# Classification of table tennis strokes using a wearable device and deep learning

## Universidade Fernando Pessoa


Nuno Micael

Faculdade de Ciência e Tecnologia

Universidade Fernando Pessoa

A thesis submitted to obtain the degree of

*Master*

2021

# Summary

The analysis of sports using everyday mobile devices is an area that has been increasingly explored aiming to help the user to improve in all aspects of the sport. The objective of the work proposed for this dissertation is to developed application capable of detecting strokes in table tennis using the iPhone and the Apple Watch, in which a recorded table tennis strokes data set performed by several table tennis athletes was created to help develop the application. Since the *Artificial Intelegence* area is increasingly present in our daily lives, the motivation in this work is to have a first contact with the current state of AI, the technologies available and most used in today's present, and as within the company, it was intended to begin research in this area, mainly using Apple devices, it was decided to try and create a mobile application capable of detecting strokes performed in table tennis that would work with devices capable of AI processing, in order to provide statistical data to help table tennis athletes and coaches, which can later be sell for. After a study of devices available on the apple market with the necessary capabilities for the purpose of the work, it was concluded that for this work, the devices to be used would be the iPhone (above the X model) and the Apple Watch (above the model 5). Also because there were no public table tennis data set available, a methodology was developed with the objective of capturing table tennis strokes trough motion data. The recording of motion data was done by using an application capable of recording sensors data using the apple watch who was used by each athlete on the wrist. The sensors used to record motion data were accelerometer and gyroscope, and the capture methodology was planned and overseen by coaches and athletes. From the methodology created, 2 base data sets were created. One consisting of a short interval between strokes and the second and last with a bigger interval between strokes. From these 2 data sets, 3 more were created with different pre processing configurations applied followed by a filtering and reformatting of data to the necessary format for the creation of a *Deep Learning* model. To generate a DL classifier model, two approaches were tested, one by using Create ML, and the other by using *Convolution Neural Network-Long Short Term Memory* and *Convolution Neural Network-Long Short Term Memory* architecture. To evaluate the models, statistics generated from training were saved during

model testing and creation. Create ML data set classifier models showed average performance except in one data set, with the generated classifier model having a maximum performance of 89.66% F1 score while CNN-LSTM and ConvLSTM approach generated good performance from all data set generated classifier models with the best classifier being the ConvLSTM with a 97.33% F1 score. After the creation of this same model, development of the application was performed consisting of two parts, one on the iPhone where it is possible to see the statistics and another on the Apple Watch where the ML model is executed and the stroke performed is detected being then sent to the application on the iPhone. The final step consisted on evaluation of the application during a live game scenario followed by an user rating application feedback questionnaire on athletes and coaches. Final application feedback was positive across all subjects with recommendations to the application interface and improvements to the classifier model. The live game application scenario with the generated classifier model obtained a 80% correct labelled strokes.

I would like to dedicate this thesis to my parents for providing me with the opportunity to take a degree in computer engineering which led me to where I am now, creating a master thesis.

# Acknowledgment

I would like to thank Professor Pedro Sobral and José Torres for proving me with advice and guidelines to be able to produce a good quality work.

# Contents

# List of Figures

# List of Tables

# Acronyms

**AGDT** *Automatically Generated Decision Tree*

**AI** *Artificial Intelegence*

**ANN** *Artificial Neural Network*

**CDTC** *Custom Decision Tree Classifier*

**CNN** *Convolution Neural Network*

**CNN-LSTM** *Convolution Neural Network-Long Short Term Memory*

**ConvLSTM** *Convolution Neural Network-Long Short Term Memory*

**DA** *Discriminant Analysis*

**DBC** *Deep Brain Chain*

**DCNN-LSTM** *Deep Convolutional Neural Network-Long Short Term Memory*

**DL** *Deep Learning*

**DT** *Decision Tree*

**GMM** *Gaussian Mixture Modelling*

**GMM** *Gaussian Mixture Models*

**HAR** *Human Activity Recognition*

**IMD** *Inercia Motion Data*

**IMU** *Inertia Measurement Unit*

**ISD** *Inertia Sensor Data*

**KNN** *K-Nearest Neighbors*

**LSTM** *Long Short Term Memory*

**ML** *Machine Learning*

**MVVM** *Model-View-ViewModel*

**NN** *Neural Networks*

**NNC** *Nearest Neighbor Classifier*

**RF** *Random Forest*

**RL** *Reenforced Learning*

**RNN** *Recurrent Neural Network*

**RNN-LSTM** *Recurrent Neural Network-Long Short Term Memory*

**SL** *Supervised Learning*

**SVM** *Support Vector Machines*

**TCN** *Temporal Convolutional Networks*

**TCN-LSTM** *Temporal Convolutional Networks-Long Short Term Memory*

**UL** *Unsupervised Learning*

# Chapter 1

# Introduction

In recent years, we have seen an increase in AI solutions in our daily lives. This can range from automatically compiling photos to more complex systems that can tell if a person has cancer or not. When looking at the meaning of an AI system, one can find several definitions (Marr, 2018), but the original creator of the name AI, (McCarthy et al.), chose the name because of its neutrality. In each domain, a different AI-composed system can be created, depending on the goals set. In this work, the goals were based on motion detection.

The aim of this dissertation is to predict the types of strokes performed by an athlete in table tennis sport using the sensors of smart devices. In table tennis sport, the way the ball is hit is one of the most important aspects because it is how the ball is played and can lead to winning a point. Although there are several strokes in table tennis, some of which can be seen on Fig 1.1, only a few strokes are identified in this dissertation.

Figure 1.1: Example of played strokes in a table tennis game. Forehand stroke, backhand drive, backhand short, forehand cut, forehand drive. Fig taken from (Lim et al., 2018)

Each stroke consists of a ball effect that the athlete tries to apply, but this application is always different from athlete to athlete even though the stroke performed is the same. To perform motion detection it was necessary to analyze the devices on the market, which sensors are available in each device and which do not hinder the athlete during the game. The main objective of this work is the creation of a mobile application using AI and Apple devices capable of running an ML or DL model using algorithms based on *Supervised Learning*. This project is important and relevant for starting the development of a new area within the current company and, on the other hand, developing an application that

will be a new way in which the user can review their game statistics improve the practice of the sport in question.

## 1.1 Problem

In many sport activities, stroke detection during train or competition provides important data to improve the athlete performance. However, reliable stroke detection in sport activities usually requires the use of several devices (Incel et al.) sometimes placed in parts of the body uncomfortable for the practice of the sport (Shoaib et al.). Although these types of solutions bring scientific content according to the study carried out, the continued use of these devices to obtain statistics in order to improve the practice of sport is not always easy enough for the sports practitioner to do or acquire. Another problem is that motion detection is not performed on the device itself and has to be sent to a cloud service, making the solution impossible to be used offline and data privacy is not guarantee. On the other hand, the use of AI to calculate the prediction of the type of stroke takes into account several factors such as: What is the best option for creating the ML model (*Supervised Learning*, *Unsupervised Learning* or *Reenforced Learning* ) and, within these options, choose an algorithm that will provide the greatest model performance. Since the realization of this work has as requirements the use of Apple devices and Apple provided frameworks, taking these factors into account, it is also necessary to verify if the device to be chosen has processing capacity for the *Machine Learning* model that will be created and what tools are already available within Apple developers work environment. To use ML a dataset is also a fundamental requirement. In case of having a public table tennis stroke dataset, a research on what type of data was captured is necessary to see it if it contains the information needed for this work or otherwise the need to create our own dataset. Data filtering is also necessary in order to respect the requirements of the approach's taken to create a DL model capable of predicting the practiced stroke. Since the objective is to create a mobile application capable of detecting table tennis strokes using Apple developer tools, it will also be necessary to learn new programming languages (Swift, SwiftUI and R). In this context, this project aims to facilitate the use of this type of device for the user, and make it possible for them to be highly effective in identifying the strokes performed and providing extra health statistics.

## 1.2 Motivation

The motivation for this topic can be divided into personal and corporate reasons.

On the personal level, the fact that I saw AI as a reference in my growth phase as a human being and noticed an increase in AI progress every year made me take an online

course on "Elements of AI" at the University of Helsinki, which for the first time exposed me to AI intensively and made me understand what it is made of, what possibilities it offers and more. Taking this course also made me curious about new scenarios in which AI could be used, and eventually led to developing a master's thesis in which AI would also play a role. On the company side, the opportunity to do something with AI presented itself and this led to the choice of the topic as it would initiate the development of a new area within the current company where products would be developed using AI. Furthermore, this area of AI was chosen because it is an area that has been little explored during the graduation period (AI and using the iOS and WatchOS operating systems). Indeed, the proliferation of using ML, DL and *Neural Networks* has created the need to acquire some basic knowledge for the future. Knowing what solutions are available can lead you to new ways of solving future problems. The fact that Apple is providing more and more interfaces and devices capable of handling AI algorithms that do not require advanced knowledge to implement solutions, and on the other hand, applying this research to a sports game, in our case table tennis, makes the research conducted a real live test and shows future difficulties that can be found for future work.

## 1.3 Objectives

The following work proposes to solve three distinct objectives: learn the current state of AI solutions available in the current market, the contribution of a dataset with the most common strokes performed in table tennis for the scientific community. The creation of an iOS application that works together with an Apple Watch, to identify and characterize the type of stroke made by the user during the table tenis game and, in the end, be able to show the statistics of the strokes performed during the game.

## 1.4 Document Structure

This thesis consists of 6 chapters. Chapter 1 presents the topic under discussion, the problem for which a solution is to be found, the objective pursued, and the rationale for the choice of topic.

Chapter 2 presents the state of the art. This chapter includes a review of previous studies conducted in *Human Activity Recognition*, followed by an overview of motion detection. It then ends with a final overview of motion detection studies in table tennis. The following chapter is Chapter 3 and contains the methodology and creation of the datasets. You will also learn how the data was first recorded, what strokes are most important in table tennis, and how a robot helped to create the dataset. Chapter 4 deals with the creation of the classification model and implementation of the same in the mobile

application. Here, the generation of the classifier using Create ML and CNN-LSTM and is ConvLSTM presented, followed by the development of the application on iOS and WatchOS. Chapter 5 provides all the tests and results from Chapter 4, the classifier model, and the test results from the live play of the application. Finally, Chapter 6 presents the conclusions on developing the proposed model and the mobile application that uses the classifier model for predicting the gesture performed.

# Chapter 2

# Activity and stroke detection in sports

## 2.1 Introduction

This chapter presents a literature review in activity and motion detection in sports, based on the use of AI.

First, an introduction on *Human Activity Recognition* using visual and inertia data is presented. This first part contains what type of data was capture, visual or inertia, the type of sensors used and also the most widely used ML and DL algorithms. Then a more detailed review about stroke detection in sports containing what type of strokes were detected and the proposed AI based solutions for each work is illustrated. Next, a mini-resume of what consists table tennis can also be seen. Going in more specific in table tennis, works whose research was based on this sport and in areas such as statistical mapping, game visualization and game preview are also presented. Research into table tennis strokes detection is the last research presented on this chapter, including the solutions proposed, the techniques applied, and the best ML and DL performance algorithms. To conclude the chapter, a conclusion where major key characteristics of each work are mentioned and a comparative study between works in presented. For each work review, the approach's, solutions and difficulties were presented and taken in consideration for the objectives of this thesis.

## 2.2 Activity Detection using visual and inertia data

In recent years, activity recognition, performance evaluation and the provision of statistics have been implemented through systems where the athlete himself can wear one or multiple devices on a part of the body (Connaghan et al.). Various types of sensors have been used: accelerometers, gyroscopes, pressure sensors, heart rate monitors, etc, (Neville et al.), (Zeng et al.), (Ordóñez and Roggen). Users can use one or more sensors at different locations on their bodies according to the activity performed.

The author (Sensorsoscar et al., 2018) presented a comprehensive research on human activity recognition systems using devices used by the athlete himself. The authors begin by revealing a first architecture on HAR as well as the respective components that constitute a general data capture architecture (training and testing). The presented a comprehensive research on human activity recognition systems using devices used by the athlete himself. The authors begin by revealing a first architecture on HAR as well as the respective components that constitute a general data capture architecture (training and testing). The problems common to HAR are mentioned as the choice of sensors, energy consumption and then the solutions to the mentioned problems were presented. Finally, this work presented the most used techniques in HAR systems such as feature extraction and learning methods in ML, using the latest generation HAR systems as examples and performing a qualitative evaluation by presenting their advantages and disadvantages.

The creation of sports exercise goals is also fundamental to understand the physical level and the specialized techniques needed in sports (Aughey and Falloon). It is equally important to evaluate the effectiveness of programs designed to improve technique execution, as well as those that focus on injury prevention and recovery (Neville et al.).

The authors (Pärkkä et al.) suggest that automatic activity classification can be used to promote physical activities that improve health and a healthier lifestyle. In this research, the authors defined as their objective to understand how the detection of sports activities can be done, which are the main sensors to be used and finally, which types of signals and classifiers to use. It was concluded that the best sensor to use was the accelerometer and, in terms of evaluation, from the three classifiers used *Artificial Neural Network*, *Automatically Generated Decision Tree* and *Custom Decision Tree Classifier*, all reaching high levels of precision but the classifier that showed the best performance was AGDT achieving an accuracy of 86%

When looking at the evolution of ML, one may consider DL as a promising answer for the creation of HAR systems. *Convolution Neural Network* networks were used for HAR in (Zeng et al.), (Chen and Xue), with only one sensor and in (Hessen et al.) with multiple sensors. However, the limits of CNN networks prevent it from being able to deal with periods of time where the data is unknown. To deal with the dynamics during the performance of activities in a more efficient way, *Recurrent Neural Network*'s with the architecture *Long Short Term Memory* (Hochreiter and Schmidhuber) were proposed for adoption in HAR systems in (Lim et al.). In this study, the authors showed that, with the use of RNN and the LSTM architecture it was possible to obtain better results than with CNN's, and not only, but as well the common ML algorithms such as *Random Forest* and *Support Vector Machines*, for HAR systems that focus on activities such as walking, running, jumping, sitting and sleeping.

In the work (Xia et al.) the authors proposed a solution for HAR systems through the application of CNN-LSTM. The authors used 3 datasets representative of activities

but each one presented different characteristics such as the number of activities, type of activities and the number of participants. As a method of evaluating the performance of the proposed neural network, the F1 score metric was used and it was concluded that when compared to two other types of neural networks (CNN and DCNN-LSTM's, the CNN neural network with the LSTM architecture obtained an improvement in all datasets, with a maximum value of 95%.

The authors (Perš and Kovačič) introduce a computer vision system to track players in team games applied to handball. In the published article the authors describe the various methods of image processing, tracking as well as camera calibration and lens distortion correction. The final result of the system consists in the visualization of the players spatial temporal trajectories based on a technique using the color and shape of the field, this technique being the one that needed less correction by an operator.

The research in (Barshan and Yüksek) compares the various methodologies for identifying human activities conducted while wearing inertial and magnetic sensor units on the chest, arms, and legs. The developed system contained a tri-axial gyroscope, accelerometer, and magnetometer recording at 25Hz and was placed in each limb .This study classified 19 activities. The classifiers used included ANN's, the *Deep Brain Chain*, three types of DT's, GMM's and SVM's. The study concluded that both ANN's and SVM's provided the best results achiving 97.6% and 98.8% respectively, and that magnetometer data should be taken with care due to a higher probability of suffering distorting when metal surfaces are close.

The goal of the work presented by (Ghazali et al.) was to use an inertial sensor to recognize popular sports activities. Ten subjects were instructed to execute a variety of popular sports activities, including stationary, walking, jogging, sprinting, and leaping. Data was colleted by using a inertial sensor attached to the chest of subject. 10 subjects were used to perform the mentioned activities. Having all the data, feature extraction was then applied to recorded data generating a total of 24 features. To create a model capable of predicting the activity by data, what sport was being practise, the authors use the app Classification Learner App where application of the DT, DA, SVM, NNC and ensemble classifiers were applied. The author concluded that the highest performance classifier was the SVM with 91.2% of accuracy.

## 2.3 Movements and Stroke Detection

The article (Cust et al.) performed a systematic review on ML and DL on the recognition of movements performed in sports activities. This article found that to perform motion detection, data capture can be mostly performed in two ways, namely by the use of mobile devices in which it is possible to measure inertia data or through computer vision data (video). This article concluded after analyzing 52 valid studies that in data capture the

methodologies vary according to the study but in terms of the models used, the main one is based on SL, and for these models, there are 3 main algorithms that are used: SVM, CNN and finally LSTM.

In (Nguyen et al.), the authors had as objective to recognize basic basketball movements activities. The authors developed a new sensing system to record and recognize human movements in basketball. Those movements were consisted of (walking, jogging, running, and sprinting), as well as discriminate shooting executions. The sensing system contained accelerometer, gyroscope and magnetometer but only accelerometer data was used. The system recorded inertial sensor data at a frequency of 200Hz but later data recorded was down sampled to 40Hz. The sensor system was placed in both foots. Authors then performed future extraction by using the mean square and domain features. Final step was the implementation of a SVM classifier. The SVM classifier showed average performance for all classes except for the pivot movement achieving a max accuracy of 90.2%. The classifier showed the most confusion when "layupshot" or "dribbling" were executed due to their similarities.

Authors from (Hölzemann and Van Laerhoven) developed a system composed by two components. The first system was a sensor system worn around the wrist while second system is a classification approach to estimate the basketball actions. The sensor system was created by the own authors to have access to raw *Inertia Measurement Unit* data. A sliding windows of 1 second was applied on recorded data and for each windows the features extracted. Only accelerometer data recorded at 25Hz was used to extract features. The data was composed of only 3 participants. Supervised learning algorithms (KNN and RF classifier were then applied to generate a ML model. KNN showed the best accuracy with 87.6%. The author also makes a mention that similarities with dribbling was the most cause-effect for confusion.

## 2.4 Table Tennis

Table tennis is one of the most popular sports in the world, with over 300 million active participants (Sports, 2021) including nearly 40 million competitive table tennis players (Org, 2021). The growing popularity and super competitive championships of table tennis have resulted in the emergence of analytical methods to better understand game tactics and improve player performance. Table tennis involves two opposing players hitting a ball back and forth on a table using rackets. A table tennis match consists of the best 4 out of 7 games, where each game is awarded to the player who scores 11 points first with at least a two point margin. Each point is awarded to the player who wins the rally, which is defined as a period during which the ball is in play. A move usually contains several to tens of moves by both players. A move means that the table tennis racket hits the ball once and a serve is considered a special move and generally represents an advantage of a

game. Each player is asked to alternately serve two points but when the competition score in a game reaches 10-10, each player changes the serve by serving one point instead of two. Throughout the course of the existence of table tennis, many works have focused on ways to help athletes, such as:

- Statistical analysis with game preview.
- Predicting the stroke performed.
- Game style analysis.

### 2.4.1 Research in Table Tennis

In (Wu et al., 2018) the authors collaborated with data analysts to understand and characterize the sophisticated domain problem of table tennis data analysis. The authors proposed the creation of a system "the iTTVis", a new interactive table tennis visualization system that allows visual analysis to analyze and explore table tennis data. iTTVis provides a holistic view of an entire match from three main perspectives, time-oriented analytics, statistics and tactics. The proposed system with well-coordinated multiple views not only supports correlation identification through statistics but also provides detection of tactical patterns with a time line in relation to scoring time, but also allows for cross-analysis to obtain new information. The mentioned work also allows the visualization of how hitting the ball was performed and where the ball ended up in the opponent's field. The authors concluded that the impact on the ball has the greatest influence on the gain of a point.

The work (Munivrana et al.) aimed to study the hierarchical structure of the general group of technical-tactical elements used in table tennis and to assess their role (play frequency and effectiveness) in different game zones in and around the table. For this, the authors first carried out a study to detect which are the most important strokes in table tennis and then carried out a questionnaire in which they relate the type of stroke with various options: Game systems, game zones, game phases , racket tack, racket materials and basic tactical means. In conclusion, it was determined that the group of technical-tactical elements that make up the table tennis technique can be divided into three basic groups: a group of technical-practical elements (A) used in the proper preparation phase and disabling the opponent's attack ; a group of technical-tactical elements (B) used in the attack and counterattack phase and a group of technical-tactical elements (C) used in the defense phase, each group containing subgroups that further characterize the technical-tactical group. In this way, it is possible for the coach to choose which style of play is most appropriate for the player.

The article Qiao proposes a model DCNN-LSTM for real-time recognition and tracking of table tennis balls, in order to provide the audience with details of the ongoing

competition. First the authors created a dataset based on video of players hitting the ball in different ways. Feature extraction was then processed by multiple algorithms. From all the algortithms applied, DDPG showed the best performance for feature extraction with a maximum accuracy rate of 89%. For target tracking CNN's algorithms provided the best results with 93% accuracy, while for trajectory prediction the LSTM algorithm provided the best results achieving 91% accuracy and when compared to existing systems, the overall accuracy was improved by 23.17%. The authors concluded that while results were good, the proposed model could not be implemented in real life cases due to this model only being effective in simulations experiences. Also the high requirements to migrate the model make it difficult to implemented in a future system.

### 2.4.2 Stroke detection in table tennis

To perform motion detection in table tennis, there are 3 sensors that can be considered as providing the best data for this purpose. They are the accelerometer, the gyroscope and the magnetometer. In the article Connaghan et al. the authors investigated the recognition of tennis beats using a single device attached to a player's forearm during a competitive game. This article evaluated the best approach for detecting strokes using accelerometers, gyroscopes or magnetometers, which are built into the device that performs the measurements. This work concluded what's the ideal training dataset for stroke classification and that, to obtain the best effectiveness in detecting the type of strokes in table tennis, better results can be obtained using the 3 sensors in sets versus only one or or set of pairs. A data filtering method performed to identify the strokes performed by the athlete is also described, and a two-part system is then created, in which one identifies the possibility of a stroke and the other identifies which stroke occurred.

The work in (Liu et al.), body network sensors are used, in which 3 devices per athlete with sensors are used on 3 different parts of the arm. Each device contains a processing unit and a IMU processing unit containing the accelerometer, gyroscope and magnetometer. In order to detect the type of stroke, the authors follow a data collection architecture, applying a stroke detection algorithm in conjunction with sliding windows, feature extraction, feature reduction and finally classification by using the SVM algorithm. In this work, 9 players were asked to perform the following moves: Forehand Top Spin, Cut Block, Forehand Cut, Backhand Cut and Strong Ball. From these strokes, it was requested that they be executed 6 times, obtaining 270 samples in total. In the presentation of the confusion matrix it is possible to see that there are strokes in which the precision is 100% and the total precision of all strokes is 97.4% which demonstrates that only 270 samples may not be enough for the creation of an ML model and that there may be "overfit". On the other hand, a comparison is also carried out between the number of devices to take into account in the creation for classification in which it was possible to verify that

with the 3 devices the best results were obtained. It should also be noted that while the authors mention the strokes detected in the article are the most common in table tennis, the authors do not mention any citation that proves the statement.

In the second work (Lim et al.) the authors developed a system to help coaches to support the practice of table tennis through the use of the LSTM algorithm for processing time series data with large dimensions together with the use of a spatial neuronal model. Three sensor modules containing accelerometers and gyroscopes were used, which were glued to the arms of 1 athlete and 1 coach. Data was then transmitted to a Raspberry Pi 3. The strokes requested from the athletes were Forehand stroke, backhand drive, backhand short, forehand cut, forehand drive, and were captured with a frequency of 5Hz during 5.4s, 10 strokes per athlete and a coach, 7 for training and 3 for testing were recorded. 1260 useful samples were obtained (2 persons x 5 skills x 7 hits x 3 axes x 2 sensors x 3 modules), being the same ones used for training the neural network. The authors obtained a RNN-LSTM neuronal model capable of identifying whether the player was a coach or a player based on the stroke performed, with an F1 score of 93%.

In (Kulkarni and Shenoy) a new method for capturing visual table tennis data and performing stroke detection and classification is presented. For this the authors built a video dataset by using a Raspberry Pi containing video cameras on the game table leaning against a net on each side of the field, as well as a vibration sensor in each side. Whenever the sensor detected vibration due to a ball hitting the table, the video camera was turned on and the stroke made by the player recorded. In this work 15 strokes were captured: topspin, block, push, flick and lob, each of them in three variants (backhand, forehand and forehand flat). For each stroke, two players were used, where they both performed the same stroke while making the ball remain at play without interruption. In total 22,111 valid samples were acquired. Due to the fact that each player can contain different heights and structures the authors used a player detection model followed by a 2d human body pose estimation model (HRNet) and then the pose was feed into the process of creating a ML and DL model. To obtain the player's pose, the authors used only 4 points (the wrist, elbow, and the two shoulders) for motion detection, thus obtaining 8 features $x, y * 4$. For creation of the model, two types of approach were used, ML and DL. When compared, the authors found that in ML the best model was SVM with an accuracy of 98.37% while in DL the best model was TCN with 99.37% of precision, and this value was also obtained using TCN-LSTM but was not the optimal choice due to this model requiring more training time.

The last article (Blank et al.) presents a system for detecting and classifying strokes in table tennis using inertia measuring devices. In this article the authors used a miPod sensor containing an accelerometer and gyroscope placed on the racket handle and recorded data at 10000Hz per second. To obtain data, 10 athletes were asked to perform 4 strokes, drive, push, block and topspin in their variations for 8 minutes, 1 minute for each stroke.

For motion detection the authors used an algorithm to detect peaks of the acceleration signal obtained by the sensors, with the motion detection interval defined at 1s. After obtaining all the intervals where a stroke occurred, for each of them and for each of the sensor axis, the generic time features were calculated based on (average, standard deviation, asymmetry) and on signal characteristics (minimum, maximum, energy, median). For classification only ML models were used by the authors. The authors concluded that the model with the best efficiency was SVM with 96.7% accuracy.

## 2.5 Conclusion

In order to predict an activity, a movement or a stroke in sports, most studies try and use one or multiple wearable devices capable of capturing IMD that contain one or more sensors on the athlete. As most studies have shown, the most used sensors are accelerometer, gyroscope and a magnetometer. While some studies also use video data in order to detect activities, most studies focus on IMU. Most used devices, are placed in locations on the athlete where the most significant changes in the sensors have an higher impact such as hands (for basketball and tennis) or legs (running, walking). Data capture is then done, by asking the athletes to repeat the activity or stroke during a period of time or a number of times. Most of the studies use a number of athletes relative low creating a dataset who samples could be considered not representative on a global scale. Low samples dataset can highly influence final results. After data capture most studies try and use ML or a DL approaches to create solutions meeting their objectives. All of them used SL in order to create a model capable of receiving a input(*Inertia Sensor Data* and predict the activity/stroke. To prepare the data to create one of this two models, pre-processing consists on applying noise removal filters and creating a sliding windows with 50% slide or by not applying any filter due to how the data was capture or algorithm applied. Following pre-processing, data feature extraction can be manual created or algorithms such LSTM already take care of extracting features by themselves. Features are then feed into the ML and DL algorithms. While some studies try and use both types of learning, most studies use DL where the focus was to prove that a certain algorithm worked better then a previous algorithm in a concrete example. When looking for results both types of learning actually show good results. On ML the best algorithm can be considered the SVM while on DL, CNN-LSTM and RNN-LSTM can be considered the best options. Looking at studies focused on stroke detection in table tennis, the same approaches written above are used with some minor differences. Most studies, during pre-processing use a peak detection algorithm in the sensors in order to detect when a stroke has occurred. Also when creating datasets, three out of five studies mentioned above have a low number of athletes participating, a low recording session time, and the number of sequences asked to each athlete can be considered low. This generates a relative small number of samples

to be used by the mentioned classifier models which can induce over fitting or when using the model, a low real-life precision. By reviewing the stated studies above one could conclude that the most important strokes in table tennis are drive, push, block and topspin and their two variants (forehand and backhand) but in none of the mentioned studies related in predicting tennis stroke, explains the methodology behind the reasoning of using this strokes. Another point is how the systems created by this studies can be intrusive for the players such as requiring 3 devices on a players arm or a applying a device into the athletes racket grip. Also, all studies have created their own device, where data is captured visually or from ISD implying that its just a one time only device created to help achieve the objectives of the study and not something an athlete can gather with ease and use it daily or release the system to be commercialized in a global scale. All studies establish the strokes that they were going to try and detect, but none of them tries to detect a rest phase, where no strokes are being made, blocking the possibility of applying any system developed to a real game due to the fact that the player while resting, the sensor data generated from this action can be labeled as another stroke that most reassembles the representation of a rest movement. While a pick detection algorithm, could be a first solution to this problem, a simple fast gesture by the player as celebrating or preparing himself to start the game could bypass this system and invalidate the statistics provided by the system in play. When searching for any activity/strokes detection or more specific table tennis stroke detection using create ml or core ml no studies have been found in this area. From research done on this studies and also public dataset repositories, none of the datasets used on the mentioned studies are public which implies an extra step to develop in this master dissertation. Table 2.1 shows the main characteristics of each study including classification metrics for each system or model proposed.

Having as reference the points above mentioned, this master thesis tries to pick up on the work done and implement a DL model into a device which can be acquired with relatively easy by the athlete in order to create a commercial application capable of giving statistics to coaches and athletes and also provide a quality dataset on table tennis strokes.

| Studies | Type of device | Sensors Used | Samples | Approach | Best algorithm | Precision | F1 score |
|---|---|---|---|---|---|---|---|
| Table Tennis Stroke Recognition Based on Body Sensor Network | Body sensor networks | Accelerometer, Gyroscope, Magnetometer | 270 | Machine Learning | SVM | 97.41% | 97.4% |
| LSTM-Guided Coaching Assistant for Table Tennis Practice | 3 IMU sensors modules(MPU-9150S) | Accelerometer, Gyroscope | 1260 | Deep Learning | RNN-LSTM | 93% | 86.3% |
| Table Tennis Stroke Recognition Using Two-Dimensional Human Pose Estimation | Raspberry Pi with cameras | Vibration sensor, Video Camera | 22111 | Machine Learning and Deep Learning | TCN | 99.37% | ? |
| Sensor-based stroke detection and stroke type classification in table tennis | miPod | Accelerometer, Gyroscope | 1982 | Machine Learning | SVM | 95.7% | 96.9% |

Table 2.1: Characteristics of each developed system/model in table tennis

# Chapter 3

# Methodology and Dataset Creation

Having no datasets available, a methodology was developed to record and create a dataset whose data represented table tennis strokes performed by athletes.

## 3.1 Data Capture

In this chapter, the processes to acquire a table tennis dataset is explained. First the steps to find which strokes are the most important in terms of statistics for the athletes and coaches is described. Secondly, the process on how data was recorded by using the apple watch together with an application capable of recording accelerometer, gyroscope, manometer and altimeter is explained in detail. As a third step, the use of a robot on helping generating motion data by the athlete, the methodology behind it and how it helped to create labeled data is also described. Finally pre-processing of data by applying filters and a stroke detector algorithm using peak accelerometer velocity in order to find where the stroke has occurred in the R language, together with creating different types of pre-process datasets based on the two available raw datasets is presented in the end of this chapter

## 3.2 Most important strokes in table tennis

As mentioned in chapter 2, no public datasets related to table tennis were found when searching for similar works in table tennis stroke detection, and with that in mind, a first approach was planned to find which strokes were the most import for coaches and athletes that they could consider a benefit getting statistics about in a training or a game session. The first step was contacting the Portugal Table Ténis Federation asking cooperation between a coach or athlete could be giving in order to present the objective of this thesis and schedule a further meeting in order to establish cooperation and talk a bit better about the project. The contact given was from Portugal Coordinator of National Youth Teams

which is also a coach. After a meeting, full cooperation between the Vila Nova de Gaia training center in which the Coordinator of National Youth Teams is responsible and also the home for some top and international players train and this project was made. Further more, contact with 2 other coaches responsible by the Valongo Table Tenis Center and the Gondomar Training Center was achieved, providing a good basis for future steps. In order to find which strokes were actually important, a questionnaire was made available to be answer by the athletes and coaches of each of the training centers and if possible more. The questionnaire consisted of 6 questions:

- What class the subject belong to, player or a coach.
- If statistics about strokes done during training could be considered important to have.
- What level of competition the athletes and coaches train or belong to.
- List of strokes that could be important for player and coaches.
- If stroke acceleration and heart rate could also be important for athletes or coaches.
- If any statistic that was not include in the questionnaire could also be use full for athletes or coaches.

When creating the questionnaire and choosing the list of strokes to be selected, it was taken in consideration that, since the 5 table tennis studies mentioned in the state of art, each one had chosen its strokes without any methodology behind it, a base approach was created by joining all of the strokes to find which were actually the most used in those studies. The first list of strokes consisted on Top Spin, block, push, flick, lob, smash, drive, all of them in 2 variants, forehand and backhand. After consulting with the coaches and ask them for their opinion on the list, it was concluded that the list needed to be updated. In conclusion, the final list consisted of top spin, flip, block, service, drive. This stroke list contained the strokes that the coaches mentioned as the most played during a game and also the ones whose statistics could be help full when training an athlete. An option mentioning all possible strokes that could be detected in table tennis was also included in the list.

Question number 4 was added by advice from the coaches when asked about what could also be a good statistic for them to see on a player during training. With the questionnaire finalized, a google form was created with the questions mentioned above and the URL distributed around the 3 coaches to be distributed again with their respect contacts,players and training centers. After 2 weeks 19 answers were received. The number of athletes and coach's who answered can be seen on figure 3.1

On question number 2, only one answer was negative when asked if statistics about the strokes could be considered important to have. Fig 3.2 shows the data.

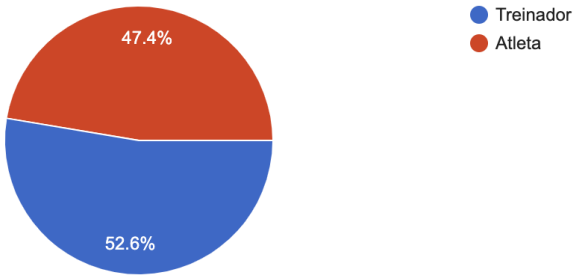



Figure 3.1: Percentage of answers between coaches and athletes

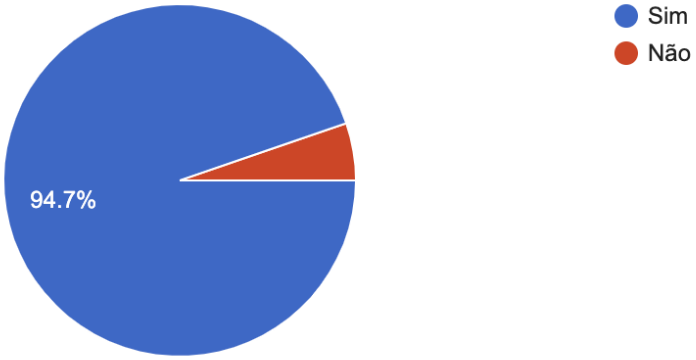



Figure 3.2: Percentage of answers who think stroke statistics are important in table tennis

Question number 3 in which the level of athletes and coaching was asked can be seen on Fig 3.4. The most answered level was cadets followed by junior,senior and kids.

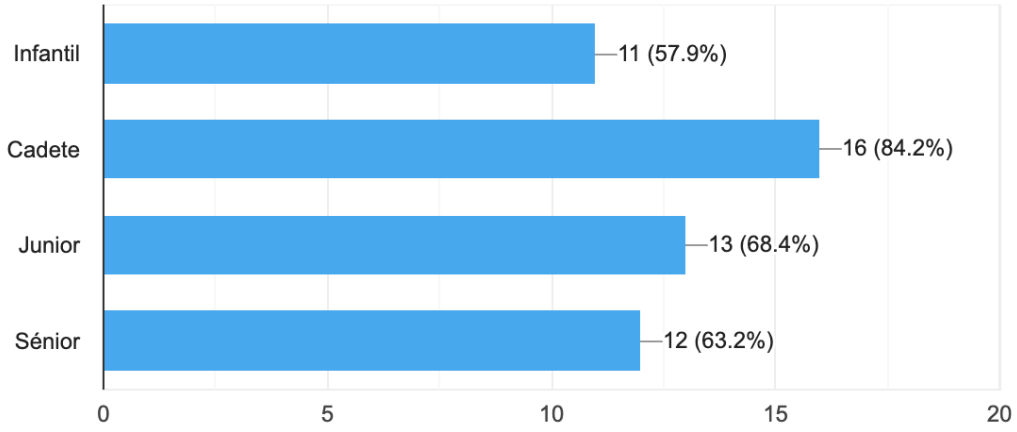



Figure 3.3: Level of athletes and coaching who answered the questionnaire.

On question number 4, the most important one, top spin was considered the most important stroke followed by block and flip with the same amount of votes, then service, any stroke that can be detected, cut and finally drive.

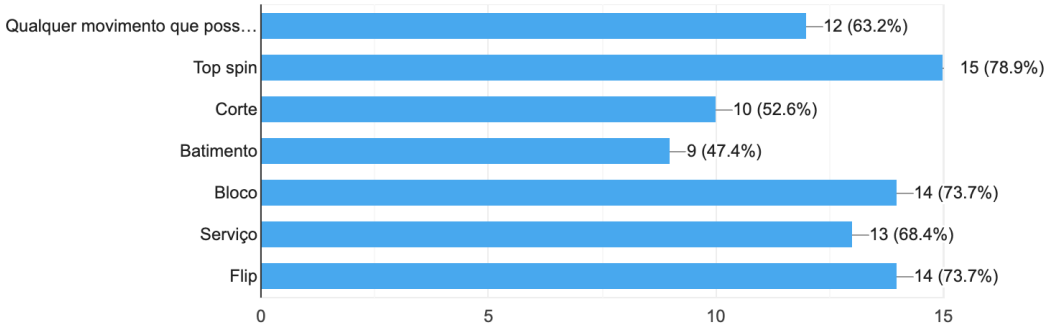



Figure 3.4: Level of athletes and coaching who answered the questionnaire.

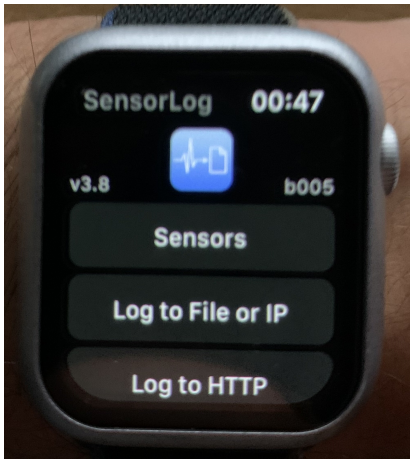
On the last question, only 7 opinions were giving about statistics that could also benefit athletes and coaches. One was invalid and the remaining six consisted on the following:

- Getting respiratory rates
- Lateral movement with the legs (x2)
- Position of the ball on the table
- Variants from the list of strokes presented in question 3 (backhand and forehand)
- Types of travel a player does when playing the game

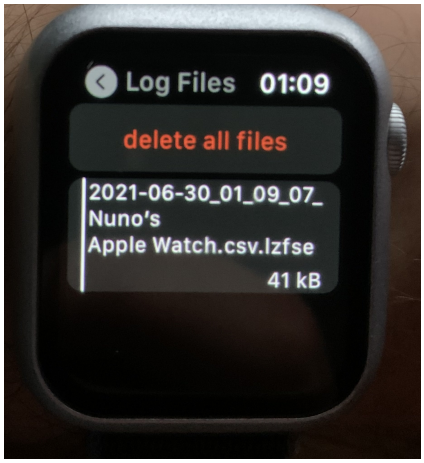
## 3.3 Recording Data

With the questionnaire done, the next step would be how could data be capture in a way capable of representing a player stroke. Since one focus of this thesis is to create a mobile application paired with a smart watch, a apple watch capable of recording ISD was chosen. The apple watch comes with 5 sensors, accelerometer, gyroscope, magnetometer, pedometer, heart rate. By using (Cust et al., 2019), (Barshan and Yüksek, 2014), (Neville et al., 2010), (Lim et al., 2018), as base for our work, it was decided to use the accelerometer, gyroscope both on 3 axis. Also on the gyroscope, attitude was also recorded. Attitude represents the orientation of a body relative to a given frame of reference. It contains 4 quaternions representing the measurement of attitude, one for each axis (x,y,z,w). This altimeter sensor data recorded was later discarded due to requiring each recording player to set a reference point before and between each recording. By using the apple watch, its possible to use the CoreMotion framework provided by apple. This framework is capable of delivering raw data of sensors or processed device motion data in which raw

data from the accelerometer and gyroscope are processed and the bias removed such as gravity, orientation relative to calibrated magnetic fields, etc. Next step was to find an app capable of saving data from the sensors and being able to export it to a computer or create our own data capture application. The second approach was a last resort method due to time available to produce this thesis. The app found that fitted perfectly for the objective is an app called SensorLog. SensorLog is an application capable of reading sensor data on iPhone, iPad and Apple Watch (Thomas, 2021). Sensor data can be sampled with up to 100Hz (depending on the version of the iOS device and fore or background mode). The sensor data (csv or JSON format) can be saved to file, streamed via TCP/UDP, send via Bluetooth, Wi-Fi and HTTP GET/POST request. Using this app on apple watch, the target data to be obtained from the application came with the following names: rotationRate, userAcceleration (accelerometer), quaternions (attitude). Sensor log interfaces can be seen on Fig 3.5 thought 3.7

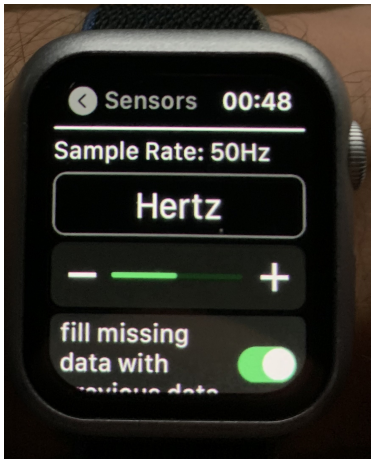



(a) SensorLog Initial Menu




(b) SensorLog Record Data File

Figure 3.5: SensorLog Interface and options part 1




(a) SensorLog Parameters UI 1


(b) SensorLog Parameters UI 2

Figure 3.6: SensorLog Interface and options part 2


(a) SensorLog recording statistics

(b) SensorLog Save File Settings

Figure 3.7: SensorLog Interface and options part 3

### 3.3.1 Frequency to record data

When choosing a frequency to record data, two requirements were defined. The frequency should be the minimum possible in order to save battery as application developed would also have to receive data at the same or less frequency to predict the athlete stroke. The frequency chosen should be able to save all characteristics represented in a athlete stroke. On (Wundersitz et al., 2015) a frequency of 100Hz was chosen but this can be inefficient as the authors discussed that 100Hz required a higher time on processing data. The authors on (Liu et al., 2019) chose a frequency of 400Hz requiring a even higher processing time and induces in a overflow of features and processing data. A frequency of 30Hz was chosen in (Xia et al., 2020) and results still indicated a high precision value. 5Hz was also reported in (Lim et al., 2018) with great precision but since the number of subjects and samples were low, more research into this frequency would be necessary and is not guarantee that a 5Hz would maintain all the details contained in a stroke. Finally, in (Lim et al., 2018) the authors use a frequency 1000Hz. This frequency would not even be possible on an apple watch due to the limitation of a max frequency of 100Hz when recording data up to one hour. A middle range was considered between: 30Hz and 60Hz. To meet the criteria above described and to find the optimal frequency, first the identification of the stroke in a graphical way would have to be determined which could ensue no data was being lost and the minimal noise introduced. Four tests were created where a robot would throw balls during 1 minute and an athlete was asked to do a Top Spin forehand. For each test a different frequency was used while recording data. Then accelerometer data was graphical represented and a cut of 10 seconds was made making two strokes visible. At frequency of 30Hz, a discontinuous graph can be seen on 3.8 containing multiple blanks and not representing what a 30Hz frequency recording should look like. A 2 second test was done with the same frequency but the results were the same. It was concluded as a

bug of the Sensorlog in which data was being recorded intermittently.

Fig 3.9, 3.10, 3.11 consisting of 40Hz, 50Hz and 60Hz captures, represent two top spin forehands strokes. Graphical analyses of the graphs showed no major difference could be seen between them and in all 3 graphics, the two corresponding strokes can be identified inclusive where the racket hit the ball. It was concluded that a 50Hz would be an optimal compromise and if any processing constrain appear in the future, data at higher frequencies can always be converted to lower frequencies by means of averaging data points.

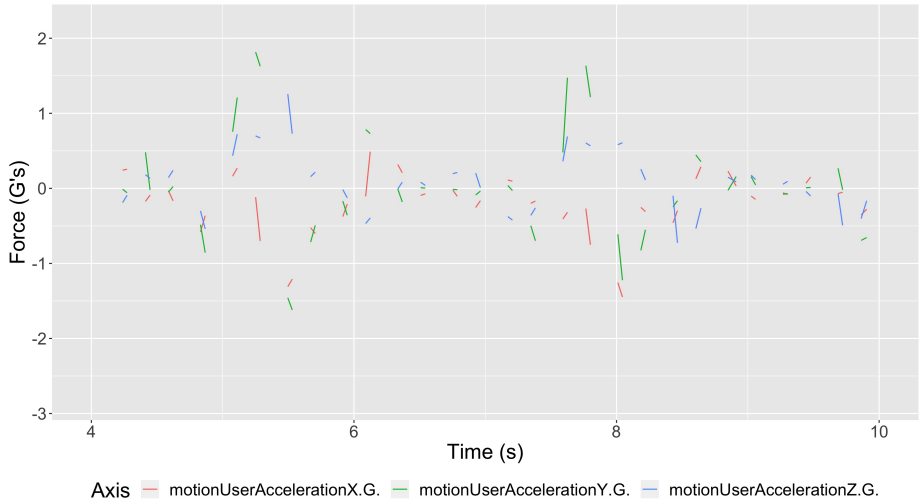



Figure 3.8: 2 Top Spin Forehand strokes recorded at 30Hz

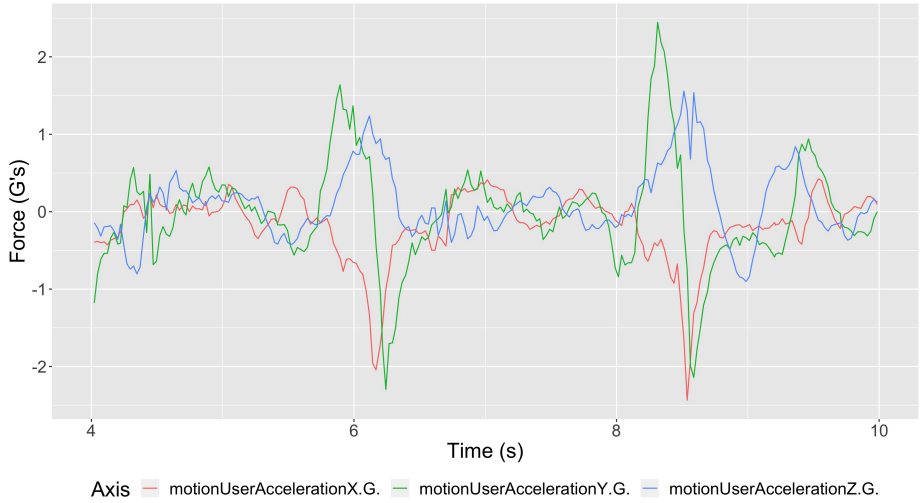



Figure 3.9: Top Spin forehand recorded at 40Hz

Having decided the frequency, a meeting was a planned with the 3 coaches to outline a plan in how data capture could be done in a efficient way without interrupting much of the athletes training. It was concluded that by using a robot capable of imitating a second player throwing balls with the same effect required for each stroke being captured would provide the available option to record stroke data.

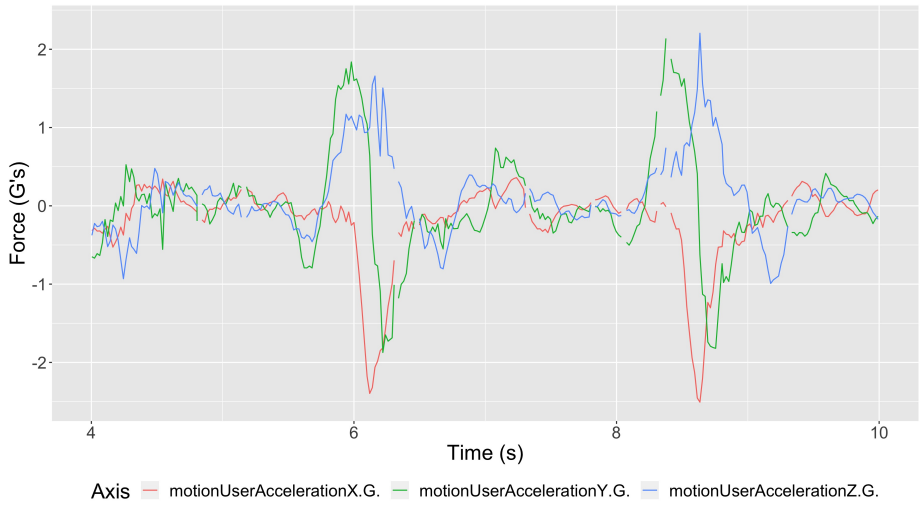



Figure 3.10: Top Spin forehand recorded at 50Hz

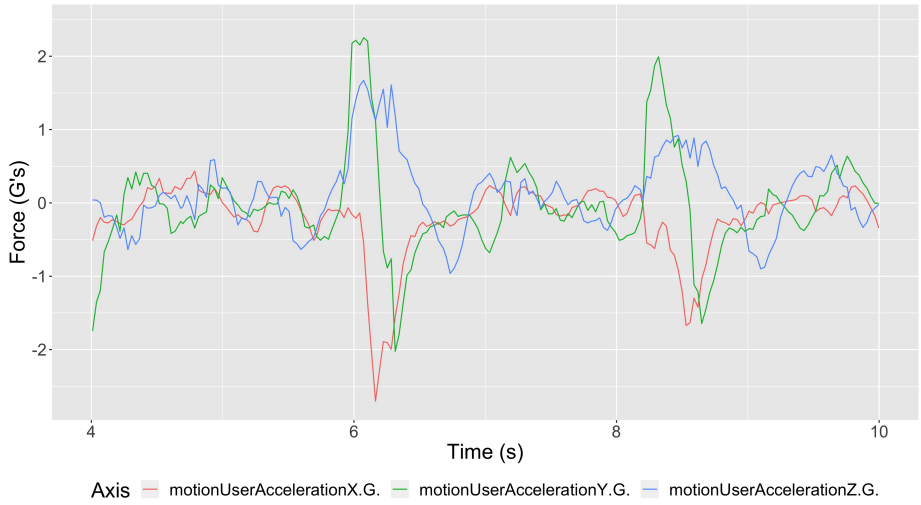



Figure 3.11: Top Spin forehand recorded at 60Hz

## 3.4 Robot methodology

Using a robot allows for continuous recording of data by the performing athlete without having to worry if the incoming ball does not contain the effect necessary for the athlete to perform the stroke being recorded. Having this restriction in consideration, a search to find the optimal robot was conducted. The robot had to meet the following characteristics:

- Capable of throwing the ball in a manner in where, when an athlete is asked to do a stroke, the thrown ball comes with a motion conducive to the player in hitting the ball with the asked stroke
- Large capacity of balls
- Variable rotation, angle, ball speed and number of balls shot per minute.
- Able to be mounted onto the official tennis table.

- Reasonable priced

First research on the market showed that most robots found on the marked consisted in 3 big difference which were classified as type 1, 2 and 3

Each type and its characteristics can be seen on table 3.1.

| Type | Strokes | Capacity (n of balls) | Rotation | Ball Catcher | Approximate cost (Eur) |
|---|---|---|---|---|---|
| 1 | Top spin, cut and drive with variable speed | 100 | Not Available | Not included | 120 |
| 2 | All existing strokes in table tennis end | 100 | Available | Not included | 200 |
| 3 | All existing strokes in table tennis | 300 | Available | Included | 500 |

Table 3.1: Types of robots and their characteristics

Type 1 robot, while being the cheaper option, did not feet the criteria defined as rotation was not available. Type 2 and 3 robots meet the required criteria but type 3 contained a higher cost. As the cost of the acquired robot also needed to be in consideration as it was being com-participated by the student doing the master thesis working company, it was concluded that the type 2 robot would be enough to meet our criteria and in case of modifications to the list of strokes, the robot would still fit for the purpose of this thesis.

The acquired robot was from MaquiGra and the model: HP-07. The robot can be seen of Fig 3.12. It can serve 40 to 70 balls per minute with a velocity of 4m/s up to 40 m/s.

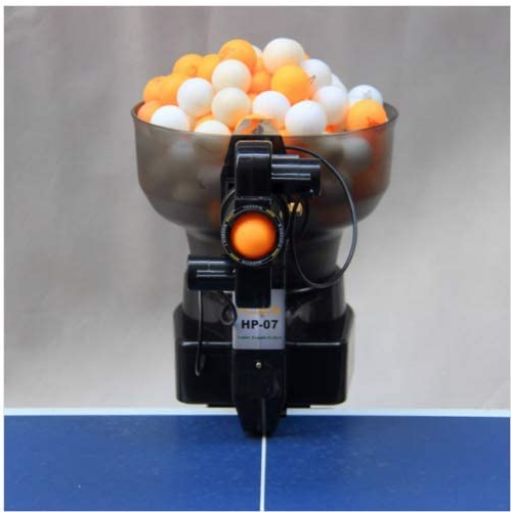



Figure 3.12: HP-07 Robot

This robot contains a ball throwing system consisting on two spinning wheels, one on top and another on the bottom, that can be controlled by the first 2 buttons included in the interface provided by the robot. The first button controls the speed of top ball and the second controls the bottom one. 7 speed levels are possible for each. Together with the

rotation tower, they create the desire effects on the ball corresponding as for example a top spin or a block. On the other 2 buttons, one controls the angle of the throw tower and the last controls the time interval between balls. The interface provided by the robot is presented on Fig 3.13

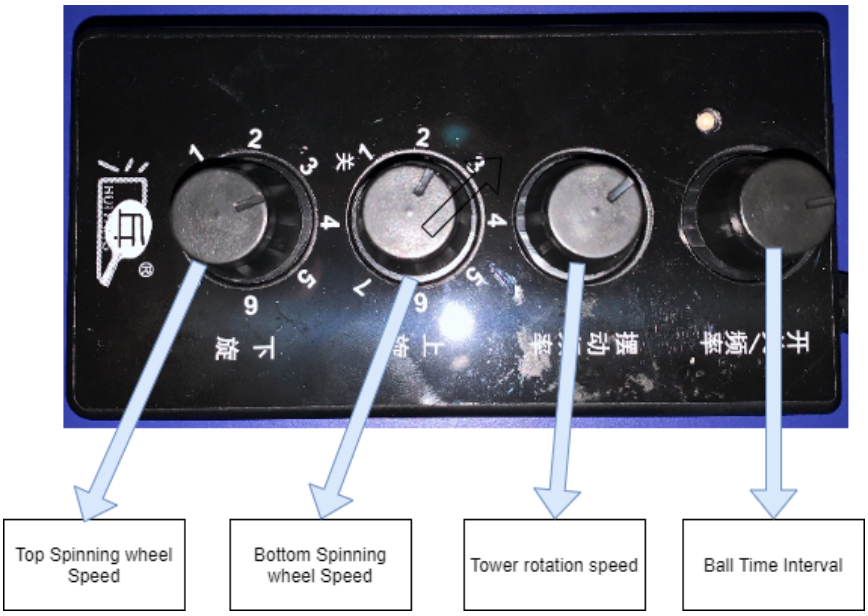



Figure 3.13: Interface provide by the robot

The last button has no levels associated, which implied a research method to find at what intervals the balls could be thrown. Since its a button, a full 360 angle paper can be cut in the center and applied onto the button. The first prototype can be seen on Fig 3.14

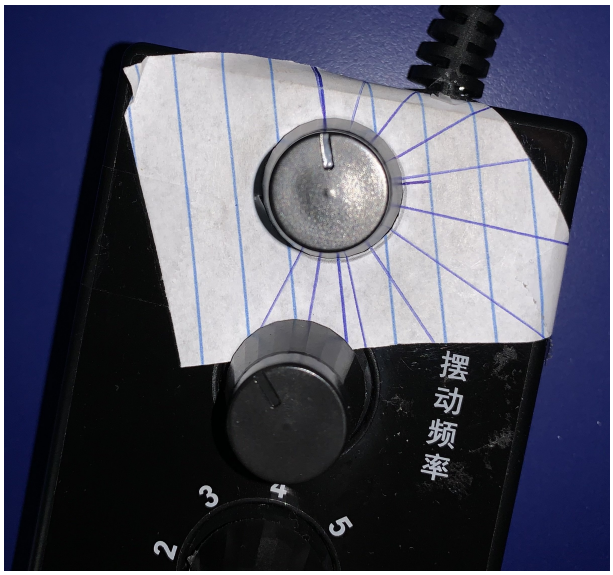



Figure 3.14: First prototype for speed button

The research found that the button did not have a full 360 free rotation. In order to throw a ball, the button must be first rotated $40^{o}$ in order to activate and pass the initial fake adjustment. At maximum rotation the button stops at $270^{a}$. It was concluded that only $230^{o}$ of range was available. This area was then divided by 11 parts meaning that by every $20.9^{o}$ one level was created. Then at each level, a video recording was made of one ball being thrown followed by a second ball, using an iPhone XR recording at 60 fps. Then by using Adobe After Effects, the first frame of the first and second ball were found and subtracted. The remaining frames were then multiplied by 0.016 (1/60 the time each frame is shown) and the time between each ball was found. The time and number of balls per minute for each level can be noted on table 3.2

| Button levels | Balls per minute | Time passed between frames (s) |
|---|---|---|
| 1 | 26.55 | 2.26 |
| 2 | 26.91 | 2.23 |
| 3 | 34.48 | 1.74 |
| 4 | 43.17 | 1.39 |
| 5 | 51.72 | 1.16 |
| 6 | 51.72 | 1.16 |
| 7 | 57.69 | 1.04 |
| 8 | 64.52 | 0.93 |
| 9 | 72.29 | 0.83 |
| 10 | 72.29 | 0.83 |
| 11 | 72.29 | 0.83 |

Table 3.2: Interval time between balls for each level

The final levels of the robot contain the same interval between balls, meaning the last $62.7^{o}$ have no meaning-full impact.

Following the list of most important strokes in table tennis together with the time available and free will of athletes to participate in this dataset, it was decided together with the coaches that the following strokes would be captured. Top Spin forehand, top spin backhand, block, flip forehand, flip backhand. The strokes considered here, are the 3 most voted on the questionnaire together with their variants with the exception of service due to the amount of ways it is possible to execute the stroke, being each service with its own characteristics.

For an athlete to do a specific stroke, the ball should come with the correct effect, making the recording stroke the perfect answer to such effect, reproducing a training or game session. In order for that to happen, parameters of the robot must be first setup such that ball can be thrown for that specific stroke. Together with the coach responsible for Vila Nova de Gaia and 4 professional athletes, multiple tests were conducted to find the optimal robot's parameters for each stroke. The following table 3.3, represents the optimal parameters for each stroke that would be required to an athlete to represent.

| Strokes | Button 1 (position) | Button 2 (Position | Direction (Reverse in case of left handed) | Rotation angle ($^o$) |
|---|---|---|---|---|
| Top Spin Forehand | 4 | 3 | Towards right side of player | 0 |
| Top Spin Backhand | 4 | 3 | Towards left side of player | 0 |
| Block | 6 | 4 | Towards left side of the player | 60 |
| Flip Forehand | 2 | 3 | Right side of player, furthest part of player half camp available | 30 |
| Flip Backhand | 2 | 3 | Left side of player, furthest part of player half camp available | 30 |

Table 3.3: Robot defined parameters for each stroke

## 3.5 Considerations

Two main datasets with different characteristics were captured with respective names D1-fast and D2-slow. D1-fast was record by having balls being shot at an interval of level 6 (1.16 seconds each time) with the interest of this dataset representing the intensity of strokes during a real live game. D2-slow was record by having balls being shot at an interval of level 3 (1.74 seconds each time) with the objective of the rest phase being visible with clearance when the player is waiting for the next ball. For each stroke in both datasets, the robot parameters were set up and the robot filled with around 100 for D1-fast and 50 D2-slow. D2-slow had a lower number of balls on the robot due to time it would take to record a single stroke session per athlete. Athletes were then asked to put the apple watch on the hand which holds the racket, and were asked to hit the ball with the corresponding stroke. Data recording started when the played pressed the record button on the apple watch followed by a human turning on the robot. Recording of data was ended by the players when the robot stopped throwing balls by pressing stop on the watch. After each stroke the recorded athlete returned to normal training and a new athlete started a new record session. This process was important because it didn't require to set up the robot each time a new player comes and no player fatigue was created. Each stroke recording session produced a csv file with the selected sensor data. A graphical representation of partial recording session can be seen on Fig 3.15.

Each peak represents a stroke. Automatic labeling was not supported by SensorLog. Manually labeling was made by using a book where name of the file - corresponding stroke - name of the player would be inserted.

Finally, after all important strokes were recorded, it was then asked for players to try to represent the hand motion corresponding to a player resting, such as when waiting for a ball or picking a new ball with sessions of around 1 minute being captured. On

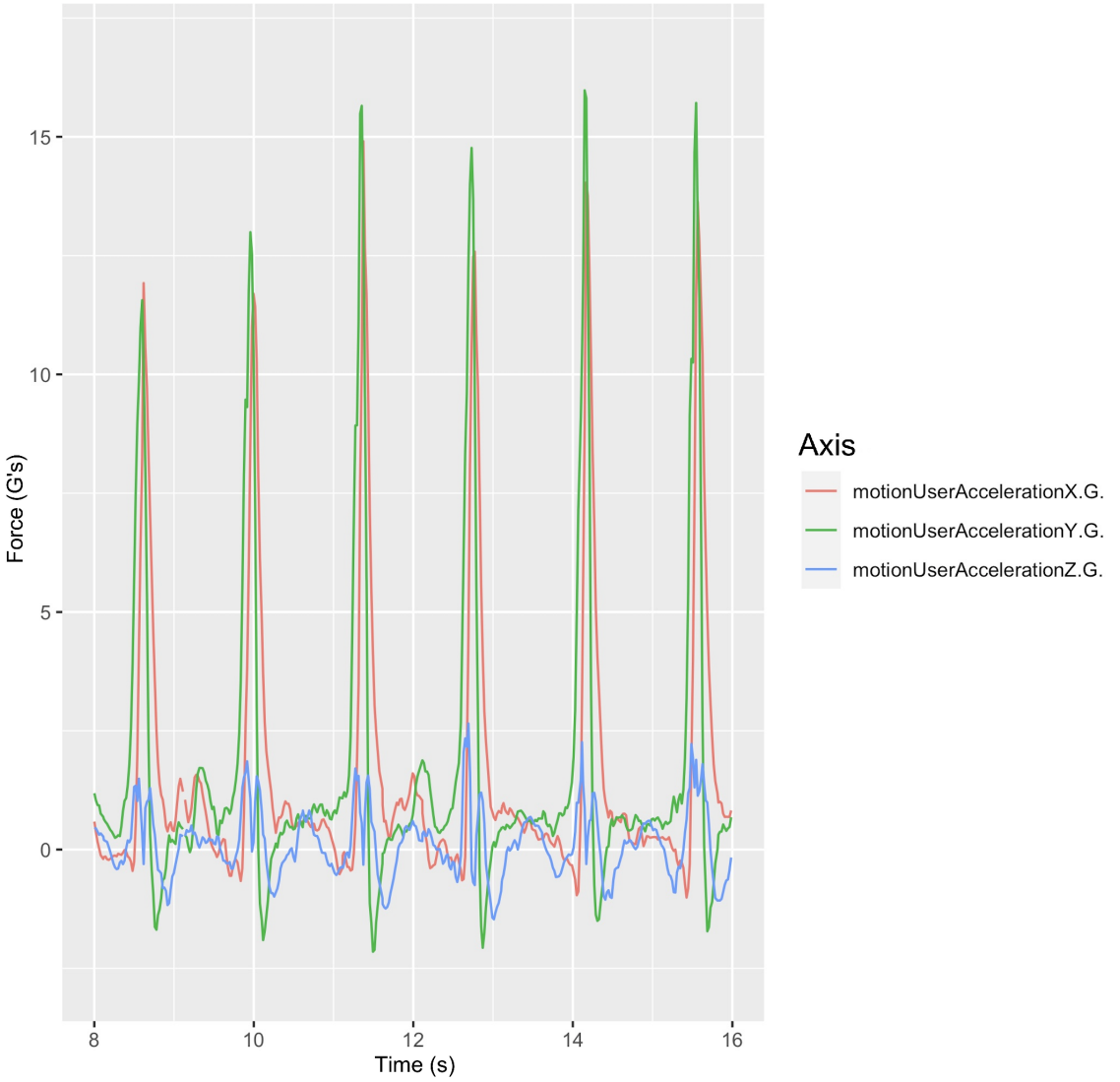



Figure 3.15: Forehand Top Spin graph in an interval of 8 seconds

both datasets and rest motion capture. 15 different athletes participated in the study, 14 of them were athletes playing at the national championships. 3 athletes were left-handed. All participants were male and the age intervals were from 14 to 55. Most athletes recorded the full list of strokes, but there were exceptions due to some athletes not showing on training days to continue recording the missing strokes and also due to injuries while training. For every athlete that participated in this study, at least one stroke was recorded. A total number of 116 samples were recorded, each corresponding with an athlete doing the respective stroke. D1-Fast contains 55 samples, while D2-slow contains 46 samples. D2-slow containing a lower number of files comes from the training center having to close due to new covid regulations. An equal or almost similar number of samples was the objective, but due to restrictions, and the amount of time left to deliver this thesis, the continuation of the work was processed with the datasets created. 15 samples each containing rest motion by the player were also acquired. These samples were then added

to each dataset for detection when a player is at rest as explained in the next chapter. The number of files per stroke can be seen on table 3.4 and 3.5.

Dataset 1

| Strokes Name | Number of files |
|---|---|
| Top Spin Forehand | 14 |
| Top Spin Backhand | 10 |
| Block | 10 |
| Flip Forehand | 11 |
| Flip Backhand | 10 |
| Rest | 15 |

Table 3.4: Number of files per stroke on dataset 1

Dataset 2

| Strokes Name | Number of files |
|---|---|
| Top Spin Forehand | 7 |
| Top Spin Backhand | 9 |
| Block | 11 |
| Flip Forehand | 9 |
| Flip Backhand | 10 |
| Rest | 15 |

Table 3.5: Number of files per stroke on dataset 2

### 3.5.1 Extracting files from the recording device

The process of transmitting files from the apple watch to the computer it's represented on Fig 3.16

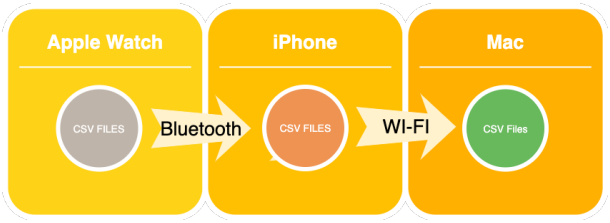



Figure 3.16: Transmission process of files from apple watch to the computer

First files names recorded on the watch were matched with the labels appointed during each recording session. After a careful confirmation of each file and its respective stroke, files were then sent to the iPhone via Bluetooth using the Sensorlog App. On iPhone, the Sensorlog app showed a list of files received by the watch. Then, after confirmation again, that all files were corrected, files were sent to a MacOS computer via Wi-Fi, followed by separation of files for the same stroke. All datasets created in this thesis are available at github repository (Nuno Ferreira, 2021), where each folder corresponds to the recorded type dataset.

Additional pre-process datasets generated from the captured datasets can also be found on the same repository.

# Chapter 4

# Table Tennis Tracker Implementation

## 4.1 Introduction

In this chapter, specification of app Table Tennis Tracker is presented.

First, the system requirements are presented, functional requirements and non-functional ones. Pre-processing of data to match CreateML Framework, CNN-LSTM and acsConvLSTM python implementations are also described. In third, the creation of DL by using CreateML is presented. Then another approach to generating the classifier model using CNN-LSTM and ConvLSTM is explained. The system architecture is then mentioned and a deep dive to the system components is represented below.

## 4.2 Requirements

### 4.2.1 Functional Requirements

**ID**: FR1.1
**Title**: Table Tennis stroke detection
**Description**: Mobile Application must be capable of labelling the correct stroke made by the athlete

**ID**: FR1.2
**Title**: Table Tennis stroke detection
**Description**: Mobile Application must be capable of providing statistics about guess strokes and acceleration.

### 4.2.2 Non-functional Requirements

**ID**: NFR1.1
**Title**: Performance
**Description**: Mobile Application must be capable of detection the stroke done with almost no delay (live).

**ID**: NFR1.2
**Title**: Reliability
**Description**: Mobile App must be capable of working offline.

**ID**: NFR1.3
**Title**: Security
**Description**: Mobile App should only allow the player to view statistics of himself and not other players.

**ID**: NFR1.4
**Title**: Security
**Description**: Mobile App should not save personal data from the athlete.

**ID**: NFR1.5
**Title**: Maintainability
**Description**: Mobile App should accept updated generated models.

**ID**: NFR1.5
**Title**: Performance
**Description**: Mobile App should use the minimal amount of processing power.

### 4.2.3 System Requirements (Software and Hardware)

**ID**: SR1.1
**Title**: Device capable of processing a DL model
**Description**: An apple watch (SE) is used to run the classification model

**ID**: SR1.2
**Title**: Device capable of communication with a wearable device
**Description**: An iPhone XR is used to receive statistics from the apple watch

**ID**: SR1.3

**Title**: Programming Language to develop and train the model
**Description**: Both CreateML(swift) and Python DL algorithms were used to train and develop the classification model

**ID**: SR1.4
**Title**: Database
**Description**: Statistical data is persisted on the apple watch and iPhone by using Core Data (Apple Api to persist data)

## 4.3 System Architecture

The development of an application capable of detecting stroke in table tennis requires multiples phases. First phase consists on processing and filtering data. Second phase is where the classification model is created based on two different approaches. One approach is using the Create ML Framework and the second using CNN-LSTM and convLSTM models to train our pre-process data and generate the classifier model. Final phase is creation of the system app that can be divided in the iOS App and the WatchOS App, and then implementation of the classification model onto the app

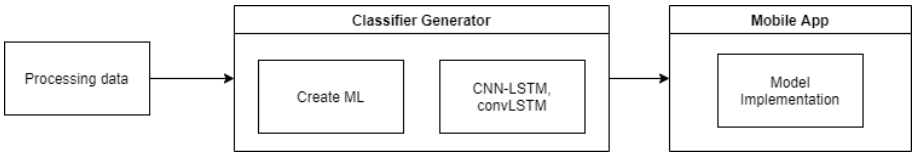



Figure 4.1: Flow Chart of detecting the stroke system

Fig 4.1 represents the common approach necessary when creating a neuronal classifier model to be implemented in an application. For the first phase, pre-processing of data was done using the R language. The R language is a free software environment for statistical computing and graphics. Its advantages focus more on an effective data handling and storage facility, a suite of operators for calculations on arrays, in particular matrices, a large, coherent, integrated collection of intermediate tools for data analysis, graphical facilities for data analysis and display either on-screen or on hard copy, and a well-developed, simple and effective programming language which includes conditionals, loops, user-defined recursive functions and input and output facilities.

In conclusion, R language was considered in this master thesis as the perfect language in our study to visualize the data captured and identify where strokes were present and in what we can consider a stroke. On the second phase, creating the classifier model is the main objective and since also one objective of this work is to use Apple's tools to produce

a app capable of detection strokes in table tennis, Apple's Create ML framework was the perfect candidate for this work making it the first approach taken to develop the solution for our objectives. Create ML is a framework capable of creating ML models by showing it representative samples. Create ML takes the complexity out of model training while producing powerful Core ML models. The Core ML models can be then used on swift applications for iOS, Mac and WatchOs. It supports image, motion, video, sound, text, and tabular data. Inside the motion category, activity classifier models can be created by meting the required criteria

- Classes must have their own folder
- Each folder must contain motion data recorded in a supported file type (csv,json, etc)

## 4.4 Datasets Pre-processing

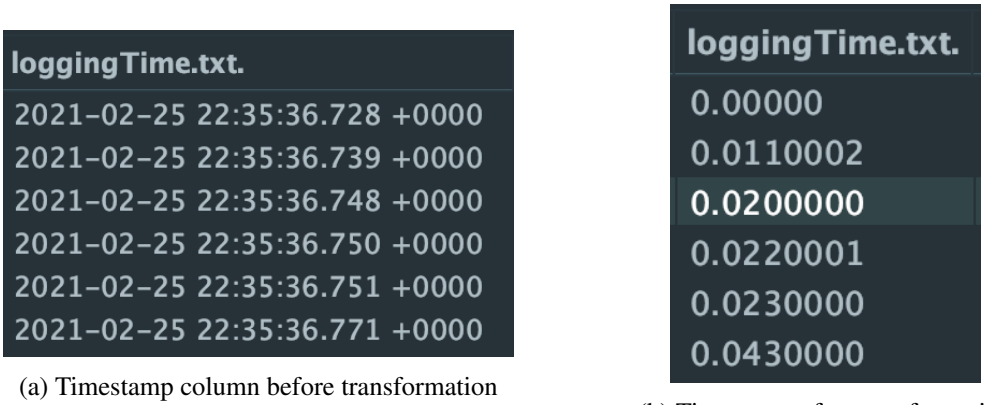
First step in pre-processing was removing all rows where column values contained null rows from the two datasets.

| | loggingTime.txt. | accelerometerTimestamp_sinceReboot.s. | accelerometerAccelerationX.G. | accelerometerAccelerationY.G. | accelerometerAccelerationZ.G. | motionTimestamp_sinceReboot.s. |
|---|---|---|---|---|---|---|
| 1 | 2021-02-25 22:35:36.728 +0000 | 165113 | 0.107697 | 0.512512 | -0.854523 | 165113 |
| 2 | 2021-02-25 22:35:36.739 +0000 | 165113 | 0.0688019 | 0.500916 | -0.860779 | 165113 |
| 3 | 2021-02-25 22:35:36.748 +0000 | 165113 | 0.0660248 | 0.497879 | -0.859894 | 165113 |
| 4 | 2021-02-25 22:35:36.750 +0000 | 165113 | 0.0673828 | 0.494690 | -0.867981 | 165113 |
| 5 | 2021-02-25 22:35:36.751 +0000 | 165113 | 0.0790100 | 0.511398 | -0.858917 | 165113 |
| 6 | 2021-02-25 22:35:36.771 +0000 | 165113 | 0.0782013 | 0.522263 | -0.851212 | 165113 |
| 7 | 2021-02-25 22:35:36.791 +0000 | 165113 | 0.0690918 | 0.517670 | -0.856094 | 165113 |
| 8 | 2021-02-25 22:35:36.811 +0000 | 165113 | 0.0689087 | 0.499802 | -0.857056 | 165113 |
| 9 | 2021-02-25 22:35:36.831 +0000 | 165113 | 0.0785217 | 0.499374 | -0.858093 | 165113 |
| 10 | 2021-02-25 22:35:36.851 +0000 | 165113 | 0.0779877 | 0.513626 | -0.859604 | 165113 |
| 11 | 2021-02-25 22:35:36.871 +0000 | 165113 | 0.0839996 | 0.515656 | -0.854111 | 165113 |
| 12 | 2021-02-25 22:35:36.891 +0000 | 165113 | 0.0822754 | 0.514801 | -0.848969 | 165113 |
| 13 | 2021-02-25 22:35:36.911 +0000 | 165113 | 0.0889282 | 0.508392 | -0.846527 | 165113 |
| 14 | 2021-02-25 22:35:36.930 +0000 | 165113 | 0.0858002 | 0.510651 | -0.845779 | 165113 |
| 15 | 2021-02-25 22:35:36.950 +0000 | 165113 | 0.0811615 | 0.513824 | -0.847168 | 165113 |
| 16 | 2021-02-25 22:35:36.971 +0000 | 165113 | 0.0901642 | 0.517517 | -0.858444 | 165113 |
| 17 | 2021-02-25 22:35:36.990 +0000 | 165113 | 0.0938263 | 0.523148 | -0.867294 | 165113 |
| 18 | 2021-02-25 22:35:37.010 +0000 | 165113 | 0.0995178 | 0.522186 | -0.872040 | 165113 |
| 19 | 2021-02-25 22:35:37.030 +0000 | 165113 | 0.114120 | 0.540222 | -0.877914 | 165113 |
| 20 | 2021-02-25 22:35:37.050 +0000 | 165113 | 0.118393 | 0.539581 | -0.884659 | 165113 |
| 21 | 2021-02-25 22:35:37.070 +0000 | 165113 | 0.110382 | 0.544312 | -0.891739 | 165113 |

Figure 4.2: CSV data represented from a top spin forehand stroke

Having removed all null values, sensor data from the altimeter was also removed as the dataset didn't record the optimal conditions for altimeter data to be reliable. Both datasets did contain a timestamp column, but its starting value was from the hour it was recorded. A function was executed on this column, converting the first value to 0 and the following values were converted to time values by calculating the time difference between each row. Fig 4.3 shows how the difference before and after applying the function.

The first and last 4 seconds of data were removed to eliminate noise created from the robot and the player, as initial delay to throw the balls and clicking on the stop button was detected. Renaming the columns to more conventional names and removing unnecessary columns was also performed. The remaining columns were: 'rotation_x', 'rotation_y',



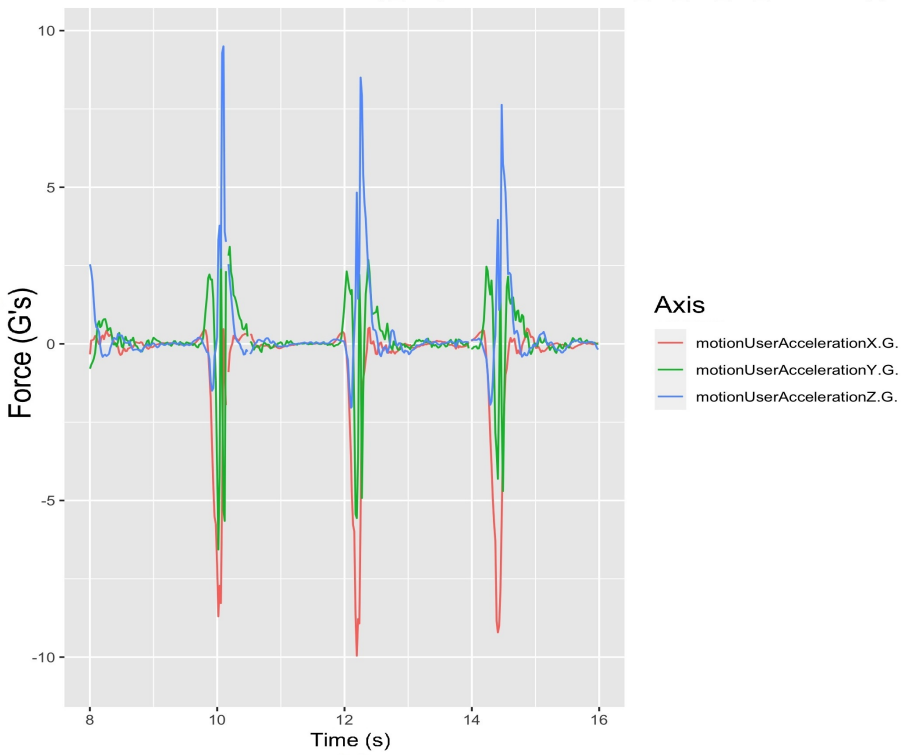

(a) Timestamp column before transformation

(b) Timestamp after transformation

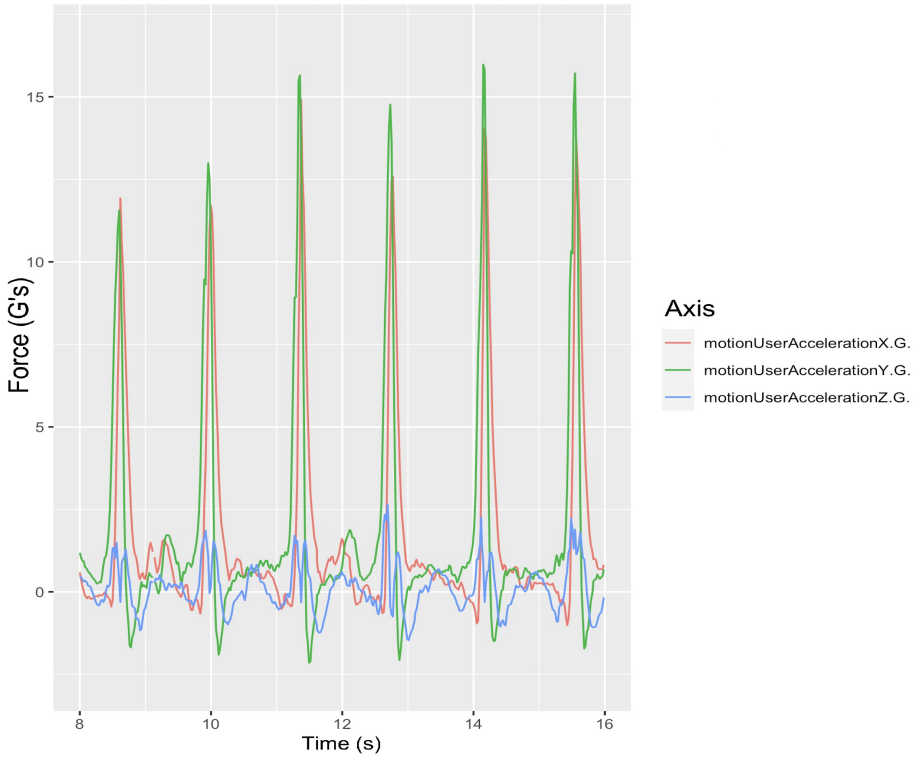
Figure 4.3: CSV Timestamp data transformation before / after transformation

'rotation_z', 'acceleration_x', 'acceleration_y', 'acceleration_z'. A third process was also created, where 3 more pre-process datasets were created from the initial two recorded.

This process consisted of first identifying when a stroke had occurred and then applying cuts consisting of data containing the beginning of the stroke, the racket hit on the ball, and end of stroke. To identify when a stroke has occurred, graphics for visualization of motion data were created for each stroke based on the accelerometer. By analyzing the graphs generated from the accelerometer data for each stroke, a pattern could be seen that for every occurring stroke, a rapid increase in user acceleration could be noted. Fig 4.4 shows a graphical representation of the accelerometer data for two different strokes.




(a) 3 Flip backhand strokes represented in a graphic

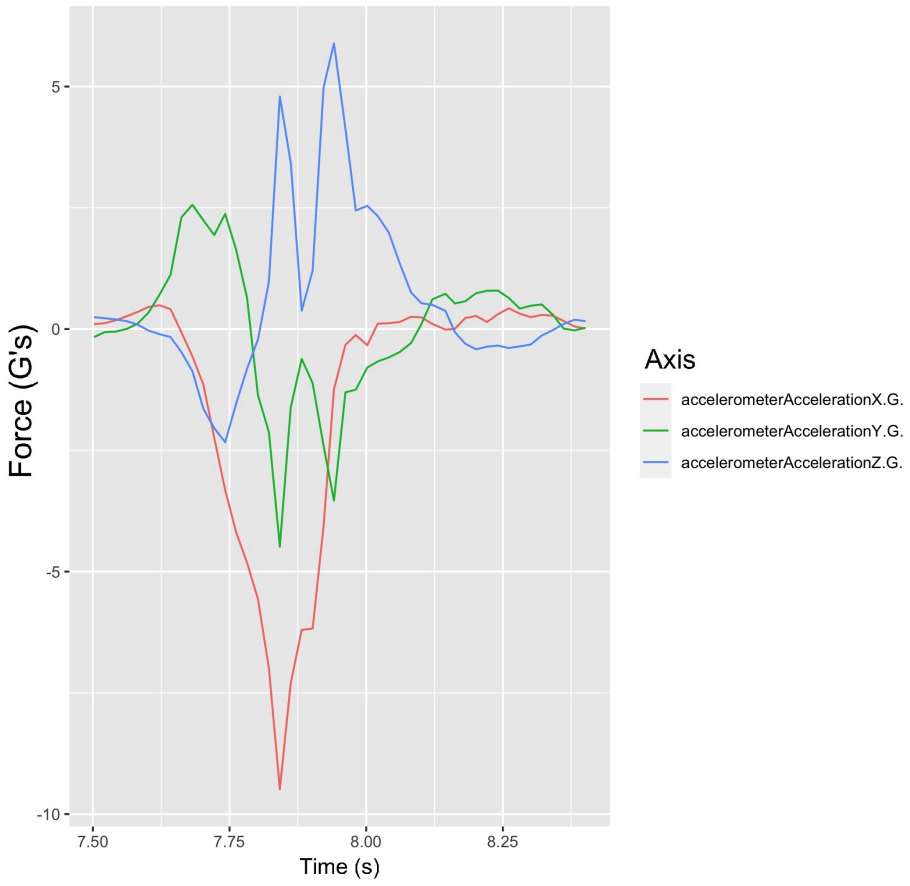



(b) 6 Top Spin forehands strokes represented in a graphic

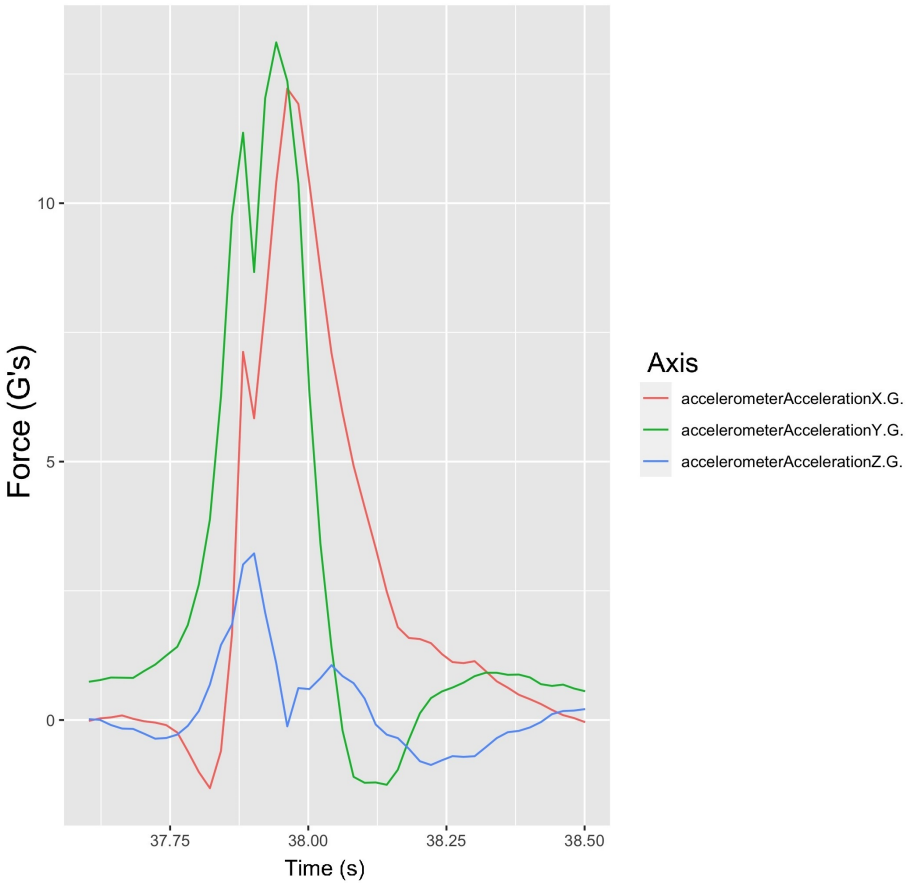
Figure 4.4: Graphical representation of motion data for two strokes

It was also noted that for different strokes, the axis on which the acceleration is greater is different from each other. With the above mentioned, a stroke detection algorithm was created. The algorithm consisted in analyzing the acceleration on each data row, and if the acceleration exceeded a defined threshold for a defined axis and value for each stroke, a cut was then applied. Thresh holds and the respective axis for each stroke were found by empiric testing until satisfactory cuts were obtained. 46 rows of sensor data were found to be capable of representing the beginning, middle, and end of the stroke.

Fig 4.5 shows a graphical represented data of the same strokes presented in 4.4 but with the cuts now applied.




(a) Cut backhand flip




(b) Cut Top Spin forehands

Figure 4.5: Graphical representation of motion data for two strokes

The optimal thresholds and each axis of the accelerometer selected for each stroke can also be seen on the table 4.1

| Strokes | Axis (Accelerometer) | Acceleration (G's) |
|---|---|---|
| Top Spin (forehand) | X | 1 |
| Top Spin(backhand) | Z | 1 |
| Block | Z | 1 |
| Flip (forehand) | X | 1 |
| Flip (blackhand) | Z | 1 |

Table 4.1: Defined preferred axis and values for each stroke detection algorithm

The datasets generated by applying this algorithm were D3-fast-cut, D4-slow-cut, D5-fast-slow-cut.

- D3-FAST-CUT contains the data from D1-fast but with the algorithm applied
- D4-SLOW-CUT contains the same data as D2-slow but again with the algorithm applied.
- D5-FAST-SLOW-CUT contains a merge of D1-fast and D2-slow and then the algorithm applied.

For the rest stroke/class, on these new pre-process datasets, the acquired data from D1-fast and D2-slow was discarded and instead, samples of this class were created on a third process of data processing by using all observations where no stroke was being detected and considering as the athlete is resting. This process of creating rest samples proved very costly regarding performance. The process would take more then one hour just when scrolling the sensor data of one type of stroke. To improve performance, the creation of the rest samples was limited by 200 for each stroke, generating 1000 samples in the end.

Having multiple processed datasets of the same data allows the possibility to work in different ways that can provide answers to how data should be treated in future work related to table-tennis. Also, a higher number of classifier models can be generated, providing a better foundation to this dissertation thesis.

For each dataset, each stroke class was then divided in a ratio o 80% / 20% in train and test folders.

## 4.5 Create ML Classifier

Create ML is a framework provided by Apple capable of creating ML models for use in your app by offering a simple UI when creating any kind of model. It contains a set of

machine learning products using Swift and other tools Programming language and macOS playground to create and train custom machine learning Model on Mac computer (Apple, 2021). The suite includes many features, including Image Classification to train a machine learning model to classify images, Natural Language Processing to classify natural language text, Tabular Data to train models that deal with labeling information or to estimate new amounts, motion activity tabular data to classify activities and motion video data to create an Action Classifier Model. It's only available on Apple Environment computers, works offline and doesn't require any fees or servers to train or use the model. By using Create ML framework, the user can abstract it self from knowing any ML algorithms or having to create features. This can be good for beginners starting to work with machine learning where a few clicks can create a ready-to-implement ML model. But it can also prove to be bad when results are lower than expected. To use the datasets acquired on the Create Ml framework, data processing and filtering had to be done first. Each row of one csv file recorded it's a frame of data based on the frequency the dataset was recorded containing data from 3 sensors. Fig 4.2 shows a class represented data.

For an activity classifier Create ML provides an interface where training, validation and test data can be set by dragging a folder. Validation data can also be split automatically from training data in case no folder is given to the framework. Fig 4.6 illustrates Create ML UI.

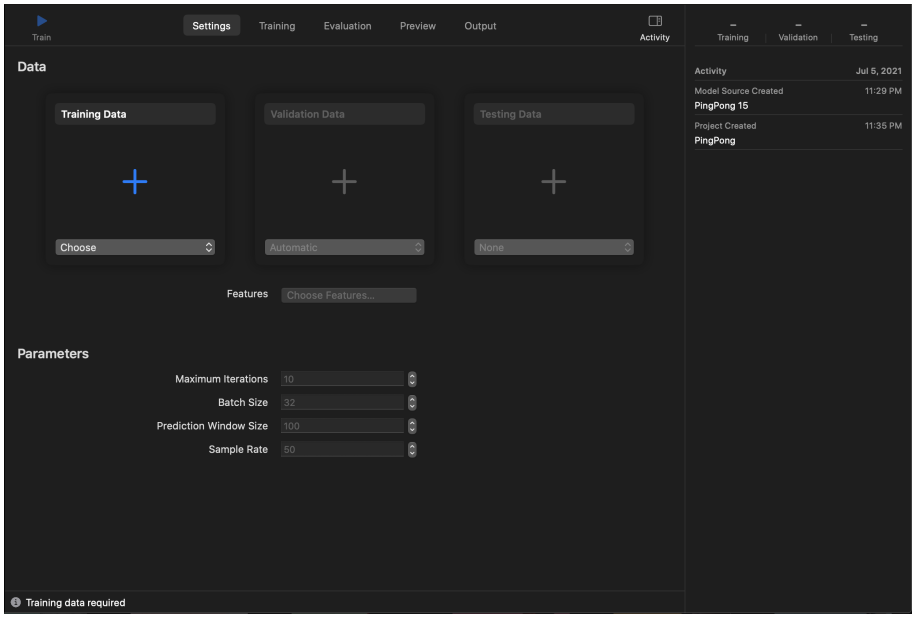



Figure 4.6: Create ML UI when creating an activity classifier

After dragging each train and test folder to their respective place and selecting validating data to be split from training data, the features option is then available to be selected. The features available to select and train the model correspond to the columns that were selected during data treatment. In this work 6 features are available. Each one corresponds

to a different axis on the accelerometer and gyroscope.

CreateML offers 4 configurable parameters, maximum iterations, batch size, prediction windows size, sample rate.

Sample Rate refers to the number of observations recorded per second by the on-device hardware. This number is mentioned in chapter 3 its value its 50Hz.

Prediction window size means how many observations of sensor data should be considered for stroke detection. This value is different depending on which dataset is being considered. For dataset D1-fast, it was mentioned in chapter 3 that thrown balls by the robot had an interval of 1.166s, meaning that for a equal amount of time, a stroke can be expected. Because the data was record at 50Hz, the predicting windows size can be calculated based on the interval of thrown balls and its frequency, which for D1-Fast, equals to a windows prediction size of 58 (1.16 / 50), for D2-slow, the value is 87 and for the datasets 3,4,5 a value of 46 was found to the optimal parameter.

For batch size and number of iterations empiric tests were done and it was concluded that 30 iterations is more then enough to train the data and for the batch size, 100 and 60 D1-fast and D2-slow proved to show better results.

Finally trained was started in Create ML framework.

The algorithms used by Create ML to train and generate an activity classifier model are not disclosed by Apple, meaning that the user has no control of the training process and can't optimize for further upgrades. Yet is possible to discover the algorithms, activation and loss functions by using an Application called Neutron. Netron is a viewer for neural network, deep learning and machine learning models that analyzes the generated output model of various ML frameworks such TensorFlow Lite, Keras, Create ML, etc...

Fig 4.7 displays the pipeline that Create ML uses to generate the classifier model. It possible to conclude that Create ML uses a convolution layer as the first layer with the activation function "ReLU", followed by a uniDirectionalLSTM, a innerProducted, a batch normalization with 128 channels proceded by a "ReLU" activation function, then another innerProduct and finally softmax activation function.

To evaluate the generated classifier models, precision, recall and the F1 score were metrics were used.

Both 5 datasets were used to generate a classifier model.

## 4.6 CNN-LSTM and ConvLSTM Classifier Models

The second approach in this work involved using CNN-LSTM and ConvLSTM models. In chapter 2, previous works demonstrated good performance and accuracy on generating good classifier models in and out of Table Tennis. Having a second approach to a problem can also help identify the limitations of each path taken. This approach consisted on creating two classifier model by using two different algorithms, CNN-LSTM and ConvLSTM.

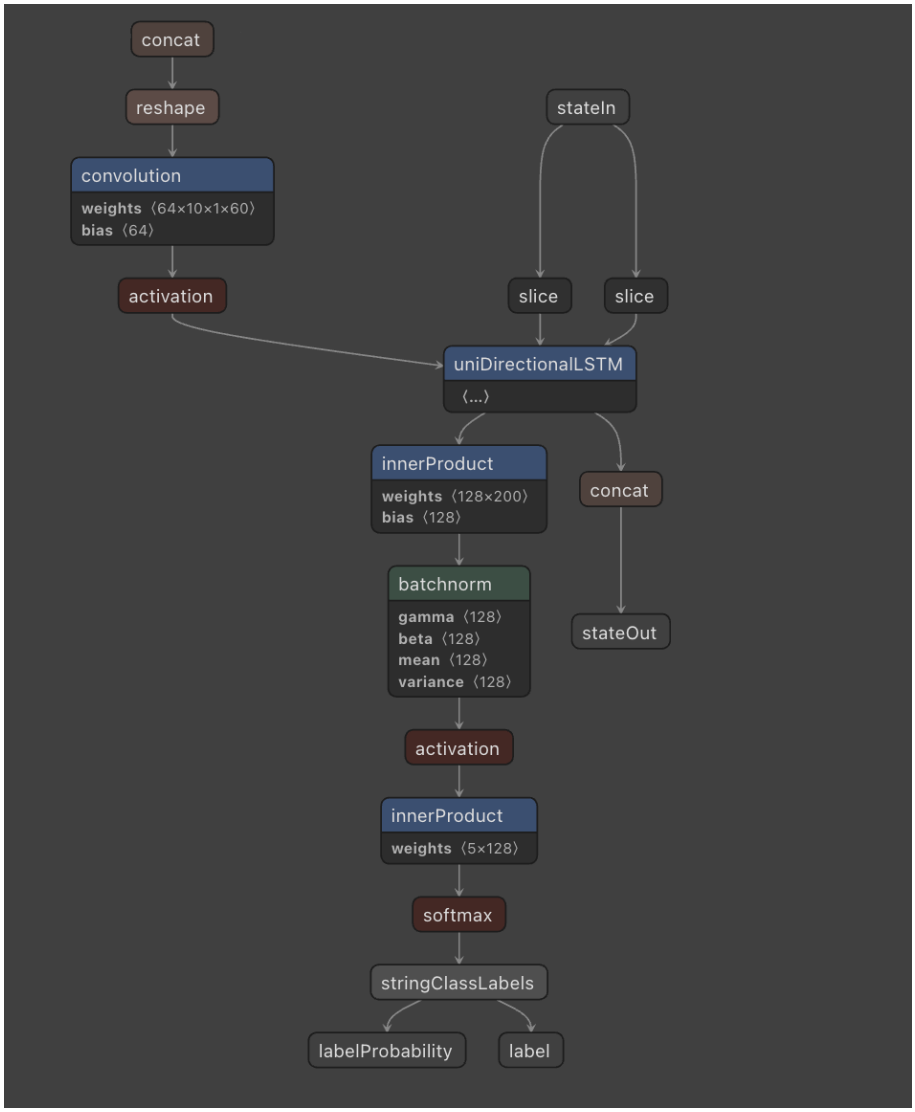



Figure 4.7: Layers used by Create ML to generate an activity classifier using the Netron visualizer

CNN-LSTM are the most powerful and well known subset type of artificial neural network designed to recognize patterns in sequences of data, such as numerical times series data emerging from sensors. CNNs are proven to reduce frequency variations and can extract the features between several variables. On the other hand, LSTMs are capable of modeling temporal information of irregular trends in time series components. What differentiates CNN and LSTM from other neural networks is that they take time and sequence into account. They also have a temporal dimension (Xia et al.). The research done by (Sainath et al., 2015) provided an example of what a CNN-LSTM unified architecture was possible, and the authors demonstrated that such architecture provided a 5 to 7% increase in words error rate. Creating of a classifier model based on CNN-LSTM architecture for activity recognition was also performed in (Xia et al.) and applied in multiple datasets, all with good results. Most examples of application of these two models,

were found using Python programming language together with Tensorflow. Tensorflow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.

ConvLSTM is further extension of the CNN-LSTM. The base of this algorithm extension is to perform the convolutions of the CNN (how the CNN reads the input sequence data) as part of the LSTM (Shi et al.). Unlike LSTM, which directly reads data to calculate internal states and state transitions, and interprets a CNN model output, ConvLSTM directly uses convolution as part of the read input to the LSTM unit itself. The ConvLSTM determines the future state of a certain cell in the grid by the inputs and past states of its local neighbors. By stacking multiple ConvLSTM layers and forming a coding-prediction structure, we can not only build network models for problems, but also build network models for more general time-space sequence prediction problems which suits our case of table tennis and because the network has multiple stacked ConvLSTM layers, it has strong representation capabilities, which makes it suitable for prediction in complex dynamic systems.

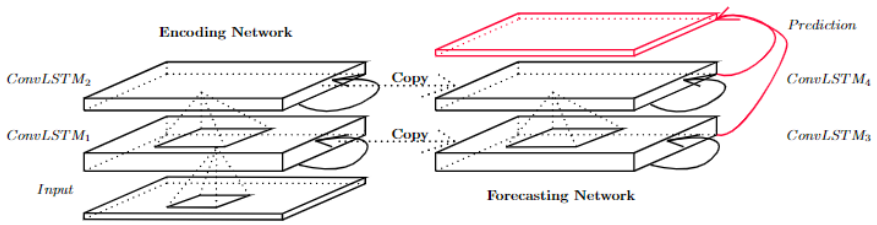



Figure 4.8: Example of Encoding-forecasting ConvLSTM. Fig taken from (Shi et al., 2015)

Only datasets D3-fast-cut, D4-slow-cut and D5-fast-slow-cut were used on both models due to having a higher number of samples, mostly noise removed, and also time constraints.

### 4.6.1 Creating CNN-LSTM model

Implementing an LSTM model using the Keras deep learning library, requires the model to have a three-dimensional input with [samples, time steps, features]. Being our samples the number of strokes detected for each class (csv files created), our time steps being the value 46, which is the number of observations needed to have the beginning, middle and end of the stroke and features the number of axis per sensor.

First, the creation of subsequences of the main sequence as blocks had to be prepared to extract the features from each block, allowing the LSTM to interpret the features extracted from each block. For that a CNN Model can be defined, in which the model is

prepared to read the sequences of 46 time steps and the 6 features or multiple sequences of the time steps.

In our dataset, the time steps in each window were split into 1 subsequence (the full sequence) of 46 time steps and 2 subsequences of 23 time steps but the performing tests revealed that no significant difference was found by using 1 subsequence or 2.

With these tests in mind, the approach taken to implement this model was to use the full 46 time steps into 1 sequence for the CNN model to process.

This implementation required further preprocessing of the data. For the D3-fast-cut, D4-slow-cut, and D5-fast-slow-cut datasets, the files associated with a particular stroke contained 46 rows of data and 6 columns representing each sensor axis. The data would be transposed to produce 6 files, each representing a sensor axis, with one row corresponding to a detected stroke on the same axis. Each row would contain 47 columns, which means that for the first 46 columns, each column represents a value of the respective axis of the performed stroke and the last column contains an ID corresponding to the identified stroke. The ID identification is shown in table 4.2 and table 4.3 shows an example of how the data was represented for input to the CNN-LSTM model. The implementation of the models CNN-LSTM and convLSTM on tensorflow used in this work is public and can also be found on github: `https://github.com/MikaPower/ai_ping_pong/tree/master/`

| Label | Id |
|---|---|
| Top Spin (forehand) | 0 |
| Top Spin (backhand) | 1 |
| Rest | 2 |
| Flip (backhand) | 3 |
| Flip (forehand) | 4 |
| Flip (backhand) | 5 |
| Rest | 6 |

Table 4.2: Corresponding stroke to its label

| V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 |
|---|---|---|---|---|---|---|---|
| 0.5303171 | 0.7587863 | 1.0144976 | 0.7206307 | 0.3129258 | 0.4986531 | 0.7612206 | 1.3007699 |

Table 4.3: Initials columns of the first row of the axis x of the accelerometer

Creation of the classifier started by adding a sequential model followed by applying a Time Distribute layer allowing the model to read in 1, 2 or multiple subsequences of the window provided. Features were then flattened and provided to the LSTM model to read and extract its own features before a final mapping to the corresponding activity was performed. The remaining layers added to the model are two consecutive CNN layers followed by dropout of 50% and a max pooling layer. This layers are the basic structures of a CNN-LSTM model.

For the loss function "Categorical Cross Entropy" was used as the problem it's a multi-class classification. As for the optimizer, Adam optimized was the chosen one, as good results were shown in previous studied works. Fig 4.9 exemplifies the implemented training model.

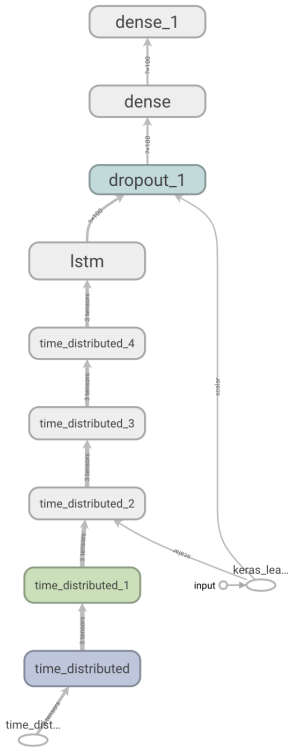



Figure 4.9: Architecture of CNN-LSTM model deployed to generate a new classifier model

### 4.6.2 Creating ConvLSTM classifier

For the ConvLSTM, the ConvLSTM2D category in the Keras library supports the ConvLSTM model for 2D information. It is frequently used to classify 1D variables containing statistics. This category input is based on [samples, time% sequences, rows, cols, channels]. By using the same approach taken when creating the CNN-LSTM classifier model, it was considered for the samples to be the value of the total rows available of sensor data, for the time sequences, the value defiend was 1 as only on sequence of the windows, was defined ( 2 was also available), row number value was 1 we are working with 1D array of data, and the number of columns represents the number of time steps in the sequence meaning a value of 46. When creating a ConvLSTM model, the CNN and LSTM parts of the model must be defined in separate. For that a 2d kernel of 1 (row) x 3 (time steps of the sequence), a 64 value for the filters, and the activation function "ReLu" were also defined at this layer, then a dropout of 50% followed by a flattening of the output must

be processed before adding the final 2 dense layers with activation functions "ReLu" and "softmax" respectively. For the model compilation, the loss function was "Categorical Cross Entropy" and for the optimizer Adam, the same was the CNN-LSTM model. Fig 4.10 presents the convLSTM composition.

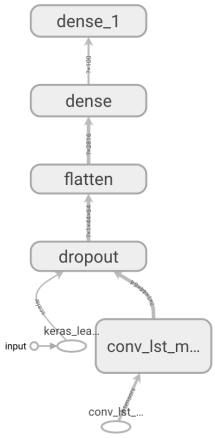



Figure 4.10: Architecture of ConvLSTM model deployed to generate a new classifier model

Model performance was measured by evaluation of the model 10 times and by calculating the mean and standard deviation of the performance. The reason for this is that the neural network outcome is always different, which means that different specific model will be produced when the same model configuration is trained on the same data. To visualize model statistics such as loss function and accuracy an add-on called TensorBoard for tensorflow was used. TensorBoard is a suite of web applications for inspecting and understanding TensorFlow runs and graphs. More details about the implementation of CNN-LSTM and ConvLSTM can be seen on Brownlee.

### 4.6.3 Metrics and Performance

Model training was performed on a MacBook Pro (2019) using the graphics card (AMD Radeon Pro 560X). While current Apple watches do not support GPU computations to run ML models, only GPU training is performed and in the application CPU is used to run the classification model.

## 4.7 Creating Core ML Model

Classifier models created with TensorFlow are not compatible for use with watchOS, macOS, and iOS devices, as the expected type of classifier model for these operating systems is a Core ML model.

However, Apple has worked with Tensorflow to create a Python package that can be used to convert the Tensorflow model to a Core ML model. To implement this package, the following command was run in our virtual environment: pip install coremltools. When installing coremltools, a warning was displayed that the latest version of coremltools 4.1 only guarantees conversion of Tensorflow models to Core ML models if the TensorFlow version is lower than 2.3.1. The initial version of Tensorflow when testing began required a downgrade as the version used was 2.5.0. A downgrade was performed and no issues were found. The results and model chosen for implementation in an iOS app were obtained after the downgrade. When converting the Tensorflow model to a Core ML model, the process consists of 3 steps. First, the model to be converted should be saved. Second, the saved model should be passed to the coremltools method called "convert" where the model passes through the Unified Conversion API. Calling this method returned a Core ML model. The last remaining step was to save the current Core ML model. By calling save again with the given filename pretended, a Core ML model was now available to implement on any watchOS, iOS, and MacOS with capable AI processors.

Fig 4.11 shows the implementation of the process explained above.

```
def save_model(model):
    model.save('tf_keras_model_lstm_cnn')
    class_labels = ['top_spin_direita','top_spin_esquerda','bloco','flip_direita','flip_esquerda','rest']
    classifier_config = ct.ClassifierConfig(class_labels)

    mlmodel = ct.convert('tf_keras_model_lstm_cnn',classifier_config=classifier_config)

    # set general model metadata
    mlmodel.author = 'Nuno Ferreira'
    #mlmodel.license = 'BSD'
    mlmodel.short_description = 'Predicts the movement of a player during a table tenis game'
    mlmodel.save("pingPong.mlmodel",)
```

Figure 4.11: Converting Tensorflow model to CoreMl model function

## 4.8 General View of the app's System

Having the classifier model in the correct format was a requirement for development of the application in WatchOS and iOS. Figure 4.12 shows the environment on which the model was to be based and implemented.

Development of the app was divided in two components, one run by the iPhone where statistics about the training or game session provided by the apple watch will be displayed and another run by the Apple watch where gyroscope and accelerometer data will be feed to the classifier model to predict the stroke.

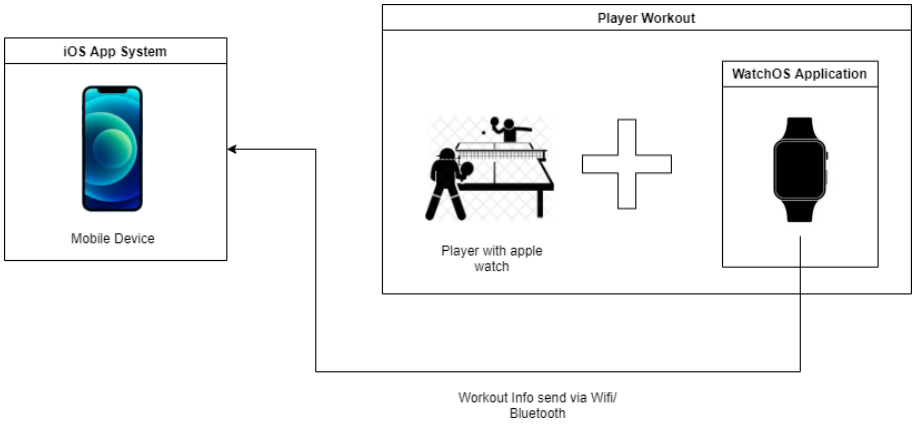



Figure 4.12: Working environment of the application

### 4.8.1 WatchOs App Architecture and Development

The architecture of the WatchOs application is based on the *Model-View-ViewModel* architecture where each view doesn't necessary rely on any external state. As seen in the Fig 4.13 representing the application flux diagram, the application starts by acquiring the accelerometer and gyroscope data from the embedded sensors. This data can be acquired by using the Core Motion framework, in which raw and device motion data are provided. As the data used to train the classifier model also came from device motion data, where

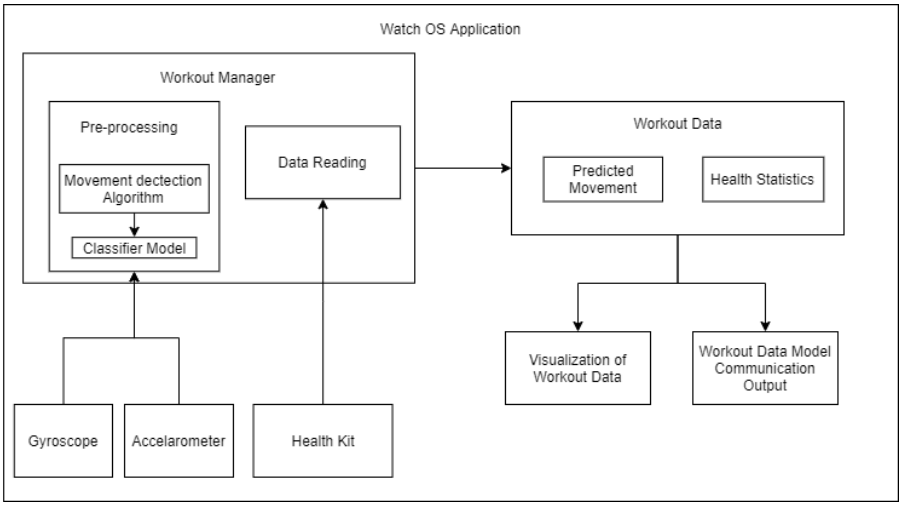



Figure 4.13: WatchOs Application Flux Diagram

data is adjusted for gravity and other forms of bias, this was the type of data acquired. Also, by using the Health Kit Framework, heart rate and steps were captured and sent to the Workout Manager. The Workout Manager is responsible for pre-processing the data received by the gyroscope and accelerometer and applying a stroke detection algorithm and storing health data obtained from the health kit. The stroke detection algorithm analyses the data obtained and in case of a positive outcome, passes the data to the classifier

model to predict the corresponding stroke. The classifier provides a dictionary with each detectable stroke and the corresponding probability. The stroke prediction was only considered as the correct one if probability would be equal or higher then 80%, otherwise it was considered as the player resting. Having the predicted stroke, a workout Data model is generated, containing the predicted stroke as well the health data. The data from this model is then displayed to the athlete Apple Watch UI, and also sent to app component on the iOS system.

### 4.8.2 Stroke Detection Algorithm

The stroke detection algorithm was created to help feed the classifier model with a full data stroke, corresponding to the beginning, half and ending of the stroke. This prevents the problem represented on Fig 4.14 where the starting point could be any point of the graph, meaning, the data that would be feed to the classifier model would represent strokes such as those that were already at the end, strokes that were at half-away, etc.

Implementation of the algorithm was based in same way as when detecting complete strokes for dataset creation in chapter 2. Its based on peak acceleration on any axis of the accelerometer. While the complete stroke detection algorithm mentioned in chapter 2 contained an activation value for each stroke in relation with the most high value changing axis, mapped based on graphical analysis of strokes performed by the athletes, instead a single value for all the axis of accelerometer was defined. This value was defined as the same minimum-mapped value used in chapter 2for complete stroke detection, 1 G's (g-force). The stroke detection algorithm consists of a windows slide mechanism where it is first stored the 10 initial samples of data. After the tenth, the algorithm would be searching for a peak value meaning a stroke may have occurred. When the algorithm activates, the next 36 frames of data are stored (46 frames represents a full stroke), and then the full 46 stroke samples are transmitted to the classifier model. Fig 4.15 represents an example of the algorithm applied to an example samples set, containing the current sliding windows and at what sample, the peak acceleration algorithm is activated.

### 4.8.3 WatchOS development

When developing an app for WatchOS, two types of interfaces for development can be chosen. One is based on Storyboards where its possible to spread out different screens and show how the scenes will connect. Storyboards give you a calculated perspective on the application, and how the scenes fit together and how they connect. StoryBoards were the only option available till 2019 where SwiftUI was introduced. SwiftUI is the second code interface available in xcode. It provides views, controls, and layout structures for declaring the app's user interface by using components as the base. Components can

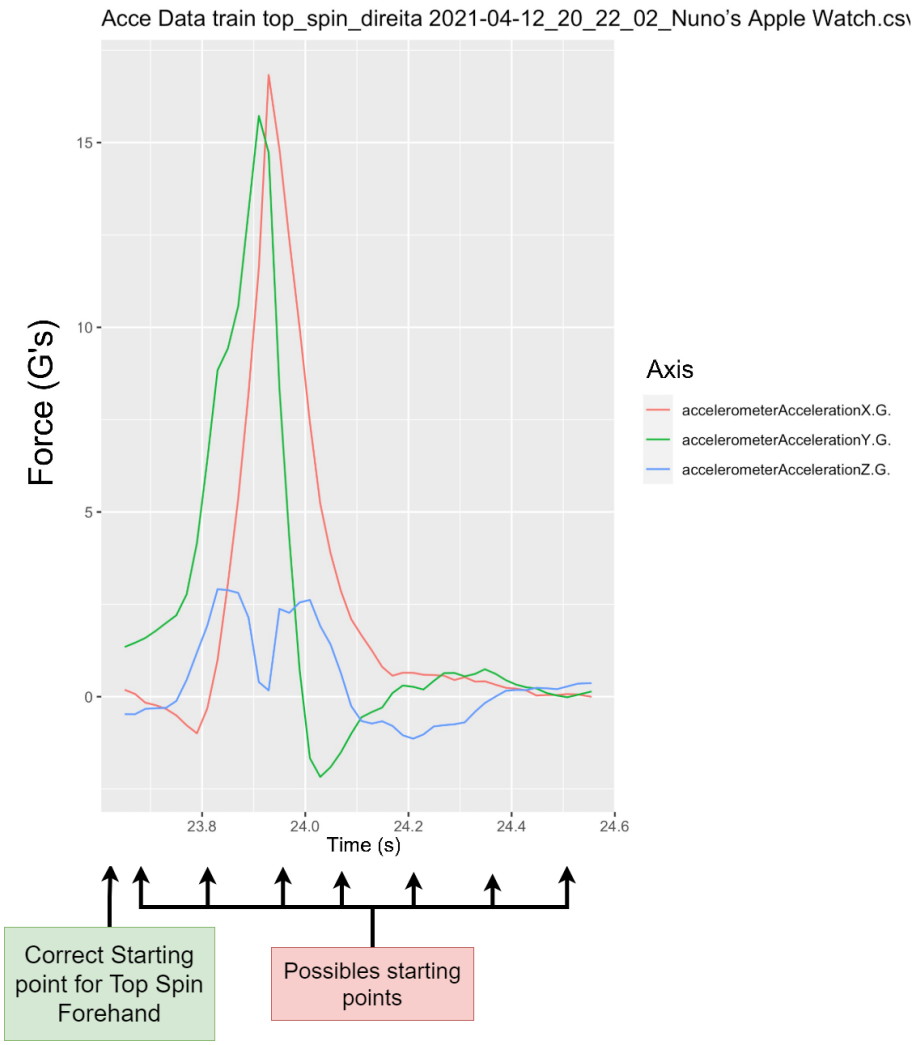



Figure 4.14: Correct and possible starting points for a table tennis stroke

be then composed of another components capable of creating a tree structure. SwiftUI advantages when compared to Storyboards are:

1. Live Preview.
2. Easy to learn and code
3. Reactive Programming
4. Can be mixed with UIKit
5. Faster Development

SwiftUI can be considered an evolution of storyboards. Every year, more SwiftUI is used in the documentation provided by Apple, indicating a foreseeable future where SwiftUI is considered the mainstream framework for future development. On the other hand, SwiftUI is still a new framework and only available for iOS13 and later (support

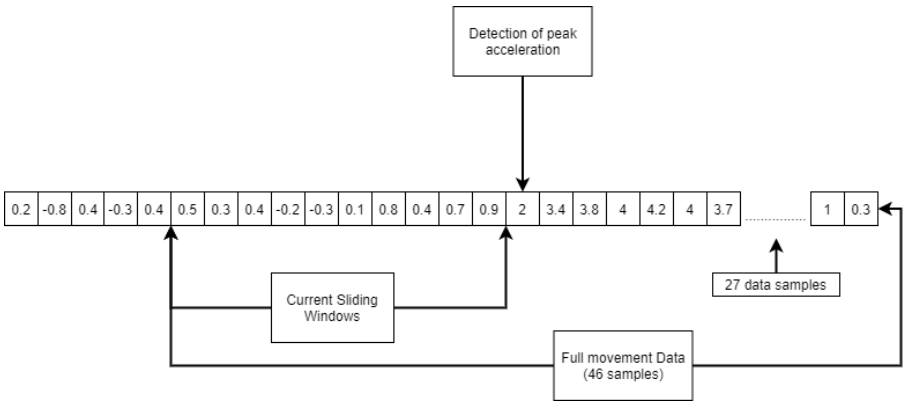



Figure 4.15: Stroke detection algorithm example containing the sliding windows and peak acceleration activation

will come later for older versions) and also the support documentation when debugging and fixing bugs can still be considered as a gap filler. After weighing the pros and cons, we have concluded to use SwiftUI for both WatchOs app development and iOS app development.

Initial app development started by requiring background processing. By needing to process and predict strokes when the application is on a background state, it must be declared as a workout app AS WatchOs restricts background use for any app that isn't a workout app. If this declaration was not done, our classifier model would only predict strokes when the Apple watch had its screen turn on and open with the app. Background mode contains a restriction where if higher CPU usage is detected (above 15%) during high periods of time, the WatchOs may suspend the application. This was not a problem as initial and final tests showed an average of 7% CPU usage recording at 50 samples per second on both gyroscope and accelerometer. As health data was also required, user permission had to be given by the user to allow the application to collect hearth rate and steps taken during the workout.

The WatchOs application consists of one main view. Figure 4.16 illustrates the available options:

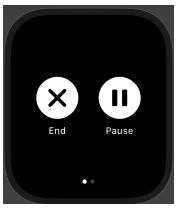
1. Play, where the athlete start a new training session
2. Statistics, where information about each training session can be access
3. Sync Statistics, Sync saved training sessions to the iPhone

The play option is then composed by the children views as seen on Fig4.17


Figure 4.16: Initial menu of WatchOs Application


(a) Start workout


(b) Type 1 table tennis robot




(c) Pause and End View

Figure 4.17: Play option sub views

The first view starts the workout by collection sensor data and running the classifier, the second view displays the available statistics such as the strokes predicted, average bpm, calories, etc... Final view pauses, continues or ends the workout, saving the workou details in the own device and sending the workout details to the iOS counterpart.

To save statistical data about each workout done, implementation of CoreDataFramework was performed. CoreData framework is a framework provided by Apple that allows to save the application's permanent data for offline use, to cache temporary data, and to add undo functionality to your app on a single device. By using the CoreData editor, provived in xcode, an entity representing a statistical model was created. This entity contained all the information correspondent to the workout session done by athlete and when the session is finished, it then sent to the iOS Application.

In the first view, training begins by collecting sensor data and running the classifier. The second view displays available statistics, such as predicted strokes, average speed per minute, calories burned, and so on. The last view pauses, resumes, or ends the workout, saves the workout details to its own device, and sends the workout details to its iOS counterpart.

To store statistical data about each workout performed, an implementation of Core Data Framework was done. Core Data Framework is a framework provided by Apple that allows to store the app's permanent data for offline use, cache temporary data, and add an undo feature to your app on a single device. Using the Core Data editor included in

Xcode, an entity was created that represented a statistical model. This entity contained all the information that corresponded to the athlete's training session and was sent to the iOS app after the session was complete.

### 4.8.4 iOS App Architecture and Development

The iOS architecture is also based on MVVM but compared to the WatchOS counterpart, the application only objective is to display the statistics obtained by the WatchOS application, making it simple regarding architecture. Its constituted by 4 views and contains a Workout Manager model that receives and stores the WorkOutSession Model by also using the Core data framework.

Fig 4.18 shows the flux diagram of the iOS application.

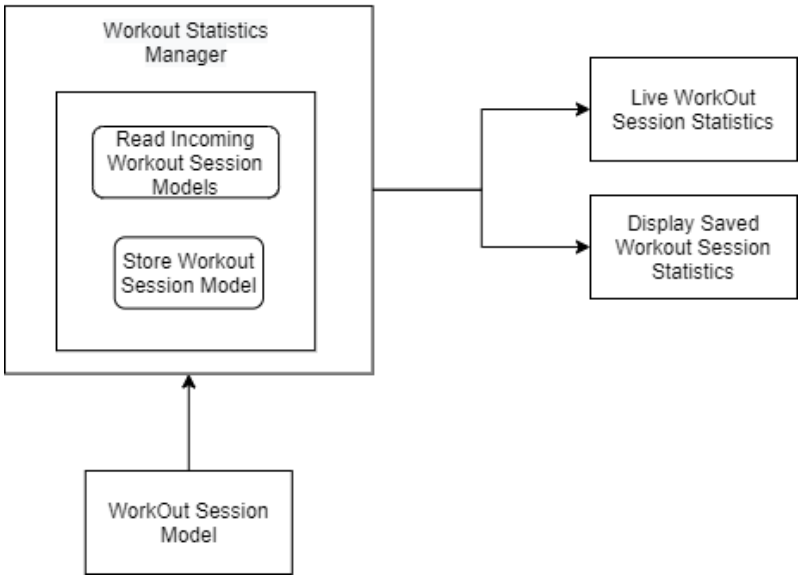



Figure 4.18: Flux diagram of iOS app

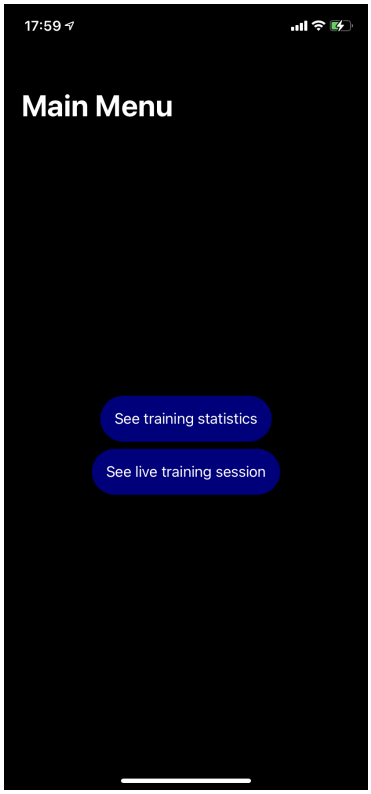
The 4 views can be seen on Fig 4.19 starting with:

**(a)** Main menu, containing 2 options, one to the see the saved workout sessions performed and another to see a live workout.

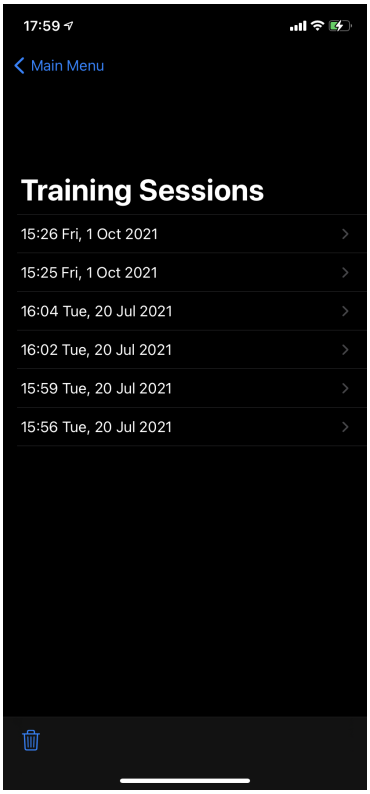
**(b)** Training sessions menu, containing a list of saved workout sessions

**(c)** Statistics about the saved workout session

**(d)** Live statistics of a workout session happening at real time

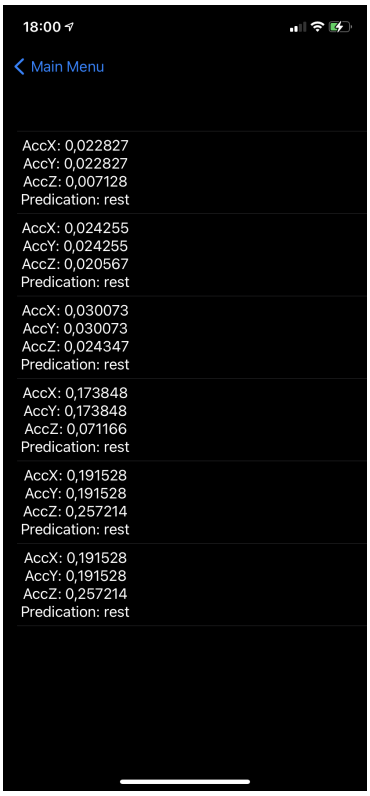



(a) App Main Screen UI




(b) List of saved workouts on the device


(c) Statistical info about the selected workout




(d) Live game feedback statistics

Figure 4.19: Paging View Sub Views

### 4.8.5 Communication between WatchOS and iOS

To communicate data between the iPhone and Apple Watch, Watch Connectivity framework provided by Apple was implemented. The framework provides multiple communication methods between the WatchOS and IOS devices. This methods are differentiated in the type of data and when the message should be delivery. Communication between devices relies on Wifi or Bluetooth, depending on what's the most efficient at the time of communication

On the WatchOS application, the following input messages methods were implemented:

1. sendMessage - method allows data to be sent to a reachable counterpart and its indented for immediate communication between the iOS application and WatchOS.

2. updateApplicationContext - used to communicate recent state information to the counterpart. When the counterpart wakes up it can use the information to update its state. Information is received by the iOS app even when the app is in background mode.

On iOS, the following counterpart output methods for received messages were implemented.

1. didReceiveMessage - This method is called in response to a message sent by the counterpart process using the sendMessage.

2. didReceiveApplicationContext - Delivers the last known context data from its counterpart that is, data sent by the counterpart using the updateApplicationContext method of its own session object. If context data is already waiting to be delivered, this method is called shortly after activating the session object for the current process.

Fig 4.20 represents the sequence diagram for when a workout session has been terminated, and Fig 4.21 represents the sequence diagram during a workout session. During a workout session, the WatchOS app will send basic statistics every time a new stroke has been predicted by using the the method sendMessage. The statistics contain the predicted stroke and the acceleration vector. When a workout is finished, the WatchOS app, sends a workout model containing all the statistics gathered during the workout by using the updateApplicationContext.

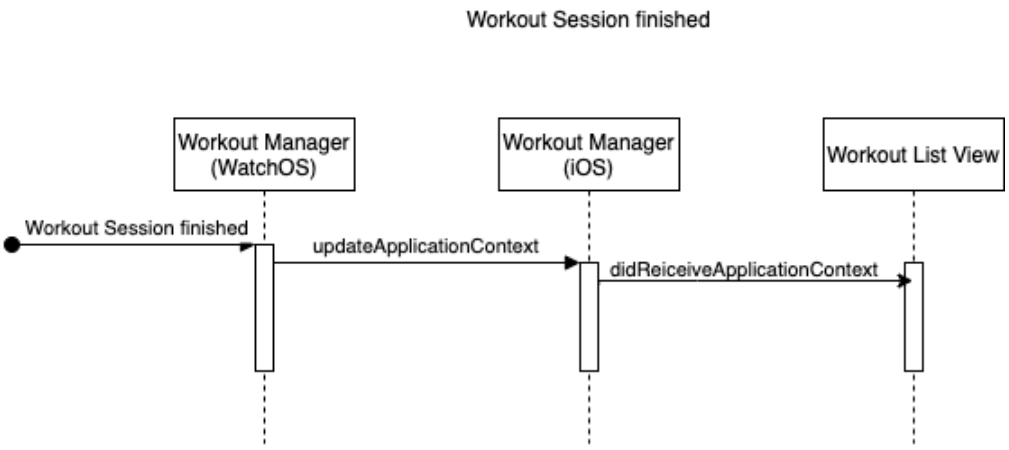



Figure 4.20: Sequence diagram of communication between WatchOS and iOS when a workout session has been finished.

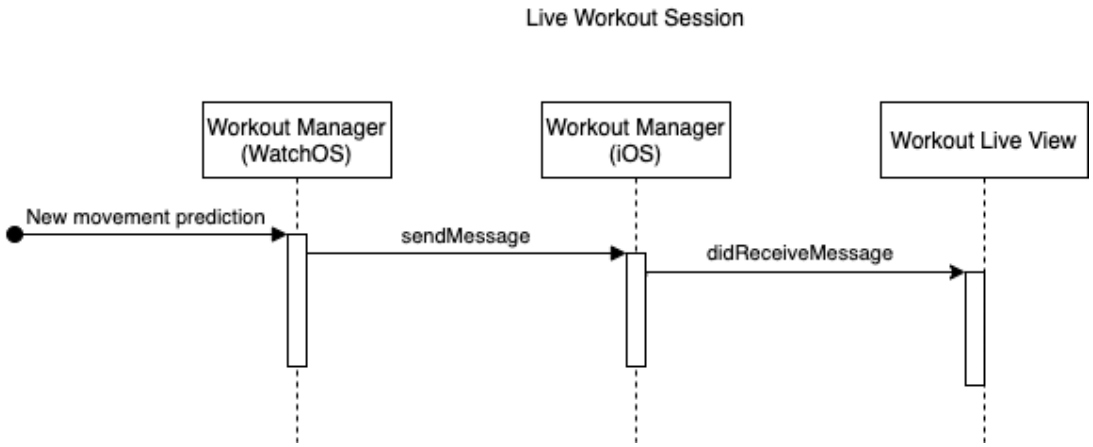



Figure 4.21: Sequence diagram of communication between WatchOS and iOS during a workout session.

### 4.8.6 CPU and Battery Consumption

CPU and battery consumption metrics were studied for both a WatchOS device ( Apple Watch SE 6) and an iOS device (iPhone XR). To access the evaluation metrics, the Xcode debugging mode in completion with the instruments application were used. On the WatchOS device, app start up followed by a workout session of 30 minutes was recorded and the CPU and battery evaluation measured. No other application were running in the background.

Fig 4.22 represents the CPU usage during a period of time inside the 30 minutes workout, starting from the app start, followed by a workout session and app closure. The workout was started at a timestamp of 00:10s. The first 10 seconds purpose were only for the application to be fully loaded and the user to have selected the play option and

be ready to click on the run button starting effectively the workout. After clicking on the run option, the application state went to the background state until the end of the test. This procedure allows to serve as a testing base for comparison for future work or app upgrades. An average of 7% CPU usage was obtained. During the recording time, spikes on CPU usage above 15% were seen but their duration were less or equal to one second and were very spaced away from each other, allowing continuous running of the application by the WatchOS. For battery metrics, the measuring tools available did not indicate the mAh( milliamp Hour) usage of the application. The only available metrics were the battery status before the app started, and after the app terminated. First, a base test was created to understand how much battery the Apple Watch uses it on its own. The test was based was conducted during 30 minutes, with no background activities and in background mode. The test indicated a 2% battery drop. For the application battery test, the Apple watch contained 100% battery before starting the application and after recording and app termination it contained 90% meaning a 8% battery usage during the 30 minutes.

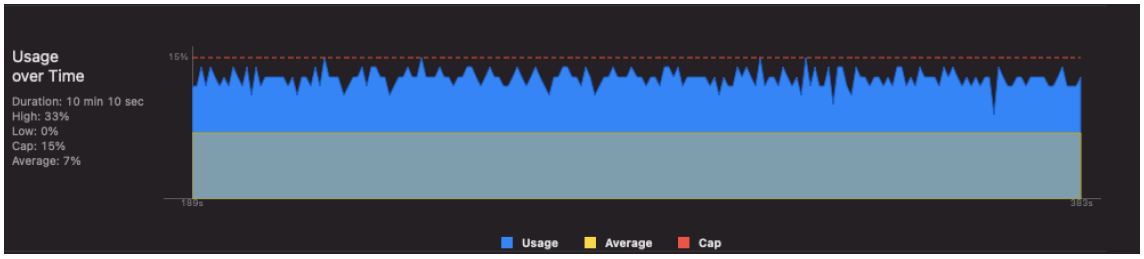



Figure 4.22: CPU usage on Watch OS, a 3:30m short representation

On iOS, the tools used to obtain the evaluation metrics did not provide a definitive average CPU usage, and for battery metrics, the same problem that happens in WatchOS also occurs on iOS. Also connection to the phone was made via cable meaning the device would be charging during the test, neglecting the battery metric. The test consisted of the user starting the app and see a live game statistics for 30 minutes. The remaining application options for the user were not tested as it showed negligible CPU usage and were not part of the focus of this thesis. Fig 4.23 shows the CPU usage on a iPhoneXR during a period of time included in the 30 minutes. The first 10 seconds reflect the application start and provide time for the user to select the option "See the live game". At the time stamp 00.10s the option was clicked and statistics already stored for the game were displayed and up coming statistics would updated the display.

The iOS application reach a max peak value of 43% of CPU usage. This value was found at application start up. By graphical analyses of the complete recorded session, it was concluded as an average of 5% to 7% CPU usage. For the battery test, first a base test was conducted with the same length of 30 minutes, the brightness at half of the

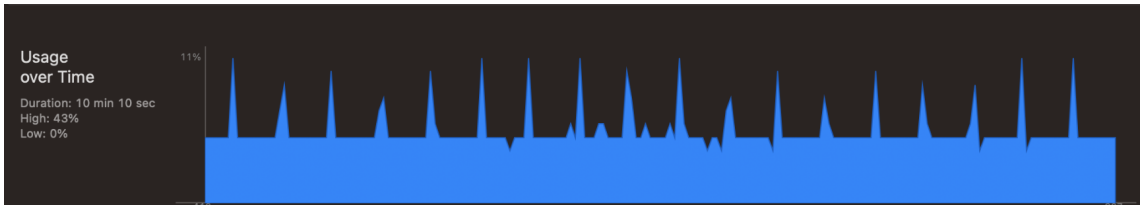



Figure 4.23: CPU usage on iOS,a 3:30m short representation

maximum available with the screen always on. Battery was at 100% before the test. The test indicated a 8% drop in battery. For the application test, the same conditions were applied. Before the start, the iPhone contained 100% battery. After the test it contained 85%. Only 7% of battery was used during the application test.

# Chapter 5

# Classifier Model and App evaluation

## 5.1 Introduction

In this chapter, a comparative development of approaches to building a classification model based on tests and statistics is made. To compare the models generated by the classifiers, the precision, recognition and F1 values were calculated for each model. First, initial batch sizes are tested and defined for both approaches. Recall, accuracy and F1 score are presented for each type of stroke. Confusion matrices for model evaluation are also presented. A final discussion details the results obtained and the advantages of each approach. A comparison between the generated models and models from other works is also shown. A scenario was created outside the laboratory to test the performance of the model when confronted with real game data. The results of this test are presented in this chapter. This chapter ends with a presentation of the usability evaluation of the app.

## 5.2 System Hardware

To generate the classifiers models, a MacBook Pro 2018 with the following hardware seen on table 5.1 was used.

| Hardware Name | Value |
|---|---|
| CPU | Intel Core i9 2.9GHz |
| GPU | AMD Radeon Pro 560X |
| OS | MacOS |
| RAM | 16GB |

Table 5.1: Hardware components of training system

## 5.3 Evolution ML classifier models

All 5 datasets were tested in Create ML Framework.

To determine the number of iterations needed for each dataset a initial test of 35 iterations, the maximum iterations available in create ML, was defined. Initial tests showed that after the 30 iteration, no gains were produced by the model generated from the datasets D1-fast, D2-slow, where a stability could be seen around 30 iterations, making it as the defined value. For D3-fast-cut, D4-slow-cut and D5-fast-slow-cut dataset 35 iterations were defined as the models benefits from more iterations.

Prediction windows size means how many observations of sensor data should be taken in consideration for a stroke detection. This value is different depending on which dataset is taken in consideration. For D1-fast and D2-slow, it is mentioned in chapter 3 that thrown balls by the robot had an interval of 1.16s for D1-fast and 1.74s for D2-slow, meaning that for a equal amount of time, a stroke can be expected. Because the data was record at 50Hz, the predicting windows size can be calculated based on the interval of thrown balls and its recording frequency, meaning a windows prediction size of 58 (1.166 / 50) for D1-fast, and 87 for D2-slow. For datasets D3-fast-cut, D4-slow-cut and D5-fast-slow-cut a windows prediction size of 46 was defined based on the length of a defined stroke mentioned in chapter 4.

Tests of Batch sizes of 16, 32, 64, 128, 256, 400 were conducted for all datasets. The results for each dataset and batch size are presented in 5.2 being the performance of each one evaluated by the F1 score metric.

| | Batch Size | | | | | |
|---|---|---|---|---|---|---|
| Dataset | 16 | 32 | 64 | 128 | 256 | 400 |
| D1 | 25.39 | 33.90 | 55.90 | 62.71 | 56,45 | 49,30 |
| D2 | 27.45 | 36,67 | 56,70 | 43,38 | 42.30 | 41,20 |
| D3 | 23,40 | 43,88 | 54,21 | 56,70 | 89,65 | 70,54 |
| D4 | 24,39 | 41,28 | 44,36 | 51,30 | 63,53 | 52.60 |
| D5 | 34,20 | 37.80 | 45,50 | 53.86 | 62,71 | 55,90 |

Table 5.2: Dataset generate classifier model performance [%]

For dataset D1-fast a batch size of 128 showed the best performance while for D2-slow a batch size of 64 was defined. For the datasets pre-process generated from the stroke detection cut algorithm, all 3 showed better results by using a batch size of 256. A higher batch size from 64 to 256 values showed a overall improvement on performance across all datasets.

Table 5.3 and 5.4 shows performance statistics about D1-fast and D2-Slow where no stroke detection cut algorithm has been applied for all the strokes. The optimal batch size calculated in Table 5.2 was used for each dataset.

Dataset D1-fast obtained a F1 score of 62.71%. The model generate from this dataset

| D1-fast dataset(fast dataset) | | | |
|---|---|---|---|
| | Precision | Recall | F1 score |
| Block | 97 | 69 | 80,63 |
| Flip (forehand) | 51 | 96 | 66,61 |
| Flip (backhand) | 35 | 21 | 26,25 |
| Rest | 94 | 100 | 96,90 |
| Top Spin (forehand) | 88 | 53 | 66,15 |
| Top Spin (backhand) | 35 | 46 | 39,75 |
| F1 score average | 62,71 | | |

Table 5.3: Dataset D1-fast classifier model generated performance metrics [%]

| D2-slow dataset (slow dataset) | | | |
|---|---|---|---|
| | Precision | Recall | F1 score |
| Block | 99 | 100 | 99,49 |
| Flip (forehand) | 54 | 70 | 60,96 |
| Flip (backhand) | 38 | 39 | 38,49 |
| Rest | 77 | 99 | 86,62 |
| Top Spin (forehand) | 47 | 53 | 49,82 |
| Top Spin (backhand) | 12 | 3 | 4,8 |
| F1 score average | 56,70 | | |

Table 5.4: Dataset D2-slow classifier model generated performance metrics [%]

had the worst performance when identifying the Flip(backhand). High performance for strokes rest and block are expected as both have unique characteristics that define their strokes.

Dataset D2-slow obtained a F1 score of 56,70%. By analyzing the precision and recall of stroke Top Spin (backhand), its possible to identify that the model can't correctly identify most Top Spin backhand strokes. No possible conclusion can me made on what's causing the model to have a bad performance when labeling this stroke compared to the others available as only accuracy, recall and F1 scores can be obtained from the Create ML Framework. Compared to D1-fast, a worst overall performance can be attributed by two factors. First a low number of samples were available. Secondly by using a windows size of 87 corresponding to 1.75s, introduction of noise can be considered a factor as the duration of stroke, beginning to the end, was defined as taking 46 observations of data corresponding to 0.92s, meaning that data with no fundamental value is being feed into our training model. In total the model performance for both datasets can be considered average, indicating that more samples should be acquired for further conclusions or a different approach should be considered.

Table 5.5 trough 5.7 represent the metrics obtained for the pre-processed datasets D3-fast, D4-slow and D5-fast-slow

When comparing the three models generated from the pre-process datasets with the

| D3-fast-cut dataset | | | |
|---|---|---|---|
| | Precision | Recall | F1 score |
| Block | 90 | 99 | 94.29 |
| Flip (forehand) | 97 | 95 | 95,99 |
| Flip (backhand) | 79 | 94 | 85.84 |
| Rest | 100 | 96 | 97.96 |
| Top Spin (forehand) | 87 | 96 | 91.28 |
| Top Spin (backhand) | 97 | 58 | 72,59 |
| F1 score average | 89.66 | | |

Table 5.5: Dataset D3-fast-cut classifier model generated performance metrics [%]

| D4-slow-cut dataset | | | |
|---|---|---|---|
| | Precision | Recall | F1 score |
| Block | 98 | 91 | 94,37 |
| Flip (forehand) | 71 | 65 | 67.86 |
| Flip (backhand) | 40 | 87 | 54.80 |
| Rest | 97 | 100 | 98.48 |
| Top Spin (forehand) | 66 | 66 | 66 |
| Top Spin (backhand) | 0 | 0 | 0 |
| F1 score average | 63.59 | | |

Table 5.6: Dataset D4-slow-cut classifier model generated performance metrics [%]

stroke detection cut algorithm applied, the model generated from dataset D3-fast-cut was the best performing model with a F1 score of 89.66%. Both 3 models show the most difficulties at identifying the stroke Top Spin (backhand). Dataset D4-slow-cut failed to identify Top Spin (backhand), and showed a low F1 score for Flip (backhand). Performance of this model was still higher compared to D5-Fast-slow-cut who could identify all 6 strokes. Introduction of noise or a lower number of samples can't be considered the reason for such low performance as noise has been almost completely removed by the stroke detection cut algorithm and classes with lower number of samples then Top Spin (backhand) and Flip(backhand) performed better than strokes with a lower sample amount. Dataset D5-fast-slow-cut had the worst F1 score when compared to the same datasets generated from the same algorithm. An increase of samples when comparing D3-fast-cut and D4-slow-cut with D5-fast-slow-cut shows no improvement in the model performance.

The output size of the classifier generate by each dataset shows no significant difference that could influence the choice of model to be implement in WatchOS App as can be seen on table 5.8. In training duration, a large difference can be seen on the models where the datasets were generated from the stroke cut algorithm. This increase can be explained by the Create Ml framework having to open a considered larger number of files, one for each stroke detected, where in datasets that were not generated from the stroke cut algo-

| D5-fast-slow-cut dataset | | | |
|---|---|---|---|
| | Precision | Recall | F1 score |
| Block | 48 | 98 | 64.44 |
| Flip (forehand) | 97 | 83 | 89.46 |
| Flip (backhand) | 69 | 18 | 28.56 |
| Rest | 99 | 96 | 97.48 |
| Top Spin (forehand) | 56 | 96 | 70.73 |
| Top Spin (backhand) | 87 | 15 | 25.59 |
| F1 score average | 62.70 | | |

Table 5.7: Dataset D5-fast-slow-cut classifier model generated performance metrics [%]

| D1-fast | D2-slow | D3-fast-cut | D4-slow-cut | D5-fast-slow.cut |
|---|---|---|---|---|
| 1.3Mb | 1.2Mb | 1.3Mb | 1.2Mb | 1,1Mb |
| 5m 10s | 04m 30s | 14m 27s | 13m 1s | 33m 47s |

Table 5.8: Model training duration and model size for each dataset

rithm, a single file represents a recording session of multiple strokes of the same player of the same stroke.

## 5.4 Evaluation of CNN-LSTM and ConvLSTM architectures models

For this two architectures, classifier models were generated for datasets D3-fast-cut, D4-slow-cut and D5-fast-slow-cut. For each architecture, model training batch sizes of 4, 16, 32, 64, 126, 256, 400 were tested.

Table 5.9 trough table 5.11 show the performance for each dataset on both architectures for each batch size. Evaluation metric used to measure the performance was the F1 score

| D3-fast-cut dataset | | |
|---|---|---|
| Batch sizes | CNN-LSTM | ConvLSTM |
| 4 | 97.89 | 97.32 |
| 16 | 97.78 | 98,04 |
| 32 | 97.98 | 97.91 |
| 64 | 97,90 | 98.07 |
| 128 | 97.94 | 98.09 |
| 256 | 97.76 | 97.82 |

Table 5.9: Dataset D3-fast-cut classifier model generated performance metrics by batch size [%]

No major performance differences between batch sizes were detected for both architectures. The optimal batch size for architecture CNN-LSTM was defined as 32 for

| D4-slow-cut dataset | | |
|---|---|---|
| Batch sizes | CNN-LSTM | ConvLSTM |
| 4 | 95.57 | 96,97 |
| 16 | 97.01 | 96,66 |
| 32 | 96,89 | 96,79 |
| 64 | 96,78 | 96,7 |
| 128 | 96,78 | 96,51 |
| 256 | 96.35 | 96,05 |

Table 5.10: Dataset D4-slow-cut classifier model generated performance metrics by batch size [%]

| D5-fast-slow-cut dataset | | |
|---|---|---|
| Batch sizes | ConvLSTM | CNN-LSTM |
| 4 | 94.30 | 95.95 |
| 16 | 96.39 | 96.02 |
| 32 | 96,29 | 95.81 |
| 64 | 95.92 | 95.99 |
| 128 | 95.54 | 95.60 |
| 256 | 95.21 | 94.97 |

Table 5.11: Dataset D5-fast-slow-cut classifier model generated performance metrics by batch size [%]

D3-fast-cut and 16 for D4-slow-cut and D5-fast-slow-cut as performance was the highest. Comparing performance on this architecture, different batch sizes show no statistical difference that can have an impact on performance. In ConvLSTM, D3-fast-cut, D4-slow-cut and D5-fast-slow-cut values of 128, 4 and 16 were defined based on the highest performing batch size result.

Different batch sizes also show no major model performance improvements.

Model performance was measure by evaluation of 10 generated models for each algorithm. The reason for this is that the neural network outcome is always different, meaning that different specific models will be produced when the same model configuration is trained on the same data.

Precision, recall and F1 score were used metrics to evaluate the generated classifier models on both architectures.

Performance evolution for each stroke and the corresponding dataset using CNN-LSTM is presented on tables 5.12, 5.13 and 5.14

All values presented are averages of the 10 generated models for each architecture and dataset.

| D3-fast-cut (128 batch size) | | | |
|---|---|---|---|
| | Precision | Recall | F1 score |
| Top Spin (forehand) | 99 | 99 | 99 |
| Top spin (backhand) | 98 | 95 | 96 |
| Block | 95 | 99 | 97 |
| Flip (forehand) | 99 | 98 | 98 |
| Flip (backhand) | 97 | 96 | 97 |
| Rest | 100 | 100 | 100 |
| F1 score average | 97.33 | | |

Table 5.12: Strokes performance for dataset D3-fast-cut [%]

| D4-slow-cut (32 batch size) | | | |
|---|---|---|---|
| | Precision | Recall | F1 score |
| Top Spin (forehand) | 95 | 93 | 94 |
| Top spin (backhand) | 91 | 97 | 94 |
| Block | 99 | 96 | 98 |
| Flip (forehand) | 99 | 97 | 98 |
| Flip (backhand) | 89 | 89 | 89 |
| Rest | 100 | 100 | 100 |
| F1 score average | 95.5 | | |

Table 5.13: Strokes performance for dataset D4-slow-cut [%]

| D5-fast-slow-cut (16 batch size) | | | |
|---|---|---|---|
| | Precision | Recall | F1 score |
| Top Spin (forehand) | 98 | 96 | 97 |
| Top spin (backhand) | 93 | 93 | 93 |
| Block | 94 | 97 | 96 |
| Flip (forehand) | 97 | 98 | 98 |
| Flip (backhand) | 92 | 92 | 92 |
| Rest | 100 | 100 | 100 |
| F1 score average | 96 | | |

Table 5.14: Strokes performance for dataset D5-fast-slow-cut [%]

Analyzing model performance across datasets, all models had a good performance with a minimum F1 score of 95.5%. The model with the highest performance was the one generated from D3-fast-cut with an F1 score of 97.33%. Both D4-slow-cut and D5-fast-slow-cut datasets had the lowest performance for Flip(backhand). A possible cause for this stroke lower performance can be a similarity with Top Spin Backhand as both strokes have the most equal characteristics. Rest motion had a 100% performance in all datasets. A possible cause for this result, is the characteristics of rest motion as it represents a major differences from the remaining strokes.

### 5.4.1 ConvLSTM

Performance evolution for each stroke and the corresponding dataset using CNN-LSTM is presented on tables 5.15, 5.16 and 5.17

| D3-fast-cut (128 batch size) | | | |
|---|---|---|---|
| | Precision | Recall | F1 score |
| Top Spin (forehand) | 98 | 99 | 99 |
| Top spin (backhand) | 99 | 90 | 94 |
| Block | 91 | 99 | 95 |
| Flip (forehand) | 99 | 96 | 97 |
| Flip (backhand) | 93 | 98 | 95 |
| Rest | 100 | 100 | 100 |
| F1 score average | 96.666 | | |

Table 5.15: Strokes performance for dataset D3-fast-cut using ConvLSTM algorithm [%]

| D4-slow-cut (4 batch size) | | | |
|---|---|---|---|
| | Precision | Recall | F1 score |
| Top Spin (forehand) | 94 | 88 | 91 |
| Top spin (backhand) | 95 | 91 | 93 |
| Block | 97 | 95 | 96 |
| Flip (forehand) | 98 | 97 | 97 |
| Flip (backhand) | 82 | 93 | 88 |
| Rest | 100 | 100 | 100 |
| F1 score average | 94.16 | | |

Table 5.16: Strokes performance for dataset D4-slow-cut using ConvLSTM algorithm [%]

The model generated from D5-fast-slow-cut had the best performance achieving 97.33%, followed by D3-fast-cut with 96.66% and 94.16% on D4-slow-cut. D4-slow-cut 4 has most difficulties at identifying Flip (backhand) with an F1 score of 88% while remaining strokes have F1 scores equal and higher then 91%. Rest class results show a 100% F1 score for all datasets. This result is equal to the results obtained using CNN-LSTM.

The worst model performance confusion matrixes out of the 10 models created, for both CNN-LSTM and ConvLSTM and for each dataset generated models are shown below and were taken to draw conclusions about the model capacity to predict a table tennis stroke, including the stroke whose model had higher difficulties to identify correctly. The CNN-LSTM confusion matrix's for the datasets tested. Fig 5.1 shows the most wrong predicted labels are the strokes Top Spin (backhand) and Flip (backhand). The probable causes for this fact, can be the similarities between both strokes. All other stroke showed a lower number of in-corrected labels. A probable cause for this can be external factors such as athlete fatigue, some noise still present on the datasets and the expected accuracy of the produced model.

| D5-fast-slow-cut (128 batch size) | | | |
|---|---|---|---|
| | Precision | Recall | F1 score |
| Top Spin (forehand) | 99 | 99 | 99 |
| Top Spin (backhand) | 96 | 95 | 95 |
| Block | 95 | 99 | 97 |
| Flip (forehand)) | 100 | 96 | 97 |
| Flip (backhand) | 97 | 94 | 96 |
| Rest | 100 | 100 | 100 |
| F1 score average | 97.33 | | |

Table 5.17: Strokes performance for dataset D5-fast-slow-cut using ConvLSTM algorithm [%]

Looking at the confusion matrix from ConvLSTM generated models, Fig 5.2, a stronger confirmation can be seen that most wrong predicted labels are between Top Spin (backhand) and Flip (backhand). Wrong predicted labels between the remaining strokes can be considered as normal when generating a classifier model, other external facts mentioned on CNN-LSTM could also be a minor representative for the incorrect labels

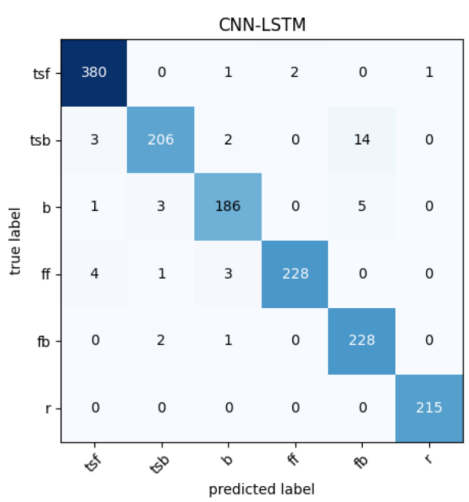
Visualization of the loss per epoch and accuracy can be seen on Fig 5.3 for CNN-LSTM and Fig 5.4 for ConvLSTM. In the graph 5.3, the orange color represents D3-fast-cut using a batch size of 32. Blue color represents D4-slow-cut using a batch size of 16, and the red color represents D5-fast-slow-cut also with a batch size of 16. On 5.4 the colors represent the same datasets as 5.3 changing only the batch size for 128, 4 and 16 values.

Both architectures started to present convergence around 20 iterations. Loss per epoch can be seen as higher using ConvLSTM, also D4-slow-cut shows some spikes on the loss graph, probably related to a low number of batch size, a consequence of Mini-Batch Gradient Descent in Adam.

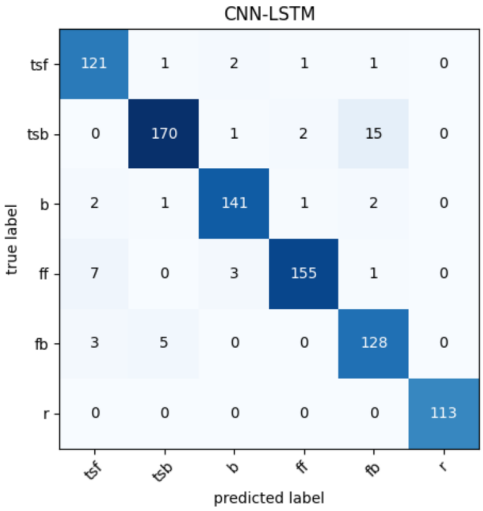
Both architectures reach good accuracy and loss values with 25 iterations.

## 5.5 Discussion

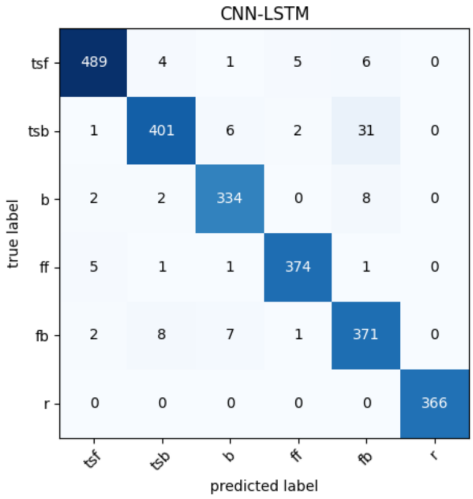
Both approaches were capable of creating an DL model capable of identifying table tennis strokes. Create ML offers a quick and easy way of creating a motion activity classifier model. By using Create ML framework, the user can abstract it self from knowing any ML algorithms or having to create features. By using a few clicks and having a dataset ready, what remains is to define the hyper parameters and a model is ready to be created. For beginners starting to work with machine learning or creating test models, Create ML can be considered an efficient approach as the skill required and the time necessary to get used to the framework are relatively low but it should also be taken in consideration that when generated results are low in accuracy and precision, than expected, the user has




(a) Confusion Matrix using CNN-LSTM on dataset D3-fast-cut




(b) Confusion Matrix using CNN-LSTM on dataset D4-slow-cut

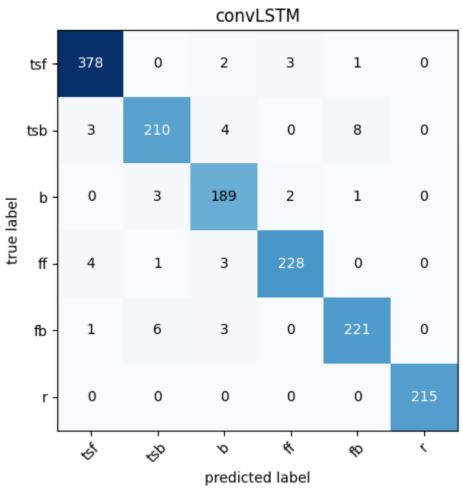



(c) Confusion Matrix using CNN-LSTM on dataset D5-fast-slow-cut

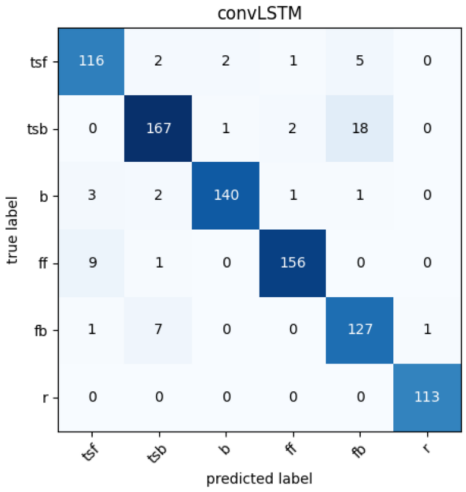
Figure 5.1: 3 Confusion matrix from CNN-LSTM for all 3 datsets used

no control of the ML training process, and the algorithms used as these are not available for consultation and changes, meaning results are much harder to improve. From the two approaches, Create ML had the lowest performance. Only D3-fast-cut dataset generated DL model presented a good performance. A possible cause for the low performance for the remaining dataset generated classifier models cannot be concluded as Create ML only offers precision and recall as evaluation measures. Confusion matrix, learning rates, or the algorithms applied by Create ML activity classifier model creating could help understand what's happening behind the scenes.

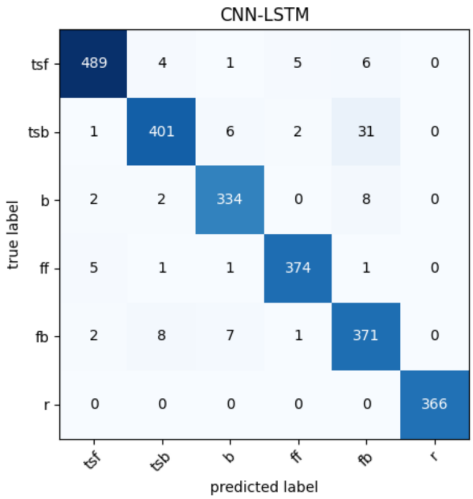
Performance wise, good results can be achieved, but when looking for maximum performance from an DL model, a controllable approach should be considered such as TensorFlow, Keras, and PyTorch. Use of the LSTM algorithm in both approaches showed good results and confirms the advantages of this algorithm when working with temporal




(a) Confusion Matrix using CNN-LSTM on D3-fast-cut




(b) Confusion Matrix using CNN-LSTM on D4-slow-cut




(c) Confusion Matrix using CNN-LSTM on D5-fast-slow-cut

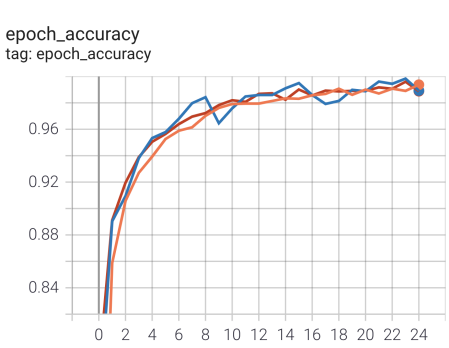
Figure 5.2: 3 Confusion matrix from CNN-LSTM for all 3 datasets used

spatial data whose works in chapter 2 had already concluded.

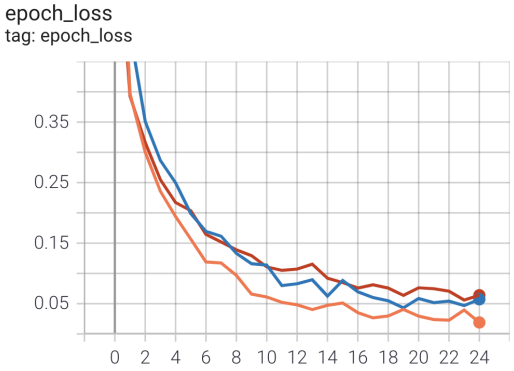
In Create ML, the batch size highly influences the performance model, while in CNN-LSTM and ConvLSTM approaches, it showed good performance across all tested batch sizes and performance was almost similar between these two

On CreateML D2-slow had the worst performance compared to the remaining datasets tested. The probable causes can be concluded as a lot of factors. First, a low number of data samples were used when comparing to D1-fast. Secondly, having a windows prediction size of 1.74s allows noise to have an impact on final results, as when comparing to D4-slow whose data samples only contain full stroke data presented better results. A possible solution to improve this dataset can be an increase in data samples recorded to improve performance as bigger datasets samples has shown better performance.

A maximum performance of 97.33 % was achieved on both CNN-LSTM and ConvL-




(a) Accuracy per epoch when using CNN-LSTM model for training.
D3-fast-cut 32 batch size
D4-slow-cut 16 batch size
D5-fast-slow-cut 16 batch size




(b) Loss per epoch when using CNN-LSTM model for training.
D3-fast-cut 32 batch size
D4-slow-cut 16 batch size
D5-fast-slow-cut 16 batch size

Figure 5.3: Loss per epoch when using CNN-LSTM architecture for model for training on the 3 datasets

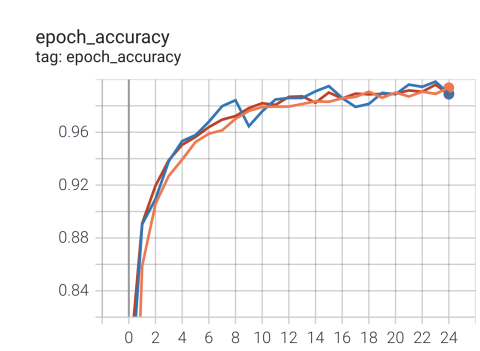
STM. By using the ConvLSTM algorithm, only a minor improvement of 0.66% versus the CNN-LSTM was achieved.

Both CNN-LSTM and ConvLSTM generated models showed difficulties on identifying correctly Flip (backhand) with the confusion matrix's indicating a higher number of wrong labeled strokes between Top Spin backhand and Top Flip (backhand). This adds to the possible cause of similarities between the strokes.

Dataset D5-fast-slow-cut containing a merge of samples from D1-fast and D2-slow processed with the stroke detection algorithm created in chapter 3 should be consider for use in applications as the number of samples is much higher than the remaining datasets and presents a high F1 score.

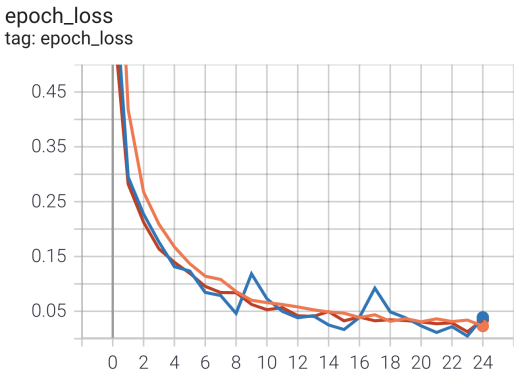
Comparing both approach's, Create ML generates a model that's only functional in the Apple Ecosystem, limiting the devices and the target market as the package used Core ML Tools, only allows to convert other ML models into a CoreML but not the reverse. On the other side, the Tensorflow model allows us to convert the model to other ML output model frameworks types but requires other frameworks to provided us with the right tools to convert the model, and it cannot be guarantee as most restrictions come from the version of Tensorflow as seen on this thesis.

Table 5.18 shows our work compared with the works developed in table tennis mentioned in chapter 2. Our work it's the first where a table tennis classifier model has been integrated into a wearable device. Only one device in the athletes wrist was needed to record motion data. The app developed for predication the stroke uses less then 15% of CPU, leaving space for data gathering of other sensors such the magnetometer who in previous studied works showed an improvement when gathered with accelerometer and gyroscope. The best performing dataset generated classifier was trained based in 1564




(a) Accuracy per epoch when using convLSTM model for training.

D3-fast-cut 128 batch size
D4-slow-cut 4 batch size
D5-fast-slow-cut 16 batch size




(b) Loss per epoch when using convLSTM model for training

D3-fast-cut 128 batch size
D4-slow-cut 4 batch size
D5-fast-slow-cut 16 batch size

Figure 5.4: Loss per epoch when using ConvLSTM architecture for model for training on the 3 datasets

samples, making it the second higher whose works used IMU devices. Achieving a precision of 97.83% and a F1 score of 97.7%, the developed classifier model can be considered the 2º best from all the studied works generated models.

| Studies | Type of device | Sensors Used | Samples | Approach | Best algorithm | Precision | F1 score |
|---|---|---|---|---|---|---|---|
| Table Tennis Stroke Recognition Based on Body Sensor Network | Body sensor networks | Accelerometer, Gyroscope, Magnetometer | 270 | Machine Learning | SVM | 97.41% | 97.4% |
| LSTM-Guided Coaching Assistant for Table Tennis Practice | 3 IMU sensors modules(MPU-9150S) | Accelerometer, Gyroscope | 1260 | Deep Learning | RNN-LSTM | 93% | 86.3% |
| Table Tennis Stroke Recognition Using Two-Dimensional Human Pose Estimation | Raspberry Pi with cameras | Vibration sensor, Video Camera | 22111 | Machine Learning and Deep Learning | TCN | 99.37% | ? |
| Sensor-based stroke detection and stroke type classification in table tennis | miPod | Accelerometer, Gyroscope | 1982 | Machine Learning | SVM | 95.7% | 96.9% |
| Classification of Table tennis strokes in wearable device using deep learning | Apple Watch | Accelerometer, Gyroscope | 1982 | Deep Learning | ConvLSTM | 97.83% | 97.7% |

Table 5.18: Comparative characteristics of each developed system/model in table tennis, 1-(Liu et al., 2019), 2-(Lim et al., 2018), 3-(Kulkarni and Shenoy, 2021), 4-(Blank et al., 2015)

## 5.6 Real Life Validation Test on Athletes

A test was conducted to evaluate the ConvLSTM model performance with a scenario most close possible to a real game but with the player strokes defined beforehand to have a final comparison. 5 Athletes were asked to perform 5 times each detectable stroke by the model. The athlete would perform 5 times a top spin (forehand), followed by top

spin(backhand), etc. All game rules were applied. In case of a drop of ball or game point won, no pauses on the application would be applied.

Table 5.19 through 5.23 shows for each athlete, the stroke and how many times it predicted it.

| Athete 1 | |
|---|---|
| Top spin (forehand) | 5 |
| Top spin (backhand) | 4 |
| Flip (forehand) | 7 |
| Flip (backhand) | 6 |
| Block | 4 |

Table 5.19: Number of detected strokes on subject 1

| Athete 2 | |
|---|---|
| Top spin (forehand) | 6 |
| Top spin (backhand) | 3 |
| Flip (forehand) | 6 |
| Flip (backhand) | 4 |
| Block | 5 |

Table 5.20: Number of detected strokes on subject 2

| Athete 3 | |
|---|---|
| Top spin (forehand) | 5 |
| Top spin (backhand) | 3 |
| Flip (forehand) | 4 |
| Flip (backhand) | 6 |
| Block | 6 |

Table 5.21: Number of detected strokes on subject 3

| Athete 4 | |
|---|---|
| Top spin (forehand) | 7 |
| Top spin (backhand) | 3 |
| Flip (forehand) | 4 |
| Flip (backhand) | 6 |
| Block | 6 |

Table 5.22: Number of detected strokes on subject 4

| Athete 5 | |
|---|---|
| Top spin (forehand) | 4 |
| Top spin (backhand) | 5 |
| Flip (forehand) | 4 |
| Flip (backhand) | 3 |
| Block | 4 |

Table 5.23: Number of detected strokes on subject 5

The first conclusion taken from the present tables indicate that on Athlete 1, the application detected more then a total of 25 strokes while on athletes 2,3 and 5 less then 25 strokes. This results can be explained due to first: the threshold defined in the stroke detection algorithm and secondly, the classifier model on the application, detecting some strokes as rest. All the athletes obtained an accuracy of at least 78% with the higher accuracy being 82% on athlete 3. Fig 5.5 shows the accuracy obtained for each athlete. The average test accuracy was 80%.

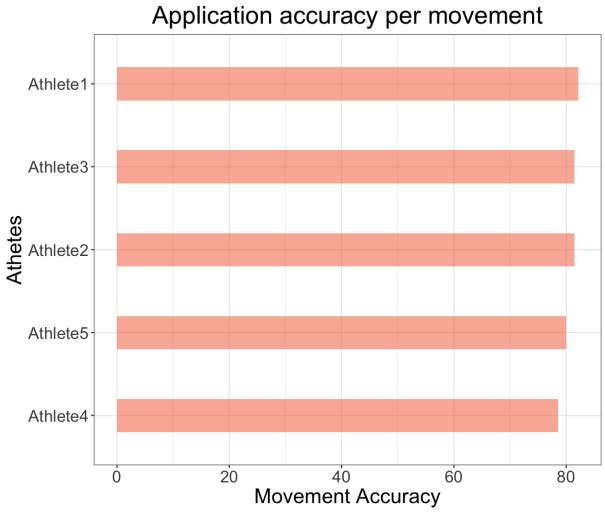



Figure 5.5: Stroke detection accuracy for each athlete

Most of the strokes detection values are within the expected values according to the classifier model accuracy for each stroke. Fig 5.6 shows the percentage for each correct and negative stroke identification. A negative identification meaning that the application identified the stroke more or less times then the 5 expected.

The lowest value of correct identified strokes corresponds to the top spin backhand. This is expected as the generated model had most difficulties on identifying strokes between top spin backhand and flip backhand. Overall the application provides a good result but improvements can be made on both stroke prediction algorithm and also on the classifier model with a focus on the top spin forehand.

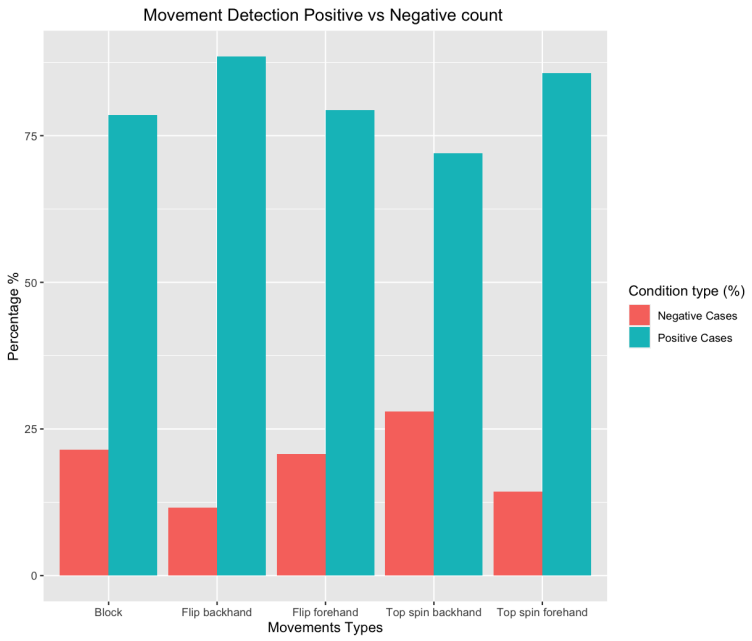



Figure 5.6: Positive vs Negative count of strokes detected

## 5.7 App user feedback

To understand if the application had real value for the athletes and coaches, an inquiry was made and presented to 7 subjects. The subjects contained 5 athletes who participated in the real game scenario plus 2 coaches. All of them were presented with the app on both components, iOS and WatchOS modules. An inquiry containing 4 questions were made to each one of the subjects. The inquiry containing the following questions:

1. Would you use the app in future training's?
2. Does the app contain interesting statistical info?
3. What things can be considered as missing ?
4. What do you think can be improved?

For question number 1 and 2, all subjects answered positively. On question number 3, the answers focused on a new UI, more statistics info (in special the lateral displacement) and account creating for saving data and the accessing it online.. Question number 4 answers consisted on having a higher accuracy on strokes detected. This questionnaire indicates a good base app has been formed to help athletes and players but for commercialization, new details should be added, in special a new appealing UI and a review to the classifier algorithm to find some minor upgrades.

# Chapter 6

# Conclusion

This dissertation aimed to develop a mobile app capable of detecting table tennis strokes. To achieve this goal, Chapter 2 researched on what was already done and how it was done. As there are no public datasets containing table tennis game data, a methodology was developed with professional players and coaches to create a high quality dataset. It started with a questionnaire developed with players and coaches to understand which strokes were most important. The list of strokes was used to explain and record the creation of 2 main datasets, one with fast strokes and one with slow strokes, using a robot monitored by athletes and coaches. The methodology is described in chapter 3. It proved difficult to capture all key strokes due to time constraints, athletes freely participating in multiple data recording sessions, separate training sessions and players being injured. As a result, a reduced number of strokes were selected to be recorded

On Chapter 4, pre-processing of data and creation of 3 new processed datasets based from the initial 2 datasets captured, where a complete stroke detection cut algorithm has been applied providing multiple samples for creation of the classifier using two approaches. One by using Create ML and the other by CNN-LSTM and ConvLSTM algorithms on Tensorflow. Difficulties in achieving a good model performance when using Create ML were found and the possible causes were not possible to be determined as minimal statistics are provided by the framework and implementation of the training algorithm is unknown (not open source), and only changes to hyper-parameters are available to change. The second approach based on CNN-LSTM and ConvLSTM required another pre-processing of data. This approach only used pre-processed datasets from the original 2.

Both CNN-LSTM and ConvLSTM achieved a maximum performance of 97.33% f1 score but on different datasets. The dataset chosen was D5-fast-slow-cut as it contained the higher number of samples and the higher f1 score. Having a good classifier model, app architecture, development and implementation of the classifier, consisting of two apps, the iOS where statistics were displayed to the user and the WatchOS who ran the classifier model, was also described in the same chapter. Restrictions on CPU-usage when in back-

ground proven to not be a problem due to the classifier model only using accelerometer and gyroscope sensor data at a frequency of 50Hz. A higher number of sensors or a higher recording frequency could prevent the app from working in background mode.

Chapter 5 evaluates and compares the generated classifier model with models created from other papers based on table tennis stroke detection, and a discussion about advantages and disadvantages was presented.

The application for both iOS and WatchOS correspond to what the company had in mind, and the performance of the classifier model allows the application to be applied to training sessions improving the efficient of a coach and the athletes training sessions. Also upgrades should be made in terms of samples from other players to train the classifier model allowing the model to fit a large population such as a normal player or world league player.

## 6.1 Future Work

For commercial use and to achieve a greater number of players, the classifier model should have as base a large number of samples from different levels competition. Even tough the model generated performances really well in test data, identifying strokes at pro competitions levels can be significantly different as the motion of the stroke is more refined when it comes from pro players. A implementation of an algorithm capable to detect when a stroke has occur, by using not only the accelerometer but also gyroscope and even the magnetometer could be an upgrade to the stroke detect algorithm as well a replacement for detecting when a player is resting. Also as mentioned in chapter 2, visual systems capable of identifying the athletes stroke were already developed. A system consisting of both visual and inertia data sensor could be the perfect training system and guidance for players and coaches. Finally the developed system only works on Apple AI capable devices. For future developments, an investigation into android AI capable devices for implementing of the same model into this devices is next as the model generated in this thesis has as base the Tensorflow framework which can be converted to other types of model due to its wide range of support and it was shown on this thesis.

# References


Apple (2021). Create ml application and framework documentation. URL: `https://developer.apple.com/machine-learning/create-ml` (accessed: 2021-10-28).

Aughey, R. J. and Falloon, C. (2010). Real-time versus post-game GPS data in team sports. *Journal of Science and Medicine in Sport*, 13(3):348–349.

Barshan, B. and Yüksek, M. C. (2014). Recognizing Daily and Sports Activities in Two Open Source Machine Learning Environments Using Body-Worn Sensor Units. *The Computer Journal*, 57(11):1649–1667.

Blank, P., Hoßbach, J., Schuldhaus, D., and Eskofier, B. M. (2015). Sensor-based stroke detection and stroke type classification in table tennis. *ISWC 2015 - Proceedings of the 2015 ACM International Symposium on Wearable Computers*, 1(September):93–100.

Brownlee, J. (2020). LSTMs for Human Activity Recognition Time Series Classification.

Chen, Y. and Xue, Y. (2016). A Deep Learning Approach to Human Activity Recognition Based on Single Accelerometer. *Proceedings - 2015 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2015*, pages 1488–1492.

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

Cust, E. E., Sweeting, A. J., Ball, K., and Robertson, S. (2019). Machine and deep learning for sport-specific movement recognition: a systematic review of model development and performance. *Journal of Sports Sciences*, 37(5):568–600.

Ghazali, N. F., Shahar, N., Rahmad, N. A., Sufri, N. A., As'ari, M. A., and Latif, H. F. (2018). Common sport activity recognition using inertial sensor. In *Proceedings - 2018 IEEE 14th International Colloquium on Signal Processing and its Application, CSPA 2018*, pages 67–71. Institute of Electrical and Electronics Engineers Inc.

Hessen, H.-O., Tessem, A. J., and Bach, K. (2016). Human Activity Recognition With Two Body-Worn Accelerometer Sensors. Technical report, Norwegian University of Science and Technology.

Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.

Hölzemann, A. and Van Laerhoven, K. (2018). Using Wrist-Worn Activity Recognition for Basketball Game Analysis. In *Proceedings of the 5th international Workshop on Sensor-based Activity Recognition and Interaction*, number 18 in 1, pages 1–6, New York, NY, USA. ACM.

Incel, O. D., Kose, M., and Ersoy, C. (2013). A Review and Taxonomy of Activity Recognition on Mobile Phones. *BioNanoScience*, 3(2):145–171.

Kulkarni, K. M. and Shenoy, S. (2021). Table Tennis Stroke Recognition Using Two-Dimensional Human Pose Estimation. *CVPR Sports Workshop 2021 (7th International Workshop on Computer Vision in Sports)*.

Lim, S. M., Oh, H. C., Kim, J., Lee, J., and Park, J. (2018). LSTM-Guided Coaching Assistant for Table Tennis Practice. *Sensors (Basel, Switzerland)*, 18(12):1–14.

Liu, R., Wang, Z., Shi, X., Zhao, H., Qiu, S., Li, J., and Yang, N. (2019). *Table tennis stroke recognition based on body sensor network*, volume 11874 LNCS. Springer International Publishing.

Marr, B. (2018). The Key Definitions Of Artificial Intelligence (AI) That Explain Its Importance.

McCarthy, J., Minsky, M., Rochester, N., and Shannon, C. E. (2006). A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955. *AI Mag.*, 27:12–14.

Munivrana, G., Petrinović, L. Z., and Kondrič, M. (2015). Structural Analysis of Technical-Tactical Elements in Table Tennis and their Role in Different Playing Zones. *Journal of Human Kinetics*, 47(1):197–214.

Neville, J., Wixted, A., Rowlands, D., and James, D. (2010). Accelerometers: An underutilized resource in sports monitoring. In *Proceedings of the 2010 6th International Conference on Intelligent Sensors, Sensor Networks and Information Processing, ISSNIP 2010*, pages 287–290.

Nguyen, L. N. N., Rodríguez-Martín, D., Català, A., Pérez-López, C., Samà, A., and Cavallaro, A. (2015). Basketball activity recognition using wearable inertial measurement units. *ACM International Conference Proceeding Series*, 07-09-September-2015(December).

Nuno Ferreira (2021). Table Tennis Data Set's.

Ordóñez, F. J. and Roggen, D. (2016). Deep Convolutional and LSTM Recurrent Neural Networks for Multimodal Wearable Activity Recognition. *Sensors 2016, Vol. 16, Page 115*, 16(1):115.

Org, O. (2021). Table Tennis - Summer Olympic Sport.

Pärkkä, J., Ermes, M., Korpipää, P., Mäntyjärvi, J., Peltola, J., and Korhonen, I. (2006). Activity classification using realistic data from wearable sensors. *IEEE Transactions on Information Technology in Biomedicine*, 10(1):119–128.

Perš, J. and Kovačič, S. (2000). A System for Tracking Players in Sports Games by Computer Vision. Technical Report 5, Norwegian University of Science and Technology.

Qiao, F. (2021). Application of deep learning in automatic detection of technical and tactical indicators of table tennis. *PLOS ONE*, 16(3):e0245259.

Sainath, T. N., Vinyals, O., Senior, A., and Sak, H. (2015). Convolutional, long short-term memory, fully connected deep neural networks. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 2015-August:4580–4584.

Sensorsoscar, S., Lara, D., and Labrador, M. A. (2018). A Survey on Human Activity Recognition using Wearable Sensors Oscar. *IEEE COMMUNICATIONS SURVEYS AND TUTORIALS*, 15(3).

Shi, X., Chen, Z., Wang, H., Yeung, D.-Y., Wong, W.-k., and Woo, W.-c. (2015). Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting. *arXiv*.

Shoaib, M., Scholten, H., and Havinga, P. J. (2013). Towards physical activity recognition using smartphone sensors. *Proceedings - IEEE 10th International Conference on Ubiquitous Intelligence and Computing, UIC 2013 and IEEE 10th International Conference on Autonomic and Trusted Computing, ATC 2013*, pages 80–87.

Sports, T. E. (2021). Most Popular Sports List.

Thomas, B. (2021). Sensorlog v4.0. URL: `http://sensorlog.berndthomas.net/` (accessed: 2021-02-11).

Wu, Y., Lan, J., Shu, X., Ji, C., Zhao, K., Wang, J., and Zhang, H. (2018). ITTVis: Interactive Visualization of Table Tennis Data. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):709–718.

Wundersitz, D. W., Josman, C., Gupta, R., Netto, K. J., Gastin, P. B., and Robertson, S. (2015). Classification of team sport activities using a single wearable tracking device. *Journal of Biomechanics*, 48(15):3975–3981.

Xia, K., Huang, J., and Wang, H. (2020). LSTM-CNN Architecture for Human Activity Recognition. *IEEE Access*, 8:56855–56866.

Zeng, M., Nguyen, L. T., Yu, B., Mengshoel, O. J., Zhu, J., Wu, P., and Zhang, J. (2015). Convolutional Neural Networks for human activity recognition using mobile sensors. In *Proceedings of the 2014 6th International Conference on Mobile Computing, Applications and Services, MobiCASE 2014*, pages 197–205. Institute of Electrical and Electronics Engineers Inc.