Football activity recognition:

A deep learning approach to football activity recognition based on Inertial Measurement Units signals

Rafael Cuperman Coifman

TU Delft

Football activity recognition:

A deep learning approach to football activity recognition based on Inertial Measurement Units signals

by

Rafael Cuperman Coifman

to obtain the degree of
Master of Science
in Applied Mathematics
track Computational Science and Engineering
at the Delft University of Technology,
to be defended publicly on June 30, 2021

Student number:	5065461	
Thesis committee:	Dr. Ir. M.B. van Gijzen,	TU Delft
	Dr. J. Söhl	TU Delft
	Prof. Dr. Ir. K.M.B. Jansen	Daily Supervisor, TU Delft
	M.Sc. M. Ciszewski	TU Delft

TU Delft

Preface

When looking for a topic for my master thesis, I came across this project. When I read its description and saw that it consisted of researching ways to classify football activities, I know it was meant for me. It was just perfect. As a biomedical engineer, I loved the idea of working around a topic that had to do with the human body; as an electronics engineer, the usage of sensors placed on body parts drew my attention; as a soon-to-be mathematician with special interest in artificial intelligence, researching on machine (deep) learning was one of my main goals; and as a football (soccer) and sports enthusiast and player, I would be applying my work in one of the things that I enjoy the most. Finding a master thesis can be difficult, but this project was exactly what I was looking for. It was not an easy path, but a very rewarding one.

This report contains the entire work that I accomplished during my Master's Thesis to fulfill the graduation requirements of the MSc. Applied Mathematics program at the Delft Institute of Technology. In this project, I investigated the usage of different architectures of deep learning models to perform Human Activity Recognition (HAR) for specific football activities based on signals obtained by Inertial Measurement Unit (IMU) sensors placed on different body parts of players. The objective was to develop a robust end-to-end pipeline that was capable of robustly recognizing football activities in a fast and accurate way.

Previous works demonstrated that deep learning approaches were promising in HAR, but they mainly focused on daily human activities and not in specific football movements, where a combination of periodic and explosive activities are present. Furthermore, the reviewed research papers specially addressed the construction of a specific model architecture, without paying much attention to the online evaluation phase of a new recording. Through this thesis, I researched the end-to-end process of building a Football Activity Recognition pipeline. Several deep architectures were designed, tested and compared. The built models achieved accuracies on the test dataset of up to 96.71%. When the input signals were normalized with a calibration recording, those values increased to 98.2%. Additionally, a robust evaluation process was proposed that allowed to obtain accurate and fast results over unseen IMU recordings.

First of all, I would like express my deep gratitude to Professor Dr. Ir. Kaspar Jansen and to PhD candidate Michał Ciszewski for their daily supervision and guidance. Their encouraging feedback, analysis, immeasurable knowledge, and ideas made this project possible and, without any doubt, helped me to accomplish the objectives. I would also like to thank Professors Dr. Ir. Martin van Gijzen and Dr. Jakob Söhl, for giving me the opportunity to work in such an enriching project. Many thanks to the CAS-P6 resarch group, with a special mention to PhD candidates Erik Wilmes and Bram Bastiaansen for facilitating me the data that this project required.

If you know me, you know how important family is to me. This is why I cannot stress enough how grateful I am with them. To my parents, brother and grandparents, this milestone is yours. Being far from home is never easy, but your support and love are felt from kilometers away. Thank you for always encouraging me to push myself and for believing in me no matter what. I am what I am because of you.

A life without friends is pointless. To Male, Alejo, Diego, and Mateo, I can only tell you that you are and will always be a family to me. You made this experience one of the best of my life and I will always be there for you. Last, but not least, to my friends in Colombia, the Netherlands, and around the world for cheering me up with your encouragement and joy. You always made me believe in myself.

Rafael Cuperman Coifman
Delft, June 2021

Abstract

With the latest developments in commercial sensors during the last years, special interest has been given to Human Activity Recognition (HAR) based on signals obtained by IMUs placed on different body parts. This thesis studies the usage of Deep Learning-based models to recognize different football activities in an accurate, robust, and fast manner. Several deep architectures were trained with data captured with IMU sensors placed on football players' bodies and their performances were compared. A combination of convolutional layers followed by recurrent (bidirectional) LSTM layers showed to achieve the best results with up to 96.71% of accuracy. When using normalized data via a calibration recording, these accuracies increased up to 98.2%. Results showed that deep learning models performed better in evaluation time and prediction accuracy than traditional machine learning algorithms.

An end-to-end pipeline for football activity recognition was developed that can be extended to any other HAR task. With it, not only the training was explored, but also a sliding window evaluation procedure was proposed that can be used to efficiently analyze unseen IMU recordings and recognize the activities there present. This pipeline showed to be fast and robust especially with signals consistently calibrated and can be used to recognize movements on a real-time basis. It can be concluded that a combination of deep learning models and a sliding window evaluation procedure is suitable for fast and accurate HAR tasks and can be used as input for research on injury prevention. By training and evaluating the models with more data, ideally from uncontrolled experiments such as football matches, we expect to further improve the generalization property of the classifiers.

List of abbreviations

ANN	Artificial Neural Network
bLSTM	Bidirectional LSTM
CART	Classification and Regression Tree
CNN	Convolutional Neural Network
DA	Discriminant Analysis
DT	Decision Tree
FC	Fully connected
FFT	Fast Fourier Transform
GMM	Gaussian Mixture Model
GRU	Gated Recurrent Unit
HAR	Human Activity Recognition
IMU	Inertial Measurement Unit
k-NN	k-Nearest Neighbor
LSTM	Long Short-Term Memory
NB	Naïve Bayes
RBF	Radial Basis Function
RF	Random Forest
RNN	Recurrent Neural Network
SVM	Support Vector Machine

List of Figures

3.1 IMUs and pants . . . 11
3.2 Simple artificial neuron model . . . 14
3.3 Artificial Neural Network composed by several fully connected layers . . . 14
3.4 Convolution operation on 1D data . . . 15
3.5 Convolutional Neural Network example. Convolutional and pooling layers one after the other followed by dense layers . . . 16
3.6 Convolutional Neural Network applied to 1D signal data . . . 16
3.7 Basic form of a Recurrent Neural Network . . . 17
3.8 Internal structure of an LSTM cell . . . 17
3.9 Adam optimization algorithm . . . 22

4.1 Location of IMUs in experiments . . . 23
4.2 Examples of activities from dataset from Wilmes, 2019 . . . 24
4.3 Examples of jumps . . . 25
4.4 Mean signals of dataset from Wilmes, 2019 . . . 25
4.5 Three different signals . . . 26
4.6 Two different signals (top) with their respective frequency spectrums (bottom) . . . 27
4.7 Time needed for evaluation with traditional and deep approaches . . . 28
4.8 Time needed for evaluation with traditional and deep approaches with respect to number of samples . . . 29
4.9 Example of starts and ends of an activity . . . 31
4.10 Overall start and end of activity from signals of figure 4.9 . . . 31
4.11 Distribution of IQRs for explosive and periodic movements. Un-normalized signals (top) and normalized signals (bottom) . . . 32
4.12 Confusion matrices for explosive vs periodic movements classifier based on IQR . . . 33
4.13 Examples of detected activities isolated from the low activity regions . . . 34
4.14 Boxplot of the duration in seconds of each activity after they were isolated from low-activity intervals. . . . 35
4.15 Mean signals of each activity after process of activity detection . . . 36

5.1 Full process of activity recognition with training and evaluation phases . . . 38
5.2 Training phase diagram . . . 39
5.3 1DCNN weight sharing convolution logic . . . 42
5.4 1DCNN per sensor convolution logic . . . 42
5.5 1DCNN combined convolution logic . . . 43
5.6 2DCNN weight sharing logic . . . 43
5.7 2DCNN per sensor logic . . . 44
5.8 2DCNN all sensors logic . . . 44
5.9 2DCNN combined logic . . . 44
5.10 General model architectures . . . 46
5.11 CNNs and RNNs sub-networks . . . 47
5.12 Unbalanced and balanced datasets for the training process . . . 50
5.13 Example of learning rate schedule for training . . . 51
5.14 Prediction accuracy and evaluation time for some deep learning-based models . . . 54
5.15 Prediction accuracy and evaluation time for traditional models in comparison to a deep learning-based model (DNN) . . . 55
5.16 Evaluation phase diagram . . . 56
5.17 Evaluation phase diagram . . . 57
5.18 Distributions of metrics of low and high activity periods . . . 59

5.19 F1 scores for different standard deviation thresholds when classifying high and low activity windows . . . 60
5.20 Evaluation phase diagram . . . 61
5.21 Predictions are not aligned when using a step larger than 1 . . . 62
5.22 Interpolate postprocessing option . . . 63
5.23 Mode postprocessing option . . . 64
5.24 Initial approach to perform the mode postprocessing option . . . 65
5.25 Improved approach to perform the mode postprocessing option . . . 66
5.26 Best score postprocessing option. . . . 67
5.27 Softmax activation function to obtain prediction confidences/scores . . . 67
5.28 Non-Max Suppression . . . 68
5.29 Sliding window evaluation procedure . . . 69
5.30 Final sliding window evaluation procedure with inclusion of “other high activity” class . . . 70
5.31 Example of results using interpolate postprocessing . . . 71
5.32 Example of results using mode postprocessing . . . 72
5.33 Example of results using best score postprocessing . . . 73
5.34 Example of outlier removal process after best score postprocessing . . . 74
5.35 Comparison of postprocessing options . . . 75
5.36 Example 1 of final results . . . 76
5.37 Example 2 of final results . . . 76
5.38 Example 3 of final results . . . 77
5.39 Summary of recognized activities of figure 5.37 . . . 77

6.1 Covariate shift . . . 83
6.2 Comparison of scales of measurements in Wilmes’ and Rozemarijn’s datasets. Sample accelerometer signal . . . 84
6.3 Comparison of scales of measurements in Wilmes’ and Rozemarijn’s datasets. Sample gyroscope signal . . . 84
6.4 Comparison of distributions in Wilmes’ and Rozemarijn’s datasets for selected signals . . . 85
6.5 Example of predictions made on part of Rozemarijn’s dataset with default values for hyperparameters . . . 86
6.6 Example of predictions made on part of Rozemarijn’s dataset without imposing a value for c_h . . . 87
6.7 Second example of predictions made on part of Rozemarijn’s dataset without imposing a value for c_h . . . 87
6.8 Evaluation of complete recording from Rozemarijn’s dataset . . . 88
6.9 Comparison of scales of measurements in Wilmes’ and new Wilmes’ datasets. Sample accelerometer signal . . . 89
6.10 Comparison of scales of measurements in Wilmes’ and Rozemarijn’s datasets. Sample gyroscope signal . . . 90
6.11 Comparison of distributions in Wilmes’ and new Wilmes’ datasets for selected signals . . . 90
6.12 Example of predictions made on part of new Wilmes’ dataset with default values for hyperparameters. Using best score postprocessing . . . 91
6.13 Example of predictions made on part of new Wilmes’ dataset with default values for hyperparameters. Using mode postprocessing . . . 91
6.14 Second example of predictions made on part of new Wilmes’ dataset with default values for hyperparameters. Using best score postprocessing . . . 92
6.15 Third example of predictions made on part of new Wilmes’ dataset. Using best score postprocessing . . . 92
6.16 Example of predictions made on part of Rozemarijn’s dataset using a normalized model 98
6.17 Example of predictions made on part of Rozemarijn’s dataset using an unnormalized model . . . 99

List of Tables

2.1 Traditional approaches for Human Activity Recognition 5
2.2 Deep kearning approaches for Human Activity Recognition 8

4.1 Time-domain manually selected features for the signals of figure 4.5 26

5.1 Mean prediction accuracies for the proposed models 52
5.2 Standard deviation of the prediction accuracies for the proposed models 53
5.3 Two-sided KS values for the metric distributions of high and low activity windows 58
5.4 Comparison of matrix sizes for both approaches when performing the mode postprocessing option 65

6.1 Mean and standard deviation prediction accuracies for the models without shanks. 5 runs 94
6.2 Comparison of mean prediction accuracies between the models with and without shanks 95
6.3 Comparison of mean prediction accuracies between the original unnormalized models and the normalized ones. 5 runs 96
6.4 Comparison of standard deviation of the prediction accuracies between the original unnormalized models and the normalized ones. 5 runs 97

Contents

1 Introduction **1**

2 Literature review **3**
2.1 Traditional approaches to HAR . . . 4
2.2 Deep Learning approaches to HAR . . . 7

3 Conceptual framework **11**
3.1 IMUs . . . 11
3.2 Traditional Machine Learning Algorithms . . . 12
3.3 Deep learning . . . 14
3.3.1 Convolutional Neural Networks . . . 15
3.3.2 Recurrent Neural Networks . . . 16
3.3.3 Additional Deep Learning Concepts . . . 18

4 Data overview and preparation **23**
4.1 Data . . . 23
4.2 Feasibility analysis of a deep learning approach . . . 26
4.3 Activity detection . . . 30

5 Activity recognition **37**
5.1 Activity recognition . . . 39
5.1.1 Window segmentation . . . 40
5.1.2 Proposed models . . . 40
5.1.3 Training . . . 49
5.1.4 Performance . . . 51
5.1.5 Evaluation times . . . 51
5.2 Low activity recognition . . . 56
5.3 Sliding window evaluation . . . 61
5.3.1 Prediction: activity recognition . . . 61
5.3.2 Postprocessing . . . 61
5.3.3 Outlier removal and “other high activity” recognition . . . 68
5.4 Results . . . 71

6 Discussion **78**
6.1 Evaluation on different datasets . . . 83
6.1.1 Rozemarijn’s dataset . . . 83
6.1.2 Wilmes’ new Dataset . . . 89
6.2 Additional experiments . . . 94
6.2.1 Exclusion of shank sensors . . . 94
6.2.2 Normalization of signals . . . 95
6.3 Conclusions and future work . . . 100

References **102**

A Appendix A: Replication of results of Kaketsis, 2020 **105**

B Appendix B: Confusion matrices **111**

1 Introduction

The world of sports has seen a continuous and rapid increase in the usage of technology in both competition and training during the last decades. Specifically in football (soccer), innovations such as the VAR (Video Assistant Referee) (introduced in 2018) and the Goal Line Technology (introduced in 2012) have revolutionized the game more than any other rule since the inclusion of the offside (the basis of the modern rule was introduced in 1925). The relevance of these two rules is that they brought technology closer to the game and this required the usage of high precision cameras and sensors. Not only technology has been used to modify or add certain rules into sports, but it has been used even more during training and for in-game analysis. It is nowadays common to see players training and even playing competitive matches wearing vests with GPS trackers below their shirts. These vests can track the players during the whole training or match and give information about their location, distance traveled, speed, power, intensity, heart rate, intensity, load, among others. These values can be processed to give the players, trainers and journalists a very detailed analysis of each player's performance. During the last one or two decades, the scientific community has showed important advancements in Human Activity Recognition (HAR) and, since technology and sports have developed a mutual beneficial relationship, there is a higher demand for systems capable of recognizing specific football activities.

If a coach, team or player has information about which activities the player does during a match or training, it is possible to perform a much more detailed analysis of the player's performance. A more complete assessment of the player's movements and loads gives the teams the possibility of a better training planning, a personalized follow-up to each player, and even a potential way to prevent, treat and understand injuries. Nowadays, this activity classification is done mainly with the aid of cameras. Either via human annotation or through artificial vision techniques it is possible to obtain such information. However, to do this, it is required to have a large amount of high-quality cameras equipped with artificial vision technologies or an army of human labellers. This is very expensive and can only be afforded by elite teams. Additionally, the usage of cameras brings another issue: during a match or a training a player can be behind another player, so the camera cannot recognize the player in question. These problems call for the usage of a low-cost activity recognition system that can be afforded by smaller teams. Wearable sensors are continuously being developed to be smaller and cheaper, so its usage is a clear solution to this problem. They can be incorporated into the player's sportswear and provide reliable, real-time measurements of different body parts.

This project studies the usage of deep learning-based models for football (soccer) activity recognition based on acceleration and velocity signals obtained from Inertial Measurement Unit sensors (IMU). This is done in contrast to traditional machine learning approaches, in which a non neural network based model such as kNN, decision tree or support vector machine is used to recognize an activity. Traditional methods require a manual process of feature extraction, while the deep models take this part into account by themselves. Furthermore, with this work it is intended to use the raw signals from the IMUs, and evaluate how robust are the deep models with respect to signals acquired on different players. This is why little or no preprocessing will be made to the sensor outputs, trying to recreate real-life scenarios. Different deep architectures will be proposed for the models and their performance

and evaluation time will be examined. Furthermore, a complete training and evaluation pipeline will be designed, in which also the preparation of the training dataset and the strategy for the evaluation phase via a sliding window approach will be taken into account.

The overall objective of this thesis is to design, implement and evaluate a complete pipeline for the training and evaluation of deep learning models for football activity classification based on raw IMU sensor's signals. The model should recognize accurately a selection of football activities in a quick and efficient way, so it can be potentially used in real-life scenarios. The recognized activities could then be used to study the movements of the players in order to do injury prevention and player follow-up.

The way this document is structured is as follows. An extensive literature review of the state of the art of Human Activity Recognition methodologies based on sensors is presented in chapter 2. In it, both traditional machine learning and deep learning approaches are shown with their comparison of results, best practices and challenges. A review of important conceptual elements is given in chapter 3. Chapter 4 presents the dataset that was used for training and validating the models. In that chapter, an initial feasibility analysis of a deep learning approach is performed, followed by the proposed activity detection algorithm, which is needed for the training phase. Chapter 5 is the main chapter of this thesis, in which the training of several deep learning models is thoroughly explained and discussed. Furthermore, the proposed evaluation pipeline is also presented, with which the activities present in a given recording can be effectively recognized. In chapter 6, the results are discussed, the proposed pipeline is evaluated on different datasets and additional experiments are performed. Finally, the conclusive remarks with future research recommendations are given.

2

Literature review

Identifying and recognizing human activities using signals obtained from Inertial Measurement Sensors is an area of machine learning and signal processing that has been studied by several authors recently. Recognition of daily activities, such as walking, climbing stairs, and sitting is especially popular amongst researchers due to the availability of public-domain datasets composed of these type of movements, and the possibility to easily compare the results with previous works (Slim et al., 2019). On the other hand, studies on recognition of sport-specific activities (such as football, tennis, ping pong or golf) are less frequent, since building these types of datasets is difficult because of the costs involved in resources and time. . However, since the nature of the signals is in many cases the same, it is possible to build upon the work of authors that have studied Human Daily Activity Recognition to build accurate and efficient methods designed for an application in a specific sport. In this case, football (soccer).

In this chapter, a review of different scientific researches about Human Activity Recognition based on wearable inertial sensor signals is presented. A big distinction can be made among the methods used to build such systems: methods based on traditional machine learning algorithms; and methods based on deep learning approaches. By traditional machine learning methods, we refer to approaches where a manual feature extraction process of the signals is needed prior to the classification of the activities, which is normally done with algorithms such as Support Vector Machines, Decision Trees or k-Nearest Neighbors, among others.

2.1. Traditional approaches to HAR

As explained before, this section will focus on what we consider traditional approaches to Human Activity Recognition. We define these approaches as methods in which the input features to the classification algorithms are manually defined and extracted. Such features are usually taken from the time domain (mean, variance, etc.) and/or from the frequency domain (coefficients of the Fourier Transform, energy, etc.). After these metrics are extracted, they are used as input features for traditional machine learning algorithms, like Support Vector Machines, Decision Trees, k-Nearest Neighbors, among others. It is important to note that these approaches require the input from the authors, in the sense that they define which features are extracted. Even though, as it can be seen in table 2.1, there is a common choice of certain metrics, each paper uses different number and types of features. This results in a highly subjective process in which the authors define, because of experience or simply gut, which features should be used as input. This does not mean, however, that the results of these approaches are bad. A big advantage of traditional machine learning algorithms is that, since the features are manually defined and extracted, it is easier to understand the inputs that the algorithm uses and it is possible to address the influence of one input on the prediction and try to explain it.

Several research papers have been written in which Human Activity Recognition is made with these approaches. A summary of some reviewed articles is presented in table 2.1. In all of them, a manually selected set of features is extracted from the signals, usually containing a combination of time and frequency domain metrics. In the vast majority of cases, since most of those metrics are scale-dependent and sensible to noise, the researchers tend to preprocess the data by filtering and normalizing the signals.

Although the the majority of reviewed literature focuses on the recognition of daily human activities, only Kaketsis, 2020 and Schuldhaus et al., 2015 focused their work on the recognition of football activities, such as passes, shots, jumps, running, etc. Both used time-dependent manually selected features, such as mean, standard deviation (variance), skewness and kurtosis, but the former also included additional metrics like median, minimum, maximum, and frequency-domain measures like the sum of the real parts of the coefficients of the Fourier Transform and the maximum of the real parts of the coefficients of the Fourier Transform.

Kaketsis, 2020 evaluated the performance of several classifiers (Naive Bayes, k-Nearest Neighbor, Support Vector Machines, Discriminant Analysis and Decision Trees) to classify between 4 or 7 football activities. He also evaluated the influence of including or not 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. He obtained the best accuracy of 92% in his dataset using the SVM approach. In appendix A, a deeper analysis of his results is presented.

On the other hand, Schuldhaus et al., 2015 developed a system capable of distinguishing between pass and shot. They first used a peak detection algorithm to detect possible passes or shoots and trained Support Vector Machines, Decision Trees and Naive Bayes classifiers to do the classification. They got the best results (88.6% accuracy on his dataset) when using linear SVM.

In table 2.1, it can be seen that a large variety of machine learning algorithms were used by different authors. It is not possible to conclude which algorithm is better from the table, since not all the papers used the same dataset, so the accuracies presented are not comparable among them. Some methods present high accuracies, while others achieve low or not very high performances. Apart from that, a huge variety of preprocessing techniques were used depending on the authors, the type of sensors, the activities to be recognized, the type of data and the machine learning algorithms used. Filtering the raw signals is the most common practice ((Liu, 2020), (Adaskevicius, 2014), (Blank et al., 2015), (Schuldhaus et al., 2015), (Kautz et al., 2017)). Other common preprocessing techniques are stroke detection when working with sports such as tennis ((Liu, 2020), (Connaghan et al., 2011)), table tennis (Blank et al., 2015) or volleyball (Kautz et al., 2017); usage of norms (or a variant of norms) of signals ((Kaketsis, 2020), (Adaskevicius, 2014), (Blank et al., 2015), (Schuldhaus et al., 2015), (Kautz et al., 2017)); and normalization or standardization of the signals to a certain range ((Liu, 2020), (Blank et al., 2015), (Kautz et al., 2017))

Table 2.1: Traditional approaches for Human Activity Recognition reviewed. The accuracies presented are the reported accuracies on the respective test dataset. Note that since each paper uses a different dataset, the accuracies must not be compared among them

Reference	Type of activities	Features	Classification algorithm	Accuracy
Kaketsis, 2020	Football	Mean, median, standard deviation, skewness, kurtosis, minimum, maximum, sum of real part of coefficients of Fourier Transform, maximum of real part of coefficients of Fourier Transform	Naive Bayes	89,0%
			K-Nearest Neighbor	86,0%
			SVM	92,0%
			Discriminant Analysis	86,0%
			Decision Trees	76,0%
Liu, 2020	Tennis	Mean, coviariance, skewness, kurtosis, minimum, maximum, magnitudes of Fast Fourier Transform, spectral energy	SVM	79,0%
Adaskevicius, 2014	Daily activities	Mean, standard deviation, maximum, minimum, frequency domain entropy, dominant frequency, average resultant acceleration	K-Nearest Neighbor	78,9%
Blank et al., 2015	Table tennis	Mean, standard deviation, skewness, kurtosis, minimum, maximum, energy, median, IQR, X-Y correlation, X-Z correlation, Y-Z correlation	Naive Bayes	87,1%
			Random Forest	95,7%
			Linear SVM	95,6%
			RBF SVM	96,7%
			K-Nearest Neighbor	94,7%
			CART	89,0%
Connaghan et al., 2011	Tennis	Not specified	Not specified	90,0%
Mannini and Sabatini, 2010	Daily activities	Correlation coefficients, DC component	Naive Bayes	97,4%
			GMM	92,2%
			Logistic Regression	94,0%
			Parzen	92,7%
			SVM	97,8%
			Nearest Means	98,5%
			K-Nearest Neighbor	98,3%
			C4.5	93,0%
Schuldhaus et al., 2015	Football	Mean, variance, skewness, kurtosis	Linear SVM	88,6%
			CART	86,1%
			Naive Bayes	87,1%
Wang and Liu, 2020	Daily activities	Standard deviation, kurtosis, skewness, root mean square, mean absolute deviation, maximum peaks of spectrum coefficients, Fast Fourier Transform	Decision Trees	86,3%
			Random Forest	90,8%

Table 2.1 continued from previous page

Reference	Type of activities	Features	Classification algorithm	Accuracy
De Vries et al., 2011	Daily activities	10th percentile, 25th percentile, 75th percentile, 90th percentile, absolute deviation, coefficient of variability, lag-1 autocorrelation	ANN	76,8%
Kautz et al., 2017	Volleyball	Median, mean, standard deviation, skewness, kurtosis, dominant frequency, amplitude of spectrum, maximum, minimum, position of maximum, position of minimum, energy, X-Y correlation, X-Z correlation, Y-Z correlation	RBF SVM	54,6%
			K-Nearest Neighbor	64,4%
			Naive Bayes	37,2%
			CART	58,7%
			Random Forest	67,1%
			Vote classifier based on the previous 5 classifiers	67,2%
Ignatov, 2018	Daily activities	40 statistical features	Random Forest	79,9%
		26 statistical features	Random Forest	79,9%
Ha et al., 2015	Daily activities	Mean, standard deviation and derivatives	K-Nearest Neighbor	83,9%-91,4%
Zebin et al., 2017	Daily activities	Mean, root mean square, autocorrelation coefficients, position of peaks, spectral peaks, amplitude	Decision Trees	87,9%-91,8%
			Linear Discriminant Analysis	80,2%
			Quadratic Discriminant Analysis	72,3%
			Linear SVM	91,8%
			Quadratic SVM	93,5%
			Cubic SVM	93,0%
			K-Nearest Neighbor	87,0%
			Ensemble	94,6%

2.2. Deep Learning approaches to HAR

The extremely fast-paced development in technology in the last decades has allowed the introduction of more complex machine learning architectures and algorithms. One of the most impressive breakthroughs in this sense was the implementation of bigger systems based on artificial neural networks and the introduction of variants of such networks, such as Convolutional Neural Networks and Recurrent Neural Networks. Nowadays we refer to these approaches as Deep Learning approaches, and are widely used in a vast amount of applications, such as artificial vision, natural language processing, signal processing, speech recognition, bioinformatics, and fraud detection, among others (Alom et al., 2018).

One of the most powerful and interesting capabilities of the deep learning approaches is their ability of feature extraction without a direct human input. Traditionally, machine learning algorithms such as Support Vector Machines or Random Forests require manually selected features extracted from the data as inputs. This process is not only heavily manual and subjective, but is also extremely time-consuming (Wang & Liu, 2020). This is one of the reasons why recent researchers in areas such as Human Activity Recognition have abandoned those traditional approaches in favor of deep learning architectures. Xia et al., 2020 explain this by saying that “researchers have turned to deep learning methods that could automatically extract appropriate features from raw sensor data during the training phase and present the low-level original temporal features with high-level abstract sequences”. This, however, does not mean that the researchers do not have to make decisions and use their domain-knowledge. Even if CNNs and RNNs have the capacity to automatically extract features from the raw data, several decisions have to been made by the scientists in terms of architecture, such as number of layers, number of neurons, configuration, type and size of convolution, type of RNN block, learning rate, optimization algorithm, loss function, among others.

Moreover, recent works have shown that the usage of deep learning approaches for Human Activity Recognition is not only beneficial in terms of feature extraction, but also because those approaches perform better. Put on Jiao et al., 2018 words, “It has been demonstrated that CNN-based models sufficiently show their dramatical superiority over the support vector machine (SVM) on behalf of the traditional methods”. Slim et al., 2019 do an extensive review of different approaches for Human Activity Recognition, and conclude that, on average, traditional machine learning algorithms obtain an accuracy of 83.3%, while systems based on deep learning achieve a much better 94.9%. They also express that there are more studies around traditional machine learning algorithms in comparison to deep learning ones, which shows that the usage of the latter in Human Activity Recognition tasks is recent and promising but not yet fully explored. This can be also seen in table 2.2 by checking that the references there presented tend to be from more recent years than the ones in table 2.1.

No relevant scientific article related to the usage of deep learning approaches with sensor data for football activity recognition was found. However, a large amount of works where these types of algorithms were used for Human Daily Activity Recognition were reviewed. Since the nature of the signals and the ultimate goal of those studies are very similar to the objective of this thesis, their methodology and results were considered. A summary of those reviewed articles is presented in table 2.2. The metrics and results in the table are not meant to be compared among them, since not all of those paper worked with the same dataset. Those numbers are shown for illustrative purposes of the high potential of these approaches. It is clear from the table that the usage of Convolutional Neural Networks is very popular, because “CNN-based models are able to extract and leverage latent feature representations in time series with high tolerance of time translation; thus, results outperform methods based on hand-crafted features” (Jiao et al., 2018). Many different deep architectures were reviewed: from CNN composed of consecutive convolutional layers to more complex and modern possibilities, such as inception CNN and residual CNN. However, “ (...) CNN lacks the capability to capture temporal dependency in time-series sensory data. RNN is designed to model time series data, and it is suitable for discovering relationships in temporal dimension” (Lv et al., 2019). This is the reason why the usage of Recurrent Neural Networks was also evaluated by other authors, mainly using LSTM units. A very interesting approach is the combination of CNNs and RNNs to build a larger and, according to the authors of such papers, better performing network. Examples of such architectures are the ones proposed by Ordóñez and Roggen, 2016, Xu et al., 2019, Xia et al., 2020, and Lv et al., 2019, where

usually a RNN (mainly composed of LSTM units) extracts temporal relations of the signals following a feature extraction process made by a CNN.

An additional very important takeaway from the reviewed literature related to deep learning approaches for HAR is the general lack of preprocessing phase of the sensor signals prior to their input into the deep networks. In the traditional approaches shown in section 2.1, it was explained that these works usually preprocessed the signals from the sensors with techniques like filtering, taking norms and standardization or normalization. When working with deep learning architectures not only the features are not manually selected, but the raw signals from the sensors can be directly used for the classification of the activities, avoiding the additional computation effort of preprocessing the signals. Although some authors used simple preprocessing techniques such as normalization or standardization, many of the reviewed articles worked directly with raw signals, understanding that the feature extraction process done by the convolutional and/or recurrent layers is robust enough to work without any signal preprocessing step.

Table 2.2: Deep learning approaches for Human Activity Recognition reviewed. The accuracies presented are the reported accuracies on the respective test dataset. Note that since each paper uses a different dataset, the accuracies must not be compared among them

Reference	**Type of activities**	**Features**	**Main type of layer**	**Accuracy**
Kautz et al., 2017	Volleyball	Raw data	CNN	83,2%
Ordóñez and Roggen, 2016	Daily activities	Raw data	CNN	0,883 (F1 score)
			CNN+LSTM	0,915 (F1 score)
Yang et al., 2015	Daily activities	Raw data	CNN	82,5%-87%
Ignatov, 2018	Daily activities	Raw data + statistical features	CNN	90,4%-94,4%
Ha and Choi, 2016	Daily activities	Raw data	CNN	91,33%-91,94%
Ha et al., 2015	Daily activities	Raw data	CNN	97,4%-98,3%
Zheng et al., 2014	Daily activities	Standardized raw data	CNN	90,3%-93,4%
Hsu et al., 2019	Multiple sports	Spectrograms of standardized filtered data	CNN	99,9%
Jiao et al., 2018	Golf	Standardized raw data	CNN	95,1-97,7%
			Inception CNN	96,0%
			Residual CNN	95,7%
Hammerla et al., 2016	Daily activities	Raw data	DNN	0,888 (F1 score)
			CNN	0,894 (F1 score)
			LSTM	0,912 (F1 score)

Table 2.2 continued from previous page

Reference	Type of activities	Features	Main type of layer	Accuracy
			bLSTM	0,927 (F1 score)
Xu et al., 2019	Daily activities	Normalized raw data	Inception CNN + GRU	0,946 (F1 score)
Xia et al., 2020	Daily activities	Raw data	LSTM + CNN	0,926 (F1 score)
Edel and Köppe, 2016	Daily activities	Normalized raw data	bLSTM	0.78-0,83 (F1 score)
Chen et al., 2016	Daily activities	Standardized raw data	LSTM	92,1%
Zhao et al., 2018	Daily activities	Standardized raw data	LSTM	0,882 (F1 score)
			bLSTM	0,892 (F1 score)
			Residual LSTM	0,902 (F1 score)
			Residual bLSTM	0,905 (F1 score)
Pienaar and Malekian, 2019	Daily activities	Raw data	LSTM	94,0%
Zebin et al., 2018	Daily activities	Raw data	LSTM	92,0%
Murad and Pyun, 2017	Daily activities	Raw data	LSTM	96,7%-97,8%
			bLSTM	92,5%
			Cascaded LSTM	92,6%-94,1%
Hernández et al., 2019	Daily activities	Raw data	bLSTM	92,7%
Lv et al., 2019	Daily activities	Raw data	CNN + LSTM	Around 85%
Wang and Liu, 2020	Daily activities	Filtered raw data	LSTM	91,7%

Conclusion

During the last years, many authors have studied the problem of Human Activity Recognition using different techniques. Initially, a traditional approach to machine learning was used, in which algorithms such as Decision Trees, Support Vector Machines and K-Nearest Neighbors were used to classify the activities after a process of manual selection of features extracted from the signals. Those manually selected features were usually taken from the time and frequency domain, being common the choice of metrics such as mean, standard deviation, maximum, minimum, kurtosis, and coefficients of the Fast Fourier Transform. However, “the traditional feature engineering methods are becoming more and more incapable” (Xu et al., 2019). With the availability of better and faster computing capabilities and algorithms, the research in this area has shifted towards a Deep Learning approach, where deep architectures such as Neural Networks, Convolutional Networks and LSTMs are now in charge of the classification. Not only these algorithms perform, in the majority of cases, better than the previous attempts, but a huge advantage of them is their ability to automatically extract relevant features. This capability is of enormous importance, not only because it achieves better results in terms of classification accuracy, but also because it avoids the need of slow and subjective manual selection of features from the signals allowing the use of raw signals. Put on Xu et al., 2019 words, “deep learning makes it more convenient to extract and classify complex data in the face of a large number of different sensor sources”. It is important to note that, at the present moment, a very low quantity of works related to Football Activity Recognition were found. The majority of reviewed papers focused on Daily Human Activity Recognition and some specific applications on stroke-based sports, like tennis, golf or volleyball. The implementation of methods applied to HAR for football-related activity has not been thoroughly discussed in the literature and is the focus of this thesis.

3

Conceptual framework

3.1. IMUs

IMUs (Intertial Measurment Units) are small, often wireless electronic devices embedded with a variety of sensors of different types. For Human Activity Recognition tasks, they are often placed on different body parts (specially limbs) of a person to capture the movements that he or she is 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. In the specific case of this thesis, IMUs are placed on 5 body parts of football players (right shank, left shank, right thigh, left thigh, an pelvis), and the measurements of the accelerometers and gyroscopes are used to recognize the activities the players do.

When explaining the dataset used for a part of the work, the type of IMU and sensors used to acquire the signals will be detailed. Nevertheless, in all the cases, unless stated the contrary, the locations of the IMUs will be the same as the ones mentioned above. The inclusion of IMUs in sport applications is very beneficial when building Human Activity Recognition applications, since they are very small and can be comfortably integrated in sport equipment without disturbing the execution of the movements. For illustrative purposes of their small size, figure 3.1 shows one of the versions of the the IMUs used for this thesis and the pants where they were integrated.

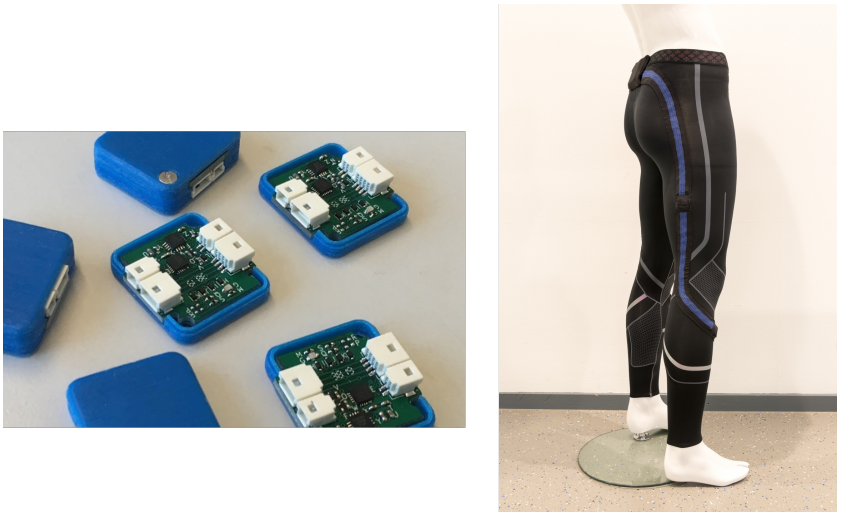


Figure 3.1: IMUs and pants

3.2. Traditional Machine Learning Algorithms

In this thesis, traditional machine learning techniques are referred to those machine learning algorithms that are not based on neural networks. Since both approaches are compared in this project, a summarized conceptual description of common traditional machine learning techniques is presented below. In this case, all of the algorithms here presented correspond to supervised learning algorithms for classification tasks, which means algorithms that are trained to classify input data into one of several predefined classes or categories.

Among the most common traditional machine learning algorithms, are the following:

- K-Nearest Neighbors (kNN)
  The kNN algorithm assumes that similar things occur next to each other. When this method sees a new, unseen sample, it is assigned to the most common class of the closest K samples of the training data to that new sample. The definition of closeness must be defined, but a typical choice is by the euclidean distance. The user must also define the value for K.

- Naive Bayes
  Naive Bayes is a classifier based on the probability distributions of each class. It assumes that the features of the input data are statistically independent and assigns a class to a new, unseen sample based on the probability of it being part of one class or another according to the Bayes' Theorem. This prediction, then , depends on the probabilistic distributions of the training data points. The feature independence assumption is very strong and is not always satisfied, which is why this method is called naive

- Discriminant Analysis (DA)
  Discriminant Analysis methods assume that the observations that are part of each one of the classes to be recognized are sampled from a normal distribution. DA estimates the parameters of a normal distribution for each class based on the training samples. By doing this, it is possible to define the boundaries between the distributions of these classes and, then, use those limits to perform classifications. When working with DA methods, it is possible to differentiate Linear Discriminant Analysis (LDA) from Quadratic Discriminant Analysis (QDA). The major difference between them is that the former assumes that the covariance matrices of all the classes are equal, resulting in a linear decision boundary. The latter, on the other hand, does not do this assumption, leading to quadratic decision boundaries.

- Decision tree
  A decision tree classifier is built by repeatedly splitting the input space based on values of its features. At each step, this algorithm finds the best variable that can be used to split the data in two and performs such split based on the value over that variable that maximizes the separation of classes. This procedure is repeated in a greedy matter, so that at the end of the process the feature space is divided in several areas that correspond to different classes. Decision trees can easily overfit to the training data so, to avoid this problem, usually the tree is pruned to a certain level to reduce the number of splits.

- Random forest
  Random forest are a type of ensemble classifiers in the sense that they are built by combining several simpler classifiers. In this case, those simple classifiers are decision trees. To build a random forest, a number of independent decision trees are trained using different subsets of the training data (this is called bagging), so that ideally no two decision trees are equal. Then, the predictions of all of those decision trees are taken into account (via averaging or majority voting) to obtain the final prediction of the random forest. This type of algorithm is very popular and tends to give very good results, because by combining the predictions of many simple classifiers, individual errors made by them are cancelled and corrected.

- Support Vector Machine (SVM)
  Support vector machines are built under the premise that they find the hyperplane that maximizes the separation margin between two classes. Because of this, SVMs are mainly used for binary problems (where only two classes are meant to be distinguished). In theory, SVM are linear

classifiers in the sense that they build linear hyperplanes, but they can be also used for nonlinear problems by projecting the data points to another space where they are easier to separate linearly. This is called the kernel trick in SVMs, which uses the so-called kernel functions to do these projections. Common kernel choices are the linear ($k(x, x') = \langle x, x' \rangle$), polynomial ($k(x, x') = \langle x, x' \rangle^d$), gaussian or RBF ($k(x, x') = \exp\left(\frac{-|x-x'|^2}{2\sigma^2}\right)$), and sigmoid (($k(x, x') = \tanh(\kappa\langle x, x' \rangle + \vartheta)$) kernels. Although SVMs are originally designed to perform binary classification, there are a number of approaches that can be used when working with multiclass classification problems:

– One vs One (OvO): The OvO approach creates a binary classifier for each pair of classes. This means that if there are N classes, the OvO SVM method creates $\frac{N(N-1)}{2}$ binary models. Then, when evaluating a new observation, all the $\frac{N(N-1)}{2}$ binary classifiers are applied and the most frequently predicted class is assigned.

– One vs Rest (OvR) or One vs All (OvA): The OvA creates N binary classifiers for a problem with N classes. Each one of those classifiers is built to distinguish between one of the classes and one of the $N-1$ others. When evaluating a new observation, all the N binary classifiers are applied and the class with the largest prediction confidence is assigned.

– Error Correcting Output Codes (ECOC): When building ECOC models, an m-bit representation of each one of the N classes is defined in advance. Then, a binary classifier is trained for each one of the bits, so m binary classifiers are built. When evaluating a new observation, the m models are applied, resulting in the predicted m-bit representation of the new data point. Then, that representation is compared with the m-bit representations of all the N classes and the one with smallest distance is chosen.

3.3. Deep learning

Unlike traditional machine learning algorithms, deep learning refers to the use of models built based on stacked layers of artificial neural networks. By using multiple layers, it is possible to train the model to progressively extract and learn complex features from the inputs. This has a huge advantage over traditional machine learning algorithms, since deep models perform internally both feature extraction and classification/regression/clustering. Deep learning models can be used for both supervised and unsupervised learning tasks, but in this thesis the former type is used. 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. In its simplest form, an artificial neuron looks like the one in figure 3.2. This means that a neuron is internally computing the equation

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

where σ is called the activation function and is used to introduce non-linearities. Among the most common activation functions are the ReLU, Sigmoid and Tanh, among others. 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 values for the weights w.

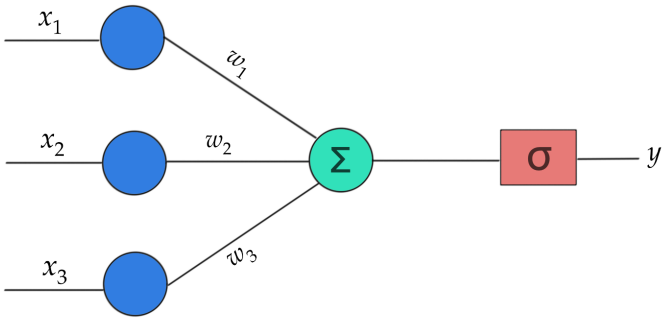


Figure 3.2: Simple artificial neuron model. Taken from (Sharma, 2020)

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, as exemplified in figure 3.3. 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.

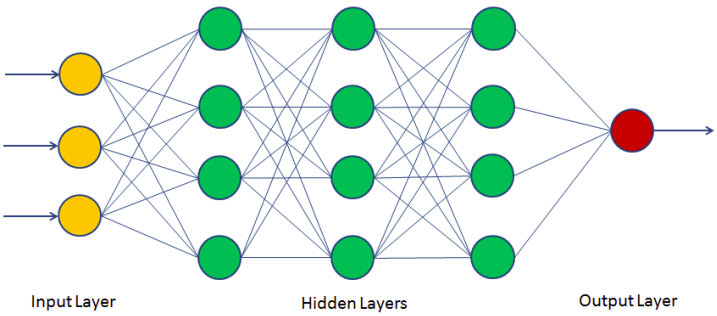


Figure 3.3: Artificial Neural Network composed by several fully connected layers. Taken from (Navlani, 2019)

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 (specifically LSTMs).

3.3.1. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) 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 showed that the neurons responsible for those tasks have receptive fields. This means that each cell responds to a certain specific pattern. The combination of these simple patterns generates more complex ones that are furthermore combined, so that the brain can finally interpret and understand the image. Convolutional neural networks try to replicate this behavior and have showed impressive results in a huge variety of machine learning applications. Although they were originally designed for recognizing shapes an figures in images, as explained, they can also be used on other types of data, such as time series. Instead of recognizing shapes and figures in images, CNNs can also be used to extract patterns from signals.

The general idea of CNNs is based on two types of operations:

- Convolutions
  In a convolutional layer, the input data is convolved with a certain number of kernels or filters. A filter k is traversed through all the points of the input image (or signal) and on each location the convolution between the filter and the overlapping area of the data is calculated (figure 3.4). These convolutions made by the chosen filter across all the input data result in a new convolved image (or signal) called a feature map. Many filters are chosen, each one of them with different weights, so that the convolution operation translates the data into a set of feature maps (one per filter). Since every kernel generates its own feature map, each one of them specializes in recognizing and extracting a specific pattern in the input data. Then, all of the feature maps are concatenated to form the output convolved image (or signal) of the convolutional layer. Training a CNN means allowing the model to learn the weights for each filter.

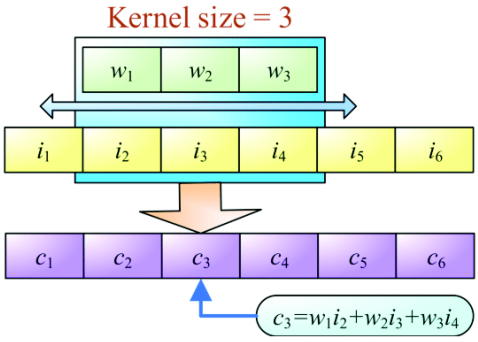


Figure 3.4: Convolution operation on 1D data. Taken from (Kuo & Huang, 2018)

- Pooling
  It is a common practice to include pooling layers after convolutional ones. Pooling layers introduce the concept of receptive fields into CNNs, because they condense information of neighboring points of the convolved data into a single value. Many types of pooling layers can be used, but one of the most common is the max pooling layer. This operation defines a small block (also called a filter) and runs it on top of the input data of the layer. At each point, the maximum of the values contained on the overlapping area of the data and the filter is extracted and the output image (or signal) is built with those maximum values. Pooling layers have the additional property that they reduce the size of the feature maps, which translates into fewer weights to be learned by the model. The outputs of these layers are smaller features maps, but each element of those maps has information about their neighbors from the previous layer.

CNNs are often combined with additional layers, such as fully connected, as it can be seen in figure 3.5. Note that it shows a CNN operating on images, since it is easier to visualize. However, this thesis applies CNNs on one-dimensional data (signals), like in figure 3.6.

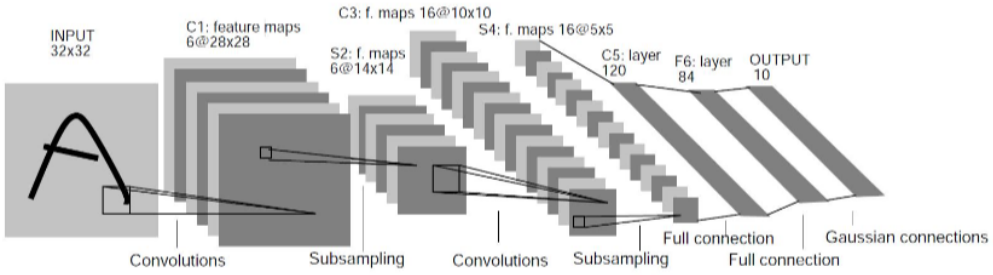


Figure 3.5: Convolutional Neural Network example. Convolutional and pooling layers one after the other followed by dense layers. Taken from (LeCun et al., 1998)

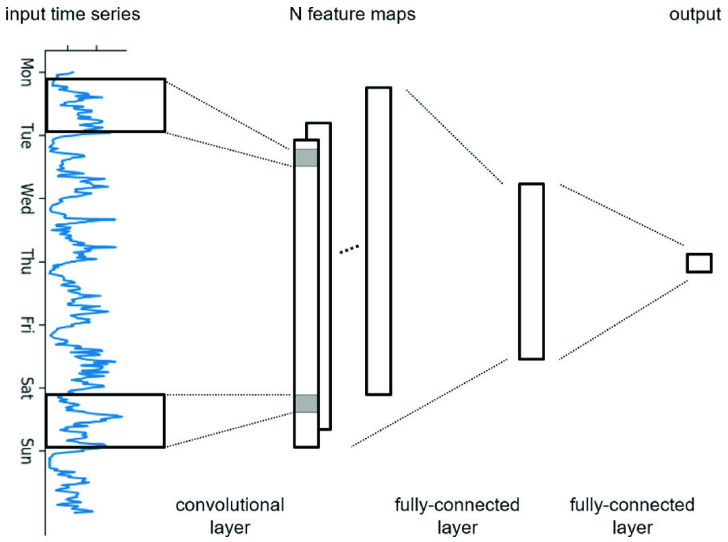


Figure 3.6: Convolutional Neural Network applied to 1D signal data. Taken from (Lang et al., 2019)

3.3.2. 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. Based on this idea, RNNs have the ability to "remember" prior inputs when generating the output. Take for example a machine that translates text from one language to another. A translation mechanism that just translates word by work will give poor results. A sentence has an underlying temporal structure based on words: the order in which they appear affects the meaning of the sentence, thus it is important to know which words came before a given word. Similar to sentences, time series and signal measurements also carry critical information on the temporal sequence of their values, so a neural network that can remember past information is valuable in these cases. In other words, RNNs produce outputs not only based on the current input vectors, but also on what are called the hidden state vectors that carry information about prior data. Figure 3.7 shows the basic form of a RNN.

As mentioned, RNNs are used for data that comes in sequence, since they are able to relate previous information to the present. This, although is theoretically possible with the architecture shown in figure 3.7, has a major drawback when implemented due to a problem called "vanishing gradients". Basic RNNs are unable to remember long-term dependencies, because the gradients that are used to train the model tend to disappear as the input sequence grows in length. When training deep learning

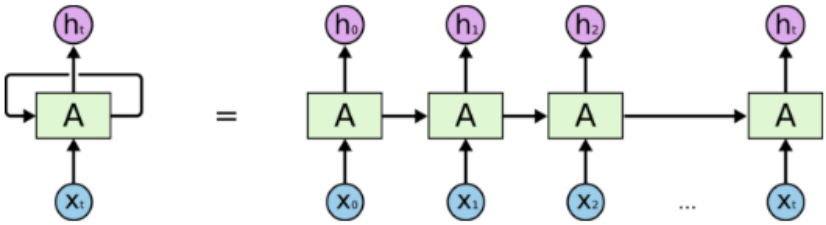


Figure 3.7: Basic form of a Recurrent Neural Network. The figure on the left shows the recurrent property with which an internal state vector of step t is used at step $t + 1$. For illustrative reasons, the figure on the left is often presented as the equivalent unrolled version on the right. Taken from (Olah, 2015)

models, small gradients are undesirable as the training takes longer and as a result becomes less effective. The RNN cells as presented in figure 3.7 can only remember information that happened in the near past.

When working with sequence data such as signals porduced by a set of sensors, 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 common RNN networks that are used to overcome that problem. 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 3.8 where x_t represents 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.

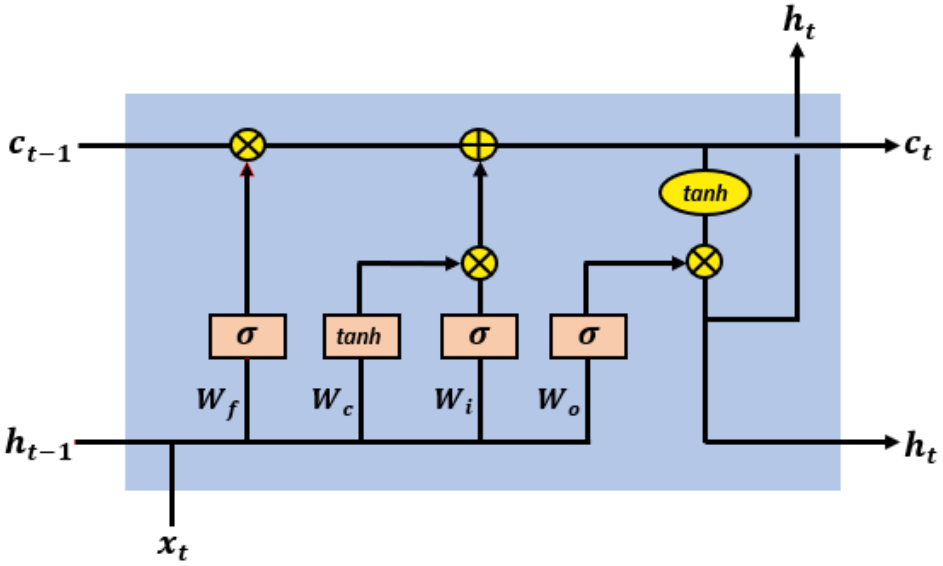


Figure 3.8: Internal structure of an LSTM cell. Taken from (Arbel, 2018)

In order to understand how LSTMs work, it is easier to analyze them by parts. One of the most important properties of LSTMs is their capacity to easily propagate information from one cell to another. This is implemented by the cell state c, which, as can be seen in the top horizontal line of figure 3.8, can run through the cell with only minor linear modifications. The cell state is the heart of the LSTM and is what carries past information of the input sequence. LSTMs can add or remove information from the cell state by using structures called gates (which are themselves regular feed-forward neural networks). There are three of them:

- Forget gate
  This gate is responsible for deciding what information must be forgotten (removed) from the cell

state. To do so, it concatenates the hidden state at time $t-1$ (h_{t-1}) and the current input x_t and calculates a value between 0 (forget) and 1 (keep) for each element of the cell state c_{t-1}. Its mathematical implementation is given by

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

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

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 0 (or 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:
  $$\tilde{c}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c),$$
  Where W_c are the candidate weights that are learned during training and b_c the bias term. 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:
  $$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i),$$
  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 the LSTM unit. To do so, it first calculates the values o_t as

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

  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:

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

3.3.3. Additional Deep Learning Concepts

Some additional deep learning concepts were used in this thesis and deserve a special mention and understanding.

Dropout Layer

A dropout layer is a special type of layer often used in deep architectures to reduce overfitting. This layer ignores randomly selected units (neurons) of the previous layer. At each training step, each unit of the layer is momentarily eliminated (with all the connections between them and other neurons) with a probability of p. While this might sound counter intuitive (at the end the goal is to train the model to find the best values of all the connections in the network), turning off a small parentage of neurons at each training phase helps considerably to reduce overfitting. This is because, if there are many connections

to be learned, networks tend to rely on certain specific neurons on connections more than on others (a neuron can be co-adapted with a previous one, so it corrects its output). By turning off a random subset of neurons during the training phase, the model cannot depend on any particular neuron to obtain a correct classification, thus reducing co-dependency between units.

Loss function

When building an artificial neural network, apart from choosing the architecture and parameters of each one of the layers, it is also needed to choose a loss function. This loss function is of major importance, because it is used to train the model via an optimization algorithm. It ultimately defines how close are the model's outputs from the desired values. The difference between those two values is quantified by the loss function to obtain a loss (or error) of the predictions. These errors are backpropagated through the network and used by the optimization algorithm to update the weights of the model so that a smaller loss is (ideally) obtained.

Depending on the task that the model is required to do, different loss functions can be selected. For the particular interest of this thesis, since HAR corresponds to a multiclass classification problem and the classes were one-hot encoded, 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. Since the ground truths are one-hot encoded, 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 used in this thesis, the calculation of its partial derivative with respect to an output $\hat{y}_l$ is as follows:

$$\begin{aligned}\frac{\partial J}{\partial \hat{y}_l} &= \frac{\partial}{\partial \hat{y}_l}\left(-\log\left(\frac{e^{\hat{y}_i}}{\sum_{j=1}^{N} e^{\hat{y}_j}}\right)\right)\\ &= -\frac{\partial}{\partial \hat{y}_l}\hat{y}_i + \frac{\partial}{\partial \hat{y}_l}\log\sum_{j=1}^{N} e^{\hat{y}_j}\end{aligned}$$

The first term of the derivative is

$$\frac{\partial}{\partial \hat{y}_l}\hat{y}_i = \begin{cases} 1 & \text{if } i = l \\ 0 & \text{otherwise} \end{cases} = \mathbb{1}(i=l)$$

For the second part,

$$\begin{aligned}\frac{\partial}{\partial \hat{y}_l}\log\sum_{j=1}^{N} e^{\hat{y}_j} &= \frac{1}{\sum_{j=1}^{N} e^{\hat{y}_j}}\frac{\partial}{\partial \hat{y}_l}\sum_{j=1}^{N} e^{\hat{y}_j}\\ &= \frac{e^{\hat{y}_l}}{\sum_{j=1}^{N} e^{\hat{y}_j}}\end{aligned}$$

Then,

$$\frac{\partial J}{\partial \hat{y}_l} = -\mathbb{1}(i = l) + \frac{e^{\hat{y}_l}}{\sum_{j=1}^{N} e^{\hat{y}_j}}$$
$$= -\mathbb{1}(i = l) + f(\hat{y})_l$$

Having this expression of the partial derivative of the loss function with respect to each one of the outputs $\hat{y}_i$ of the network, it is then possible to obtain the partial derivative of the loss function with respect to any particular weight/parameter w by using the chain rule of differentiation:

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

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/parameter. 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, then, depend on the activation function σ used by each neuron and by their inputs x (which can be themselves outputs from previous neurons).

ADAM optimization algorithm

As explained, training an artificial neural network ultimately means repeatedly finding values for the weights of the network so that the loss function is minimized. Since a model has a large amount of weights to be optimized, it is required to know how much each one of them must be tweaked and in what direction (increase it or decrease it). 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. In other words, the gradient of the loss function is needed. With these partial derivatives, the weights w are updated as follows:

$$w := w - \alpha \frac{\partial J}{\partial w}$$

Where α is called the learning rate. This update equation is called the gradient descent algorithm and, even if it is very simple, can be used to effectively train 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 respective weight (the gradient points to the direction of greatest increase, so we take the opposite direction). The amount of this update is determined by the step function α.

Gradient descent can be successfully used to train neural networks, but it can converge slowly. To improve the training speed of neural networks (which is of huge importance in deep learning), additional algorithms have been proposed that imply improvements on the basic functionality of the gradient descent. One of the most widely used algorithms for this purpose is called the ADAM (Adaptative Moment Estimation) optimization algorithm. Instead of using fixed steps, it combines two elements to converge faster: momentum and adaptative 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 ridges, several local minima, and saddle points. To address this, it is possible to understand instead the optimization

algorithm as a ball rolling down the loss function. This ball will have some inertia and will be able to take larger steps when the ball follows a somehow regular path. This additionally dampens the learning and makes it less oscillating. In other words, we would like an optimization algorithm that moves fast and accelerates if we move in the same direction, but slows down if we have to turn to another direction. Parameters whose gradients change directions should be slowly updated, but those who point in the same (or similar) directions should be modified with larger steps. Momentum is referred to the capacity of the algorithm to accelerate the learning step by using an exponentially weighted average of the past gradients:

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

Where β_1 is a hyperparameter between 0 and 1.

The momentum term at time t, as it can be seen, takes a weighted sum of the previous momentum term $t - 1$ and the gradient at time t. This is called an exponentially weighted average.

Then, a gradient descent step includes the momentum term m_t by doing

$$w_{t+1} = w_t - \alpha m_t$$

- Adaptative Learning Rates

The adaptative learning rate property is extracted from another optimization algorithm called RMSprop. Similarly to momentum, RMSprop starts with an exponentially weighted average, but in this case of the squared gradients. This is also called the second moment:

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

Where β_2 is a hyperparameter between 0 and 1.

By taking the square of the gradients, we always add up instead of potentially cancelling out positive and negative values. 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 on determining the learning rate for the step to be taken. Again, we move in the direction opposite of the gradient, but the step size is determined by the second moment previously calculated. A predefined learning rate α is divided by the square root of that second moment, so α is modified for each one of the parameters independently. A parameter with a large exponential average (because in that parameter's direction the function has steeper slopes) will then be updated with small steps, while a parameter with small second momentum will have a larger learning rate. This is why it is said that RMSprop uses a different learning rate for every weight w_i. Those learning rates are continuously being modified based on the gradients that have been previously computed for each weight. Mathematically, it is defined as:

$$w_{t+1} = w_t - \alpha\frac{1}{\sqrt{v_t} + \varepsilon}\frac{\partial J}{\partial w_t}$$

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 RMSprop. 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:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

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

Since these two moments are initialized as zeros, they tend to be biased towards zero, especially during the initial steps. To avoid that, those moments are bias-corrected:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Where t is the time step or iteration number.

Finally, it does the parameter update by combining both bias-corrected moments with their respective properties (momentum for $\hat{m}$ and adaptative learning rate for $\hat{v}$):

$$w_{t+1} = w_t - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon}$$

The full ADAM algorithm is presented in figure 3.9. In that figure (which was taken from the original paper), the loss function is represented by f and the weights/parameters by θ. The commonly chosen values for the hyperparameters are $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\varepsilon = 10^{-8}$. The learning rate is still denoted by α and it should be manually selected.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_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 θ_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 θ_t (Resulting parameters)

Figure 3.9: Adam optimization algorithm. Taken from (Kingma & Ba, 2014)

4

Data overview and preparation

4.1. Data

In every machine learning application, one of the most important elements is the data used as input information for the models. This project used datasets designed and acquired by different authors that worked on similar problems before. In particular, a dataset taken from the work made by Wilmes, 2019 was used for the training phase and for an initial evaluation. A brief explanation of this dataset will be given below.

Wilmes, 2019's data

This dataset was mainly used to train the models and do some evaluation of the results because the activities were already clearly extracted and labeled.

The experiments conducted by Wilmes, 2019 included 11 male soccer players with 5 IMUs (Ivensense MPU-9150) attached to their bodies in the following locations: pelvis, right thigh, left thigh, right shank and left shank, as it can be seen in figure 4.1. Each one of the IMUs had a tri-axial accelerometer, gyroscope and magnetometer. The range for the accelerometers was set to $\pm 16g$ and for the gyroscopes to $\pm 2000°/s$ (the magnetometers' range was not specified). The sample frequency of the signals was set to 500 Hz. Each one of the participants executed a number of football-related activities, including passes, shots, jumps, sprints, among others. The experiments were designed in orer to simulate an actual football match in order to have reliable and real movements.

Figure 4.1: Location of IMUs in experiments (Wilmes, 2019)

The usage of those 5 IMUs allowed the measurement of the tri-axial accelerations and velocities of the 5 respective body parts related to each one of the activities performed by the subjects. This resulted

in several signals representing the movements, each one of them with their manually annotated category (activity). It is important to note that in each experiment, before and after the subject performed one of the activities, they had a period of walking or standing. This resulted in recordings where, before and after the desired activity, some measurements of standing or walking were recorded. These intervals had to be excluded from the analysis. This procedure, called activity detection, will be explained in more detail in section 4.3. More information about the experimental protocol and synchronization of the signals can be found in the original document (Wilmes, 2019).

Examples of signals for four different activities are presented in figure 4.2. Since this figure is just for illustrative reasons, only the accelerometer signals are shown and not the gyroscope or magnetometer. Each color of in the plots represents a different axis (X, Y or Z).

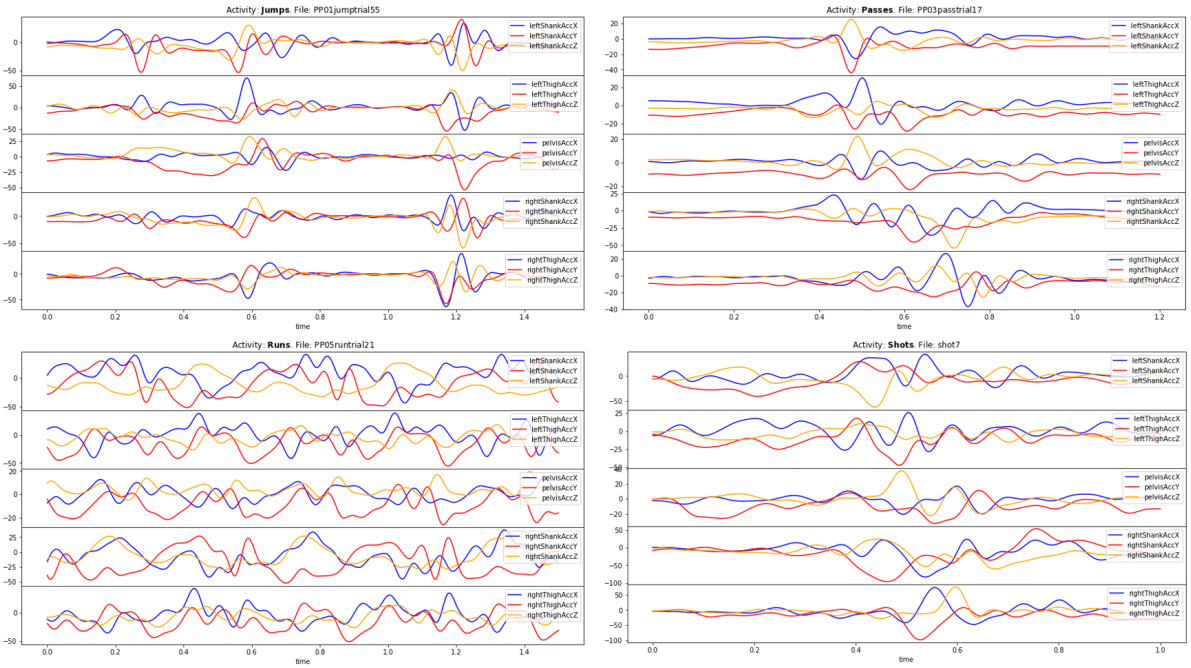


Figure 4.2: Examples of activities from dataset from Wilmes, 2019. Only accelerometer data is presented. From left to right, from top to bottom: jump, pass, run and shot.

Previously, it was explained that the recordings contained some miliseconds before and after the actual activities where pre- and post- activities such as walking or standing happened. However, these intervals were not standardized so each recording had different lengths of pre- and post- activities and, as can be seen in figure 4.3, the actual activities did not always happen at the same moment. This situation resulted in the impossibility to obtain a clear pattern of the average recording of each of the activities: the recordings for each activity were not centered, so the signals cancelled themselves. This can be seen in figure 4.4, where the mean signals do not show a clear structure representing the typical pattern of the activities (compare these mean signals with the single instances of figure 4.2, for example).

Because of this reason, since this dataset was used to train the models, it was first needed to eliminate the pre- and post- activities from the recordings, so that the desired activities were isolated. This procedure is explained in detail in section 4.3.

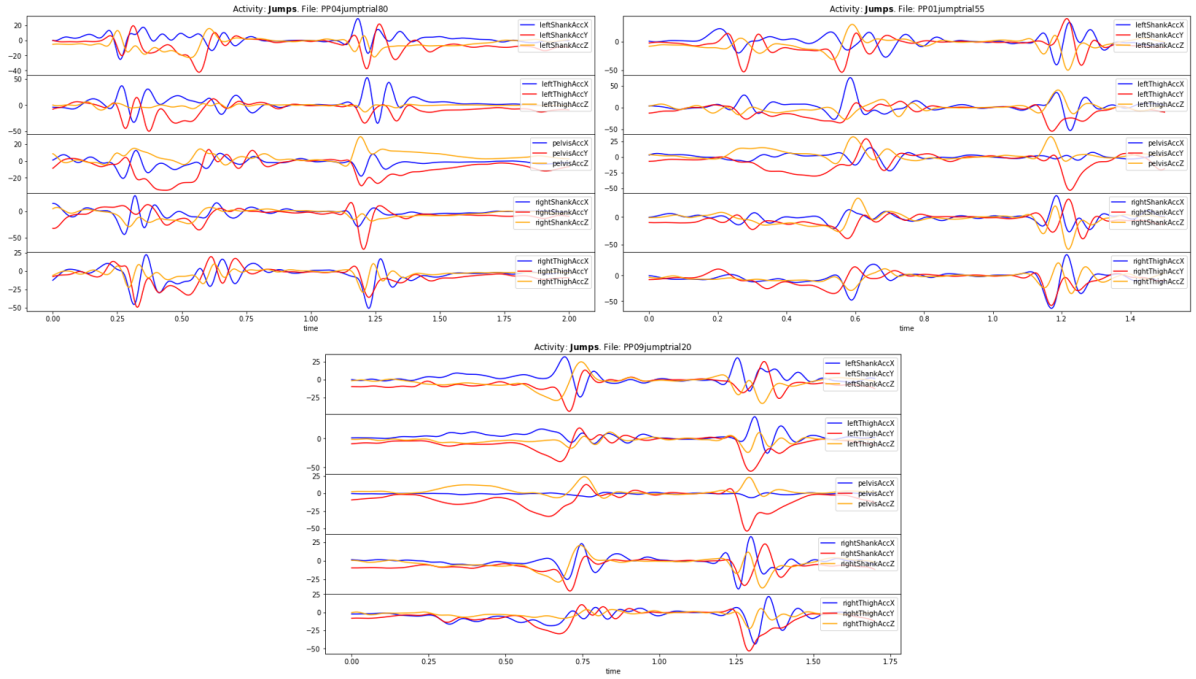


Figure 4.3: 3 examples of jumps from dataset by Wilmes, 2019. Because of the pre- and post- activites, the actual jumps happen at different moments in each recording. Only accelerometer data is shown

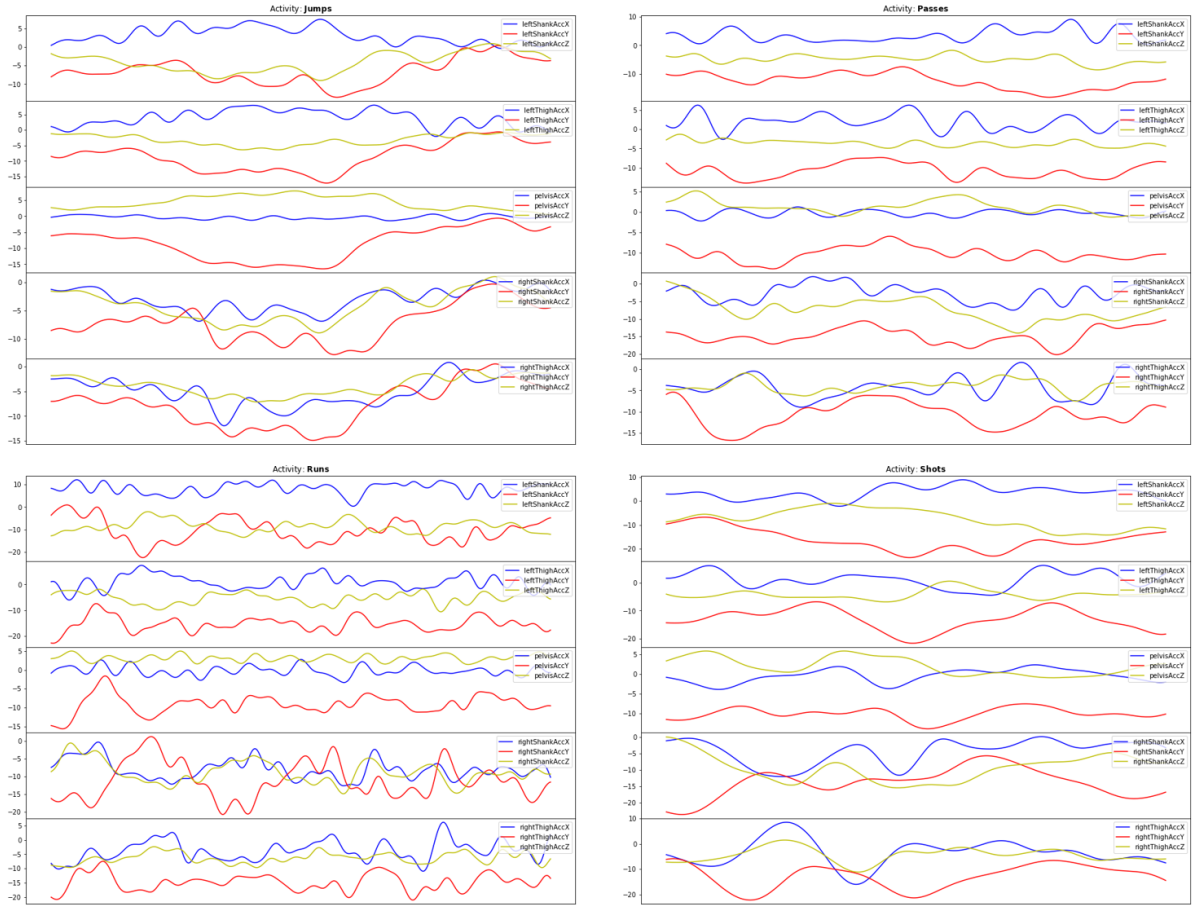


Figure 4.4: Mean signals from dataset by Wilmes, 2019 for selected activities. From left to right, from top to bottom: jump, pass, run and shot.

4.2. Feasibility analysis of a deep learning approach

As it was seen in the literature review, either a traditional approach to machine learning or a deep learning architecture can be used to recognize the football activities in the datasets. In chapter 2 it was concluded that traditional approaches not only tend to perform worse, but they also heavily rely on the manual selection of features of the signals, which is not ideal (in terms of performance, time and information retrieval). In order to better explain this limitation of traditional methods, the following synthetic examples were prepared.

The problem with time-related manually extracted features is that they do not capture patterns and the specific structures present in the signals. They are, in general terms, a mere summary of how the signal is as a whole. Take for example the three signals shown in figure 4.5. They are three simple signals with clearly different structure and patterns, so after a feature extraction process, we would expect to have different features, so that each signal could be distinguished from the others.

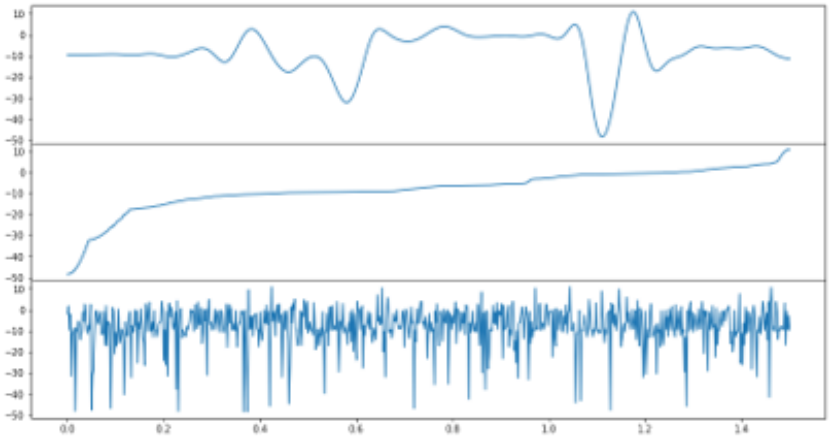


Figure 4.5: Three different signals: signal A (top), signal B (middle), signal C (bottom)

A common set of time-domain manually extracted features is composed of the mean, standard deviation, minimum, maximum, skewness, and kurtosis of the signal. In table 4.1 those metrics for the signals of figure 4.5 are presented. As it can be seen, all the features have exactly the same values for the three signals, which means that it would be completely impossible to distinguish between the three signals using only the selected time-domain features. A big part of the information of the signals is not captured with this way.

Table 4.1: Time-domain manually selected features for the signals of figure 4.5

Feature	Signal A (top)	Signal B (middle)	Signal C (bottom)
Mean	-8,01	-8,01	-8,01
Standard deviation	9,68	9,68	9,68
Minimum	-48,49	-48,49	-48,49
Maximum	10,86	10,86	10,86
Skewness	-1,67	-1,67	-1,67
Kurtosis	4,12	4,12	4,12

Time-domain features are not the only type of metrics used to describe the signals when working with traditional machine learning algorithms. The Fourier Transform of the signals allows the extraction of frequency-domain features, which enrich the previously described time-domain ones. However, these frequency-domain features can also fail to capture important patterns of the signals. In figure 4.6 two signals are presented on the top and their frequency spectrum (via de FFT) on the bottom. It can

be observed that the spectrums of both signals are the same, but the two signals are clearly different.

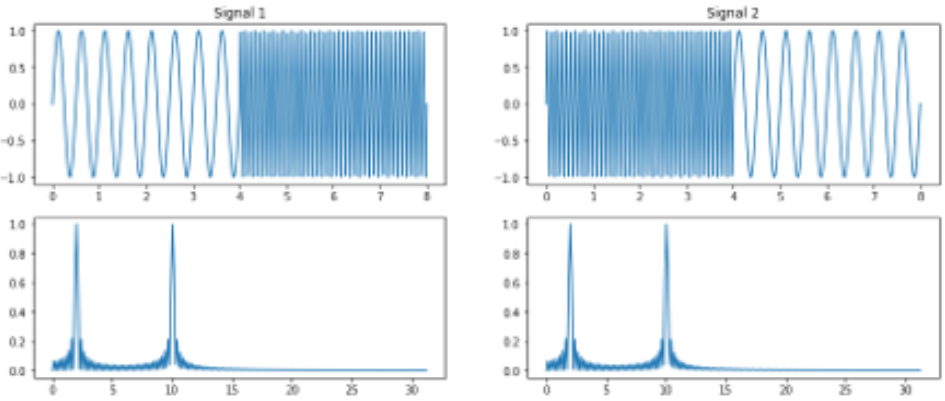


Figure 4.6: Two different signals (top) with their respective frequency spectrums (bottom)

The previous two examples are simple scenarios used to exemplify a big disadvantage of manually selected features: a big amount of information of the signals can be lost in this process. On the other side, note that the time and frequency domain features used in these examples were manually chosen to be those. Another person could decide to use a different set of features that could help to distinguish better the signals. This is another problem of the usage of manually selected features: they are inherently dependent on the person building the system.

If those features are not descriptive enough, then how is it possible that several authors achieved good results in HAR when using manually selected features (see chapter 2)? The previous examples used extreme-case signals to exemplify the problem of information loss when using time- and frequency-domain manually selected features. In real HAR applications the signals are much more complex that the ones used in the examples, there are several channels and signals, and a large combination of time- and frequency-domain features are extracted from each one one the signals. This process results in a large amount of time and frequency-domain features that represent all the sensors capturing the movement. This high number of metrics fed into a traditional machine learning algorithm allows HAR with acceptable or good results.

The need to use a large amount of manually selected features to describe the signals has another problem: the time required to compute each one of the features. Wang and Liu, 2020 identified this situation and concluded that “the traditional classification model spends much time on extracting feature vectors, leading to potential failure in the data preprocessing stage”. In order to check this claim and have an initial estimate of how fast is a deep learning approach to HAR with respect to traditional methods, the following experiment was conducted.

Using the exact same data that Kaketsis, 2020 used in his work, several HAR models based on traditional methods and on deep learning architectures were built. For the traditional methods, the features were manually selected to be the ones that Kaketsis, 2020 used: mean, median, standard deviation, skewness, kurtosis, maximum, minimum, sum of real coefficients of FFT, and maximum of real coefficients of FFT. For the deep learning approaches, the raw signals were used.

The following algorithms were built using the traditional approach:

- K-Nearest Neighbors.
- Naïve Bayes.
- LDA.
- QDA.

- Decision Tree.
- Random Forest.
- Linear SVM.
- RBF SVM.

The following deep learning architectures were used:

- LSTM(8) + LSTM(8).
- LSTM(32) + LSTM(32) + FC(16).
- CNN1D(64,3) + MaxPool + LSTM(32) + LSTM(32) + FC(16).

Where LSTM(n) means a LSTM layer with n units, FC(n) a fully connected layer with n units and CNN1D(k,n) a 1D-CNN layer with n filters of size k.

This experiment was not focused on achieving a good classification accuracy, but on having an initial estimate of how fast a deep learning approach is with respect to traditional approaches for the prediction phase (not the training phase, since the training only has to be done once). Because of that, no hyperparameter tuning was made for any model. All the models were trained with the same set of samples. The plots on figure 4.7 show the seconds needed to classify the same 280 samples using each model.

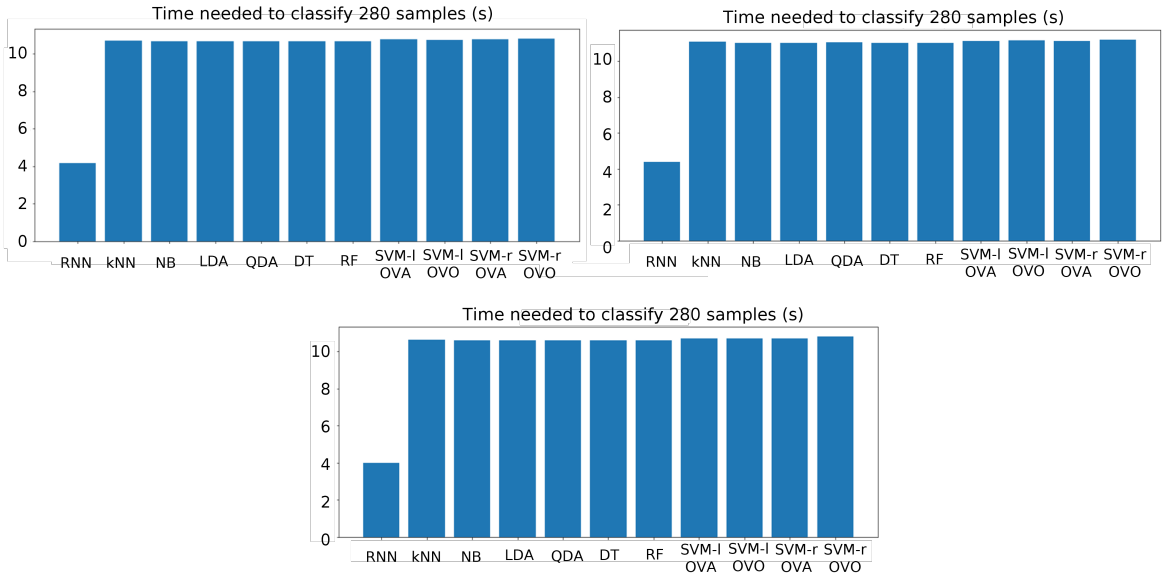


Figure 4.7: Comparison of time needed to evaluate 280 samples with traditional and deep approaches. In each plot, RNN (left bar) refers to the Deep Learning approach. Three deep architectures were tried: LSTM(8) + LSTM(8) (top left), LSTM(32) + LSTM(32) + FC(16) (top right), and CNN1D(64,3) + MaxPool + LSTM(32) + LSTM(32) + FC(16) (bottom).

It is clear that the evaluation time when using deep learning approaches is much smaller than the one needed with traditional methods. In fact, since all the traditional machine learning methods require approximately the same amount of time, it can be concluded that the manual feature extraction process is the part that takes the longest. On the other hand, it can be seen that the three deep architectures took approximately the same time to perform the predictions, even if they were increasingly more complex among them.

A second part of the previous experiment wanted to evaluate how the number of sensors affects the evaluation time of the same models. It was expected to see that “the number of time and frequency domain features (predicting variables) obtained from the accelerometer and gyroscope affects

the performance of the classifier differently" (Zebin et al., 2017). To address this question, the same traditional models and the third deep architecture were trained with three different types of data: only accelerometer signals, accelerometer + gyroscope signals, and accelerometer + gyroscope + magnetometer signals. In all the cases, the same 280 samples were used, but the usage of more sensors meant that more features were needed to extract for the classification. The results of this part of the expreriment can be seen in figure 4.8. It is interesting to note that the evaluation time for each of the three cases is nearly the same in the deep learning approach, while with the traditional approaches it increases when more signals are used.

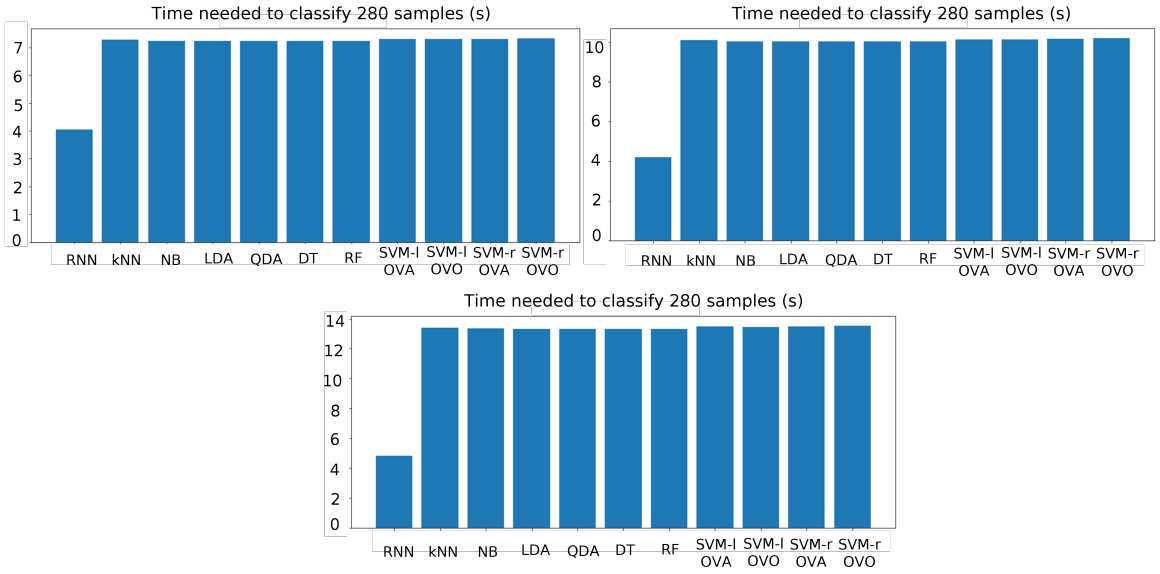


Figure 4.8: Comparison of time needed to evaluate 280 samples with respect to number of signals. In each plot, RNN (left bar) refers to the Deep Learning approach. The same deep architecture was used for the three plots: CNN1D(64,3) + MaxPool + LSTM(32) + LSTM(32) + FC(16). Results using only accelerometer signals (top left), accelerometer + gyroscope signals (top right), accelerometer + gyroscope + magnetometer (bottom)

A number of conclusions were drawn from the previous experiments:

- The usage of time-domain manually selected features can result in a loss of information of the signal.
- The usage of frequency-domain manually selected features can result in a loss of information of the signal.
- When using manually selected features, in order to preserve the most amount of information of the signals, it is needed to combine time- and frequency-domain features and several signals and sensors.
- Deep Learning approaches are significantly faster in evaluation time than traditional methods for HAR.
- The computation of manually selected time- and frequency-domain features is the process that takes the most time in evaluation time when using traditional approaches for HAR. Deep learning approaches does not rely on this process.
- Evaluation time of deep learning approaches does not depend much on architecture (see figure 4.7).
- Evaluation time of deep learning approaches stays nearly constant with respect to number of signals (see figure 4.8).
- Evaluation time of traditional methods is highly sensible to the number of signals. It increases proportionally. (see figure 4.8).

4.3. Activity detection

When the dataset that was used to train the models was described in section 4.1, it was shown in figure 4.2 that the recordings included some measurements of low activity (walking and standing) before and after the actual activity to be recognized. This meant that those activities were not isolated from surrounding noise and, in order to build a more reliable model, it was a good idea to clean the signals so that only the desired activities were present. This follows the logic that the deep model would learn to extract features by itself. If a lot of irrelevant information would be present in the training phase, it could be possible that the model would learn features from the low activity patterns and not from the actual activities, as it is expected. Therefore, to make sure that the model learns to recognize accurately the football movements, an activity detection procedure was developed. This procedure prepared the recordings for the training phase by isolating the important activities from the mentioned low activity intervals. This algorithm is presented in this section.

A low activity measurement is characterized, as it names suggests, by signals with low magnitude and variance, which lies in contrast to the behavior of the signal during a high activity movement. This is specially true when we focus on the acceleration values. When a football player makes a movement after being still or relaxed, the lower limbs accelerate quickly. In figure 4.3, for example, it is not hard to identify when each one of the jumps start and end only by looking at the acceleration patterns of the body parts. This is the reason why only the accelerometer signals (and not gyroscope and magnetometer) were used for the activity detection phase.

On the other hand, this big change in acceleration between a low activity interval and a high activity one can happen in any of the measured body parts. A standing player can start to run with the right leg while another player can move the left leg first. This is also true with the axes (X, Y and Z): when jumping the movement is primarily vertical, but when passing we expect the longitudinal component to be more present. In other words, the transition between a low and a high activity interval can and should be detected with any of the body parts and in any axis. This is the reason why the norm of X, Y and Z axis of each sensor location is used. Each sensor is treated independently and, at the end of the process, they are combined for the final result.

Since there are five sensor locations (pelvis, left thigh, left shank, right thigh, and right thigh), and for each one of them the norm of the X, Y and Z components of their acceleration is taken, we obtain 5 different signals. In order to identify when a high activity happens, a baseline value for each signal is obtained: the mean value. When the player is still or walking, the norm of the signal signal is mainly smaller its mean. However, when the player performs a more intense activity, the norm of the signal presents large peaks larger than its mean value. So, the beginning of a high activity measurement can be found by identifying the moment when the norm of the signal exceeds the mean value. To avoid small meaningless peaks, the algorithm looks for the moment where the norm of the signal and the following 50 timesteps exceed the the mean value. Similarly, the end of the high activity interval is identified by looking at the moment where the norm of the signal and the previous 50 timesteps are larger than the mean value. This procedure is shown in figure 4.9. The five norm signals are shown one on top of the other with their respective starts and ends of activity.

The previous process is done with each one of the five norm signals. This results in five “starts” and five “ends” of the activity. The final step combines these values into a single start and end of the activity. To do so, the minimum along the five “starts” is taken as the overall start and the maximum among the five “ends” is taken as the overall end of the activity. Figure 4.10 shows the result of this part.

In summary, the algorithm proceeds 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 X, Y and Z axis of each sensor location. This results in 5 norm signals.
3. For each one of the 5 signals:

    3.1. Calculate the mean value.

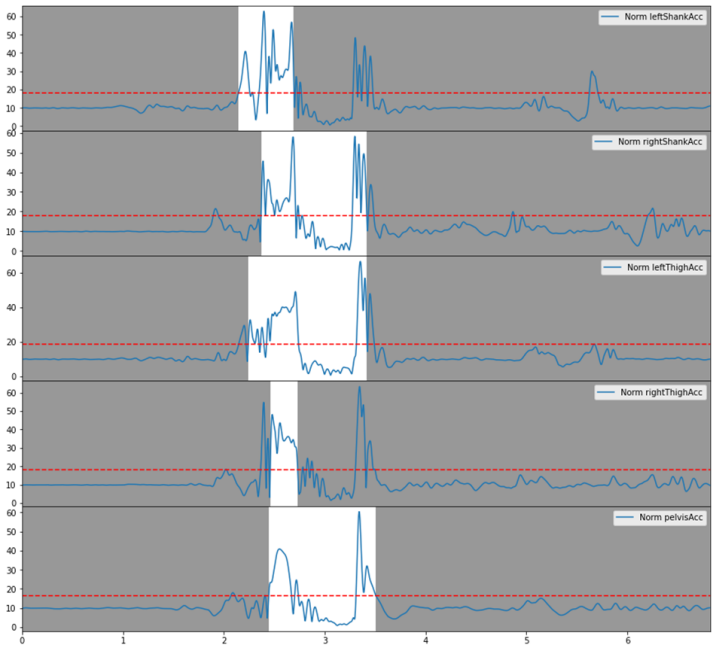


Figure 4.9: Example of starts and ends of activity for the five norm signals. The detected activity for each sensor location is shown in white background.

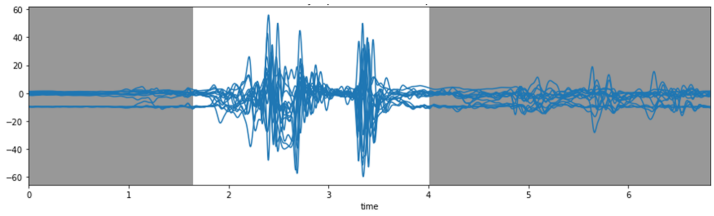


Figure 4.10: Overall start and end of activity from signals of figure 4.9. All the original accelerometer signals are shown. The desired activity is correctly detected.

3.2. Start of activity: find the first timestep where the signal and the following 50 timesteps are larger than the mean value. Take that timestep as the start of the activity.

3.3. End of activity: find the last timestep where the signal and the previous 50 timesteps are larger than the mean value. Take that timestep as the end of the activity.

4. Find the overall start and end of the activity.

4.1. Overall start of activity: take the minimum among the starts of activity from the previous step. Subtract 250 timesteps (if possible).

4.2. Overall end of activity: take the maximum among the ends of activity from the previous step. Add 250 timesteps (if possible).

Note that 250 timesteps (0.5 ms) are taken as margins before and after the overall start and end of the activity. This is done to preserve some context, which is also important.

When this algorithm was implemented and tested, it was found that it worked very good for some activities (see figure 4.10). However, for some other high intensity activities, it was not working as expected. Upon researching the reason of this behavior, it was found that the algorithm was working good with certain type of activities, while with another specific recordings it was having problems. Additionally, it was observed that, for the second group, the results were much better if the threshold was set as 1.5 times the mean of the norm of the signal instead of using simply the mean. Interestingly, after a more detailed investigation of both groups of movements, it was identified that activities such as

jogs, runs and sprints required the mean of the norm signal as threshold, while activities such as jumps, passes and shots performed better when 1.5 times the mean was used. By understanding the nature of these two groups of movements, the former group was renamed as *periodic activities*, in which the activity is repeatedly performed in a periodic manner; and the latter as *explosive activities*, in which the activity is performed only once without a repetitive pattern.

Knowing that periodic activities required the mean as the threshold, but explosive activities performed better when using 1.5 times the mean, the next step was to find a way to automatically detect to which group a movement belonged, so that the correct threshold could be used and the activity could be correctly isolated. Once again, by understanding that explosive activities tend to be quicker and without repetitive patterns, the Interquartile Range (IQR) was proposed as the metric to use to discriminate between both groups of movements. 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. In figure 4.11, the distribution of the IQR values for periodic and explosive activities can be seen. The plot on the top uses un-normalized signals and the bottom one shows the IQR distribution after normalization of the values. Both plots show that the IQR is a good metric to distinguish between both types of activities.

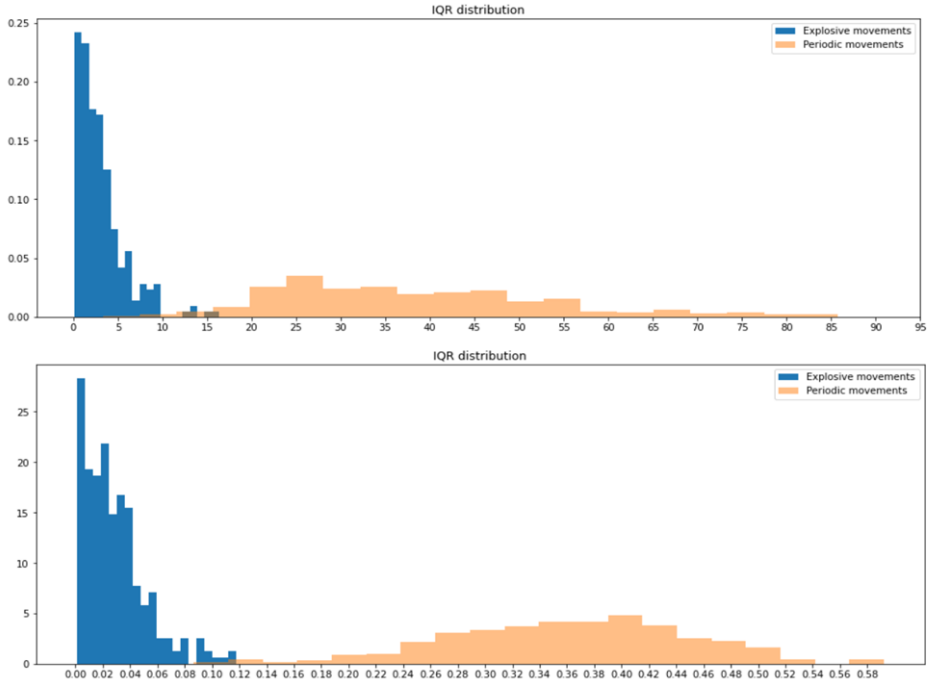


Figure 4.11: Distribution of IQRs for explosive and periodic movements. Un-normalized signals (top) and normalized signals (bottom)

The best threshold for the IQR to make this distinction can be set to 10 for un-normalized signals and 0.12 for normalized ones. These values were selected and the confusion matrices of these classifiers were built (see figure 4.12). Both options had great performance, being the one with normalized signals just better. In terms of performance metrics, the classifier with un-normalized data achieved a classification accuracy of 98.85% and a F1 score of 99.17%. Using normalized data, the values were slightly better: 99.42% and 99.58% respectively.

In conclusion, the full algorithm proposed to detect and isolate a high activity interval from a recording is as follows:

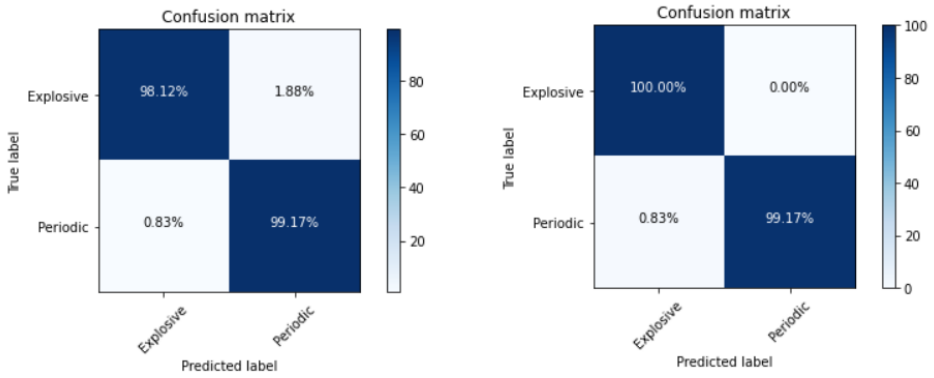


Figure 4.12: Confusion matrices for explosive vs periodic movements classifier based on IQR. Un-normalized signals with threshold 10 (left) and normalized signals with threshold 0.12 (right)

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.
   2.1. Calculate the IQR of the signal.
   2.2. 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:
   4.1. Calculate the mean value.
   4.2. 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
   4.3. 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.
   4.4. 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.
   5.1. Overall start of activity: take the minimum among the starts of activity from the previous step. Subtract 250 timesteps (if possible).
   5.2. Overall end of activity: take the maximum among the ends of activity from the previous step. Add 250 timesteps (if possible).

The previously described algorithm successfully detected high activities from low activity intervals, isolating the relevant regions of high intensity for a potentially more effective training of the model. Some results of detected activities are shown in figure 4.13.

After this process was successfully done, it was possible to investigate the distribution of the lengths of each one of the activities, since they were isolated from the pre- and post- low activity intervals. Figure 4.14 presents these distributions in the form of boxplots. As expected, explosive activities, such as jumps, passes and shoots are shorter and have smaller spread among their duration. Periodic actions like runs and jogs happen usually for longer periods and they can be executed for a more variable length of time.

In figure 4.4, the mean signals for each one of the activities were presented. It was explained that, since those activities did not happen at the same moment in the recordings and they were not centered,

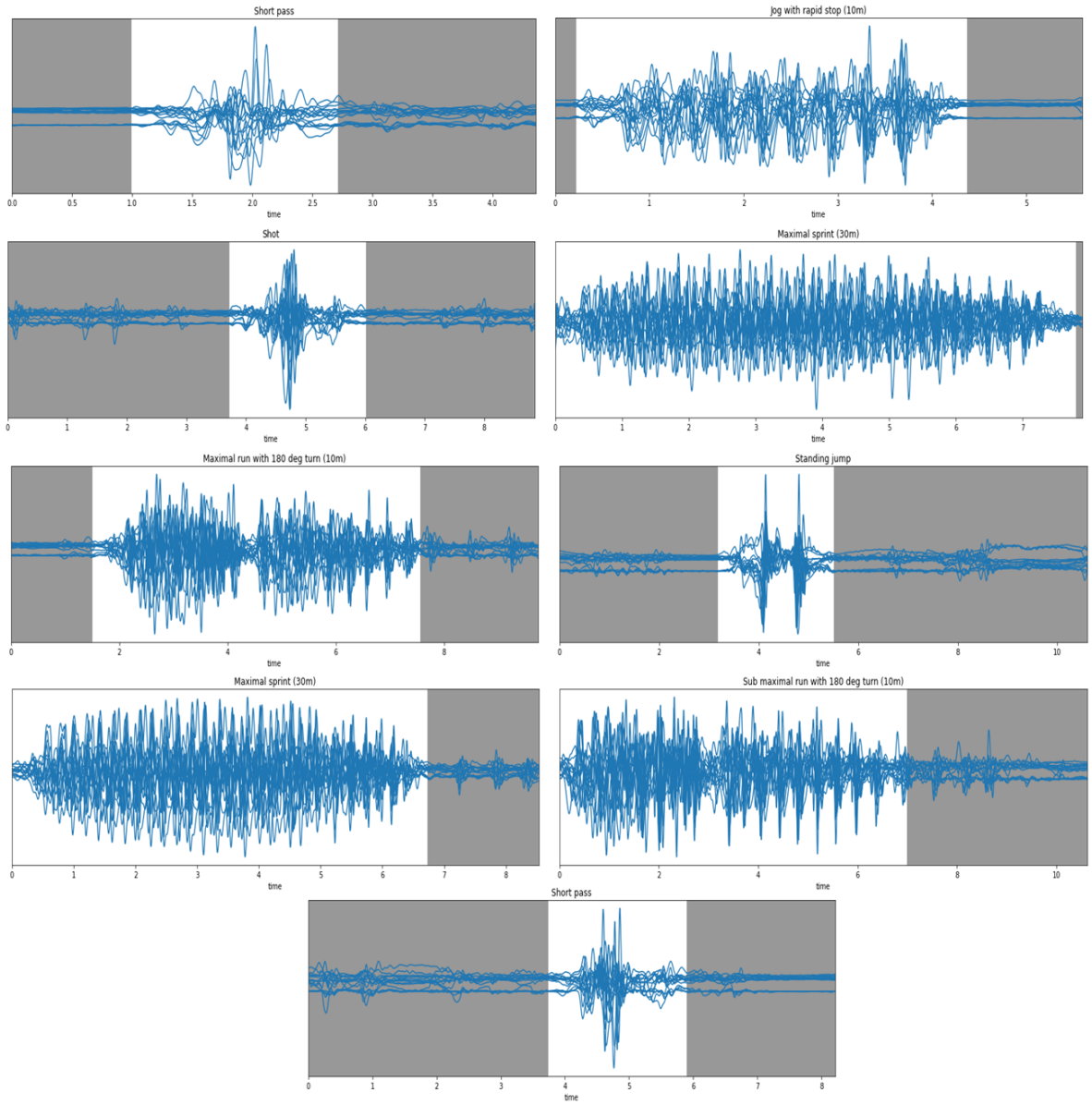


Figure 4.13: Examples of detected activities isolated from the low activity regions

it was impossible to detect some clear pattern of the average behavior of each activity. Nevertheless, after preprocessing the recordings with the activity detection algorithm previously described, the activities were isolated and centered, so the mean activities were again extracted and shown in figure 4.15. In these plots is now possible to identify certain patterns in the activities in contrast to what happened in the plots of figure 4.4. These mean signals allowed to draw another important conclusion: the magnetometers do not provide any valuable information in this case that could help to the final task of football activity classification. The average signals obtained by the magnetometers are, in all the cases, very flat for all the activities and are not going to be taken into account from now on. The signals from the accelerometers and gyroscopes, on the other hand, exhibit detectable patterns and differences among the activities and will be used for the training of the models and classification.

It is very important to mention that this process of activity detection was only applied to the recordings for the training phase, so that the labels used for that phase were correct and not contaminated with low activity measurements. For the evaluation phase, a sliding window approach was used, as will be explained afterwards, so the activities did not have to be isolated in advance.

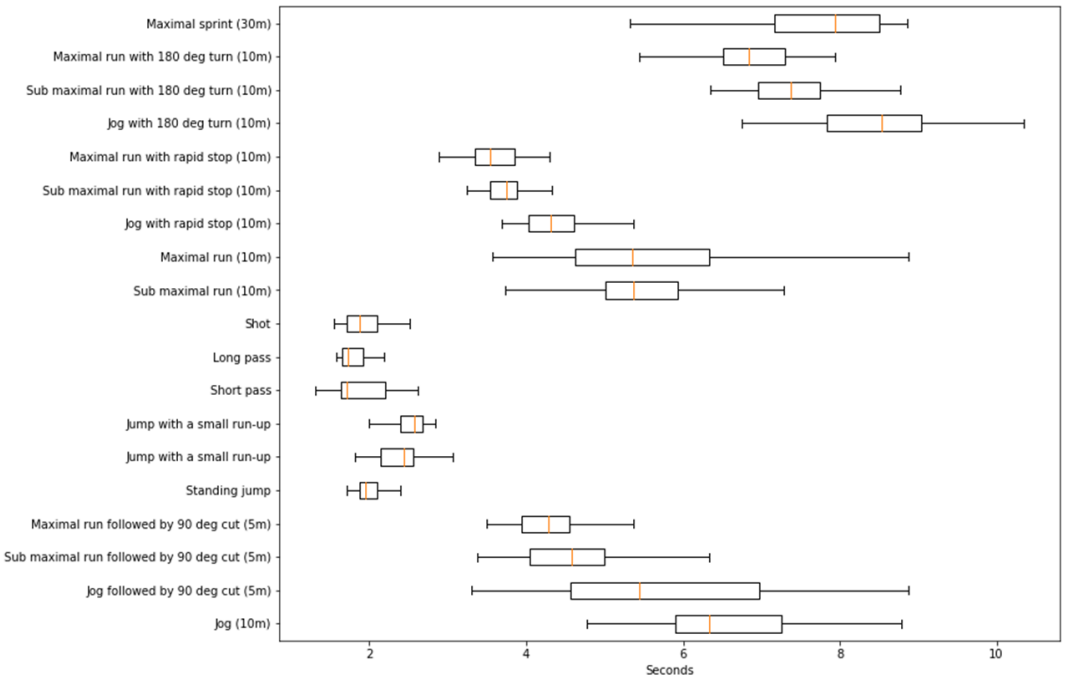


Figure 4.14: Boxplot of the duration in seconds of each activity after they were isolated from low-activity intervals.

Figure 4.15: Mean signals of each activity after process of activity detection. Accelerometer data on the left, gyroscope on the center and magnetometer on the right. Activities from top to bottom: jog, jog followed by 90 degree cut, sub-maximal run followed by 90 degree cut, maximal run followed by 90 degree cut, standing jump, jump with a small run-up, jump with a small run-up, short pass, long pass, shot, sub-maximal run, maximal run, jog with rapid stop, sub-maximal run with rapid stop, maximal run with rapid stop, jog with 180 degree turn, sub-maximal run with 180 degree turn, maximal run with 180 degree turn, maximal sprint

5

Activity recognition

In this chapter, the main part of this work is explained and presented: the design, building and training of Deep Learning models to perform Human Football Activity Recognition. Sticking to the definitions used in the literature, “recognition” here refers to the classification of a part of signal in the specific activity the person who is being sensed is performing. This term must not be confused with what we called “activity detection” in section 4.3, in which the part of the recording where a high intensity activity is detected and isolated from low activity intervals. As it will be seen, the goal was not only to build a model that could achieve an accurate recognition of the activity, but also to do this prediction quickly. A model that can recognize an activity but takes several minutes (or hours) to do so, was not desired. We were looking for a good balance between classification accuracy and time performance.

The most common football activities were used as the different classes to be recognized: pass, shoot, jump, sprint and jog. An additional category called “low activity” was also considered so that the previous five activities could be also distinguished from periods of low activity, such as walking or standing. The deep learning-based models that will be presented in section 5.1 are responsible of the recognition of those five high intensity activities, while the detection of low activity intervals was made using a different technique that will be explained afterwards in section 5.2.

A full diagram summarizing the whole process is shown in figure 5.1. The left part of the figure is where the training of the model happens and will be explained in section 5.1. In sections 5.2 and 5.3, the evaluation phase will be detailed. Finally, in section 5.4, results of the complete pipeline will be presented with their respective analysis.

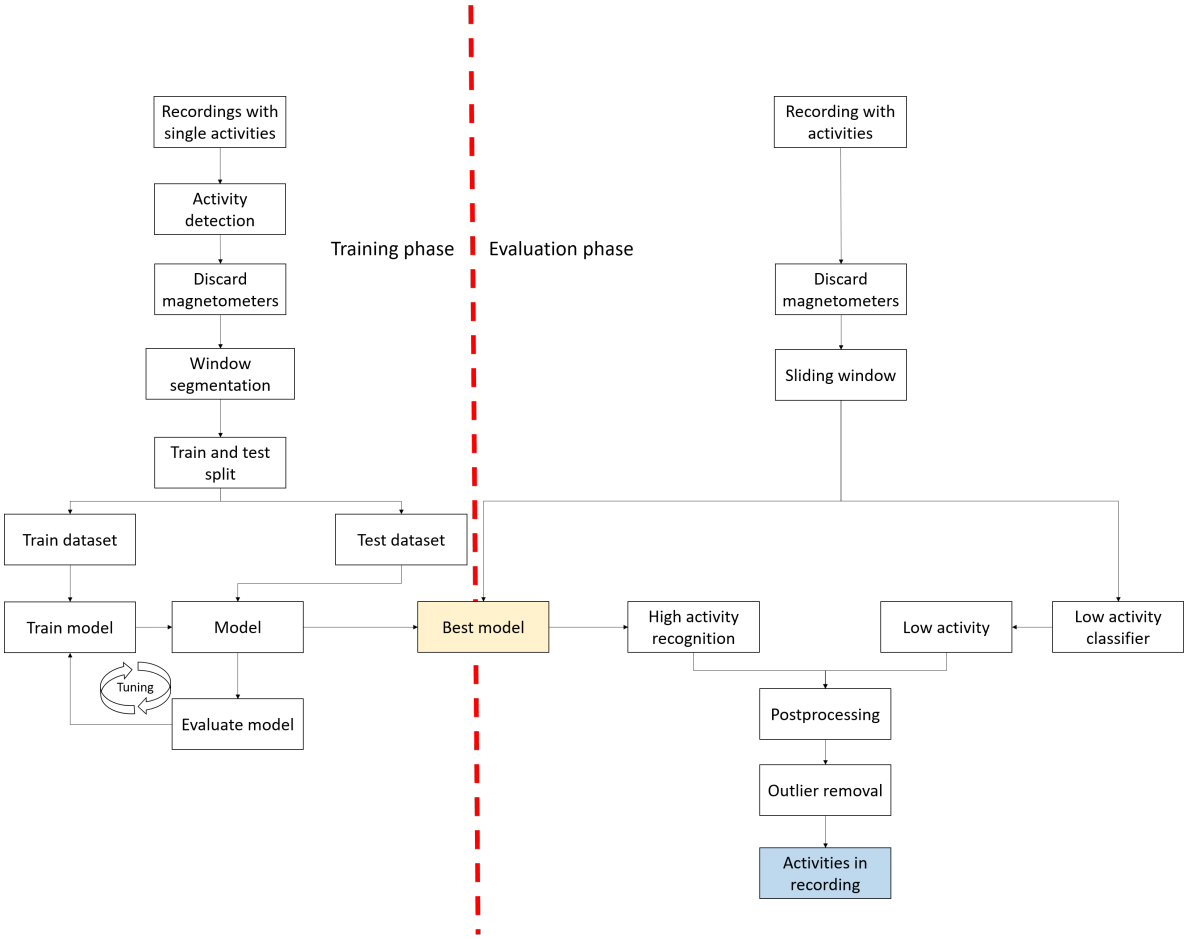


Figure 5.1: Diagram showing the full process of activity recognition with training and evaluation phases

5.1. Activity recognition

A large variety of deep learning models were designed, built, trained and tested to evaluate their capability to recognize the five most common activities in a football match:

- Shot: a strong kick to the ball usually with direction to the opponent team's net in an attempt to score a goal.
- Pass: a softer kick to the ball with the purpose of giving the ball to a teammate.
- Jump: a movement in which the player pushes him or herself vertically off the surface with the feet and legs. Common when attempting to head the ball.
- Sprint (or run): a fast and high intensity horizontal displacement in one direction.
- Jog: a horizontal displacement in one direction faster than a walk and slower than a sprint.

This thesis focused on the previous activities, because they are the most common during football practice and the given datasets were already built and labeled with those movements. Building and labeling new datasets was out of the scope of this work. However, the proposed methodology can be easily used to extend the set of detected activities to capture additional movements that could happen in a football match.

The methodology that was used to build the classifiers (models) is shown in figure 5.2. The first top three blocks, that were already discussed, prepare the dataset for the training and testing phase by following the process of activity detection (section 4.3). The isolated activities are then passed through a process of window segmentation, followed by a split in the train and test datasets. These two datasets are used to train the different models in a recursive manner: using the train subset a model is trained, then it is evaluated using the unseen test split and the process is repeated (tuning) until a best model is achieved in terms of performance. This best model is represented with the light yellow block in the figure. A detailed explanation of each part of this diagram will be given in this section.

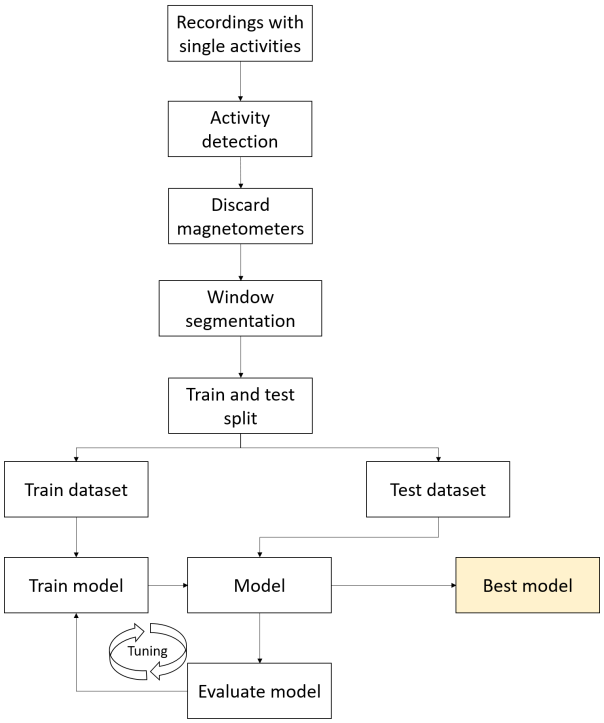


Figure 5.2: Training phase diagram

5.1.1. Window segmentation

In order to build the models for activity recognition, an initial process of activity detection was made (section 4.3), in which recordings that were going to be used to train the models were passed through the algorithm previously described to extract and isolate the actual movements to be recognized. This resulted 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.

The process of activity detection resulted in a dataset with recordings of isolated activities that looked like the parts with white background of image 4.13. As mentioned before, the magnetometer data was not used, so each one of the samples of this dataset was composed of 30 signals (5 sensor locations (pelvis, left thigh, left shank, right thigh, right shank) $\times$ 3 axis (X, Y, Z) $\times$ 2 modes (accelerometer, gyroscope)). It is important to recall that, in order to explore the capabilities of deep networks to work with raw data, the signals were not preprocessed in any matter: no filtering, noise removal or smoothing.

Windows of certain length were extracted from the signals. To do so, a window was traversed through the recordings, extracting at each time the respective interval. The length of the windows was set to be of 1 second. This was defined after an analysis of the duration of the activities (figure 4.14) showed that intervals of 1 second would allow us to capture explosive activities, such as shots, passes and jumps. Although, in general, sprints and jogs last for much more than 1 second, they are periodic movements, which means that the same dynamic is repeated several times, so a window of 1 second would also allow us to extract their patterns correctly. Furthermore, in order to have a large amount of training samples and capture temporal dependencies, an overlap of 75% in the windows was used, meaning that every 250 miliseconds of a recording a new interval of 1 second was extracted.

This strategy of window segmentation is very flexible in the sense that if more windows are required for the training, they can be obtained by using a larger overlap of the sliding windows. A larger overlap will mean that more windows are extracted for each recording, so that the dataset is also expanded. It is clear that the choice of using 1 second-long intervals with 75% interval is somehow subjective (although it was previously explained the reasoning behind those numbers) and these two values could also be used as hyperparameters to explore the difference in performance when windows with different sizes and overlaps are used. This, however, was not explored in this thesis and is left for future experiments.

At the end of this process, a large number of 1 second windows were obtained. The set of these windows is what was used as the datasets when building the models. When building a machine learning model, it is very important to divide the dataset in two (sometimes three) sub-datasets. This is made to reduce the chances or overfitting, where the model performs very good on the samples that are used to train, but poorly with new samples that were not used in the training phase. An overfitted model simply memorizes the patterns of the first phase, but is incapable to generalize to new data. To avoid this situation, the dataset with the windows was divided in two: a train dataset, composed of 70% of the samples; and a test dataset, with the remaining 30%. The former was used to train the models and the latter to evaluate them with unseen samples. A good model would give good classifications on the train and test datasets, but it would also have similar performances on both splits.

5.1.2. Proposed models

In section 2.2, a thorough analysis of previous works on deep learning approaches for Human Activity Recognition was made. It was seen that the usage of Convolutional Neural Networks, Recurrent Neural Networks or a combination of both is a good idea for these tasks. The question to answer, however, is which architecture outperforms the others in our particular application? There are endless possibilities to combine RNN and CNN, both in their internal architectures and in the way they are stacked one after (or in parallel to) the other. A widely known theorem in the Machine Learning world is the *No Free Lunch Theorem*. It states that “for any two learning algorithms, there are just as many situations (appropriately weighted) in which algorithm one is superior to algorithm two as vice versa” (Wolpert, 2001).

This basically means that a particular model or architecture is not going to be the best possible option for all the tasks related a problem. In other words, the architecture that performs the best for Football Activity Recognition is not necessarily going to be the best for Human Daily Activities Recognition or Tennis Activity Recognition. Even more, the different types of sensors and their locations can influence the performance of the model. Nevertheless, this thesis explores different types of deep models for Football Activity Recognition by combining CNNs and RNNs in several ways and by varying the ways the convolutions are made to extract relevant features from the signals. The final objective is to have not only a good accuracy but also to be allow to do the recognition in short time. Because of the No Free Lunch Theorem, it is impossible to conclude that the developed models will perform equally good when recognizing activities of other nature or sport without doing the proper tests. However, the proposed methodology can be used to explore different architectures when facing such situation, so that a good and fast deep model can be built for another specific types of movements. Additionally, the exploration of numerous deep configurations allows to draw significant conclusions that could be extrapolated to more general HAR applications.

As explained before, HAR tasks could benefit from the usage of Convolutional Neural Networks and Recurrent Neural Networks. The former would be responsible of extracting relevant patterns of features from signals and the latter would use those features and give them temporal meaning by understanding the signals as a time series. A combination of both types of layers would be, in theory, very powerful. But, from the literature review and their understanding, it is known that CNNs could not only extract relevant features, but also combine them in subsequent convolutional layers so that (some sort of) temporal relationships could be detected in the signals. This opens the possibility to explore, as many other authors did, the usage of architectures with only CNNs and not RNNs. In a similar way, models that only use RNNs without CNNs can also be explored, giving the responsibility of feature extraction to the RNNs. Wrapping up, this thesis explored architectures solely composed of CNNs, solely composed of RNNs and built with a combination of both.

Convolutional layers
Since the recordings are composed of several signals from different body parts and types of sensor (accelerometer and gyroscope), the first question that arose was about the size and type of convolutions to be made. Should the convolutions be made across all signals, just across each sensor, or just across each signal? Should the convolutional kernels (or filters) be shared among all the signals or should each sensor have independent convolutional kernels? Is only one of those options the best possibility or should we combine more than one type of convolution?

To solve these questions, different variations of convolutional layers were tried to explore the influence of the type of the convolutions in the model's performance. 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 with the same filters for all the signals. In this type of convolution, filters of size $1 \times m$ are used, where m is a hyperparameter that determines the timesteps used in the convolution. The 1 implies that each signal is convolved alone. Additionally, weight sharing means that the same filters are used for all the signals. Figure 5.3 explains this logic. Note that each sensor is composed of three signals (X, Y and Z axis of the sensor). The convolutions are made for each signal using the same set of k filters (represented in red).

- 1DCNN per sensor:
  One-dimensional convolutions with the same filters for all the signals of the same sensor, but different filters for each sensor. In this type of convolution, filters of size $1 \times m$ are used, where m is a hyperparameter that determines the timesteps used in the convolution. The 1 implies

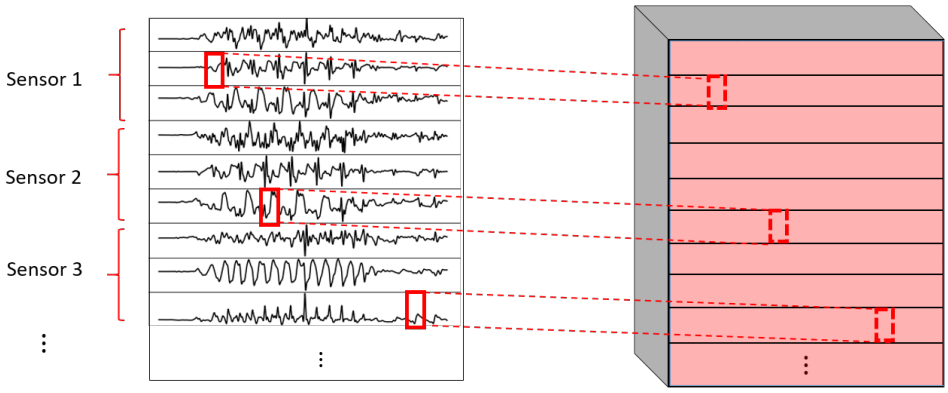


Figure 5.3: 1DCNN weight sharing convolution logic

that each signal is convolved alone. However, each sensor has its own set of k filters, meaning that the filters are not shared among the sensors. This type of convolution follows the idea that since each sensor can present completely different patterns than another sensor, it deserves to have their own filters. This way the features extracted per sensor are specifically optimized for it. Figure 5.4 shows this logic. The convolutions are made for each sensor using, for each one of them, a different set of k filters (but the same set of filters are used for the three axis of the same sensor). The different set of filters are represented with different colors in the figure. This clearly results in a larger model than the 1DCNN weight sharing, since it has now $k \cdot NumSensors$ filters to be learned instead of k of them.

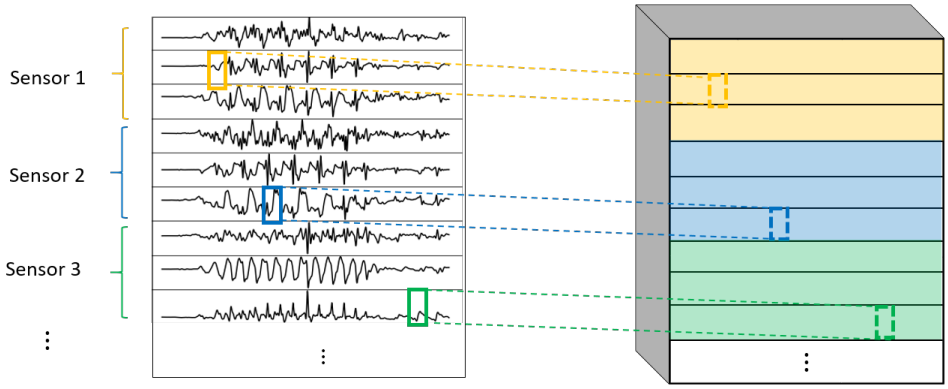


Figure 5.4: 1DCNN per sensor convolution logic

- 1DCC combined:
  Combination of 1DCNN weight sharing and 1DCNN per sensor. Both types of convolutions are performed and their results (feature maps) are concatenated one on top of the other. Figure 5.5 shows this logic. This variation tries to capture the best of the previous one-dimensional convolutions.

- 2DCNN weight sharing:
  Two-dimensional convolutions with the same filters for all the sensors. In this type of convolution, filters of size $3 \times m$ are used, where m is a hyperparameter that determines the timesteps used in the convolution. The 3 means that the 3 axis of the same sensor are used together in the convolution. In order to convolve each sensor by itself so that specific patters can be extracted by combinations of the X, Y and Z signals of the same sensor, a spatial stride of 3 is used. Additionally, weight sharing means that the same filters are used for all the sensors. Figure 5.6 explains this logic. As it can be seen, the convolutions are now made using the signals of the three axis of the same sensor, instead of only one at a time. This results in a smaller feature map, but in larger filters.

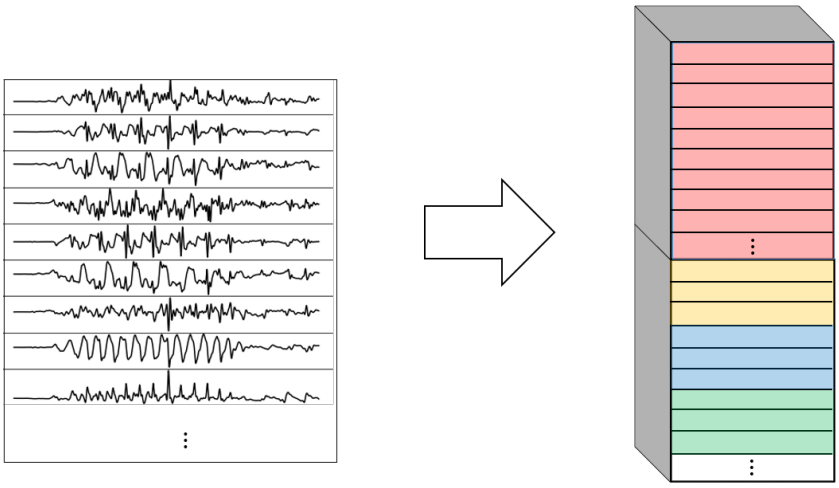


Figure 5.5: 1DCNN combined convolution logic

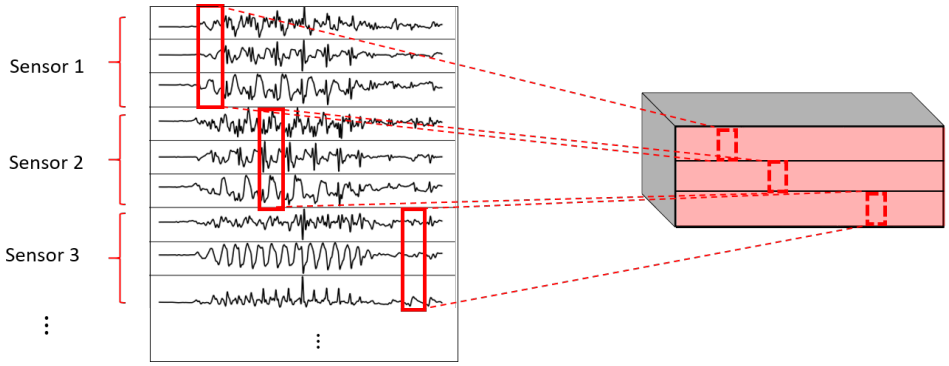


Figure 5.6: 2DCNN weight sharing logic

- 2DCNN per sensor:
  Two-dimensional convolutions with different filters for each sensor. In this type of convolution, filters of size $3 \times m$ are used, where m is a hyperparameter that determines the timesteps used in the convolution. The 3 means that the 3 axis of the same sensor are used together in the convolution. In order to convolve each sensor by itself so that specific patters can be extracted by combinations of the X, Y and Z signals of the same sensor, a spatial stride of 3 is used. Just like 1DCNN per sensor, this type of convolution follows the idea that since each sensor can present completely different patterns than another sensor, it deserves to have their own filters. This way the features extracted per each sensor are specifically optimized for it by combining the information of its three axis in a convolution operation. Figure 5.7 explains this logic. The convolutions are made for each sensor using, for each one of them, a different set of k filters. The different set of filters are represented with different colors in the figure. This clearly results in a larger model than the 2DCNN weight sharing, since it has now $k \cdot NumSensors$ filters to be learned instead of k of them.

- 2DCNN all sensors:
  Two dimensional convolutions made across all the sensors (thus also signals) at once. In this type of convolution, filters of size $NumSensor \times m$ are used, where m is a hyperparameter that determines the timesteps used in the convolution. By performing convolutions that include all the sensors in the operation, we would expect to extract features that capture information about the relationships between all the sensors and signals. Figure 5.8 explains this logic. The filters used for this type of convolution are larger, but the feature map is smaller: the spatial dimension has now size 1.

- 2DCNN combined:

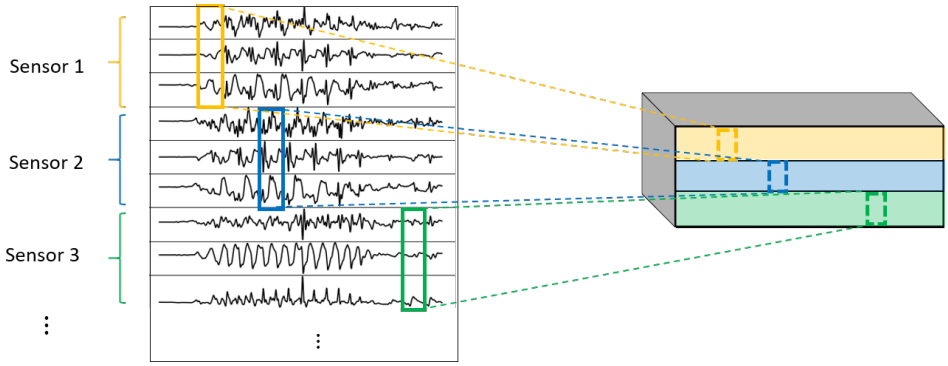


Figure 5.7: 2DCNN per sensor logic

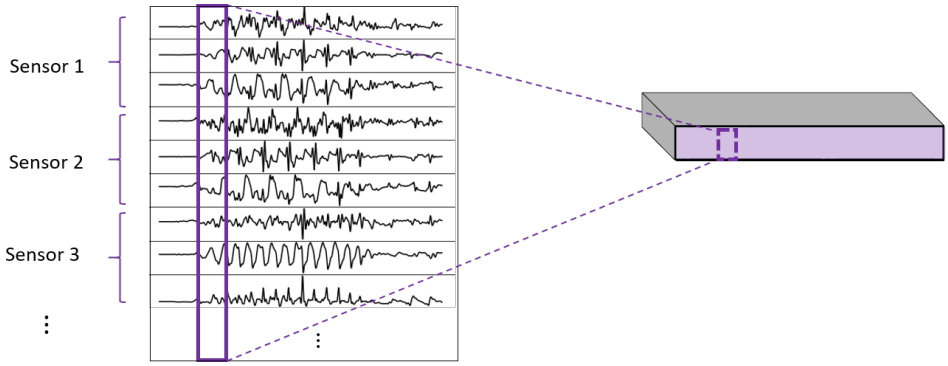


Figure 5.8: 2DCNN all sensors logic

Combination of 2DCNN weight sharing, 2DCNN per sensor, and 2DCNN all sensors. The three types of convolutions are performed and the resulting feature maps are concatenated one on top of the other. Figure 5.9 shows this logic. This variation tries to capture the best of the previous two-dimensional convolutions.

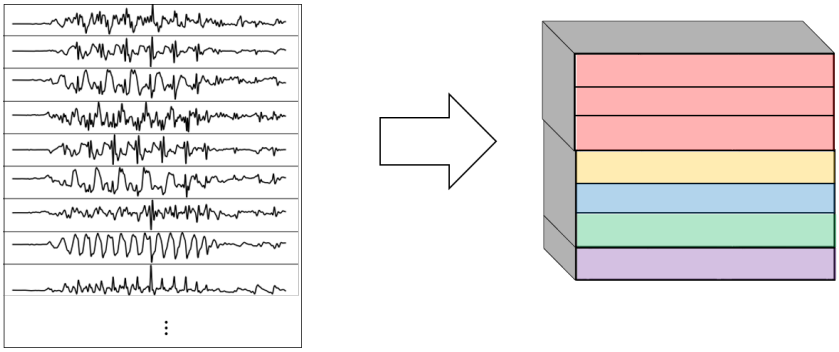


Figure 5.9: 2DCNN combined logic

Recurrent layers

With respect to the usage of Recurrent Neural Networks and their variations, Long Short Term Memory cells (LSTMs) were used. This kind of RNN was chosen to overcome the short-term memory problem that occurs when regular RNNs are used. When working with not too short sequences, like the signals obtained from the sensors, regular RNNs are unable to remember correctly important long-term dependencies, resulting in what is called short-term memory. This is mathematically explained due to the vanishing of gradients along the input sequence: as more recurrent units are used to capture the whole sequence, the gradients of the loss function tend to disappear. And these very small gradients make the training extremely slow and inefficient. The usage of the three different gates in a LSTM cell allows

information to be “remembered” or “forgotten”, and this makes the gradients to vanish much less. As a result of this, LSTMs, as their name suggest, have both long- and short-term memory, allowing the retention of information from both long and short sequences. For this thesis, two different options of LSTMs networks were tried:

- Unidirectional LSTM or simply LSTM, in which the information travels only in one direction.
- Bidirectional LSTM or bLSTM, in which the information travels in both directions: forward and backward. This allows the model to understand temporal dependencies from things that happened in the past and from things that happen in the future.

General architectures
With these two types of neural networks (CNNs and RNNs) and the previously described variations of them, the several models were built, based on only CNNs, only RNNs or a combination of both.

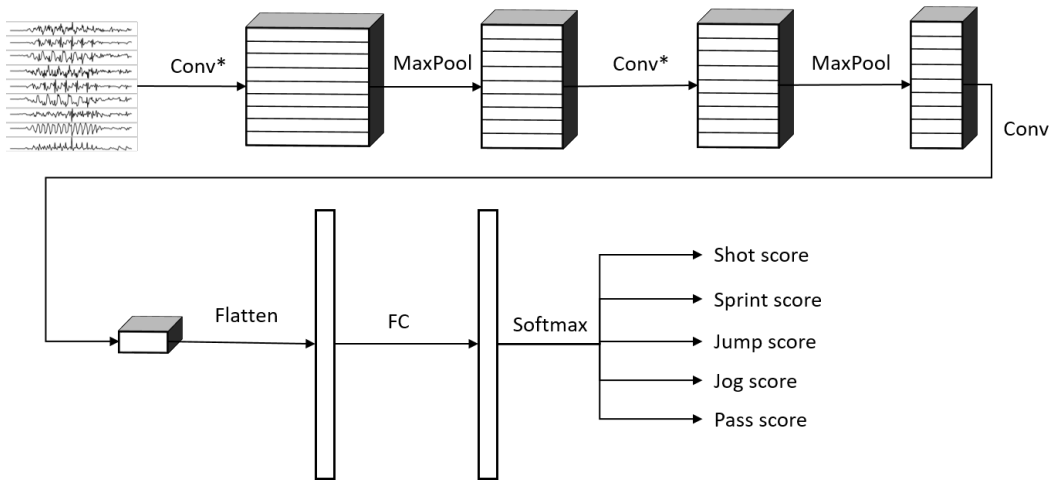
For the models that were only based on CNNs, their general structure was as shown in figure 5.10a. In this architecture, the input signals go twice through a combination of a convolutional layer followed by a max pooling layer. Afterwards, a third convolutional layer is used to obtain a tensor of 2 dimensions, which is then flattened into a one-dimensional vector. That vector is passed through one fully connected layer and finally an output fully connected layer with a softmax activation function is applied to obtain the probability scores for each one of the 5 classes (activities). The first two convolutions (written with an asterisk) vary depending on which type of the previously explained convolutional variations is chosen (figures 5.3 through 5.9). Additionally, to reduce possible overfitting, dropout layers (not shown in the diagram) are applied before each fully connected layer.

For the models that were only based on RNNs, their general structure was as shown in figure 5.10b. In this architecture, the input signals go twice through a RNN-based layer. The first RNN layer returns the whole sequence, so that its output has the same amount of timesteps as the signal. The second RNN layer only returns the value for the final processed timestep, meaning that its output is already a flattened one-dimensional vector. Then, that vector is passed through two fully connected layers and finally an output fully connected layer with a softmax activation function is applied to obtain the probability scores for each one of the 5 classes. The RNN layers there depicted with an asterisk vary depending on which type of the previously explained RNN units is chosen (LSTM or bLSTM). Additionally, to reduce possible overfitting, dropout layers (not shown in the diagram) are applied before each fully connected layer.

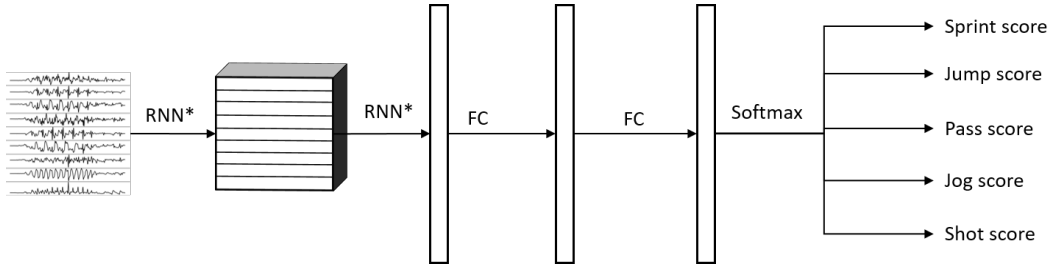
For the models that combined both CNNs and RNNs, their general structure was as shown in figure 5.10c. In this architecture, the input signals go first through a CNN sub-network (figure 5.11a) that implements the convolutional part that does feature extraction. The output of this convolutional sub-network is a three-dimensional tensor. However, the recurrent units need the input to be two-dimensional. This is done with the help of an additional convolutional layer followed by a layer (reshape) that simply rearranges the order or the dimensions to have it ready for the recurrent part. At that moment, the tensor is two-dimensional and is ready to go through the RNN sub-network (figure 5.11b) which is again composed by two layers of LSTMs or bLSTMs that are responsible of extracting temporal dependencies of the extracted features. The output of the RNN-subnetwork is a one dimensional vector that goes through one fully connected layer and finally an output fully connected layer with a softmax activation function to obtain the probability scores of each one of the 5 classes. Additionally, to reduce possible overfitting, dropout layers (not shown in the diagram) are applied before each fully connected layer.

Specific architectures
In summary, the following models were built:

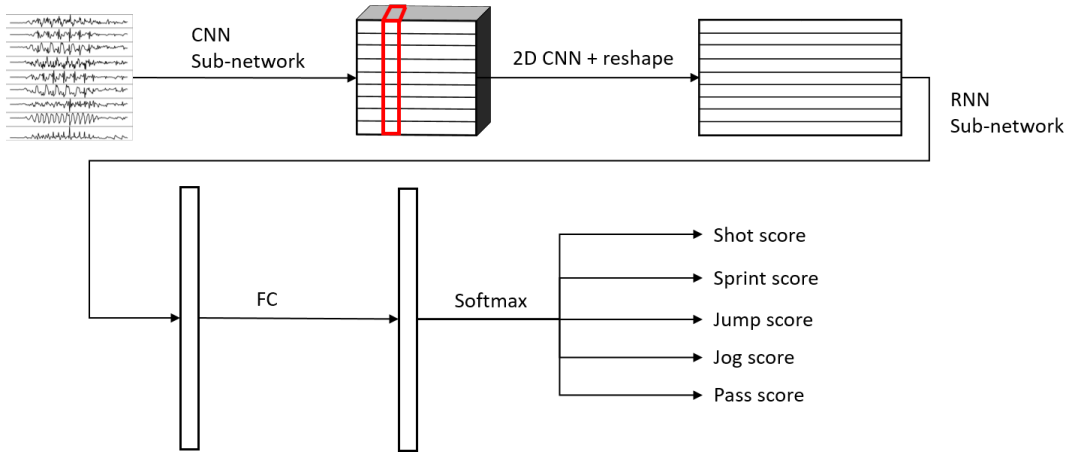
- 1D CNN weight sharing: model based on 1DCNN weight sharing type of convolution.
  Architecture: Input + Conv(16, (1,5), 1, relu) + MaxPool(1,4) + Conv(32, (1,5), 1, relu) + MaxPool(1,4) + Conv(64, (d,1), 1, relu) + Flatten + Dropout(0.2), FC(64, relu) + Dropout(0.5) + FC(classes, softmax)



(a) General architecture for models based on CNNs



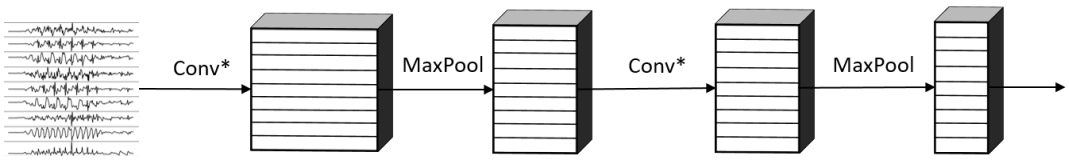
(b) General architecture for models based on RNNs



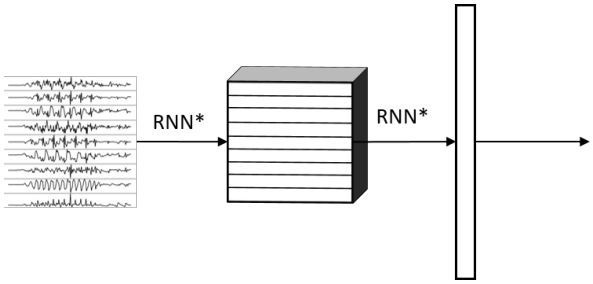
(c) General architecture for models based on combination of CNNs followed by RNNs

Figure 5.10: General architectures of the different types of models built. Three types of models were evaluated: using only CNN, only RNN, or a combination of both. The asterisks on the Conv layers mean the usage of all the variations of convolutions previously explained. The asterisks on the RNN layers represent either LSTMs or bidirectional LSTMs. (FC = Fully Connected Feed Forward Neural Network)

- 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,



(a) General CNN sub-network for the models that combine CNNs and RNNs



(b) General RNN sub-network for the models that combine CNNs and RNNs

Figure 5.11: General CNN and RNN sub-networks for the models that combine CNNs and RNNs. The asterisks on the Conv layers mean the usage of all the variations of convolutions previously explained. The asterisks on the RNN layers represent either LSTMs or bidirectional LSTMs. (FC = Fully Connected Feed Forward Neural Network)

(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)

The input data is given as a matrix of where the rows are the spatial dimensions (signal location) and the columns have the temporal dimension (timesteps of the signals).

For the previous list, the following notation was used:
Conv(f, (m,n), s, act): Convolutional layer with f filters of size (m,n) with s strides in the vertical (spatial direction) followed with act activation function.
Conv_perSensor(f, (m,n), s, act): 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 act 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, act): Fully connected feed-forward layer of n units followed with act activation function.
d: Number of spatial dimensions of the output tensor of the previous layer.
$classes$: Number of classes.

5.1.3. Training

The previous 23 network architectures were built, but each one of them was trained 4 times: using both accelerometer and gyroscope data with and without standardization; and using only accelerometer data with and without standardization. By standardization we mean setting the values of each signal to mean 0 and variance 1. This can be done with the formula:

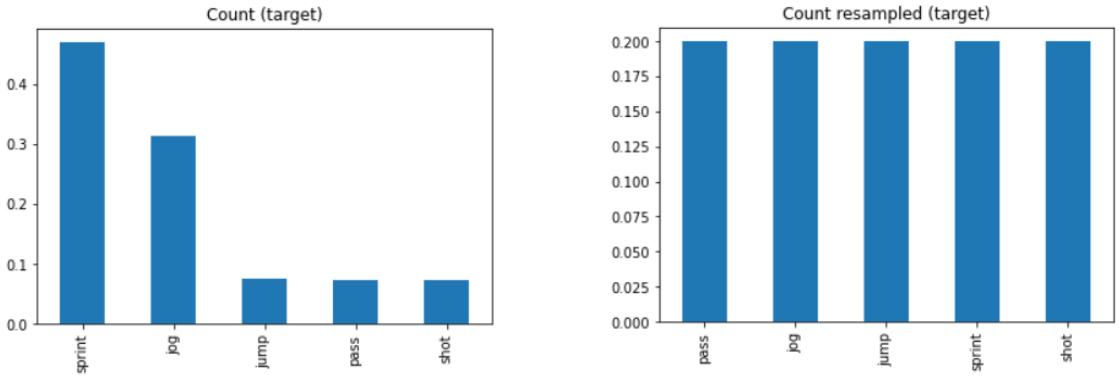
$$X_{\text{standardized}} = \frac{X - \mu_X}{\sigma_X}$$

Where μ_X is the mean of X and σ_X is its standard deviation.

The reason to train the models with those 4 variations of the input data was because we wanted to explore the effect of the inclusion of gyroscope data or if the accelerations were descriptive enough for the predictions. Standardized data was used to explore the effect of modifying the internal scale of the

signals.

Before the training process of the models, an additional procedure was performed: the dataset was balanced. When building classification models it is a good practice to balance the dataset if it is heavily unbalanced, because an unbalanced dataset could force the model to be biased towards the common classes, without learning properly the not so common ones. As it can be seen in figure 5.12a, the dataset used for training had a very unbalanced distribution of the classes, having significantly more samples of sprints and jogs than of the other activities. There are several ways to balance a dataset if it is not feasible to enlarge naturally the dataset: from re-sampling, to generating new artificial data with techniques such as SMOTE (Chawla et al., 2002) and their variations (Fernández et al., 2018), or even with generative neural networks. Due to the nature of the data and the complications related to verify experimentally the validity of artificially generated samples, balancing of the dataset was made with under-sampling of the classes, so that they all had the same amount of samples as the least frequent one. The result of this procedure is shown in figure 5.12b, where all the activities in the training dataset had the same amount of samples.



(a) Unbalanced dataset for training

(b) Balanced dataset for training

Figure 5.12: Unbalanced and balanced datasets for the training process

To train the models, the ADAM optimization algorithm (Kingma & Ba, 2014) was used. In all cases, the chosen parameters where $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$. The learning rate ($\alpha$) was set to decay during the learning phase using a learning rate scheduler. By doing this, the model was able to take large steps towards the optimum during the initial phases of the training, and, as the model approached this point, the steps taken were smaller. This resulted in a better and faster training scheme. The learning rate schedulers for the models were defined as:

- For all the models without any LSTM or bLSTM component, the learning rate was initialized at 0.001. After every 10 epochs of training, it was reduced to its 75%.
- For all the models with only LSTM or bLSTM (no convolutional part), the learning rate was initialized at 0.0001. After every 10 epochs of training, it was reduced to its 50%.
- For all the models that combined a CNN part with a LSTM or bLSTM part, the learning rate was initialized at 0.00005. After every 10 epochs of training, it was reduced to its 75%.

An example of a learning rate schedule can be seen in figure 5.13.

The training was made with batches of 32 samples for at most 200 epochs. Additionally, an early stopping criteria was defined in order to reduce overfitting and unnecessary training: the validation loss was monitored and if it did not improve (reduce) for 5 consecutive epochs, the training was halted.

As explained before, the dataset was divided in two: a train dataset, composed of 70% of the samples; and a test dataset, with the remaining 30%. The former was used to train the models and the latter to evaluate them with unseen samples. Each model was trained 5 times using different train-test partitions, so that every case used a different subset of data for training and testing. The resulting 5

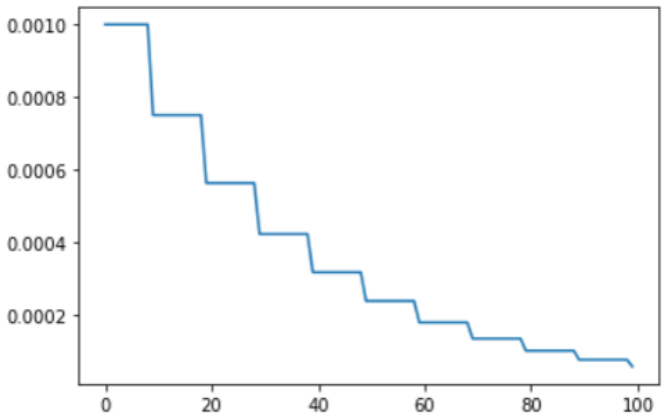


Figure 5.13: Example of learning rate schedule for training

train and test accuracies of the trainings of each model were averaged to obtain their overall performance.

To take advantages of the capabilites of GPUs to train deep neural networks, the models were trained in Google Colab. This online platform offers reliable access to GPU systems that helped to accelerate the experiments and training processes.

5.1.4. Performance

To evaluate the performance of the models, since the datasets were previously balanced, the chosen metric was the prediction accuracy. This is defined as the ratio of correctly classified data points out of all the labelled samples. This number lies between 0 and 1 (100%), being 0 the worst case.

$$\text{Accuracy} = \frac{\text{Number of correctly classified samples}}{\text{Number of samples}}$$

The mean accuracies and standard deviations for the 5 trainings for each model can be seen in tables 5.1 and 5.2 respectively. In those figures, a red-to-blue color code was used as background color of the accuracies on the test dataset, where red is worse and blue is better. No color code was given for the train dataset for visualization clarity. Also, the differences between train and test accuracies are also depicted, in order to have an idea of how overfitted each model was. The confusion matrices for one of the trainings of each model (for the version with unstandardized data from accelerometers and gyroscopes) are shown in appendix B. The models were correctly trained and effectively learned to classify football activities obtaining, in general, high accuracies. By training each model 5 times with different train-test splits, we obtain more reliable accuracy metrics which are not dependent on the luck of choosing an “easy” pair of training and test subsets. Further detailed analysis of these results will be discussed in chapter 6.

5.1.5. Evaluation times

The main goal of the built model was not only to have a high accuracy when recognizing football activities, but it was also desired that those classifications could be made in short time, since the idea is to have real-time (or near to real-time) classifications. In section 4.2, a initial and preliminary analysis of the classification times for traditional and deep models was performed using the data and procedure from Kaketsis, 2020. That first analysis suggested that deep learning approaches would be faster in evaluation time than traditional models, mainly explained by the need of a manual process of feature selection when using the latter type of classifiers.

In this section, the evaluation times for both deep and traditional models were computed again but using the dataset used to train the models. By doing this, it was possible to draw more specific conclu-

	Acc + Gyro						Acc					
	Standardized			Unstandardized			Standardized			Unstandardized		
	Train	Test	Difference	Train	Test	Difference	Train	Test	Difference	Train	Test	Difference
1D CNN weight sharing	99,74%	90,79%	8,95%	99,13%	94,36%	4,77%	99,11%	88,00%	11,11%	99,15%	93,53%	5,62%
1D CNN per sensor	99,88%	92,93%	6,95%	98,35%	93,92%	4,43%	99,06%	88,05%	11,01%	98,26%	93,86%	4,40%
1D CNN combined	99,76%	92,22%	7,54%	98,66%	93,97%	4,69%	97,44%	87,45%	9,99%	96,00%	92,82%	3,18%
2D CNN weight sharing	98,96%	90,58%	8,38%	98,80%	93,59%	5,21%	98,42%	85,53%	12,89%	96,96%	92,27%	4,69%
2D CNN per sensor	99,67%	91,45%	8,22%	99,32%	94,79%	4,53%	97,84%	86,36%	11,48%	97,79%	93,86%	3,93%
2D CNN all signals	99,98%	93,37%	6,61%	97,58%	92,88%	4,70%	99,11%	89,97%	9,14%	96,61%	92,82%	3,79%
2D CNN combined	99,79%	94,74%	5,05%	99,34%	95,34%	4,00%	99,81%	91,56%	8,25%	96,02%	91,73%	4,29%
1D CNN weight sharing + LSTM	97,91%	92,93%	4,98%	99,01%	96,05%	2,96%	96,21%	90,96%	5,25%	97,84%	95,45%	2,39%
1D CNN per sensor + LSTM	99,15%	95,07%	4,08%	98,31%	95,95%	2,36%	93,69%	87,62%	6,07%	98,31%	95,56%	2,75%
1D CNN combined + LSTM	99,27%	94,68%	4,59%	98,87%	96,55%	2,32%	97,27%	91,34%	5,93%	97,41%	94,79%	2,62%
2D CNN weight sharing + LSTM	95,18%	90,36%	4,82%	98,87%	96,27%	2,60%	94,33%	87,84%	6,49%	97,74%	95,23%	2,51%
2D CNN per sensor + LSTM	98,54%	92,82%	5,72%	99,32%	96,71%	2,61%	96,35%	89,81%	6,54%	98,21%	95,18%	3,03%
2D CNN all signals + LSTM	97,06%	91,07%	5,99%	98,94%	95,45%	3,49%	95,11%	90,41%	4,70%	98,00%	95,45%	2,55%
2D CNN combined + LSTM	98,56%	93,48%	5,08%	99,32%	96,71%	2,61%	97,91%	92,11%	5,80%	98,99%	96,55%	2,44%
1D CNN weight sharing + bLSTM	98,61%	94,25%	4,36%	99,08%	96,55%	2,53%	95,46%	90,74%	4,72%	98,87%	95,73%	3,14%
1D CNN per sensor + bLSTM	99,15%	94,96%	4,19%	98,66%	96,38%	2,28%	96,26%	90,85%	5,41%	98,40%	96,38%	2,02%
1D CNN combined + bLSTM	99,32%	95,40%	3,92%	99,22%	96,71%	2,51%	95,29%	91,23%	4,06%	97,39%	95,07%	2,32%
2D CNN weight sharing + bLSTM	99,32%	93,75%	5,57%	99,20%	96,22%	2,98%	96,78%	91,29%	5,49%	98,75%	95,95%	2,80%
2D CNN per sensor + bLSTM	98,35%	93,21%	5,14%	98,99%	96,11%	2,88%	96,92%	91,51%	5,41%	98,64%	95,89%	2,75%
2D CNN all signals + bLSTM	98,59%	92,99%	5,60%	98,45%	94,96%	3,49%	94,99%	90,47%	4,52%	97,88%	95,01%	2,87%
2D CNN combined + bLSTM	99,22%	94,36%	4,86%	99,29%	96,00%	3,29%	97,74%	92,60%	5,14%	98,89%	96,11%	2,78%
LSTM	85,69%	75,29%	10,40%	91,48%	77,48%	14,00%	79,46%	71,67%	7,79%	84,92%	72,66%	12,26%
bLSTM	98,52%	85,21%	13,31%	99,65%	87,07%	12,58%	90,82%	78,19%	12,63%	96,73%	83,73%	13,00%

Table 5.1: Mean prediction accuracies for the proposed models

	Acc + Gyro				Acc			
	Standardized		Unstandardized		Standardized		Unstandardized	
	Train	Test	Train	Test	Train	Test	Train	Test
1D CNN weight sharing	0,24%	1,26%	0,81%	1,33%	1,10%	3,55%	0,29%	0,28%
1D CNN per sensor	0,13%	0,92%	1,68%	1,78%	0,69%	2,73%	0,48%	1,12%
1D CNN combined	0,13%	1,71%	0,33%	1,48%	1,84%	3,80%	0,89%	1,90%
2D CNN weight sharing	1,23%	2,64%	0,49%	1,60%	0,97%	3,68%	1,30%	1,02%
2D CNN per sensor	0,54%	1,80%	0,46%	1,42%	1,95%	3,69%	1,58%	0,45%
2D CNN all signals	0,05%	1,90%	0,87%	1,38%	1,22%	3,07%	2,56%	2,87%
2D CNN combined	0,22%	0,97%	0,44%	1,42%	0,16%	1,36%	2,65%	1,42%
1D CNN weight sharing + LSTM	1,05%	1,35%	0,52%	1,21%	1,53%	3,06%	0,57%	0,82%
1D CNN per sensor + LSTM	0,31%	1,39%	1,22%	0,87%	1,54%	2,28%	0,42%	0,50%
1D CNN combined + LSTM	0,58%	1,44%	0,62%	1,60%	1,18%	1,76%	0,61%	0,67%
2D CNN weight sharing + LSTM	6,61%	6,05%	0,92%	1,00%	4,12%	6,02%	0,73%	0,51%
2D CNN per sensor + LSTM	0,72%	0,84%	0,39%	1,31%	1,25%	1,45%	0,77%	1,12%
2D CNN all signals + LSTM	1,32%	1,23%	0,58%	1,02%	1,31%	1,19%	1,18%	0,51%
2D CNN combined + LSTM	0,99%	1,18%	0,37%	1,05%	0,88%	1,68%	0,27%	0,51%
1D CNN weight sharing + bLSTM	1,01%	1,55%	0,34%	0,73%	2,31%	1,79%	0,40%	1,16%
1D CNN per sensor + bLSTM	0,51%	1,76%	0,60%	0,53%	0,87%	0,51%	0,86%	1,02%
1D CNN combined + bLSTM	0,49%	1,02%	0,53%	1,54%	1,55%	1,87%	0,56%	0,79%
2D CNN weight sharing + bLSTM	0,35%	1,68%	0,41%	0,96%	1,30%	1,52%	0,86%	0,56%
2D CNN per sensor + bLSTM	1,05%	1,36%	0,75%	1,10%	1,94%	2,46%	0,42%	0,90%
2D CNN all signals + bLSTM	0,86%	1,39%	0,96%	0,99%	0,79%	1,50%	1,03%	0,66%
2D CNN combined + bLSTM	0,71%	1,13%	0,36%	1,50%	0,63%	1,70%	0,74%	1,00%
LSTM	3,02%	0,76%	2,17%	3,36%	4,51%	4,06%	2,08%	1,37%
bLSTM	0,66%	2,39%	0,24%	2,25%	2,62%	1,86%	1,73%	1,63%

Table 5.2: Standard deviation of the prediction accuracies for the proposed models

sions related to the research question.

As discussed in the previous paragraphs, several type of architectures were implemented and trained to recognize football activities. In general, the majority of them showed good results in terms of prediction accuracy. Nevertheless, it was also important to verify the time the models required to perform a classification of an activity. Different types of architectures imply different types of computations and layers, which are translated in more or less time needed to classify. Therefore, each model is defined by both its prediction accuracy and evaluation time. Figure 5.14 shows these two metrics for one run of a selection of models. The blue bars (with left vertical axis) represent the time needed to classify 365 windows, while the accuracy for those predictions can be seen with the blue line (with right vertical axis). The usage of LSTM and bLSTM cells allows the model to take into account temporal relationships of the extracted features by the CNN. This makes the model more accurate, as more complex patterns can be detected, but, at the same time, makes the models slower in general. Therefore, a choice has to be made to balance a higher prediction accuracy by using RNN cells with a larger evaluation time. With this in mind, based on figure 5.14, a good choice of model could be model 2DCNN_perSensor_bLSTM, that achieves a large prediction accuracy with relative small evaluation time.

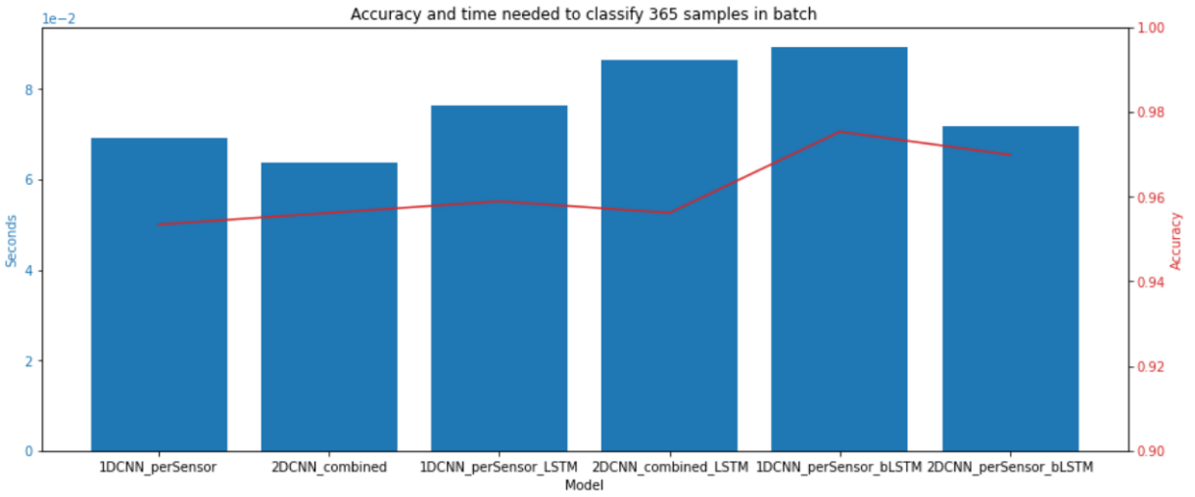


Figure 5.14: Prediction accuracy and evaluation time for some deep learning-based models. These models were trained using Accelerometers and Gyroscopes without standardization. Blue bars (left vertical axis) represent evaluation time and the red line (right vertical axis) the prediction accuracy.

Similarly to what was done in section 4.2, various traditional models were trained using the same data that the deep learning-based models used. In particular, the following classifiers were built: k-Nearest Neighbors, Naïve Bayes, Quadratic Discriminant Analysis, Decision Tree, Random Forest, SVM with linear kernel version ECOC (Error Correcting Output Codes), SVM with gaussian kernel version ECOC, SVM with linear kernel version One-vs-One, SVM with gaussian kernel version One-vs-One, SVM with linear kernel version One-vs-Rest, and SVM with gaussian kernel version One-vs-Rest. For all of them, the following manually selected features were extracted: mean, median, standard deviation, maximum, minimum, skewness, kurtosis, sum of real coefficients of Fast Fourier Transform, and maximum of real coefficients of Fast Fourier Transform. The evaluation times and accuracies of these models in comparison to a particular deep learning-based model (2DCNN per sensor + bLSTM, abbreviated as DNN on the graph) are presented in figure 5.15. For the traditional methods, these times include the manual feature extraction process. As expected, and following the same conclusions that were given in section 4.2, deep learning-based models are significantly faster in evaluation time than traditional approaches for the current problem. For the performed experiments, traditional models required around 4 to 9 times more time to perform the classification of the same windows than the deep learning-based models. Additionally, the traditional models tend to have lower prediction accuracies than the deep ones. In conclusion, these experiments support the hypothesis that, for Football Activity

Recognition tasks with the selected sensor configurations, deep learning-based models make better and faster activity recognitions than traditional machine learning models. Their ability to automatically extract features and abstract temporal and spatial relationship among them, make neural network-based models better in general for the task in question.

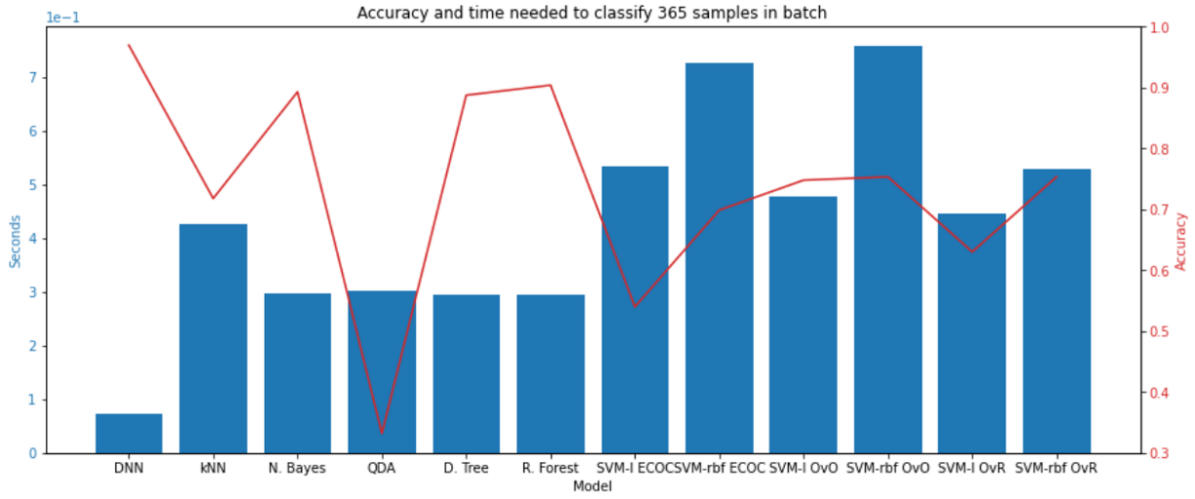


Figure 5.15: Prediction accuracy and evaluation time for traditional models in comparison to a deep learning-based model (DNN). Blue bars (left vertical axis) represent evaluation time and the red line (right vertical axis) the prediction accuracy.

5.2. Low activity recognition

The previous sections explained in detail how the main part of the Football Activity Recognition models were built and trained, the input data, the window segmenting approach, the proposed models, their training scheme, their results, and their evaluation times. The result of this process is a set of viable, efficient and accurate deep learning-based models capable of distinguishing different football activities among them. For this case, the five most common activities were used (shots, passes, runs, sprints and jogs), however, the same process could be followed to train the models to recognize additional activities, as long as we would have enough isolated and labeled training samples of those new movements.

Once the models were successfully trained, the next step consisted on using the model to actively recognize and classify the activities present in a recording. This is called the evaluation phase and is shown with the diagram of figure 5.16. The best model, which is represented with the light yellow block in the figure is one of the models that were previously trained that perform well in terms of evaluation time and accuracy. An unseen recording of a player performing a number of activities is used as the input. This recording can be of a few seconds or even minutes with one or more than one activity measured. An important restriction is that the signals must have the same sampling frequency than the signals that were used to train the models, in this case 500 Hz. So, if the input recording is not sampled at 500 Hz, it is needed to resample it so it complies with the required value. Additionally, the recording must have the same number and type of signals (i.e. triaxial accelerometer and/or gyroscopes on the same body parts) and the sensors must be ideally calibrated and oriented in the same way as the training measurements. These requirements will make the signals to have a similar scale and distribution as the data that was used to train the model on, so that the model will be better suited to handle this new unseen data and classify it correctly, represented with the light blue block on the diagram.

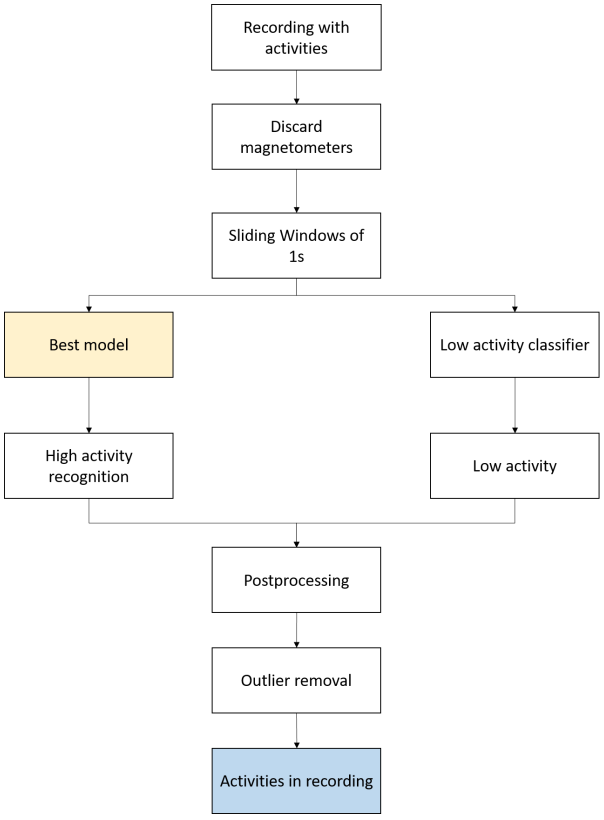


Figure 5.16: Evaluation phase diagram

A sliding window approach is followed, which is explained in detail in section 5.3. This process implies that a window of 1 second is swept through the recording and, for each position of the window, a prediction of the activity there present is made. By doing this, several windows will have periods

where no relevant activity is made, either because the player is just standing or walking passively. Therefore, the model also needs to be able to recognize these low activity periods and classify them as such. There are, in general, two ways to do this:

1. Consider “low activity” as another activity, extract training samples and retrain the deep learning-based models to also recognize this category just like it recognizes shots, sprints, jogs, passes and jumps.
2. Build a binary classifier on top of the already trained deep learning-based model that would be responsible to recognize whether a window consists of a low activity period.

The second approach was chosen. The reasons of this choice are mainly two. Firstly, it is clear that the structure and pattern of the signals during a low activity period are significantly different than when any of the high activities are performed. 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. In second place, the models that were previously built were capable of recognizing high activities with a very small error rate (up to 3-4%). Adding an additional class to the model such as “low activity” will, in the best case scenario, keep the accuracy as it is, but will probably increase the error by some few percentage points. Because of this, it made more sense to build a very accurate binary classification model responsible of discriminating windows with low activities to windows with high activities. If a window would be classified by this binary classifier as a high activity window, it would be passed then through the previously built accurate deep learning-based model to classify the interval as a shot, pass, sprint, jog or jump. By doing this, the general model would be very accurate by combining two highly discriminant classifiers. In figure 5.17, this process is summarized in a diagram.

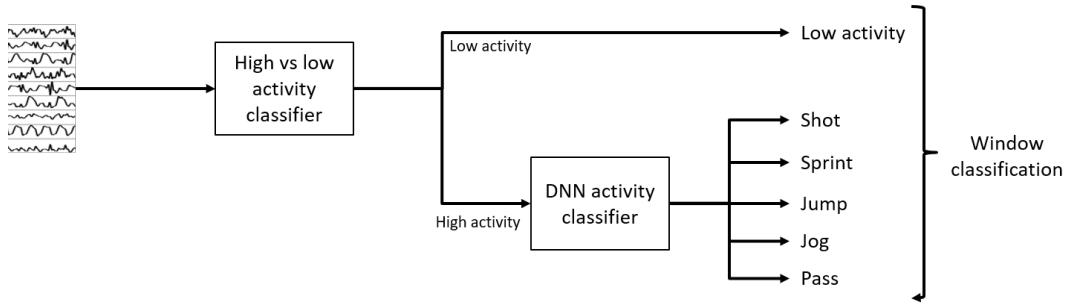


Figure 5.17: Evaluation phase diagram

To build this binary classifier, several values were calculated and evaluated for their discriminant power between low and high activity periods. This classifier could also be built with a machine learning or deep learning approach, but due to the clear difference in signals between high and low activities (low activities have smaller amplitudes and variances), it was not needed. The following metrics were evaluated for the euclidean norm of the accelerometer signals:

- Mean
- Standard deviation (std)
- Coefficient of variation (cv)
- Interquartile range (IQR)
- Range

All of these metrics were calculated for windows with high activities and low activities taking the euclidean norm of the accelerometer signals. The distributions of the metrics for those two groups were plotted (figure 5.18) and examined to identify which one of the measurements would have more

discriminatory power between both categories. Additionally, a two-sample Kolmogorov-Smirnov (KS) test was used to evaluate the similarity of these high and low activity distributions. The two-sample KS test is a statistical test used to check if two samples come from the same distribution. This test returns a value (KS value). The larger the KS value, the more certain we are that both samples come from different distributions. These KS values should be addressed with the help of p-values to draw statistically consistent conclusions about both distributions, but in this case, they were just used to quantify in some sense the similarity of both histograms. The resultant KS values for the selected metrics are summarized in table 5.3.

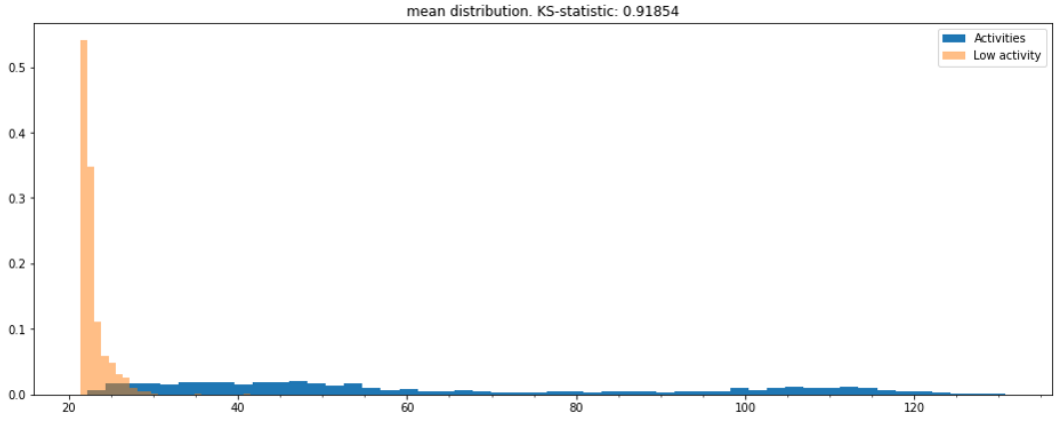
Table 5.3: Two-sided KS values for the metric distributions of high and low activity windows

Metric	**KS value**
Mean	0.9185
Std	0.9302
CV	0.7997
IQR	0.9237
Range	0.9155

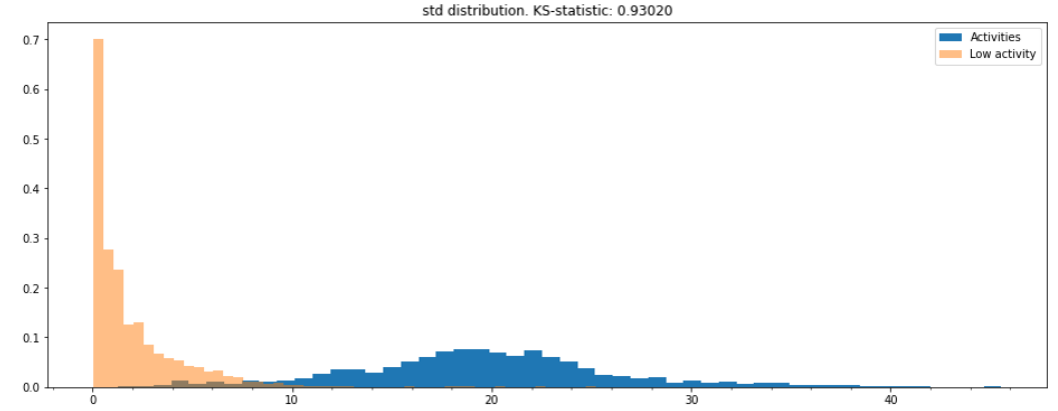
It is clear from the plots of the distributions that both categories (high and low activity windows) have significantly different values for the evaluated metrics. Specifically, low activity windows tend to have smaller values for each one of them in comparison to windows with high activities. Visually, the standard deviation appears to be the metric in which both distributions can be more easily separated by setting a threshold. In addition to that, the KS value for the standard deviation is also the largest, suggesting that both visually and analytically, that metric is the best option among the other four to discriminate the two categories.

The question now was about the threshold that should be used for the standard deviation so that the separation between low and high activity was as clean as possible. To answer this question, a number thresholds were used to classify a set of new unseen windows with high or low activities: if the standard deviation of the euclidean norm of the accelerometer signals was larger than a given threshold, the window would be classified as a high activity, and as a low activity otherwise. The F1 score was calculated for each one of the thresholds and the value where the F1 score was the largest was taken as the optimal threshold. This experiment is depicted in figure 5.19 and shows that the usage of a threshold of 8.5 gives the best F1 score of 96.67% (and accuracy of 96.56%), which is high enough to consider this binary classifier as good performing.

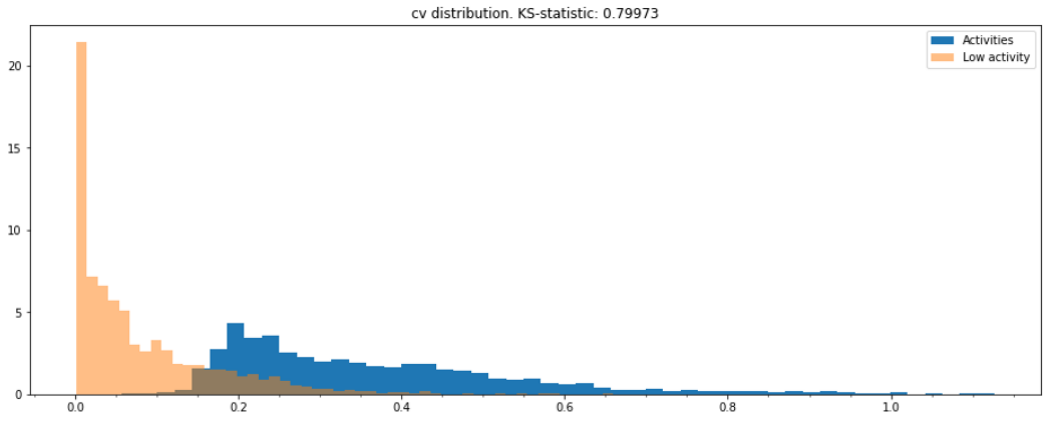
The binary high-or-low activity classifier was built, then, by defining a threshold of 8.5 on standard deviation of the euclidean norm of the accelerometer signals. For a given window, the euclidean norm of all the acceleration signals is calculated and the standard deviation of its values is obtained. 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 5 specific activities: pass, jog, jump, sprint or shot.



(a) Mean distributions

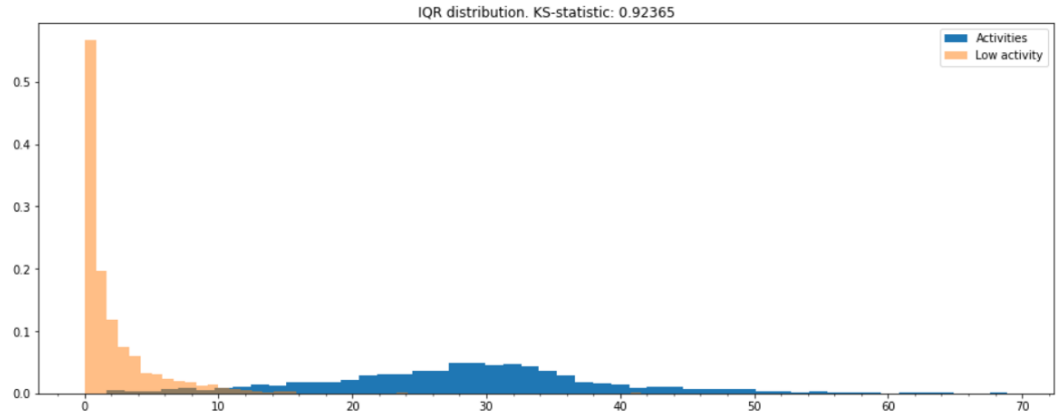


(b) Standard deviation distributions

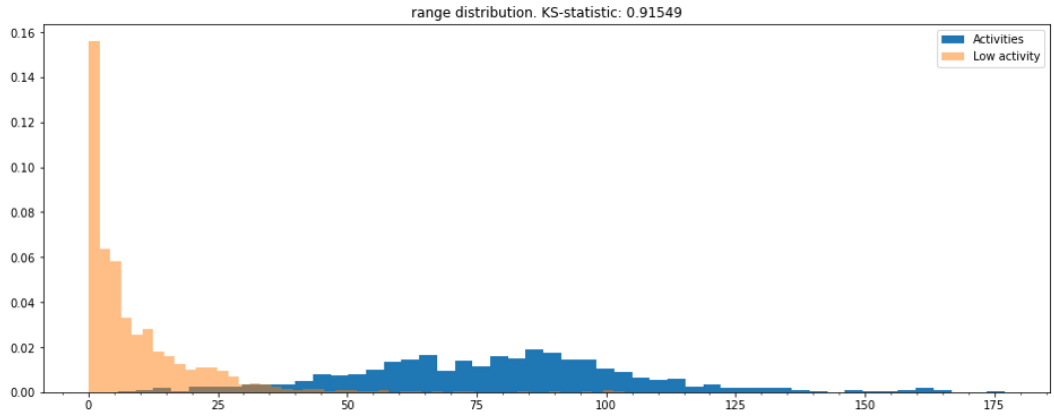


(c) CV distributions

Figure 5.18: Distributions of metrics of low and high activity periods



(d) IQR distributions



(e) Range distributions

Figure 5.18: Distributions of metrics of low and high activity periods (cont)

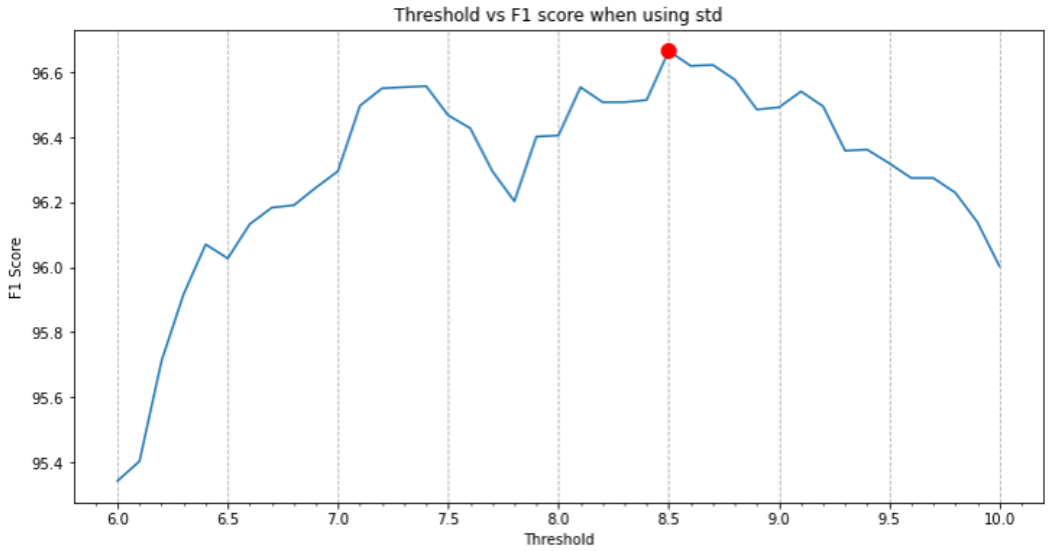


Figure 5.19: F1 scores for different standard deviation thresholds when classifying high and low activity windows

5.3. Sliding window evaluation

5.3.1. Prediction: activity recognition

Unlike what was done in the training phase, this new recording does not need to go through the activity detection process (section 4.3) to isolate the activities. If this input already complies with the correct sampling frequency and scale, as explained above, the raw signal can be fed to the model to generate the activity classifications. This can be done because a sliding windows approach is followed as shown in figure 5.20. A window of the same length as the one used for the training phase (1 second = 500 timesteps) is traversed through the recording, extracting at each time a window of such length. Each one of these windows is fed through the classifier, returning the recognized activity by the model for that interval. By sliding the window through the signals, it is possible to obtain activity classifications for each one of the timesteps of the recording. Furthermore, the overlap of these windows can also be set as a hyperparameter to fine tune the whole process even more.

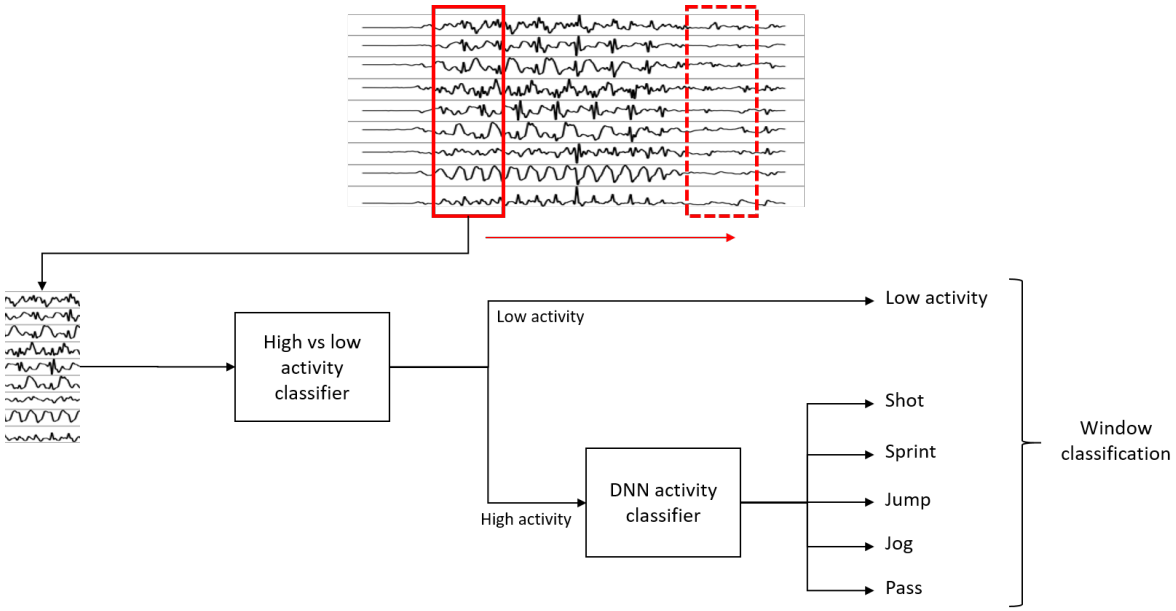


Figure 5.20: Evaluation phase diagram

The extracted window is first classified by the previously explained binary classifier as either a high or low activity window (using only accelerometer data). If the classification returns “low activity”, the window is understood as such. But if the binary classifier says that the window corresponds to a high activity, then the window is passed through the deep learning-based model that further classifies the interval as one of the activities: shot, sprint, jump, jog or pass. This means that, at the end of this process, the window is classified as either shot, sprint, jump, jog, pass or low activity. This is what we call a prediction. The sliding window then moves a certain amount of steps through the recording and a new window is extracted and then classified following the same process.

5.3.2. Postprocessing

After this process is finished and all the windows of the recording are classified in one of the possible activities, a subsequent phase of postprocessing is performed. This part is needed for two reasons. First, the predictions must be aligned to the recording; and second, it is needed to clean the predictions of undesired artifacts and misclassifications. Both reasons will be explained in detail below. For visualization purposes, on the following paragraphs, the images explaining the postprocessing phase will show each recognized activity with a different color.

Since the evaluation of a recording is made with a sliding window approach, it is necessary to choose the step that the sliding window takes when moving through the recording. Ideally the window would move one timestep at a time, however this would result in a very large number of windows and

the evaluation would take a long and unnecessary time. Instead of that, since some overlap between contiguous windows is desired to capture most of the information of the signal, a small step is taken between windows. There is a compromise to be made between choosing a very small step (and having more detailed predictions but with a large evaluation time) and a somehow larger step. We recommend to use steps between 10ms and 100ms (they translate to windows of 99% and 90% overlap respectively). It is important to note that, even if the training of the model was made with windows with 75% overlap, this value does not need to be the same as the overlap (step) chosen for the evaluation phase. The length of the window (1 second in our case), on the other hand, must be exactly the same to fit the model.

The sliding window approach with a certain step comes with the issue that there are less predictions that timesteps of the recording, since the windows are not evaluated for each time point. See figure 5.21 where this is depicted. On that figure, each vertical colored line corresponds to a location of the sliding window, therefore to a prediction. Each color represents a recognized activity. If we suppose that the whole recording has length N, since the sliding window generates a prediction every certain number of steps larger than 1, then the evaluation generates n predictions. The issue, as it can be seen in the figure, is that $n < N$ ($n \leq N$ if the step is 1), so there are less predictions than timesteps and postprocessing is needed to associate an activity to each moment of the recording, hence aligning the predictions.

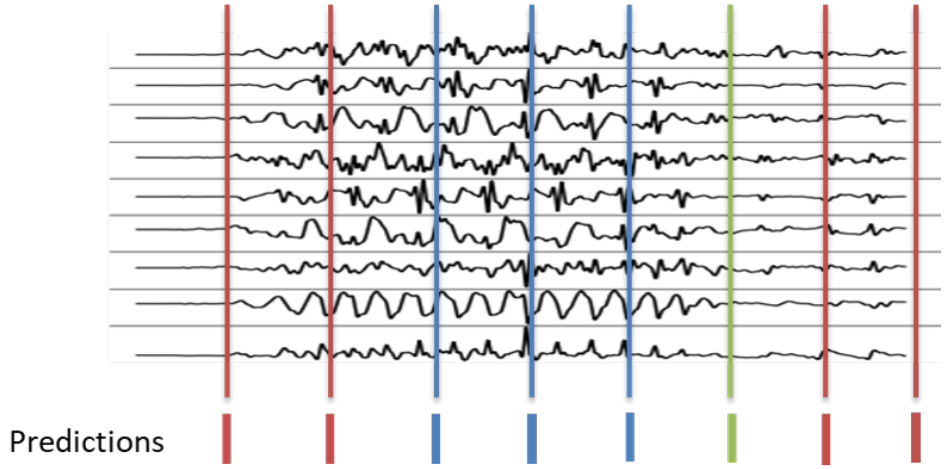


Figure 5.21: Predictions are not aligned when using a step larger than 1

Three different options were designed, implemented and tested to solve the previous problem:

- Interpolation postprocessing.

  It is defined as: *All the timesteps between the end of window i and window $i+1$ are assigned to the prediction of window $i+1$.*

  Figure 5.22 explains this process in a graphical, simplified way with a toy example. Suppose that the horizontal multicolored bar on the top of the image represents the recording. That recording is traversed with a sliding window and, at each position of the window, a prediction of the activity there present is made. Those windows are depicted in the figure as rectangles with thick borders and are named $W1, W2, W3, \ldots$. The prediction made by each window is represented by a color: blue, red or green. Then, the top, horizontal bar shows the predictions made by the sliding windows using that color code. That bar is the output of the evaluation pipeline explained in figure 5.20.

  The interpolation postprocessing option assigns the prediction of the window to the last timestep of such window, as it can be seen with the small vertical colored bars in the middle of the figure.

Then, as it was mentioned, it assigns that prediction to all the timesteps between the end of that window and the end of the previous one.

The bottom, horizontal bar in the figure shows the resulting predictions for the toy example after this interpolation postprocessing.
By taking the last timestep of the window as its prediction, the resulting postprocessed recognitions will be slightly shifted towards the right. If this was done instead with the initial timestep of the window, the final result would be shifted to the left.

This postprocessing option is very intuitive and fast and easy to compute, however it does not "clean" undesired and isolated predictions, as it will be shown later.

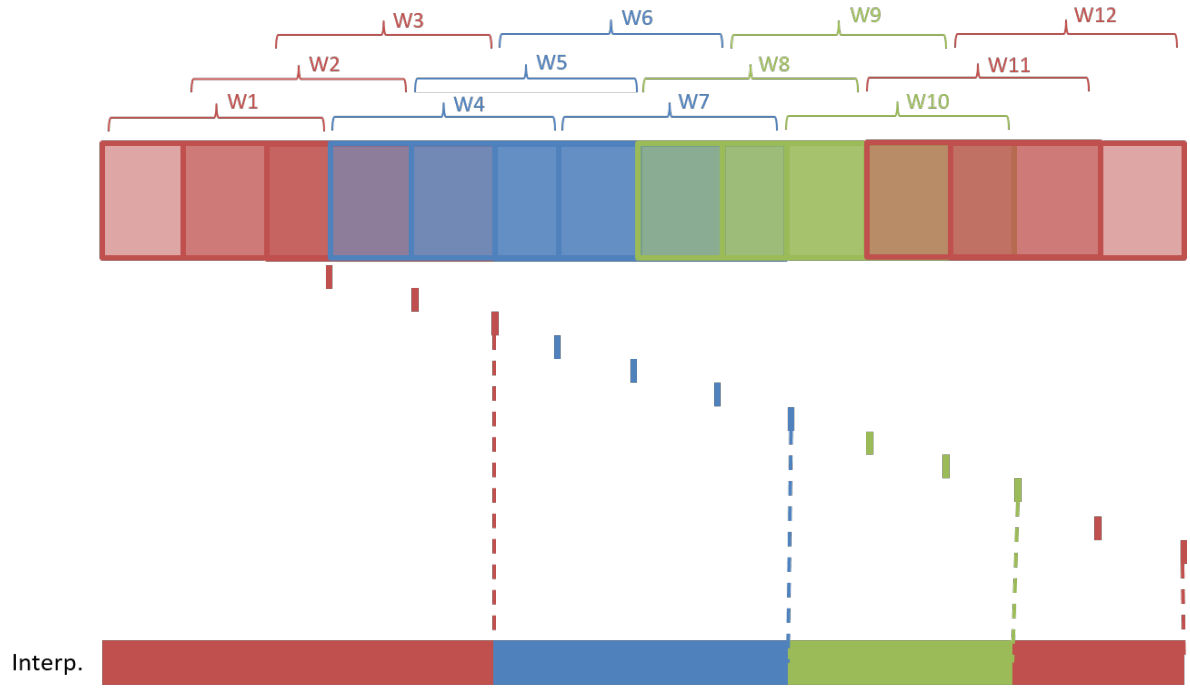


Figure 5.22: Interpolate postprocessing option

- Mode postprocessing

  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 mode of the predictions of all the windows that contained that timestep.*

  Figure 5.23 explains this process in a graphical, simplified way with a toy example. Suppose that the horizontal multicolored bar on the top of the image represents the recording. That recording is traversed with a sliding window and, at each position of the window, a prediction of the activity there present is made. Those windows are depicted in the figure as rectangles with thick borders and are named $W1, W2, W3, ...$. The prediction made by each window is represented by a color: blue, red or green. Then, the top, horizontal bar shows the predictions made by the sliding windows using that color code. That bar is the output of the evaluation pipeline explained in figure 5.20.

  The mode postprocessing option assigns the prediction of the window to all the timesteps contained in that window, as it can be seen with the small horizontal colored bars in the middle of the figure. Then, as it was mentioned, for each timestep of the recording, the most common predic-

tion (mode) is taken as the final prediction of the respective timestep.

The bottom, horizontal bar in the figure shows the resulting predictions for the toy example after this mode postprocessing. Unlike the interpolation postprocessing option, the result does not get shifted to the right or left.

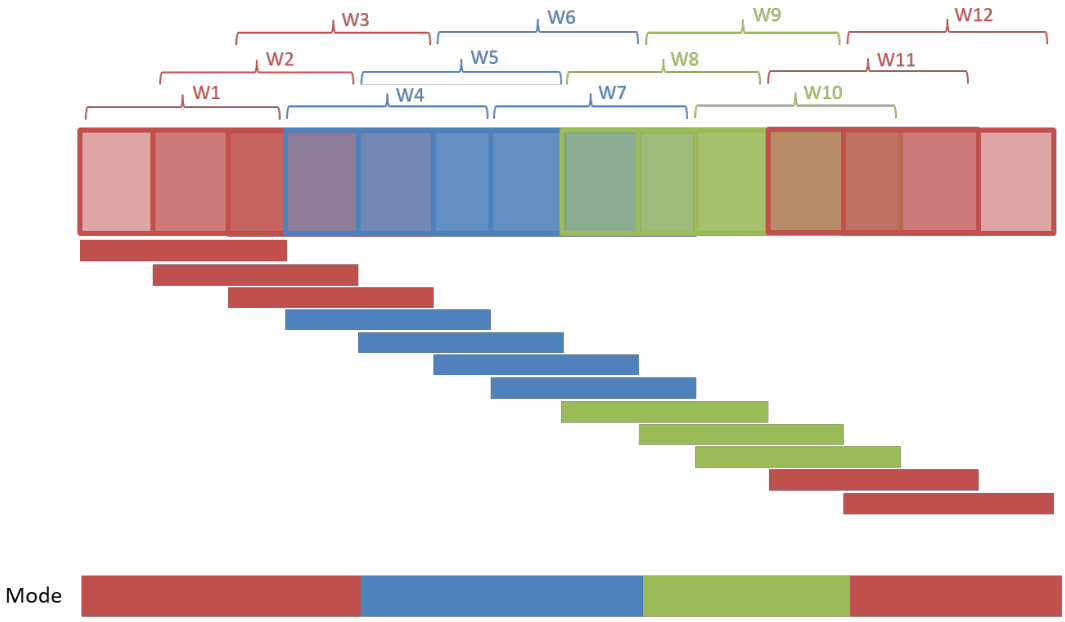


Figure 5.23: Mode postprocessing option

This postprocessing option gives more importance to consecutive predictions of the same activity, so it has the additional property of "cleaning" the results of short undesired and isolated predictions. Since the step of the sliding window is very small, and the recordings take several seconds or minutes with a sampling rate of 500 Hz, it is needed to store several predictions and then calculate the mode for each timestep. This can be memory-wise very expensive.

Initially, the approach to perform this procedure was defined as the one presented in figure 5.24: a large matrix was used to store all the predictions and then the mode for each column was taken. This resulted in a very large (and sparse) matrix that translated into a slow postprocessing phase that, for recordings of more than a couple minutes, overwhelmed the computer memory. It was not scalable and not efficient.

Because of that, an improved way to perform this mode postprocessing was designed, as shown in figure 5.25. With this approach, instead of building a large sparse matrix, a smaller matrix was built that stored the number of predictions of each class for each timestep. Then, by taking the argmax over each column, the final prediction of the respective timestep can be obtained. This is equivalent as taking the mode of the predictions. By performing the mode postprocessing option like this, not only the matrix that is used is much smaller in terms of amount of rows (the number of possible activities is much smaller than the number of windows), but also the subsequent operation (argmax or mode) over the matrix's columns is much faster, since there are less numbers per column. Additionally, the function to calculate the argmax of an array in Python (`numpy.argmax()`) is much faster (more than 1000 times faster according to our experiments) than the function to calculate the mode of an array (`scipy.stats.mode()`). This approach makes the mode postprocessing option fast, scalable and memory-friendly.

Table 5.4 shows the comparison in the size of the matrices needed to perform the mode post-

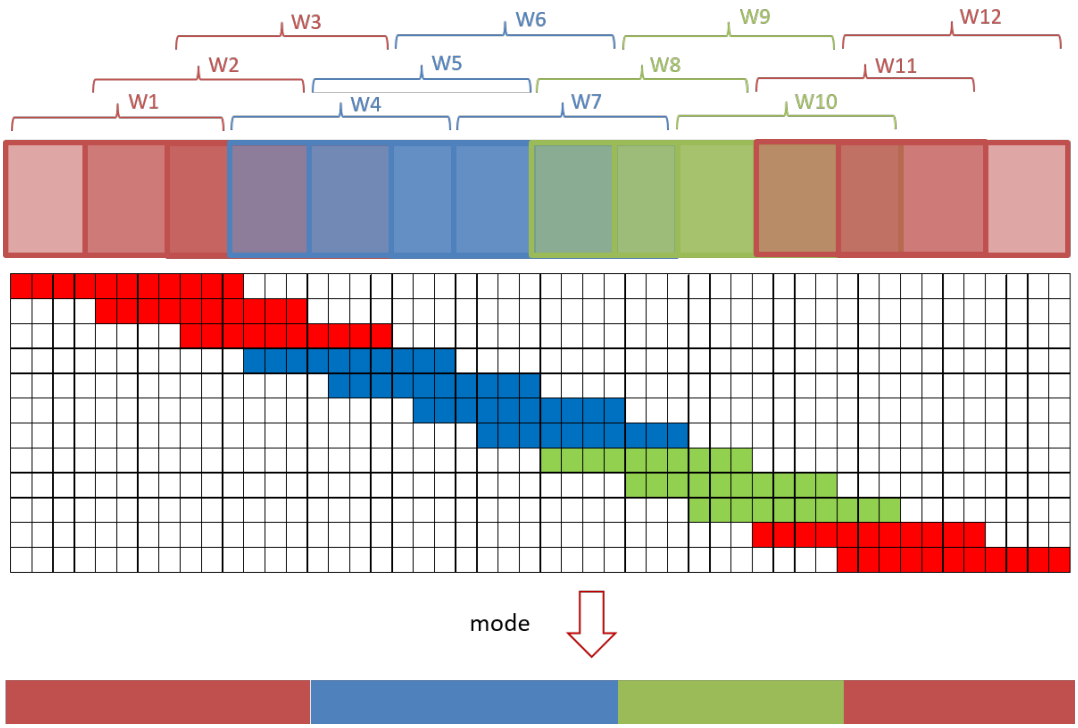


Figure 5.24: Initial approach to perform the mode postprocessing option

processing option with the two previously described approaches. In that table, l is the length of the recording in seconds, f is the sampling frequency in Hz, $classes$ is the number of possible activities to detect, and $step$ is the step of the sliding window in seconds. Since the sliding window step is much smaller than 1, then it is clear that the number of elements of the matrix using the improved approach is much smaller than the one when using the initial approach:

Since $step \ll 1$, then

$$l \cdot f \cdot classes \ll f \frac{l^2}{step}$$

For the initial approach, the matrix size will increase proportional to the square of the length of the recording and inversely proportional to the step of the sliding window. On the other hand, for the improved version, the matrix will increase proportional to just the length of the recording. In fact, the improved version will have a matrix $\frac{l}{step \cdot classes}$ smaller than the matrix of the initial approach. This is a huge difference knowing that $l \gg 1$ and $step \ll 1$.

For example, a 100 second long recording, using a sliding window step of 50 ms and 6 classes to predict (shot, sprint, jump, jog, pass, and low activity) would require a matrix with approximately 100 million elements for the initial approach. For the improved approach, the matrix would have 300 thousand elements. This is a difference of more than 300x. For a 30 minute long recording with the same 50 ms timestep the difference is now 6000x (the initial approach would have a matrix with around 32 billion elements).

Table 5.4: Comparison of matrix sizes for both approaches when performing the mode postprocessing option. l is the length of the recording in seconds, f is the sampling frequency in Hz, $classes$ is the number of possible activities to detect, and $step$ is the step of the sliding window in seconds.

Mode postprocessing approach	**Matrix size**	**Number of elements in matrix**
Initial approach (figure 5.24)	$\approx \frac{l}{step} \times (l \cdot f)$	$\approx f \frac{l^2}{step}$
Improved approach (figure 5.25)	$\approx classes \times (l \cdot f)$	$\approx l \cdot f \cdot classes$

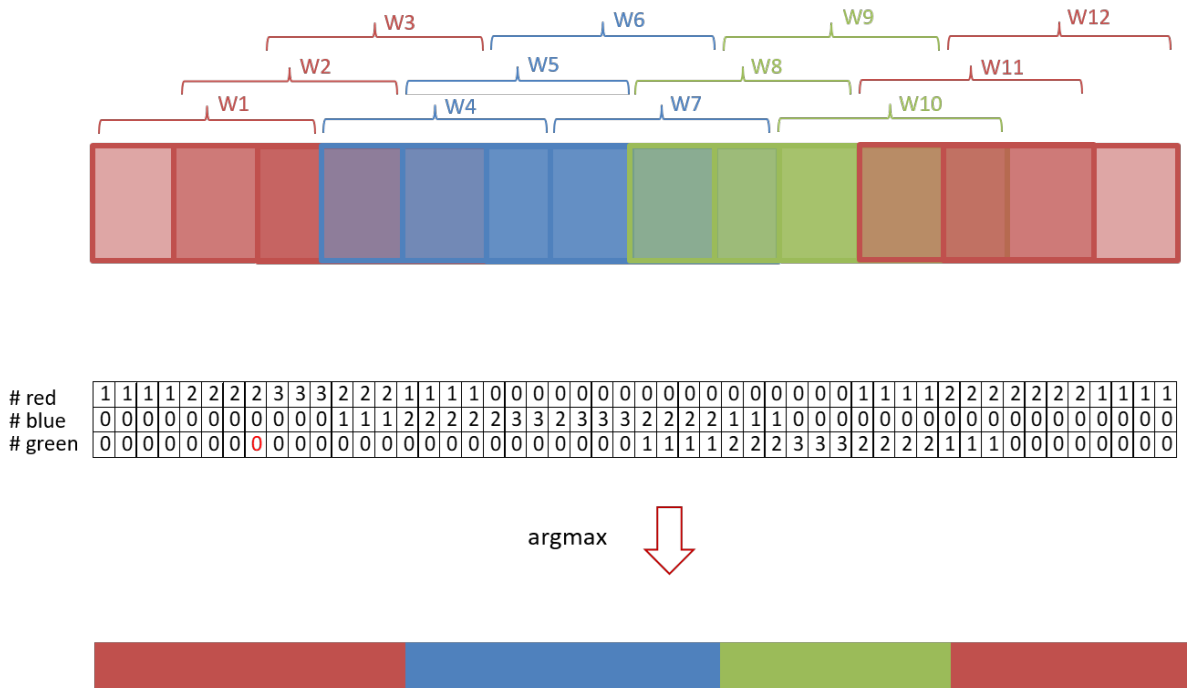


Figure 5.25: Improved approach to perform the mode postprocessing option

- Best score postprocessing

  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.*

  Figure 5.26 explains this process in a graphical, simplified way with a toy example. Suppose that the horizontal multicolored bar on the top of the image represents the recording. That recording is traversed with a sliding window and, at each position of the window, a prediction of the activity there present is made. Those windows are depicted in the figure as rectangles with thick borders and are named $W1, W2, W3, ...$ The prediction made by each window is represented by a color: blue, red or green. Then, the top, horizontal bar shows the predictions made by the sliding windows using that color code. That bar is the output of the evaluation pipeline explained in figure 5.20.

  The best score postprocessing option assigns the prediction of the window to all the timesteps contained in that window, as it can be seen with the small horizontal colored bars in the middle of the figure. Each one of these predictions are composed of the recognized activity (red, blue or green in the figure) and the confidence of the prediction (light to dark tone of the color). Then, as it was mentioned, for each timestep of the recording, the prediction with largest confidence is taken as the final prediction of the respective timestep.

  The bottom, horizontal bar in the figure shows the resulting predictions for the toy example after this best score postprocessing. Unlike the interpolation postprocessing option, the result does not get shifted to the right or left.

  The implementation of this postprocessing option was made in a similar way as the improved mode postprocessing approach (see figure 5.25). Instead of registering the number of windows that predicted a certain timestep as each one of the possible activities, the maximum confidence for each activity among all the windows predicting that activity for the timestep was captured in

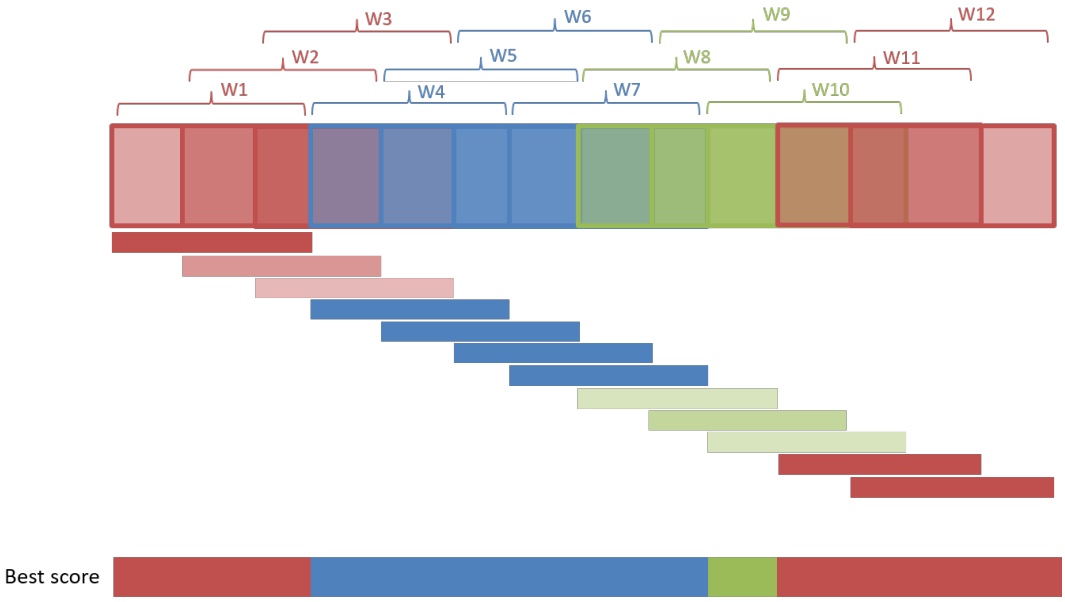


Figure 5.26: Best score postprocessing option. The darker the color, the larger the confidence of the prediction.

the matrix. Then, the argmax of those confidences are taken to obtain the final prediction. This makes this postprocessing option fast, scalable and memory-friendly as well.

This postprocessing option gives more importance to predictions with high confidences. If, for instance, a prediction of a shot with 99% confidence is surrounded by several predictions of jogs with 40% confidence, it would make more sense to trust the prediction of the timestep being a shot, since the score is very high. This is what this postprocessing option achieves. It also has the additional property of "cleaning" the results of short, isolated, low score predictions surrounded by predictions with a larger confidence.

In order to calculate such confidences or scores, the softmax activation function of the last layer of the neural network model was used. Since the problem is a multiclass classification task, the last layer of the model has a softmax function. This activation function at the output of the model represents the probability distribution over the possible categories (see figure 5.27) . The softmax activation function is defined as:

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

Where i is the i-th activity among all the K possible activities to detect, and $\vec{z}$ is the output of the last layer of the neural network.

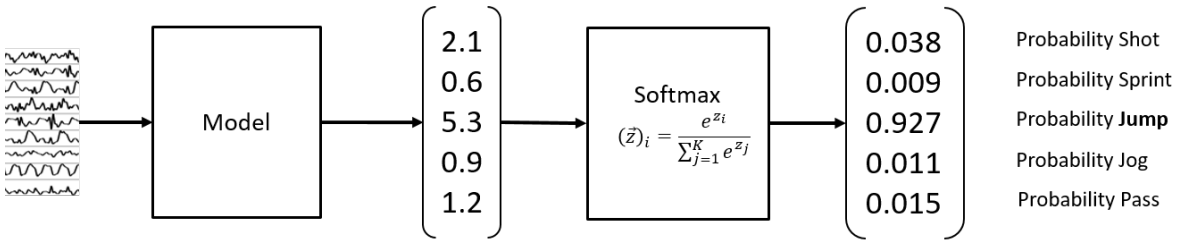


Figure 5.27: Softmax activation function to obtain prediction confidences/scores

These outputs of the softmax activation function of the last layer were used to obtain the confidences/scores of the predictions of the window. Furthermore, the prediction of the window was

defined as the activity with the largest confidence. As for “low activity” windows (since the binary classifier was built based on a strong threshold to differentiate between high and low activities) their confidences were set to $c_l = 98\%$, so that they had very high score, but not enough to beat high activity predictions with even larger scores. This confidence of 98% was also set as a hyperparameter c_l, so it can be manually modified if it is desired.

This best score postprocessing logic is analogous to a common postprocessing technique widely used on computer vision when working with object detection applications: the Non-Max Suppression (or NMS) algorithm, as seen in figure 5.28. This algorithm is used to obtain a single bounding box for each detected object in the image. After an object detection algorithm is run on an image, that algorithm returns a set of bounding boxes of the objects there present with their confidence score. Then, the NMS algorithm works as following:

1. Select the bounding box A with the highest confidence score.
2. Compare bounding box A with all the other bounding boxes using a metric called Intersection Over Union (IoU).
3. Remove all the bounding boxes that have IoU with bounding box A larger than a certain threshold.
4. The next bounding box A is selected and the process is repeated until there are no more proposed bounding boxes.
5. Final bounding boxes

This thesis is clearly not focused on computer vision or object detection, so the NMS algorithm will not be further explained or explored. However, it is interesting to note that, for the task of human activity recognition using the sliding window approach, the proposed best score postprocessing variant fulfills a function equivalent to what the NMS algorithm does for object detection problems. They are both used to obtain a final prediction with high confidence among other low confidence predictions in the surrounding.

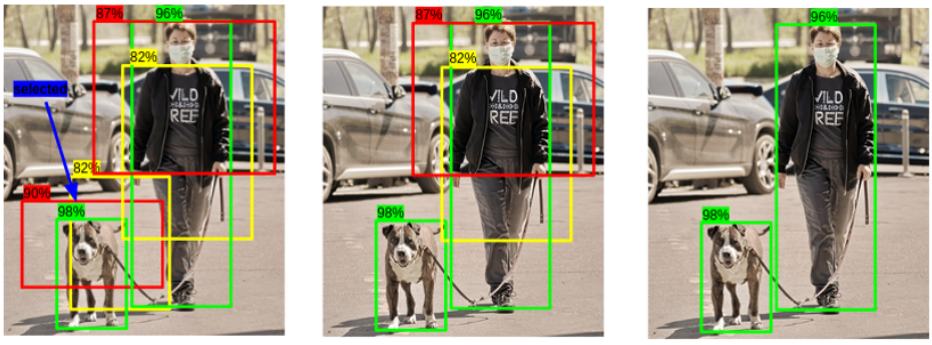


Figure 5.28: Non-Max Suppression. Taken from (K, 2019)

Among the three proposed postprocessing options, it is recommended to use the best score variant, since it gives more importance to predictions with high confidence, obtaining final predictions with an overall better prediction confidence. Additionally, as explained, this type of postprocessing also cleans the predictions from isolated and low confidence activities.

5.3.3. Outlier removal and “other high activity” recognition

After postprocessing, an outlier removal procedure is applied to the predictions. Even if the mode or best score postprocessing option is used and the predictions are cleaned, as explained, there is still a chance that short peaks of isolated activities survive. It is clear that isolated activities lasting less

than a couple of miliseconds do not happen in real life. That is why, if there is a detection of an activity that lasts less than a certain number of miliseconds, that prediction is considered an outlier and has to be cleaned out. To do so, a rule was implemented: if a prediction of an activity lasts less than τ miliseconds, the predicted activity of that interval is replaced with the next predicted activity that lasts at least τ miliseconds. For this thesis, good results were obtained when using values of τ between 100 and 300 miliseconds.

The whole sliding evaluation process can be summarized with the diagram shown in figure 5.29, where each position of the sliding window is classified, then the prediction is postprocessed and finally the outliers are removed. The block called “Activity Classifier” in the diagram is the process of activity classification, depicted previously in figure 5.20.

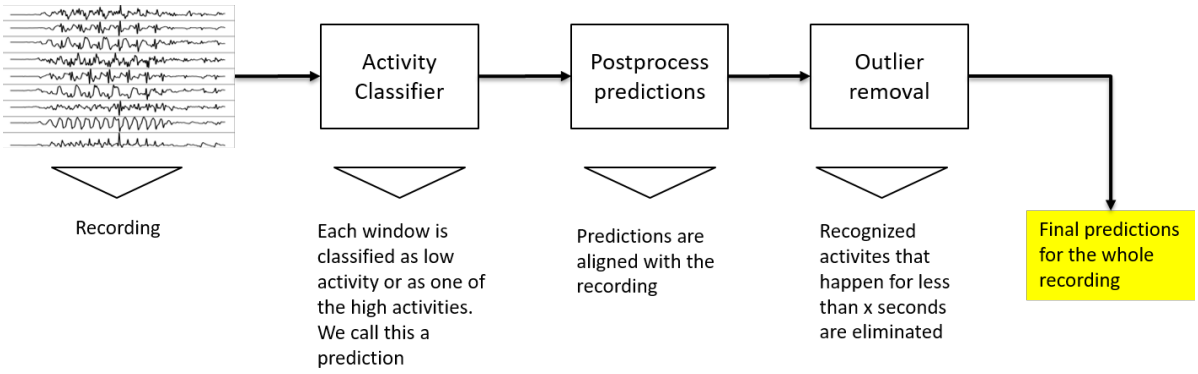


Figure 5.29: Sliding window evaluation procedure

The softmax activation function of the last layer was not only used to build the best score postprocessing option. By knowing the confidence of each one of the recognized activities made by the windows, it was also possible to visualize these scores during the evaluation phase by defining a color gradient. Knowing that the sum of the outputs of the softmax always sum up to 1, to be selected among the others, a predicted activity must have a confidence larger than 1 over the number of possible activities. The maximum confidence is clearly 1. Because of this, the color gradient was defined as follows:

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

This means that a prediction shown in yellow would have a very low confidence, but if it is shown in black it would have a very large score, so we could rely more on it. At the end, what the max score postprocessing does is remove low-confidence (yellow) predictions.

Furthermore, this yellow-black color code of the predictions was not only used for visualization purposes, but also to allow the model to recognize an additional type of activity called “other high activity”. The model was built and trained to recognize specifically 5 types of high activities: shot, sprint, jump, jog and pass. However, football is a complex sport, in which players perform more activities than those five. Turns, for instance. In order to allow the model to somehow recognize these additional high activities, the prediction confidence was used. A rule was set that defined this new “other high activity” class as a prediction of any of the five predefined activities with a confidence lower than a threshold h_h. In other words, if a prediction made by the model has a confidence score of less than h_h, the window prediction would be considered as “other high activity” instead of the recognized activity by the model. This logic was built before the postprocessing phase, so that “other high activities” would also be postprocessed with the other activities just like any of them. Since the best score postprocessing gives more importance to predictions with large confidence, and these “other high activities” have, by definition, low confidences, they would be always eliminated from the final predictions, making the whole process pointless. In order to avoid this, after a prediction is transformed into an “other high

activity", just like it is done with low activities, its confidence c_h is set to a large number smaller than 100% (between 95% and 98% gave good results in the experiments). The threshold h_h is a hyper-paremeter that can be modified if desired. The larger that value is, the easier (or less demanding) it is to recognize "other high activities". For the results presented below, a value of 50% was used. If it is not desired to have this additional class of "other high activity", the threshold h_h can be simply set to 0.

It is very important to understand, however, that the built deep learning based model is not trained to recognize these "other high activities", since it was optimized to recognize specifically shots, sprints, jumps, jogs and passes. The inclusion of this part to translate low confidence predictions into "other high activities" is not optimal, because the model will always try to understand each window as one of the 5 mentioned activities. The model is not designed to recognize "other high activities", so if additional activities are meant to be robustly recognized, it is recommended to retrain the model to specifically detect those new classes.

The addition of this part to recognize other high activities modifies slightly the pipeline of figure 5.29 into the one shown in figure 5.30, which is the final pipeline of the complete model in a summarized form.

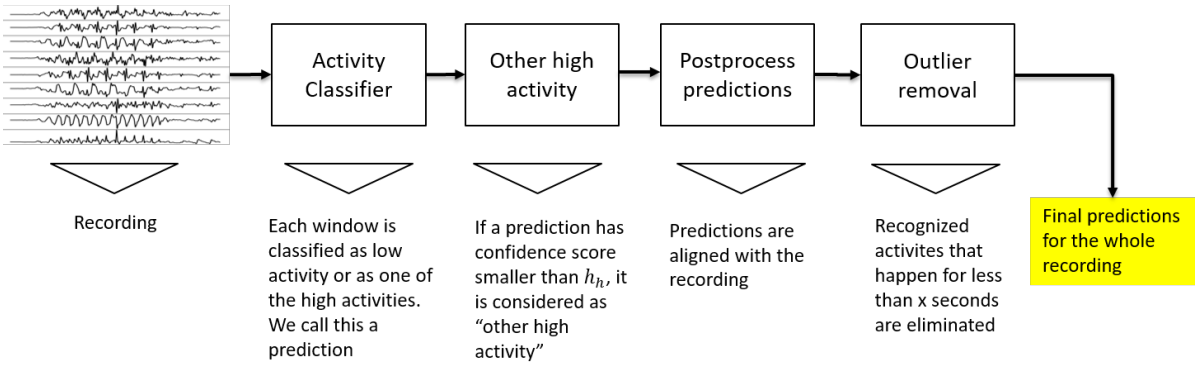


Figure 5.30: Final sliding window evaluation procedure with inclusion of "other high activity" class

5.4. Results

In this section, some results obtained with the previously explained activity recognition pipeline will be presented. Unless stated otherwise, the figures in this section where the final predictions are shown, will be composed of four graphs. The one on the top will be called the *predictions* and are the predictions obtained by the model (outputs of figure 5.16) with the yellow-black confidence colorcode. The second one will be called the *postprocessed predictions* and is the result of postprocessing the predictions. The third graph will be called the *final predictions* and is the final predictions after the postprocessed predictions are passed through the outlier removal process (outputs of figure 5.30). Finally, the bottom graph is just for reference and contains only 3 of all the sensor signals of the original recording. The horizontal axis (time) is shared among all the 4 graphs. For the top 3 graphs, the vertical axis corresponds to the predictions made by the model.

Initially, a comparison of the results of the three postprocessing options will be shown. It will be possible to see the differences in the final predictions when each one of these three variants are used. In figures 5.31, 5.32, and 5.33 an example of the results of the whole pipeline using interpolate, mode and best score postprocessing is presented. The input recording consists of three consecutive activities: sprint, pass and shot, in this case with low activity periods in between them.

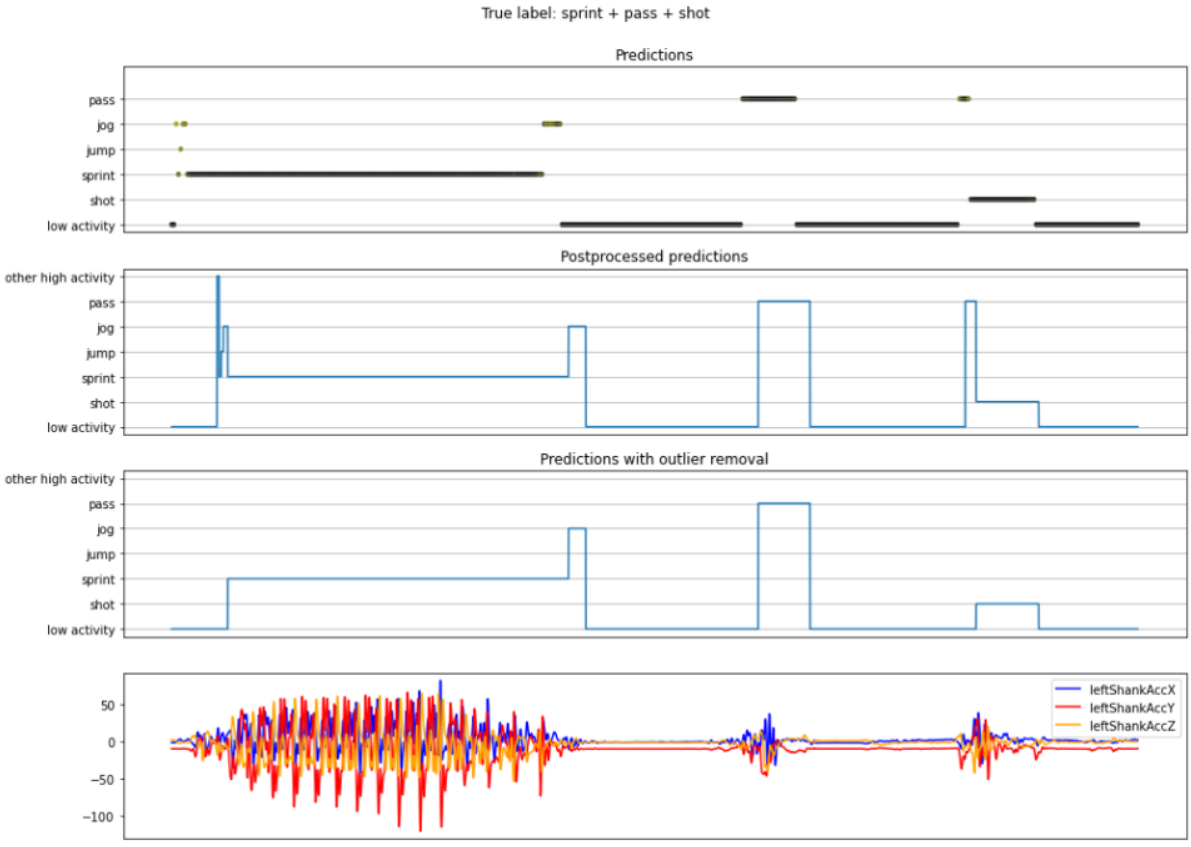


Figure 5.31: Example of results using interpolate postprocessing

In the bottom plot of the three figures there are the measurements obtained by 3 of the sensors (left shank axis X, Y, and Z in this case). The top plot is the same in the three cases, since it corresponds to the predictions without any postprocessing. It can be seen that the model performs really well in recognizing the activities present in the recording. There are some incorrect predictions, but they happen mainly at the start and the end of the actual movements. This is, in fact, expected, due to the fact that the sliding windows at those points cannot capture enough information of the real movement to recognize it correctly. More interestingly, the incorrect predictions show low confidences: they appear with a yellower tone than the correct predictions which appear during the activities are made. The

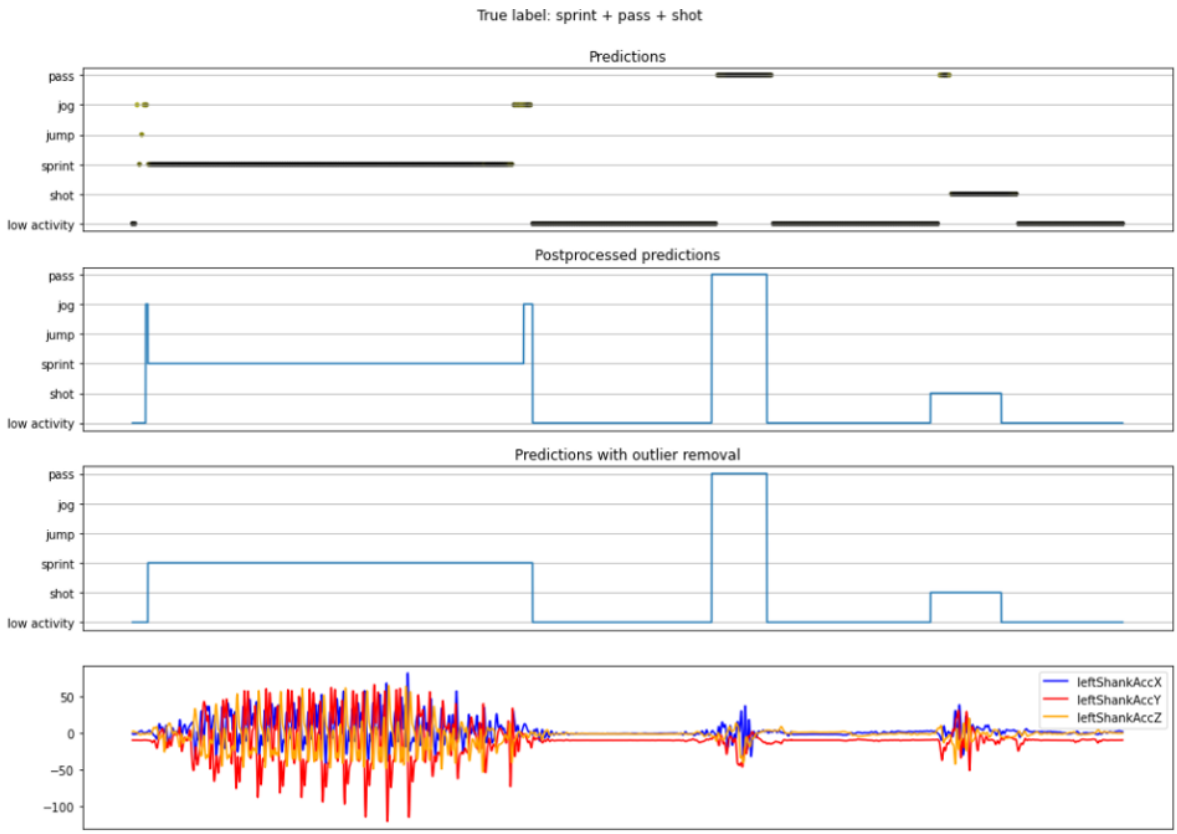


Figure 5.32: Example of results using mode postprocessing

model is trained to recognize one activity at a time and not transitions between them, so it is natural and expected that the model will give low score predictions (and sometimes even incorrect ones) for those moments where an activity transitions to another, as it is seen.

As it was explained, the interpolate postprocessing option does not clean the predictions en any matter. This is evidenced in the short peaks that appear in the second plot of figure 5.31. Any prediction made by the model will appear after this type of postprocessing. It possible to see this by looking at the yellow predictions at the beginning of the recording in the top graph of the figure. Those are isolated and low-confidence predictions and they appear as short peaks in the second plot after the interpolate postprocessing is made. Note that even some of the predictions have such low scores, that they are transformed into “other high activities”. However, the outlier removal process eliminates those short artifacts, as it can be seen in the third plot of the figure. The outlier removal process is crucial when using the interpolate postprocessing option to remove undesired artifacts.

In the previous paragraph it was mentioned and evidenced that short, isolated predictions survive the interpolate postprocessing scenario. The mode postprocessing option takes this into account and removes most of them them without the need of the outlier removal phase. In figure 5.32 we see that the most part of those short peaks do not appear after the predictions are postprocessed with the mode option. This is why we say that the mode postprocessing option cleans the predictions of short-lived predictions in between predictions of another class. The outlier removal process is still applied after the postprocessing to further remove short predictions that persist, as it can be seen in the third plot of the figure.

In the top plot of figures 5.31, 5.32, and 5.33 it is seen that there are indeed predictions with low confidences (shown in yellow). The interpolate postprocessing option does not eliminate them (second plot of figure 5.31) and the mode postprocessing variant preserves them if they happen for some timesteps consecutively (second plot of figure 5.32). However, if those short-lived low-confidence pre-

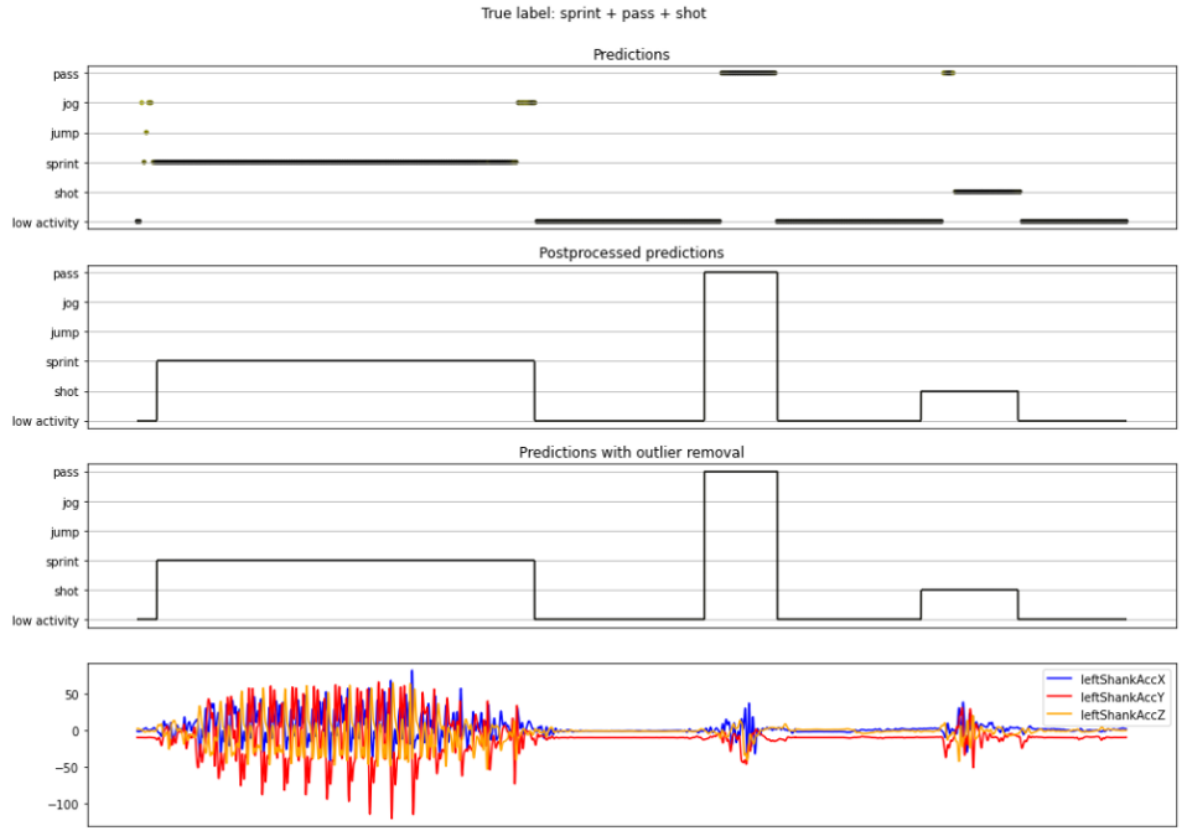


Figure 5.33: Example of results using best score postprocessing

dictions happen between high-confidence predictions, they should be removed and more importance should be given to predictions with a larger score. This is what the best score postprocessing does, as seen in the second plot of figure 5.33), where the short, low-confidence predictions are completely substituted by ones with larger score. This means that those activities for which the model is most confident in its prediction persist. This highly reduces the occurrence of wrongly detected activities, that tend to have low scores. It is safe to say, then, that the best score postprocessing option, cleans the predictions made by the model even more. Similar to what is done with the mode case, the outlier removal is still used after the best score postprocessing to further clean out predictions that survive and last just a few miliseconds. In many cases the outlier removal will not be necessary, but in other cases it will be useful, as seen in the predictions of the sprint in figure 5.34. There is a very short prediction of a jump that “survived” the postprocessing but is cleaned out for the final result. Interestingly, in this example, the model even recognizes the end of a sprint as a jog (which makes sense, because the player slows down before stopping). An additional advantage of using the best score postprocessing option is that, by using it, we can give the confidence scores for the final predictions with the same yellow-black color code. For figures 5.33 and 5.34, the final predictions are not only correct in terms of the recognized activities, but they are drawn in black, showing additionally that the pipeline has an overall large confidence on the results. For the interpolate and mode postprocessing examples, the predictions are shown in blue, meaning that they do not include their confidence.

The postprocessing examples presented in figures 5.31, 5.32, and 5.33 are compared one below the other in figure 5.35. There, the second, third and fourth plots (from top to bottom) show the results of the predictions after being postprocessed by the interpolate, mode, and best score options respectively, but before outlier removal. In this figure, it is possible to see that the interpolate option does not clean the predictions, but the mode and best score ones do so (the latter more extensively). The first variant also generates predictions that are slightly shifted to the right, as explained in section 5.3.2. As for the best score postprocessing, there are two things that are important to not: first, that the final predictions are drawn with the yellow-black color code, which gives also information about the confidence

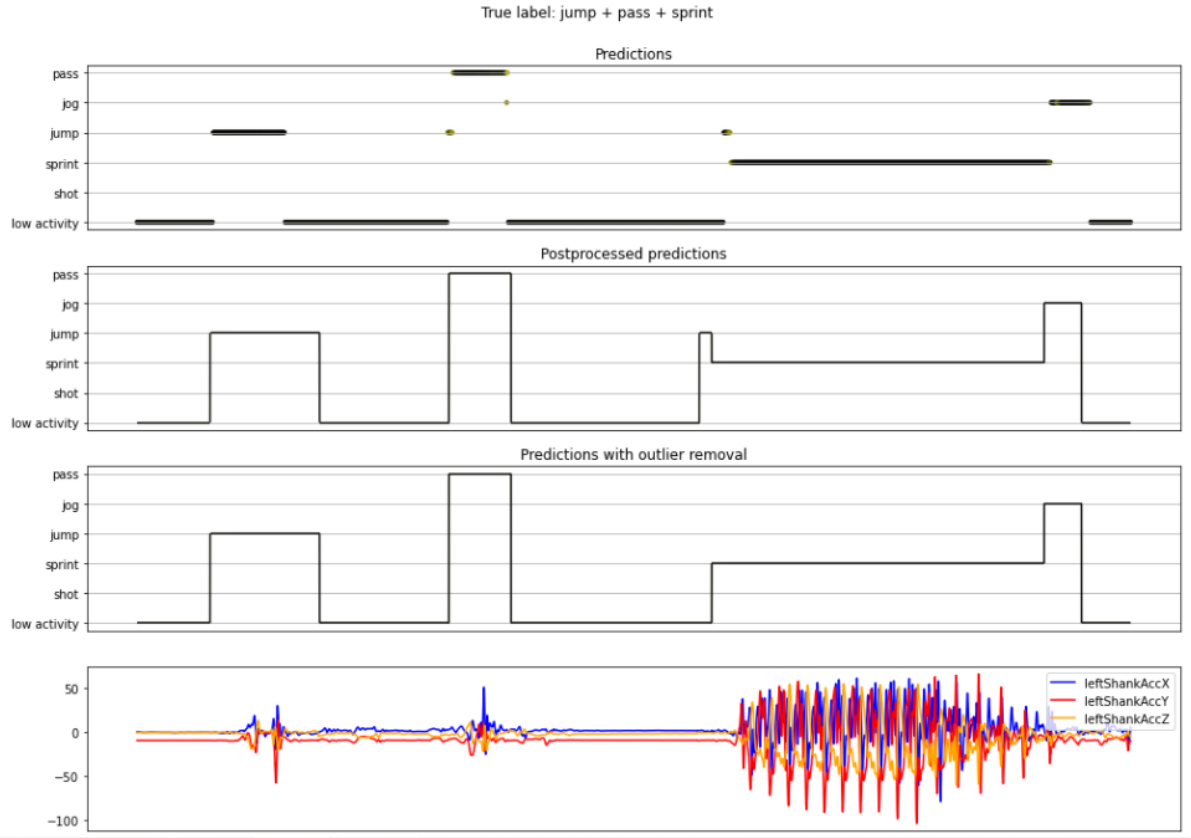


Figure 5.34: Example of outlier removal process after best score postprocessing

of these final results; and second, that the predictions of high activities tend to be slightly wider. This happens because of the choice of confidence score given to low activity windows. For this example, as recommended, they were given 98% of confidence, meaning that high activities with large scores take over. If a larger confidence c_l is chosen for the low activities, those high activity predictions will be narrower.

Some additional examples of predictions made by the whole pipeline of 5 consecutive activities are shown in figures 5.36, 5.37, and 5.38. For these examples, the best score postprocessing was applied. All of them show a very good performance of the model, both in terms of recognizing the correct movement and in doing such recognition with a large confidence. As it can be seen, the model is not perfect, and some “incorrect” classifications are made in some cases, specially at the beginning or end of individual movements. These situations are due to the nature of the sport, since one activity can look like another at its early or late phases: jogging before or after sprinting, for instance (see figure 5.38), or jumping after a shot is made due to the inertia of the movement. Because of this, we do not consider those cases as erroneous classifications, but as even more accurate ones.

The proposed pipeline can also be used to obtain, in text format, the sequence and duration of the activities performed in the recording. As an example of this, for the recording of figure 5.37, the summarized text with the recognized activities, their duration and count, are depicted in figure 5.39.

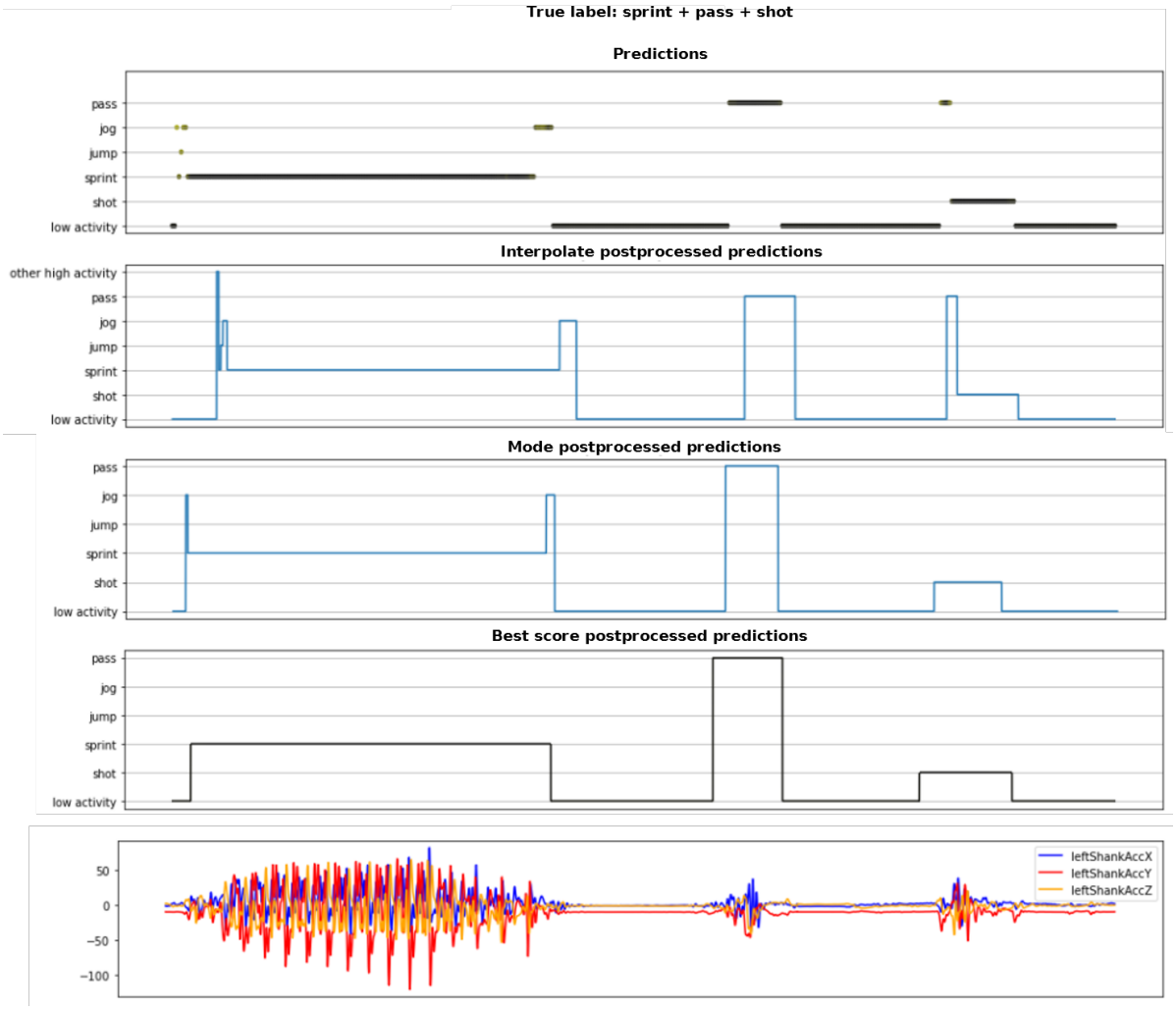


Figure 5.35: Comparison of postprocessing options

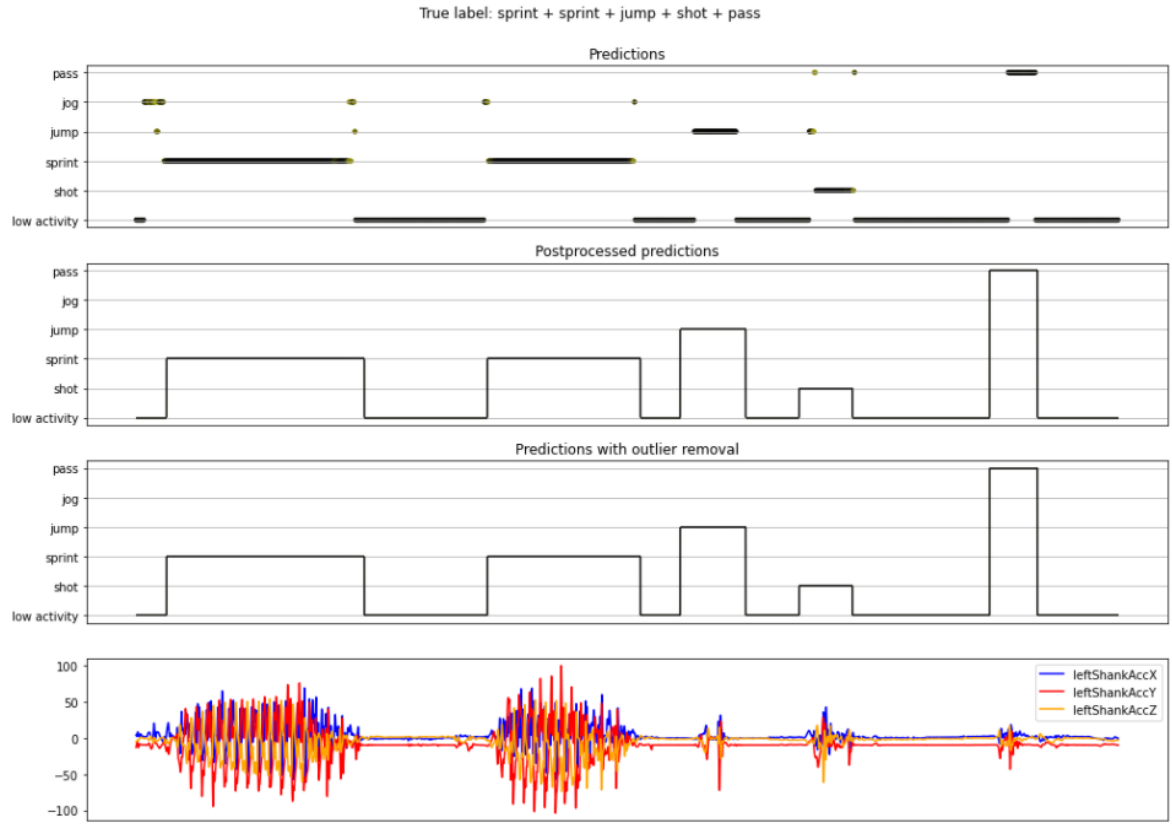


Figure 5.36: Example 1 of final results. True labels: sprint, sprint, jump, shot, and pass with low activity periods in between.

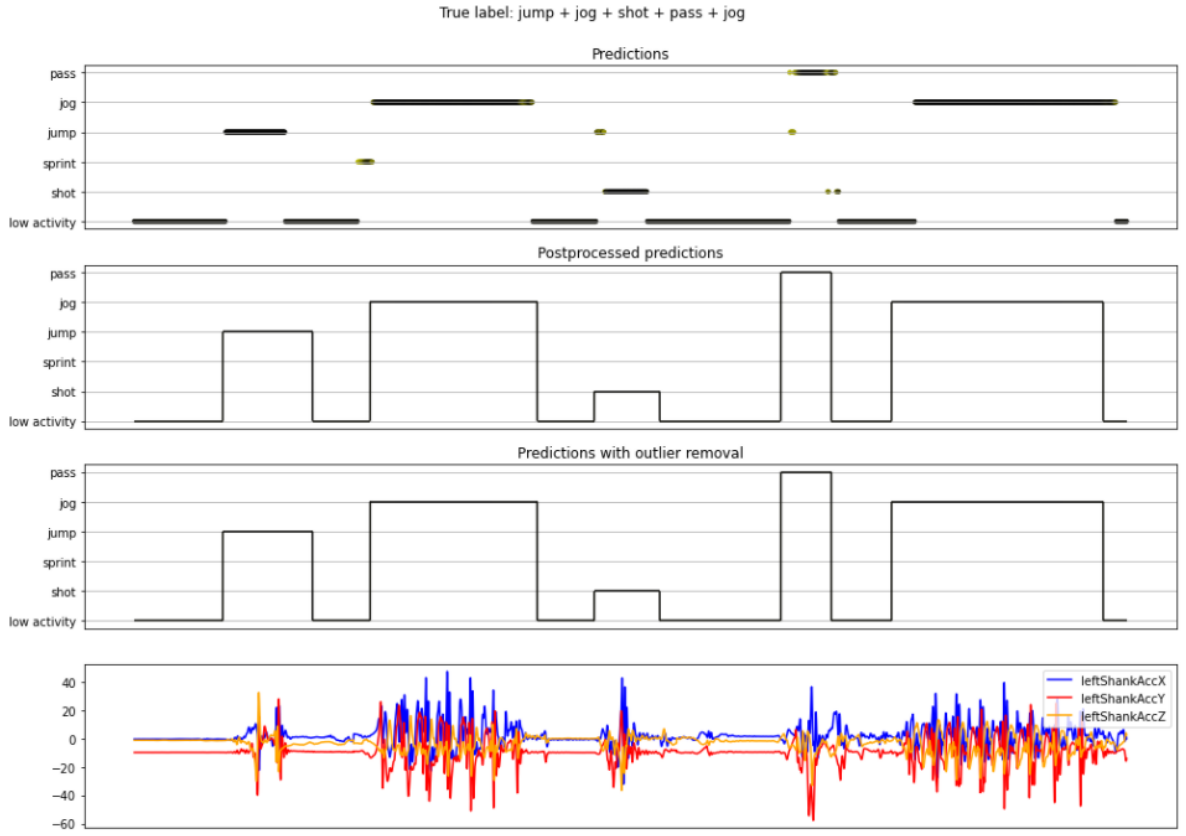


Figure 5.37: Example 2 of final results. True labels: jump, jog, shot, pass, and jog with low activity periods in between.

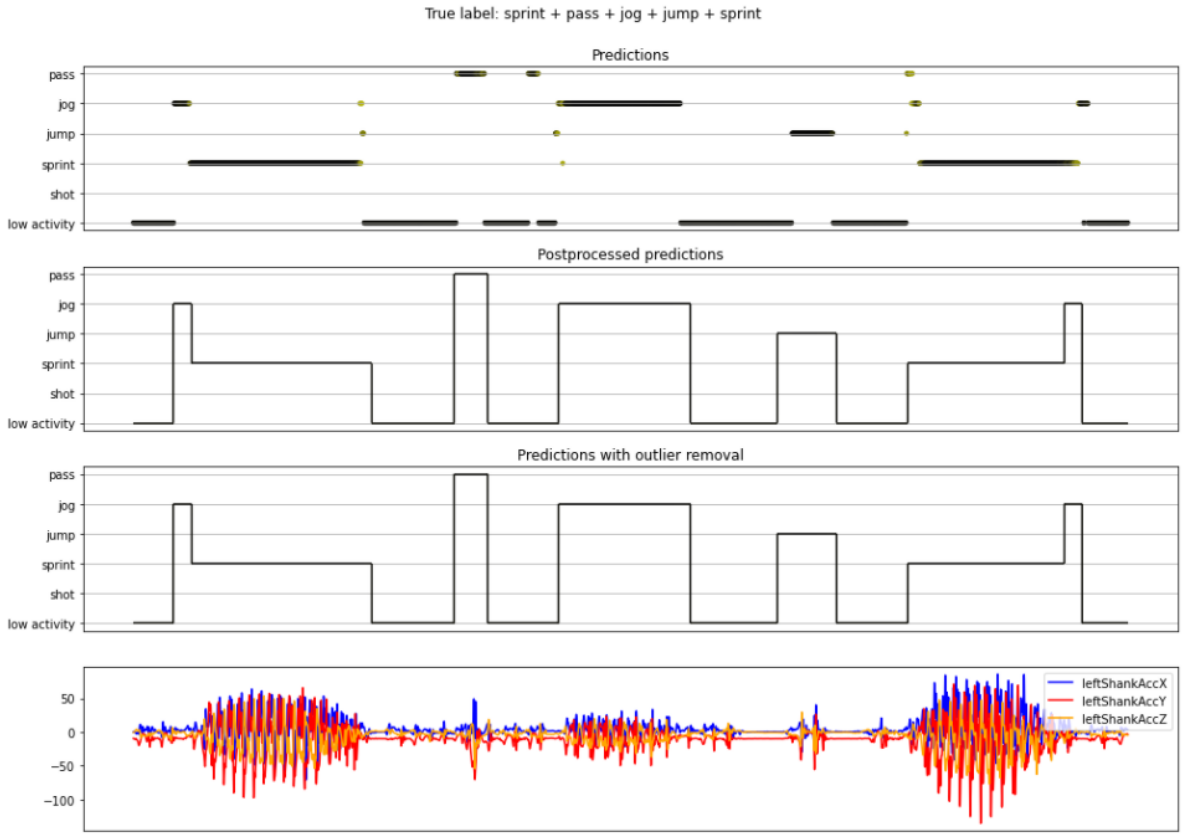


Figure 5.38: Example 3 of final results. True labels: sprint, pass, jog, jump, and sprint with low activity periods in between.

```
Summary of activities for recording:
low activity: 2.70 seconds
jump: 2.74 seconds
low activity: 1.76 seconds
jog: 5.10 seconds
low activity: 1.74 seconds
shot: 2.00 seconds
low activity: 3.70 seconds
pass: 1.54 seconds
low activity: 1.84 seconds
jog: 6.46 seconds
low activity: 0.72 seconds
_____________________________________
Total: 30.30 seconds

Number of times each activity was made:
jump: 1
jog: 2
shot: 1
pass: 1
```

Figure 5.39: Summary of recognized activities of figure 5.37

6

Discussion

In chapters 4 and 5, a detailed explanation of the data used and the activity recognition pipeline was given including procedure, design, and implementation with their respective analysis. This chapter will focus on the most important, relevant and interesting elements to be considered. Furthermore, additional experiments and tests will be presented, such as evaluation of the model on different datasets and building variants of the models in terms of the data used. At the end, the final conclusions will be presented with recommended next steps for future work.

One of the main goals of this thesis was to design, train and evaluate deep learning-based approaches for football activity classifications. For this sport, some previous works were found where traditional machine learning algorithms were used, but, as shown in chapter 2 and section 4.2, the usage of deep learning-based approaches was very promising due to several reasons. Chapter 2 showed that deep learning approaches have been taking over traditional machine learning algorithms when dealing with HAR applications, but the majority of the works have been made about daily human activities and not specific sports. Additionally, having Wilmes, 2019 worked with the same dataset using traditional approaches, it was possible to have an accurate comparison. Another important advantage of the usage of deep learning models that was exploited is the capability of such models to automatically extract relevant features without the need of manual selection and extraction of time and/or frequency domain metrics. This is one of the main characteristics of these models that allows them to capture more relevant and robust information without the need of human interaction. This ability, along with potential better performances and faster evaluation times, were the reasons why deep learning-based models were chosen.

Data is one of the most important parts when dealing with a machine (deep) learning application. It is said that “garbage in, garbage out”, which means that if the data that us used to train and test the model is not enough and has bad quality, the model will never perform good. For this thesis, the training of the model was done with data originally built by Wilmes, 2019 in a controlled and well documented experimental protocol. The way this dataset was built allowed the extraction of several samples (recordings) of single football activities, that were used to train and validate the model. One of the most important research questions of this thesis was whether it is possible to build an effective football activity classifier with raw signals: no filtering and no preprocessing of the signals. This is why the values and nature of the signals were not modified before entering to the model.

A first analysis of the dataset showed that the activities were not completely isolated from surrounding noise (low activity periods before and/or after the execution of the activity). It was feared that those low activity periods could confuse the training of the model by making it focus its attention on those irrelevant intervals. Because of this, it was proposed to build an activity detection algorithm, which was in charge to further refine the the input dataset by removing undesired low activity periods from before and after the actual movements. The usage of a threshold over the (scaled) mean of the norm of the accelerometer signals was found to perform well for this task. However, this algorithm turned out to be somehow dynamic in the sense that the aforementioned threshold had to be chosen as the mean of

the norm signal for some activities, and as 1.5 times that value for others. Interestingly, it was also evidenced that the first of those two groups corresponded to sprints and jogs (periodic long-term activities) and the second one to shots, passes and jumps (explosive, short-term activities). In order to have a automatic process without the need of human interaction, it was found that the interquartile range could be used to distinguish those two groups: periodic activities tend to have a larger and more spread IQR distribution. Without planning, an algorithm capable of differentiating periodic from explosive activities with 99.58% accuracy was developed by setting a threshold on the IQR of the recordings. However, since this was not the focus of the thesis, further experimentation and testing should be done to use this approach as a standalone periodic-vs-explosive activity classifier. This activity detection algorithm effectively cleaned the recordings and isolated the required activities from pre and post low activity periods (see figure 4.13). These recordings were the ones used to train the football activity recognition models. It is very important to note that this procedure was only made for the training phase. For the evaluation phase, since a sliding window approach was used, no activity detection algorithm was needed.

After the activity detection algorithm cleaned the dataset, it was possible to see the distribution of the duration of each type of activity. As expected, explosive activities such as passes, shots and jumps lasted for just a couple of seconds, while periodic activities lasted much more time. A sliding window of 1 second was chosen. By choosing windows of this length, it was possible to capture a big part of the explosive activities and, since periodic activities are repetitive, their pattern of movement could also be extracted. The usage of different sizes of sliding windows was not explored and it is a hyperparameter that could be further evaluated to verify how the model performance improves or worsens because of it. Additionally, since the activities were already isolated and centered, it was possible to obtain their average patterns, as shown in figure 4.15. In that figure, it is possible to appreciate the common patterns of each type of movement (see for example the jumps, where two peaks of activities happen before and after a short period of low movement, which represents the airtime of the jump). But more importantly, these graphs were used to argument why the magnetometers were not further used: they carried little or no information useful to distinguish between classes.

Having the isolated activities, it was possible to build the datasets to train and test the models. Although in football there is a high variety of movements, the five most common activities were chosen as the classes to be recognized via the deep learning based model: passes, shots, jumps, sprints, and jogs. The pipeline presented in this thesis can be, however, expanded to build and train models with additional activities if needed, such as turns or slides. As explained, a sliding window approach was used that extracted, for each recording, 1 second-long windows with 75% of overlap among them. This was made because of two reasons mainly: to have a consistent input size of the models, as they require; and to expand of the original dataset, since now there were several more samples of each one of the activities instead of only one instance per recording. Since recordings for sprints and jogs lasted more in the original dataset, the sliding window methodology generated a highly unbalanced dataset, as it was seen in figure 5.12a. Unbalanced datasets are usually a problem when dealing with classification problems such as this one, because the model will eventually see more examples of the most frequent class during the training phase and will not learn enough about the least frequent ones. To overcome this issue, the dataset was balanced by under-sampling the most frequent activities, so that all the classes had the same amount of samples as the least frequent one (see figure 5.12b). This, however, has the problem that a large amount of training examples are discarded, reducing the size of the dataset, which is undesired. Other re-sampling techniques could have been used, such a over-sampling the minority, capturing more data on the field, or even synthesizing artificial samples of the classes with techniques such as SMOTE (Synthetic Minority Oversampling Technique). At the end, the more (with high quality) data there is for the model, the better.

Based on literature review, it was seen that the usage of Convolutional Neural Networks and Recurrent Neural Networks was recommended. For this reason, several models were built that combined CNNs and RNNs. For the CNNs, 7 different types of convolutional layers were evaluated. Convolutions along only one the signals, along the three axial signals of the same sensor, and along all the signals were designed. Also, convolutions with weight sharing and with independent kernels for each signal or sensor were proposed. Even combinations of these types of convolutions were built. The

logic behind these types of convolutional layers was given in section 5.1.2, but in summary, the idea was to capture different type of shared and independent information among the signals and sensors. With respect to the recurrent layers, LSTMs and bidirectional LSTMs were chosen to extract temporal information of the recordings. To train the models quick and effectively, a learning rate scheduler was defined that modified dynamically the learning rate during the training phase. Additionally, to reduce overfitting, early stopping was executed on the training and dropout layers were implemented across the architecture. At the end, 23 deep learning architectures were proposed, trained and fine-tuned over 4 different variants of the input data: with and without standardization; and using accelerometer and gyroscope signals or only accelerometers. This resulted in 92 models, that were trained 5 times with different 70%-30% train-test splits. The average and standard deviation of their resulting prediction accuracies were presented in tables 5.1 and 5.2. By standardization of the input data, it is referred to the centering of the values of each window by subtracting the mean and dividing by the standard deviation of each one of the windows. It is, then, a local standardization.

The results of training and testing the 92 aforementioned models showed that, in general, good performances were achieved in terms of prediction accuracy and with low overfitting. Models only based on LSTMs or bLSTMs, however, did not perform well and obtained low test accuracies with clear overfitting. Furthermore, it can be seen that models with a combination of convolutional and recurrent layers had better accuracies than those that only rely on convolutions. This can be explained by understanding the role that both CNNs and LSTMs play in the architecture. Convolutional layers perform, as their name suggests, convolutions over the signals and combinations of them. These operations extract relevant features that are present in the recordings and are characteristic of the activities to be recognized. The initial convolutional layer extracts simple features, which are then combined into more complex ones with the deeper convolutional layer. This is the power of CNNs: the ability to extract relevant features without the need of specific human interaction. LSTMs are placed after these convolutional layers and they are responsible of taking those extracted features and giving them temporal meaning, knowing that the IMU signals are time series and should be treated as that. The combination of these two types of layers allows models to effectively extract features and time-relate them so that final accurate classifications are made. It is interesting to note that only CNN models performed better than only LSTM models. This implies that the usage of convolutional layers to extract relevant features is critical. The usage of bidirectional LSTMs gave better results than the usage of regular unidirectional LSTMs, suggesting that important elements of the recordings are captured when looking at them from right to left. The usage of both acceleration and gyroscope data allowed the training of better models than when only accelerometer data was used.

Accuracies of up to 96.71% with low deviation were obtained, specifically when using what we called 1D CNNs in combination with bLSTMs. Good results were obtained for almost all the models that had combination of CNNs and (b)LSTMs (between 94,94% and 96,71% accuracy). Some of the differences in accuracy are very small, so it is not correct to say that one specific model is better than the other good performing ones just based on the test accuracy. To choose a model as the final one, both the evaluation time and prediction accuracy were evaluated. Based on its high average accuracy, low overfitting and fast evaluation time, a model such as 2DCNN_perSensor_bLSTM could be chosen. This does not mean that other models cannot be used depending on the necessities, such as 1DCNN_perSensor_bLSTM, which is also a good choice, since it has a slightly larger evaluation time, compensated by a smaller error. Figure 5.15, on the other hand, compared the prediction accuracy and evaluation time of one of the chosen deep learning-based model and several traditional machine learning-based classifiers built using the exact same dataset. It is clear from the graph that deep learning models perform better and faster for this problem, supporting their usage for football activity recognition.

Table 5.1 also showed that the usage of locally standardized windows worsened the prediction accuracy for almost all the models with respect to the unstandardized ones. Even more, locally standardized models had more overfitting and larger deviations on their accuracies. Although normalization or standardization is a common and recommended practice when building deep learning models, locally standardizing the input windows deteriorated the performance of the models. This is because by doing so, the signals lose the relationships between the scales among them, which are very impor-

tant for the classification. By doing local standardization, the signals of each window are standardized using the means and standard deviations of the signals of that specific window. This makes that, for instance, windows of a shot and a pass are standardized independently and their signals look similar at the end because they are both locally standardized. Nevertheless, a pass has smaller signal values than a shot and the standardization should reflect this. If standardization or normalization is desired, it must preserve the relative scale of the activities. This can be done by standardizing or normalizing all the windows with the values of a calibration recording (a recording where many different activities are performed) instead of using the local values of the window. This experiment was performed and will be discussed in section 6.2.2.

The general pipeline also allowed the recognition of low activity periods, which are of huge importance when distinguishing one activity from another in a row. As explained in section 5.2, different methodologies could have been followed to make the model additionally understand these low activities. Since the deep model had a very high accuracy when recognizing the 5 high activities (jog, jump, shot, sprint, and pass), in order to maintain this good performance, it was decided to implement a binary high-vs-low-activity classifier on top of the deep model. So, the window to be classified passes first through the binary classifier, which identifies the window as a high or low activity. If it is detected as the former, it is then fed to the deep model that further classifies it as one of the 5 high activities. Because of the evident differences in the signals of these both types of activities, it was possible to build an accurate binary classifier responsible of that distinction using solely the standard deviation of the norm of accelerometer signals. Different options could have been used to build that classifier: low activity could have been added as an additional class to a two-output deep model so that it would recognize passes, jumps, shots, sprints, or jogs on one output and high or low activities on the other; or an independent binary high-vs-low-activity classifier could have been built using another deep neural network. The threshold-based binary classifier achieved an accuracy of 96.56% on the test dataset.

Postprocessing and outlier removal processes were implemented to align the predictions with all the timesteps of the recording and clean the outputs of the model from unwanted short-lived and low confidence predictions. Three postprocessing options were proposed and implemented: interpolate, mode and best score. Each one of them was thoroughly explained. It was shown that the interpolate postprocessing was not useful to remove unwanted predictions, and depending on which part of the window is used as the location of the prediction, it moves the predictions across time to the future or the past. To take the most out of the sliding window approach for the evaluation phase, the mode and best score postprocessing variants were designed. They both use the fact that a single timestep will be evaluated by several windows, so the final prediction must take into account all of them. These two options have the additional property of cleaning the final outputs: the former one gives more importance to predictions of the same class over the timestep; the latter focuses on the confidence of the predictions, giving more weight to predictions with high confidence, even if they are isolated. In order to obtain those prediction confidences or scores, the output of a softmax activation function was used on top of the last layer of the deep model. These values can be understood as the probabilities of the window being classified as each one of the 5 high activities. The best score is the type of postprocessing recommended, because intuitively it makes more sense to focus on predictions with high confidence, it gives information about the score of each prediction, and it gives better and clearer results. Additionally, this postprocessing option is analogous to what the commonly used Non-Max Supression algorithm does in object recognition tasks. The subsequent outlier removal process eliminates short-lived predictions, since activities that last less than a certain amount of time are not physically possible and should be considered outliers. Its operation is dependent on the hyperparameter τ.

The usage of the output of the softmax activation function to obtain class probabilities allowed the usage of the best score postprocessing and the visualization of the confidences of the predictions with a yellow-black color gradient. However, since low activities are detected used a hard threshold classifier, they do not come with any confidence values (this is a reason why it could make sense to build soft binary high-vs-low activity classifiers using another deep model, for example). Because of this, windows classified as low activities were artificially set to have confidence c_l. This value, that was set to 98% for the presented results, must be high and determines how much the binary classifier should be trusted in detecting low activities. Larger values of c_l would make low activities more important,

making the predictions of high activities between low activity periods narrower. Furthermore, the prediction confidences allowed the inclusion of an additional class: other high activity. This class is defined as any window recognized by the binary classier as a high activity, but classified by the deep model as any of the 5 activities with a low confidence (lower than h_h). These other high activities have, by definition, low confidence, so to avoid them from being eliminated from the final predictions due to the best score postprocessing, they are given an artificial large score c_h (95% was used for the presented results). Similar to the confidence c_l of low activities, c_h must be large and is set as a hyperparameter to allow experimentation. Note that, however, the general pipeline is not designed to recognize this "other high activity" class. Its inclusion is just a way to consider this class depending on the chosen value for h_h. The deep model is optimized to recognize a window as a jump, pass, shot, sprint, or jump, so it will always try to understand a recording as one of those 5 activities. If an additional type of activity wants to be recognized (such as turns or slides, for instance), the model must be retrained with enough examples of that class to be able to specifically recognize it.

6.1. Evaluation on different datasets

The model was trained and evaluated using the dataset provided by Wilmes, 2019, as explained. The figures that have been presented in section 5.4 are the results of evaluating the model on unseen samples of the same dataset. However, it is important to see if the built pipeline can generalize and also give correct results on different datasets. To evaluate this, the following datasets were used:

6.1.1. Rozemarijn's dataset

This dataset was provided by Schotel, 2019. It was solely used to evaluate the performance of the trained models on a more complex dataset, such as this one. This dataset had several recordings of activities happening one after the other in a more real scenario instead of short recordings where single activities happened, making the task of activity recognition more difficult.

To build this dataset, Schotel, 2019 attached 5 IMUs (Shimmer3) in the exact same locations as shown in figure 4.1 on the participants. Five male subjects performed drills that included activities such as jogging, sprinting, passing and shooting. In particular 2 drills were executed, each one of them composed of several situations where different activities were performed. A complete and detailed explanation of the experiments and procedures can be found in the original paper (Schotel, 2019). These experiments resulted in much longer recordings (almost 1 hour) of more complex data.

Similar to the dataset from Wilmes, 2019, each one of the IMUs had a tri-axial accelerometer, gyroscope and magnetometer. The range for the accelerometers was set to $\pm 16g$, for the gyroscopes to $\pm 2000°/s$, and for the magnetometers to $\pm 4.7Ga$. A big difference with the previous dataset is that this one used a sample frequency of of 200 Hz instead of 500 Hz.

When working with a different dataset, one of the first things that should be done is to check whether the distribution of these new data is similar to the one of the data that was used to train the model. As quoted in McGaughey et al., 2016, "a common prerequisite in supervised learning algorithms is that the training and prediction data arise from the same distribution and are independently and identically distributed (iid). Intuitively this is justified, as one should not expect to learn a classifier on one distribution of examples and apply it to accurately predict labels of examples drawn from a different distribution". This problem is often called covariate shift (figure 6.1). A model evaluating a testing or prediction dataset with a different scale and distribution than the training one will not perform as good as expected. This is actually one of the reasons why it is a good practice to normalize the data when training and evaluating models.

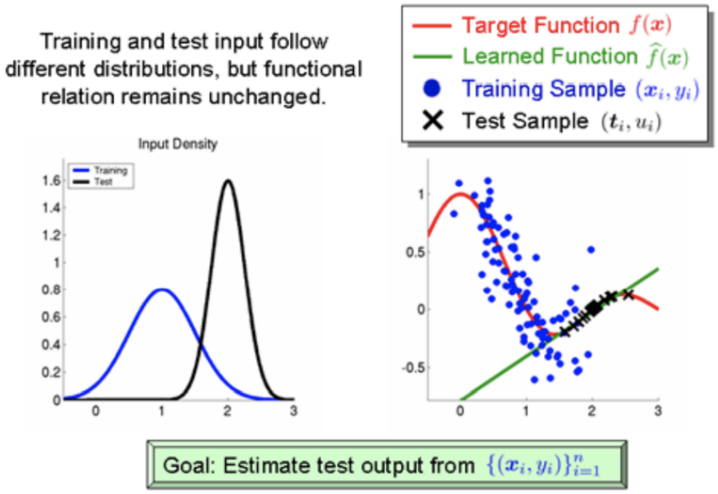


Figure 6.1: Covariate shift. Taken from (Stewart, 2019)

By plotting the time series of Wilmes' (train) and Rozemarijn's (test) datasets, it was evident that

both had different scales. Rozemarijn's sensors provided measurements with considerable larger values, specially with the gyroscope signals. Examples of these plots can be seen in figures 6.2 and 6.3. This change in scale will be a problem when using the model to predict Rozemarijn's recordings, since the model is not normalized.

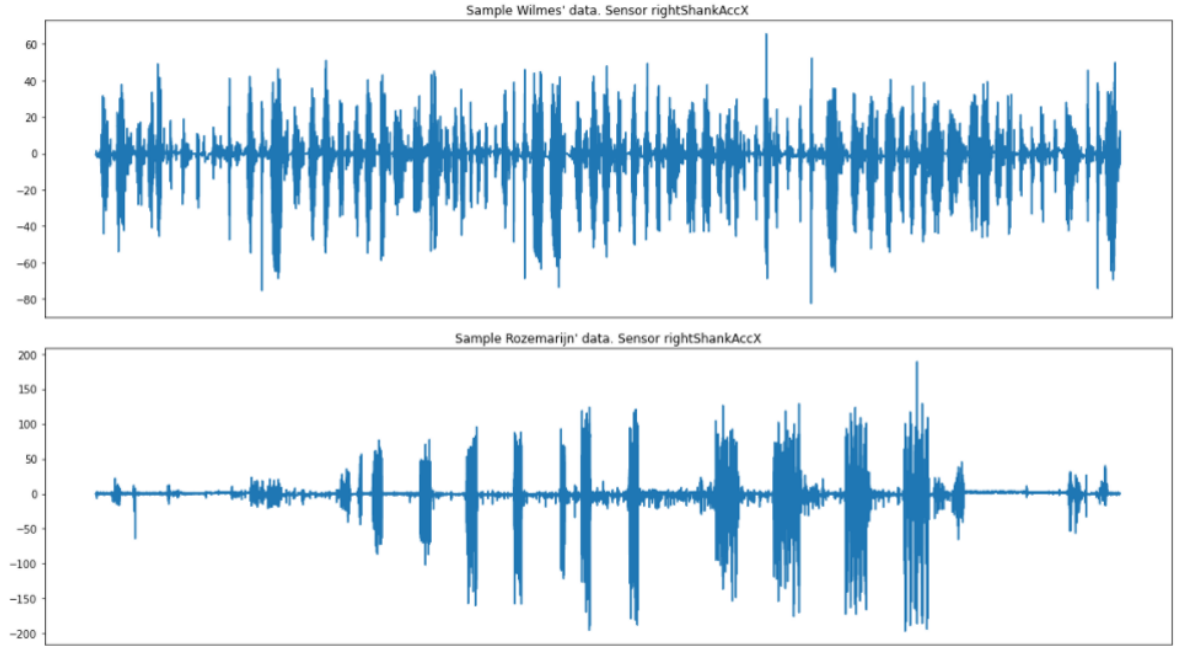


Figure 6.2: Comparison of scales of measurements in Wilmes' and Rozemarijn's datasets. Sample accelerometer signal

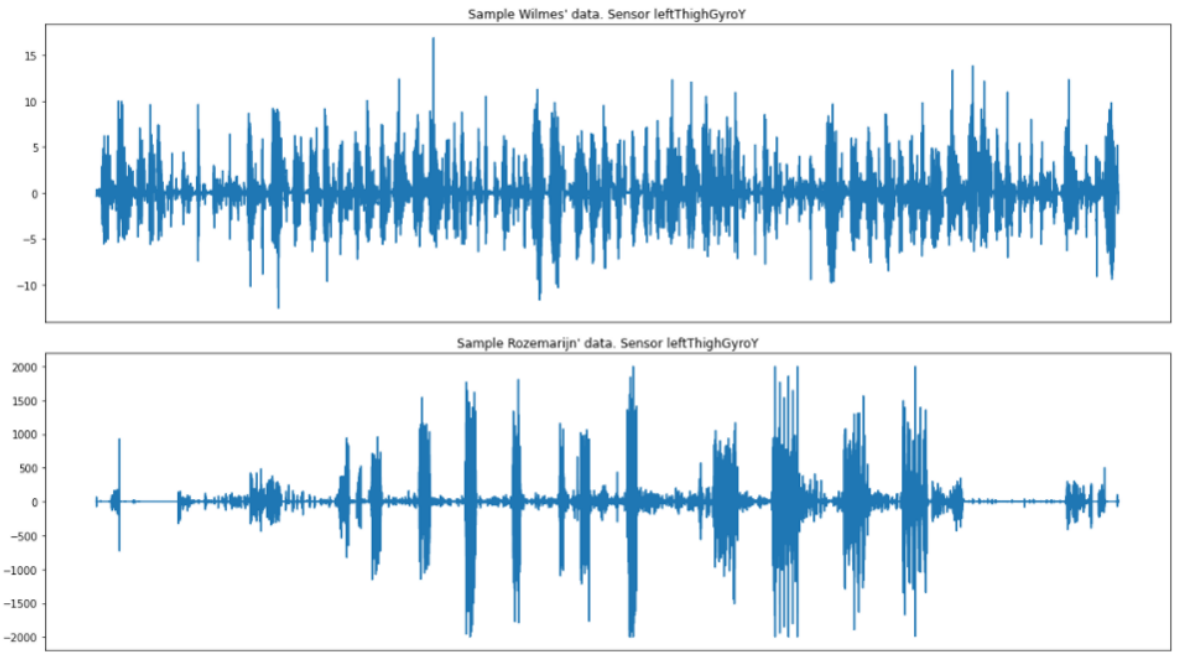


Figure 6.3: Comparison of scales of measurements in Wilmes' and Rozemarijn's datasets. Sample gyroscope signal

Not only the scales were different, which implied the use of different calibration methods of the sensors, but also some of the signals had significant different distributions, as can be seen in figure 6.4.

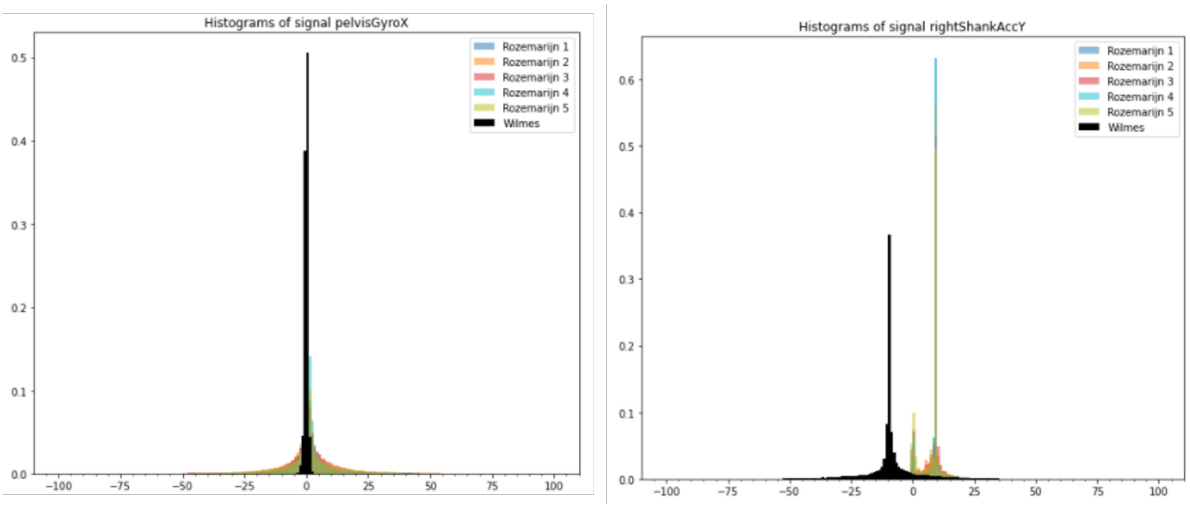


Figure 6.4: Comparison of distributions in Wilmes' and Rozemarijn's datasets for selected signals. Histograms limited from -100 to 100 for visualization purposes

The difference in scale, distribution, and even sampling frequency (the signal had to be resampled to 500 Hz) had terrible consequences for the model using the aforementioned values for the hyperparameters h_h, c_h, c_l. As it can be seen in figure 6.5 with the yellow dots, all the predictions of high activities had very low confidence. The example there depicted corresponds to an interval of the recording where a drill consisting of sprints, turns and shots are performed 10 times in a row. All the predictions made by the model had very low confidence, so many of them were considered as "other high activity". But, since c_h assigns by default 95% confidence to "other high activities", they end up having scores larger than all the predictions of high activities and the final prediction is composed of solely "other high activities". However, if we check the initial predictions (yellow dots on the top graph), it can be seen that there is a certain tendency of the model to predict more jogs and shots. So, by not imposing that 95% confidence on "other high activities", the results of figure 6.5 are obtained. The final predictions in this case had low scores, as seen with the yellow tone in them, but it was possible to evidence the model recognizing the activities, even with bad confidences. As mentioned, this sample corresponded to repetitions of sprints, turns and shots. The model recognized patterns composed by jogs (slow sprints) and "other high activities" (turns) followed by shots. It is clear that the predictions are not entirely correct, and, as explained, this can be attributed to the large difference distribution and specially scale for this dataset. However, this experiment allowed us to confirm that the model can still generate not so inaccurate predictions on when the scale is different, but it is needed to tweak the hyperparameters c_h and h_h to see some results.

Another example of results obtained by the model on a different interval of Rozemarijn's dataset is presented in figure 6.7. To obtain those final results and graphs it was necessary to not impose c_h, just like with the previous example. Additionally, the minimum activity duration was set to 100ms for the outlier removal phase. The recording there presented corresponds to 5 repetitions of consecutive sprints and turns (zigzag) separated by short low activity periods. Eventhough the model is not completely accurate in this case, it still has the ability to recognize the 5 activity intervals and that there are, in general, jogs (slow sprints) interspersed with "other high activities".

Finally, a whole recording of one of the experiments of this dataset was evaluated to address the time required to predict a long series (figure 6.8). This recording lasts 2476 seconds (about 42.3 minutes). It was evaluated using a sliding window step of 100 ms which resulted in 24751 windows predicted. The time needed to recognize (predict) all the windows was 24.06 seconds and the best score post-processing took 7.46 seconds. In total, after the signal was re-sampled to 500 Hz (because it was sampled originally at 200 Hz), the final predictions for this 2476 seconds-long recording were obtained in 31.52 seconds (1.27% of the length of the recording). These numbers were obtained when running the code on a local computer without GPU (Intel i5-8250 CPU @ 1.60-1.80 GHz with 8GB of RAM).

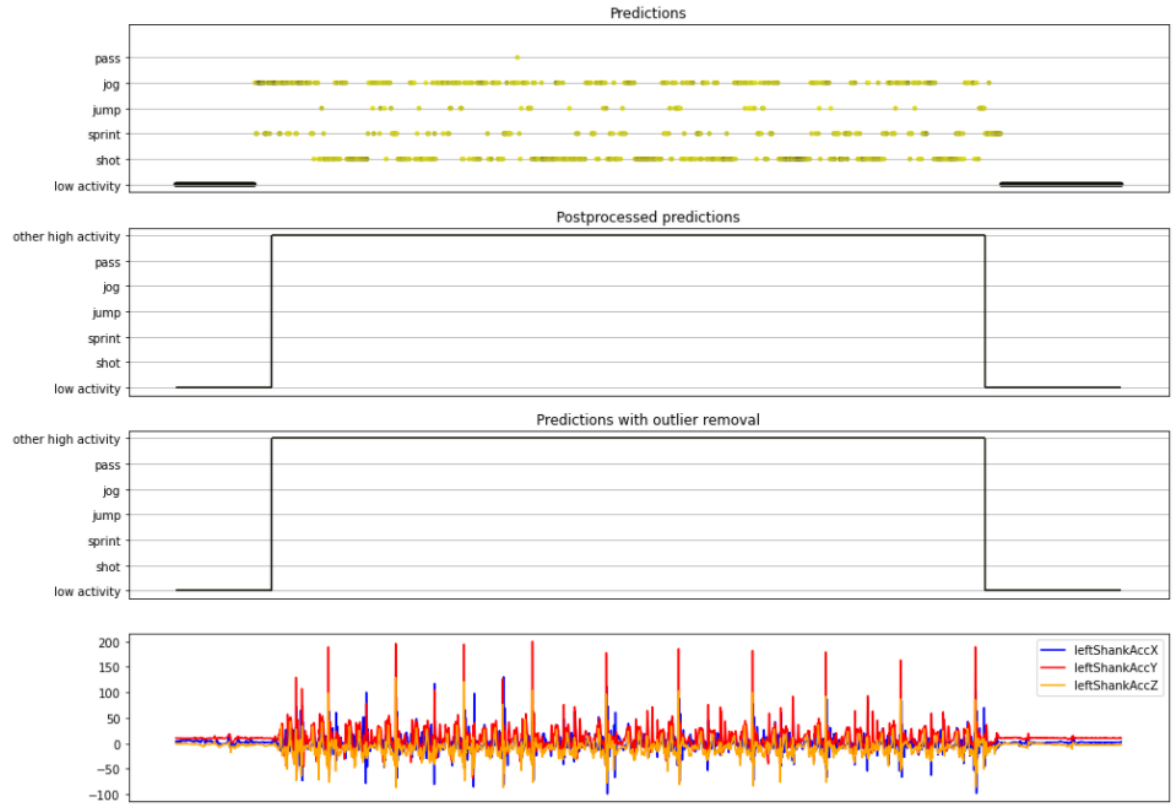


Figure 6.5: Example of predictions made on part of Rozemarijn's dataset with default values for hyperparameters

When running this using Google Colab's GPU, the prediction and postprocessing times decrease to 7.08 seconds and 6.88 seconds respectively (13.96 seconds in total, that is just 0.56% of the duration of the recording).

The main conclusion of these experiments is that the scale and the distribution of the input recording has a big influence on the model's performance. In those cases, the predictions obtained by the model were not completely wrong, but their confidence were significantly low. This implied that certain hyperparameters such as τ and c_h had to be modified to obtain predictions other than "other high activities". The different scale and distribution clearly confuses the model. It is, then, recommended to use recordings with similar scales as the ones used to train and validate the model. This can be achieved by calibration of the sensors.

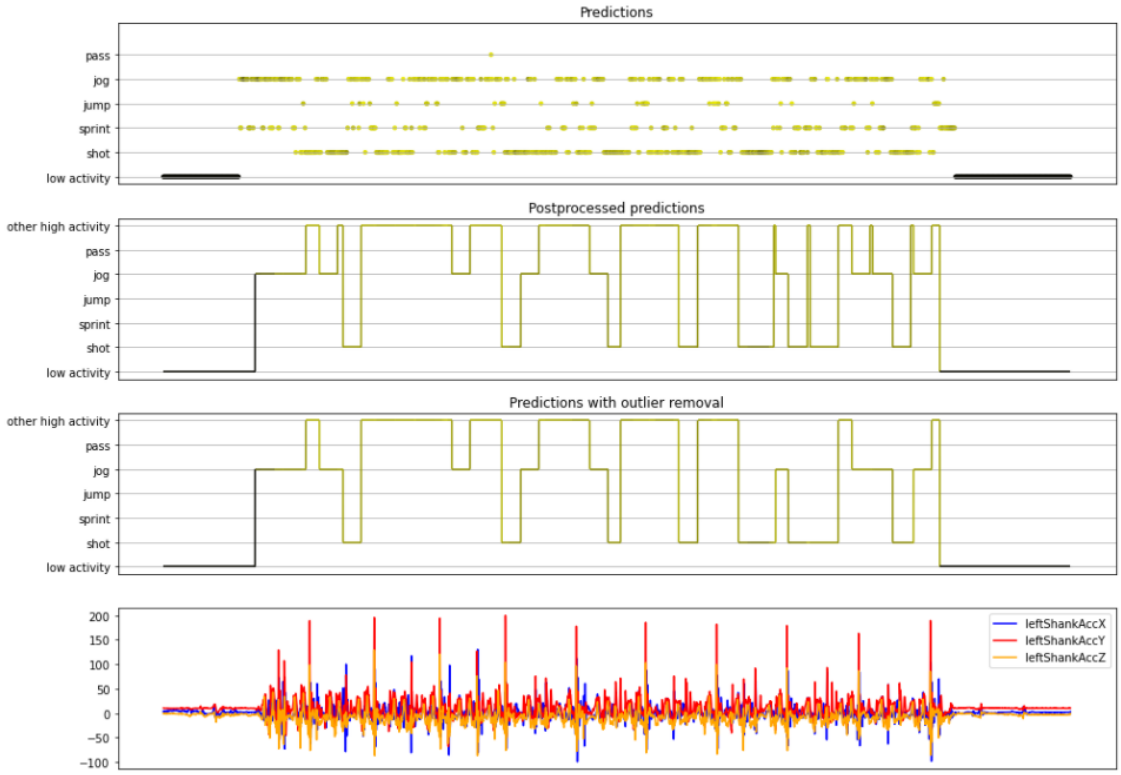


Figure 6.6: Example of predictions made on part of Rozemarijn's dataset without imposing a value for c_h. True label: sprint, turn, sprint and shot 10 times in a row

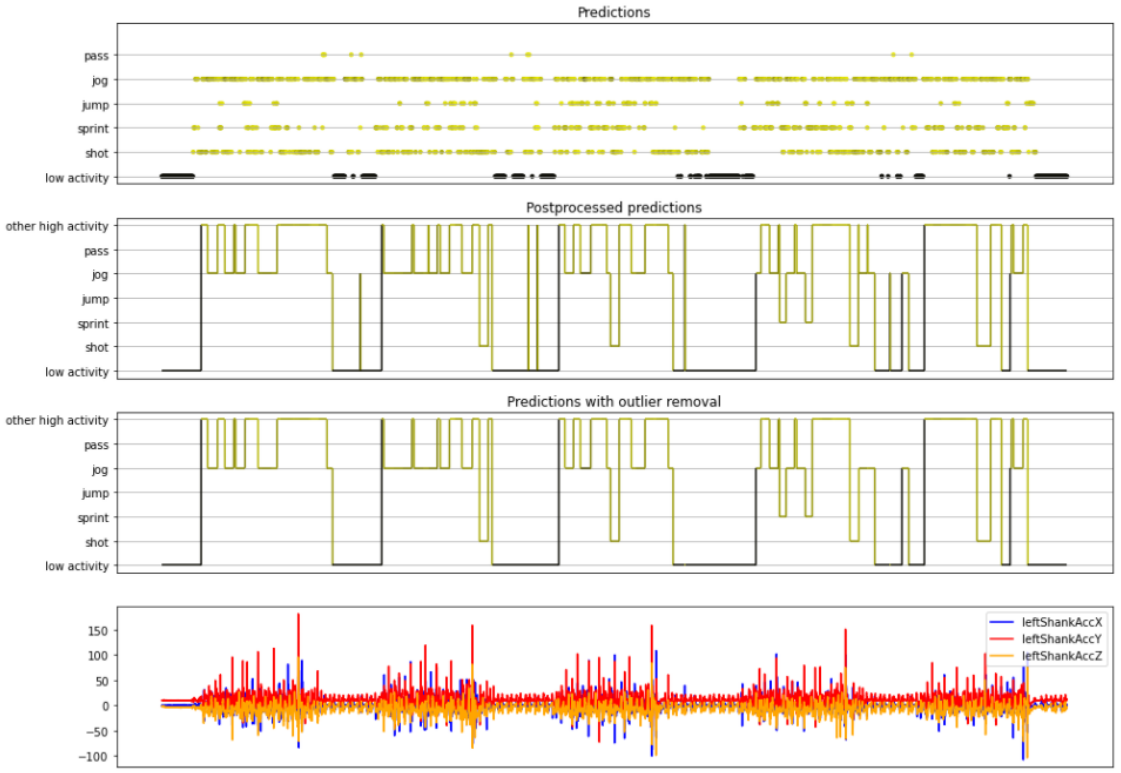


Figure 6.7: Second example of predictions made on part of Rozemarijn's dataset without imposing a value for c_h. True label: 5 repetitions of sprints and turns separated by short low activity periods

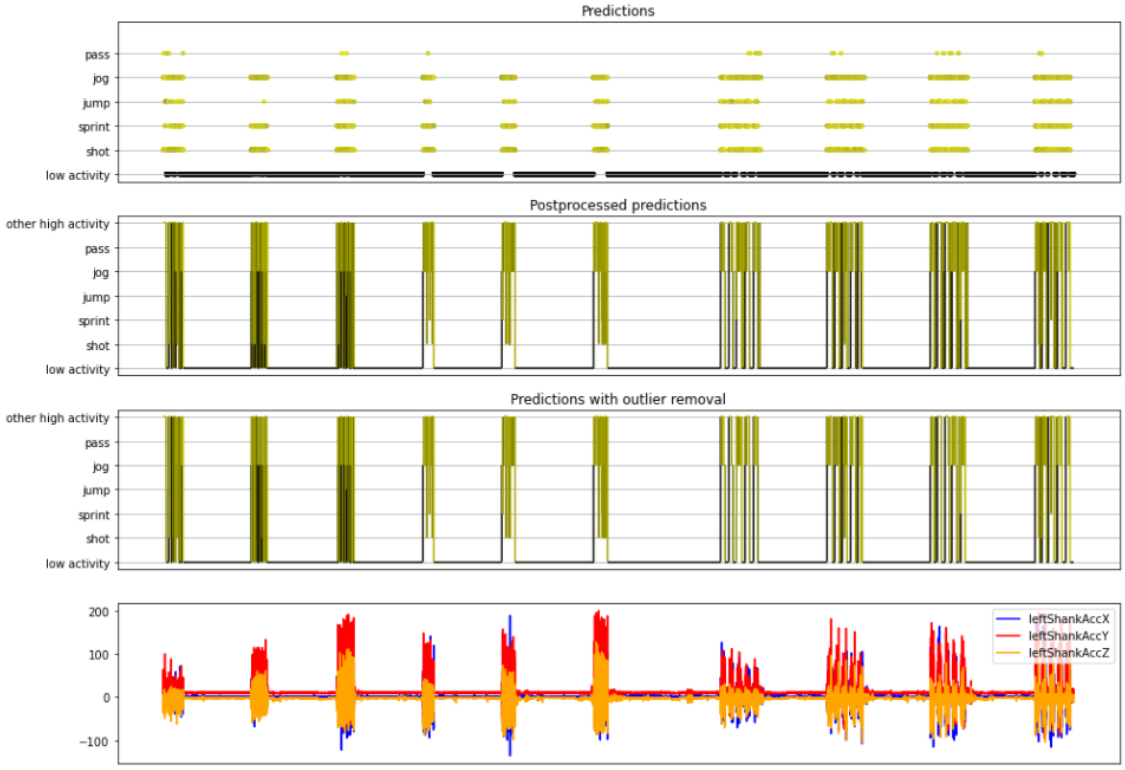


Figure 6.8: Evaluation of complete recording from Rozemarijn's dataset

6.1.2. Wilmes' new Dataset

This dataset was also provided by Mr. Erik Wilmes (same author of Wilmes, 2019). Just like Rozemarijn's dataset, it was solely used to evaluate the performance of the model on a more complex dataset that looked closer to an actual football match. Several exercises were made one after the by the players. The data was acquired in the same way as the data from Wilmes, 2019 was obtained: similar IMUs were placed on the same 5 body parts (left thigh, right thigh, left shank, right shank, and pelvis) and both tri-axial accelerometer and gyroscope measurements were captured.

Following the same procedure that was explained on section 6.1.1 with Rozemarijn's dataset, the first thing that was done was an exploratory data analysis to visualize the data. This was made with the purpose of checking for covariate shift: whether this new dataset had similar scale and distribution as the training dataset or not (like what happened with Rozemarijn's data).

Examples of time series from both Wilmes' (train) and new Wilmes' (test) measurements can be seen in figures 6.9 and 6.10. Unlike Rozemarijn's data, where the signals had significantly larger scales than the training data (especially with the gyroscopes), this new Wilmes' dataset had more consistent values with respect to the training data. There is some difference, however, in the accelerations, but the measurements coming from the gyroscopes have now a very similar scale. In terms of their distribution, as it can be seen in figure 6.11 (showing just two out of the 30 signals for illustrative reasons), the values were also consistent with those of the training dataset. In conclusion, there was little covariate shift on the new Wilmes' dataset, and we expected to obtain good results when applying the models on the raw data.

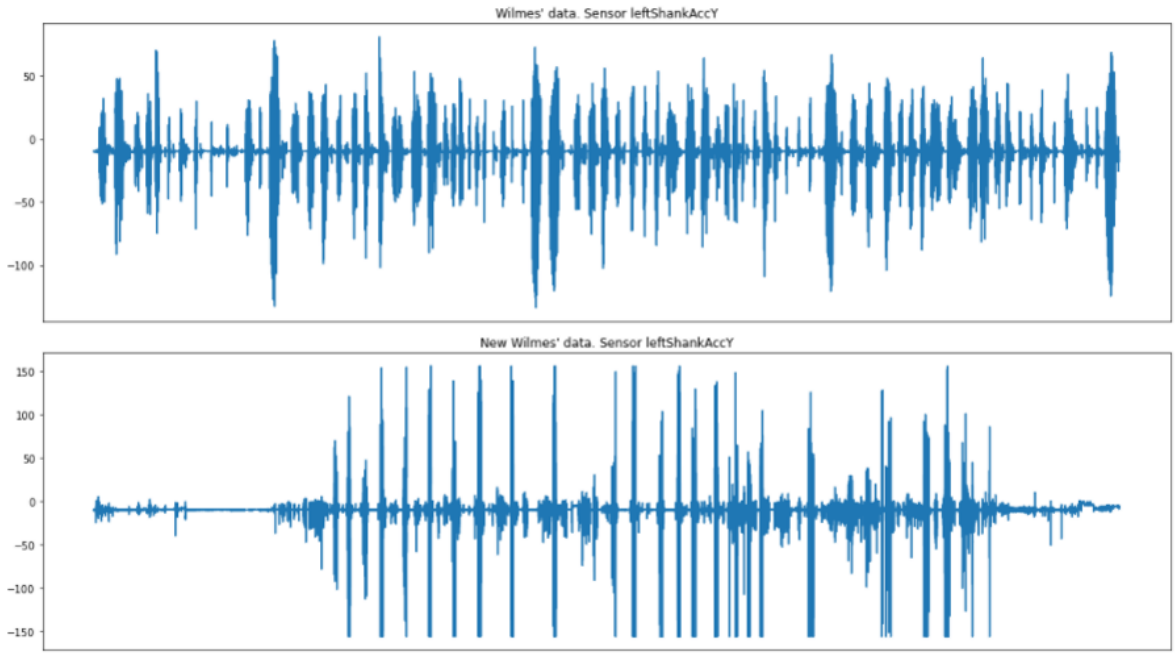


Figure 6.9: Comparison of scales of measurements in Wilmes' and new Wilmes' datasets. Sample accelerometer signal

In figure 6.12, the evaluation of a section of this new dataset is shown. That specific interval corresponds to a player jogging and turning 5 times with a short jog at the end. The recommended default parameters were not changed. The first thing that deserves special interest is the confidence of the predictions made by the model. In this case, since the dataset does not suffer from covariate shift, the predictions have very high confidence when one of the 5 original activities is present (in this case a jog). However, it can be seen that when turns occur, the model goves predictions of low confidence but of a consistent activity (it finds that the most similar activity to the turns are sprints). Having those predictions very low score, they are converted into other high activities, obtaining at the end a nearly perfect activity recognition process. The 5 turns are identified with perfectly defined jogs in between.

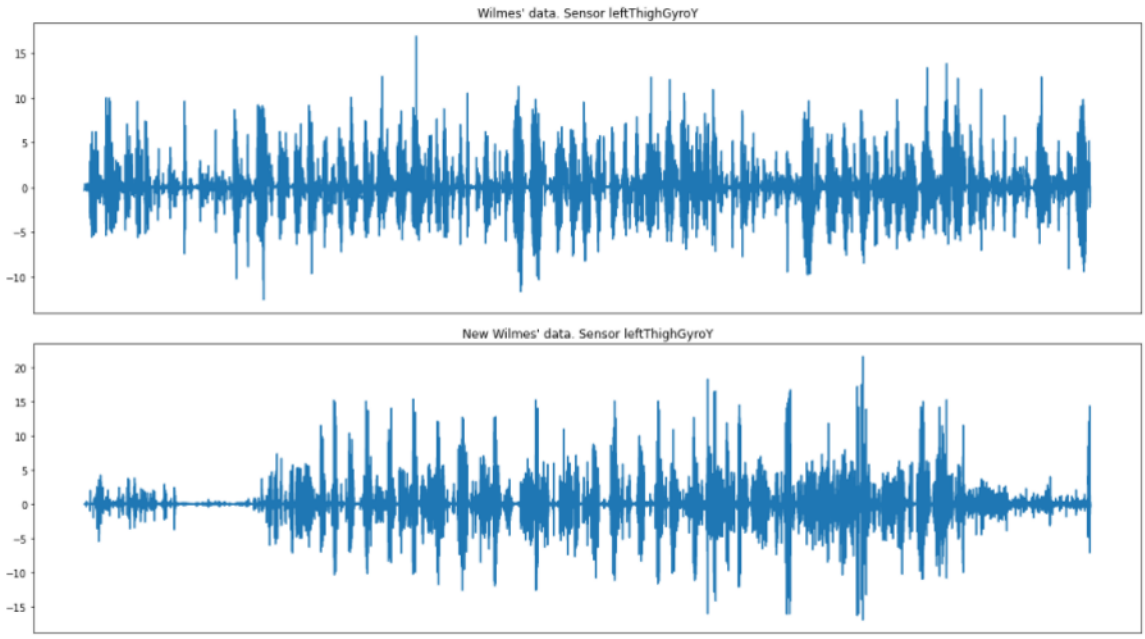


Figure 6.10: Comparison of scales of measurements in Wilmes' and Rozemarijn's datasets. Sample gyroscope signal

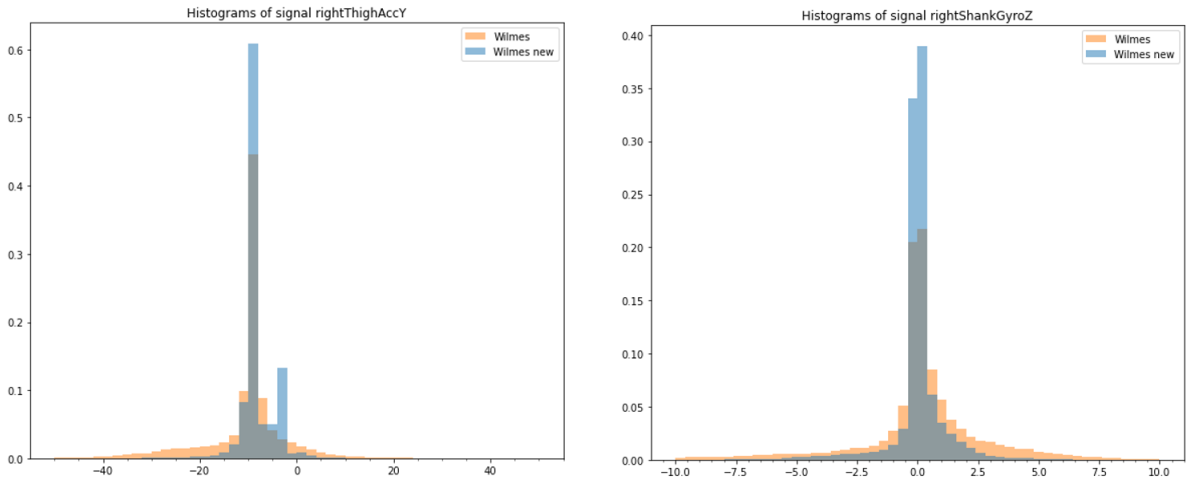


Figure 6.11: Comparison of distributions in Wilmes' and new Wilmes' datasets for selected signals

An initial period of other high activity is identified by the model, which can be explained by it capturing the first movement of the exercise. Note that the mode postprocessing option also gives very good results when the input dataset does not suffer from covariate shift (see in figure 6.13 the evaluation of the same interval using the mode postprocessing instead). In this case, since the low confidence predictions happen together during certain time, the mode postprocessing option also captures them as other high activities that occur for those periods of time.

Another two examples of results obtained by the model on different intervals of new Wilmes' dataset are shown in figures 6.14 and 6.15. Those two intervals consist of more complex exercises where various different activities are performed one after the other. In both cases, the model recognizes almost perfectly all the activities.

Figure 6.14 corresponds to the player sprinting and then jumping for 6 times. Then, he rests for some seconds and then performs 3 shots one after the other. The first part of the exercise is perfectly

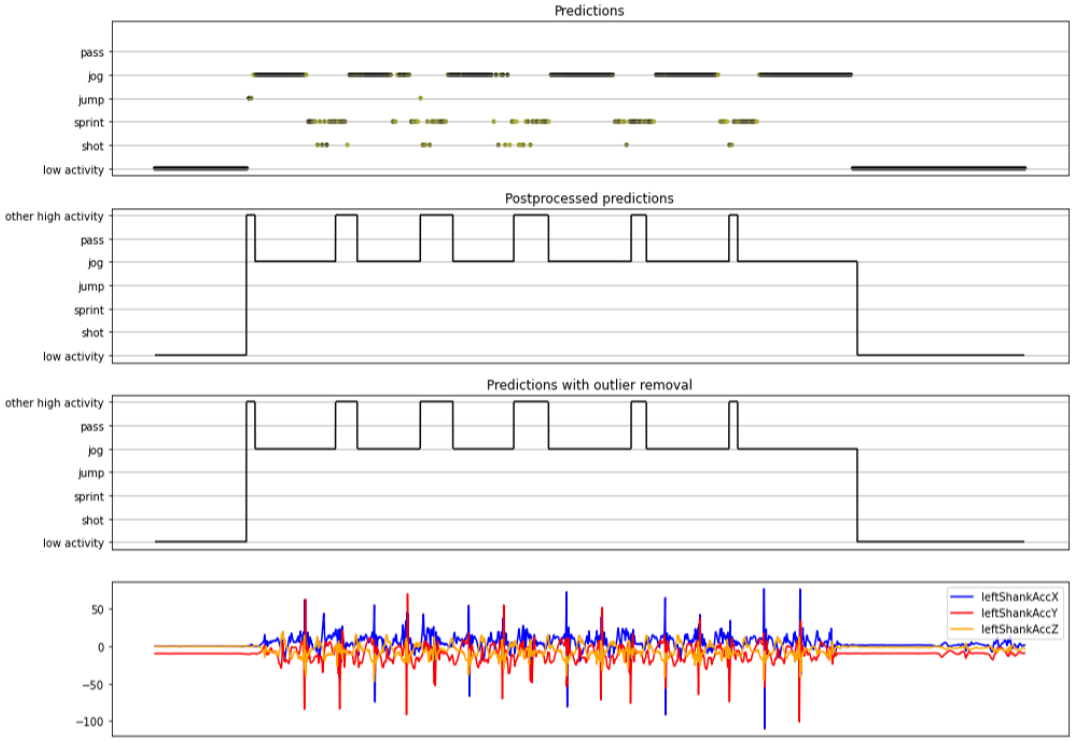


Figure 6.12: Example of predictions made on part of new Wilmes' dataset. Using best score postprocessing. True label: jog and turn 5 times with a final jog.

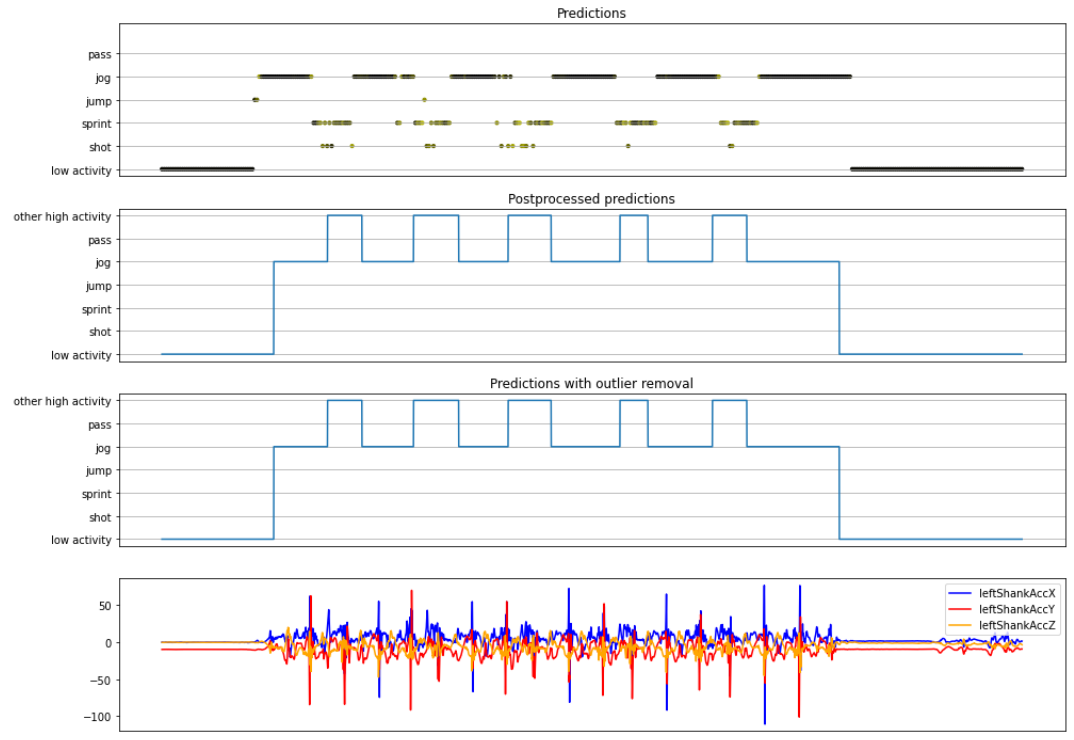


Figure 6.13: Example of predictions made on part of new Wilmes' dataset. Using mode postprocessing. True label: jog and turn 5 times with a final jog.

identified by the model with even an initial jump being detected when the person makes the first move. Then, the three shots are recognized, but it is interesting to note that they are not identified as pure shots. For the three cases, they are preceded by a short jog and followed by a jump. This makes

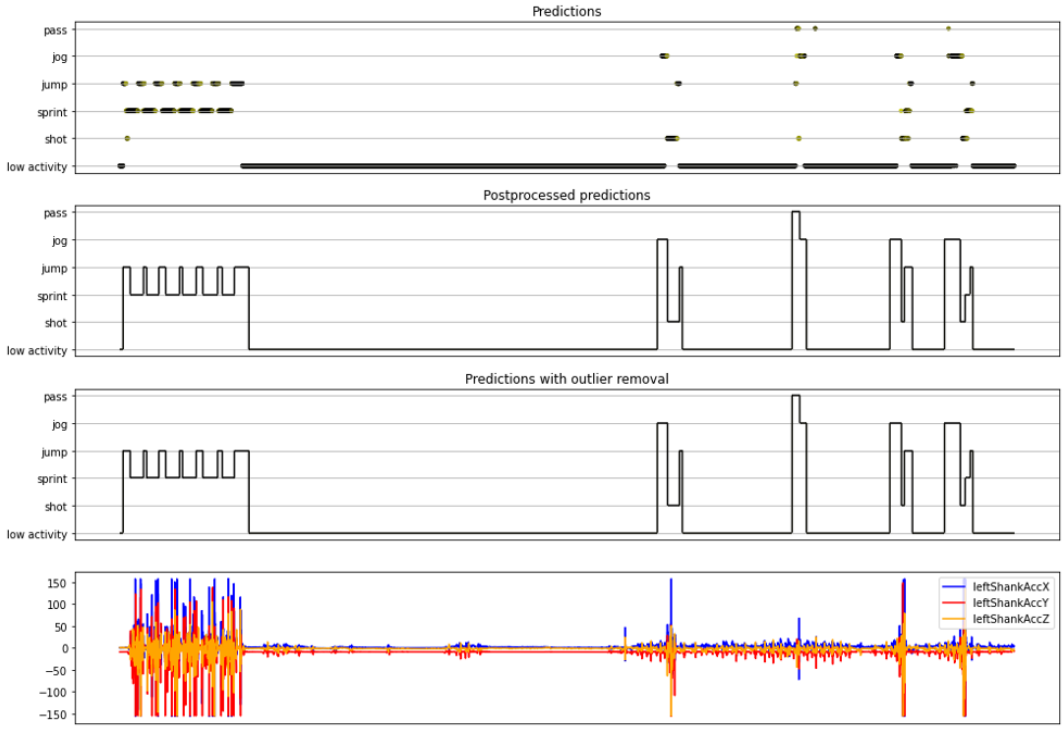


Figure 6.14: Second example of predictions made on part of new Wilmes' dataset. Using best score postprocessing. True label: period of alternating sprints and jumps done 6 times and then 3 shots

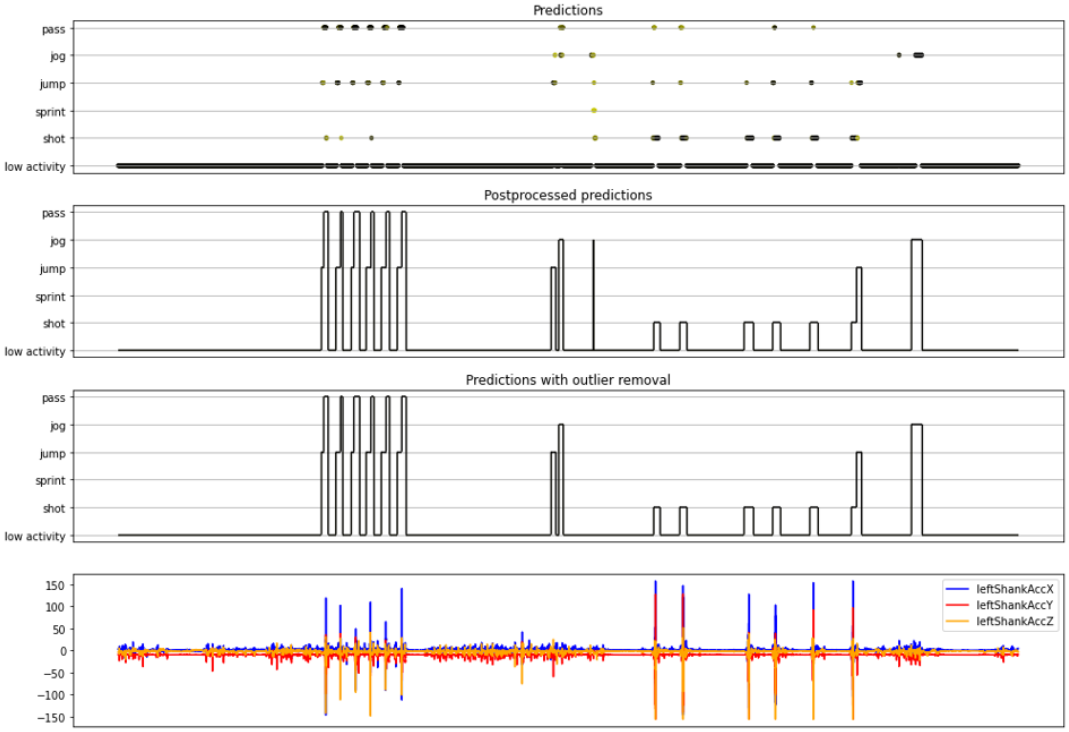


Figure 6.15: Third example of predictions made on part of new Wilmes' dataset. Using best score postprocessing. True label: 6 passes followed by 6 shots

complete sense by understanding how a shot happens in real life scenarios: the player must first take a jog (or sprint) towards the ball, then shoot and, because of the inertia of the high intensity of the movement, usually the player ends up jumping. The model effectively identified this dynamic. Note

that there is a short jog identified between the first and second shot. Although this movement was not part of the experiment, the player did a slow jog to position himself back to make a shot. The model, again, identified this situation.

The other example, in figure 6.15, shows the player making 6 passes and then 6 shots. The model identifies correctly the 6 passes, but with a short jump before each one of them. These short jumps represent the player positioning his body to perform the pass just before making the contact with the ball. Then, the 6 shots are also precisely recognized. In this case, unlike the previous example, 5 out of the 6 shots are purely shots with no jump afterwards. However, the raw predictions of the shots still show a tendency to find jumps next to the shots. There is a jog being recognized after the last shot, which corresponds to the player going to another position after the exercise is finished. Additionally, between the passes and the shots, the model found a short jump and a jog which were not part of the exercise and are treated as low intensity movements made by the player when preparing to perform the shots.

These experiments show that if the input data has similar scale, distribution and frequency as the training dataset (there is little covariate shift), the model and pipeline perform very accurate predictions and the activities can be identified. This whole recording had a duration of 4971.2 seconds (about 82.5 minutes). Its evaluation with a sliding window step of 100 ms using Google Colab's GPU took in total 14.84 seconds including prediction and postprocessing. This is just 0.3% of the duration of the recording.

6.2. Additional experiments

Apart from the evaluation of the model on different datasets, additional experiments were conducted in order to investigate the effect of other variables on the model performance.

6.2.1. Exclusion of shank sensors

All the datasets that were used in this thesis consisted of recordings containing IMUs in 5 different locations: right thigh, left thigh, right shank, left shank and pelvis. To capture the data, Wilmes, 2019 and Schotel, 2019 used specially designed pants with the sensors there integrated. However, football is a sport that must be played with shorts instead of pants. So, for real-life scenarios, it would be only possible to use pelvis and thigh IMUs integrated in special shorts. It would be very hard to place sensors on the shanks (they could be placed on special long socks, but that would make the communication of the system more complex). Because of that, it was decided evaluate the performance of the models if only 3 out of 5 sensors were available (left thigh, right thigh, and pelvis). To do so, the same procedure was followed but eliminating the right and left shank sensors from the datasets. Both accelerometer and gyroscope signals were included and raw data (without any type of standardization) was used. All the models were trained with the same architectures, parameters, and algorithms to have a fair comparison.

Table 6.1: Mean and standard deviation prediction accuracies for the models without shanks. 5 runs

	Train accuracy	Test accuracy
1D CNN weight sharing	98.05% ± 0.72%	91.01% ± 1.72%
1D CNN per sensor	98.64% ± 1.17%	93.64% ± 1.5%
1D CNN combined	98.12% ± 0.8%	92.49% ± 0.71%
2D CNN weight sharing	98.21% ± 0.77%	91.62% ± 0.82%
2D CNN per sensor	98.64% ± 0.82%	93.1% ± 1.31%
2D CNN all signals	97.88% ± 0.61%	92.71% ± 1.44%
2D CNN combined	94.38% ± 8.82%	89.64% ± 11.99%
1D CNN weight sharing + LSTM	97.44% ± 0.85%	94.25% ± 0.49%
1D CNN per sensor + LSTM	97.58% ± 0.26%	94.79% ± 0.57%
1D CNN combined + LSTM	98.47% ± 0.63%	95.51% ± 1.08%
2D CNN weight sharing + LSTM	98.19% ± 0.61%	95.12% ± 0.27%
2D CNN per sensor + LSTM	98.26% ± 0.52%	96.0% ± 0.71%
2D CNN all signals + LSTM	97.46% ± 0.45%	93.86% ± 0.66%
2D CNN combined + LSTM	99.13% ± 0.36%	96.71% ± 0.77%
1D CNN weight sharing + bLSTM	98.28% ± 1.2%	95.12% ± 1.37%
1D CNN per sensor + bLSTM	97.88% ± 0.59%	94.85% ± 0.78%
1D CNN combined + bLSTM	97.79% ± 1.01%	94.74% ± 0.87%
2D CNN weight sharing + bLSTM	97.81% ± 1.25%	94.14% ± 1.26%
2D CNN per sensor + bLSTM	99.13% ± 0.53%	95.78% ± 0.82%
2D CNN all signals + bLSTM	98.66% ± 0.59%	94.63% ± 0.82%
2D CNN combined + bLSTM	99.25% ± 0.49%	96.22% ± 0.36%
LSTM	92.33% ± 2.38%	76.49% ± 2.32%
bLSTM	97.27% ± 1.21%	83.73% ± 2.67%

Table 6.1 shows the mean and standard deviation prediction accuracies for this experiment. 5 runs were performed. In general terms, the models also showed good performance with small variation between runs. The comparison of the mean prediction accuracies of these models with the models that included the 5 sensors is presented in table 6.2. From this figure, it is clear that the inclusion of shanks is beneficial to the model. The data comes with more information and the models can extract more patterns from the recordings and better learn to recognize the activities. Because of this, in practically all the cases, the models have a better prediction accuracy when shanks are included. Nevertheless, the performance of the models is just slightly compromised when not including the shanks. In particular,

Table 6.2: Comparison of mean prediction accuracies between the models with and without shanks

	With shanks		Without shanks	
	Train	Test	Train	Test
1D CNN weight sharing	99,13%	94,36%	98,05%	91,01%
1D CNN per sensor	98,35%	93,92%	98,64%	93,64%
1D CNN combined	98,66%	93,97%	98,12%	92,49%
2D CNN weight sharing	98,80%	93,59%	98,21%	91,62%
2D CNN per sensor	99,32%	94,79%	98,64%	93,10%
2D CNN all signals	97,58%	92,88%	97,88%	92,71%
2D CNN combined	99,34%	95,34%	94,38%	89,64%
1D CNN weight sharing + LSTM	99,01%	96,05%	97,44%	94,25%
1D CNN per sensor + LSTM	98,31%	95,95%	97,58%	94,79%
1D CNN combined + LSTM	98,87%	96,55%	98,47%	95,51%
2D CNN weight sharing + LSTM	98,87%	96,27%	98,19%	95,12%
2D CNN per sensor + LSTM	99,32%	96,71%	98,26%	96,00%
2D CNN all signals + LSTM	98,94%	95,45%	97,46%	93,86%
2D CNN combined + LSTM	99,32%	96,71%	99,13%	96,71%
1D CNN weight sharing + bLSTM	99,08%	96,55%	98,28%	95,12%
1D CNN per sensor + bLSTM	98,66%	96,38%	97,88%	94,85%
1D CNN combined + bLSTM	99,22%	96,71%	97,79%	94,74%
2D CNN weight sharing + bLSTM	99,20%	96,22%	97,81%	94,14%
2D CNN per sensor + bLSTM	98,99%	96,11%	99,13%	95,78%
2D CNN all signals + bLSTM	98,45%	94,96%	98,66%	94,63%
2D CNN combined + bLSTM	99,29%	96,00%	99,25%	96,22%
LSTM	91,48%	77,48%	92,33%	76,49%
bLSTM	99,65%	87,07%	97,27%	83,73%

the average decrease in accuracy is of only around 1.4% points. The models without shanks obtain test accuracies of up to 96%, with even one of the models achieving the same maximum 96.71% accuracy. In terms of overfitting, the models are not significantly affected. This experiment demonstrated that it is not required to use pants that include shank sensors to build good models. The loss in performance is very low if IMUs placed on those sensors are ignored, meaning that the usage of shorts with only thigh and pelvis sensors is possible.

6.2.2. Normalization of signals

The second additional experiment that was performed was based on what is recommended to do when building machine learning: normalization. During the present work it was mentioned that one of the main ideas of the thesis was to evaluate the building of models using raw data without any filtering or normalization. However, it was stated in many cases that a process of global standardization or normalization could be used. The importance of this procedure was seen in section 6.1.1, where another dataset with a clear different scale was used to evaluate the model and the results were not as good as expected. When the models were initially built, they were also trained with locally standardized windows, but the resulting accuracies were always worse than the ones without local standardization (tables 5.1 and 5.2). By local standardization it is referred to standardization of each window by using its own mean and standard deviation. This, as explained, destroyed the relationship in scales between all the signals, making it more difficult for the model to recognize each activity.

Instead of that, a global normalization was proposed in this experiment. By doing global normalization, a per-recording normalization was done instead of a per-window one. This was done by normalizing all the windows of a recording with the values of the whole recording instead of using the local values of the window. This is similar to what is done in image recognition tasks, where all the pixels

of an image are normalized with respect to the global values of that image. This procedure normalizes the values of the signals while maintaining the relative scales between sensors. Since the signals have both positive and negative values and it is desired to keep information about their positiveness and negativeness (positive or negative speeds and accelerations have important meanings in terms of limb movement), instead of using a $[0, 1]$ or $[-1, 1]$ normalization, the values were normalized to have maximum absolute value equal to 1 (take for example a signal with maximum value of 100 and minimum value of -5. If $[-1, 1]$ normalization is applied, -5 will be transformed to -1 and 100 to 1, losing the valuable information that the original maximum value was much larger in magnitude than the minimum). To do this normalization, we find, across all the values of the signal, the value with maximum absolute value and then all the values of the signal are divided by that absolute value. The resulting series has the same shape and structure as the original, but all its values lie between -1 and 1 without losing the positive-negative distribution and relationship. This type of normalization was applied on each one of the signals and the models were trained with these new normalized windows. To have a fair comparison, all the models were trained with the same architectures, parameters, and algorithms as the ones previously built. The only thing that had to be changed, since the values were normalized and were smaller and the training happened at a different pace, was the initial learning rate of the models that had LSTM or bLSTM layers. It was set to 10 times the original value of the respective non-normalized model.

Table 6.3: Comparison of mean prediction accuracies between the original unnormalized models and the normalized ones. 5 runs

	Acc + Gyro					
	Normalized			Unnormalized		
	Train	Test	Difference	Train	Test	Difference
1D CNN weight sharing	99,93%	97,53%	2,40%	99,13%	94,36%	4,77%
1D CNN per sensor	99,34%	97,26%	2,08%	98,35%	93,92%	4,43%
1D CNN combined	99,46%	97,21%	2,25%	98,66%	93,97%	4,69%
2D CNN weight sharing	99,29%	96,44%	2,85%	98,80%	93,59%	5,21%
2D CNN per sensor	99,44%	96,55%	2,89%	99,32%	94,79%	4,53%
2D CNN all signals	99,76%	97,81%	1,95%	97,58%	92,88%	4,70%
2D CNN combined	99,44%	98,08%	1,36%	99,34%	95,34%	4,00%
1D CNN weight sharing + LSTM	97,81%	96,11%	1,70%	99,01%	96,05%	2,96%
1D CNN per sensor + LSTM	99,20%	97,10%	2,10%	98,31%	95,95%	2,36%
1D CNN combined + LSTM	98,54%	97,04%	1,50%	98,87%	96,55%	2,32%
2D CNN weight sharing + LSTM	98,54%	96,71%	1,83%	98,87%	96,27%	2,60%
2D CNN per sensor + LSTM	98,31%	96,71%	1,60%	99,32%	96,71%	2,61%
2D CNN all signals + LSTM	98,87%	97,32%	1,55%	98,94%	95,45%	3,49%
2D CNN combined + LSTM	99,08%	97,53%	1,55%	99,32%	96,71%	2,61%
1D CNN weight sharing + bLSTM	99,39%	98,03%	1,36%	99,08%	96,55%	2,53%
1D CNN per sensor + bLSTM	99,51%	97,86%	1,65%	98,66%	96,38%	2,28%
1D CNN combined + bLSTM	99,44%	98,25%	1,19%	99,22%	96,71%	2,51%
2D CNN weight sharing + bLSTM	99,39%	97,48%	1,91%	99,20%	96,22%	2,98%
2D CNN per sensor + bLSTM	99,48%	97,04%	2,44%	98,99%	96,11%	2,88%
2D CNN all signals + bLSTM	99,13%	97,26%	1,87%	98,45%	94,96%	3,49%
2D CNN combined + bLSTM	99,46%	97,32%	2,14%	99,29%	96,00%	3,29%
LSTM	63,51%	60,44%	3,07%	91,48%	77,48%	14,00%
bLSTM	80,49%	76,71%	3,78%	99,65%	87,07%	12,58%

Tables 6.3 and 6.4 show the comparison in mean and standard deviation accuracies for 5 training runs for the original unnormalized models of section 5.1 and these new normalized models. These results show that global normalization of the recordings is effective and the models perform, in almost all the cases, better. With exception of purely LSTM and bLSTM models, the prediction accuracy of the normalized models is larger in all the cases. These new normalized models improved the results on almost 2 percentage points, achieving accuracies on the test set up to 98.2%. The worst performing

Table 6.4: Comparison of standard deviation of the prediction accuracies between the original unnormalized models and the normalized ones. 5 runs

	Acc + Gyro			
	Normalized		Unnormalized	
	Train	Test	Train	Test
1D CNN weight sharing	0,09%	0,87%	0,81%	1,33%
1D CNN per sensor	0,41%	0,74%	1,68%	1,78%
1D CNN combined	0,44%	0,70%	0,33%	1,48%
2D CNN weight sharing	0,20%	0,81%	0,49%	1,60%
2D CNN per sensor	0,29%	0,37%	0,46%	1,42%
2D CNN all signals	0,15%	0,67%	0,87%	1,38%
2D CNN combined	0,55%	0,35%	0,44%	1,42%
1D CNN weight sharing + LSTM	0,79%	0,56%	0,52%	1,21%
1D CNN per sensor + LSTM	0,58%	1,44%	1,22%	0,87%
1D CNN combined + LSTM	0,54%	0,56%	0,62%	1,60%
2D CNN weight sharing + LSTM	0,50%	0,65%	0,92%	1,00%
2D CNN per sensor + LSTM	0,76%	1,47%	0,39%	1,31%
2D CNN all signals + LSTM	0,75%	0,66%	0,58%	1,02%
2D CNN combined + LSTM	0,30%	0,83%	0,37%	1,05%
1D CNN weight sharing + bLSTM	0,44%	0,63%	0,34%	0,73%
1D CNN per sensor + bLSTM	0,53%	0,68%	0,60%	0,53%
1D CNN combined + bLSTM	0,27%	0,66%	0,53%	1,54%
2D CNN weight sharing + bLSTM	0,65%	0,97%	0,41%	0,96%
2D CNN per sensor + bLSTM	0,35%	1,17%	0,75%	1,10%
2D CNN all signals + bLSTM	0,55%	0,62%	0,96%	0,99%
2D CNN combined + bLSTM	0,92%	0,82%	0,36%	1,50%
LSTM	19,13%	18,19%	2,17%	3,36%
bLSTM	24,22%	20,14%	0,24%	2,25%

normalized models had accuracies close to the ones of the best performing unnormalized models: at least 96%. The difference in results between train and test datasets also show that normalized models had less overfitting across all the architectures. On the other hand, the normalized models gave more consistent results than the unnormalized ones, as seen in the standard deviation of the prediction accuracies: the new models had, in general, lower variance. In conclusion, normalizing the recordings is not required to obtain good models, but, by doing so, the results appear to be better, so it is recommended.

These models were used to evaluate the additional datasets of section 6.1 and compare the results when using normalized and unnormalized classifiers. For the normalized model, the 1D CNN combined bLSTM architecture was used for these experiments.

In figure 6.16 the evaluation of an interval of Rozemarijn's dataset using the normalized model is presented. To show the difference, the results of the same interval evaluated with the unnormalized model is shown in figure 6.17 (note that in this case it is needed to not impose a value for c_h as explained in section 6.1.1). The true label for that interval corresponds to a player sprinting, turning, then sprinting back and shooting. This done 10 times in a row. As it can be seen, the normalized model achieves much better results than the unnormalized model. Recall that Rozemarijn's dataset has a very different scale and distribution, so the normalized model take care of these differences, so the hyperparameter c_h does not need to be modified and the default parameters can be used. The normalized model effectively predicts 9 out of 10 of the shots (see that, anyway, the last shot is correctly recognized by the model in the top plot of figure 6.16). Between each one of the 10 shots the model identifies mainly sprints interrupted by a short jog. This short jog is the model's prediction of a turn. As it was explained, even if there is an implementation that allows the classifier to "recognize" other high activities, it is not optimized to do so, and it will always try to associate everything to a sprint, shot, jog,

jump or pass. In this case, the model found that the most similar activity to those turns is a jog, so it is recognized as such. Furthermore, the raw predictions made by the normalized model are not only more accurate in terms of the predicted activity, but have very high confidences, which is ideal. The unnormalized model, on the other hand, gives very low confidence on its predictions, resulting in an inaccurate activity recognition process and the need to modify the parameter c_h.

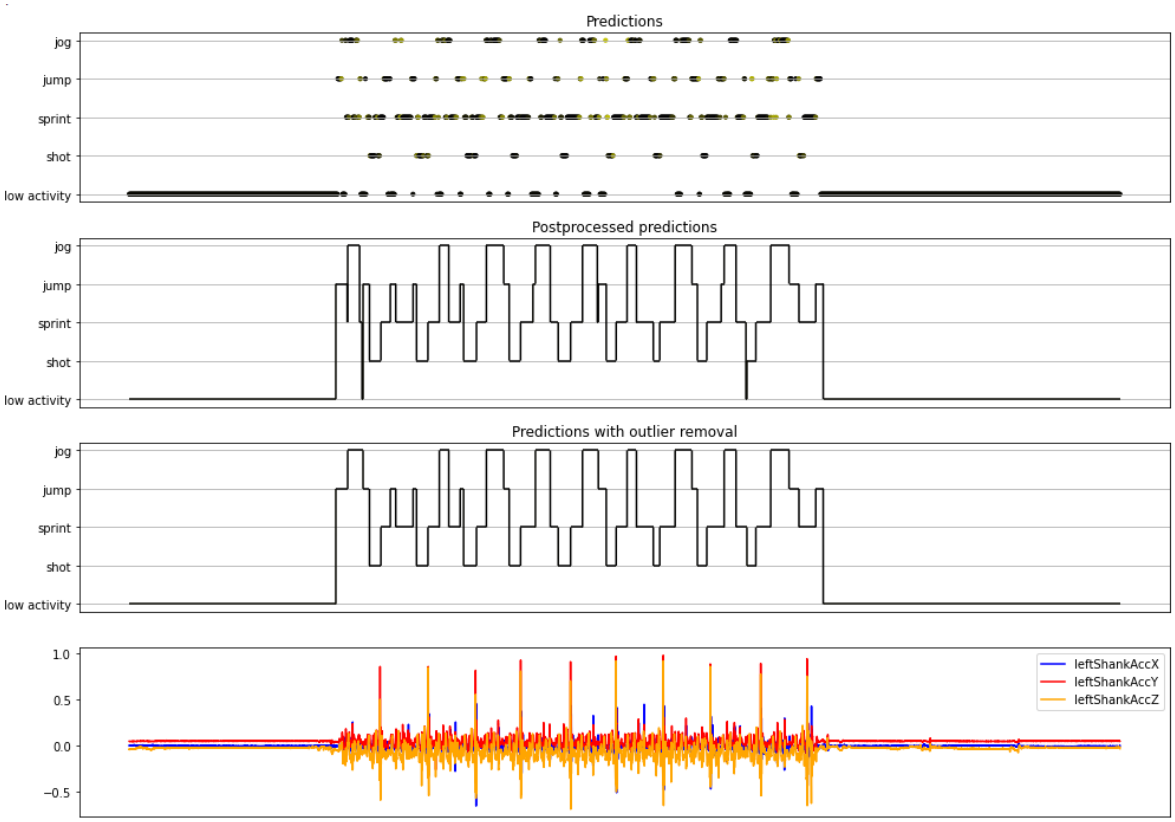


Figure 6.16: Example of predictions made on part of Rozemarijn's dataset using a normalized model. True label: sprint, turn, sprint and shot 10 times in a row

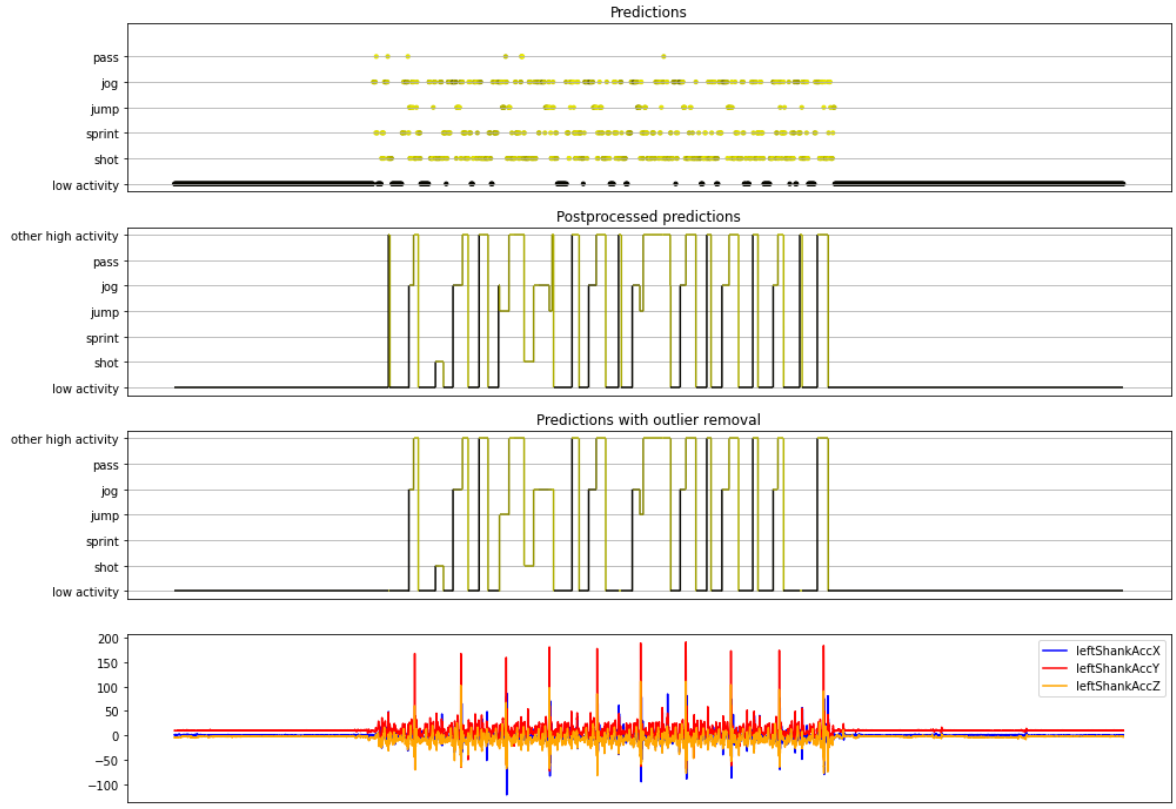


Figure 6.17: Example of predictions made on part of Rozemarijn's dataset using an unnormalized model. True label: sprint, turn, sprint and shot 10 times in a row

6.3. Conclusions and future work

This thesis explored the usage of deep learning based methods to recognize football activities in a fast and accurate manner based on measurements obtained from IMUs located on different body parts. Literature review showed that, for Human Activity Recognition, these types of techniques have been taking over the usage of traditional machine learning techniques such as kNN, decision trees and SVMs where a manual process of feature extraction is required. The majority of reviewed articles focused on deep learning models to recognize daily human activities, but this study worked with activities that are by nature more repetitive and explosive, since they happen during a football match. The possibility to build deep learning-based models on raw, not pre.processed IMU signals was one of the main goals.

A robust and end-to-end pipeline was proposed that included activity detection and isolation in order to prepare the required datasets, training of the model, evaluation of a recording via a sliding window approach, and postprocessing to obtain the final results. Although the built deep models were trained to recognize the 5 most common football activities (sprints, passes, shots, jumps, and shots), the proposed methodology can be applied to train models able to recognize additional specific activities provided that there is enough training samples of those movements.

A dynamic activity detection algorithm was proposed that was used to isolate activities in recordings from low activity periods that happen before and after. This algorithm was used to build the training and testing datasets. Internally, it uses a simple classifier that distinguishes between periodic (sprints and jogs) and explosive (shots, passes, and jumps) activities. Several deep learning architectures were built, trained, and evaluated in terms of prediction accuracy, overfitting and evaluation time. Those architectures were based on novel variations of convolutional layers acting across signals, sensors and/or combination of them with and without weight sharing. Recurrent layers (LSTMS and bidirectional LSTMs) implemented after the convolutional layers were used to further give temporal meaning to the features previously extracted by the CNNs. The proposed models obtained high accuracies (up to 96.71%) with fast evaluation times. Compared to traditional machine learning algorithms, deep learning models achieved better accuracies and faster evaluation times, showing that their usage is recommended for HAR tasks. The combination of CNNs and (b)LSTMs was beneficial and achieved better results than the usage of only CNNs. On the other hand, models with only (b)LSTMs did not generate good results, implying that convolutional operations are critical to extract relevant features. Further experimentation is encouraged over the training of the models, not only with respect to proposed architectures, but also around fine tuning of architecture and training variables, such as number of layers, number of convolutional filters, type of convolutional operations, learning rate, optimization algorithm, among others. The usage of batch normalization layers has been growing in popularity and its inclusion in the models could be also explored. On top of the deep model that recognizes the 5 types of activities, a binary threshold-based classifier was built responsible of distinguishing a high activity period (where any of the 5 activities is present) from low activity intervals (such as standing or walking). The application of this binary classifier before the deep model allowed the recognition of low activity periods without compromising the high accuracy of the main model. Additional ways to build this binary model were mentioned, such as a multi-output deep model or a binary deep model on top of the main model.

A sliding window approach for the evaluation phase was implemented following recommendations and best practices found in the literature. With this approach, a full recording can be evaluated by the built models. Postprocessing of the predictions made by the model was found to be needed, so three possible postprocessing algorithms were designed, implemented and compared, concluding that the recommended one is the proposed best score postprocessing. This algorithm acts analogous as how the non-max suppression algorithm works for object detection on computer vision applications. The best score postprocessing not only aligns the predictions to the original recording, but also cleans them from short-lived, undesired activities. An outlier removal process was implemented at the end of the pipeline to further refine the final recognized activities.

The final pipeline is dependent on several parameters that could be further investigated to address their exact influence on the final predictions, such as the sliding window length, sliding window step or overlap, minimum activity length τ for outlier removal, low activity confidence c_l, other high activity

confidence c_h, and other high activity thershold h_h. For the last three of those parameters, their recommended values were stated according to the performed experiments, but it was also shown that the final results are very sensible to them.

Experiments on additional datasets were performed to examine the model's performance on more complex data. It was evidenced that when the models are built without data normalization, different scales and distributions of the new data considerably affect the results. However, when working with IMUs calibrated in the same way so that the measurements have a consistent scale, the proposed pipeline is capable of performing good activity recognition tasks. A global data normalization process via a calibration recording is beneficial for the models, since they improved their accuracies to up to 98.2%. Applying the normalized models on these new datasets gave better results than when using the original not-normalized models. It is recommended, then, to perform a previous normalization of the signals and to further refine the binary low-vs-high activity classifier on these normalized signals. It is encouraged to further test the proposed pipelines in real-life scenarios (i.e. a football match). Furthermore, the exclusion of shank sensors just slightly penalized the models, which means that only thigh and pelvis sensors are enough to obtain reliable models.

Even if the obtained accuracies were high, it is recommended to acquire more data and retrain the models. The more data that a model has to be trained on, the better the learning will be. It is also recommended to use training data from different sources and experiments to have a better generalization. Literature review showed that HAR is an active research area in deep learning, so several open-source datasets are present online ((Roggen et al., 2010), (Kwapisz et al., 2011), and (Anguita et al., 2013), to name a few). Although the majority of them are about daily human activities, such as walking, sitting or standing, those large datasets could be used to build a base model, and then use transfer learning to fine-tune it with football data to allow it to recognize specific football activities. Transfer learning is a deep learning technique in which a base model is trained with a large, different but related dataset, and then it is used as the starting point of the training of a more specific model. Its usage has shown to be very effective when building deep models for tasks that do not have large training datasets and is expensive or time-consuming to build them. It is, then, highly recommended to explore this transfer learning approach to potentially further improve the models here presented.

Finally, it is encouraged to use this work as an input to further research on injury prevention on athletes. This thesis focused on football, but it the proposed methodology and pipeline can be applied to other sports. Recognizing and locating each one of the activities that a player does while playing a sport is just one of the steps that must be done in order to analyze his or her movements to prevent injuries. The recognized activities can be further combined with information from video recordings, biomechanical analysis, and intensity of the movements, among others, to study the prevalence and early detection of injuries.

The code of this thesis is available at:
https://github.com/rafaelcuperman/football_activity_recognition

References

Adaskevicius, R. (2014). Method for recognition of the physical activity of human being using a wearable accelerometer. *Elektronika ir Elektrotechnika*, *20*(5), 127–131.

Alom, M. Z., Taha, T. M., Yakopcic, C., Westberg, S., Sidike, P., Nasrin, M. S., Van Esesn, B. C., Awwal, A. A. S., & Asari, V. K. (2018). The history began from alexnet: A comprehensive survey on deep learning approaches. *arXiv preprint arXiv:1803.01164*.

Anguita, D., Ghio, A., Oneto, L., Parra, X., & Reyes-Ortiz, J. L. (2013). A public domain dataset for human activity recognition using smartphones. *Esann*, *3*, 3.

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

Blank, P., Hoßbach, J., Schuldhaus, D., & Eskofier, B. M. (2015). Sensor-based stroke detection and stroke type classification in table tennis. *Proceedings of the 2015 ACM International Symposium on Wearable Computers*, 93–100.

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). Smote: Synthetic minority oversampling technique. *Journal of artificial intelligence research*, *16*, 321–357.

Chen, Y., Zhong, K., Zhang, J., Sun, Q., & Zhao, X. (2016). Lstm networks for mobile human activity recognition. *2016 International conference on artificial intelligence: technologies and applications*, 50–53.

Connaghan, D., Kelly, P., O'Connor, N. E., Gaffney, M., Walsh, M., & O'Mathuna, C. (2011). Multi-sensor classification of tennis strokes. *SENSORS, 2011 IEEE*, 1437–1440.

De Vries, S. I., Engels, M., & Garre, F. G. (2011). Identification of children's activity type with accelerometer-based neural networks. *Medicine and science in sports and exercise*, *43*(10), 1994–1999.

Edel, M., & Köppe, E. (2016). Binarized-blstm-rnn based human activity recognition. *2016 International conference on indoor positioning and indoor navigation (IPIN)*, 1–7.

Fernández, A., Garcia, S., Herrera, F., & Chawla, N. V. (2018). Smote for learning from imbalanced data: Progress and challenges, marking the 15-year anniversary. *Journal of artificial intelligence research*, *61*, 863–905.

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.

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.

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

Hernández, F., Suárez, L. F., Villamizar, J., & Altuve, M. (2019). Human activity recognition on smartphones using a bidirectional lstm network. *2019 XXII Symposium on Image, Signal Processing and Artificial Vision (STSIVA)*, 1–5.

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.

Ignatov, A. (2018). Real-time human activity recognition from accelerometer data using convolutional neural networks. *Applied Soft Computing*, *62*, 915–922.

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.

K, S. (2019). Non-maximum suppression (nms). https://towardsdatascience.com/non-maximum-suppression-nms-93ce178e177c

Kaketsis, G. (2020). *Classification in football: Activity classification using sensor data in football* (Master's thesis). Delft University of Technology.

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.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
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
Kwapisz, J. R., Weiss, G. M., & Moore, S. A. (2011). Activity recognition using cell phone accelerometers. *ACM SigKDD Explorations Newsletter*, *12*(2), 74–82.
Lang, C., Steinborn, F., Steffens, O., & Lang, E. W. (2019). Applying a 1d-cnn network to electricity load forecasting. *International Conference on Time Series and Forecasting*, 205–218.
LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.
Liu, X. (2020). *Tennis stroke recognition: Stroke classification using inertial measuring unit and machine learning algorithm in tennis* (Master's thesis). Delft University of Technology.
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.
Mannini, A., & Sabatini, A. M. (2010). Machine learning methods for classifying human physical activity from on-body accelerometers. *Sensors*, *10*(2), 1154–1175.
McGaughey, G., Walters, W. P., & Goldman, B. (2016). Understanding covariate shift in model performance. *F1000Research*, *5*.
Murad, A., & Pyun, J.-Y. (2017). Deep recurrent neural networks for human activity recognition. *Sensors*, *17*(11), 2556.
Navlani, A. (2019). Neural network models in r. https://www.datacamp.com/community/tutorials/neural-network-models-r
Olah, C. (2015). Understanding lstm networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs/
Ordóñez, F. J., & Roggen, D. (2016). Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition. *Sensors*, *16*(1), 115.
Pienaar, S. W., & Malekian, R. (2019). Human activity recognition using lstm-rnn deep neural network architecture. *2019 IEEE 2nd Wireless Africa Conference (WAC)*, 1–5.
Roggen, D., Calatroni, A., Rossi, M., Holleczek, T., Förster, K., Tröster, G., Lukowicz, P., Bannach, D., Pirkl, G., Ferscha, A., et al. (2010). Collecting complex activity datasets in highly rich networked sensor environments. *2010 Seventh international conference on networked sensing systems (INSS)*, 233–240.
Schotel, R. (2019). *Monitoring local muscle load in football* (Master's thesis). Delft University of Technology.
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.
Sharma, M. (2020). A brief introduction to perceptron. https://becominghuman.ai/a-brief-introduction-to-perceptron-f3b9bade8f67
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.
Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The journal of machine learning research*, *15*(1), 1929–1958.
Stewart, M. (2019). Understanding dataset shift. https://towardsdatascience.com/understanding-dataset-shift-f2a5a262a766
Wang, L., & Liu, R. (2020). Human activity recognition based on wearable sensor using hierarchical deep lstm networks. *Circuits, Systems, and Signal Processing*, *39*(2), 837–856.
Wilmes, E. (2019). *Measuring changes in hamstring contractile strength and lower body sprinting kinematics during a simulated soccer match* (Master's thesis). Delft University of Technology.
Wolpert, D. H. (2001). The supervised learning no-free-lunch theorems. *Proceedings of the 6th Online World Conference on Soft Computing in Industrial Applications*, 1–20. https://doi.org/10.1007/978-1-4471-0123-9_3
Xia, K., Huang, J., & Wang, H. (2020). Lstm-cnn architecture for human activity recognition. *IEEE Access*, *8*, 56855–56866.

Xu, C., Chai, D., He, J., Zhang, X., & Duan, S. (2019). Innohar: A deep neural network for complex human activity recognition. *Ieee Access*, *7*, 9893–9902.

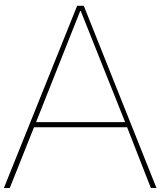
Yang, J., Nguyen, M. N., San, P. P., Li, X., & Krishnaswamy, S. (2015). Deep convolutional neural networks on multichannel time series for human activity recognition. *Ijcai*, *15*, 3995–4001.

Zebin, T., Scully, P. J., & Ozanyan, K. B. (2017). Evaluation of supervised classification algorithms for human activity recognition with inertial sensors. *2017 IEEE SENSORS*, 1–3.

Zebin, T., Sperrin, M., Peek, N., & Casson, A. J. (2018). Human activity recognition from inertial sensor time-series using batch normalized deep lstm recurrent networks. *2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 1–4.

Zhao, Y., Yang, R., Chevalier, G., Xu, X., & Zhang, Z. (2018). Deep residual bidir-lstm for human activity recognition using wearable sensors. *Mathematical Problems in Engineering*, *2018*.

Zheng, Y., Liu, Q., Chen, E., Ge, Y., & Zhao, J. L. (2014). Time series classification using multi-channels deep convolutional neural networks. *International conference on web-age information management*, 298–310.



Appendix A: Replication of results of Kaketsis, 2020

The main experiments performed in (Kaketsis, 2020) were replicated in order to have a better understanding of the process and the results obtained in that work. In his work, Kaketsis used traditional machine learning techniques to perform football activity classification.

He used the following models:

- Naive Bayes
- K-Nearest Neighbors
- Support Vector Machine
- Discriminant Analysis
- Decision Trees

Since he used a traditional machine learning approach, he had to manually select and extract features from the recordings. In particular, he chose the following features:

- Spatial features
  - Mean
  - Median
  - Standard Deviation
  - Skewness
  - Kurtosis
  - Maximum
  - Minimum
- Spectral features
  - Sum of real coefficients of FFT
  - Maximum of real coefficients of FFT

Additionally, he performed several experiments:

- Use euclidean norm of sensors or raw signals
- Use only accelerometer, only gyroscope data, or both

- Use only spatial features, or spatial and spectral features
- Recognize 4 types of activities or 7

Kaketsis used data obtained by (Wilmes, 2019), which is the same data that we used to train the deep learning-based models in this thesis. So, checking Kaketsis results and conclusions was important to have a starting point to compare with the deep learning-based models to be built. This appendix will show the results obtained by recreating Kaketsis' experiments, showing that they are consistent. For more information and detailed discussion of Kaketsis' results, refer to the original document.

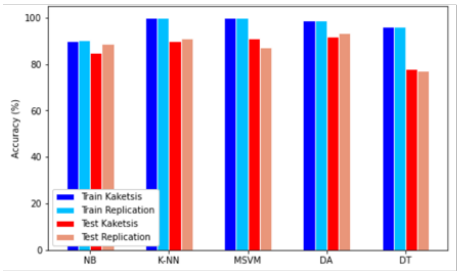
In the barplots below, the comparison between the accuracies reported in (Kaketsis, 2020) and the ones obtained by replicating the experiments are shown. The blue bars show the training accuracies and the red ones the accuracies on the test dataset. Furthermore, dark colored bars correspond to the numbers presented by Kaketsis in his work and the light colored ones, the values obtained by repeating his experiments. It is important to mention that these replications were made in Python, while the original work was done in Matlab.

The most important conclusions that were drawn after replicating Kaketsis' results are:

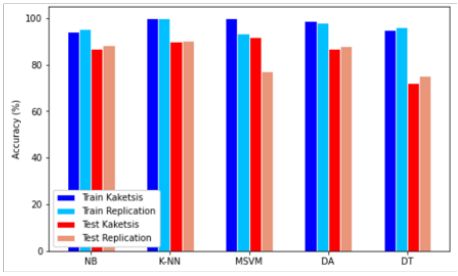
- Consistent results were obtained with respect to the ones that reported in (Kaketsis, 2020).
- No hyperparemeter tuning was made. The default parameter were used for all the models.
- Support Vector Machines do not converge in several cases.
- There is some overfitting in the great majority of the model. There is in almost all the cases a large difference between the train and test accuracies. By using deep learning-based models, it is expected to reduce this problem.
- The original work in (Kaketsis, 2020) used windows manually cropped to isolate the movements for the training phase. The proposed pipeline in this thesis will perform this activity detection phase automatically.
- Some of the experiments show promising results. Even if some overfitting is seen, accuracies on the test dataset of around 90%+ are obtained. This shows that the dataset can be used to build activity recognition models with a good performance.
- The goal of the present work will be to design, build and evaluate deep learning-based models for football activity recognition. Having now a reference on Kaketsis' results, the focus will be mainly on (but not limited to) the following items: accuracy/performance of the model, evaluation speed, low overfitting, robustness, and reduction of necessity of human interaction (no manual feature extraction and no manual activity detection to clean the training dataset).

Using euclidean norm of signals and 4 types of activities

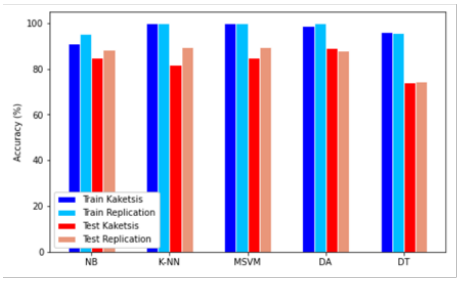
S1: Euclidean Norm of the acceleration data of the sensors using only spatial domain features



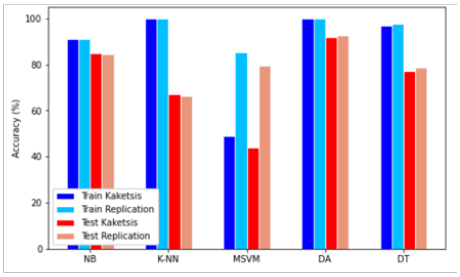
S2: Euclidean Norm of the gyroscope data of the sensors using only spatial domain features



S3: Euclidean Norm of the mixed acceleration and gyroscope data of the sensors using only spatial domain features

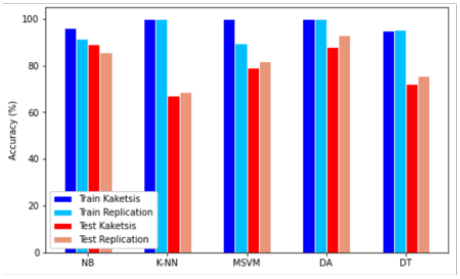


S4: Euclidean Norm of the acceleration data of the sensors using spatial and spectral domain features



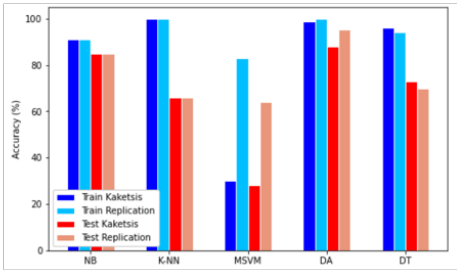
* MSVM does not converge

S5: Euclidean Norm of the gyroscope data of the sensors using spatial and spectral domain features



* MSVM does not converge

S6: Euclidean Norm of the mixed acceleration and gyroscope data of the sensors using spatial and spectral domain features

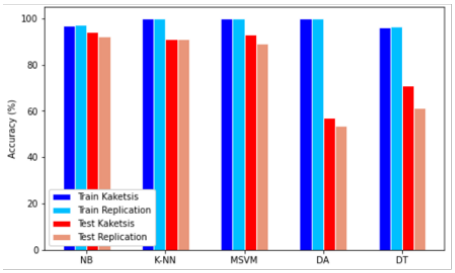


* MSVM does not converge

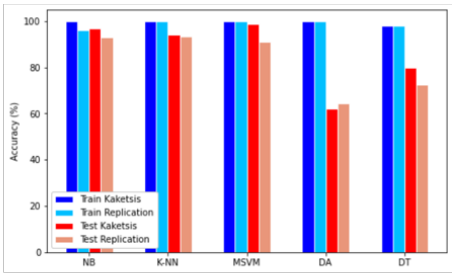
Figure A.1: Using euclidean norm of signals and 4 types of activities

Using raw signals and 4 types of activities

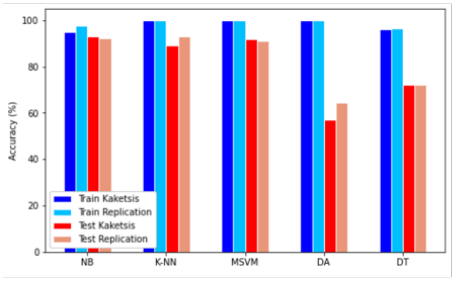
S1: Raw acceleration data of the sensors using only spatial domain features



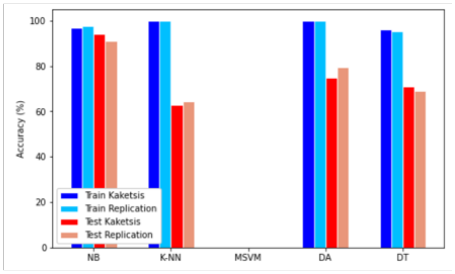
S2: Raw gyroscope data of the sensors using only spatial domain features



S3: Raw mixed acceleration and gyroscope data of the sensors using only spatial domain features

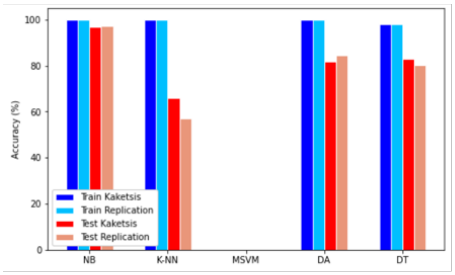


S4: Raw acceleration data of the sensors using spatial and spectral domain features



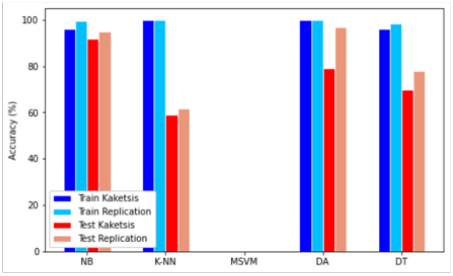
* MSVM not trained following Kaketsis' procedure

S5: Raw gyroscope data of the sensors using spatial and spectral domain features



* MSVM not trained following Kaketsis' procedure

S6: Raw mixed acceleration and gyroscope data of the sensors using spatial and spectral domain features

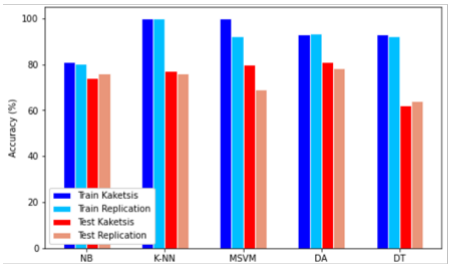


* MSVM not trained following Kaketsis' procedure

Figure A.2: Using raw signals and 4 types of activities

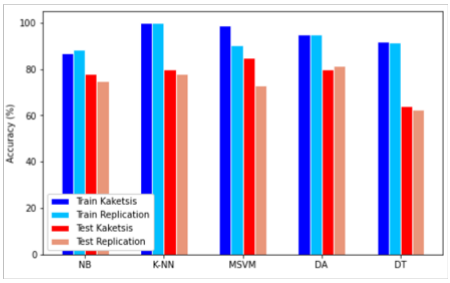
Using euclidean norm of signals and 7 types of activities

S1: Euclidean Norm of the acceleration data of the sensors using only spatial domain features

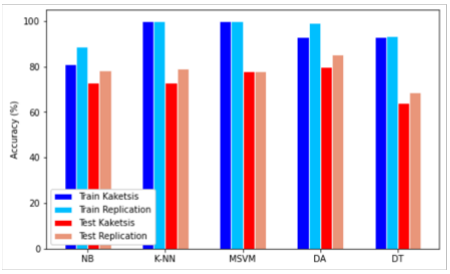


* MSVM does not converge

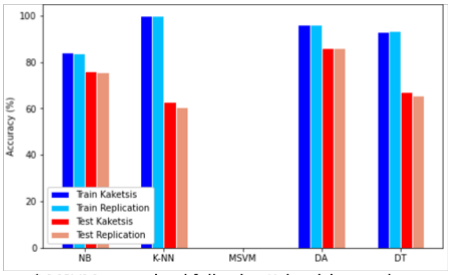
S2: Euclidean Norm of the gyroscope data of the sensors using only spatial domain features



S3: Euclidean Norm of the mixed acceleration and gyroscope data of the sensors using only spatial domain features

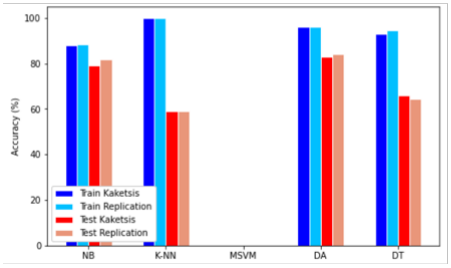


S4: Euclidean Norm of the acceleration data of the sensors using spatial and spectral domain features



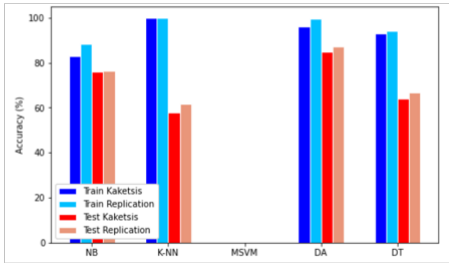
* MSVM not trained following Kaketsis' procedure

S5: Euclidean Norm of the gyroscope data of the sensors using spatial and spectral domain features



* MSVM not trained following Kaketsis' procedure

S6: Euclidean Norm of the mixed acceleration and gyroscope data of the sensors using spatial and spectral domain features

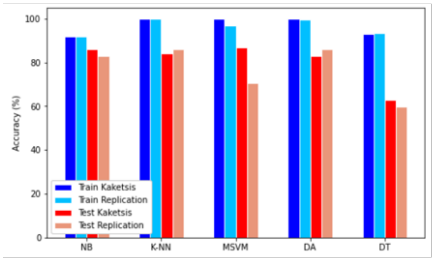


* MSVM not trained following Kaketsis' procedure

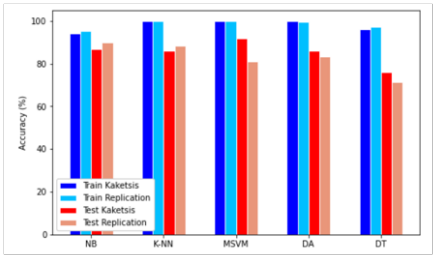
Figure A.3: Using euclidean norm of signals and 7 types of activities

Using raw signals and 7 types of activities

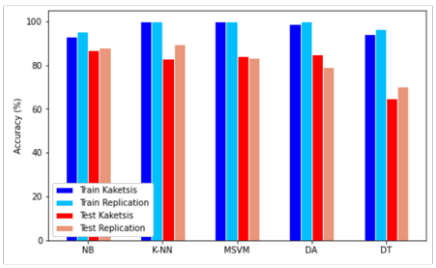
S1: Raw acceleration data of the sensors using only spatial domain features



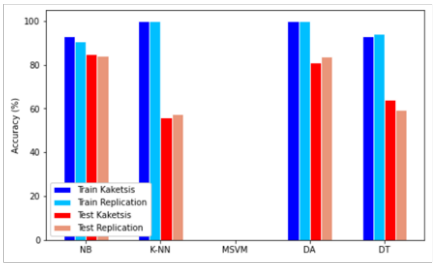
S2: Raw gyroscope data of the sensors using only spatial domain features



S3: Raw mixed acceleration and gyroscope data of the sensors using only spatial domain features

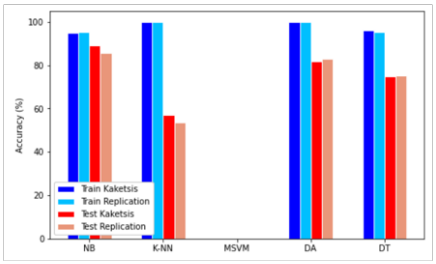


S4: Raw acceleration data of the sensors using spatial and spectral domain features



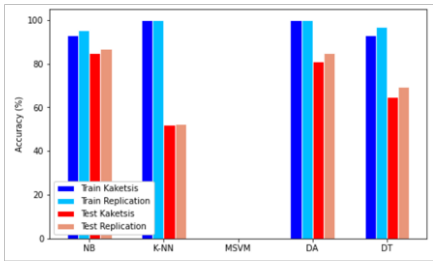
* MSVM not trained following Kaketsis' procedure

S5: Raw gyroscope data of the sensors using spatial and spectral domain features



* MSVM not trained following Kaketsis' procedure

S6: Raw mixed acceleration and gyroscope data of the sensors using spatial and spectral domain features

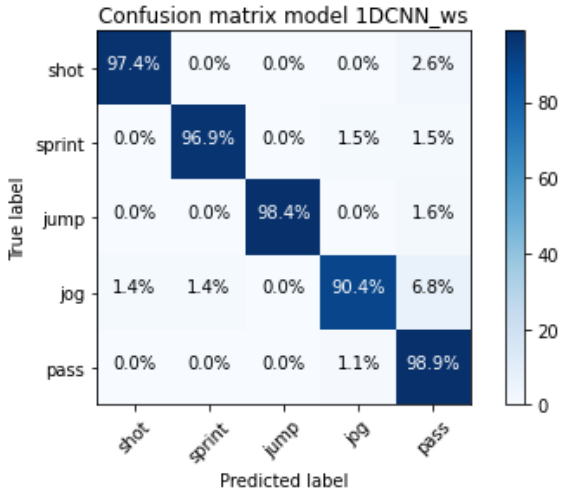


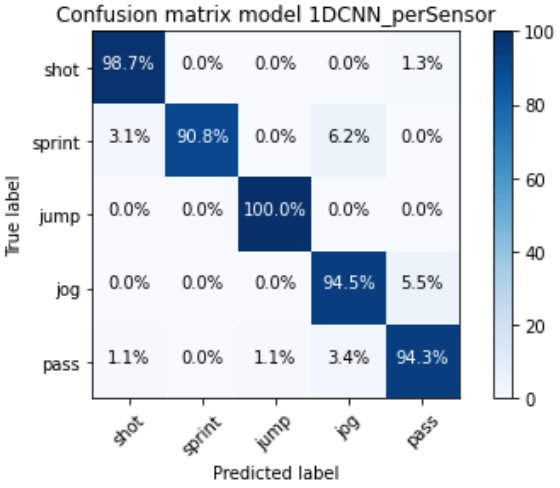
* MSVM not trained following Kaketsis' procedure

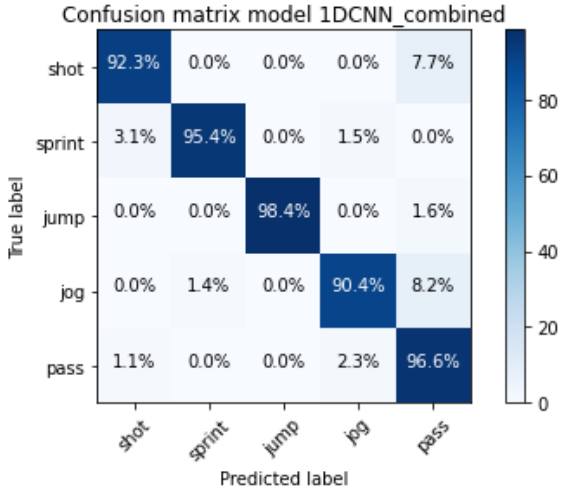
Figure A.4: Using raw signals and 7 types of activities

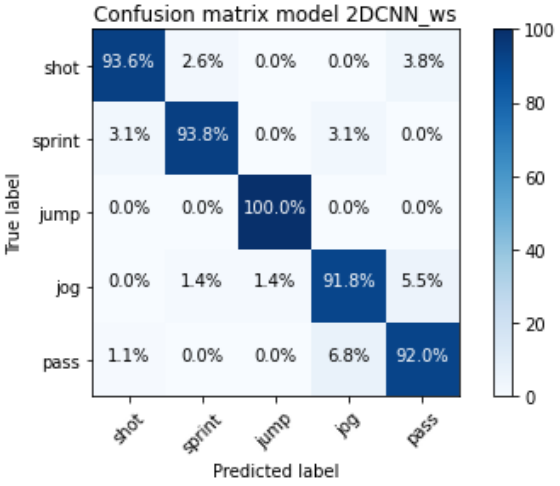
B

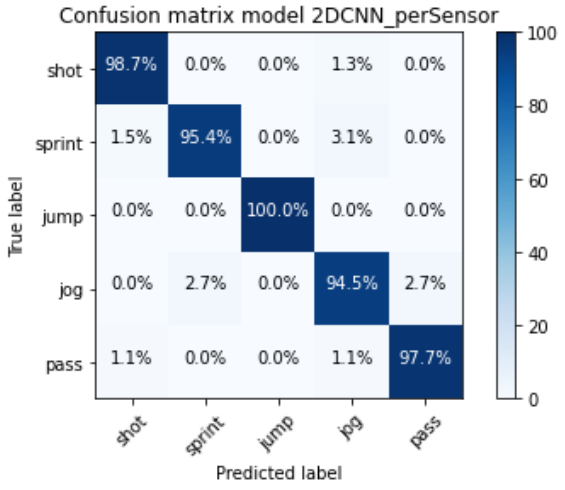
Appendix B: Confusion matrices

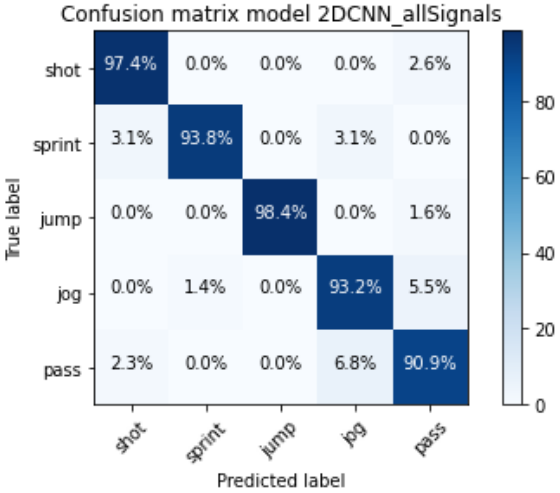

Confusion matrix model 1DCNN_ws
shot 97.4% 0.0% 0.0% 0.0% 2.6%
sprint 0.0% 96.9% 0.0% 1.5% 1.5%
jump 0.0% 0.0% 98.4% 0.0% 1.6%
jog 1.4% 1.4% 0.0% 90.4% 6.8%
pass 0.0% 0.0% 0.0% 1.1% 98.9%
shot sprint jump jog pass
True label
Predicted label
80 60 40 20 0

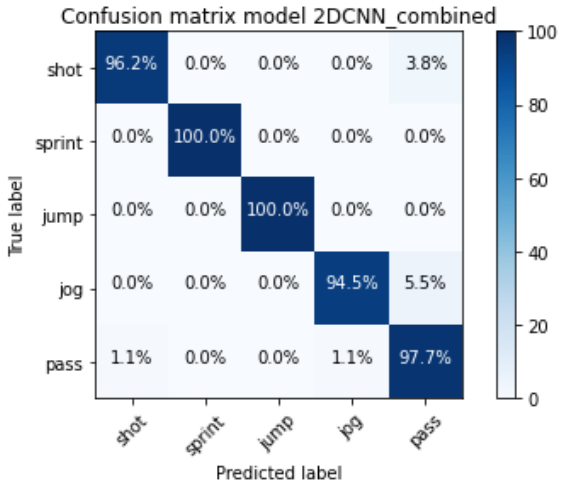

Confusion matrix model 1DCNN_perSensor
shot 98.7% 0.0% 0.0% 0.0% 1.3%
sprint 3.1% 90.8% 0.0% 6.2% 0.0%
jump 0.0% 0.0% 100.0% 0.0% 0.0%
jog 0.0% 0.0% 0.0% 94.5% 5.5%
pass 1.1% 0.0% 1.1% 3.4% 94.3%
shot sprint jump jog pass
True label
Predicted label
100 80 60 40 20 0

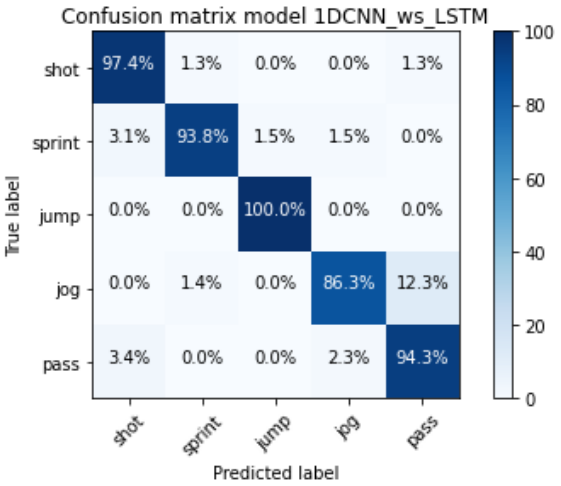

Confusion matrix model 1DCNN_combined
shot 92.3% 0.0% 0.0% 0.0% 7.7%
sprint 3.1% 95.4% 0.0% 1.5% 0.0%
jump 0.0% 0.0% 98.4% 0.0% 1.6%
jog 0.0% 1.4% 0.0% 90.4% 8.2%
pass 1.1% 0.0% 0.0% 2.3% 96.6%
shot sprint jump jog pass
True label
Predicted label
80 60 40 20 0

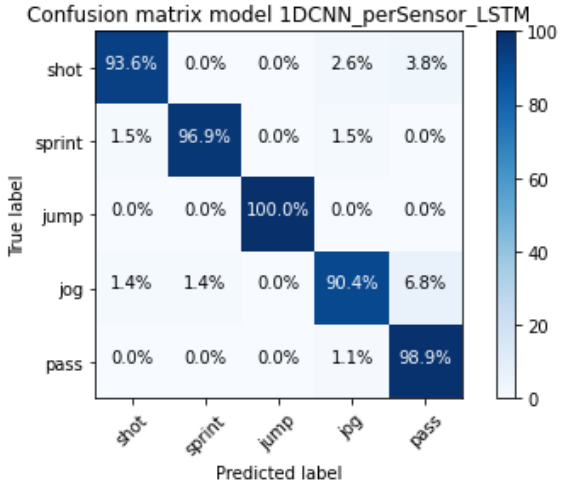

Confusion matrix model 2DCNN_ws
shot 93.6% 2.6% 0.0% 0.0% 3.8%
sprint 3.1% 93.8% 0.0% 3.1% 0.0%
jump 0.0% 0.0% 100.0% 0.0% 0.0%
jog 0.0% 1.4% 1.4% 91.8% 5.5%
pass 1.1% 0.0% 0.0% 6.8% 92.0%
shot sprint jump jog pass
True label
Predicted label
100 80 60 40 20 0

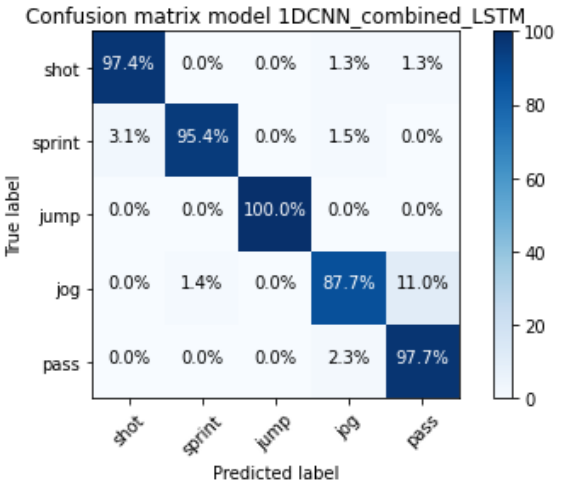

Confusion matrix model 2DCNN_perSensor
shot 98.7% 0.0% 0.0% 1.3% 0.0%
sprint 1.5% 95.4% 0.0% 3.1% 0.0%
jump 0.0% 0.0% 100.0% 0.0% 0.0%
jog 0.0% 2.7% 0.0% 94.5% 2.7%
pass 1.1% 0.0% 0.0% 1.1% 97.7%
shot sprint jump jog pass
True label
Predicted label
100 80 60 40 20 0

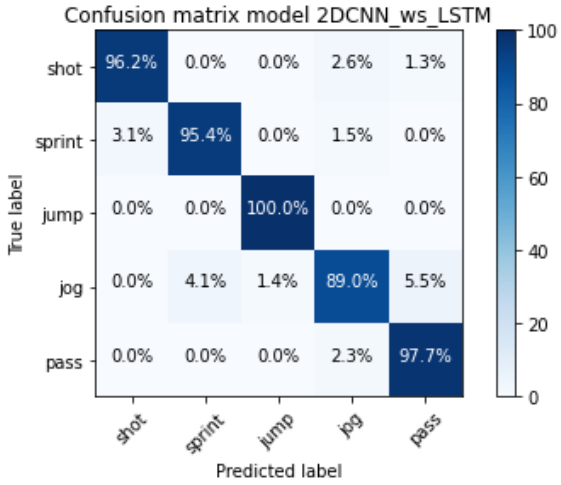

Confusion matrix model 2DCNN_allSignals
shot 97.4% 0.0% 0.0% 0.0% 2.6%
sprint 3.1% 93.8% 0.0% 3.1% 0.0%
jump 0.0% 0.0% 98.4% 0.0% 1.6%
jog 0.0% 1.4% 0.0% 93.2% 5.5%
pass 2.3% 0.0% 0.0% 6.8% 90.9%
shot sprint jump jog pass
True label
Predicted label
80 60 40 20 0

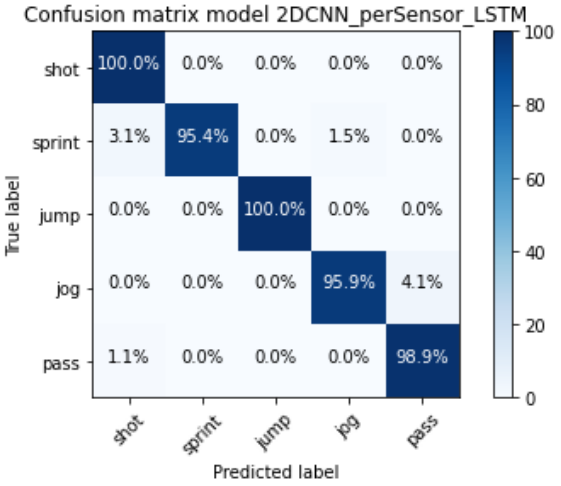

Confusion matrix model 2DCNN_combined
shot 96.2% 0.0% 0.0% 0.0% 3.8%
sprint 0.0% 100.0% 0.0% 0.0% 0.0%
jump 0.0% 0.0% 100.0% 0.0% 0.0%
jog 0.0% 0.0% 0.0% 94.5% 5.5%
pass 1.1% 0.0% 0.0% 1.1% 97.7%
shot sprint jump jog pass
True label
Predicted label
100 80 60 40 20 0

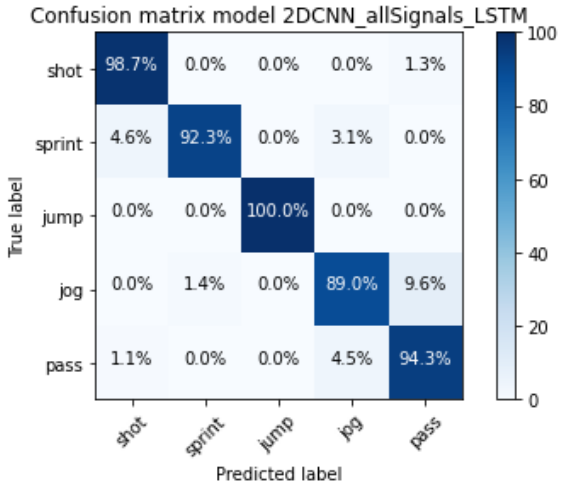

Confusion matrix model 1DCNN_ws_LSTM
shot 97.4% 1.3% 0.0% 0.0% 1.3%
sprint 3.1% 93.8% 1.5% 1.5% 0.0%
jump 0.0% 0.0% 100.0% 0.0% 0.0%
jog 0.0% 1.4% 0.0% 86.3% 12.3%
pass 3.4% 0.0% 0.0% 2.3% 94.3%
shot sprint jump jog pass
True label
Predicted label
100 80 60 40 20 0

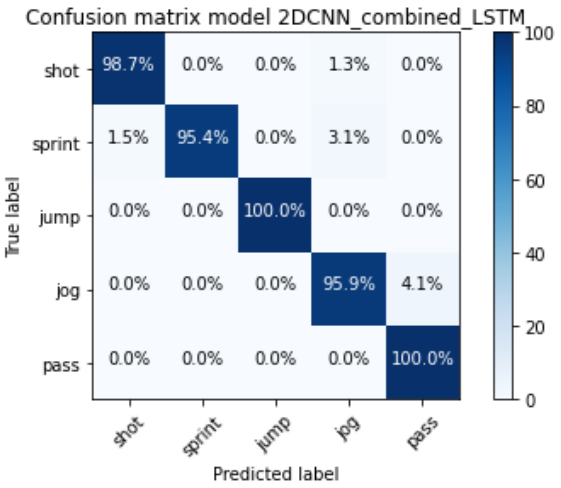

Confusion matrix model 1DCNN_perSensor_LSTM
shot 93.6% 0.0% 0.0% 2.6% 3.8%
sprint 1.5% 96.9% 0.0% 1.5% 0.0%
jump 0.0% 0.0% 100.0% 0.0% 0.0%
jog 1.4% 1.4% 0.0% 90.4% 6.8%
pass 0.0% 0.0% 0.0% 1.1% 98.9%
shot sprint jump jog pass
True label
Predicted label
100 80 60 40 20 0

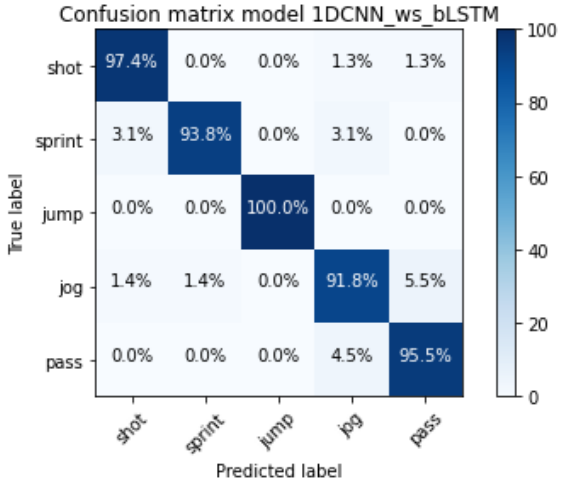

Confusion matrix model 1DCNN_combined_LSTM
shot 97.4% 0.0% 0.0% 1.3% 1.3%
sprint 3.1% 95.4% 0.0% 1.5% 0.0%
jump 0.0% 0.0% 100.0% 0.0% 0.0%
jog 0.0% 1.4% 0.0% 87.7% 11.0%
pass 0.0% 0.0% 0.0% 2.3% 97.7%
shot sprint jump jog pass
True label
Predicted label
100 80 60 40 20 0

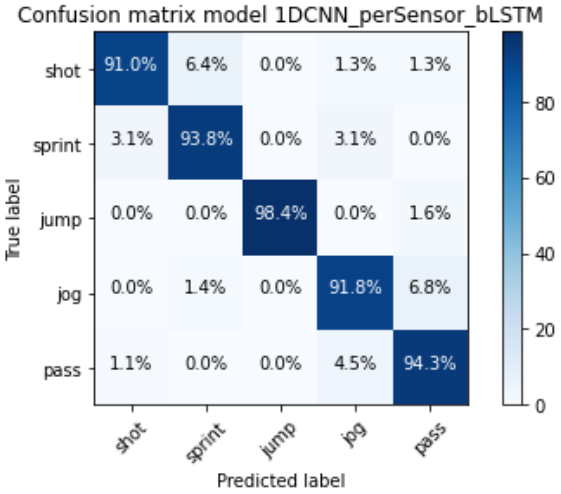

Confusion matrix model 2DCNN_ws_LSTM
shot 96.2% 0.0% 0.0% 2.6% 1.3%
sprint 3.1% 95.4% 0.0% 1.5% 0.0%
jump 0.0% 0.0% 100.0% 0.0% 0.0%
jog 0.0% 4.1% 1.4% 89.0% 5.5%
pass 0.0% 0.0% 0.0% 2.3% 97.7%
shot sprint jump jog pass
True label
Predicted label
100 80 60 40 20 0

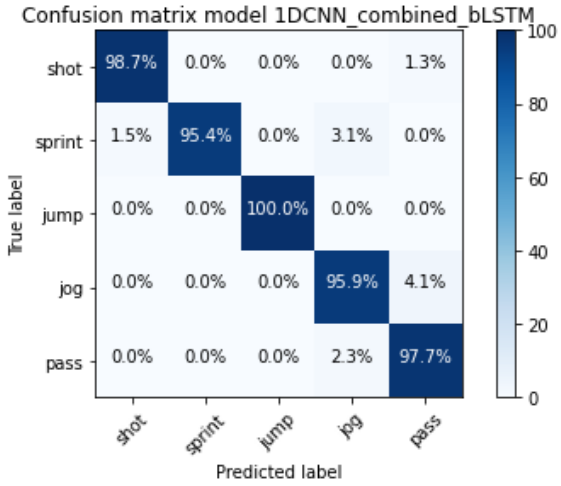

Confusion matrix model 2DCNN_perSensor_LSTM
shot 100.0% 0.0% 0.0% 0.0% 0.0%
sprint 3.1% 95.4% 0.0% 1.5% 0.0%
jump 0.0% 0.0% 100.0% 0.0% 0.0%
jog 0.0% 0.0% 0.0% 95.9% 4.1%
pass 1.1% 0.0% 0.0% 0.0% 98.9%
shot sprint jump jog pass
True label
Predicted label
100 80 60 40 20 0

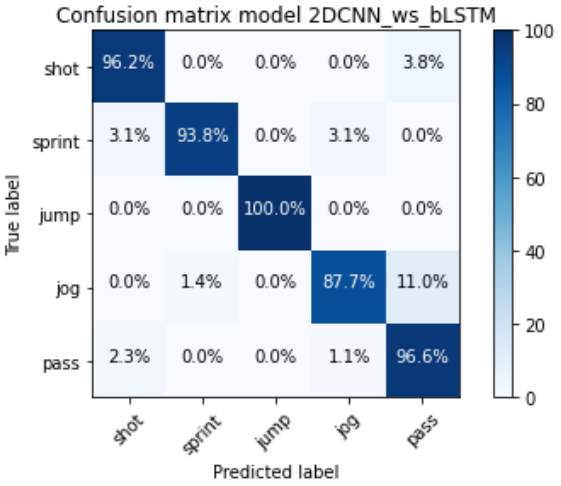

Confusion matrix model 2DCNN_allSignals_LSTM
shot 98.7% 0.0% 0.0% 0.0% 1.3%
sprint 4.6% 92.3% 0.0% 3.1% 0.0%
jump 0.0% 0.0% 100.0% 0.0% 0.0%
jog 0.0% 1.4% 0.0% 89.0% 9.6%
pass 1.1% 0.0% 0.0% 4.5% 94.3%
shot sprint jump jog pass
True label
Predicted label
100 80 60 40 20 0

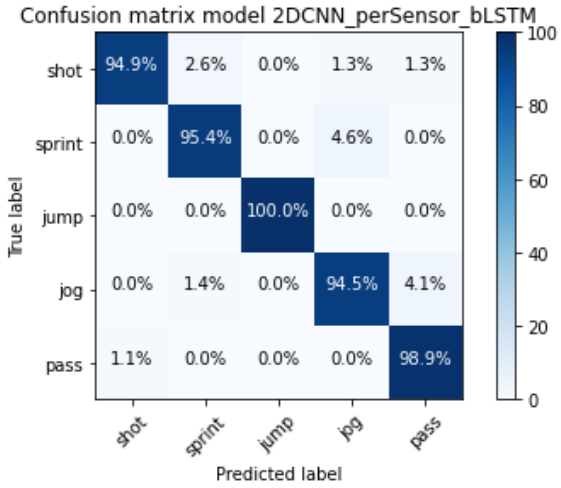

Confusion matrix model 2DCNN_combined_LSTM
shot 98.7% 0.0% 0.0% 1.3% 0.0%
sprint 1.5% 95.4% 0.0% 3.1% 0.0%
jump 0.0% 0.0% 100.0% 0.0% 0.0%
jog 0.0% 0.0% 0.0% 95.9% 4.1%
pass 0.0% 0.0% 0.0% 0.0% 100.0%
shot sprint jump jog pass
True label
Predicted label
100 80 60 40 20 0

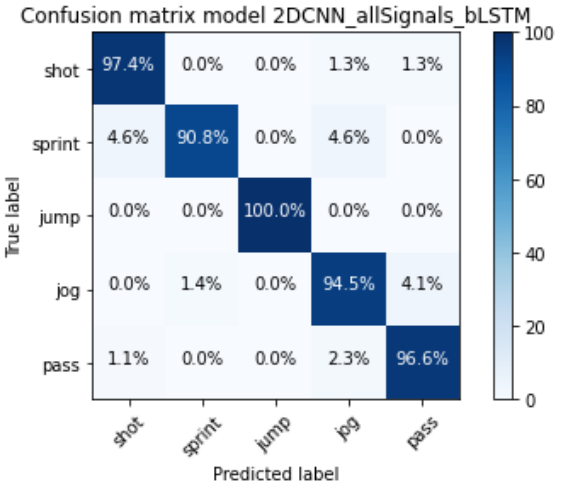

Confusion matrix model 1DCNN_ws_bLSTM
shot 97.4% 0.0% 0.0% 1.3% 1.3%
sprint 3.1% 93.8% 0.0% 3.1% 0.0%
jump 0.0% 0.0% 100.0% 0.0% 0.0%
jog 1.4% 1.4% 0.0% 91.8% 5.5%
pass 0.0% 0.0% 0.0% 4.5% 95.5%
shot sprint jump jog pass
True label
Predicted label
100 80 60 40 20 0

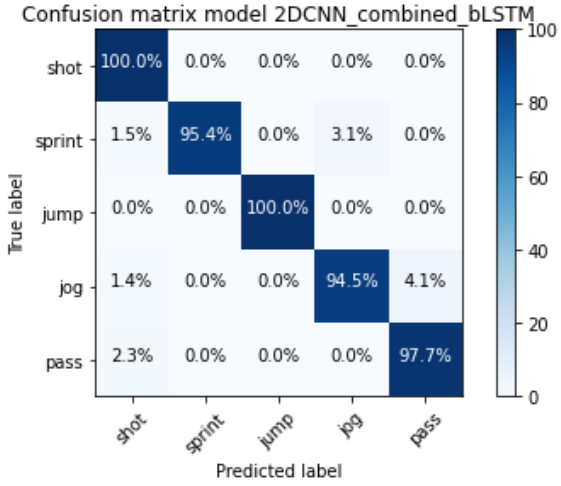

Confusion matrix model 1DCNN_perSensor_bLSTM
shot 91.0% 6.4% 0.0% 1.3% 1.3%
sprint 3.1% 93.8% 0.0% 3.1% 0.0%
jump 0.0% 0.0% 98.4% 0.0% 1.6%
jog 0.0% 1.4% 0.0% 91.8% 6.8%
pass 1.1% 0.0% 0.0% 4.5% 94.3%
shot sprint jump jog pass
True label
Predicted label
80 60 40 20 0

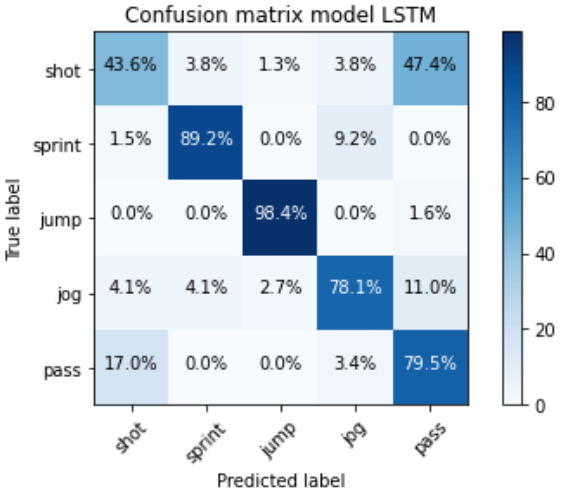

Confusion matrix model 1DCNN_combined_bLSTM
shot 98.7% 0.0% 0.0% 0.0% 1.3%
sprint 1.5% 95.4% 0.0% 3.1% 0.0%
jump 0.0% 0.0% 100.0% 0.0% 0.0%
jog 0.0% 0.0% 0.0% 95.9% 4.1%
pass 0.0% 0.0% 0.0% 2.3% 97.7%
shot sprint jump jog pass
True label
Predicted label
100 80 60 40 20 0

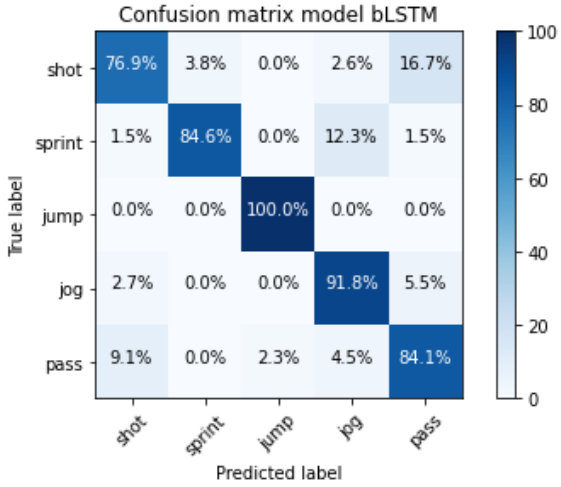

Confusion matrix model 2DCNN_ws_bLSTM
shot 96.2% 0.0% 0.0% 0.0% 3.8%
sprint 3.1% 93.8% 0.0% 3.1% 0.0%
jump 0.0% 0.0% 100.0% 0.0% 0.0%
jog 0.0% 1.4% 0.0% 87.7% 11.0%
pass 2.3% 0.0% 0.0% 1.1% 96.6%
shot sprint jump jog pass
True label
Predicted label
100 80 60 40 20 0


Confusion matrix model 2DCNN_perSensor_bLSTM
shot 94.9% 2.6% 0.0% 1.3% 1.3%
sprint 0.0% 95.4% 0.0% 4.6% 0.0%
jump 0.0% 0.0% 100.0% 0.0% 0.0%
jog 0.0% 1.4% 0.0% 94.5% 4.1%
pass 1.1% 0.0% 0.0% 0.0% 98.9%
shot sprint jump jog pass
True label
Predicted label
100 80 60 40 20 0


Confusion matrix model 2DCNN_allSignals_bLSTM
shot 97.4% 0.0% 0.0% 1.3% 1.3%
sprint 4.6% 90.8% 0.0% 4.6% 0.0%
jump 0.0% 0.0% 100.0% 0.0% 0.0%
jog 0.0% 1.4% 0.0% 94.5% 4.1%
pass 1.1% 0.0% 0.0% 2.3% 96.6%
shot sprint jump jog pass
True label
Predicted label
100 80 60 40 20 0


Confusion matrix model 2DCNN_combined_bLSTM
shot 100.0% 0.0% 0.0% 0.0% 0.0%
sprint 1.5% 95.4% 0.0% 3.1% 0.0%
jump 0.0% 0.0% 100.0% 0.0% 0.0%
jog 1.4% 0.0% 0.0% 94.5% 4.1%
pass 2.3% 0.0% 0.0% 0.0% 97.7%
shot sprint jump jog pass
True label
Predicted label
100 80 60 40 20 0


Confusion matrix model LSTM
shot 43.6% 3.8% 1.3% 3.8% 47.4%
sprint 1.5% 89.2% 0.0% 9.2% 0.0%
jump 0.0% 0.0% 98.4% 0.0% 1.6%
jog 4.1% 4.1% 2.7% 78.1% 11.0%
pass 17.0% 0.0% 0.0% 3.4% 79.5%
shot sprint jump jog pass
True label
Predicted label
80 60 40 20 0


Confusion matrix model bLSTM
shot 76.9% 3.8% 0.0% 2.6% 16.7%
sprint 1.5% 84.6% 0.0% 12.3% 1.5%
jump 0.0% 0.0% 100.0% 0.0% 0.0%
jog 2.7% 0.0% 0.0% 91.8% 5.5%
pass 9.1% 0.0% 2.3% 4.5% 84.1%
shot sprint jump jog pass
True label
Predicted label
100 80 60 40 20 0