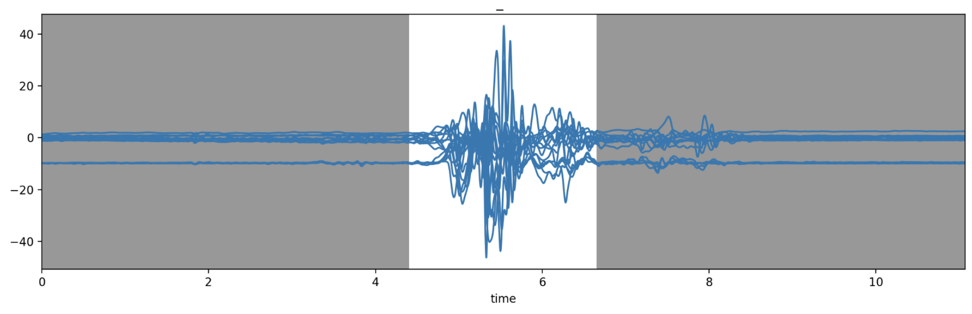


Figure 2.3: The raw accelorator data of a pass. The grey area is classified as low activity, and is therefore scrubbed.

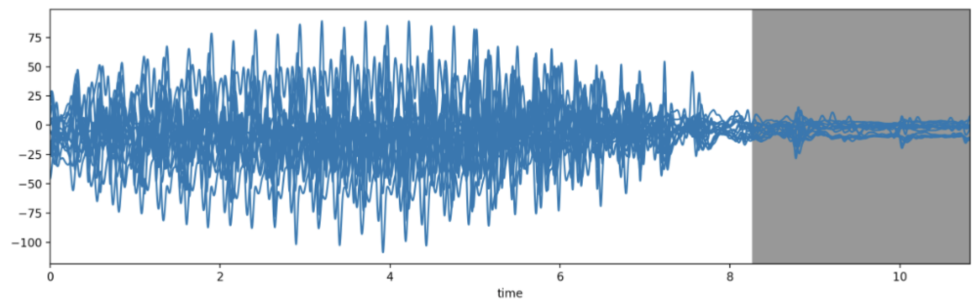


Figure 2.4: The raw accelorator data of a sprint. The grey area is classified as low activity, and is therefore scrubbed.

3
Data overview

Our goal is to recognize football-related activities based on readings from wearable sensors. First, we will explain the specifics of the data sets. This contains the structure of the data, the sensors that were used to obtain them and how the data sets were structured in terms of football activities.

3.1. Sensors

For the collection of data used in this thesis, two kinds of sensors were used. Most important are the Inertial Measurement Units. These give the data that will be used in the activity classification. The other sensors used were VICON sensors. Here, the sensors are explained in more detail.

3.1.1. Inertial Measurement Units

Inertial Measurement Units (IMU's) are small, often wireless electronic devices embedded with a variety of sensors of different types. These sensors are shown in Figure 3.1. For HAR tasks, they are often placed on different body parts of a person to capture the movements that they are doing. These devices usually contain triaxial sensors such as accelerometers, gyroscopes and magnetometers, so that it is possible to capture information about the changes in acceleration, velocity and magnetic field that occur when the person wearing the IMUs performs an activity.

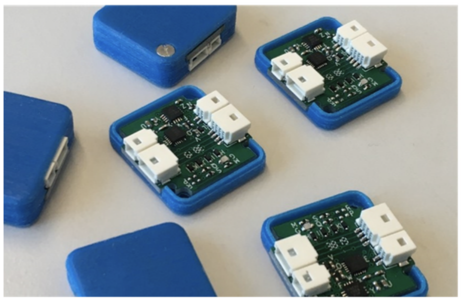


Figure 3.1: IMU's [2].

All experiments had the same sensor setup. Participants equipped with wearable sensors were to perform several football-related activities. Each of them had 5 IMU's attached to their body parts, namely: both shanks, both thighs and the pelvis. The placement of the sensors is shown in figure 3.2. Each IMU contained an accelerometer, gyroscope and magnetometer, each triaxial and with sensor frequency 500 Hz. The accelerometers had a range set to $\pm 16g$ while the range of the gyroscopes was $\pm 2000°$ per second. The data from the magnetometer was not used in this thesis.

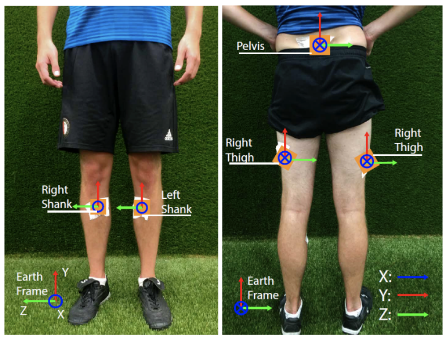


Figure 3.2: Sensor placement [2].

3.1.2. VICON

Other sensors that were used were VICON sensors. VICON sensors are often used as a ground truth from which to compare the IMU data. These work with the use of infra-red camera's and markers placed on several places on the body. The camera's send out a light, which is reflected. This reflected light is then captured again, and with the use of that, the exact locations of the markers are determined. These locations come in the form of a triaxial datapoint and are measured with a frequency of 250 Hz. The datapoints consists of "coordinates". This will later be used in determining the running speed of participants. Note that these measurements are solely used during the training phase of a model, once a model is trained, the VICON measurements are no longer used.

3.2. Data sets

Several different data sets were used. Firstly, we have a data set where the football activities are isolated, which makes it ideal to use for training. Afterwards, we have a data set to test these trained models. Moreover, it contains football activities that follow up on each other, which makes it a more realistic representation of a real football game. Finally, we use a data set of a football training, where there is no laid out order what activities the participants have to perform.

3.2.1. Wilmes, 2019

The first data set used was obtained by Wilmes in 2019 [17]. The experiment had 11 male football players performing each several isolated football activities. The activities were not coinciding with each other as in, before and after each activity there was a period of standing still. This makes it easy to separate the different activities. The activities performed were

- 10 meter run at several intensities (jog, sub-maximal and maximal)
- 10 meter run with rapid stop, run at at several intensities
- 10 meter run with a 180 degree turn in the middle, run at at several intensities
- 5 meter run ending with a 90 degree turn, run at at several intensities
- Jumps, from standing still and with small run-up
- Kicks, including short passes, long passes and shots
- 30 meter full sprint

Since the activities were isolated, this data set is ideal to train the models with. Also were they used as a first check whether the trained model performed well. For these experiments, the participants have worn both IMU and VICON sensors.

3.2.2. Bastiaansen, 2021

The second dataset used was obtained by Bastiaansen in 2021. It is still unpublished, but could used with Bastiaansen's approval. It contains the data of one participant performing several football activities, including jumping, sprinting, shooting and football training drill. This training drill consists of several stage, namely

- Walking forward/backward/sideways
- Jumping plus heading
- Running at different velocities
- Passing over several distances
- Change of direction while running

This data set was used to test how a trained model performs on different data, where the football activities are no longer isolated. The focus here is on the football training drill, since it consists of multiple different football activities that follow up on each faster than in the previous data set. In this experiment, the participant did not wear any VICON sensors. Later, several parts were of the data was also used to further train the model.

3.2.3. Bastiaansen, 2022

The third dataset used was obtained by Bastiaansen in 2022. It is still unpublished, but could used with Bastiaansen's approval. It contains a physiotherapy football training with three participants. During this training, the participants performed several football activities including running, passing, shooting and dribbling. This dataset is especially interesting, since the actions of the participants could be seen as "semi-random". This means that the participants are not told when to perform a certain action, or how many actions should be performed. This results in data which is a more realistic representation of how football players perform during a real game.

4
Deep Learning

In this thesis, deep learning is the basis of all computations. In the section, the theory behind deep learning will be explained in detail.

4.1. General introduction

Deep learning is a form of Artificial Intelligence (AI), and an advanced form of Machine Learning. The principle of deep learning is simple: let a computer learn by example. As an example, as children we all learned the differences between a car and plane. A car has four wheels and drives on a road, while a plane has wings and flies in the air. Of course there are many more complex differences such as forms and shapes. This gives a certain complexity hierarchy where features as it having wheels or wings is the least complex, and certain shapes are more complex. For humans, these complexities are easy to notice. For computers, this is more complicated. Therefore, deep learning is using this hierarchy in the form of an artificial neural network. This is done with the help of so-called hidden layers. Figure 4.1 shows this procedure.

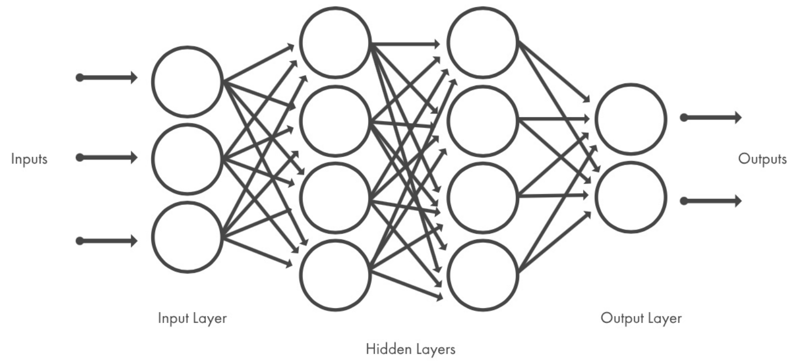


Figure 4.1: Schematic overview of an artificial neural network [18].

From the left, a picture of a plain comes in, this is the input layer, where the three balls could represent groups of bits in that picture. In the first hidden layer, it would for example search for wings. Once it located the wings, it moves to the second hidden layer where it looks for the general shape of the plane. Afterwards, once the model has found wings and the shape, it can output whether the model thinks the input represents a car or a plane.

These hidden layers can be trained to look for certain characteristics. This is done very intuitively, namely show a lot of pictures of planes, tell the model it's a plane, and it will look what could be characteristics of a plane. Similar for the cars. After having trained enough, it should be able to see the difference between a car and a plane, just like humans notice the difference because we have seen many cars and planes in our lives. So once it's done training, the model can be used to automatically detect cars and planes in pictures.

This general concept is similar for using IMU measurements to classify football activities. The artificial neural network is fed a string of sensor data, it's told what kind of activity it represents, and the network looks for certain patterns in the data that could characterize that data.

Artificial neural networks, as their name suggest, are designed to mimic the structure and function of neurons of the human brain. They are composed by single units of artificial neurons that specialize in learning specific structures or patterns of the input data. This means that a neuron is internally computing the equation

$$y = \sigma(w^T x).$$

This can be seen in figure 4.2. Here is x the input, y is the output, w is the weight between neurons and σ is called the activation function. An activation function decides whether a neuron should be activated or not. This means that it will decide whether the neuron's input to the network is important or not in the process of prediction. If we include a bias b in the inputs of the neuron, its internal operation can be written as $y = \sigma(w^T x + b)$. When an artificial neuron is trained, it learns the weights w.

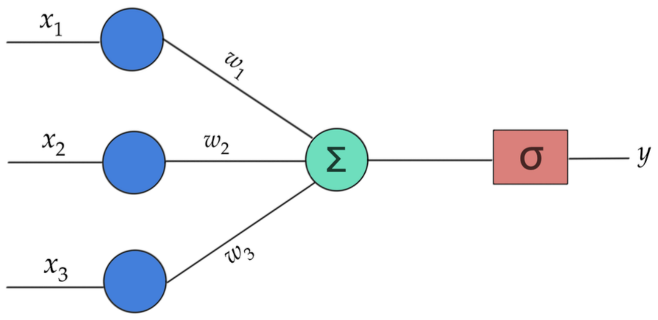


Figure 4.2: Simple artificial neuron model [19].

Several artificial neurons can be used together. When this is done and all the neurons between one layer and another are connected, it is called a dense or fully-connected layer. The combination of several fully connected layers one after the other is the basic configuration of a neural network. This can also be seen in figure 4.1. Even though each one of the individual signals compute a rather simple operation on its inputs, the combination of neurons along several layers allows the network to learn complex functions.

There are many different types of neural networks based on variations of the basic structure of artificial neurons. In particular, this thesis focuses on two of those types: Convolutional Neural Networks and Recurrent Neural Networks.

4.2. Convolutional Neural Networks

Convolutional Neural Networks (CNN's) are a type of deep learning models that were originally designed to tackle artificial vision and image processing problems. Studies on vision and perception of shapes show that the neurons responsible for those tasks have receptive fields. Hence, each neuron has the task to find its own specific pattern. In the previous example of cars and planes, this would mean that certain neurons look for wings, some for the engine of the vehicle, some for wheels, and so on. Once these patterns are located, it will combine these patterns to find more complex patterns. This would include specific shapes of the wings, or what model of car it represents. This way, the brain can fully understand the image. CNN's follow this principle. As explained, they were originally designed for image classification or object detection. However, they also have their use classifying several types of data, such as time series. Its task remains similar for these kinds of data, namely detecting patterns in strings of data. This makes it a useful tool for the data used in this thesis. The general idea of CNN's is based on two types of operations, namely convolutions and pooling, which will now be explained in more detail.

4.2.1. Convolutions

In a convolutional layer, a procedure is performed called weight sharing. This is done by passing a filter k, also called a kernel, through all the point of the input data and on each location the convolution between the filter and the overlapping area of the data is calculated. An example of this calculation is shown in Figure 4.3. Here a filter of size 3 passes through datapoints i_n. The filter consists of 3 weights, similar to its size. These weights are multiplied by their respective input datapoint, and added up afterwards. For this we would get the following formula

$$c_n = w_1 i_{n-1} + w_2 i_n + w_3 i_{n+1},$$

where c_n is the output. Doing this for input data results in a new image called a feature map. Multiple different filters can be chosen simultaneously, which results in multiple different feature maps. The goal of these filters is to detect specific pattern in the input data, where each filter is able to recognize a different pattern. When training a CNN, the user allows the model to learn the weights for each filter.

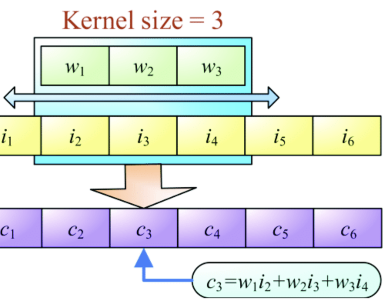

Kernel size = 3
w_1 w_2 w_3
i_1 i_2 i_3 i_4 i_5 i_6
c_1 c_2 c_3 c_4 c_5 c_6
$c_3=w_1i_2+w_2i_3+w_3i_4$

Figure 4.3: A convolution performed on 1 dimensional data [20].

4.2.2. Pooling

After convolutional layers, there also following several pooling layers. These layers take a block of subsequent datapoint, called a receptive field, and translate them into a single neuron. This is shown in Figure 4.4. Note that here only neurons from the receptive field are connected to the neuron in the hidden layer, which is different from the typical neural network as seen in Figure 4.1 where the layers are fully-connected. Many types of pooling layers can be used, but one of the most common is the max pooling layer. Hence, from every receptive field, the maximum value is send to the neuron in the hidden layer. The biggest advantage of pooling layers is that it reduces the amount of neurons, which results in less weights the model needs to learn.

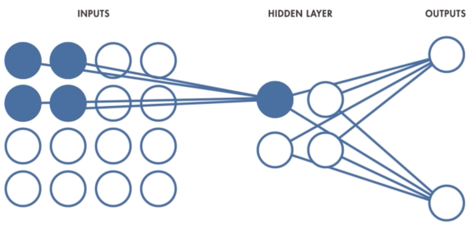


Figure 4.4: Schematic overview of pooling [21].

4.2.3. Combining the layers

CNN's are often combined with additional, fully-connected layers. A full example of a CNN is shown in Figure 4.5. Here it can be seen that a single image goes through several layers of convolution and pooling, ending with a a fully-connected layer and finally classified using softmax.

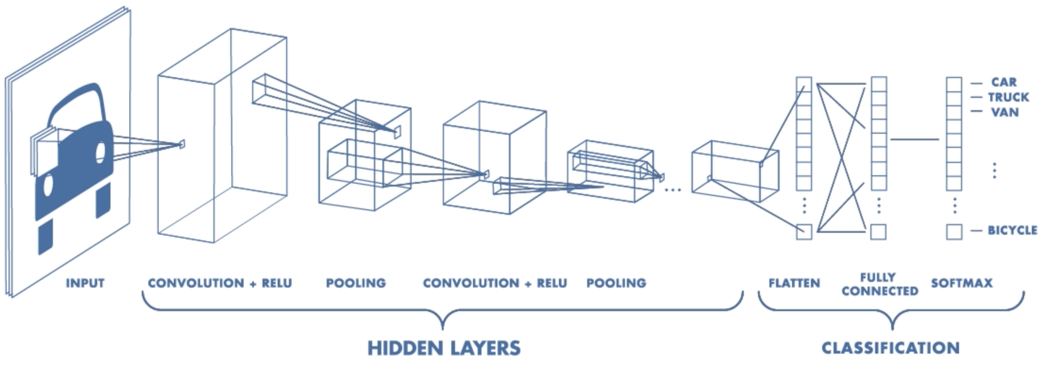


Figure 4.5: Example of a Convolutional Neural Network [21].

This thesis is however not focused on classifying images, but one-dimensional data. CNN works similar for those types of data and is shown in Figure 4.6. It shows that data can be seen as one-dimensional images, which then will be convolved and pooled in a similar manner.

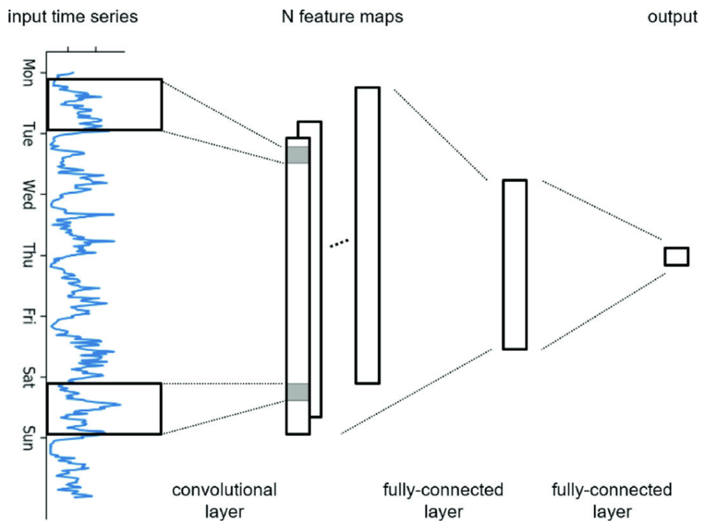


Figure 4.6: Convolutional Neural Network applied to signal data [22].

4.3. Recurrent Neural Networks

Recurrent Neural Networks (RNN) are a type of deep learning models that are especially designed to work with data that have an underlying temporal sequence. Because of that, they are typically used for natural language processing and signal understanding. With this type of data it is important to take into account past information, since the information is treated as a sequence and what has happened in the past has influence on what will happen in the future. Hence, not only the input decides what the outcome, RNN's also take prior inputs and predictions into account. An easy example of this would to let the model write a book based on a list of given words. The words should be set in a certain order to make it a logical sentence. Take for instance the following word list: Bob, Alice, sees, and a dot to finish the sentence. A logical sentence would be "Bob sees Alice." or "Alice sees Bob.", whereas "Bob sees Bob sees Alice Bob sees." is not. That means that when the current sentence is "Bob sees", the network should consider that the sentence already mentions Bob, so it now need to choose Alice. The basic form of a RNN is shown in Figure 4.7. It shows that prediction A of input x_0 is given as extra input for the prediction with input x_1.
The downside of these dependencies is that they are generally short-term. This comes from a problem called

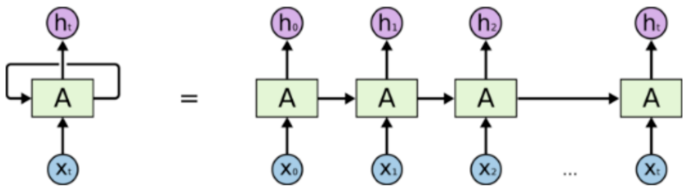


Figure 4.7: Basic form of a Recurrent Neural Network [23].

"vanishing gradients". This problem states that long-term dependencies tend to disappear when training the model for longer input sequences. In the above mentioned example, this would mean that RNN could work for shorter sentences, but it would run into trouble when the amount of words grow, and therefore also the sentences. It would also mean that it has more difficulties to create sentences that would be logical following the previous sentences.

4.3.1. Long Short Term Memory

When working with sequence data such as signals produced by a set of sensors, such as in this thesis, it is important to have a model able to handle long-term dependencies. This is the reason why more complex RNN cells were developed. Long Short Term Memory (LSTM) cells are one of the most popular in the regard. They are built upon the basic RNN cells in which prior information is captured in hidden cells and used to generate outputs of the present time based on past information of the data. The internal functioning of a LSTM cell is depicted in Figure 4.8 where x_t rep resents the input at time t, h_t is called the hidden state at time t and c_t is the cell state at time t.

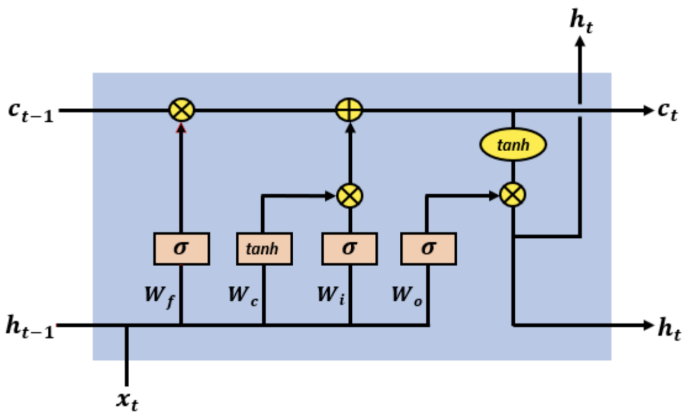


Figure 4.8: Internal structure of an LSTM cell [**?**].

To understand this schematic of a LSTM cell, it is best to go over it piece by piece, starting with the the cell state. This state carries all the long-term dependencies. Important to note hear that this state can only run through easy linear modifications, which means it holds no biases or weights. Hence, it does not experience the problem of vanishing gradients. The hidden state carries the short-term dependencies. It also contains several sets of weights, in the figure represented by the several W's. The input and hidden state pass through several gates, which are themselves regular feed-forward neural networks. These gates are:

- Forget gate
  This gate is responsible for deciding what information must be forgotten from the cell state. To do so, it concatenates the hidden state at time $t-1$, which is h_{t-1} and the current input x_t and calculates a value between 0 and 1 for each element of the cell state c_{t-1}. Its mathematical implementation is given by

$$f_t = \sigma\left(W_f[h_{t-1}, x_t] + b_f\right),$$

  where W_f are the forget weights that are learned during training, b_f the bias term and σ is a sigmoid activation function defined by

$$\sigma(x) = \frac{e^x}{e^x + 1}.$$

By element-wise multiplying the values of f_t with the ones of c_{t-1}, certain elements of c_{t-1} are set to close to 0 while others are kept.

- Input gate
  This gate is responsible for deciding what new information will be stored in the cell state and where. It is composed of two parts. The first part calculates candidate values $\tilde{c_t}$ to be included in the cell state c_t:

  $$\tilde{c_t} = \tanh\left(W_c[h_{t-1}, x_t] + b_c\right),$$

  where W_c are the candidate weights that are learned during training and b_c the bias term. Note here that the tanh functions gives it values between -1 and 1, hence it can give a negative value to a candidate value. This would mean that that candidate should not be stored in the cell gate. The second part decides which parts of the cell state will be updated with the candidate values by calculating the values of i_t as:

  $$i_t = \sigma\left(W_i[h_{t-1}, x_t] + b_i\right),$$

  where W_i are the input weights that are learned during training and b_i the bias term. The values of i_t and $\tilde{c_t}$ are then element-wise multiplied, and that result is added to the c_{t-1} vector that was previously multiplied with f_t. This completes the update of c_{t-1} into c_t determined by the forget and input gates. In other words, the cell state is calculated as

  $$c_t = f_t * c_{t-1} + i_t * \tilde{c_t},$$

  where $*$ represents element-wise multiplication.

- Output gate
  This gate is responsible for deciding which elements of the cell state will be given as the output of a LSTM unit. To do so, it first calculates the values o_t as

  $$o_t = \sigma\left(W_o[h_{t-1}, x_t] + b_o\right),$$

  where W_o are the output weights that are learned during training and b_o the bias term. Meanwhile, the values of the cell state are passed through a tanh function. Finally, those values are element-wise multiplied with o_t, so that only the desired parts of the cell state are output as the new hidden state values h_t:

  $$h_t = o_t * \tanh(c_t).$$

4.3.2. Bidirectional Long Short Term Memory

LSTM cells only look at information it has seen in the past. However, there are cases when the information in the future is also important. Let us look at the sentence "Bob loves apple, because they are healthy". This would indicate that with apple, we mean the fruit. However, in the sentence "Bob loves apple, because they design cool phones", it is clear we are talking about the company Apple. Hence, the information later in the sentence is important in how we classify the word "apple". For this reason, another form of RNN was developed, namely Bidirectional Long Short Term Memory (bLSTM). A schematic overview of this is shown in Figure 4.9. The usage of bLSTM is similar as the regular LSTM, now it only involves an additional hidden and cell state going backwards in time.

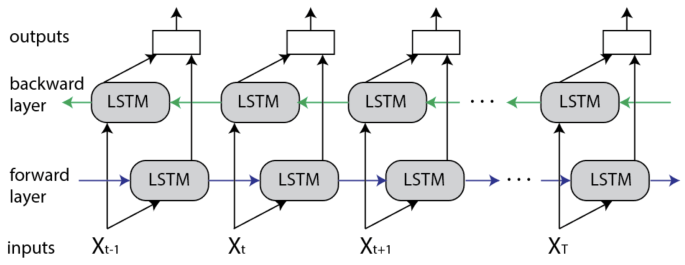


Figure 4.9: Schematic overview of bLSTM [25].

4.4. Additional Deep Learning Concepts

In this thesis, additional deep learning concept were used that need explaining. Those will be explained here.

4.4.1. Dropout Layer

In a dropout layer, a group of randomly selected neurons from the previous layer are ignored. Which group of neurons that is, differs with each step of the training, where a neuron is selected with a fixed probability p. This selected group of neurons is then totally left out of that part of training, including all their connections with other layers. This is shown in Figure 4.10. The goal of a dropout layer is to reduce overfitting. During training, the model is expected to learn many different weights. Neural networks tend to start relying on certain weights, while others are ignored. By forcing the model to ignore certain neurons for parts of the training, it is forced to train all weights as good as possible, thus reducing co-dependency between neurons.

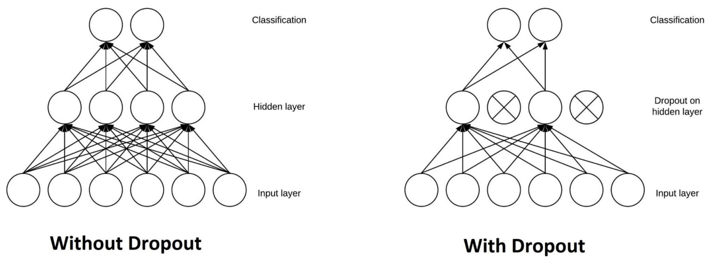


Figure 4.10: The usage of dropout layers. Note that during each step in training, other neurons in the hidden layer will be selected [26].

4.4.2. Loss function

Apart from choosing the types of layers and parameters used in an artificial neural network, it is also important to quantify the error from a wrong classification. This is done by using a loss function. This function is used to quantify the difference between the label output by the model, and the true label. Training a deep learning model updates the weights such that this quantified loss is minimized. Which loss function is chosen, depends on the task the model is expected to perform. These tasks can for instance be regression, binary classification or multi-class classification, where the last one is of use in this thesis.

For this reason, in this thesis the categorical cross-entropy loss function was chosen. It is defined as:

$$J(y,\hat{y}) = -\sum_{i=1}^{N} y_i \log f(\hat{y}_i) = -\sum_{i=1}^{N} y_i \log\left(\frac{e^{\hat{y}_i}}{\sum_{j=1}^{N} e^{\hat{y}_j}}\right),$$

where y_i are the ground truth and $\hat{y}_i$ are the predicted scores for each class i in the N possible classes. The function f refers to the softmax activation function that is applied to the predicted scores at the final layer of the model. The ground truths are one-hot encoded, which essentially means that each possible outcome from the classification is assigned an integer value. For this reason, there is only one non-zero element of the target vector y. Then, the loss function can be rewritten as

$$J(y,\hat{y}) = -\log\left(\frac{e^{\hat{y}_i}}{\sum_{j=1}^{N} e^{\hat{y}_j}}\right).$$

When training an artificial neural network, as it will be explained below, the gradients of the loss function with respect to the weights/parameters are needed. For the categorical cross-entropy loss function, its partial derivative with respect to an output $\hat{y}_l$ is found as

$$\frac{\partial J}{\partial \hat{y}_l} = -\delta_{il} + f(\hat{y}_l),$$

where δ_{il} is the Kronecker delta function. Having this expression of the partial derivative of the loss function with respect to each output of the network, it is possible to obtain the partial derivative of the loss function with respect to any particular weight w as

$$\frac{\partial J}{\partial w} = \sum_{i=1}^{N} \frac{\partial J}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial w}.$$

The terms $\frac{\partial \hat{y}_i}{\partial w}$ will change depending on the types of layers and their activation functions. The weights from the initial layers will also require that term to be expanded using the chain rule. This way the error quantified by the loss function will be able to backpropagate through the layers until it finds the corresponding weight. However, since a neuron is ultimately computing a function of the form $y = \sigma(w^T x + b)$, those inner derivatives can be easily computed as

$$\frac{\partial y}{\partial w} = \sigma'(w^T x + b)x.$$

Note that those internal derivatives, depend on the activation function σ used by each neuron and by their inputs x.

4.4.3. ADAM optimization algorithm

As mentioned, training an artificial neural network consists of finding weights such that the loss function is minimized. These models contain a large amount of weight, which makes it useful to find a way to efficiently decide which weights should be altered and how much. To do this, artificial neural networks rely on the backpropagation algorithm, which uses the partial derivatives of the loss function with respect to each one of the weights. Hence the gradient of the loss function is needed. When these partial derivatives are obtained, the weights w are updated as follows:

$$w := w - a\frac{\partial J}{\partial w},$$

where a is called the learning rate. We call this the gradient descent algorithm and it is used to effectively train artificial neural networks. In gradient descent, the weight is updated in the opposite direction of the partial derivative of the loss function with respect to the weight. The amount of this update is determined by a.

Gradient descent can be used to train neural networks, but it usually converges slowly. To speed up the training proposed of neural networks, additional algorithms have been proposed that improve the basic functionality of the gradient descent. One of the most widely used algorithms is called the ADAM optimization algorithm, which stands for Adaptive Moment Estimation. It uses two elements to converge faster: momentum and adaptive learning rates.

- Momentum
  Gradient descent can be understood as taking steps of a certain length in direction opposite of the steepest path. However, the loss function can be very complex and have extremes. To address this, the optimization algorithm can also be seen as a ball rolling down the loss function. This ball will have some inertia and will be faster when the ball follows a somehow regular path. This dampens the learning and makes it less oscillating. Hence the optimization algorithm moves fast when we go in a similar direction, but it goes slower when it finds resistance. Therefore wights with direction changing gradients are updated slowly, while those with gradients in similar directions are updated faster. This momentum can be quantified as follows

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\frac{\partial J}{\partial w_t},$$

where β_1 is a hyperparameter between 0 and 1. Here the momentum at time t takes a weighted sum of the momentum at time $t-1$ and the gradient at time t. This is called an exponentially weighted average. Afterwards, a weight w_t is updated as

$$w_{t+1} = w_t - a m_t.$$

- Adaptive Learning Rates
  The adaptive learning rate property is extracted from another optimization algorithm called RMSprop, which stands for Root Mean Square Propagation. This algorithm starts with an exponentially weighted average of the squared gradients, which looks as follows

  $$v_t = \beta_2 v_{t-1} + (1-\beta_2)\left(\frac{\partial J}{\partial w_t}\right)^2, \tag{4.1}$$

  where β_2 is a hyperparameter between 0 and 1. By taking the square of the gradients, it is made sure v_t stays positive. Note that this is calculated for each parameter to be optimized, so each parameter has its own second momentum and it depends exclusively on itself and the history of its gradients.

  The second part consists of finding an appropriate learning rate each step. As before the moving direction is opposite of the gradient, but now the step size is determined by the second moment calculated as in (4.1). The learning rate a is divided by the square root of that second moment, so a is modified for each one of the parameters independently. A parameter with a large exponential average will then be updated with small steps, while a parameter with small exponential average will be updated faster. Hence RMSprop can be said to use different learning rates for every weight w_i. Those learning rates are continuously being modified based on the gradients that have been previously computed for each weight. This is defined as

  $$w_{t+1} = w_t - a\frac{1}{\sqrt{v_t}+\epsilon}\frac{\partial J}{\partial w_t},$$

  where ϵ is a very small value that is added to avoid division by zero.

The ADAM optimization algorithm combines these ideas from momentum and Adaptive Learning Rates. An ADAM step starts by getting the gradients of the loss function with respect to the weights:

$$g_t = \frac{\partial J}{\partial w_t}.$$

It then calculates the first and second moments of the gradients determined by parameters β_1 and β_2 respectively:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2.$$

These two moments are generally initialized at zero, which results in them being biased towards zero. To avoid this, the moments are bias-corrected as follows

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t},$$

where t is the time step or an iteration number. Finally, ADAM updates the parameter by combining both bias-corrected moments with their respective momentum for $\hat{m}$ and adaptive learning rate for $\hat{v}$. And thus the weights are updated as

$$w_{t+1} = w_t - a\frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon}.$$

The full ADAM algorithm is presented in Figure 4.11. In that figure (which was taken from the original paper), the loss function is represented by f and the weights by θ. Usually the hyperparameters are chosen as $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. The learning rate is still denoted by a and it should be manually selected.

```
Require: $\alpha$: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters $\theta$
Require: $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while $\theta_t$ not converged do
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return $\theta_t$ (Resulting parameters)
```

Figure 4.11: ADAM optimization algorithm [27].

5 Methodology

In this section, the building, training and usage of the deep learning model will be explained. This model has the goal of classifying several different football related activities. Important here is to note that a big part of this section was already discussed in [2], and will therefore not be explained into the finest detail. This section will not contain any training results. These will all be presented in section 6.

5.1. Activity recognition

Before using deep learning models to classify large datasets, these models should be trained. In this section, this training will be explained, together with what models will be used and how the data is prepared. The methodology used to build these model is shown in Figure 5.1.

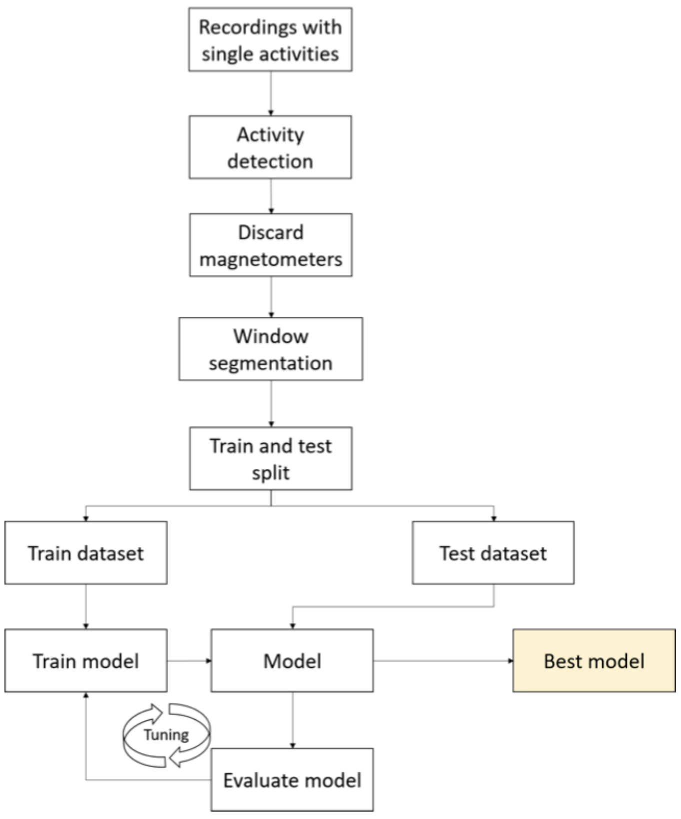


Figure 5.1: Training phase diagram [2].

5.1.1. Window segmentation
The first step into training a good deep learning model, is creating windows in the training data. As mentioned in section 3.2.1, only 40 to 45 recordings are available per football activity. Since deep learning models become more accurate when it is trained with more different training data, this could cause the model to become inaccurate. Moreover, this could give the possibility to take the length of an activity into account and could cause wrongful classifications. This for instance would be that a run of 10 meters would be different activity than a run of 50 meters. Therefore it is beneficial to create several windows of the same length within one recording to artificially create more training data and to make the training indifferent to the length of the activity. The way of splitting up recordings into multiple windows will create problems later on, but these will be explained in sections 5.1.3 and 5.1.4.

Creating the windows start by first applying activity detection to each recording, as explained in section 2.3.2. This results in a more robust dataset in the sense that the recordings of the activities were clean of low activity intervals. This process was crucial to have better models, since the presence of these intervals in the training samples could confuse the model by making it learn features and patterns from the parts with low activity and not from the actual activities that were meant to be recognized. As mentioned in section 3.1, these recordings consist of 5 sensor locations, each with 2 modes, namely the accelerometer and gyroscope, in 3 axis. This results in a total of 30 different signals per recording. Note that the raw signal data is used in these recordings, without any kind of pre-processing.

After the activities are detected in the recordings, the recordings were split in windows with a length of 1 second each. This 1 second is based of the analysis done in section 2.3.2. The 1 second allows us to capture the explosive activities, such as shots and jumps. Since periodic activities have contain patterns of repeated data, the 1 second window allows us to capture that pattern. A 75% overlap was used in the windows. This was done to create more training data and to temporal dependencies. Hence every 250 milliseconds of a recording a new interval of 1 second was extracted. As an example, a recording of 1.5 seconds long would give us 3 windows, namely 0 to 1000 milliseconds, 250 to 1250 milliseconds and 500 to 1500 milliseconds.

5.1.2. Model selection
As seen in section 2.1, both CNN and RNN are used for Human Activity Recognition (HAR), whereas sometimes also a combination of the two is used. Interesting to see is that all three of these structures can have promising results. This gives rise to the questions which of these structures is the optimal to use for Football Activity Recognition (FAR). This question might be easier easier asked than answered. In the field of machine learning, there is widely known theorem called the *No-Free-Lunch Theorem*. This theorem states that "for every type of learner, there exists a task on which it it fails, even though that task can be successfully learned by another learner" [28]. Hence, a specific type of model could obtain good results with FAR, whereas the results for HAR using that model could be inaccurate, and vice versa. This problem could even arise within FAR itself, which can also be seen later in section 6. Some models work good for recognizing turns, whereas other models are better at recognizing sprint speeds. For this reason, multiple models of CNN, RNN and combinations of the two will be considered.

Convolutional layers
As mentioned, one recording consists of 30 different signals. This gives rise to the question how the convolutions should be set up. They can be per sensor, per signal or even similar for all sensors. Therefore, all these options and variations are tried and trained. It is important to note that, in this thesis, even if all the convolutions are theoretically two-dimensional, some of them are referred as one-dimensional and some others as two-dimensional to distinguish among them. By one-dimensional convolution, we refer to convolutions in which the spatial dimension of the filter is 1, so that each signal is processed alone and the filters do not process more than one signal at the same time. By two-dimensional convolutions, we refer to convolutions in which the spatial dimension of the filter is more than 1, so that several signals are convolved at the same time. The following variations of convolutions were built:

- 1DCNN weight sharing
  One-dimensional convolutions involve applying the same filters to all signals. To perform this type of convolution, filters of size $1 \times m$ are employed, with m representing the number of timesteps used in

the convolution. The "1" refers to the fact that each signal is convolved separately, and weight sharing means that the same filters are used for all signals. It's worth noting that each sensor is comprised of three signals, corresponding to the X, Y, and Z axes of the sensor. For each signal, the convolution is performed using the same set of k filters. This variation can be observed in Figure 5.2.

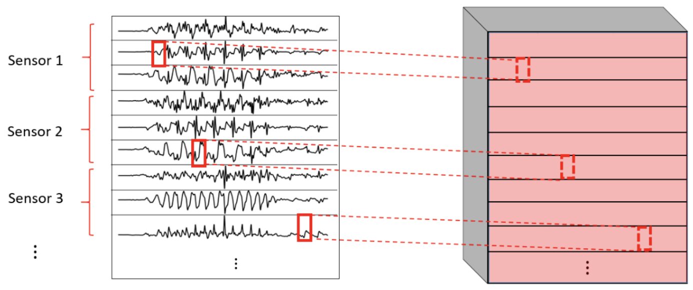


Figure 5.2: 1DCNN weight sharing convolution [2].

- 1DCNN per sensor
  One-dimensional convolutions can also use the same filters for all signals of the same sensor, but with different filters for each sensor. For this type of convolution, filters of size $1 \times m$ are used, with m representing the number of timesteps used in the convolution. The "1" indicates that each signal is convolved independently. However, each sensor has its own set of k filters, which are not shared among the sensors. This approach is based on the idea that since each sensor can exhibit completely different patterns than another sensor, it's better to have filters specifically optimized for it. As a result, the features extracted per sensor are more tailored to the unique characteristics of that sensor. The convolutions are carried out for each sensor using a distinct set of k filters, although the same set of filters is used for the three axes of the same sensor. In the figure, the different sets of filters are represented by different colors. It's worth noting that this approach results in a larger model compared to the 1DCNN weight sharing, as there are now $k \cdot NumSen$ filters to be learned instead of just k. This variation is illustrated in Figure 5.3.

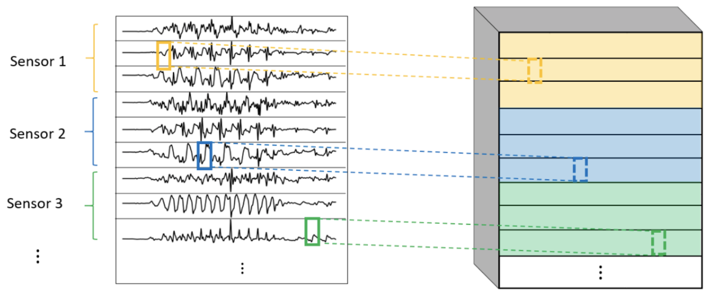


Figure 5.3: 1DCNN per sensor convolution [2].

- 1DCC combined
  Another variation of one-dimensional convolutions is the combination of 1DCNN weight sharing and 1DCNN per sensor. This approach involves performing both types of convolutions and concatenating their results. The goal is to leverage the advantages of both previous convolution types. Figure 5.4 illustrates this variation.

- 2DCNN weight sharing
  Another type of convolution is the two-dimensional convolution with the same filters for all sensors. This approach involves using filters of size $3 \times m$, with m representing the number of timesteps used in the convolution. The "3" indicates that the three axes of the same sensor are convolved together. To extract specific patterns from each sensor, a spatial stride of 3 is used to convolve each sensor independently. Weight sharing means that the same filters are used for all sensors. Unlike one-dimensional convolutions,

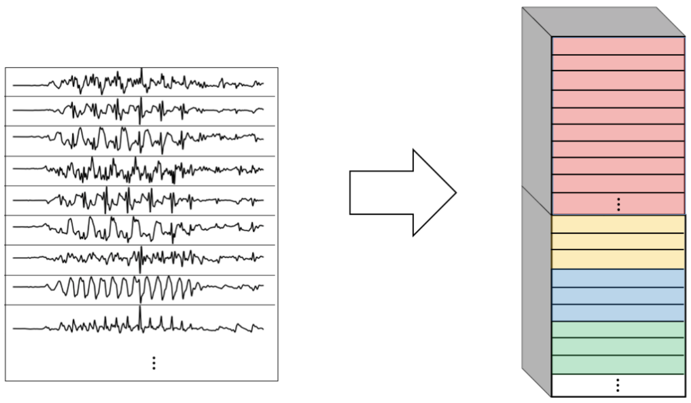


Figure 5.4: 1DCNN combined convolution [2].

two-dimensional convolutions use signals from all three axes of the same sensor, resulting in a smaller feature map but larger filters. Figure 5.5 shows this variation.

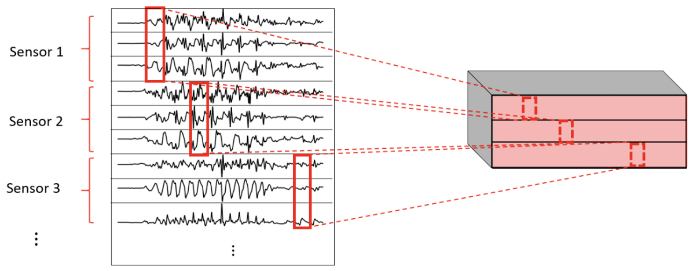


Figure 5.5: 2DCNN weight sharing [2].

- 2DCNN per sensor
  Another variation of two-dimensional convolutions is the use of different filters for each sensor. This approach involves using filters of size $3 \times m$, where m represents the number of timesteps used in the convolution. The "3" means that the three axes of the same sensor are convolved together. To extract specific patterns from each sensor, a spatial stride of 3 is used to convolve each sensor independently. This variation follows the same idea as 1DCNN per sensor, where each sensor can present unique patterns and thus deserves its own set of filters. By combining the information of all three axes in a convolution operation, the features extracted for each sensor are optimized for that specific sensor. For each sensor, a different set of k filters is used, represented by different colors in the figure. However, this approach results in a larger model compared to 2DCNN weight sharing since it requires learning $k \cdot NumSen$ filters instead of just k. Figure 5.6 illustrates this variation.

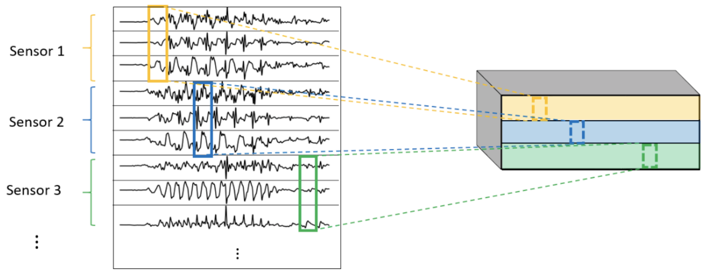


Figure 5.6: 2DCNN per sensor [2].

- 2DCNN all sensors
  Two-dimensional convolutions are used to extract features from input data with spatial relationships, such as images or multi-sensor signals. In this specific variation, the convolutions are made across all the sensors and signals at once, using filters of size $NumSen \times m$. This means that the filters are as wide as the number of sensors and as tall as the number of timesteps used in the convolution. By performing convolutions that include all the sensors in the operation, it is expected to extract features that capture information about the relationships between all the sensors and signals. However, this approach results in a smaller feature map, as the spatial dimension has a size of 1. This variation is illustrated in Figure 5.7.

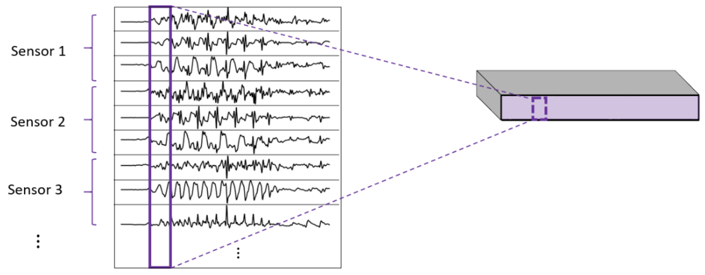


Figure 5.7: 2DCNN all sensors [2].

- 2DCNN combined
  This variation involves using a combination of three different types of 2D convolutions - 2D convolutions with weight sharing across all sensors, 2D convolutions with different filters for each sensor, and 2D convolutions made across all the sensors and signals at once. The feature maps resulting from these three types of convolutions are concatenated, resulting in a larger model than any of the individual approaches. Figure 5.8 shows a diagram of this combined approach.

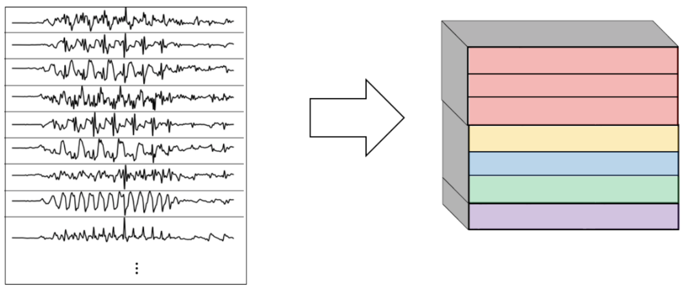


Figure 5.8: 2DCNN combined [2].

Recurrent layers
For the RNN layers, only LSTM's and bLSTM's are used. This is done since these models are able to remember trend seen throughout the recording, while regular RNN's only remember the very recent trends. LSTM is able to remember everything that had previously happened in a recording, while bLSTM is also able to "remember" what will happen later on in the recording. This makes it interesting to see how these layers could affect the overall performance of the model.

General architectures
Using these CNN's and RNN's, several models will be build and tested. They are based on solely CNN's or RNN's, or a combination of the two. The full models, including the chosen parameters, are listed in Appendix A.

The CNN-based models followed a general structure illustrated in Figure 5.9. The input signals were processed by a combination of a convolutional layer and a max pooling layer, repeated twice. Next, a third convolutional layer was used to obtain a 2-dimensional tensor, which was then flattened into a one-dimensional vector.

The resulting vector was fed into a fully connected layer, followed by an output fully connected layer with a softmax activation function to generate probability scores for the 7 activities. The first two convolutional layers, displayed by Conv* varied depending on the chosen convolutional variation. To prevent overfitting, dropout layers were added before each fully connected layer.

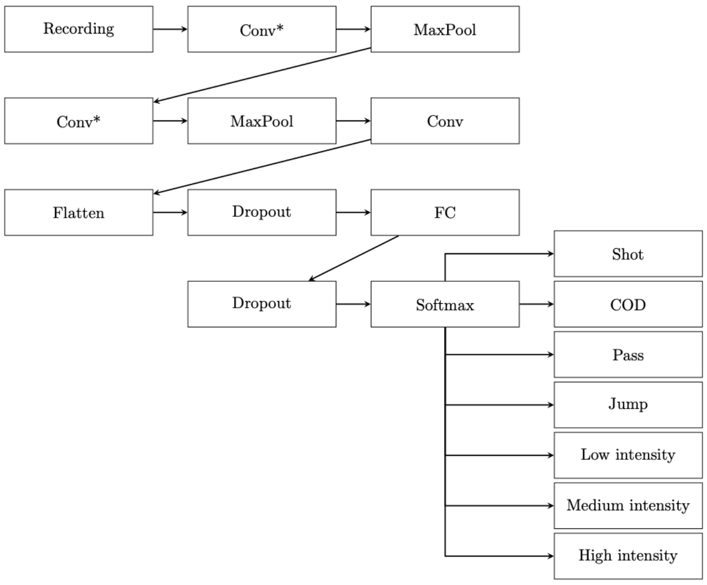


Figure 5.9: General architecture for models based on CNN's. Conv* layers mean the usage of all the variations of convolutions previously explained. Conv is a regular convolutional layer. FC means Fully Connected Feed Forward Neural Network.

The RNN-based models followed a general structure illustrated in Figure 5.10. The input signals were processed by a RNN-based layer, repeated twice. The first RNN layer returned the entire sequence, resulting in an output with the same number of timesteps as the input signal. The second RNN layer only returned the output value for the final timestep, resulting in a flattened one-dimensional vector. The resulting vector was fed into two fully connected layers, followed by an output fully connected layer with a softmax activation function to generate probability scores for the 7 activities. The RNN layers depicted with an asterisk varied depending on the chosen RNN units, so LSTM or bLSTM. To prevent overfitting, dropout layers were added before each fully connected layer.

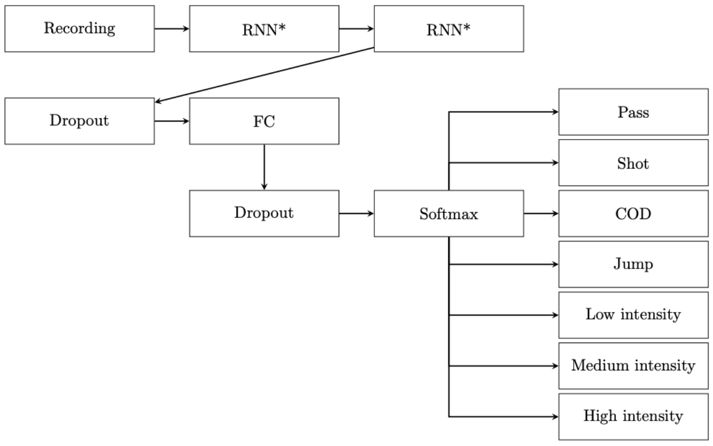


Figure 5.10: General architecture for models based on RNN's. RNN* layers represent either LSTM's or bidirectional LSTM's. FC means Fully Connected Feed Forward Neural Network.

The combined CNN and RNN models followed a general structure as depicted in figure 5.11. The input signals were first passed through several layers of convolutional operations. The output of these operations was a three-dimensional tensor, but the recurrent units required a two-dimensional input. To address this, an additional convolutional layer was added, followed by a reshape layer that rearranged the dimensions and prepared the tensor for the recurrent part. Once the tensor was two-dimensional, it was ready to be processed by two layers of LSTMs or bLSTMs responsible for capturing temporal dependencies of the extracted features. The output of the RNN layers was a one-dimensional vector, which was passed through a fully connected layer and finally an output fully connected layer with a softmax activation function to obtain the probability scores for each of the 7 activities. To prevent overfitting, dropout layers were applied before each fully connected layer.

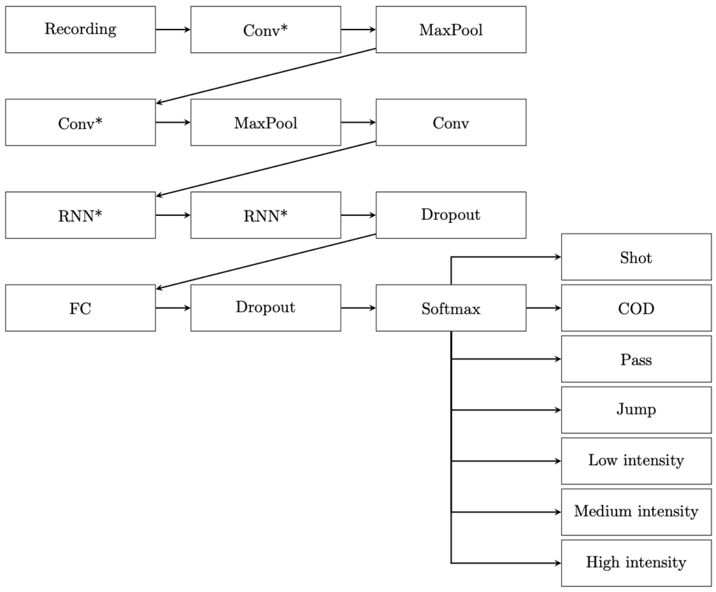


Figure 5.11: General architecture for models based on combination of CNN's followed by RNN's. Conv* layers mean the usage of all the variations of convolutions previously explained. Conv is a regular convolutional layer. RNN* layers represent either LSTM's or bidirectional LSTM's. FC means Fully Connected Feed Forward Neural Network.

5.1.3. Training and performance

The first problem that follow from the window segmentation as explained in section 5.1.1, is that it causes the dataset to be unbalanced. An unbalanced dataset could cause the model to be biased towards the more common activities. It has seen this ones more often, so it better trained to find those. Hence, when the model is in doubt which activity it should classify as, it will be more likely to choose a model which it has seen more often. To prevent this problem, it is beneficial to first balance the data. This can be done in various ways, such generating new artificial data with techniques such as SMOTE [29] or standard re-sampling techniques. However, these approaches are troublesome due to the high complexity of the data. For the re-sampling, there is also another reason. One of the goals of the model, is for it to be as "participant-robust" as possible. This means that the trained models should perform well for everyone, independent of whose data was used to train the model. If data is re-sampled, specific features of a participant used in the obtained training data, could become too important and would thus decrease accuracy. For this reason, it is chosen to use under-sampling. This indicates that each activity loses windows until they all have the same amount of windows as the least frequent one. This can be seen in Figures 5.12 and 5.13. It can seen in Figure 5.12 that activity 2 has more than 2000 windows, whereas activities 6 and 5 have around 250 windows. In Figure 5.13, it can been that all activities now have around 250 windows.

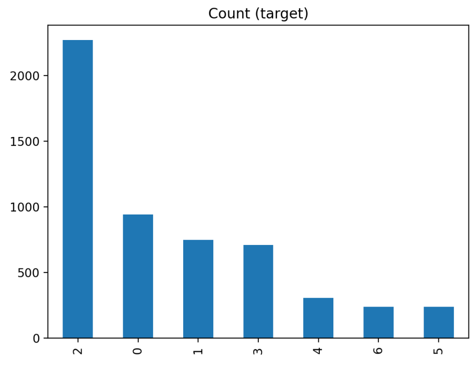


Figure 5.12: Unbalanced dataset for training.

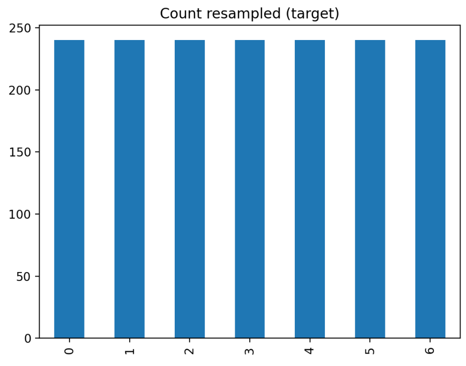


Figure 5.13: Balanced dataset for training.

To train the models, the ADAM optimization algorithm was used. As mentioned in section 4.4.3, in all cases, the chosen parameters were $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. The learning rate a was set to decay during the learning phase using a learning rate scheduler. Hence the weights are updated fast in the initial phases of the training, but it is slowed down once the weight come close to their optimal value. This resulted in a better and faster training scheme. These learning rate schedulers for the models are defined as

- When a model does not contain a RNN component, the learning rate was initialized at 0.001. After every 10 epochs of training, it was reduced to 0.00075.
- When a model solely contains a RNN component, the learning rate was initialized at 0.0001. After every 10 epochs of training, it was reduced to 0.00005.
- When a model contains both CNN and RNN components, the learning rate was initialized at 0.00005. After every 10 epochs of training, it was reduced to 0.0000375.

As explained before, the dataset was divided in two subsets. The first subset is used as train data. This data is used to train the models and optimize the values of the weights, and consists of 70% of the dataset. The other subset is test data. This is used at the end of each epoch to test how well the model and the weights perform at that time. The test data consists of the remaining 30%.

Once the models are trained, the models will be evaluated. This will be done by dividing the amount of correctly classified samples by the total number of classified samples. Later on, it will also be interesting to look at the confusion tables of a model. These tables show per activity how they were classified. Hence, we do not only see how much percent was classified correctly from a certain activity, but also which activity was guessed for the wrongly classified ones.

5.1.4. Change of Direction recognition

The first problem to tackle compared to [2], is adding Change of Directions (COD's) to the list of classifiable activities. Being able to classify these activities are of great importance in injury prevention. According to [30], "a large proportion of Anterior cruciate ligament (ACL) injuries in sports occur during non-contact change of direction manoeuvres". This kind of injury can have short-term and long-term effects, both physically as psychologically. A good example of this is Brazilian former football player Ronaldo Luís Nazário de Lima, who suffered an ACL injury on 3 different occasions. This cost him almost 800 days of recovery, which let him to miss over a 100 games of football [31]. This makes COD's while running a vital activity in football.

A next question one could ask, is what can be defined as a COD. Are 90 degree COD's, also called cuts, similar to complete 180 degree COD's? According to [32], these COD's are indeed similar in a way. They give all directional changes between 60 and 180 degree a so called Red sign, which basically means "Slam the brakes". Hence, it is strongly recommended to limit ones speed substantially prior to the COD. Since 60 degree cuts, 180 degree COD's and everything in between is classified similarly, they will also be classified similarly in this thesis.

Simple implementation

The first way to add COD's into the list of activities, is to simply add the recordings with COD data to the training data. One such recording is shown in Figure 5.14. This recording shows a COD in the middle of a jog. The part where the COD takes place is clearly visible around time 4.5. Simply labeling this recording as a COD gives rise to a big problem, namely the window segmentation as described in section 5.1.1. The window segmentation splits up this recording into several windows and labels them all as COD's. However, the part where the COD takes place is only a small portion of the recording. This gives as a result that windows only containing jog data is labeled as a turn, which results in both COD's and jogs to be negatively influenced in classifying these activities.

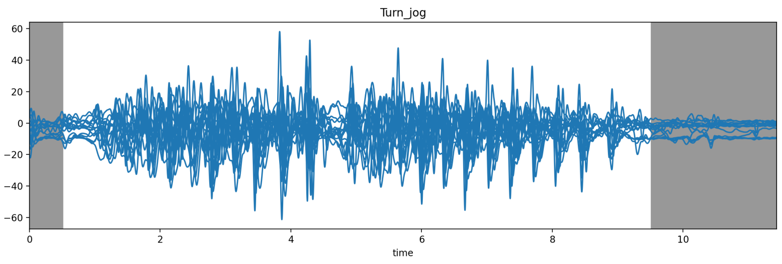
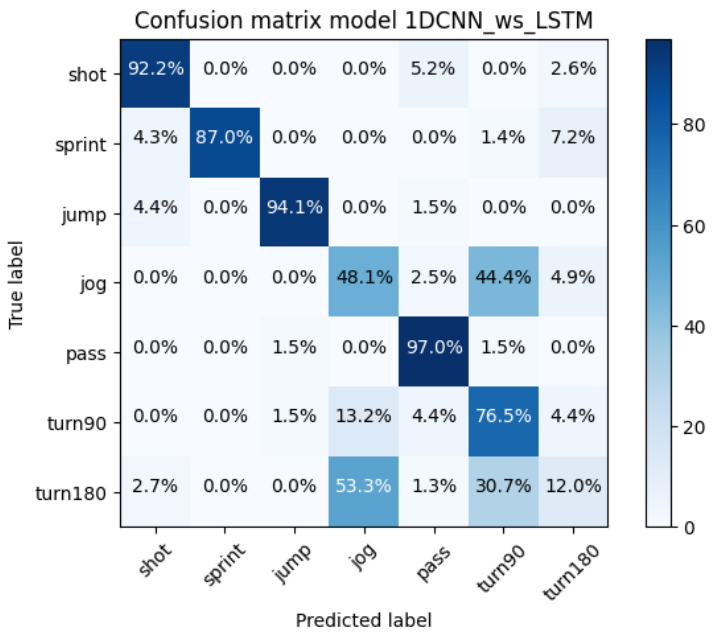


Figure 5.14: Raw accelorator data of a jog containing a 180 degree COD.

This negative effect is shown is in Figure 5.15. This was the model with the highest test accuracy, of 0.9056. Even though that accuracy is not so bad, it is clear that the accuracy of activities such as sprints, jogs and COD's are taking a big hit. Not even half of all jog data is classified correctly. This makes the argument that simply adding COD's to the list of activities does not work properly. Note that in this example 90 degree COD's and 180 degree COD's are trained separately, even though we now consider them as the same activity. That they were trained here individually, comes from the fact that in the training data 90 and 180 degree COD's were handled separately.



	shot	sprint	jump	jog	pass	turn90	turn180
shot	92.2%	0.0%	0.0%	0.0%	5.2%	0.0%	2.6%
sprint	4.3%	87.0%	0.0%	0.0%	0.0%	1.4%	7.2%
jump	4.4%	0.0%	94.1%	0.0%	1.5%	0.0%	0.0%
jog	0.0%	0.0%	0.0%	48.1%	2.5%	44.4%	4.9%
pass	0.0%	0.0%	1.5%	0.0%	97.0%	1.5%	0.0%
turn90	0.0%	0.0%	1.5%	13.2%	4.4%	76.5%	4.4%
turn180	2.7%	0.0%	0.0%	53.3%	1.3%	30.7%	12.0%

Figure 5.15: Confusion matrix where the "1DCNN weight sharing LSTM" model is trained with additional 90 and 180 degree COD's.

Manual correction

The first and most simple way is to manually select the windows that contain the COD activity and only label those as COD's. As could be seen in Figure 5.14, with the human eye it is easy to find these regions, and therefore also gives high accuracy. The obvious downside of this is that it is much and tedious work to perform, especially when using it on a larger scale. The model should be trained in such a way that there is minimal need for human interference, and this approach obviously needs a bunch of human interference. It was still performed however, to check whether COD's could actually be trained by the models, and such that the following approach could be tested and see how it performs.

Extremes height difference

The most efficient way to find which windows should be classified as COD's, was to look at the height difference between the minima and maxima of the sensor data. First the method will be explained step for step, afterwards it will be shown in an example. The method is as follows

1. Take the raw accelerator data from the left and right thigh, both solely in the X and Z axis.
2. For all signals, perform the following computations
   (a) Create a list of all minima and maxima of the signal, in chronological order.
   (b) Compute the absolute value of the differences between all consecutive minima and maxima.
   (c) Compute the mean of the differences.
   (d) Output the largest interval in which the differences are below 1.25 multiplied by the mean. In this step we exclude the first and last 25 extremes, since these are generally lower resulting from starting or stopping to run.
3. Take the intersection of the four found intervals.
4. Label all windows that intersect with this final interval as COD's.

To show how this procedure works, we will look at an example. In Figure 5.16, an example of a COD is shown. Here only the data from the X and Z axis from the right and lift thigh are shown. From this image, it is clear to see that the COD takes place around time 3.5.

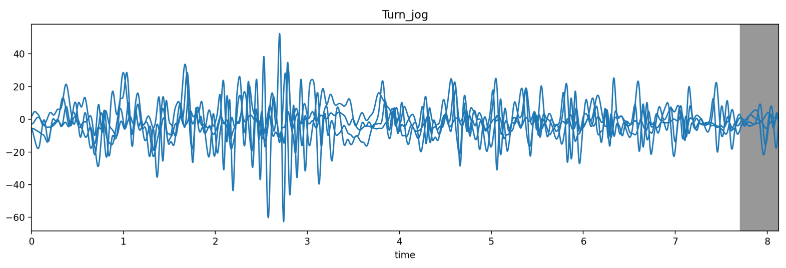


Figure 5.16: Raw accelorator data from a jog with an 180 degree COD in the middle. Here only the data from the thighs in the X and Z axis are presented.

If we now apply the above method to the X axis of the left thigh signal, we find Figure 5.17. The horizontal line represents the mean difference of the maxima and minima. The largest interval where the differences are below this horizontal line, is between timesteps 1400 and 2000. When this process is repeated for the Z axis and for the X and Z of the right thigh and the intervals are intersected, we find the interval of timesteps 1607 to 1894. If then the timesteps are translated back to seconds, the found time interval is from 3.2 seconds to 3.8 seconds. Checking this with Figure 5.16 and when using the labelling found by manual correction, this is indeed the correct interval. Important to note is that this method is only used in the training phase. Once the deep learning model is trained, classification of COD's is solely done by those models.

This method indeed proves to be an accurate alternative for manually correcting and choosing the windows. Applying this technique to the dataset used for this thesis gave a success rate of 98.6% compared to the manually corrected windows. Hence, 97.6% of all windows classified as COD's by this method, were correctly classified.

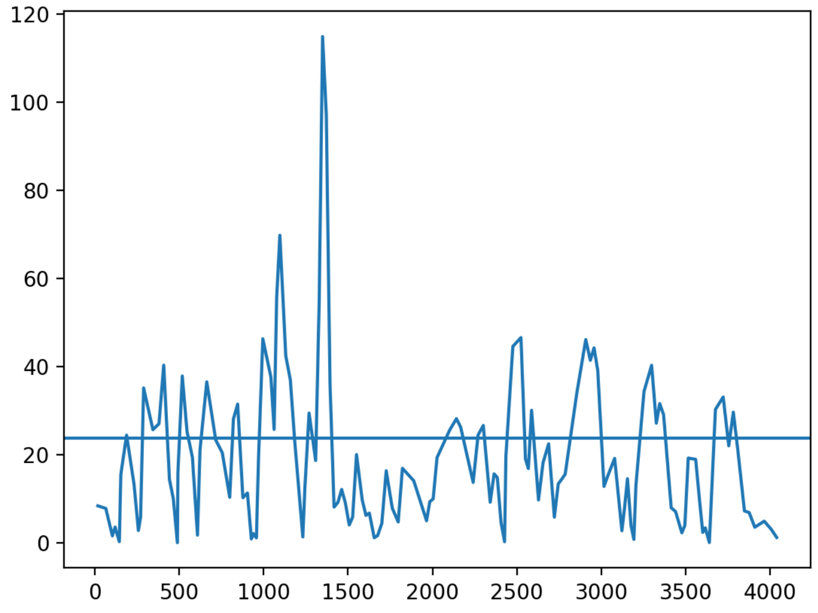


Figure 5.17: The extremes height difference method applied to the X axis data from the left thigh.

5.1.5. Run intensity recognition

In 2, only two types of running were introduced, namely jogging and sprinting. There are however two problems with this approach. First of all, It limits the options a lot. There is a whole lot of running the would be classified as faster than jogging, but slower than sprinting. These types of running are now ignored. Another problem is that jogging and sprinting are different per person. As an example, the fastest human speed ever recorded was from Jamaican sprinter Usain Bolt in 2009 with a top speed of 44.2 kilometers per hour [33], while the average human top speed lies around the 29.3 kilometers per hour [34]. This shows that running fast or slow is dependent on the person or even the situation. A football player that has played for over an hour, is less likely to reach high sprint speeds than a player who just started playing.

To classify all different sprint speeds, literature gives us 4 different categories [35].

1. Jogging (< 15 kilometers per hour)
2. Running (15 – 19.8 kilometers per hour)
3. Running at high speed (19.8 – 25.1 kilometers per hour)
4. Sprint (> 25.1 kilometers per hour).

Since the training data used in this barely has any running data with a higher speed than > 25.1 kilometers per hour. Therefore, in this thesis we will classify 3 different running intensities, namely

1. Low (< 15 kilometers per hour)
2. Medium (15 – 19.8 kilometers per hour)
3. High (> 19.8 kilometers per hour).

Before the deep learning models can train on the data to learn these intensities, the data should be labeled with the correct intensity. To do this, three different approaches were used.

Mean of the signal heights

The first method used for assigning intensities was based on signal heights. In Figure 5.18 a jog is represented, while in Figure 5.19 a sprint can be seen. Looking at the Y axis, it can be seen that for the jog, the maximal signal height is around 40, while for sprinting this signal height can be up to 75. This would indicate that low intensities runs generally have lower signal value. Research into the data also supports this. When looking at the mean of the signal data, it can be seen that for low intensity runs this lies around the value of 21.6, for medium at 27.8 and for high at 34.8.

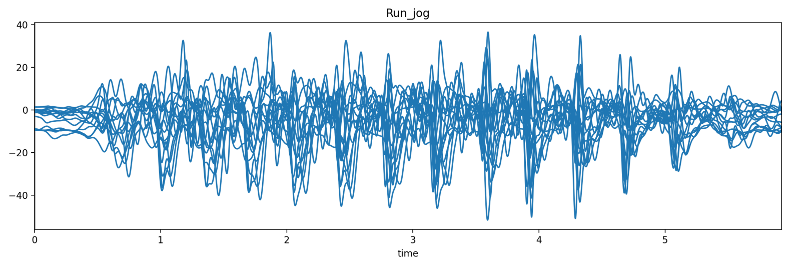


Figure 5.18: Raw accelorator data from a jog.

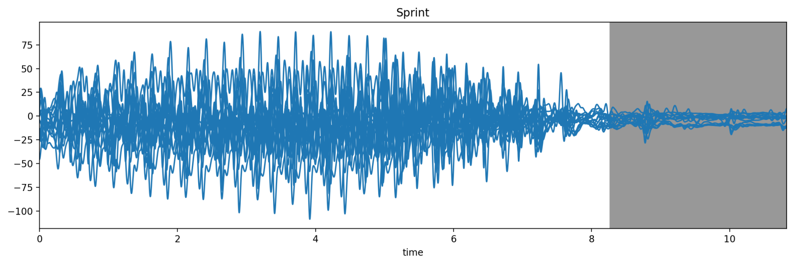


Figure 5.19: Raw accelorator data from a sprint.

This method could be used by doing the following procedure

1. Take the accelerator data from all 5 sensors and compute the norms from the X, Y and Z axis.
2. Compute the means of these norms, and use that to compute the mean of these means.
3. Classify the intensity according to the following
   (a) When the mean is below 24.5: classify as low intensity.
   (b) When the mean is between 24.5 and 33: classify as medium intensity.
   (c) When the mean is higher than 33: classify as high intensity.

Using this method gives high results, as can be seen in Figure 5.20. It gives a test accuracy up to 94.58%. However, it is recommended not to use this method. This comes from the fact that classifying this way does not give a lot of information to the user. It would only suggest that the acceleration of certain sensors is high for a participant, but it does not say much about his speed or the use of the muscles. For example, when a tall person has the exact same acceleration mean as a smaller person, he would still run faster, since he can take bigger steps.

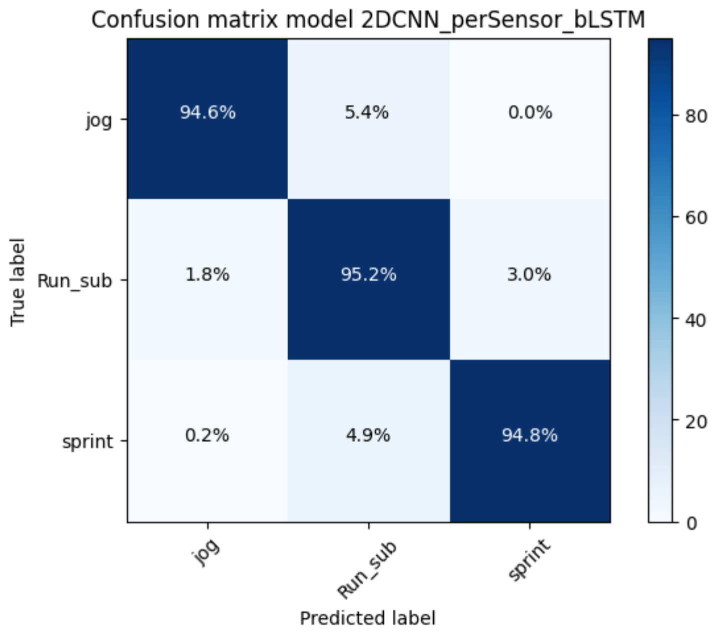


Figure 5.20: Confusion matrix of deep learning model trained only to classify different run intensities.

Distance divided by time
Another approach that could be used, is the use of basic physics. In the training data, there is noted that what distance the participants had to run. If this distance is divided by the amount of time the activity took to complete, one would get the average speed that the participant ran. This however, has many inaccuracies. Firstly, when running a certain distance, the participant is mostly not able to stand still right after finishing running. There is always a short stopping zone, where the runner is slowing down but still moving forward. This would result in a higher ran distance than initially assumed. another problem is the time. As seen in section 2.3.2, every running period starts and end with a short period of standing still. Even though activity detection eliminates most of this standing still data, there will always be a little bit left. This would result that the time taken could be longer than it actually was. Lastly, the run activity is a complete activity from stop to stop. This means that the participant starts standing still, then accelerates, then reaches top speed, and then slows down again. This would indicate that the average speed calculated by this method is substantially lower than the top speed.

Use of VICON data
The final method is using VICON data to determine the speed. As mentioned in section 3.1.2, VICON data gives highly accurate data of the locations of specific sensors, with an X, Y and Z axis. The locations of these sensors is shown in Figure 5.21. Here we are mostly interested in the LPSIS and RPSIS locations, since these lay closest to the pelvis.

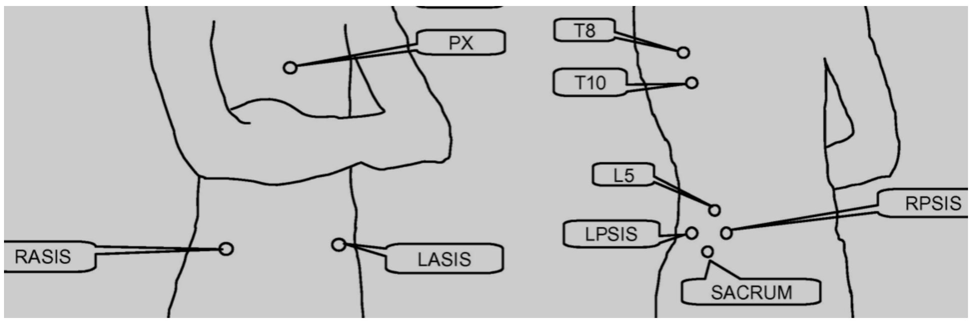


Figure 5.21: Placements of the VICON markers, where solely the LPSIS and RPSIS are used [36].

This method would go as follows

1. Take Y axis of the LPSIS and the RPSIS data. The Y axis is needed, since when obtaining the training data, the Y axis was set as going forward. This could differ when doing different experiments or data collections.
2. For every timestep, compute the centre of gravity between the LPSIS and RPSIS data. This should result in the location of the pelvis.
3. Per timestep, compute the difference of this locations and divide it by the length of the timestep. This gives the the speed between each timestep.
4. Take the mean of all these speeds.

Let us look at an example to show how this would work. Consider the jog data presented in Figure 5.22.

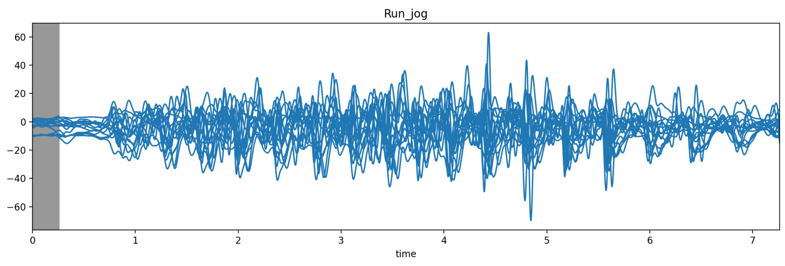
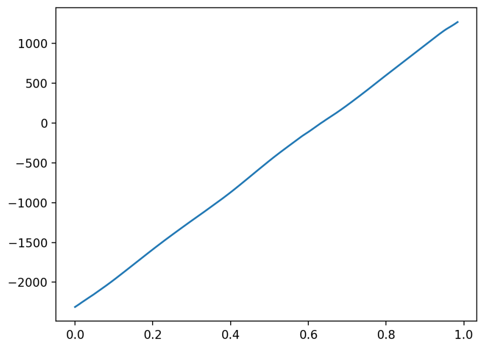


Figure 5.22: Raw accelorator data from a jog.

After using the LPSIS and RPSIS data to compute the centre of mass of these two locations, we find location data as presented in Figure 5.23.

After computing the differences between the location points and dividing them by the length of the timesteps, we obtain the running speeds. These are shown in Figure 5.24. Interesting to note here is that the speed fluctuates between 11 and 14 kilometers per hour. A possible explanation for this would be that, while running, the participants shortly speed up after each step.

This method gives high accuracies, as can be seen in table 5.1. It shows that the test accuracy using this technique can go up to 94.99%. This is reached when the models are solely trained to classify different run intensities. Each model was trained 5 times. The data was unstandardized.



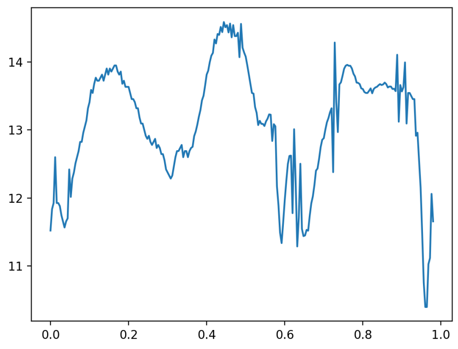


Figure 5.23: The location data from the centre of mass of the LPSIS and RPSIS.

Figure 5.24: The velocity data from the centre of mass of the LPSIS and RPSIS.

Model	Train	sd	Test	sd
1DCNN ws	0.9552	0.0140	0.9312	0.0065
1DCNN per Sensor	0.9470	0.0049	0.9467	0.0160
1DCNN combined	0.9623	0.0149	0.9269	0.0121
2DCNN ws	0.9531	0.0065	0.9397	0.0123
2DCNN per Sensor	0.9426	0.0155	0.9461	0.0097
2DCNN all Signals	0.9678	0.0103	0.9269	0.0083
2DCNN combined	0.9902	0.0115	0.9417	0.0171
1DCNN ws LSTM	0.9527	0.0136	0.9413	0.0129
1DCNN per Sensor LSTM	0.9463	0.0154	0.9456	0.0073
1DCNN combined LSTM	0.9337	0.0138	0.9445	0.0174
2DCNN ws LSTM	0.9442	0.0141	0.9499	0.0153
2DCNN per Sensor LSTM	0.9492	0.0055	0.9413	0.0056
2DCNN all Signals LSTM	0.9296	0.0061	0.9360	0.0069
2DCNN combined LSTM	0.9200	0.0045	0.9344	0.0184
1DCNN ws bLSTM	0.9545	0.0065	0.9461	0.0064
1DCNN per Sensor bLSTM	0.9239	0.0073	0.9451	0.0194
1DCNN combined bLSTM	0.9568	0.0132	0.9419	0.0121
2DCNN ws bLSTM	0.9588	0.0066	0.9467	0.0128
2DCNN per Sensor bLSTM	0.9545	0.0128	0.9451	0.0140
2DCNN all Signals bLSTM	0.9524	0.0115	0.9328	0.0064
2DCNN combined bLSTM	0.9531	0.0065	0.9467	0.0065

Table 5.1: Training results of the deep learning models when they are solely trained to classify different run intensities. "sd" represents the standard deviation

5.2. Sliding window evaluation

In section 5.1, it is explained how the models are trained to recognize 7 different activities: shots, passes, jumps, COD's, jogs, runs and sprints. The next step is to use these models to evaluate new strings of data, also ones that last longer than the 1 second windows. However, these 1 second windows are the key to evaluating these larger data strings. This evaluation will be done using the sliding window evaluation. This process is shown in Figure 5.25. The set-up of this evaluation is similar as the training. A recording comes in, is split up in 1 one second windows with 75% overlap, and is evaluated one window at a time. In this section, it will be explained how these windows are classified, postprocessed and how outlier are removed.

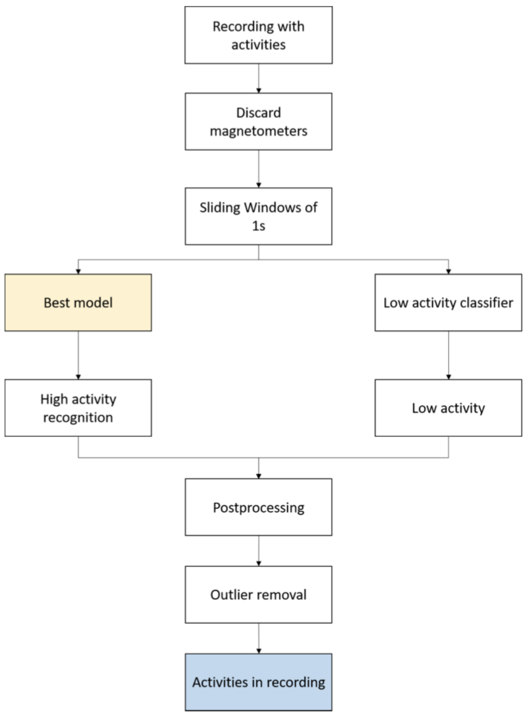


Figure 5.25: Evaluation phase diagram [2].

5.2.1. High/Low activity recognition

As discussed in section 2.3.2, activity detection was used filter out periods of standing still or walking slowly out of the data, such that the trained activities are isolated. However, in actual data that needs to be processed, like football training and games, these periods of standing still do happen and need to be classified appropriately. These periods of standing still or walking slowly will be called low activity. Low activity periods are characterized by, as the name suggests, intervals where the person does not move much, so the signals have very low amplitude and especially small variations. These characteristics could be exploited to accurately differentiate a low activity interval from a high one. [2] reasoned that recognizing these low activity periods is best when building a binary high/low classifier, before using the trained models to classify the activities. This procedure is shown in Figure 5.26. Here, it can be seen that the signal data is first classified as a high or low activity. If it is classified as high, it will be send to the trained model to be classified as a specific activity. If it is low, the output will just set as "low activity".

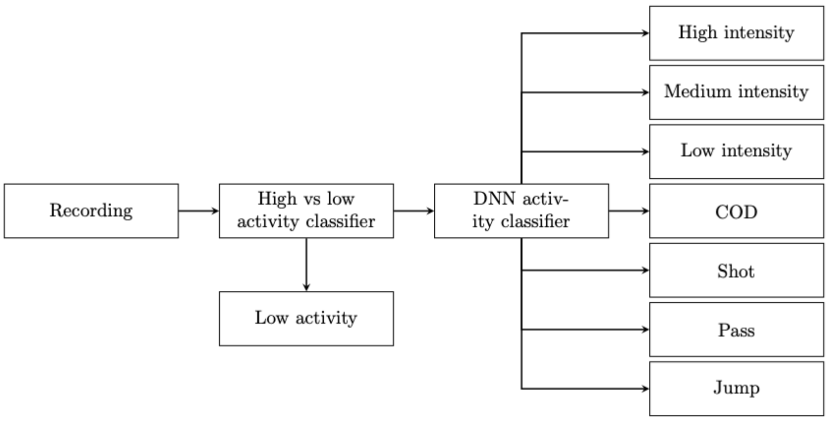


Figure 5.26: Evaluation phase diagram

This classifier works similar as the activity detection used in section 2.3.2, using the euclidean norm of the accelerometer signals. From [2], it could be concluded that the standard deviation of these norms was the best to use to distinguish the two categories, with a threshold of 8.5. This would result in the following procedure. For a given window, the euclidean norm of all the acceleration signals is calculated and the standard deviation of its values is obtained. If that metric is smaller than 8.5, the window is classified as a low activity. Otherwise, it is identified as a high activity, and it then goes through the previously built deep learning-based model to further classify the movement in one of the 7 specific activities: shots, passes, jumps, turns, jogs, runs and sprints.

This classifier used outputs a confidence to every activity. This confidence is a number between 0 and 1, and the sum of the confidences of all activities adds up to 1. This confidence represents how sure the model is that the classified window represents a specific activity. The activity with the highest confidence, is called the prediction.

5.2.2. Post-processing

Once each window is classified, the predictions should be post-processed. This has a very specific reason, mainly because the windows have overlap. This overlap was done to capture most information in the data. If it turns out that in important piece of data is exactly at the edge of two windows, it could cause this data to be overlooked. The problem with this overlapping in windows is that one window could classify as activity A, while the next window, which is still overlapping, is classified as activity B. This would cause a problem, since it is not preferable for the model to give two different classifications for the same timestep. for this reason, we post-process the results. This can be done in various ways, such as using interpolations or using the mode of the predictions. [2] reasoned however that the optimal method is the best score post-processing.

Best score post-processing

It is defined as: *All the timesteps of window i are assigned to the prediction of window i. The final prediction for each timestep is the prediction that had the largest probability/confidence among the predictions of all the windows that contained that timestep.* A toy example is presented in Figure 5.27 to explain this process. In this example, the recording is presented as the multicolored bar at the top of the image. This recording is sliced into several windows, represented by W_1 to W_{12}. For each window, a prediction for which activity is performed in that window, is made. These predictions are all represented by a color: blue, red or green.
The prediction for each window is represented by the horizontal colored bars in the middle of the figure. Each bar has two characteristics. First is the color, which represents the predicted activity in that window. The second is the tone, which represents the confidence of that prediction. Does the bar have a dark tone, the prediction was made with high confidence. For light bars, there was a small confidence. It can be noted that these bars overlap in most timesteps. The final prediction in that timestep is the activity with the largest confidence. As an example, it can be seen windows W_2, W_3 and W_4 all include tho fourth timestep. W_2 and W_3 give very light, red predictions, and thus have low confidence. On the other hand, W_4 gives a dark, blue

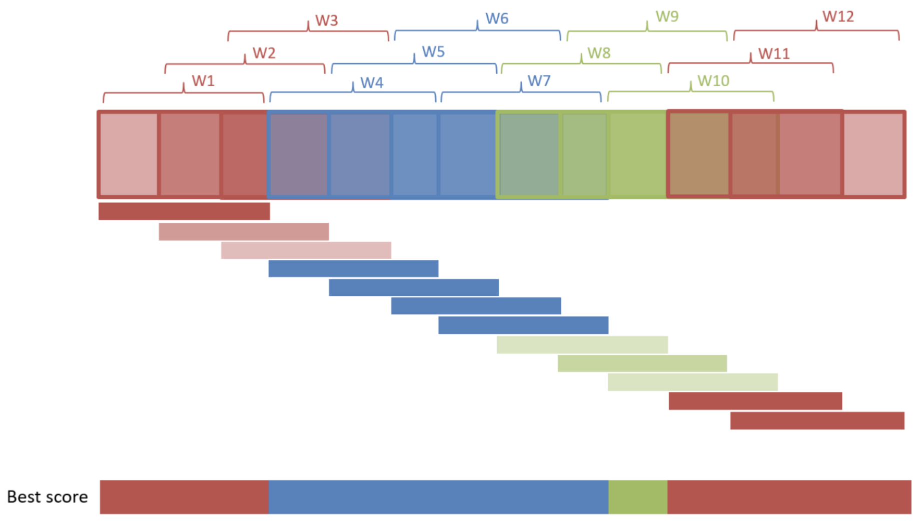


Figure 5.27: Best score post-processing [2].

prediction and thus has a big confidence. Since the blue prediction has the highest confidence, it is chosen as the final prediction, despite there being two red predictions and only one blue. The final prediction for all timesteps are shown in the horizontal bar at the bottom of the figure.

What this achieves is that the best score post-processing mostly looks at the predictions with high confidences, instead of a "follow the crowd" approach where short instances are filtered out. This has two advantages. First it cleans up the results of short, low confidence actions. When a player is running and for example slips for a millisecond, the model could confuse that small slip as a different activity even when it has low confidence. With this method, these extremely short instances are filtered out. The other advantage is the direct opposite of this. If a player is running and has to make a quick COD, and this is detected with high enough confidence, it is not filtered out. Hence stand-alone, high confidence activities are detected better, while short low confidence activities are filtered out.

In order to calculate such confidences or scores, an activation function is needed in the last layer of the neural network. Since this project handles a multi-class classification task, the softmax activation function was chosen. This activation function at the output of the model represents the probability distribution over the possible categories. The softmax activation function is defined as

$$\sigma(z) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}},$$

where i is the i'th activity among all the K possible activities to detect, and z is the output of the last layer of the neural network.
The confidences and scores of the predictions for each window were obtained by using the outputs of the softmax activation function in the last layer. The window's prediction was determined as the activity with the highest confidence. For windows with "low activity," their confidence values were assigned as $c_l = 98\%$, resulting in a high score, but not enough to surpass predictions with even higher scores for high-activity windows. This value of 98% was set as a hyperparameter, c_l, and can be adjusted manually if desired.

5.2.3. Outlier removal and “other high activity” recognition

Once the predictions have undergone post-processing, an outlier removal procedure is implemented to address any isolated activities that may remain. Despite the use of mode or best score post-processing methods, there is still a chance that short peaks of activity may persist. However, these brief activities lasting less than a couple of milliseconds do not occur in reality, and if detected, are considered outliers that must be removed. To accomplish this, a rule was established such that if a predicted activity lasts less than τ milliseconds, it is replaced with the next predicted activity lasting at least τ milliseconds. In this thesis, τ values of 310 milliseconds were found to produce satisfactory results. The overall sliding evaluation process is illustrated in Figure 5.28, wherein each sliding window position is classified, followed by post-processing and outlier removal. The "Activity Classifier" block in the diagram corresponds to the activity classification process shown in Figure 5.26.

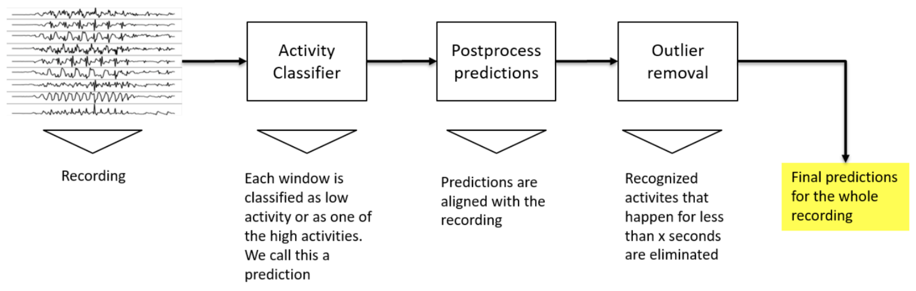

Activity Classifier
Postprocess predictions
Outlier removal
Recording
Each window is classified as low activity or as one of the high activities. We call this a prediction
Predictions are aligned with the recording
Recognized activites that happen for less than x seconds are eliminated
Final predictions for the whole recording

Figure 5.28: Sliding window evaluation procedure [2].

The final layer's softmax activation function served a dual purpose, beyond facilitating the best score post-processing option. It also enabled visualization of the scores of recognized activities during evaluation by using a color gradient. As the outputs of softmax always sum up to 1, a predicted activity must have a confidence greater than 1 divided by the number of possible activities to be considered over others. The highest possible confidence is 1. Therefore, the color gradient was defined as follows

$$(\text{Yellow}, \text{Black}) \rightarrow \left(\frac{1}{classes}, 1\right).$$

This means that a prediction displayed in yellow would indicate a very low confidence level, while a prediction displayed in black would indicate a much higher confidence level, which could be relied upon more. Ultimately, the purpose of the max score post-processing technique is to eliminate predictions with low, yellow confidence levels.

In addition to its use in visualization, the yellow-black color code was utilized to enable the model to identify a new category of activity termed "other high activity". Although the model was created to recognize seven high activities specifically, football is a multi-faceted game in which players perform various activities beyond those seven. To enable the model to recognize these activities to some extent, the prediction confidence was utilized. A rule was established that defined this new category as a prediction of any of the five predetermined activities with a confidence below a threshold h_h. Thus, if a prediction made by the model has a confidence score of less than h_h, the window prediction is considered as "other high activity" rather than the identified activity by the model. This logic was created before the post-processing stage to ensure that "other high activities" would be post-processed alongside other activities. As the best score post-processing prioritizes predictions with high confidence, and "other high activities" have low confidence by definition, they are consistently removed from the final predictions, rendering the entire process pointless. Therefore, after a prediction is transformed into an "other high activity," its confidence c_h is assigned to a high value lower than 100%. The threshold h_h is a hyperparameter that can be adjusted as desired, and a larger value enables easier recognition of "other high activities." If the additional category of "other high activity" is not required, the threshold h_h can be set to 0.

It is crucial to note that the deep learning model developed was not trained to identify the "other high activities" mentioned earlier, as it was specifically optimized to recognize the seven activities previously mentioned. The method used to convert low-confidence predictions into "other high activities" is not ideal because the model's main objective is to categorize each window as one of the seven activities. Therefore, if there is a need to robustly recognize additional activities, it is recommended to retrain the model to specifically detect those new classes.

The inclusion of the component to identify "other high activities" causes a slight modification to the pipeline illustrated in Figure 5.29, resulting in the updated version shown in Figure 5.28. This revised pipeline represents the final configuration of the complete model, presented in a concise form.

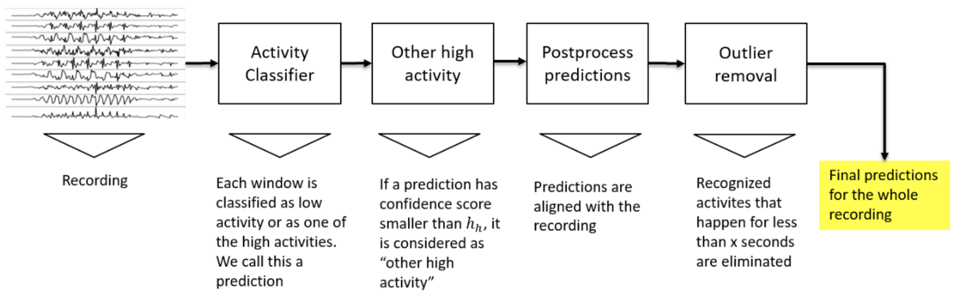

Recording
Activity Classifier
Each window is classified as low activity or as one of the high activities. We call this a prediction
Other high activity
If a prediction has confidence score smaller than h_h, it is considered as "other high activity"
Postprocess predictions
Predictions are aligned with the recording
Outlier removal
Recognized activites that happen for less than x seconds are eliminated
Final predictions for the whole recording

Figure 5.29: Final sliding window evaluation procedure with inclusion of "other high activity" class [2].

6 Results

Now that the methodology is explained, it is time to look at the results. First, the results obtained by [2] will be recreated. Important to note here is that this includes less activities, since in that report only 5 activities were considered, namely shot, sprint, jump, jog and pass. Afterwards, there will be looked at the results from training the models and which should be used for evaluating the test datasets. Finally, we will then look how those models performed on the larger datasets.

6.1. Training

Before using deep learning models to evaluate bigger datasets, these model need to be trained. Here the results of the training and the accuracy the models bring.

6.1.1. Recreation results Cuperman Coifman

First, the training results from [2] will be recreated. Note that these results are solely based on 5 different activities: shots, passes, jumps, jogs and sprints. Here jogs and sprints are labelled as it is by the training data. This means that the analyses explained in section 5.1.5 to compute the running speed, is not performed. The results are shown in Table 6.1. It shows the average training and test accuracy for each Deep Learning model and its standard deviation, where each model was trained 5 times. As can be seen, the results for both papers are similar, within the same margin of error. In both these computations, the data was unstandardized. [2] also investigated the accuracies with standardized data, however since these were less accurate, these were not computed here.

	[2]				Recreation			
Model	Train	sd	Test	sd	Train	sd	Test	sd
1DCNN ws	0.9913	0.0081	0.9436	0.0133	0.9881	0.0015	0.9333	0.0083
1DCNN per Sensor	0.9935	0.0168	0.9392	0.0178	0.9893	0.0017	0.9389	0.0078
1DCNN combined	0.9866	0.0033	0.9397	0.0148	0.9881	0.0026	0.9556	0.0147
2DCNN ws	0.9880	0.0049	0.9359	0.0160	0.9714	0.0108	0.8917	0.0075
2DCNN per Sensor	0.9932	0.0046	0.9479	0.0142	0.9989	0.0030	0.9583	0.0090
2DCNN all Signals	0.9758	0.0087	0.9288	0.0138	0.9964	0.0057	0.9472	0.0103
2DCNN combined	0.9934	0.0044	0.9534	0.0142	0.9897	0.0092	0.9722	0.0126
1DCNN ws LSTM	0.9901	0.0052	0.9605	0.0121	0.9964	0.0056	0.9667	0.0103
1DCNN per Sensor LSTM	0.9831	0.0122	0.9595	0.0087	0.9964	0.0051	0.9694	0.0136
1DCNN combined LSTM	0.9887	0.0062	0.9655	0.0160	0.9964	0.0036	0.9778	0.0061
2DCNN ws LSTM	0.9887	0.0092	0.9627	0.0100	0.9964	0.0083	0.9667	0.0059
2DCNN per Sensor LSTM	0.9932	0.0039	0.9671	0.0131	0.9940	0.0054	0.9639	0.0108
2DCNN all Signals LSTM	0.9908	0.0058	0.9545	0.0102	0.9917	0.0057	0.9667	0.0073
2DCNN combined LSTM	0.9866	0.0037	0.9638	0.0105	0.9917	0.0030	0.9583	0.0067
1DCNN ws bLSTM	0.9922	0.0034	0.9671	0.0073	0.9934	0.0063	0.9722	0.0096
1DCNN per Sensor bLSTM	0.9920	0.0060	0.9622	0.0053	0.9917	0.0106	0.9528	0.0160
1DCNN combined bLSTM	0.9899	0.0053	0.9611	0.0154	0.9962	0.0060	0.9639	0.0146
2DCNN ws bLSTM	0.9920	0.0041	0.9622	0.0096	0.9988	0.0038	0.9667	0.0110
2DCNN per Sensor bLSTM	0.9899	0.0075	0.9611	0.0110	0.9927	0.0045	0.9583	0.0123
2DCNN all Signals bLSTM	0.9845	0.0096	0.9496	0.0099	0.9917	0.0032	0.9694	0.0117
2DCNN combined bLSTM	0.9929	0.0036	0.9600	0.0150	0.9929	0.0030	0.9639	0.0054
LSTM	0.9148	0.0217	0.7748	0.0336	0.9488	0.0401	0.8333	0.0251
bLSTM	0.9965	0.0024	0.8707	0.0225	0.9798	0.0084	0.8472	0.0091

Table 6.1: Recreation of the training results from [2]. The original on the left, the recreated results on the right. This training contained shots, passes, jumps, jogs and sprints. "sd" represents the standard deviation.

6.1.2. Training Results

Now it is time to add the extra activities to the training data. These additional activities are the COD's, and replacing the jog/sprint by high, medium and low intensity running. Those results are shown in Table 6.2. It shows the average training and test accuracy for each Deep Learning model and its standard deviation, where each model was trained 10 times. Again, the data was unstandardized. The models that stand out negatively are the LSTM and the bLSTM. These models have very low train and test accuracy, whereas all other models have a test accuracy between 0.84 and 0.89. This shows that these two models on their own are not applicable. However, when a LSTM or bLSTM is added to a convolutional neural network, it generally increases its accuracy.

Model	Train	sd	Test	sd
1DCNN ws	0.9266	0.0093	0.8426	0.0112
1DCNN per Sensor	0.9355	0.0090	0.8472	0.0106
1DCNN combined	0.9752	0.0122	0.8542	0.0103
2DCNN ws	0.8968	0.0060	0.8218	0.0132
2DCNN per Sensor	0.9464	0.0054	0.8611	0.0146
2DCNN all Signals	0.9623	0.0050	0.8704	0.0157
2DCNN combined	0.9435	0.0069	0.8519	0.0073
1DCNN ws LSTM	0.8641	0.0090	0.8380	0.0145
1DCNN per Sensor LSTM	0.8710	0.0046	0.8519	0.0060
1DCNN combined LSTM	0.8899	0.0061	0.8519	0.0079
2DCNN ws LSTM	0.8601	0.0064	0.8380	0.0132
2DCNN per Sensor LSTM	0.8690	0.0068	0.8519	0.0127
2DCNN all Signals LSTM	0.9038	0.0102	0.8796	0.0106
2DCNN combined LSTM	0.9097	0.0066	0.8889	0.0083
1DCNN ws bLSTM	0.9058	0.0069	0.8750	0.0102
1DCNN per Sensor bLSTM	0.8889	0.0125	0.8727	0.0068
1DCNN combined bLSTM	0.8938	0.0077	0.8657	0.0075
2DCNN ws bLSTM	0.8968	0.0068	0.8889	0.0064
2DCNN per Sensor bLSTM	0.8740	0.0103	0.8495	0.0120
2DCNN all Signals bLSTM	0.9107	0.0069	0.8611	0.0074
2DCNN combined bLSTM	0.8770	0.0082	0.8472	0.0102
LSTM	0.3716	0.0217	0.3591	0.0336
bLSTM	0.8648	0.0024	0.7401	0.0225

Table 6.2: Training results when the models are trained for 7 activities: shots, passes, jumps, COD's, and low, medium and high intensity runs. "sd" represents the standard deviation.

Another factor which is noteworthy is the fact that the test accuracy of the models in significantly lower compared to the results of training the model with only 5 activities. In Table 6.1, the minimum test accuracy, excluding LSTM and BLSTM, was 0.9778. Now we find that the maximum accuracy is 0.8889. Why this is the case, can be explained using an example in Figure 6.1. This figure shows the confusion matrix of the model where there is a combination of 2DCNN weight sharing, 2DCNN per sensor, and 2DCNN all sensor, with a LSTM cell added. This was the model with the highest test accuracy. It shows that the inaccuracies of model mostly lies in the run intensity. For instance, 13.2% of all medium intensity runs is classifies as low intensity, and 19% of all high intensity runs is classified as medium intensity. This is the result of the fact that run intensity is a continuous spectrum. A run of 19.5 kilometers per hour would be considered a medium intensity run, whereas a run of 20 kilometers per hour is a high intensity run. However, this speed difference is so small that it is very difficult for a model to distinguish these two different activities. Consequently, the deep learning models tend to have lower overall test accuracy.

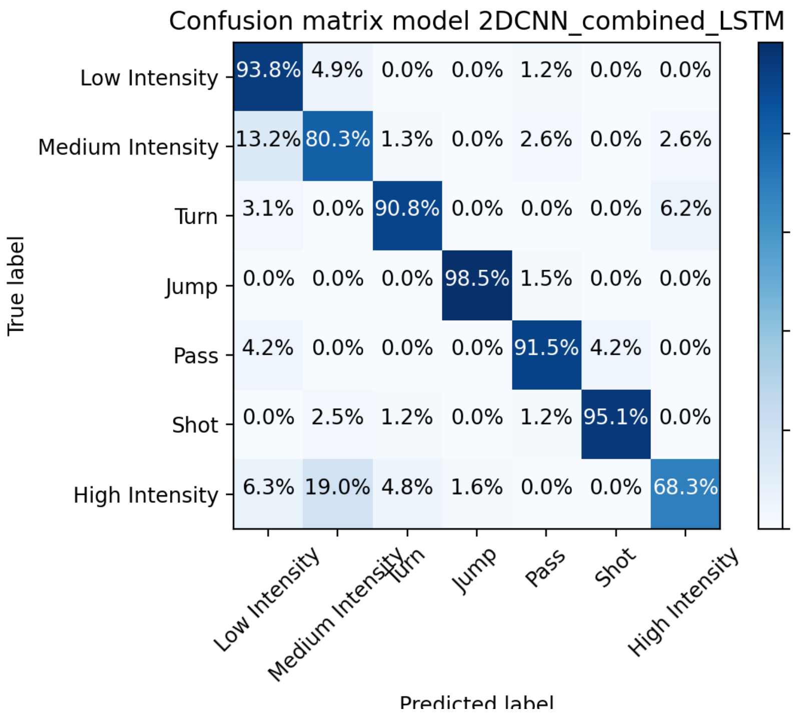


Figure 6.1: Confusion matrix of the "2DCNN combined LSTM" model.

6.1.3. Exclusion of shank sensors

As mentioned before, the results are based of 5 IMU sensors: on the pelvis, both thighs and both shanks. In the case of the dataset [17], the data was collected where the data was collected using specifically designed pants where these IMU's were integrated in the fabric. This pants can be seen in Figure 6.2. This pants however is not practical in use. It covers the whole leg, which could be too warm for players to wear with higher temperature. For this case it would be beneficial if the pants would only reach until the knees of a player. This would mean that no IMU data from the shanks could be obtained. Therefore it is interesting to see how the exclusion of shank data would influence the accuracy of the deep learning models. this is shown in Table 6.3. It shows the average training and test accuracy for each Deep Learning model and its standard deviation, where each model was trained 10 times. The data was unstandardized.

Interesting to see is that the test accuracy does not suffer much from excluding shank data. Compared to the results shown in Table 6.2, the highest test accuracy is only 0.0020 higher in the model which uses all the IMU sensors. This would indicate that it would be interesting also develop a pants with integrated IMU's that only reach until the knees. Downside from excluding the shank sensors is that the standard deviation for the test accuracy is generally higher. This would indicate that the results can be less trustworthy.

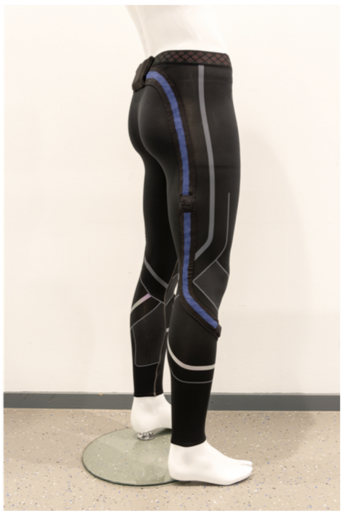


Figure 6.2: The pants with build-in IMU sensors [2].

Model	Train	sd	Test	sd
1DCNN ws	0.9575	0.0060	0.8353	0.0113
1DCNN per Sensor	0.9124	0.0137	0.8234	0.0102
1DCNN combined	0.9294	0.0144	0.8651	0.0090
2DCNN ws	0.9116	0.0121	0.8274	0.0182
2DCNN per Sensor	0.9192	0.0137	0.8373	0.0129
2DCNN all Signals	0.9473	0.0120	0.8571	0.0121
2DCNN combined	0.9116	0.0111	0.8552	0.0192
1DCNN ws LSTM	0.8724	0.0060	0.8651	0.0182
1DCNN per Sensor LSTM	0.8886	0.0137	0.8770	0.0177
1DCNN combined LSTM	0.8861	0.0065	0.8651	0.0094
2DCNN ws LSTM	0.9107	0.0072	0.8671	0.0194
2DCNN per Sensor LSTM	0.8750	0.0122	0.8671	0.0084
2DCNN all Signals LSTM	0.8878	0.0085	0.8552	0.0141
2DCNN combined LSTM	0.9082	0.0128	0.8710	0.0157
1DCNN ws bLSTM	0.8912	0.0061	0.8591	0.0193
1DCNN per Sensor bLSTM	0.8733	0.0078	0.8532	0.0109
1DCNN combined bLSTM	0.9226	0.0112	0.8631	0.0151
2DCNN ws bLSTM	0.9269	0.0109	0.8869	0.0184
2DCNN per Sensor bLSTM	0.8878	0.0136	0.8790	0.0146
2DCNN all Signals bLSTM	0.9413	0.0065	0.8869	0.0104
2DCNN combined bLSTM	0.8980	0.0084	0.8810	0.0095
LSTM	0.4158	0.0217	0.4008	0.0336
bLSTM	0.7849	0.0024	0.6448	0.0225

Table 6.3: Training results when the data from the shank sensors is excluded. "sd" represents the standard deviation.

6.2. Sliding window evaluation

Now that the models are trained, they can be used to evaluate larger datasets. This will be done using the sliding window evaluation, as explained in section 5.2. The results of this analysis will be presented here.

6.2.1. Isolated activities

The first datasets to be looked at are isolated activities. Hence between every activity, the participant had period of standing still. These datasets come from 17, the same as the training data. However, these activities were not used in the training of the models. Results of several datasets are shown in Figures 6.3 and 6.4. In Figure 6.3, a dataset with several isolated activities is presented: a run at medium intensity, a jump, a pass, a shot and finally a run at high intensity. As can been seen, the model is able to classify these activities perfectly. Interesting to note is that in the original prediction, there are some "lonely" points in several locations. This indicates that several individual windows are classified wrongly. These predictions however generally have a yellow color, hence its confidence is low. Since the confidence is low and these are small in number, they are corrected by the post-processing and the outlier removal.

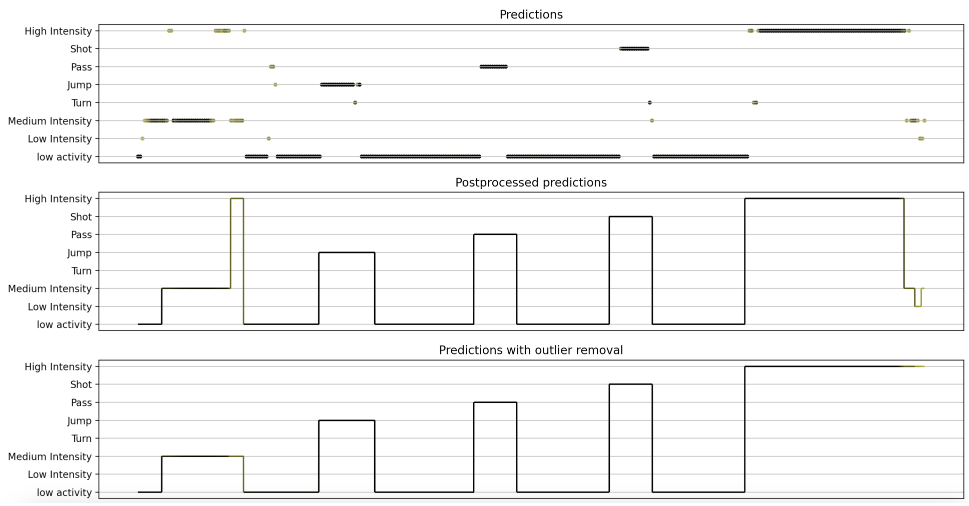


Figure 6.3: Evaluation results. True labels: medium intensity run, jump, pass, shot, high intensity run.

In Figure 6.4, several runs are performed in sequence. These runs have rising intensity, starting with low intensity, followed by medium and ended by a high intensity run. The model was able to classify these intensities correctly. When looking at the original predictions, before post-processing and outlier removal, it can be seen that during the medium intensity run, the predictions jump several times to high intensity. This can have multiple reasons. First could be the accuracy of the model. As Figure 6.1 showed, there are slight inaccuracies between the different run intensities. Another reason could be that the participant for a short amount of time actually performed a high intensity run. As discussed in section 5.1.5, the running speed seems to increase slightly every time there is a push-off. This short increase in speed could cause the participant to go faster than medium intensity.

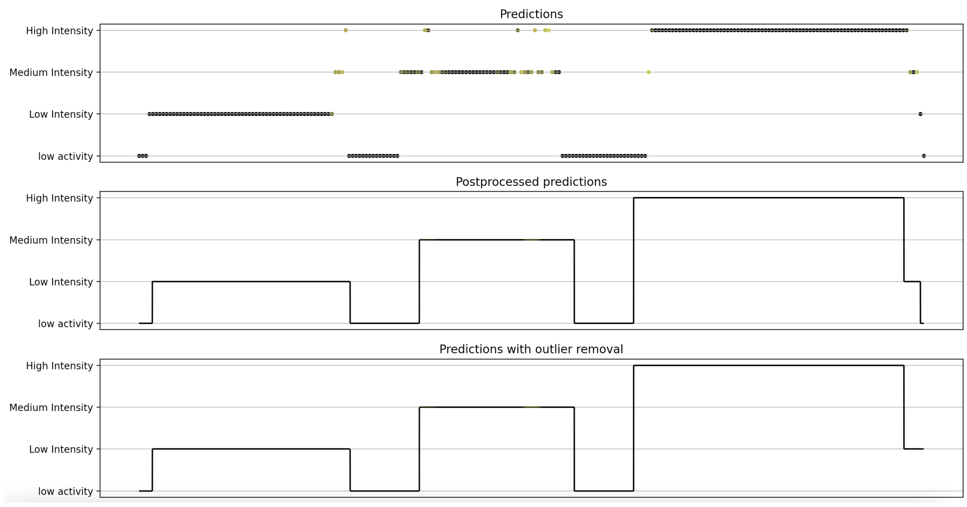


Figure 6.4: Evaluation results. True labels: low intensity run, medium intensity run, high intensity run.

6.2.2. Change of Direction recognition

So far, sliding window evaluation was only used to evaluate isolated activities, where after each activity, a moment of low activity was performed before moving on with the next activity. However, in football games this would be impossible. There it is normal the activities follow up on each other without moments of pause. To test if sliding window evaluation can still up on these changes, recognizing COD's might be a good place to start. Essentially, COD's consist of three consecutive activities: a run, the COD, and another run. It is interesting to see how sliding window evaluation handles these situations. To analyse this, there will be looked at two separate cases: 180 degree turn and 90 degree cuts. These COD's are unused datasets from 17.

180 degree COD

First is the 180 degree COD. The model should recognize these well, since it is trained to recognize it. An example of the sliding window evaluation used on a dataset with a 180 degree COD is shown in Figure 6.5. The model shows indeed that the activity is started by a high intensity run, is interrupted by a COD and then continued by a run which is decreasing in intensity. Also interesting to see here is that in the predictions there is a clear overlap between running and turning. The predictions on the outside of the COD have a more yellow color. This indicates the confidence is low in those windows. This could come from the fact that the run smoothly transitions into a turn and vice versa.

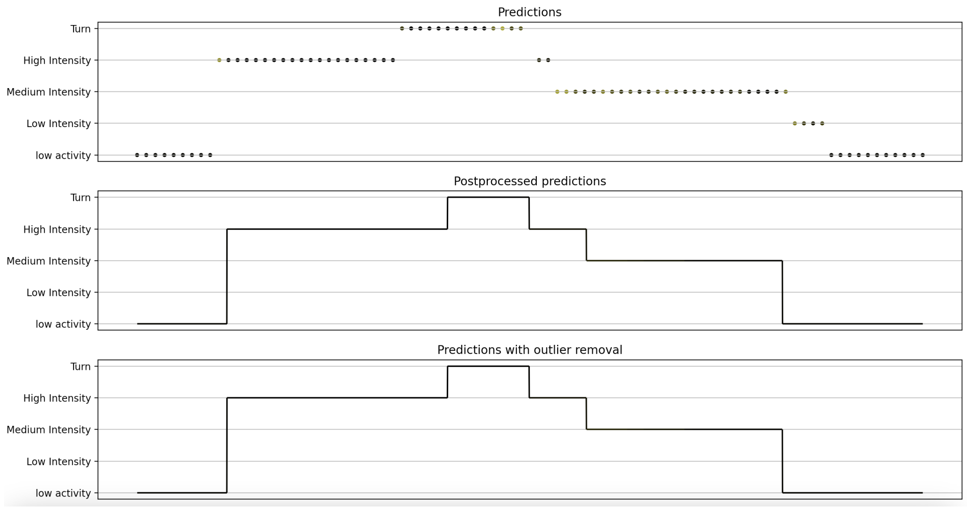


Figure 6.5: Evaluation results of high intensity run with a 180 degree COD contained in it.

90 degree COD
Now we will look at 90 degree COD's, also referred to as a cut. The difference between a cut and a 180 degree COD, is that cuts are not specifically trained by the model. The reason for this could be best explained by Figures 6.6 and 6.7. Here Figure 6.6 represents a 180 degree COD, and Figure 6.7 represents a cut. Note that Figure 6.6 is the same as Figure 5.14 from section 5.1.4, but is displayed here again to make to comparison easier. In the COD, there is a clearly visible area where the COD takes place. In the case of the cut this is not the case. This makes it difficult to decide in which part of the data the cut takes place. Also using the extreme height difference method, as explained in section 5.1.4 does not work here. For that reason, it was decided not to include cuts into the training data.

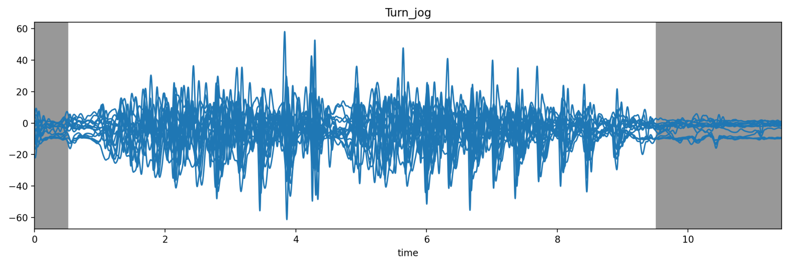


Figure 6.6: Raw accelorator data of a run with a 180 degree COD contained in it.

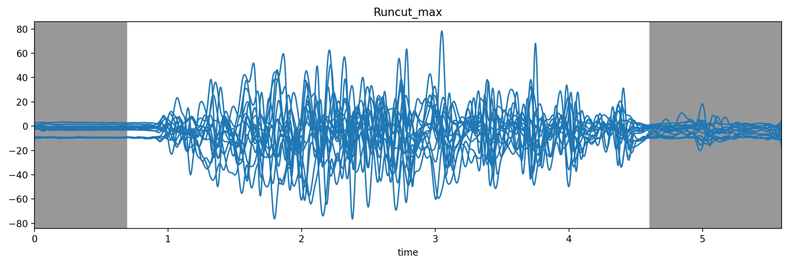


Figure 6.7: Raw accelorator data of a run with a 90 degree COD contained in it.

An example of the sliding window evaluation used on a dataset with a cut is shown in Figure 6.8. It shows a high intensity run, which is eventually interrupted by a COD. This suggests that the model still recognizes the cut as a COD, even if it did not specifically trained to recognize cuts. This recognition works for most of the data containing a cut, since in 93.3% of them the cut was recognized correctly. Hence, the model seems to be able to recognize COD's from degrees between 90 and 180, while only training with 180 degree COD's.

Note that after the COD, right away a period of low activity starts. This is due to the set up when the data was collected, where the VICON sensors would only be able to capture data in a small space. Running left or right would result in the VICON sensors being out of range. That is why after a 90 degree cut, no further run is performed.

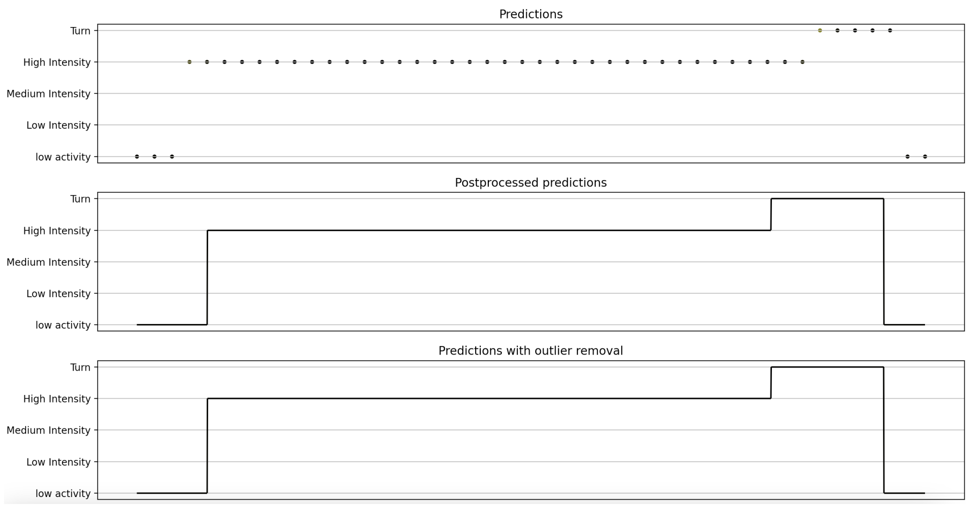


Figure 6.8: Evaluation results of high intensity run with a 90 degree COD contained in it.

6.2.3. Football drill

The next step is to see how the model performs a so called football drill. This drill, as explained in section 3.2.2, contains of several different footballing activities. Here, these activities will be evaluated using the "2DCNN combined LSTM" model. This model was chosen, since section 6.1.2 showed that this had the highest test accuracy. The evaluation had to be done manually. Hence the chosen model chosen classified the data, and the classification was manually checked using video footage of the football drill. In the classification of the football drill, there will be an option to look for "Other high activity", as it was explained in section 5.2.3. This is done, because the football drill contained certain activities that the model was not trained for, such as walking backwards.

For the football drill, there will be 3 parts that will be analysed. The first part is part of a warm-up exercise. This is an exercise where the contestant jumps sideways with one foot, and then closes the gap with his other foot. This is shown in Figure 6.9. The contestant's face has been blurred out for privacy reasons.

Figure 6.9: Participant performing sideways jumps.

The second exercise is a combination of slowly running, jumping and passing the ball. This can be seen in Figures 6.10 and 6.11. The goal for the contestant is to run from pawn to pawn, and at that pawn to either perform a jump with heading the ball, or to give a pass. This activity contains a total of 3 jumps and 3 passes, where the two are always alternating and separated by a period of running. Important to note here is that the contestant knows beforehand whether he needs to pass or jump.

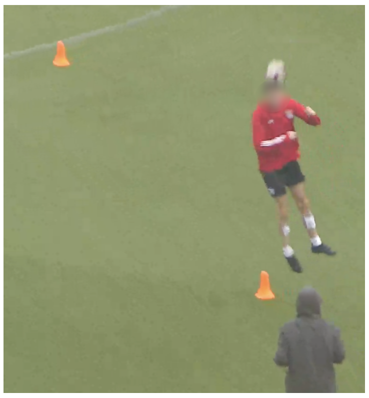


Figure 6.10: Participant jumping to head the ball.

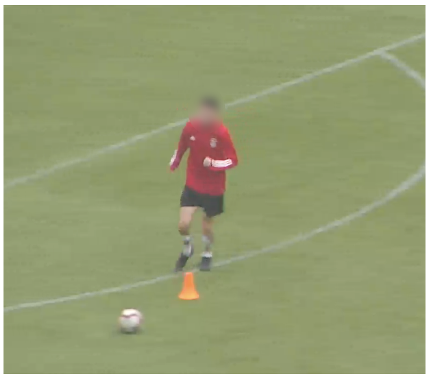


Figure 6.11: Participant passing the ball.

The last activity of the drill is a zigzag between pawn, followed by a single pass and ended with a sprint. This is shown in Figures 6.12, 6.13 and 6.14. During the zigzag, the participant needs to run between several pawns, that are at 90 degree angles form each other. Once he reaches a pawn, he is expected to go around that pawn and then run to the next one. After the last pawn, he is expected to shoot a ball away. Afterwards, he should turn around and end with a sprint.

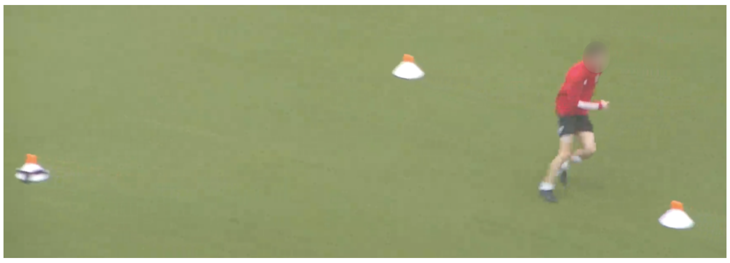


Figure 6.12: Participant performing a zigzag, by running from pawn to pawn.

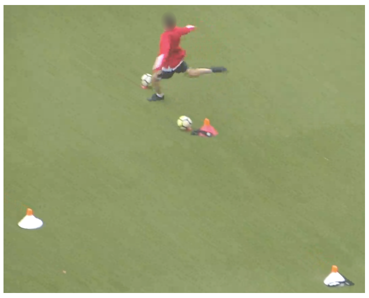


Figure 6.13: Participant shooting the ball.

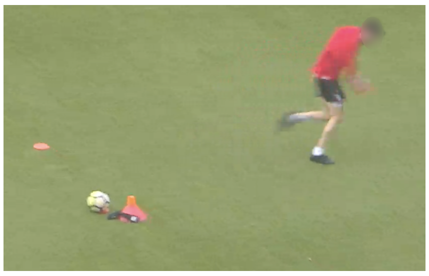


Figure 6.14: Participant closing with a sprint.

The result of the evaluation of these activities is shown in Figure 6.15. Here the 3 aforementioned activities are evaluated in sequence, and the length of the recording was around the 44 seconds. Which activity is which is easily distinguishable by the periods of low activity between them. Starting with the first activity, the classification shows a long period of jumping, mixed with a pass and an "other high activity". This seems like a logical classification. This comes from the fact that during the first activity the contestant jumps sideways constantly, which makes it plausible for the model to recognize it as jumping. Since the model is not properly trained to detect this activity, it clearly gets confused since it also classifies a pass and an "other high activity".

For the second activity, a total of 2 jumps can be counted, and 5 passes/shots. This seems odd, since there were 3 jumps and 3 passes/shots performed. The failure in recognizing the jumps could be caused by the difference in jumps between training and during this drill. The jumps the model used to train, were jumps performed

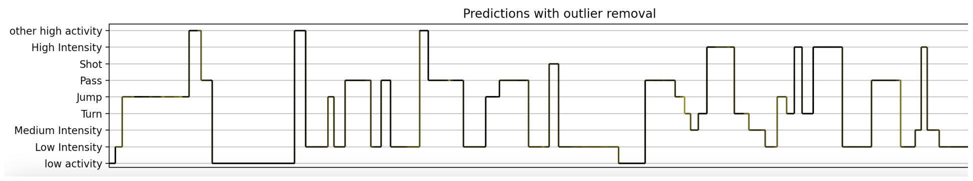


Figure 6.15: Evaluation results of the football drill.

without a ball and jumping vertically. As seen in Figure 6.10, during the drill the jump was with the intention to head the ball. To give power to heading the ball, the participant has to use his arms and legs to give the ball extra momentum. This results in a different way of jumping that the model has not seen and trained with yet. The inaccuracy in the passes could relate to this. It can be seen that model combines a jump, immediately followed by a pass. This is impossible, since between each jump and pass is a period of running. The different way of jumping could result in incorrectly classifying a jump as a pass. Another possibility could be the run-up to the pass. After a jump, the participant is expected to run backwards towards the ball. Once he reaches it, he should turn around and pass it. This one motion of turning and passing could result in inaccuracies in the classification. This can be best seen in the case where an "other high activity" is followed by a pass.

This wrong classification of a pass can also be seen at the beginning of the third and final activity. Before starting the zigzag, the participant is expected to walk backwards towards the first pawn. One he reaches it, he should turn around and run from pawn to pawn. This could be the reason why the third activity starts with a pass and a jump. Afterwards, it can be seen that the model recognizes the COD at every pawn, separated by periods of running at several intensities. This is interesting, since each COD is of around 90 degrees, while the model was only trained with COD's of 180 degrees. This supports the intuition from section 6.2.2 that training the model for COD's of 180 degree is enough to classify COD's from degree between 90 and 180. After the zigzag, it can be seen that the pass/shot and the end sprint are classified correctly.

6.2.4. Physiotherapy training

The final dataset that can be evaluated is a physiotherapy training. As explained in section 3.2.3, it involves 3 participants performing several football related activities. These activities are not "scripted", in the sense that the participant know what the football assignment is, but they can choose themselves how to complete them. Hence the activities have a more random nature then in the previous datasets, where each shot, pass, jump, sprint or change of direction was told to the contestants in advance. First, the data for each participant was evaluated using the "2DCNN combined LSTM" model, which is similar as the evaluation in section 6.2.3. The evaluation had to be done manually. Hence the chosen model chosen classified the data, and the classification was manually checked using video footage of the football drill. For the evaluation of this dataset, the option to look for "other high activities" was turned off. This was done since the exercises that will be evaluated are all trainable by the model. Therefore, each action a contestant performs, should be recognizable for the model. To detect smaller activities, the threshold τ for the minimum length is lowered from 310 milliseconds, as mentioned in section 5.2.3, to 200 milliseconds. Hence small actions are less likely to be classified as outliers. Here, only the results from one of the participants are analysed, while the results of the other participants were solely used to verify whether this analysis was consistent.

The first exercise is a passing exercise. This is shown in Figure 6.16. During this exercise, there were 3 football pawns, one for each participant. The point of the exercise is that player A passes to player B and runs to the pawn of that player. When player B receives the ball, he passes it to player C and runs to his pawn. Then player C passes the ball to player A and runs to the pawn where player A started. This way, it forms a chain of passing, running and COD's. This exercise took approximately 220 seconds, which corresponds to almost 4 minutes.

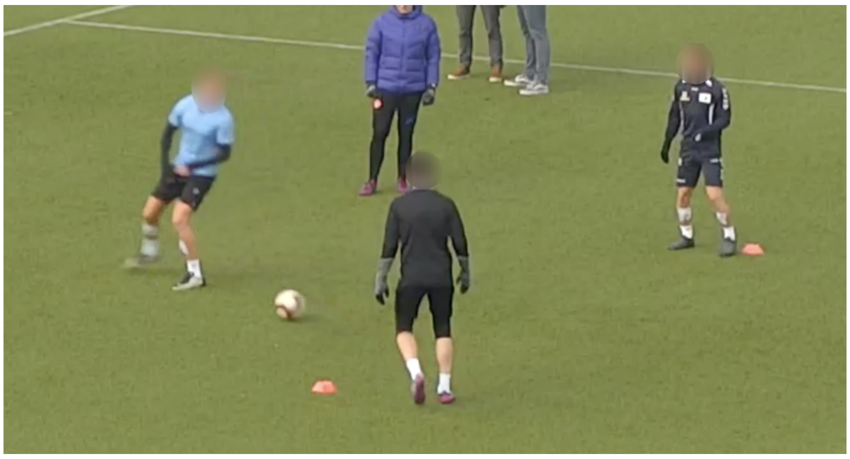


Figure 6.16: First exercise: Participants passing the ball and running in a triangle.

The results are shown in Figure 6.17. Here only the results for one of the players is presented, namely the participant at the most right in Figure 6.16. Interesting to see that the model does recognize some of the passes, but not a lot. In the footage it was shown that this contestant played a total of 43 passes, while the model only recognized 13. The model also recognized a total of 4 COD's, while between each pass also a COD was performed. This lack in classified actions could have several specific reasons. The first is that the passes, the runs and the COD's are all performed at very low intensity. The physiotherapy is part of injury recovery, which makes it likely that the participants are not capable of doing these kind of exercises at high activity. Moreover, the pawns are fairly close it each other. Hence there is no need to pass the ball with a lot of power. That the actions are performed at low intensity, could cause the model to have difficulties detecting them. Another big part could be the way of passing. During the training of the data, passing a ball was done using a full leg swing. In real football situations during a match or training, this is not always the case. Especially when the pass is over a short distance, it is normal for a player to pass only using is shanks, while his thigh stays almost completely still. Also, when the ball is passed during running, often this goes in one smooth movement. Hence the player does not make any different movements then just running and passing the while making a step. This all could result in the model having difficulties detecting passes.

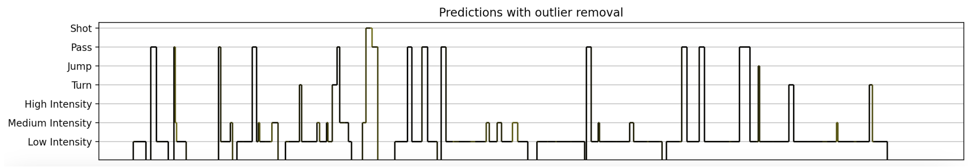


Figure 6.17: Evaluation results of the first exercise.

The next exercise is not part of the training, but it shows that the participant truly perform random activities during the training. As can be seen in Figure 6.18, a participant is juggling the ball with his feet in between different exercises. The model actually picks this up as several passes, as seen in Figure 6.19. This shows that the model can be able to pick up small actions.

Figure 6.18: Participant juggling the ball.

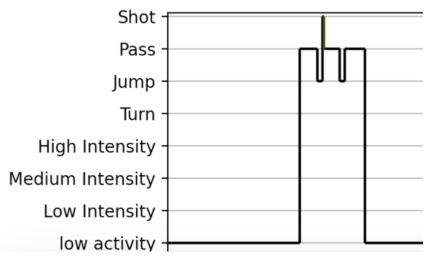


Figure 6.19: Evaluation results of juggling the ball.

The next exercise consists of the participant dribbling around and shooting at small goals. This can be seen in Figure 6.20. During this exercise, each participant had to dribble with the ball randomly. Whenever the physiotherapist gave the sign, they had to shoot at one of three goals surrounding the center circle of the field. During this dribbling, it was important that the participants also avoided each other. Also, when they missed the goal, the participant had to sprint to get the ball back as fast as possible. This exercise took in total 6 minutes. Note that the model is not trained for dribbling or COD's with the ball, what makes it interesting to see how it will perform.

Figure 6.20: Second exercise: Participants dribbling at random in a closed area, shooting at a goal when the trainer says so.

The result is shown in Figure 6.21. These are again the results for only one of the participants. In this exercise a pass or a shot was both deemed correctly classified, since the goals were at such a distance that it is difficult to determine whether a shot should really count as a shot, or that it should be classified as a pass. It can be noted that a lot of passes or shots are detected, in total 34. However, this is way too much since the exercise only consisted of 14 passes/shots. This over estimation of the amount of shots can be explained with the dribbling. As mentioned before, the model is not trained with any dribbling data. This means that is also not trained for players twisting and turning with the ball. In football it is normal when performing a COD with the ball, to automatically perform a feint. This is to deceive a possible opponent. These feints also included many fake shots/passes. Hence a player acts like he will pass/shoot the ball, but instead performs a COD with the ball. This could result in the high number of classifies passes/shots, which are in fact fake passes or body feints while performing a COD. This can also be seen by the amount of COD's classified in the model, which is a lot lower than the actual number. Interesting to see is the performance of the run intensities. There are a lot of fluctuations visible between low, medium and high intensity. This makes sense, since the participants were expected to dribble at different speeds and were expected to sprint to retrieve the ball after shooting it.

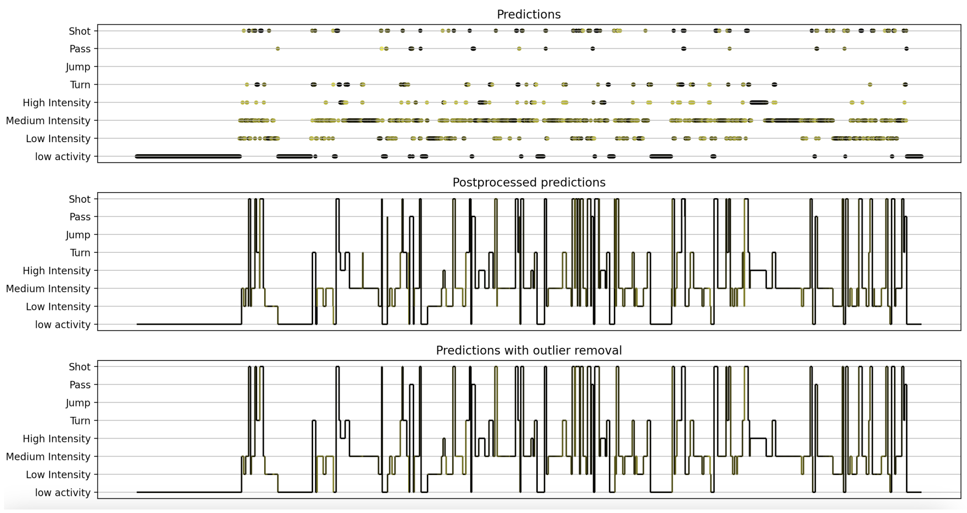


Figure 6.21: Evaluation results of the second exercise.

The final exercise that was analysed is again a passing and turning exercise. This is shown in Figure 6.22. In this exercise, two of the participants stand next to each other with the ball. The other participant stand between them and the boarding on the side of the field, without a ball. One of the two participants passes the ball, the one receiving it controls the ball, turns around with the ball and passes against the boarding. Once he receives the back back from the boarding, he controls it, turns around and passes back to the first player. Then the second player passes the ball and the procedure repeats itself. After a while, one of the passers switches with the participant standing in the middle. This means that each participant is a passer twice and once the person standing in the middle. The total activity takes around 4 minutes.

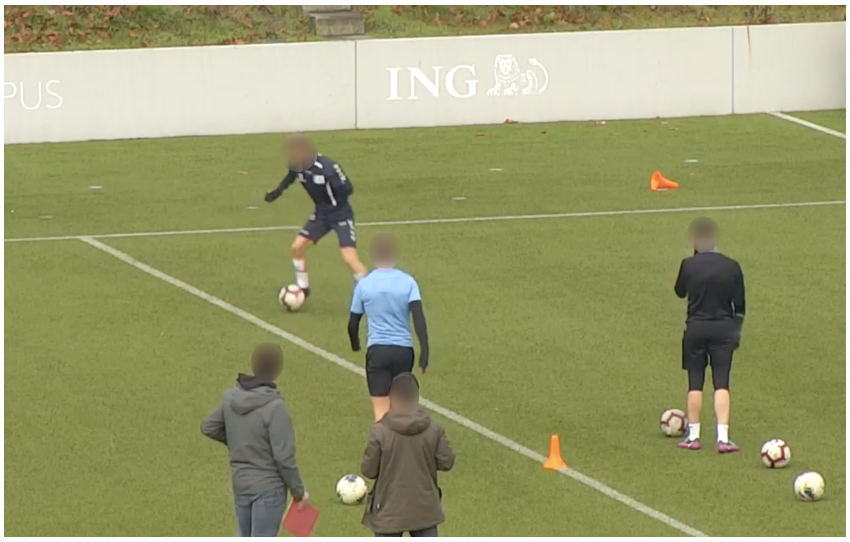


Figure 6.22: Third exercise: Two participants passing the ball, while the third receives it, turns bounces the ball against the boarding, turns back and passes it back to the other participants.

The results are shown in Figure 6.23. In this case, the participant is first a passer, then the man in the middle and then a passer again. As a passer, the model classified a total of 10 passes. This is however not enough, since total of 12 passes were given. This could again be explained by how the passes were given. For most of those passes, the participant only used his shank to pass, while keeping his thigh almost still. This could cause the model to have difficulties detecting these passes. An interesting detail is the short periods of medium run intensity and a jump that is found in these intervals. These come from the passer trying to control the ball. Several times the man in the middle passed the ball back to the passer at some distance next to him. Hence the passer had to run and leap to be able to control the ball. This causes these short periods of run intensity or jumping.

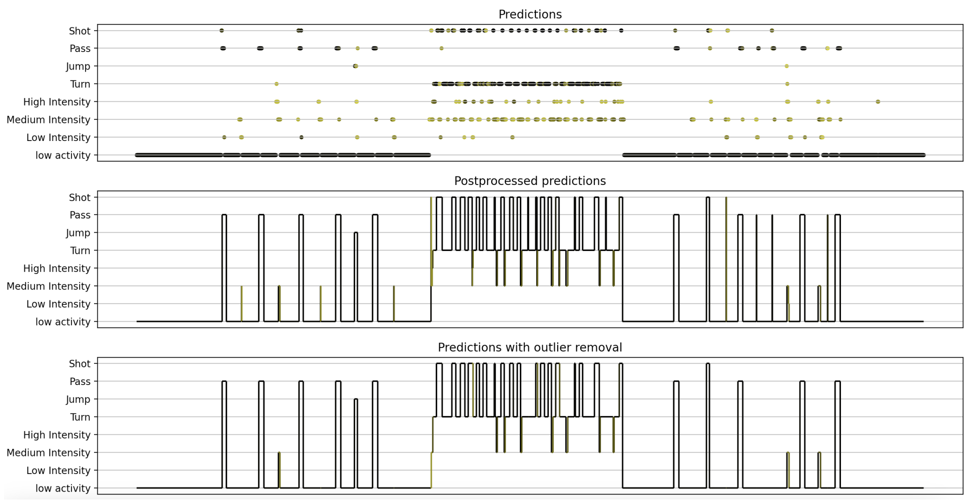


Figure 6.23: Evaluation results of the third exercise.

For the part where the participant is in the middle, it can be seen that many of the passes are correctly registered: 18 of the 24 passes in total. Also, between each pass, also a COD is detected. This is accurate for the exercise, where the participant had to turn around with the ball before passing. This turning with the ball could also be the reason that some of the passes are not classified. When receiving the ball, the participant in some occasions does the COD and the following pass in one flow. This could result in that the pass is not classified individually, but thought of it being a part of the COD. This is a similar problem in the previous exercise, where a COD with the ball was classified as a pass/shot due to the participant performing a feint.

6.2.5. Physiotherapy training with additional training

In the previous section, it was found that the current model performs relatively well in real life football situations. It is able to detect several small jumps, passes, runs and COD's. However, it was also seen that it missed many smaller movements, or movements performed at low intensity. A possible reason that was mentioned was the lack in training in smaller passes where the participant only uses his shank while keeping his thigh mostly still. Also the fact that the model was only trained with COD's of 180 degrees. For this reason, data from Bastiaansen 2021, explained in section 3.2.2 and analysed in section 6.2.3, was added to the training. This extra data consists of some small passes, increasing the amount of trainable pass windows with 27%, and cuts of 90 degrees, increasing the amount of trainable COD windows with 34%. These extra windows were manually labelled and added to the training data, securing that the are indeed the correct actions. Using this added training data, the previously used "2DCNN combined LSTM" model was trained again. This model was now used to analyse the physiotherapy training once again.

Starting with the first exercise, the three participants passing the ball in a sort of triangle with three pawns, moving to the next pawn after each pass. The results are shown in Figure 6.24. When compared to the results seen in Figure 6.17, it seems that this classification gives more details. Here the model detects a total of 21 COD's, while in the previous model it recognized only 4. This would suggest the training the model with extra COD's is indeed beneficial for the recognition of them. On the other hand, the amount of classified passes stayed the same, 13 in both cases. This suggests that the additional training has not had its intended effect. Interesting to note here is that in the model shown in Figure 6.17, it recognizes most passes in the first half of the classification. However, in the new model shown in Figure 6.24, it recognizes more passes in the second half.

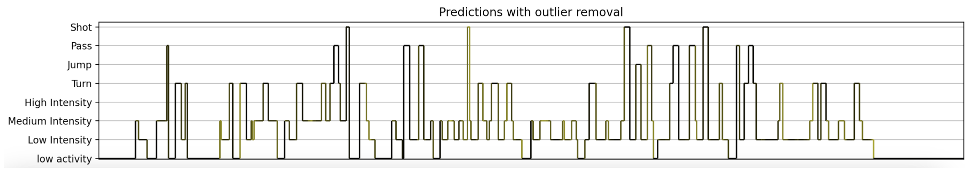


Figure 6.24: Evaluation results of the first exercise, with additional training.

For the next exercise, we will again look at the dribbling with shots at goal. The results are shown in Figure 6.25. Also here it seems that the model is much more focused on the COD's. Where in Figure 6.21, the model classified a total of 34 shots/passes, it now only detects 20. Moreover, the amount of COD's seemed to have increased substantially in Figure 6.25. Previously, the problem occurred that dribbling and turning with the ball resulted in wrongly classified passe or shots. With the additional training, it seems that the model can recognize this distinction between COD's with ball and a shot better.

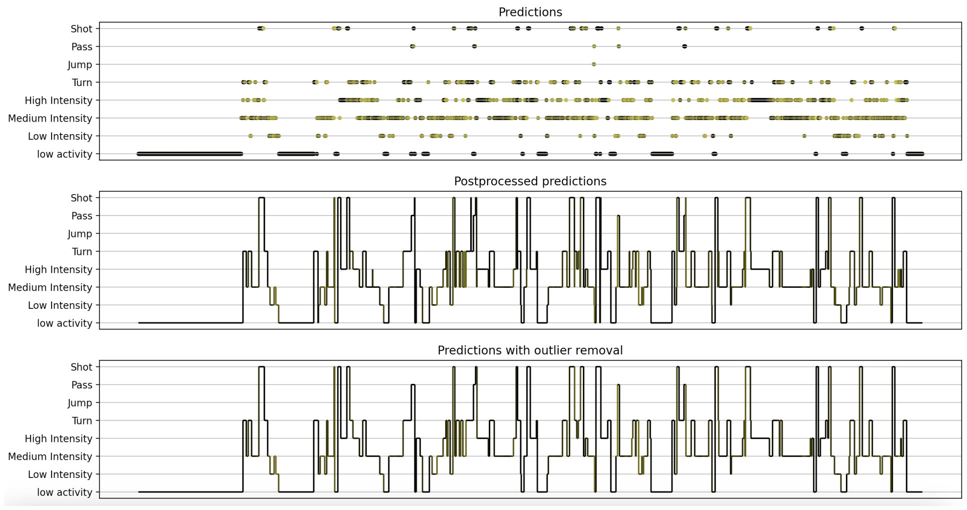


Figure 6.25: Evaluation results of the second exercise, with additional training.

The last exercise is again the passing and turning exercise, where two participants had to pass the ball to a third, who then turned with the ball, passed against a fence, again turned with the ball and then passed it back. The new results are shown in Figure 6.26. Again, the results seem more accurate with extra training. Compared with the 18 passes found in Figure 6.23, it now finds 23 of the 24 passes when the participant is the man in the middle. And again between each pass, a COD is detected. When the participant is one of the passers however, something strange happens. The model detects a total of 19 passes, while the footage showed only 12 were given. These "extra" passes might be the result of the participant trying to control the ball when it is passed back to him, trying to reach it or to make sure that the ball is still right after he controls it.

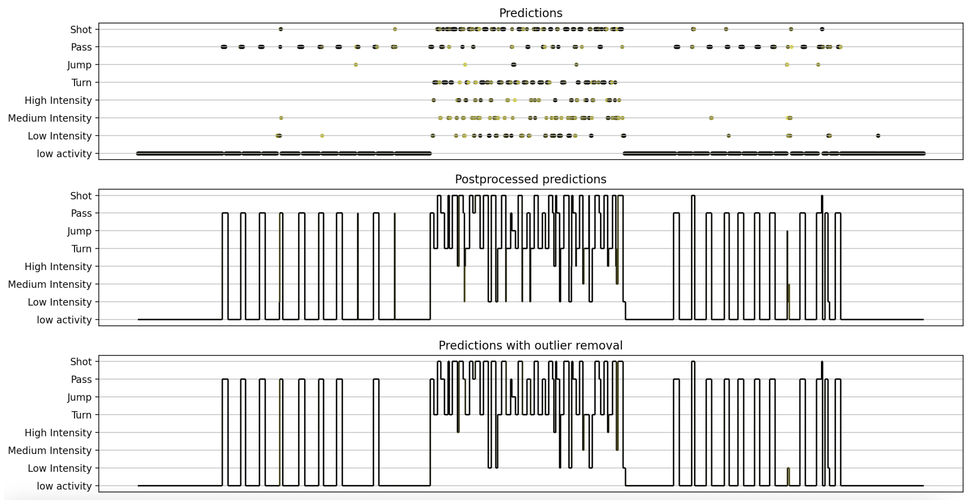


Figure 6.26: Evaluation results of the third exercise, with additional training.

6.2.6. Physiotherapy training excluding shank sensors

As mentioned in section 6.1.3, it could be interesting to consider what would happen when the shank sensors are excluded from the deep learning models. It could be noticed that the accuracy during training is barely affected by the exclusion. Now it is time to test this in a real training. For this reason a "2DCNN all Signals bLSTM" model was trained excluding the shank sensors and the exercises from the physiotherapy training, also evaluated in the previous sections, were evaluated. This model was chosen since in section 6.1.3 it turned out this model had the highest test accuracy. During this evaluation, the extra training data used in section 6.2.5 was excluded from training the model.

The result for the first exercise is shown in Figure 6.27. The first point is concern is the amount of detected passes by the model. In total only 7 of the 43 were recognized correctly. This is significantly less than seen in sections 6.2.4 and 6.2.5. This could again be the result of the way of passing. As mentioned several times before, during this exercise many passes were short passes where the participant only had to move his shank. Since these sensors are excluded, it is not strange that the model fails to detect these passes. On the other hand, it detects 14 COD's. This is significantly more than the original model tested in section 6.2.4. That could mean that the exclusion of shank sensors could give the model more time and space to train for smaller, more specific actions such as COD's and jumps.

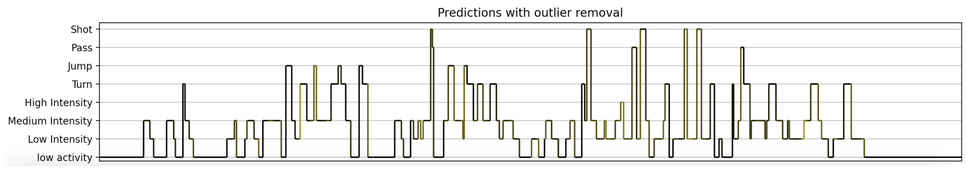


Figure 6.27: Evaluation results of the first exercise, when shank sensors are excluded in training and in evaluation.

The result of the second exercise, where the participants dribble with the ball and have to shoot at a small goal, is shown in Figure 6.28. Interesting to see here is that the model basically recognizes the whole exercise as a pass. Contrast to results seen in section 6.2.4, the model barely recognizes any types of running and turning. This could be the result of the dribbling done by the participants. During dribbling, a player essentially pushes the ball a small distance forward each time, which could be seen as a small pass. Due to this, the model recognizes the whole exercise, which consists for a large part of dribbling, as a pass. This shows that excluding shank sensors could negatively influence the model during recognition for activities it is not explicitly trained for, which is the case with dribbling.

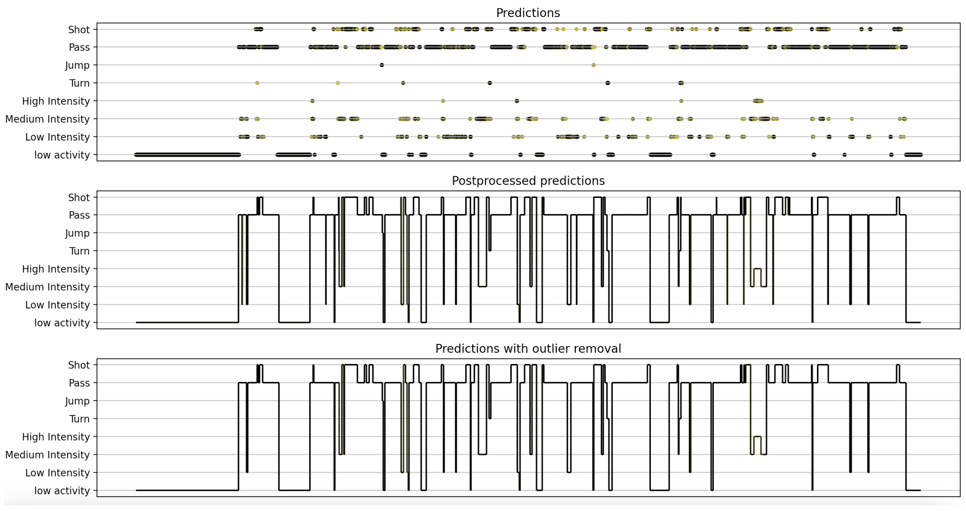


Figure 6.28: Evaluation results of the second exercise, when shank sensors are excluded in training and in evaluation.

For the final exercise, the results are shown in Figure 6.29. First, it can be noticed that when the participant is one of the passers, the model barely notices these passes. Only 5 of the total 12 passes are recognized by the model. Also during the exercise itself, it misses more of the passes given. Only 16 of the 24 passes were recognized. The could again be the result of the way of passing where only the shank moves, as explained several times before. Interesting detail to notice here is the high amount of "high intensity" prediction van be seen. This is consistent with what can be soon on the footage. After each pass, both to the board alongside the field and to a different participant, the participant very briefly speeds up in the passing direction. This normally involves only one or two steps, but is still picked up by this model, whereas the models in sections 6.2.4 and 6.2.5 only detected this sporadically.

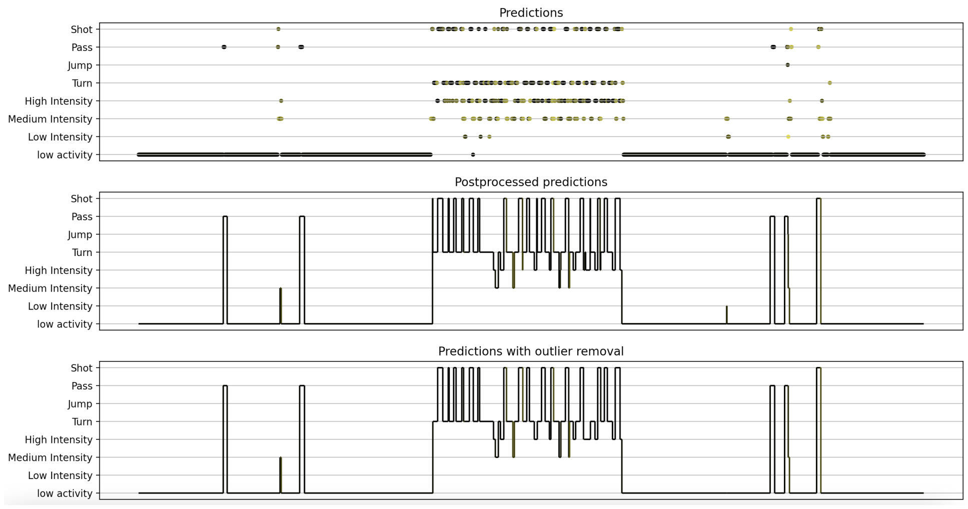


Figure 6.29: Evaluation results of the third exercise, when shank sensors are excluded in training and in evaluation.

7 Conclusions and discussion

Now that the analysis is concluded and results are presented, it is time to draw conclusions from them. Here the results will be shortly summarized, conclusions will be drawn and they will be discussed. In this discussion, ideas will be given for continuing and improving this project.

Starting with the results obtained by the training. As seen, adding change of directions to the list of possible actions and replacing jog and sprint with a continues spectrum of run intensities based on speed, lowered the training accuracy on the test data significantly. Whereas first the deep learning models reached an accuracy of 0.9778 on the test data, it now only went as high as 0.8889. This decrease is easily explained by the fact that run speed is now viewed as a spectrum. In the setting at the start of this thesis, only jogs and sprint were available activities. However, that leaves a lot of space for other kinds of run intensities. This space is filled by putting run intensity in 4 categories: jogging (< 15 km/h), running ($15-19.8$), running at high speed ($19.8-25.1$) and sprint (> 25.1). Due to lack of data, the fastest category was excluded in this thesis. Per run data, the intensity was computed by use of VICON sensors. These give accurate location datapoint, so that per second the speed can be determined. The speed and thus intensity of a run was then decided by the average of all those intermediate running speeds. The downside of involving all types of runs, is that two runs can be almost similar in speed and intensity, but still be classified differently.

A possible solution for this problem could be to add an extra step in the sliding window evaluation method. As seen in section 5.1.5, when a model is trained with the soul purpose of classifying run intensities, this model could reach a test accuracy of 95%. This indicates that when a deep learning model does not have to think about several different activities but can purely focus on recognizing run speeds, it is able to train a lot better. To use this, the sliding window evaluation method could be adjusted in the following way. Instead of training a deep learning model to train the activities shots, passes, jumps, COD's and low/medium/high intensity running, it would train the activities shots, passes, jumps, COD's and general running. This running should include all different kinds of run speeds and run intensities. Furthermore, another deep learning model should be trained which can only classify run intensities. The evaluation should then go as follows. All windows should be classified by the first model. After post-processing, each window is classified as a specific activity. Is this activity running, then that window will go through the second model to classify which intensity. This could improve the accuracy for training and classification of run intensities. This approach was tried during this thesis, but it was abandoned due to time constraints and the computer power it takes to classify a big dataset using two types of models simultaneously.

Another way of improving results is by changing the input data. The IMU's used to capture the required data also have magnetometers. These are used to compute the angle between certain movements. Now this data is excluded from training, since only accelerometer and gyroscope data is used. Now that activities such COD's are included in the activities, it might be interesting to see if magnetometers could have a beneficial effect. A different possibility is to normalize the signals. This was briefly discussed in [2] and gave promising results. However during this thesis, the signals were not normalized. Normalizing signals could possibly contribute to higher test accuracy and better classification.

Also, COD's were introduced as possible activities. This involved detecting the area in the data where the actual COD took place. This was solved using Extremes height difference method. Here the goal is to filter the area where the height difference between consecutive minima and maxima is minimal. Using this method, all 97.6& of all COD windows were found correctly. This however could only been done for COD's of 180, since the height difference for 90 degree angles is not significant enough to detect using this method. Even though the deep learning models were only trained for 180 degree COD's, sliding window evaluation showed that it is still able to detect 90 degree COD's.

After training the models, they were used to evaluate larger datasets using the sliding window method. First this was done with isolated activities. Here the models worked well to correctly classify all activities, including run intensities. Afterwards, an football drill was analyzed. This contained multiple football related activities such as passing, heading/jumping and run in zigzags. The downside of this dataset was that it contained only one participant, which made it difficult to verify results. It was found that activities as passing and running in zigzags was mostly classified correctly. For the zigzag, this might be surprising. Running in zigzag is a combination of short runs and 90 COD's. Even tough the model is only trained for 180 degree COD's, the 90 degree COD's were still classified correctly. On the other hand, the classifications for heading activities were less successful. This could be result of training. During training, only regular jumps without ball were trained. However, when heading a ball, players tend to make a different kind of movement to give the ball some extra velocity. This detail could be included in the training phase to make jumps and headers easier to detect.

With the analysis for the football drill completed, it was time to move on to an even more realistic setting. This time, it is a physiotherapy training for football players. Here the participants had to do several different football related exercises. These exercises involved dribbling, passing, running and COD's. The results showed that running, dribbling and most COD's were detected well by the model. The problem lies in the passing and COD's at low intensity or COD's with the ball. The lack in pass detection is most likely the result of the who of passing. During most short passes, players solely use their lower legs, while keeping their thighs mostly still. Moreover, while dribbling passes were smoothly integrated in the dribble and were therefore no big movements. These reasons could explain the amount of missed passes.

This confusion with passing is also notable in other areas. During dribbling, when touching the ball the model occasionally recognizes this as a pass. Also when controlling the ball or performing a COD with the ball, the model occasionally classifies at as a pass. This could best be seen in an exercise were the participants were required to dribble with the ball randomly, and shoot/pass at a small goal when the trainer gave the signal. In this exercise, the model recognized a total of 34 passes/shots, while only 14 were performed. Potentially this could be solved by adding more trainingdata containing exercises with the ball, including dribbling, COD's with ball, short passing.

To test whether extra trainingdata would indeed help, COD and passing data from the football drill were added as trainingdata and using that model the physiotherapy training was analyzed again. The extra training resulted in better classification for both passes and COD's. In one exercise, the original model only found 18 of the 24 passes, while the model with extra training detected 23 of the 24 passes. In a different exercise with the dribbling, the original model only found substantially less COD's than the newly trained model. Also the aforementioned overshoot of pass predictions was decreased radically, since the new model only detected 20 passes/shots instead of 34. All this indicates that extra training would indeed help the model increase its classification accuracy. The problem here lies in the extraction of more trainingdata. The trainingdata taken from the football drill was added and labeled manually. This can be a tedious and time consuming assignment, while in the ideal case this would be done automatically. Finding a way to automate this would be an important step into added extra trainingdata to the model.

Lastly, it was analyzed what would happen if the data coming from the shank sensors is excluded from the training and evaluation. This is done with the intention whether sports shorts with build-in sensors could potentially exclude shank sensors, such that the pants do not cover the whole leg. The training gave promising results in this regard. Where the original models with all 5 sensors had an test accuracy up to 0.8889, excluding shank sensors gave accuracies up to 0.8869. This is only slightly worse. The negative side here is that the standard deviation is slightly higher, which makes training and using such a model more unreliable. When using this newly trained model excluding shank data was used to evaluate the physiotherapy training, several bigger problems came to light. It could be seen that the amount of detected passes decreased drastically. Moreover, the model does not seem to recognize dribbling as a run, but as a constant form of passing. This problem can be explained similarly as in the original model, namely the short passes. As mentioned, during this type of passing, only the lower part of the leg moves, while the thighs stay mostly still. Excluding shank data and therefore all information from the lower legs is not taken into account. Extra training with these type of short passes could increase accuracy, but even with the additional training, these types of movements seem to be difficult to detect when excluding shank sensors.

In conclusion, the deep learning models, the training and the evaluation using the sliding window method give promising results. Training and test accuracy seems reliable. When evaluating larger datasets, the models are able to detect and classify many actions correctly, both bigger and smaller actions. However, before it can be used in practice, there are still several crucial steps to be taken. Firstly, the trainingdata. So far that data only includes simple, isolated activities, such as running, COD's, jumping and shooting. Except for shooting, no ball is involved in the rest of the activities. However during realistic football situations in training and games the ball is way more involved, and thus should the models also be trained more for activities including the ball. Another big step would be to extract trainingdata from larger sets of data. As mentioned, for extracting data from the football drill to add to the trainingdata, labeling and adding had to be done manually. Ideally this should be automated as much as possible. If these two previous steps are taken successfully, football activity recognition with sensor data using deep learning should be useful and applicable in real life situations.

Bibliography

[1] https://olympics.com/en/sports/football/

[2] Cuperman Coifman, R. (2021). *Football activity recognition: A deep learning approach to football activity recognition based on Inertial Measurement Units signals.* Master thesis at Delft University of Technology.

[3] Slim, S., Atia, A., Elfattah, M., Mostafa, M.-S. M. (2019). *Survey on human activity recognition based on acceleration data.* Intl. J. Adv. Comput. Sci. Appl, 10, 84–98.

[4] Kautz, T., Groh, B. H., Hannink, J., Jensen, U., Strubberg, H., Eskofier, B. M. (2017). *Activity recognition in beach volleyball using a deep convolutional neural network.* Data Mining and Knowledge Discovery, 31(6), 1678–1705.

[5] Jiao, L., Bie, R., Wu, H., Wei, Y., Ma, J., Umek, A., Kos, A. (2018). *Golf swing classification with multiple deep convolutional neural networks.* International Journal of Distributed Sensor Networks, 14(10), 1550147718802186.

[6] Hsu, Y.-L., Chang, H.-C., Chiu, Y.-J. (2019). *Wearable sport activity classification based on deep convolutional neural network.* IEEE Access, 7, 170199–170212.

[7] Campbell, A., Chai, K., Hendry, D., Hopper, L., O'Sullivan, P., Straker, L. (2020). *Development of a Human Activity Recognition System for Ballet Tasks.* Sports Medicine - Open volume 6, Article number: 10 (2020)

[8] Ha, S., Yun, J.-M., Choi, S. (2015). *Multi-modal convolutional neural networks for activity recognition.* 2015 IEEE International conference on systems, man, and cybernetics, 3017–3022.

[9] Ha, S., Choi, S. (2016). *Convolutional neural networks for human activity recognition using multiple accelerometer and gyroscope sensors.* 2016 International Joint Conference on Neural Networks (IJCNN), 381–388.

[10] Ordóñez, F. J., Roggen, D. (2016). *Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition.* Sensors, 16(1), 115.

[11] Hammerla, N. Y., Halloran, S., Plötz, T. (2016). *Deep, convolutional, and recurrent models for human activity recognition using wearables.* arXiv preprint arXiv:1604.08880.

[12] Murad, A., Pyun, J.-Y. (2017) *Deep recurrent neural networks for human activity recognition.* Sensors, 17(11), 2556.

[13] Lv, M., Xu, W., Chen, T. (2019). *A hybrid deep convolutional and recurrent neural network for complex activity recognition using multimodal sensors.* Neurocomputing, 362, 33–40.

[14] Gerats, B.G.A. (2020). *Individual action and group activity recognition in soccer videos.* Master thesis at University of Twente.

[15] Schuldhaus, D., Zwick, C., Körger, H., Dorschky, E., Kirk, R., Eskofier, B. M. (2015). *Inertial sensor based approach for shot/pass classification during a soccer match.* KDD workshop on large-scale sports analytics, 1–4.

[16] Kaketsis, G. (2020). *Classification in football: Activity classification using sensor data in football.* Master thesis at Delft University of Technology.

[17] Wilmes, E. (2019). *Measuring changes in hamstring contractile strength and lower body sprinting kinematics during a simulated soccer match.* Master thesis at Delft University of Technology.

[18] MathWorks. *What Is Deep Learning? 3 things you need to know* https://nl.mathworks.com/discovery/deep-learning.html

[19] Sharma, M (2020). *A brief introduction to perceptron.* https://becominghuman.ai/a-brief-introduction-to-perceptron-f3b9bade8f67

[20] Kuo, P.-H., Huang, C.-J. (2018). *A green energy application in energy management systems by an artificial intelligence-based solar radiation forecasting model.* Energies, 11, 819. https://doi.org/10.3390/en11040819

[21] MathWorks. *Introduction to Deep Learning: What Are Convolutional Neural Networks?* https://nl.mathworks.com/videos/introduction-to-deep-learning-what-are-convolutional-neural-networks–1489512765771.html

[22] Lang, C., Lang, E. W., Steffens, O., Steinborn, F. (2019). *Applying a 1d-cnn network to electricity load forecasting.* International Conference on Time Series and Forecasting, 205–218.

[23] Olah, C. (2015). *Understanding lstm networks.* http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[24] Arbel, N. (2018). *How lstm networks solve the problem of vanishing gradients.* https://medium.datadriveninvestor.com/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577

[25] Baeldung (2022) *Differences Between Bidirectional and Unidirectional LSTM* https://www.baeldung.com/cs/bidirectional-vs-unidirectional-lstm

[26] Baeldung (2022) *How ReLU and Dropout Layers Work in CNNs* https://www.baeldung.com/cs/ml-relu-dropout-layers

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

[28] Ben-David, S., Shalev-Shwartz, S. (2014). *Understanding Machine Learning: From Theory to Algorithms.* Cambridge University Press, http://www.cs.huji.ac.il/ shais/UnderstandingMachineLearning.

[29] Chawla, N. V., Bowyer, K. W., Hall, L. O., Kegelmeyer, W. P. (2002). *Smote: Synthetic minority over-sampling technique.* Journal of artificial intelligence research, 16, 321–357.

[30] Comfort, P., Dos Santos, T., Jones, P.A., Thomas, C. (2018). *The Effect of Training Interventions on Change of Direction Biomechanics Associated with Increased Anterior Cruciate Ligament Loading: A Scoping Review.* Sports Med. 2019; 49(12): 1837–1859.

[31] Transfermarkt. *Ronaldo: Blessurehistorie.* https://www.transfermarkt.nl/ronaldo/verletzungen/spieler/3140

[32] Comfort, P., Dos Santos, T., Jones, P.A., Thomas, C. (2018). *The Effect of Angle and Velocity on Change of Direction Biomechanics: An Angle-Velocity Trade-Off.* Sports Medicine (2018) 48:2235–2253.

[33] Planet Science. *How fast is Usain Bolt?* http://www.planet-science.com/categories/over-11s/human-body/2012/06/how-fast-is-usain-bolt/

[34] Sayer, A. (2022). *What's The Average Human Sprint Speed? + Top Sprint Speeds* https://marathonhandbook.com/average-human-sprint-speed/

[35] Gualtieri A, Rampinini E, Dello Iacono A, Beato M. (2023). *High-speed running and sprinting in professional adult soccer: Current thresholds definition, match demands and training strategies. A systematic review* Front Sports Act Living. 2023 Feb 13;5:1116293. doi: 10.3389/fspor.2023.1116293. PMID: 36860737; PMCID: PMC9968809.

[36] Stucovitz, E., Vitale, J., Galbusera, F. (2018). *Biomechanics of the Spine. Basic Concepts, Spinal Disorders and Treatments* Chapter 11 - In Vivo Measurements: Motion Analysis. https://doi.org/10.1016/B978-0-12-812851-0.00011-2

A
Used models

Here all deep learning models used for training are presented, including the chosen parameters.

- 1D CNN weight sharing: model based on 1DCNN weight sharing type of convolution.
  Architecture: Input + Conv(16, (1,5), 1, relu) + MaxPool(1,4) + Conv(32, (1,5), 1, relu) + MaxPool(1,4) + Conv(64, (d,1), 1, relu) + Flatten + Dropout(0.2), FC(64, relu) + Dropout(0.5) + FC(classes, softmax)
- 1D CNN per sensor: model based on 1DCNN per sensor type of convolution.
  Architecture: Input + Conv_perSensor(16, (1,5), 1, relu) + MaxPool(1,4) + Conv_perSensor(32, (1,5), 1, relu) + MaxPool (1,4) + Conv(64, (d,1), 1, relu) + Flatten + Dropout(0.2), FC(64, relu) + Dropout(0.5) + FC(classes, softmax)
- 1D CNN combined: model based on 1DCNN combined type of convolution.
  Architecture: Input + Concatenate[Conv(16, (1,5), 1, relu) + MaxPool(1,4) + Conv(32, (1,5), 1, relu) + MaxPool(1,4)], [Conv_perSensor(16, (1,5), 1, relu) + MaxPool(1,4) + Conv_perSensor(32, 1,5), 1, relu) + MaxPool (1,4)] + Conv(64, (d,1), 1, relu) + Flatten + Dropout(0.2), FC(64, relu) + Dropout(0.5) + FC(classes, softmax)
- 2D CNN weight sharing: model based on 2DCNN weight sharing type of convolution.
  Architecture: Input + Conv(16, (3,5), 3, relu) + MaxPool(1,4) + Conv(32, (1,5), 1, relu) + MaxPool(1,4) + Conv(16, (d,1), 1, relu) + Flatten + Dropout(0.2) + FC(64, relu) + Dropout(0.5) + FC(classes, softmax)
- 2D CNN per sensor: model based on 2DCNN per sensor type of convolution.
  Architecture: Input + Conv_perSensor(16, (3,5), 3, relu) + MaxPool(1,4) + Conv_perSensor(32, (1,5), 1, relu) + MaxPool(1,4) + Conv(32, (d,1), 1, relu) + Flatten + Dropout(0.2) + FC(64, relu) + Dropout(0.5) + FC(classes, softmax)
- 2D CNN all sensors: model based on 2DCNN all sensors type of convolution.
  Architecture: Input + Conv(32, (d,5), 1, relu) + MaxPool(1,4) + Conv(32, (1,5), 1, relu) + MaxPool(1,4) + Flatten + Dropout(0.2) + FC(64, relu) + Dropout(0.5) + FC(classes, softmax)
- 2D CNN combined: model based on 2DCNN combined type of convolution.
  Architecture: Input + Concatenate[Conv(16, (3,5), 3, relu) + MaxPool(1,4) + Conv(32, (1,5), 1, relu) + MaxPool(1,4)], [Conv_perSensor(16, (3,5), 3, relu) + MaxPool(1,4) + Conv_perSensor(32, (1,5), 1, relu) + MaxPool(1,4)], [Conv(32, (d,5), 1, relu) + MaxPool(1,4) + Conv(32, (1,5), 1, relu) + MaxPool(1,4)] + Conv(64, (d,1), 1, relu) + Flatten + Dropout(0.2) + FC(64, relu) + Dropout(0.5) + FC(classes, softmax)
- 1D CNN weight sharing + LSTM: model based on 1DCNN weight sharing type of convolution followed by LSTM.
  Architecture: Input + Conv(16, (1,5), 1, relu) + MaxPool(1,4) + Conv(32, (1,5), 1, relu) + MaxPool(1,4) + Conv(128, (d,1), 1, relu) + LSTM(128) + LSTM(128) + Dropout(0.3) + FC(128, relu) + Dropout(0.5) + FC(classes, softmax)

- 1D CNN per sensor + LSTM: model based on 1DCNN per sensor type of convolution followed by LSTM.
  Architecture: Input + Conv_perSensor(16, (1,5), 1, relu) + MaxPool(1,4) + Conv_perSensor(32, (1,5), 1, relu) + MaxPool(1,4) + Conv(128, (d,1), 1, relu) + LSTM(128) + LSTM(128) + Dropout(0.3) + FC(128, relu) + Dropout(0.5) + FC(classes, softmax)

- 1D CNN combined + LSTM: model based on 1DCNN combined type of convolution followed by LSTM.
  Architecture: Input + Concatenate[Conv(16, (1,5), 1, relu) + MaxPool(1,4) + Conv(32, (1,5), 1, relu) + MaxPool(1,4)], [Conv_perSensor(16, (1,5), 1, relu) + MaxPool(1,4) + Conv_perSensor(32, (1,5), 1, relu) + MaxPool(1,4)] + Conv(128, (d,1), 1, relu) + LSTM(128) + LSTM(128) + Dropout(0.3) + FC(128, relu) + Dropout(0.5) + FC(classes, softmax)

- 2D CNN weight sharing + LSTM: model based on 2DCNN weight sharing type of convolution followed by LSTM.
  Architecture: Input + Conv(32, (3,5), 3, relu) + MaxPool(1,4) + Conv(64, (1,5), 1, relu) + MaxPool(1,4) + Conv(128, (d,1), 1, relu) + LSTM(128) + LSTM(128) + Dropout(0.3) + FC(128, relu) + Dropout(0.5) + FC(classes, softmax)

- 2D CNN per sensor + LSTM: model based on 2DCNN per sensor type of convolution followed by LSTM.
  Architecture: Input + Conv_perSensor(32, (3,5), 3, relu) + MaxPool(1,4) + Conv_perSensor(64, (1,5), 1, relu) + MaxPool(1,4) + Conv(128, (d,1), 1, relu) + LSTM(128) + LSTM(128) + Dropout(0.3) + FC(128, relu) + Dropout(0.5) + FC(classes, softmax)

- 2D CNN all sensors + LSTM: model based on 2DCNN all signals type of convolution followed by LSTM.
  Architecture: Input + Conv(32, (d,5), 1, relu) + MaxPool(1,4) + Conv(64, (1,5), 1, relu) + MaxPool(1,4) + LSTM(128) + LSTM(128) + Dropout(0.3) + FC(128, relu) + Dropout(0.5) + FC(classes, softmax)

- 2D CNN combined + LSTM: model based on 2DCNN combined type of convolution followed by LSTM.
  Architecture: Input + Concatenate[Conv(32, (3,5), 3, relu) + MaxPool(1,4) + Conv(64, (1,5), 1, relu) + MaxPool(1,4)], [Conv_perSensor(32, (3,5), 3, relu) + MaxPool(1,4) + Conv_perSensor(64, (1,5), 1, relu) + MaxPool(1,4)], [Conv(32, (d,5), 1, relu) + MaxPool(1,4) + Conv(64, (1,5), 1, relu) + MaxPool(1,4)] + Conv(128, (d,1), 1, relu) + LSTM(128) + LSTM(128) + Dropout(0.3) + FC(128, relu) + Dropout(0.5) + FC(classes, softmax)

- 1D CNN weight sharing + bLSTM: model based on 1DCNN weight sharing type of convolution followed by bidirectional LSTM.
  Architecture: Input + Conv(16, (1,5), 1, relu) + MaxPool(1,4) + Conv(32, (1,5), 1, relu) + MaxPool(1,4) + Conv(128, (d,1), 1, relu) + bLSTM(128) + bLSTM(128) + Dropout(0.3) + FC(128, relu) + Dropout(0.5) + FC(classes, softmax)

- 1D CNN per sensor + bLSTM: model based on 1DCNN per sensor type of convolution followed by bidirectional LSTM.
  Architecture: Input + Conv_perSensor(16, (1,5), 1, relu) + MaxPool(1,4) + Conv_perSensor(32, (1,5), 1, relu) + MaxPool(1,4) + Conv(128, (d,1), 1, relu) + bLSTM(128) + bLSTM(128) + Dropout(0.3) + FC(128, relu) + Dropout(0.5) + FC(classes, softmax)

- 1D CNN combined + bLSTM: model based on 1DCNN combined type of convolution followed by bidirectional LSTM.
  Architecture: Input + Concatenate[Conv(16, (1,5), 1, relu) + MaxPool(1,4) + Conv(32, (1,5), 1, relu) + MaxPool(1,4)], [Conv_perSensor(16, (1,5), 1, relu) + MaxPool(1,4) + Conv_perSensor(32, (1,5), 1, relu) + MaxPool(1,4)] + Conv(128, (d,1), 1, relu) + bLSTM(128) + bLSTM(128) + Dropout(0.3) + FC(128, relu) + Dropout(0.5) + FC(classes, softmax)

- 2D CNN weight sharing + bLSTM: model based on 2DCNN weight sharing type of convolution followed by bidirectional LSTM.
  Architecture: Input + Conv(32, (3,5), 3, relu) + MaxPool(1,4) + Conv(64, (1,5), 1, relu) + MaxPool(1,4) + Conv(128, (d,1), 1, relu) + bLSTM(128) + bLSTM(128) + Dropout(0.3) + FC(128, relu) + Dropout(0.5) + FC(classes, softmax)

- 2D CNN per sensor + bLSTM: model based on 2DCNN per sensor type of convolution followed by bidirectional LSTM.

Architecture: Input + Conv_perSensor(32, (3,5), 3, relu) + MaxPool(1,4) + Conv_perSensor(64, (1,5), 1, relu) + MaxPool(1,4) + Conv(128, (d,1), 1, relu) + bLSTM(128) + bLSTM(128) + Dropout(0.3) + FC(128, relu) + Dropout(0.5) + FC(classes, softmax)

- 2D CNN all sensors + LSTM: model based on 2DCNN all signals type of convolution followed by bidirectional LSTM.
  Architecture: Input + Conv(32, (d,5), 1, relu) + MaxPool(1,4) + Conv(64, (1,5), 1, relu) + MaxPool(1,4) + bLSTM(128) + bLSTM(128) + Dropout(0.3) + FC(128, relu) + Dropout(0.5) + FC(classes, softmax)
- 2D CNN combined + bLSTM: model based on 2DCNN combined type of convolution followed by bidirectional LSTM.
  Architecture: Input + Concatenate[Conv(32, (3,5), 3, relu) + MaxPool(1,4) + Conv(64, (1,5), 1, relu) + MaxPool(1,4)], [Conv_perSensor(32, (3,5), 3, relu) + MaxPool(1,4) + Conv_perSensor(64, (1,5), 1, relu) + MaxPool(1,4)], [Conv(32, (d,5), 1, relu) + MaxPool(1,4) + Conv(64, (1,5), 1, relu) + MaxPool(1,4)] + Conv(128, (d,1), 1, relu) + bLSTM(128) + bLSTM(128) + Dropout(0.3) + FC(128, relu) + Dropout(0.5) + FC(classes, softmax)
- LSTM: model based on LSTM.
  Architecture: Input + LSTM(128) + LSTM(128) + Dropout(0.2) + FC(64, relu) + Dropout(0.5) + FC(64, relu) + Dropout(0.3) + FC(classes, softmax)
- bLSTM: model based on bidirectional LSTM.
  Architecture: Input + bLSTM(128) + bLSTM(128) + Dropout(0.2) + FC(64, relu) + Dropout(0.5) + FC(64, relu) + Dropout(0.3) + FC(classes, softmax)

For the previous list, the following notation was used:

- Conv$(f,(m,n),s,\sigma)$: Convolutional layer with f filters of size (m,n) with s strides in the vertical (spatial direction) followed with σ activation function.
- Conv_perSensor$(f,(m,n),s,\sigma)$: Convolutional layer applied independently at each sensor (X, Y and Z) with f filters of size (m,n) with s strides in the vertical (spatial direction) followed with σ activation function.
- MaxPool(m,n): Max Pooling layer with filters of size (m,n).
- LSTM(n): LSTM layer of n units.
- bLSTM(n): Bidirectional LSTM layer of n units.
- FC(n,σ): Fully connected feed-forward layer of n units followed with σ activation function.
- d: Number of spatial dimensions of the output tensor of the previous layer.
- $classes$: Number of classes.