POLITECNICO DI TORINO

III Facoltà di Ingegneria dell'Informazione

Corso di Laurea in Ingegneria Informatica (Computer Engineering)

Tesi di Laurea Magistrale

MY-AIR Project

Study on Semantic Location and Activity Recognition Algorithms
for iOS Systems

Relatori:
prof. Giovanni Malnati
prof. Elena Baralis

Candidato:
Giovanni Clemente Monna

Anno accademico 2017-2018

Dedicated to my father, my mother, and my sister.

They gave me the possibility to study

thousands of miles away from home,

supporting me every moment.

ACKNOWLEDGMENTS

I am grateful to my advisor Professor Ouri Wolfson. He supported me, together with Professor Jie Lin, to whom I am very grateful, too, by telling me where I had to improve my work and focus more during my research. I cannot thank them enough for their hard work. Thank you, Professor Wolfson and Professor Lin.

I would like to thank also my Italian academic advisor, Professor Giovanni Malnati, who accepted to be my advisor even if several time zones separate us. I would finally like to give wholehearted thanks to Professor Elena Baralis, who accepted to be part of the committee to replace Professor Malnati, unable to be present during the defense in Chicago, and who gave me some precious advice in order to achieve the best possible results.

I would also like to thank everyone who helped me get through this long period of work, my friend, my colleagues, and my family, without whose support I would not have been able to conclude this thesis.

GCM

TABLE OF CONTENTS

CHAPTER **PAGE**

TABLE OF CONTENTS (continued)

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES (continued)

LIST OF ABBREVIATIONS

CNN	Convolutional Neural Network
DT	Decision Tree
FB	Feed-back
FF	Feed-forward
FNN	Feedforward Neural Network
GMM	Gaussian Mixture Model
MSE	Mean Squared Error
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
SLAR	Semantic Location and Activity Recognition
SVM	Support Vector Machine

SUMMARY

This thesis provides an algorithm for concurrent detection of semantic location and activity of the user, within a range of nine different possibilities. They are the combination between two possible semantic location states (”indoor” and ”outdoor”) and five different human activities (”stationary”, ”walking”, ”running”, ”biking” and ”automotive”). The final output values are {*automotive, indoor_stationary, indoor_walking, indoor_running, indoor_biking, outdoor_stationary, outdoor_walking, outdoor_running, outdoor_biking*}. The recognition of these nine possible states is based on data coming from different smartphone sensors, selected between the less consumptive ones and basing on previous research works, for the application to be feasible and implementable. This branch of the research has been conducted on iOS systems, trying to overcome the limitations that this operative system presents, if compared to the Android one.

For semantic location recognition, the research exploits: the behavior of the magnetic field variation, which is different indoors from outdoors, considering its variance during a 1 second time window with a fixed segmentation approach; the difference of direct sunlight between indoors and outdoors, and also between daytime and nighttime (reason for taking into consideration current time of day, also); the cellular signal strength attenuation when indoors, due to ceiling and walls surrounding the user; the amplitude of noise surrounding the user, which is much higher outdoors than indoors, due to traffic and other environmental noises.

SUMMARY (continued)

For human activity recognition, the research exploits: the variance of accelerometer values on a 1 second window with a fixed segmentation approach, not the raw ones but rather the ones coming from a refined and processed gathering done by the Device Motion framework, that fixes them using the gyroscope, in order to obtain an acceleration measuring that is not affected by gravity and orientation of the device; the current speed of the user, only outdoors, which helps distinguishing between various activities; the cadence of the user using the pedometer, i.e., the number of steps per second she takes, which helps distinguishing between different ”on foot” activities.

All these data have been collected together with a label, to find a machine learning model capable of classifying the nine different activities and then to implement it into a smartphone application by using the Core ML framework for iOS. The best model found for classification is a Deep Learning Neural Network. Besides the model itself, the application, which classifies in real-time, needs to implement a particular algorithm in order to maximize the accuracy of classification (by helping it with the Core Motion activity framework embedded in iOS systems), minimize the delay for recognizing a switch of activity (by studying the length of the time window for the delay and the associated behavior of the identification) and minimize the power consumption of the application (by exploiting the GPS sensor only in certain circumstances), since it needs to run continuously for a proper classification. All the details about the algorithm can be found in chapter 3 of this thesis.

CHAPTER 1

INTRODUCTION

The World Health Organization (WHO) identified air pollution as the world's most significant single environmental health risk. In the U.S., the Environmental Protection Agency's AirNow program has been providing hourly air quality data and daily forecasts to the public since 1998. The data source for AirNow is the ambient air quality monitoring data from the U.S. EPA's air monitoring station network. In addition to air quality data, AirNow's use of the EPA Air Quality Index (AQI) ensures air quality data is presented with human health in mind. However, nowadays there are several very crowded zones (40 million inhabitants and more) that are located about 25 miles away from the closest $PM_{2.5}$ monitor. In addition, the published AQI or pollutant concentration values do not reflect the actual pollutant intake of individuals. This separation happens because personal intake also depends on the person's microenvironment (indoor, outdoor, in vehicle), activity (such as sitting, sleeping, running, walking, biking), and physiology (age, gender, health condition), in addition to the detected air pollution levels.

The study proposed in this thesis constitutes a branch of the project that was born to focus on determining the feasibility and accuracy of the MY-AIR (Monitor Your Air-Pollution Intake and Risk) app. MY-AIR app will try to combine the AirNow data with the users' location and activity recognized by their smartphone and user-specified physiological conditions to estimate personalized intake of $PM_{2.5}$ (i.e., PPI, personal pollutant intake) over time. The app will display the PPI value on the user's smartphone so that individuals can adjust their daily activities accordingly to avoid harmful exposure and even take proactive actions to mitigate and prevent further air pollution.

The objective of this branch of the MY-AIR project is to estimate the semantic location and activity of the user within this location. Overall, SLAR is one of the three main factors determining the rate at which a user ingests pollution. The other two are local outdoors air-pollution level and medical record. The semantic locations will be classified as indoor, outdoor and in vehicle. The activities will be classified as stationary, walking, running and biking. The reason to recognize semantic location and activity jointly is that the two recognition tasks reinforce each other. For example, a biking user is likely to be outdoors rather than in a vehicle. Therefore, the activity will be classified in one of the nine categories in the activity set {*automotive, indoor_stationary, indoor_walking, indoor_running, indoor_biking, outdoor_stationary, outdoor_walking, outdoor_running, outdoor_biking*}. Motion-sensor based SLAR has been studied extensively in past years, and currently both iOS and Android provide Application Program Interfaces (APIs) for continuous activity recognition, which we exploit in this research to reinforce the detection of our model and to increase the accuracy of prediction. Reports about the accuracy of Google's activity recognition vary widely (52% to 99%) depending on the activity, sampling rate, device, and other conditions. The proposed algorithm for semantic location and activity recognition is a classifier which reads raw sensor data, processes it to extract features, based on which it then classifies the motion activity, trying to overcome the limitations described in section 1.1.

1.1 Technical challenges

Classification done by a machine learning model is often **uncertain** in this case. For example, GPS receiver accuracy is usually much higher outdoors than indoors, but in some situations an indoors smartphone close to a window may have higher accuracy than an outdoor one in an urban canyon. Thus the output of the SLAR classifier will be a vector of nine confidence values, each one associated with a single target value within the nine possible outputs. The confidence value is the probability of the i_{th} category, according to the characteristics of the Core ML framework, which will be used to implement the machine learning classifier into the iOS application.

The iOS application algorithm will implement some studied workarounds, to overcome the uncertainty of model's output and to maximize the accuracy of the recognition and provide some segmentation. These workarounds will not only be able to maximize accuracy, but they will also reduce **battery power consumption**, which is another overwhelming technical challenge. A significant challenge which arises in continuous recognition, as required by this research, is in fact to minimize power consumption. The reason is that there is a trade-off between classification accuracy and power consumption in the sense that accuracy depends on the sensors used and the sensor sampling rate. Power-efficient recognition is often achieved by reducing the use of power-inefficient sensors such as GPS, using power-efficient sensors such as accelerometer, cellular signal strength, and light instead. For example, it is possible to allow the GPS sensor to be read only when the power-efficient sensors have determined that, with high likelihood, the user has shifted from indoors to outdoors. Furthermore, it turns out that the combination of periodical and triggered indicators can help with another problem of activity recognition, namely **delay**. Specifically, for a single classifier, the delay from the time the activity commences until the time it is recognized and indicated can be up to 30 seconds, and the combination can overcome this limitation, which the iOS application algorithm tries to minimize, as well.

To these technical challenges related to the project itself, we need to add the technical challenges arising from the usage of the iOS operating system, which is a very closed system where most of the sensors cannot be accessed and need to be used by studying some workarounds. All of them are described in next chapters.

In chapter 2, there is a review of relevant work and research that has been done in both fields of indoor/outdoor detection and on human activity recognition; in chapter 3, the big picture of the algorithm can be read, starting from the single sensors exploited for the creation of the Deep Learning Neural Network model to the whole flow chart of the iOS application capable of classifying the nine different target values in real-time; in chapter 4, there is a summary of the experiments done following the algorithm described in chapter 3, that lead to the results described in chapter 5, where the accuracy of the machine learning model is studied, along with the accuracy of the iOS application and its performances on delay and power consumption minimization; in chapter 6, final discussion, opinions, and conclusions can be appreciated.

CHAPTER 2

PREVIOUS WORK

In both fields, we are going to study some previously conducted research works, and we are going to analyze the most important of them. Some of these works are listed only for a matter of knowledge about other efficient methods, while a limited part of them acts as a basis for the research proposed in this thesis.

2.1 Indoor/Outdoor Detection

Several studies on indoor/outdoor user's location detection have been done in past years, each of them with good results. Five of them will be presented, each one using different techniques and different sensors to achieve the same result, with different performances and power consumptions.

The first and most common method for indoor/outdoor detection consists in exploiting the GPS signal to detect user's location basing on signal intensity. In fact, GPS signal is usually feeble, or even unavailable, indoors, where the device itself is encircled by the walls, while it is usually more powerful outdoors where the satellites are more likely reachable. For this reason, the GPS signal's accuracy can be used to distinguish user's semantic location between indoors and outdoors [1]. The worst thing about this sensor is that it is one, or even the first, of the most power consumptive ones on the iPhone, and on any other smartphone. Researchers in [2] proved that the GPS sensor consumes almost twice battery power than other inertial sensors, plus its reliability as an indoor/outdoor detector is less robust than other methods. For instance, in fact, if the user is indoors but close to a "boundary", the GPS signal could be more precise than some outdoor places. In their experiments, those researchers found that the accuracy of the indoor/outdoor detection using the strength of the GPS signal could reach an average of 75%, also caring about the selected localization accuracy to find a tradeoff between this sensor's power consumption and its performance.

Another possible method ([3]) consists in generating an ultrasonic ping with the phone and, using its built-in speakers, periodically emitting it. At the same time, the phone should keep hearing using the microphone, to detect the echoes generated by the ultrasonic pings. This method is smart and efficient, and surely also a low power consumptive one, but, since our research is on iOS devices, it is even impossible to test it, since those devices do not present any possibility to generate or hear ultrasonic waves.

There is another very accurate and low power consumptive method, described in [4], that is based on GSM signal intensity. The basis of this approach is to use the Global System for Mobile (GSM) stations' signal strength to detect the user's semantic location. How? They use a machine learning model to classify the signal strength pattern coming from the nearest four GSM antennas, aiming at distinguishing between four different environments, called "indoors, outdoors, semi-outdoors and light indoors". By using this method, an accuracy near to 100% was reached on the training set. Like in the previous one, also this method is unfeasible to be developed on iOS devices since the access to GSM signal intensity values is forbidden by Apple.

In addition to this limitation, there is an overhead of work not needed in our research, since we need only a binary indoor/outdoor detection.

A fourth known method is the IODetector approach [5]: in this work, the researchers tried to achieve an indoor/outdoor detection (and also to classify specific transitionary states) with the lowest possible power consumption on the smartphone, by using sensors that have a small impact on the battery. These sensors are the magnetic field one, the ambient light intensity one, and the cellular signal strength one. Each sensor acts by itself, "to estimate a state and a confidence value". Then, every one of these confidence values is exploited with the aim of summing them up and find out the final state. However, the IODetector approach comprehends fixed thresholds to exceed, for all the sensors involved. These thresholds represent a strict limitation for this approach because it is impossible for the system to be sensitive to any changes in the surrounding environment, such as "weather conditions, seasons or latitude". Nowadays, these parameters are essential for the indoor/outdoor detection.

It is possible to overcome the limitations of IODetector, though, by using machine learning techniques, to go beyond the hard-coded and fixed thresholds, and by exploiting even more smartphone sensors. A first approach, using the proximity sensor, the microphone to detect surrounding noise amplitude, the time of day, the battery temperature, in addition to the abovementioned three sensors, has been described in [2], "robust IO detection by using semi-supervised learning with co-training". This method is the one on which our indoor/outdoor detection study will be based. The technique aims to build a network capable of distinguishing the indoor location from the outdoor one by training it using semi-supervised learning with co-training. This approach means that the features used in the algorithm are split into two different sets and balanced using the SVM ranking attribute, in order for each one of those sets to constitute a different classifier (the researchers used Naïve Bayes classifiers).

A fixed number of labeled instances coming from a single environment, plus a much higher number of unlabeled instances (because of semi-supervised learning) coming from two environments different from the first one, are used to train the network. A new instance is classified by both classifiers (co-training technique), and the value with the highest confidence is used to re-train the network.

By using this co-training method with an SVM ranking attribute to separate the different sets of features, an accuracy of 93% over the classification of instances has been reached in [2], with power consumption that is much lower than a solution using GPS, described in the first method.

In chapter 3, in the section dedicated to the indoor/outdoor detection, our methodology design, starting from the study done in [2], is described.

2.2 Activity Recognition

During last years, human activity recognition exploiting smartphone's sensors has become essential and vast in the Computer Science field. A paper exposing all the modalities, the sensors involved, the difficulties that can be encountered and the possible, available and usable machine learning models, can be found at [6], while a review of recent studies on activity recognition using mobile smartphone sensors can be found in [7]. We will take some hints from this "explanatory" document, trying to weave them together with research coming from other papers. We are indeed not only looking for a model capable of classifying the *simple activities* listed in that document (such as walking, jogging or staying) but also those listed as *complex activities* (riding a bus or driving a car, both contained in the automotive category). Like the previous section, some methods will now be presented, each one using different approaches and reaching different results.

Activity recognition regarding motion or location of the user is the primary study related to human activity recognition done using mobile smartphones. When activity recognition based on location is performed (developing systems that usually exploit cell towers, WLAN, Bluetooth beacons data in addition to GPS), the objective is to recognize specific activities that are directly related to the place in which they are recognized, like, for instance, the Reality Mining project [8], where the three fundamental recognized activities are "home", "work" and "elsewhere". However, these systems can only give imprecise information about the ongoing activity, in fact, for instance, when "work" is recognized it is impossible to know whether the user is talking on the phone, is in a meeting or whatever else. Most of the activity recognition studies, instead, are based on recognizing the ongoing motion activity performed by the user, mostly exploiting inertial sensors, as, for instance, wireless receivers such as cellular or WiFi [9; 10; 11] or different sensors [12; 13] such as the GPS. The most used inertial sensor in activity recognition is the accelerometer, which is the most accurate sensor that can be used for recognizing people and smartphone motion and the orientation of the phone (with the help of the gyroscope [14; 15]) derived from information about gravity. Also, the impact of using magnetic field sensor is investigated in some of the related studies [16; 17]. In [18] there is an exhaustive investigation of inertial sensors' usage in human activity recognition and their performance when used in combination with other sensors or even when used alone. There is a whole lot of recent survey articles reviewing the literature in exploiting inertial smartphone sensors for human activity recognition [6; 7; 19; 20]. We will focus our study and our methodology in gathering sensor data where orientation independence and gravity independence has been achieved.

For example, in the study done in [21], three sensors have been exploited: the accelerometer (not the linear one, since the tested devices had no gyroscope integration), the magnetic field sensor and the sound frequency sensor. Those sensors are all used to discriminate between four activity classes: "in car", "on foot", "tv watching" and "by bike". The classification, then, has been done using two different types of Support Vector Machine models, the first one with a linear approach, the second one with a non-linear approach, using a radial basis function kernel. The two approaches have reached an accuracy of 97% for the linear approach, and an accuracy of 99.5% for the non-linear approach, both considering the testing set.

A more complex study has been done in [22], where the researchers used a convolutional deep neural network to classify six different types of human activity using data coming from both the accelerometer and the gyroscope. Those data have been collected by thirty different individuals who performed various activities, keeping the device in a pocket. "Convolutional networks provide an efficient way to automatically and data-adaptively extract relevant and robust features." They were able to reach an overall accuracy of 94% on the test set, classifying activities like "walking", "walking upstairs", "walking downstairs", "laying", "sitting", "standing".

Since we also need to classify the automotive category, a good starting point for this study can be the research explained in [23], i.e., understand if the user is in a vehicle (train or bus, which is practically the same as the car) by checking the natural vibrations that the device can detect in a vehicle. They extract natural vibration features of any vehicle (such as bus, subway or car) to overcome the uncertainty that can arise since the activities, regarding user movements, are all static. They gathered data from the accelerometer, but they fixed those data using readings coming from the gyroscope. By using this sensor, in fact, it is possible to use the gravity detection to "achieve an orientation-free use of the accelerometer", like our work does by exploiting the Device Motion framework for reading sensor data on iOS systems. Moreover, they found a way to minimize the smartphone's battery consumption, with a specific algorithm. The results obtained showed that this approach could reach an accuracy better than any other one when classifying various activities including the detection of bus and subway trains. In their experiment, on the bus battery usage decreased by about 5% over 35 min, while for the other activities it decreased about 2% over 30 min. For what concerns performance, this algorithm reaches an average accuracy of more than 95% for subway, bus, and walking detection, using a Gaussian Mixture Model (GMM) classifier for classification.

2.3 Segmentation

Many research works on activity recognition use an approach called signal segmentation to increase the accuracy of classification and to deal with transitions between different activities. The explanation for this approach is simple: when dealing with data coming from accelerometer readings, or from other types of time-sensitive sensors, it is necessary to consider the variations of sensors' values over a time window, since those variations do define the behavior of the user in the specific time window. Approaching activity recognition with signal segmentation is used not only for plain classification of activity but also for recognizing transitions between different ones in a way that minimizes the accuracy loss during the transition itself.

The data coming from time-sensitive sensors are gathered during the window, and then some features are extracted (such as variance, mean or maximum and minimum) and used as instances in the machine learning model adopted for classification. There are a bunch of studies on how to approach to segmentation in different ways: we are going to analyze some of the relevant ones and to take some hints for our research work.

There are many studies on how to do activity recognition with a **dynamic** segmentation [24; 25; 26], and one of them is described next. The analyzed work is the one at [24], where they used a dynamic approach to signal segmentation in activity recognition using the accelerometer, reaching promising results both with non-overlapping time windows, where two different windows do not share any common samples, and overlapping ones. A dynamic approach to segmentation means that the time segments are not fixed, but they adapt each time. When a relevant data change happens while gathering, a new window is created to separate between the two different behaviors. As relevant data change, they intend a series of decreasing values for the readings, and the actual cut happens when the difference between the maximum and the minimum of this series exceeds a fixed threshold. The various activities were "standing", "walking", "on all fours", "sitting", "sitting on the ground" and "lying". With dynamic signal segmentation they obtained an accuracy of 97.5% when detecting only activities, and an accuracy of 92.9% when detecting transitions between them, too.

This particular approach can also be used for other types of activity recognition, which are not strictly related to our work but can be classified as relevant. For example, in [27], a method for home activities recognition in the Ambient Intelligence field, using smart-home sensors, is described. In this case, the time windows are dynamic too, and the separation between them is chosen by studying the sensor correlation and time correlation in gathered data. They obtained their best accuracy as 94.62% using a Bayesian Network for classification.

There is also a time-based approach for creating windows for segmentation, described as one of the various approaches in [28]. This method has some limitations, as described in the other cited studies but, if the proposed length of the window is appropriately studied, this approach can reach results with an accuracy that is comparable to other results with dynamic segmentation. The time-based approach is the one we are going to use in our research, both for indicators of the indoor/outdoor detection part and the activity recognition part.

CHAPTER 3

ALGORITHM DESIGN

In this chapter, the design of the whole project is explained. First, a machine learning model for classifying the different semantic locations and activities of the user is created and evaluated. After this first step, the model is integrated into an iOS application, with the aim of continuously classifying data in the real world, to evaluate the accuracy and the performance of the model and the application in their final version. All this work will then be used to develop a more complex application, capable of exploiting the semantic location and activity recognition to compute an estimation of the daily air pollution intake of the user.

In the beginning, the algorithm was based on creating a machine learning model for indoor/outdoor detection and then let the activity recognition be performed by Apple's Core Motion framework, which classifies the same activities that this project needs. However, after some tests on Core Motion efficiency for this case (confusion matrices can be found in chapter 4) and after some tests on this method's efficiency, the algorithm was modified. Now it includes the presence of a model that classifies all the nine different categories, whose accuracy and detection are reinforced for overcoming model's limitations by exploiting the Core Motion framework (further explanations in section 3.2 and chapter 4).

This result has been achieved because the various indicators for indoor/outdoor detection and activity recognition reinforce each other, too: as said in chapter 2, in fact, magnetic field values and information about noise can be exploited for both categories.

3.1 Model creation

The proposed algorithm aims at creating a classifier capable of distinguishing semantic location and activity of a user between nine different possibilities: {*automotive, indoor_biking, indoor_running, indoor_stationary, indoor_walking, outdoor_biking, outdoor_running, outdoor_stationary, outdoor_walking*}. The classification is done on raw data coming from smartphone sensors, combined to achieve the best results trying to minimize the battery consumption and to maximize the performance of the smartphone's implementation itself.

The process of model searching followed this scheme: "gather raw sensor data – try different models and find the one that can classify those data better – evaluate final accuracy". The various machine learning models are tested on RapidMiner. The output of the model is an array of confidence values that sum up to 1, one for each entry of the previous targets list, indicating which one is the activity that has been detected with the highest probability.

3.1.1 Indoor/Outdoor Detection

The research for this part is fundamentally based on the study done in [2], but it has to differ from it since iOS is a very closed system. In fact, to gather some raw sensor data different frameworks (IOKit, [29]), directly accessing hardware, are needed (like for cellular signal strength, light sensor or battery temperature) that are not approved by the Apple Developer Program. Their usage would indeed imply that the app would be rejected by the App Store. All the indicators of [2] will be taken into consideration, except, of course, for the battery temperature evaluation, that is forbidden to access in any way on iOS devices, if using public frameworks.

Fundamentally, this approach is based on some experimental observations about indoors and outdoors differences. So, for first, it is necessary to explain in detail which type of data will be gathered and why are those data essential.

3.1.1.1 Magnetic field

When many sources, like sockets or metal objects, are disturbing the magnetic field, the smartphone's sensor detects values that change very quickly, and this happens, of course, indoors (Figure 1) more than outdoors (Figure 2). As studied in [30], in fact, the magnetic field indoors is disturbed by the presence of significant steel members in the buildings, that "warp the geomagnetic field in a way that is spatially variable but temporally stable". For this reason, the actual variance of the magnetic field components, using a fixed segmentation approach over a time window, following the same pattern as [2], is taken, instead of single values that do not mean anything if looking for variations in the short-term. In fact, this sensor gathers samples five times faster than the other one, to compute the magnetic field variance over the time window frame and feed it into the model with the other sensors' samples, all gathered five times slower. As it is visible in Figure 5 of the study done in [2], those researchers also base their magnetic field study on the variance of the components.

On iOS devices, there are two different ways to get the magnetic field surrounding the device, both integrated into the Core Motion framework [31].

The first way to get magnetometer sensor data is to use the `CMMagnetometerData` property, whose value represents the whole "raw" magnetic field surrounding the iPhone. This value is

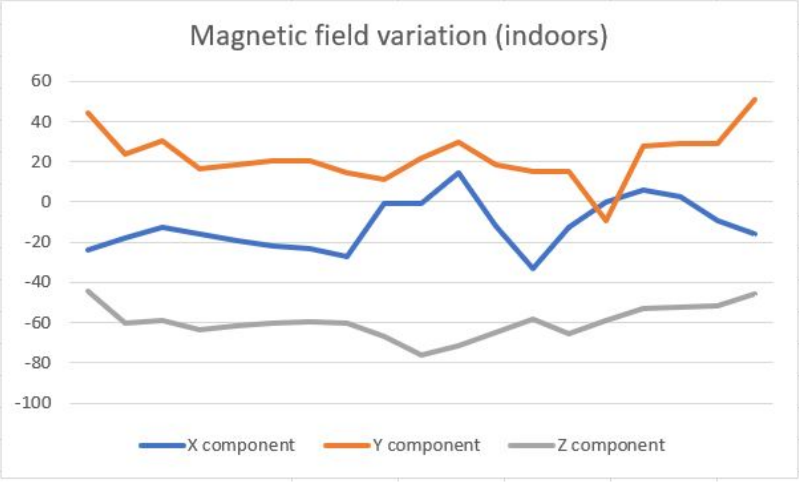


Figure 1: Example of variation of magnetic field indoors, within a window of 10 seconds

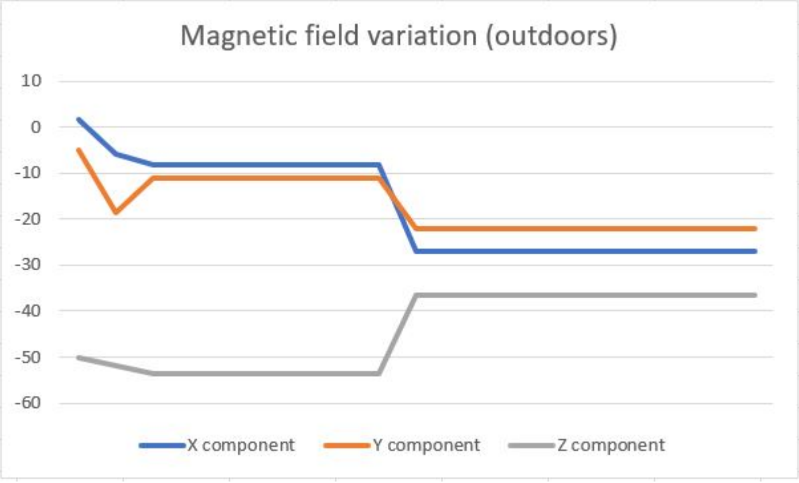


Figure 2: Example of variation of magnetic field outdoors, within a window of 10 seconds

in a ”raw” state because it consists of the Earth’s magnetic field plus the variation introduced from the smartphone itself and what surrounds it.

The second way is to use the `CMDeviceMotion` class, getting data from the property called `CMCalibratedMagneticField`, which contains the whole sensed magnetic field surrounding the device, without considering the variation introduced from the smartphone itself. The variation above is called bias, and it could negatively affect the magnetic field values measurements. In conclusion, the difference between the two methods is that the first one (the `magneticField` attribute of the `CMMagnetometer` class) is affected by the device’s bias, while the second one it is not.

This is the choice we include in our design since the ”Device Motion” service automatically processes and refines the raw sensor data coming from the available sensors. It uses a *sensor fusion algorithm*, which refines the raw data by measuring the device’s attitude, rotation rate, and the gravity metrics. According to the article at [32], ”sensor fusion is the art of combining multiple physical sensors to produce accurate *ground truth*, even though each sensor might be unreliable on its own”.

In that article, it is possible to read also that the most popular hardware implementing this algorithm is the Kalman filter. It is ”an application of the more general concepts of Markov Chains and Bayesian Inference, which are mathematical systems that iteratively refine their guesses using evidence”. The main idea of the algorithm mentioned above is to have an ensemble of ”belief” states for each sensor. During each iteration, these sensors’ readings are exploited to refine the guess, also evaluating the efficiency of each sensor.

It has not been cleared by Apple, and it is difficult to find anything online, but many developers [33; 34] say that iOS devices do not implement a Kalman filter, but rather a custom filter. This data refining is fundamental for our magnetometer data, since its samples are often very noisy, especially if there are motors nearby, as said in [32].

3.1.1.2 Ambient light

Usually, in the daytime and outdoors, light intensity in *lux* is very high, while indoors there are different values, which depend on the type of illumination. For this reason, recording current ambient light level is fundamental for creating a model capable of distinguishing indoors and outdoors positioning of the user. Light affects the evaluation of user position in different ways for these two cases: when the user is indoors, the light intensity is almost always the same, with a medium value on its interval scale. While outdoors, instead, the light intensity is strongly correlated to the time of day, since it is very high during the day and very low during the night. A more precise explanation about *lux* levels indoors, outdoors and during day and night is provided at [35].

For this reason, the **time of day** in minutes is also recorded, since we have to build a model capable of detecting the correlation between light intensity level, time of day and being outdoors. Otherwise, a wrong model could be built, a one that could detect outdoors positioning only with high levels of light intensity. A better explanation of this correlation between the three components (target value, ambient light and time of day), is given in Figure 3.

On iOS devices, it is impossible to access the ambient light sensor with public frameworks. Hence this indicator is not entirely based on the research done in [2], but an innovative

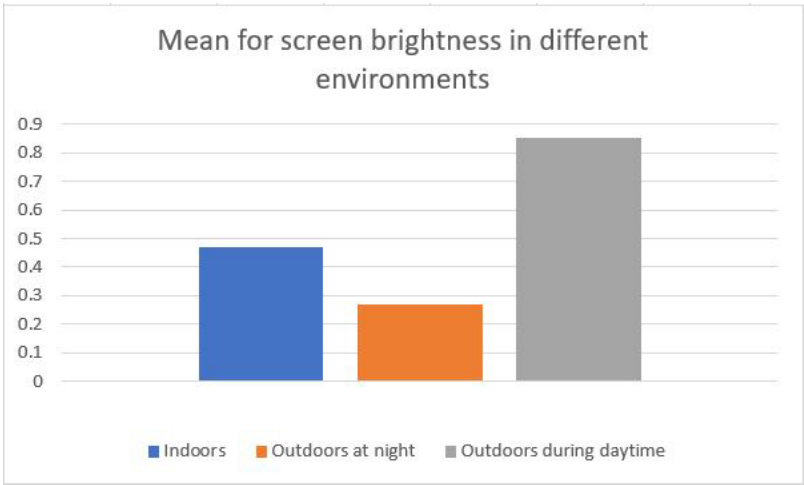


Figure 3: Mean value for screen brightness/ambient light in different environments and different times of day, over a 50 seconds time window

workaround has been found. In fact, since the automatic brightness of the screen is based on current level of light sensed by the ambient light sensor, the model is fed with the current level of brightness, on a scale between 0.12 and 1, in place of the ambient light intensity measured in *lux*. After having tested the correlation between user positioning with different levels of ambient light and the device's brightness, it is possible to assert that it reflects the light level appropriately, plus its recognition is very fast, almost immediate.

Another point supporting this thesis is that ambient light sensor on iOS devices cannot distinguish between direct sunlight and light artificially generated, since it senses only the current level of *lux* in the environment. For this reason, basing the detection on brightness, that in turn is based on ambient light sensing, can result in an accurate method of light detection.

Of course, there are always cases in which detection can be affected by external factors, like for instance when the current weather makes the device sense a low level of light during the day, but other sensed parameters are used to overcome these limitations.

3.1.1.3 Cellular signal strength

Typically, when the user's semantic location switches from outdoors to indoors, the cellular signal strength suddenly decreases because of the signal's weakening caused by the walls (Figure 4). This theory is taken from the study in [2], as the other ones, but its validity is also reinforced by the indoor/outdoor detection method cited in [4], which, as mentioned in chapter 2, classifies user location between four different indoor/outdoor environments just using the strength of GSM signal from nearby stations.

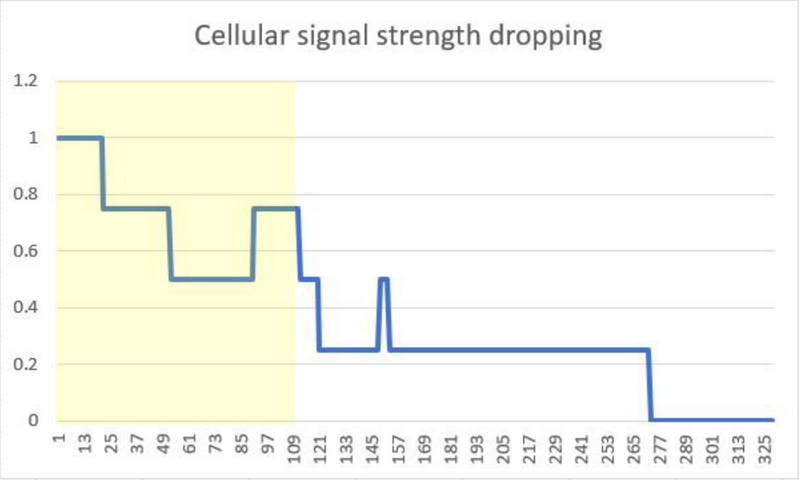


Figure 4: Signal strength dropping when switching from outdoors (yellow blurred area) to indoors, within a 160 seconds window

For this reason, among other parameters, the strength of the cellular signal is also used. Since, as in the ambient light sensor case, the cellular signal sensor cannot be accessed on iOS devices with public frameworks, an alternative and innovative workaround has to be used. In fact, while it is impossible to get the proper value of strength in dBm, the actual signal strength is read from the status bar of the device, by counting the current number of signal bars. This problem represents the strictest limitation given by iOS systems closeness.

3.1.1.4 Noise amplitude

Another useful parameter to study, to distinguish user's position between indoors and outdoors, is the amplitude of surrounding noise, basing on [2]. In fact, in a closed environment like indoors, the noise level should always be less than outdoors, where there is traffic generated noise.

As studied in [36], noise levels indoors change as the contact with the outdoors environment increases. In fact, their experiments were accomplished with the help of more than one hundred people living in houses exposed to traffic noise. Data about noise amplitude were gathered in three different situations (i.e., with windows opened, closed, and semi-opened), together with information about characteristics like orientation or sound insulation. "The median indoor-outdoor sound level differences were of 10 dB(A) for open, 16 dB(A) for tilted, and 28 dB(A) for closed windows."

In conclusion, noise amplitude detection is a critical parameter to distinguish indoor from outdoor environments. On iOS, this is done with the AudioKit framework ([37]) by generating a *silence* wave (with an amplitude in the order of 10^{-4}) to start the microphone detection, and then taking current amplitude of noise with the same interval as other sensed parameters. In Figure 5 it is possible to see the difference between the noise in indoor environments and the noise in outdoor ones.

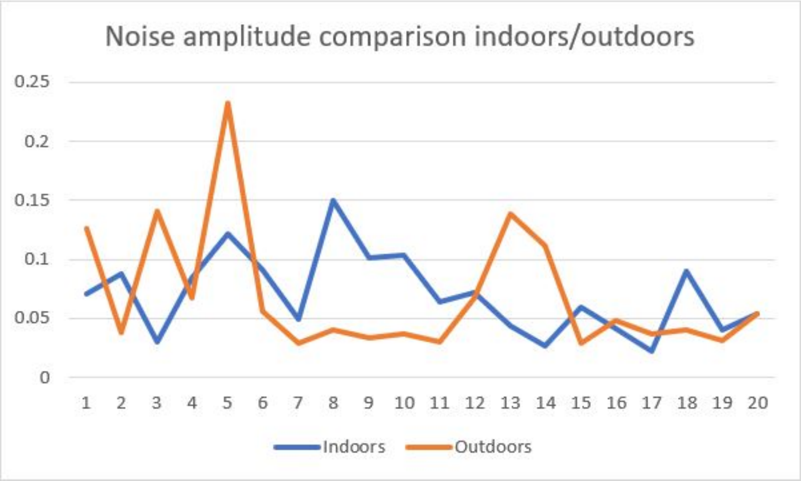


Figure 5: Noise amplitude comparison between indoors and outdoors environments over a 10 seconds time window

As it is visible in Figure 5, usually the peaks of noise amplitude are much higher outdoors than indoors, due to traffic intensity and environmental sounds. Of course, there are cases in which the amplitude outdoors could also be unusually low, depending on in what kind of place the user is.

3.1.2 Activity Recognition

Activity recognition using smartphone sensors is a field that has been extensively studied in past years, as said in chapter 2, much more intensively than indoor/outdoor detection algorithms. All smartphone's operating systems now integrate frameworks that can detect which is the current activity of the user. The iPhone has, since the release of iPhone 5 in September 2013, a coprocessor (the first one was the M7 coprocessor), also called motion processor, which "collects, processes, and stores sensor data even if the device is asleep" [38].

The `CMMotionActivity` framework [39], in fact, can detect if the user is currently in an automotive, stationary, walking, running or cycling activity condition. As said at the beginning of this chapter, these are the five activities we need to classify with the model we are building, and since the results are promising, this framework is used only to reinforce the detection of the machine learning model.

The sensors involved are the ones described at [6], i.e., all the common ones that are used for human activity recognition, with the difference that we fix the accelerometer data with the gyroscope to achieve an orientation-free human activity recognition, as used in [23]. In this case, too, the sensors that are exploited in the research are listed in next sections, with their meaning and their mutual correlation.

Studies like the one at [21] also use the magnetic field sensor and sound frequency detection to achieve a better classification for cycling and automotive categories, and we already collect them for the indoor/outdoor detection. For this reason, these features reinforce each other for classification in both categories.

3.1.2.1 Accelerometer fixed by the gyroscope

The first and most important parameter to take into account when studying activity recognition is, of course, the accelerometer, as said in chapter 2.

It is used for measuring the proper acceleration, which is defined as acceleration relative to the non-accelerating reference point. There are a variety of usage areas such as medical applications, inertial navigation systems, structural monitoring, transportation systems and consumer electronics, since they are capable of detecting tilt, vibration, and orientation along with acceleration. In the domain of electronic systems, accelerometer technology is widely used in smartphones. It enables the device to sense the surrounding environment.

The original purpose of the implementation of the accelerometer in smartphones was to improve the user experience. In fact, their readings were initially used only for detecting if the user had her smartphone in portrait or landscape orientation, and the system automatically changed display's orientation. However, later accelerometers were used to study activity recognition and user's motion. On iOS devices, the accelerometer values are made of three components, representing the three different spatial axes (x, y, and z, Figure 6). Data coming from the accelerometer is also affected by the acceleration of gravity.

The gyroscope, instead, is a sensor that detects the orientation of a device, by exploiting the rules of the angular momentum. It is a mechanic disc that responds to an external force following the rules of the moment of inertia. In the beginning, gyroscopes were used as devices to support compasses in navigation, but nowadays they are implemented in electronic devices, to detect the orientation and the rotation of those devices. This implementation of the gyroscope

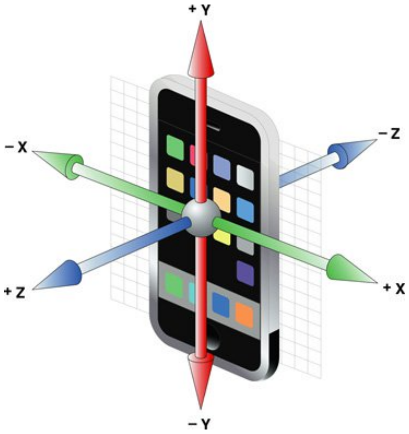


Figure 6: The three axes of an iPhone accelerometer

suddenly improved the detection of movement, combined with the accelerometer. On iOS platforms, the gyroscope readings are constituted by three rotational components following the rule of thumb, and they are evaluated in radians per second.

Figure 7: The gyroscope implementation in an iPhone

On iOS devices, we need to exploit the Core Motion framework to access the accelerometer and the gyroscope, like in the magnetometer sensor case. The primary class of this framework is the `CMMotionManager` one, in which there are all the methods accessing the various sensors, via their APIs. However, like in the magnetometer sensor case, this class provides only raw and undefined data, also for the accelerometer and the gyroscope. Thus, `CMDeviceMotion` is used again because, after some experiments with the raw and undefined values, we noticed that, even if the device is placed still on a table, the vertical component of the accelerometer reading is different from zero because of the acceleration of gravity g. The accelerometer is not the only sensor affected by external factors, in fact the gyroscope readings show that the iPhone has a rotational speed, even if motionless.

For explaining this, a little digression is needed. However, for more detailed reasons and studies, it is possible to read the article at [40]. All the test on the accelerometer readings showed that they were damaged by an excessive amount of noise, due to the action of the acceleration of gravity.

After several tests and studies, the iOS software development kit developers were able to divide the acceleration of the device from the acceleration of gravity, by using a low-pass filter for the first one, and a high-pass filter for the second one. However, when reading sensors' data, the presence of many filters erroneously affects the detection and its accuracy.

While the most critical problem of the accelerometer was gravity, the crucial reason for accuracy loss of the gyroscope was the "shift of sensor readings". In fact, the data coming from the gyroscope is never equal to zero, even if the iPhone is motionless and placing still. The situation is worse when the speed is used for detecting the angular position, because the error starts growing linearly.

After all these studies, the developers came to the conclusions that the readings originated by the accelerometer and the gyroscope need to be joined, to reach the best accuracy and remove the issues. These reasons brought to the development of the `CMDeviceMotion` framework, which keeps readings for the accelerometer and the gravity acceleration divided one from the other, plus returns gyroscope readings free from the abovementioned error, in order to appropriately determine iPhone's orientation.

Using this framework helps gathering the acceleration fixed with the usage of the gyroscope, as it is described in [23], to obtain a reading that it is not affected by device's rotation. In fact, as described in chapter 2, they managed "to achieve an orientation-free use of the sensor", by using the gyroscope to clean accelerometer data from the influence of the acceleration of gravity. In fact, with their method, the accelerometer data returned a value of zero from each of the three spatial axes, if left still and untouched. In this way, the variations on the vertical axis indicate that the device is moving "up-and-down". Thus, it is vibrating because of the vehicle. If orientation-independence is not addressed, users are required to carry the phone in a specific orientation, and this limits the practical usage of activity recognition.

Position-independence, i.e., where the phone is carried, is also another vital factor in activity recognition process. Every user brings her smartphone in a different location from any other one (like in a pocket, in hand or in a bag/purse), and due to the differences in signals coming from the sensors at different positions, it is a challenge to detect the user activities. Although, a few of the works jointly analyze the orientation and position dependency [41; 42] on the activity recognition accuracy, either they analyze the performance with only one method (using orientation-independent features, usually the magnitude or signal transformation), or they do not focus on user-independency and use cross-validation tests [41], or do not focus on how to estimate where the phone is carried.

Finally, the same approach to segmentation is used for the accelerometer values; in fact, they are taken in the same fixed time window already mentioned for the magnetic field sensor. This fixed segmentation approach is not borrowed from any other research work; it is a novel method that expresses its efficiency in combination with all the other expedients found to maximize performances, in the real-time classification done by the iOS application.

3.1.2.2 User's speed

For better distinguishing between the various considered activities, user's speed using GPS is also taken into account, since it can be, in concurrence with the cadence recognition (see next paragraph), a considerable discriminator when trying to classify the different activities pattern. The usage of speed is taken in part from the study at [23], where they use the location of the user to also distinguish between bus and train. However, this distinction is not needed in this research work.

So far, this study has focused on acceleration data for activity recognition. Therefore, we collect GPS data along with acceleration data to acquire speed information. The efficiency of the GPS signal changes depending on its surroundings. If this signal is not accurately received, speed data could be incorrect. Nonetheless, since the speed is not an essential value to classify activities, this is not a problem worthy of mention. The evidence of the utility of recognizing speed when doing this type of pattern classification is glaring and can be seen in Figure 8, where there is a clear difference of speed between human activities done on foot and speed when being in a vehicle. The only problem to correct is when being in a stopped vehicle, and it is fixed in the iOS application by reinforcing the detection with Core Motion framework (see next section).

When the vehicle (such as a car, a bus or a train) stops at a semaphore or a station, or when the GPS conditions are too weak to detect the signal accurately, the *automotive* state recognition could be affected and generate wrong results or noisy ones. This limitation can be overcome with the iOS application, which makes the final decision reinforcing its detection with Apple's Core Motion framework.

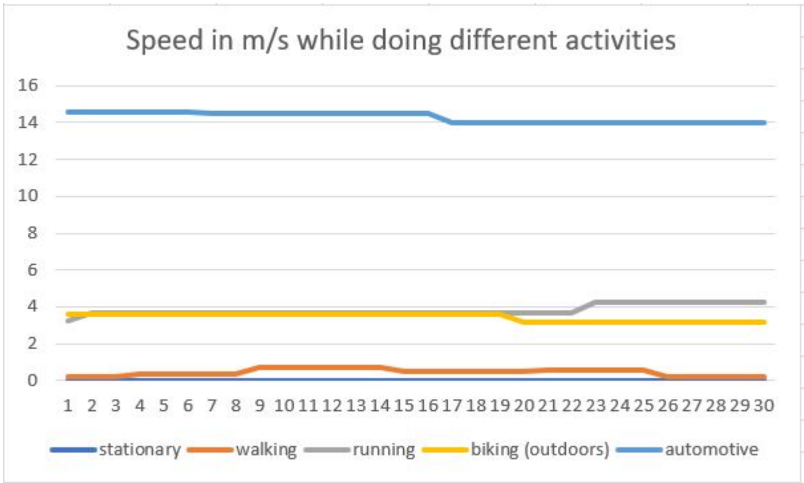


Figure 8: User's speed in m/s while performing different activities in a 6 seconds window

3.1.2.3 User's cadence

Many researchers base their activity recognition studies only on the accelerometer, or on this sensor joined with the gyroscope. For a more elaborate activity recognition application the help of embedded pedometer and GPS signal is needed and, with them, it is less complicated to detect if the user is in a moving or stationary state. When more complex activities are recognized, all the differences between various inertial sensors could help the detection because of the detailed discrepancies in their readings [43].

In all iOS devices, it is possible to find a pedometer, i.e., a sensor that counts the user's number of steps during the whole day, and it is called `CMPedometer` [44]. It has some mention-worthy functions, one of which consists in measuring the rate at which steps are taken, measured in steps per second. The novel idea (not found in any other research work) is to take this measurement (called *cadence*) for semantic location and activity recognition, to distinguish some of the activities the user can do on foot. In fact, two activities like walking and running can be easily distinguished by looking at how many steps the user is taking per second, associating it with the currently measured speed of the user; it can overcome the limitations of the accelerometer measuring a null acceleration when running or walking at the same speed. When considering automotive state, otherwise, the cadence will always be null, and for this reason, the speed itself can help recognizing the pattern of being in a vehicle and not doing anything else.

For the biking activity, the cadence measurement has to be studied more carefully, since some movements while biking can seem the same as when walking or running, depending on the frequency of the pedaling. For this reason, we collected data while biking either with null cadence and with greater than zero cadence, for the model to be adaptive to those different situations. The different cadence recognitions for the user biking can be seen in Figure 9.

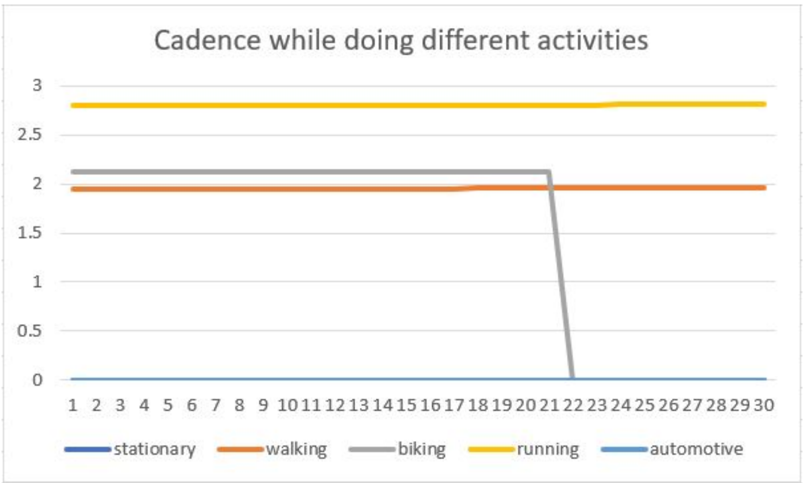


Figure 9: User's cadence, measured in steps per second, while performing different activities (automotive, stationary and final part of biking are overlapped) in a 6 seconds window

In conclusion, all these sensors form together a robust method for classifying semantic location and activity of an iPhone user, since each feature is strictly connected to each other, or compensates the issues that are found with another one. More information about construction and performance of the machine learning model, created using the described indicators, can be found in chapters 4 and 5.

3.2 iOS Application Development

All the indicators listed before are used to create a model that fits into an iOS application, using Core ML library. Thanks to it, a developer can create a model based on standard services and integrate it into the application to classify data: in this case, raw sensor data. More information about Core ML library at [45] and more information about the model conversion in section 4.2. The only issue using Core ML consists in having a fixed model in the iOS application, meaning that it cannot evolve and cannot learn user's habits, but an update is always needed to change it.

The toughest challenge in this research study is trying to minimize the power consumption of the whole application, thus by minimizing the usage of the GPS, which is the most power consumptive sensor in smartphones' hardware. For this reason, the iOS application tries to minimize its usage by switching it on only when detecting a transition from an indoor environment to an outdoor one. In fact, the GPS usage for detecting speed is needed only outdoors, where the user can then get in an *automotive* state or a *biking/running* state (more likely than indoors) and then speed detection would positively affect the activity recognition.

After various tests on the accuracy of recognition using different frequencies for gathering data (starting from high values like 10 Hz), and progressively lowering it down to check the loss of accuracy, the best tradeoff between accuracy and frequency of data gathering has been found as 1 Hz. This tradeoff needed to be studied because if the frequency of gathering data from sensors is low, resources consumption on the iPhone is also low, particularly the battery consumption, which is the most critical limitation. The sensors asked more frequently are the magnetic field sensor and the accelerometer using Device Motion; they are read with a frequency of 5 Hz, to have five samples in a second and compute the variance of the magnetic field and the acceleration, using the fixed segmentation approach described in previous sections. After having computed the variance, the application feeds those data, plus all data coming from other sensors, to the Core ML model stored in it, with the aim of recognizing the ongoing activity of the user each second. The application then uses all the activities recognized at each second to compute the total time doing each activity throughout the day itself.

This application will be integrated into a more complex one, that will be capable of exploiting the semantic location and activity recognition to compute an estimation of the daily air pollution intake of the user. For this reason, it follows an algorithm for minimizing the classification error, the delay for recognizing the switch to another activity and the battery consumption by avoiding using the GPS sensor all the time. A complete picture of the algorithm can be read in the flow chart on the last page of this chapter (Figure 10), with its detailed explanation preceding it. This algorithm is used, among the other reasons, to overcome the limitations presented by the fixed approach to segmentation, and achieve a real-time classification, transitions between activities included, with the highest possible accuracy.

After having fed the sensors samples to the machine learning model, the output is an array of confidence values, each one indicating a probability of that particular activity happening, and they are alphabetically ordered.

If the maximum confidence value among the nine output values is **higher than 0.8**, and the recognized target is the same as previous classifications, 1 second of doing that target is stored.

If, otherwise, the activity recognized remains the same but the semantic location changes, there has been an indoor/outdoor switch or vice versa, and this means that the new output target can be stored. If the I/O switch has been from an indoor environment to an outdoor one, the GPS is turned on, and user's speed is started to be gathered with other sensor data (indoors, without using the speed feature, the performance of the model remains the same). If there is not any I/O switch but only a modification of the current target output, the application checks the value of the ongoing activity recognized by the Core Motion framework from Apple. If the new target output's activity coincides with the one detected by it, 1 second of the new activity is stored. Otherwise, if Core Motion still recognizes the previous activity, 1 second of the previous activity is stored.

If otherwise, the maximum confidence value is **less than 0.8**, more checks need to be done to be sure to minimize the classification error. In this case, the application looks at the previous five detections that have been done by the machine learning model, and if it notices that the current detected maximum's confidence has a monotone decreasing behavior during the window, and the second highest confidence is increasing over the time window, it means that an activity switch is going on. In this case, the application needs to detect and store all target outputs until the final switch, without actually registering the most probable output. When the activity switch happens (i.e., the second highest confidence value becomes the maximum one) the switching time window (all the seconds left blank) is stored in this way: $\frac{1}{3}$ of it as the previous activity, while $\frac{2}{3}$ of it as the new ongoing activity.

If, when checking, there is no activity switch behavior during last five detections, the application looks again at the current activity recognized by Core Motion framework, acting in the same way as described before.

At the end of the 24 hours, i.e., midnight, the application uses all the stored classifications from the day to compute the actual time spent doing each one of the activities.

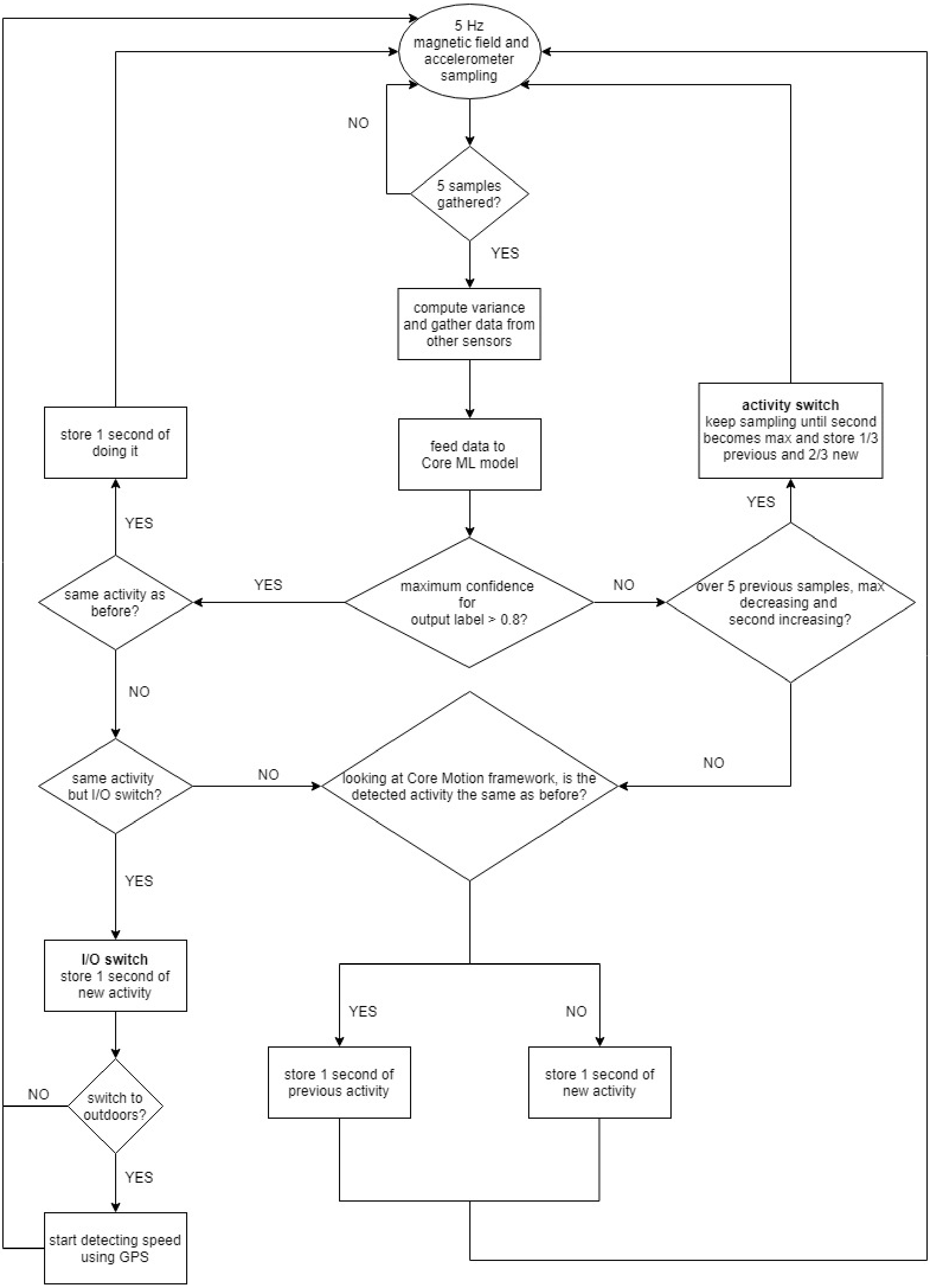


Figure 10: The flow chart of the iOS application algorithm

CHAPTER 4

EXPERIMENTAL METHODOLOGY

This chapter explains what experiments have been made, how were they conducted and what were the issues found. The experiments regarded both the study of the machine learning model, capable of offline classifying the different patterns characterizing the activities we want to target, and the final iOS application implementing it, capable of classifying activities and transitions between them in real-time.

4.1 Machine Learning model

The creation of the machine learning model followed three steps, as usual: data collection, with data analysis and data pre-processing, followed by a phase of training the model (different models have been tested) and a testing phase on an unlabeled dataset. This last part can be found in chapter 5, while in this chapter is possible to find a description of the various steps made during the experiments for this research.

4.1.1 Data collection

In order to respect the limitations given by the Apple Developer Program, different choices have been made for data gathering, as described in chapter 3. The battery temperature evaluation, which is one of the device's features gathered by the study in [2], is forbidden to access in any way on iOS devices, and therefore it has not been exploited. The specific combination of indicators needed for this type of research prevented us from using an already gathered dataset to train the specific model required for building the application, because indicators like screen brightness or the number of bars indicating cellular signal strength were never used together in any other type of research. Nevertheless, a dataset containing ambient light intensity in lux and cellular signal strength in dB, with the nine different labels needed for this type of work, was also impossible to find anywhere. For this reason, after weeks of seeking for this type of dataset, data have been gathered with an iOS application created exclusively for this purpose.

The abovementioned application ran on an iPhone 6 with iOS 11.2.6 installed on it. The iOS version means that it is almost impossible for the developer to access private frameworks, to try and collect data from forbidden sensors. The iPhone model, instead, means that the motion processor coming with it is the M8 coprocessor (more powerful and accurate than the first M7 coprocessor but less than M9, M10, and M11), released in September 2014.

However, the proposed approach has been validated on various public datasets, gathered for the single modules (further description in section 4.2).

The app gathered the magnetic field and accelerometer data with a frequency of 5 Hz and other sensors at 1 Hz, and it had a very simple UI (Figure 11). Why a 5 Hz sampling? Because after some experiments it has been found that the minimum acceptable variance that can be computed for both sensors needed at least five samples. The application recorded sensor data and then wrote them in a temporary CSV file, which could be exported anywhere at any time thanks to a simple button. The CSV file also contained the label for each row of gathered data, since, while recording information, there was the possibility to tell the application, with some buttons, what the user's ongoing activity was. Another parameter registered while recording

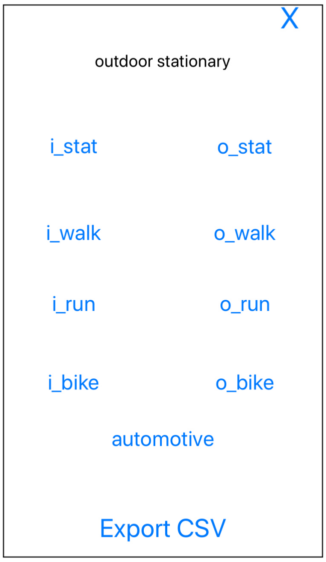


Figure 11: Simple UI for data gathering application, with buttons to change the current label that is stored in the CSV file

data from sensors was the Core Motion activity detected by the device. The purpose was to evaluate, then, its accuracy compared to our model, and to study how to improve our evaluations using the Apple motion framework. The CSV files with all sensor data created by the application have then been used for creating a machine learning model.

In an earlier stage, data have been gathered on UIC East Campus and in the surroundings (Automatic Lofts, Target, UIC Pavilion), both indoors and outdoors, during day and night, and at the Chicago Auto Show (McCormick Place) during the night. In the second phase, data have been gathered in various California cities and different environments (both indoors and outdoors, during day and night): San Francisco and Alcatraz, Los Angeles, San Diego. In a third phase, data have been collected in the Detroit area. Samples of all the possible activities and all the possible positions (in hand, in a pocket, in a fitness armband) of the smartphone have been collected, reflecting the habits of a typical user during a single day. This means, for instance, that the walking and stationary samples are more copious than the running or biking ones, while automotive is a stand-alone category (for which samples in cars, buses and trains have been gathered). An overall number of 65000 samples has been collected, equally distributed between the two indoor and outdoor environments. In this case of research work, it is more important to have a diversified dataset (for it to adapt to all situations) rather than a huge but plain one. Data distribution can be seen in Figure 12 and, as previously explained, it reflects the ordinary user daily habits, while keeping a similar division for indoor and outdoor samples, overall.

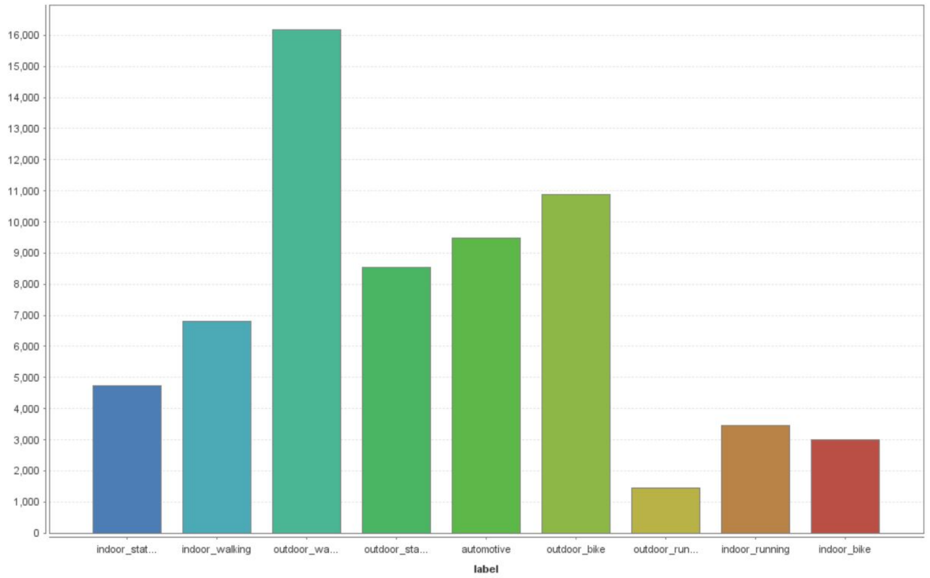


Figure 12: Count of samples per each target category in the dataset

After that, some data preprocessing has been done on the initial dataset regarding semantic location recognition: for what concerns magnetic field, we already said that refined and processed data have been taken from the magnetometer sensor, instead of raw and unprocessed ones; ambient light intensity (screen brightness) was already on a scale between 0.12 and 1, as said before; cellular signal strength has been normalized in the [0, 1] closed interval; noise amplitude was already a processed datum; time, in conclusion, has been normalized over the total number of minutes during the day (1440), in order to distinguish faster between daytime and nighttime. For activity recognition, instead, there was no preprocessing, since the acceleration data have been taken using the `CMDeviceMotion` framework, as it has been done for the magnetic field sensor, and speed and cadence were already in the unity measures we need.

In chapter 3 it is possible to see the correlation between the various output targets and the values of gathered data, with charts created starting from the CSV dataset. However, a couple further and more complex considerations about the gathered dataset can be done.

In Figure 13 there is a series chart showing how the acceleration in x, y and z axes changes along the whole database. The plain zones where no one of the three components changes are the ”stationary” activity samples, while the most common ones are the samples gathered during the ”walking” activity and the most various ones are the samples for the ”running” activity. The amplitude of the domain for the acceleration magnitude changes following the activity that is done by the user: more physically hard the activity, wider is the interval of detected acceleration. This is the reason for approaching accelerometer readings with segmentation.

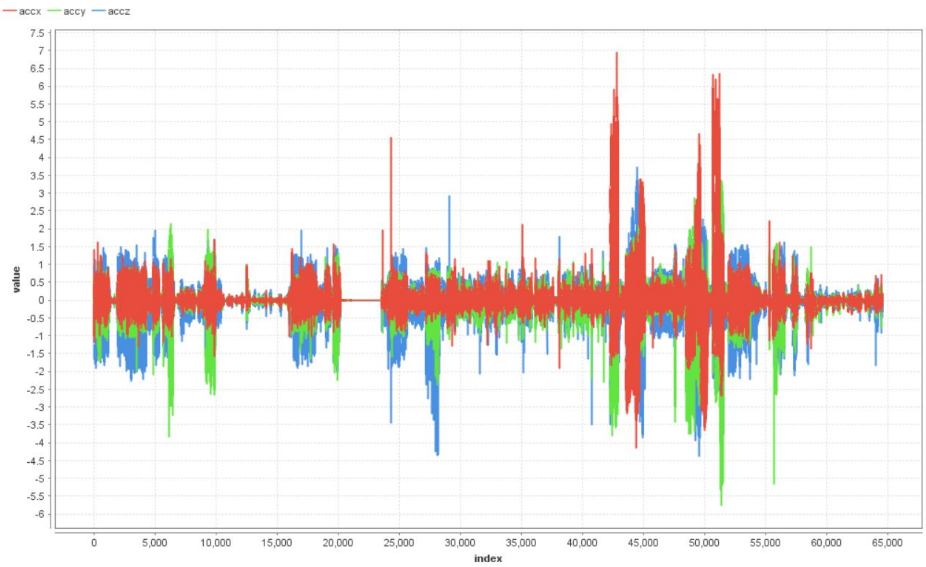


Figure 13: Acceleration variation in the whole dataset

Another more focused analysis on data should be a deepening on the correlation between the amplitude of noise, time of day, and target label. A visual explanation can be found in Figure 14. In that figure, it is possible to see the discretized time of day on the horizontal axis, and the amplitude of noise on the vertical axis, while the dots' color refers to the various output labels. As it is visible in the figure, almost all samples with an amplitude greater than 0.275 are gathered outdoors, at any time of the day. The interesting fact is that the average amplitude of noise decreases during nighttime, making this parameter less useful for classification at night.

The last complex correlation we should talk about is the one between light level, time of day and output label. On the horizontal axis of Figure 15 there is the time of day during which the

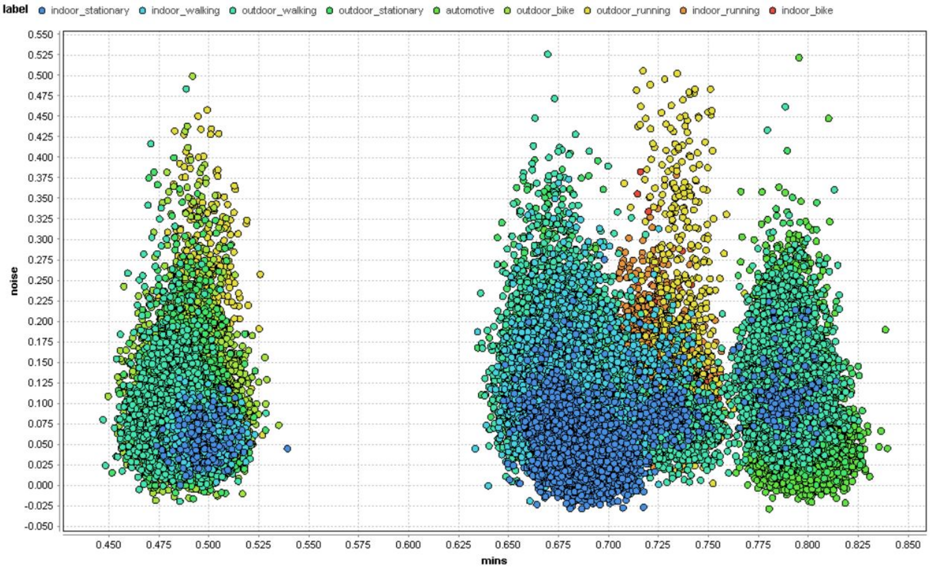


Figure 14: Correlation between level of noise amplitude detected, time of day and target label

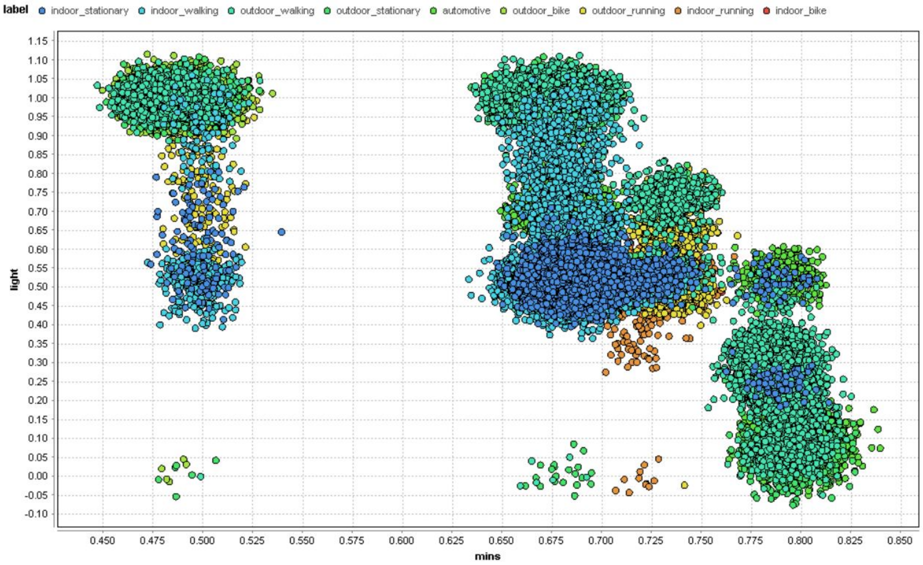


Figure 15: Scatter plot of all the samples of the dataset, comparing the time of day, light level, and target level

samples have been gathered and, on the vertical axis, it is possible to observe the detected level of light, while the color of each dot represents the target label of that sample. It is possible to appreciate that most of the samples with a light level above 0.65 have been gathered outdoors before the value 0.75 for time of day on a [0, 1] interval, where 0 is midnight, and 1 represents 11.59pm. Therefore, 0.75 represents 6pm during the day, i.e., the time of sunset when data were gathered. All the samples with a light intensity lower than 0.4 have been gathered outdoors at nighttime, following the thesis explained in chapter 3. In conclusion, all indoor samples have a light value between 0.4 and 0.65, independently on the actual time of day.

As it is visible in Figure 16, related exclusively to this correlation, there are some **outliers** in the dataset. Taking the definition in [46], "an outlier is a data point that deviates too much from the rest of dataset". It may be due to variability in the measurement, or it may indicate experimental error; the latter are sometimes excluded from the dataset. In our dataset there are some outliers, as said before, stored for the light attribute, because the ambient light level is derived from the screen brightness and sometimes there are sudden changes, that almost every time are performed by the user. In Figure 16 it is possible to see the outliers for the light attribute, related to the time and the label with which they have been stored.

There are several methods to detect outliers in a dataset, and we used the distance-based detection method, which works very well with multi-dimensional analysis. This type of outlier search has been performed according to the detection approach recommended in [47]. In their paper, they propose a distance-based approach, measuring the distance of each point from its k_{th} closest neighbor. The points are sorted in decreasing order, according to their distance to

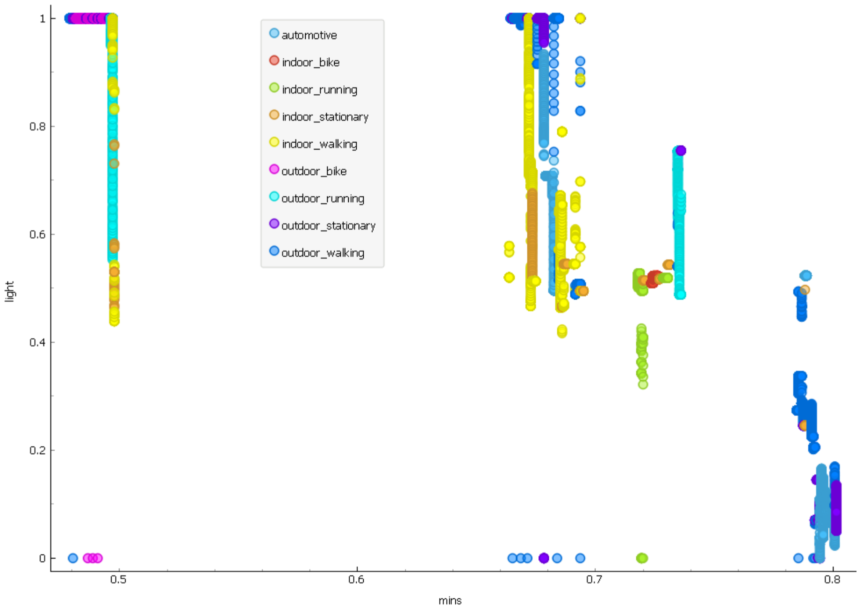


Figure 16: Outliers for the light attribute, abnormal values differing from others with the same label and a similar time of day

the k_{th} closest neighbor, and the first n points are the *outliers* of the dataset, according to that attribute. Therefore, k is the "*number of neighbors*", set as **4**, while n is the "*number of outliers*", set as **12**. This approach is basically founded on the definition of distance-based outliers given by Knorr and Ng: "a point p in a dataset is an outlier with respect to parameters k and d if no more than k points in the dataset are at a distance of d or less from p". The outliers have been removed before training the model, but, since they are rare if compared to the number of gathered samples, they are not considered in the real-time classification.

After all data analysis and outlier detection and removal have been performed, the RapidMiner tool was used to perform some sensitivity analysis, i.e. to find the best parameters and to study the correlation between the independent variables and the dependent variable, and to study various machine learning models with the aim of finding the best one. All the training and testing performances can be appreciated then in chapter 5.

4.1.2 Sensitivity analysis

Several machine learning models have been tested for pattern recognition on this dataset, such as Naïve Bayes, Random Forest, Gradient Boosted Trees, Decision Tree and Deep Learning. The first ones had abysmal performances on the dataset, compared to the last two. The only ones that have been considered are the Decision Tree and the Deep Learning, which is the one that has been finally chosen, for its properties described in section 4.1.3.

The decision tree has been created mostly for performing some sensitivity analysis, i.e. to study how the various attributes impact the prediction under a given set of assumptions, with the aim of removing the ones that do not contribute to the detection, before training the deep learning model. This type of study can be conducted with a decision tree because of its interpretability, since the importance of every attribute is clearly visible, due to how the tree is constructed. The deep learning neural network, instead, is non-easily understandable, since it is a "black box", where it is difficult to interpret how the decisions were made during training.

Sensitivity analysis also comprehends the calibration of the proper values to obtain the best possible result when training a model, and we performed it both for the decision tree and for the deep learning, obviously.

Why is the decision tree more interpretable? It is a machine learning model which classifies (by returning an output value, the "*decision*") an array of inputs constituted by some attributes.

Each branch of the tree represents a split on a specific attribute A_i, according to its value $A_i = v_{ik}$. A decision tree can be directly inducted by examples, and this is the way our decision tree was built.

There are several methods for deciding which attributes will be selected for the upcoming split, and the most used is the **information gain**, with whom the split is performed on the attribute with the lowest entropy, which is the "measure of the uncertainty of a random variable". Of course, the reduction in entropy is the aim of this model, and this can happen only by acquiring information. Therefore, the "expected reduction in entropy" is computed as test before the split. The correlation between different attribute values and between attributes and outputs is clearly visible, when looking at the tree and at its splits. Therefore, this model (correctly calibrated) can be used to study the attributes themselves.

The studied decision tree used **gain ratio** as splitting method, which is an improvement of the information gain approach. In fact, with this method the information gain is accomodated to allow the equality of each attribute value. In Table I it is possible to see that the best performance of this model was found with the depth of 40. However, a fully grown and large tree leads to a high variance and a too high classification error. From the cited table, it is possible to see that the depth of 25 is the first optimal value for performance (only 0.8% less than the depth of 40, but leading to a much smaller tree). In conclusion, the interval [25, 40] is the best one for considering the depth of the tree, and of course the best choice is to select the lowest value.

TABLE I: OVERALL PERFORMANCE VALUE OF CORRECT CLASSIFICATIONS ON TRAINING SET, COMPARING DIFFERENT MAXIMUM DEPTHS OF THE TREE

Maximum depth	**Performance**
2	0.294
3	0.457
5	0.639
7	0.760
10	0.843
15	0.859
25	0.964
40	0.972

When dealing with the gain ratio, there is a *minimal* value of this gain that has to be studied and chosen, for properly adapting the tree to the features of each attribute. The higher the value, in fact, the fewer the splits. Of course, if this value was chosen too high, the tree would have had only a single node and no splitting. The choice for the developed tree was a value of **0.05**, that is the value giving the best performance in the interval found to be optimal for this kind of dataset: [0.02, 0.09].

When training a decision tree model, **pruning** is used to ”fight” the overfitting phenomenon, which is very negative for machine learning (accurately described in next section). Applying pruning means that, if the test appears to be irrelevant with only noisy data, the sub-tree is eliminated and substituted with a single leaf (only tests with leaf descendants can be considered). A test is irrelevant when the information gain is close to 0. In fact, if data deviate from a perfect absence of pattern by a mean of 5% probability, this is good evidence of the presence of a significant pattern in the data, and there is no pruning. The deviation is computed according to the χ^2 **distribution**, which a random variable X has when it can be rewritten as a sum of squares, like in Equation 4.1. The k different "Y" are random variables, mutually independent one from the other.

$$X = Y_1^2 + ... + Y_k^2 \tag{4.1}$$

In conclusion, pruning with a confidence value of 0.25 (best value in the interval [0.18, 0.40] found to be optimal), which specifies the confidence level used for the pessimistic calculation of pruning, has been used in our model.

All decision tree and decision tree pruning definitions have been taken from the Artificial Intelligence (S. Russell, P. Norvig) book at [48], while all parameters tuning has been done following the trial and error method.

The performances were good, in fact we obtained an accuracy of **95.60%** on the cross-validation, but the best choice was still the deep learning, not only for its better performance. The reason was that the decision tree is not a model as good as deep learning for sophisticated pattern recognition, mostly because of magnetic field's and acceleration's components and the segmentation approach to them.

With the trained decision tree, we had the possibility to study the importance of the various attributes. For instance, originally, the proximity sensor (used in [2], indicating whether the phone is near an obstacle or not) was used together with the other indicators, but thanks to the DT we noticed that its correlation to the output labels was less than 1%, so it was deleted.

This study on the DT also was useful to reinforce the thesis about some indicators. In fact, the correlation between the *automotive* class and the *speed* attribute is over 50%, meaning that its usage for better detecting a vehicle is actually effective. The *cadence*, instead, that is the only novel indicator introduced, has a strong negative correlation with the *automotive* class, meaning that it is not useful for detecting a vehicle (obviously, it is 0). It also has a negative correlation with the *stationary* classes, both indoors and outdoors, while it has a strong correlation with the *walking* and *running* categories, in both locations. This was the objective we tried to reach by including this indicator.

Another analysis performed regarded acceleration and magnetic field intensity. The three components of the acceleration are strongly correlated between each other, while the variances of the magnetic field components are almost independent from each one.

More complex correlations between two indicators and an output label have been considered in section 4.1.1. In conclusion, all the indicators considered after the deletion of the proximity sensor have a correlation with some of the output targets (without exceeding the minimum of 1% or the maximum of 50%), they do not have values that are all different (0% ID-ness), but neither values that are all the same (near 1% stability), and also do not contain missing values. For this reason, they were used for training the Deep Learning model.

4.1.3 Model training

With the help of the RapidMiner tool, that is user friendly and very easy to use, we found out that the best possible model for classifying the gathered dataset was a Deep Learning model, as said in previous subsection.

The considered Deep Learning model is a multilayer feed-forward neural network. It has been trained using a back-propagation approach, with stochastic gradient descent. First, a short digression to explain those concepts is needed.

The artificial neural network is a parallel processing unit consisting of neurons and the connections between them, that they can learn by examples. One of the most important features of this model is its *adaptivity*, meaning that it can easily percept environment changes and adapt to them.

The neuron is mainly constituted by a non-linear function ϕ that combines altogether all the $\{x_1, ..., x_n\}$ input signals, each one with its weight contained in the vector $\{w_1, ..., w_n\}$, and outputs the signal y.

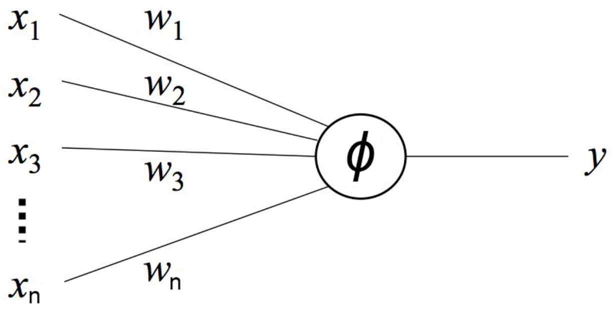


Figure 17: Sample model of a neuron with n inputs and one output

The function ϕ takes a weighted sum of the inputs and returns y.

$$y = \phi(\sum_{i=0}^{n} w_i x_i) = \phi(w^T x) \tag{4.2}$$

The function ϕ is also called **activation function** of the neuron, and it creates the non-linear relationship between the inputs and the outputs. If the activation function used a unit step function, this single neuron would be a **perceptron**. In a neural network, there is always a difference between the desired output d and the actual output $\phi(w^T x)$, and the goal of training a neural network is to minimize the distance between those two values, measured by an **objective function**.

The weight vector needs to reach a final state in which the objective function is minimized, and this training is done via **stochastic gradient descent**, using the derivative of the activation and the objective function. Many functions can be used to estimate the distance between the desired output and the actual one, and the most common is the **squared error function**:

$$E = \frac{1}{2}(d - y)^2 = \frac{1}{2}(d - \phi(w^T x))^2 \tag{4.3}$$

"In SGD, the weight parameters are iteratively updated in the direction of the gradient of the loss function until a minimum is reached" [49]. This approach is an improvement with respect to the traditional gradient descent, and it presents various benefits. The most important one is that SGD requires less memory exploitation. In fact, for training, it uses a single entry at a time, or at most a batch of some entries. Traditional gradient descent needs the whole dataset for training, all loaded into the main memory. For this reason, SGD can converge faster and with crucial updates done with less memory consumption.

From the definition of the objective function E it is possible to calculate its gradient with respect to any of the weights w_i, and then use it in the stochastic gradient descent update equation.

$$w_i = w_i^{old} - \eta \cdot \frac{\partial E}{\partial w_i} \tag{4.4}$$

with $\eta > 0$ called *learning rate.* Obviously, the gradient of the objective function with respect to one of the weights is a combination of the derivative of the objective function itself, the derivative of the activation function, and the input signal for that weight.

$$\frac{\partial E}{\partial w_i} = (d_i - y_i) \cdot \phi'(v_i) \cdot x_i \tag{4.5}$$

with v_i that is the signal preceding the weight in the graph. Therefore, with this approach all the points belonging to the dataset are progressively fed to these equations, until the optimal weight vector is computed.

In a real neural network, there could be lots of inner layers, hidden between the inputs and the outputs. There are various types of network, such as recurrent ones (with the output reentering as input), feed-forward ones (with connections only in one way, from the inputs to the outputs) and more. As said before, the considered model uses a feed-forward type of network. The outputs, also, could be more than one, constituting an entire vector of values, like in the case presented by our research where nine different values are needed. This network structure is better illustrated in Figure 18.

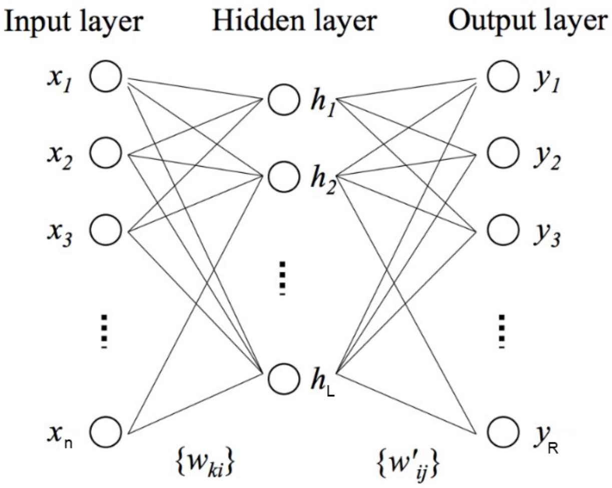


Figure 18: Example of a neural network with n inputs, R outputs and one hidden layer with L neurons

The activation function is computed in the same way it was computed before, and every neuron belonging to the same layer has the same activation function. The objective function, instead, is the same as before, but it is the sum of the objective functions of all output neurons.

$$E = \frac{1}{2}\sum_{i=1}^{R}(d_i - y_i)^2 \tag{4.6}$$

The Equation 4.6 is the most common objective function used for constructing neural networks, also known as **MSE** (mean squared error). Each one of the gradients needed for the stochastic gradient descent update equation can be computed by designing the feed-back (FB) network, i.e., the feed-forward (FF) network with all the weights and activation functions heading back from the outputs to the inputs. The construction of the FB network is effortless: each splitting point in the FF becomes a summing point in the FB, while each summing point in the FF (each neuron) becomes a splitting point in the FB. For each considered weight w_i in the network, the gradient descent for the update is computed this way:

$$\frac{\partial E}{\partial w_i} = -\begin{pmatrix} signal\, before\, w_i \\ in\, the\, feedback\, graph \end{pmatrix} \cdot \begin{pmatrix} signal\, before\, w_i \\ in\, the\, feedforward\, graph \end{pmatrix} \tag{4.7}$$

This straightforward equation can be computed by only looking at the two networks and then calculated for each weight, to update all of them.

This process is known as **backpropagation** because it begins with the final error $d_i - y_i$ for the output neuron i and this error gets propagated backward throughout the network to update the weights. There are several optimization techniques used for making the backpropagation perform better and to make it avoid overfitting; most of them have been used in the experiments, as described in next part of this section.

The deep learning model implementing an artificial neural network has been chosen for its adaptivity, as said before, but mainly because it is the best machine learning model for pattern recognition. The factors are layered and strongly correlated with each other, and interacting altogether can provide outstanding results [50]. The correlation between the indicators' values is the basis of this research work and its approach.

The best approach was found on the RapidMiner tool, but the actual training was made by using the Python library Keras. It is a wrapper that simplifies the more complex Tensorflow library, which can be considered as the "back-end" of Keras. Tensorflow is the most popular library for developing deep learning models in Python. The model was tuned with the features described next, and their order of application is not essential (tuning of parameters has been done with a trial and error method, stopping when the best result was reached).

- The activation function used by the neurons in the hidden layers was the *rectifier linear unit* (or ReLU), which chooses the maximum in the interval $(0, x)$ where x is the input value, approximating the function $log(1 + e^x)$. This is becoming the most used function for activating neurons in hidden layers, more than any other, in particular the sigmoid function $\frac{1}{1+e^{-x}}$, which has a domain interval on [0, 1], and for this reason ReL is better for modeling positive real numbers. Furthermore, the gradient of the sigmoid function (other popular activation function) vanishes as increasing or decreasing x, while the gradient of the ReL function does not vanish as increasing x. *Softmax* ($\frac{1}{1+e^{-\theta^T x}}$) is instead the activation function used in the output layer, since this function remarks the differences between each one of the classes and can give a set of confidence values (probabilities) for the output labels, all values summing to 1. Generally, in the literature, in the output layer of different kind of neural networks, softmax is used when each input can belong to exactly one class [51; 52], while in other cases sigmoid is better (for example in a binomial classification), because it is like a marginal distribution. The study of this thesis belongs to the first case and for this reason we used softmax. Moreover, the usage of sigmoid in the output layer seems to spoil the results obtained with ReL in the hidden layers, another plus for using softmax.

- The aforementioned hidden layers are two, with 70 neurons the first one, and with 120 neurons the second one, since each layer adds levels of complexity that result in a more accurate (but slower) training. For what concerns layers, we tried to build a network with a single hidden layer, with poor performances, but we also tried to add more layers, obtaining similar accuracies but resulting in a much slower training. For what concerns the number of neurons, instead, the optimal results were obtained in the interval [65, 130] for both layers, and after several tests we chose the combination that led to the best performance.
- The dataset training has been iterated for 50 epochs. We also tried with a different number of epochs, and we noticed that from 42 epochs we started obtaining good results, but while approaching to 50 epochs the accuracy increased even more. Using more than 50 epochs resulted in a waste of time, because the accuracy was almost the same, while training time was much slower.
- The adopted modifying learning rate algorithm was Adadelta. This approach is useful in deep learning algorithms because it does not keep track of all the past squared gradients by "restricting the window to some fixed size **w**" [53]. In this way, it avoids slow convergence, but it also prevents a vigorous decrease of the learning rate. It simplifies the setting of

new parameters by asking only two new ones: ϵ, the error rate testing the maximum difference allowed between the target value and the actual output, has been tested as 10^{-7}; ρ, similar to the **momentum** (most widespread technique for helping backpropagation algorithm escaping by local minima), has been tested as 0.99. There is an extension of this algorithm called ADAM, that uses the momentum itself, but its usage did not improve the performance.

- Instead of the usual MSE function for evaluating the objective function of the network, the categorical cross-entropy has been exploited. In fact, when using a neural network for prediction, it is better in some ways to exploit the cross-entropy error rather than the MSE, because it is slightly more accurate for evaluating the effectiveness of a neural network. The MSE, in fact, gives too much significance to the incorrect outputs, while the cross-entropy function (Equation 4.8), thanks to the natural logarithm, better considers the closeness of a prediction and it is a more coarse-grained way to compute error. The formula for cross-entropy is this one:

$$L(\theta) = -\frac{1}{n}\sum_{i=1}^{n}[y_i \ln(p_i) + (1 - y_i)\ln(1 - p_i)] = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m} y_{ij}\ln(p_{ij}) \tag{4.8}$$

  where i refers to the sample, j to the classes, y is the sample label and p_{ij} is the prediction for a sample.

A prevalent lousy phenomenon that affects many types of machine learning models is **overfitting**, which is "the production of an analysis that corresponds too closely or exactly to a particular set of data" [54], affecting future observations. Usually, the overfitting phenomenon is described by the graph in Figure 19, where the "particular set of data" is the training set and the "future observation" is the testing set. It occurs when the test error starts increasing, while the training error keeps decreasing.

In deep learning algorithms there are various methods to prevent overfitting, and two of them have been exploited in our model:

- Regularization is a crucial component in preventing overfitting for large datasets, and it is described by Ian Goodfellow as *"any modification we make to the learning algorithm that is intended to reduce the generalization error, but not its training error"*. **Generalization** in machine learning refers to how well the concepts learned by the model apply to examples which were not seen during training. The goal of most machine learning models is to generalize well from the training data, to make right predictions in the future for unseen data. The overfitting phenomenon happens when the generalization error is too high and the machine learning model adapts too much to the characteristics of the training dataset, noise and outliers included.

  To apply regularization there are two parameters that can be tuned: $L1$ and $L2$, we set only the first one. That parameter is used to reduce the complicatedness of a model and avoid overfitting, by applying some constraints on weights and their absolute value, dropping some of them. It has been set as 10^{-5}.

- 10-fold cross-validation has been used and the model was trained on the whole dataset, using this technique. It is an approach for evaluating machine learning models which consists in partitioning the original dataset into k non-overlapping folds with the same size. Then, for k times, the model is trained using $k-1$ partitions (*folds*) as the training set, while the remaining one is used as the validation set. At each iteration, the testing set fold is changed, until all folds have been used as validation only once. When this process ends, the various accuracy results can be averaged or combined to produce the final estimation. ”The advantage of this method is that all observations are used for both training and validation” [55].

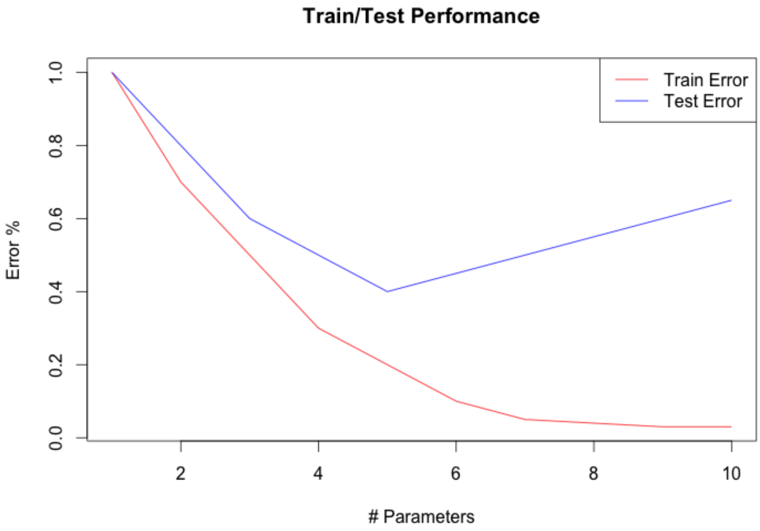


Figure 19: Overfitting phenomenon, comparing train error and test error

The deep learning model has been created and tested on a laptop mounting the Intel i7-7700HQ CPU @ 2.80 GHz, 16 GB of RAM DDR3, NVIDIA GeForce GTX 1050. The neural network training, considering the whole dataset randomly shuffled for cross validation, took 41 minutes. The overall performance of this model during training is left for the "Results of experiments" chapter.

4.2 Method validation on existing datasets

In section 4.1.1 we affirmed that there are no public datasets gathered for the combination of semantic location and activity of the user. However, in order to evaluate the approach proposed by us, various public datasets regarding the single modules have been considered. They were not used to train the model necessary for the real-time recognition, but rather to validate the appropriateness of the proposed feedforward neural network (FNN from now on) for indoor/outdoor detection and for various activities recognition.

Indoor/outdoor detection is a field in which research is much less developed than the activity recognition one, in fact it was possible to find only one already gathered dataset (the public one used in [56] only for the indoor/outdoor classifier), while three public datasets were used for human activity recognition. This is a brief description of all of them:

1. Indoor-Outdoor classifier data [56]: data colleted on an iPhone 6s. They used various indicators to train the detector, such as the barometric pressure, the GPS horizontal and vertical accuracy, the speed, the cellular signal strength intensity (by reading the count of signal bars from the status bar, like in our approach) and the resultant of the magnetic

field. The total dataset contains 4665 samples, each one consisting of 13 features registered with a frequency of 5 Hz. There were other 933 separated samples, used as testing set.

2. UCF - iPhone Data Set [57]: data collected on an iPhone 4. They used the three components of the 3D accelerometer, the gyroscope, and the magnetometer to classify some activities that are really close to the ones we classify. In fact, the targets are: biking (treated as *outdoor_biking*), gym biking (equivalent to our *indoor_biking*), running, standing (subset of *stationary*), treadmill walking (subset of *indoor_walking*), walking, and other labels that could easily be deleted, since they are not useful for this research. The total dataset, regarding those targets, contains 123753 samples, each one consisting of 9 features registered with a frequency of 60 Hz. Since there was no testing set, the dataset was split into 80% training and 20% testing.

3. WISDM - WIreless Sensor Data Mining [58]: data collected on an Android system. They used the three components of acceleration to classify six different activities: walking, jogging, sitting, standing, walking upstairs, walking downstairs. The total dataset contains 1098207 samples, each one consisting of 3 features registered with a frequency of 20 Hz, by 36 different users. However, this raw dataset did not lead to a good performance, with any model. For this reason they transformed the dataset to obtain 5424 samples, with 46 features each. They took 10 seconds worth of accelerometer samples (200 raw records) and transformed them in a single tuple. The features are: 10 bins for each component, with values that are the fraction of accelerometer samples that fell within that bin; the average value for each component over the 200 samples; the approximation of the

dominant frequency for each component over the 200 samples; the absolute and standard deviation; the resultant. Since the results of other approaches were all based on 10-fold cross validation, we evaluated the accuracy of the proposed FNN in the same way.

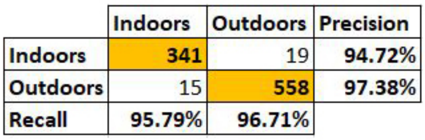
4. UCI - HAR [59]: data collected on a Samsung Galaxy S II. They used the three components of the total acceleration and the estimated body acceleration, the three components of the gyroscope, a 561-features vector with time and frequency domain variables to classify almost the same six activities seen with the WISDM, with laying instead of jogging. The latter, in fact, is an alternative to this dataset, which is way more famous and exploited in the activity recognition field. The total dataset contains 7353 samples, each one consisting of 563 features (only 477 were used in this research, because some of them were unuseful) registered as sliding windows of 2.56 seconds and 50% overlap, by 30 different users. The creators of the dataset also provided a testing set of 2948 samples.

4.2.1 Results with FNN approach

The FNN approach proposed in this thesis was applied to each one of the previous four datasets, and it has been compared to other previously analyzed methods, in order to show a comparison between the various studies exactly on the same datasets. On the vertical side of the confusion matrices there are **predicted** values for the various rows while on the horizontal side there are the **actual** values for the various dataset rows.

1. For the indoor/outdoor classifier dataset, our proposed feedforward neural network approach has been compared to the original work itself [56], where on the same dataset a Random Forest model, an SVM one, an LSTM neural network and another FNN were

tested. The proposed FNN seems to outperform all the other methods on the testing set (Figure 20).



	Indoors	Outdoors	Precision
Indoors	341	19	94.72%
Outdoors	15	558	97.38%
Recall	95.79%	96.71%	

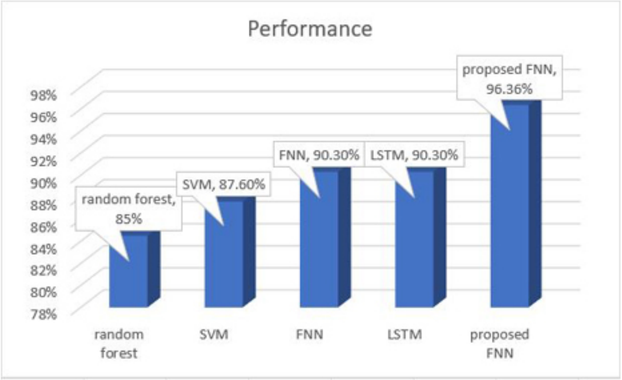
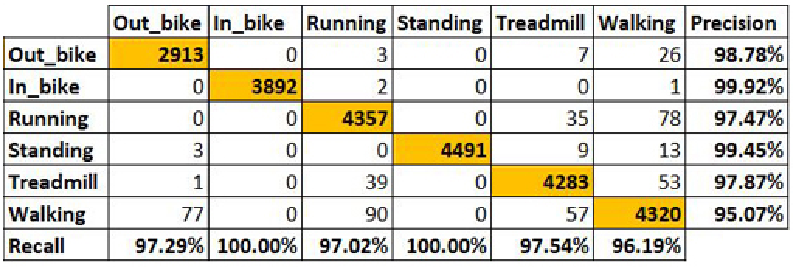


Figure 20: Indoor/outdoor dataset: confusion matrix of the proposed FNN approach on the left, and comparison with other existing approaches on the right

2. For the UCF iPhone dataset, our proposed FNN approach has been compared to the hierarchical K-NN classifier and the top-level clustering built in [57] itself. The proposed FNN seems to outperform the other methods on the testing set (Figure 21), considering that two classes have been deleted for training our version of the model.

3. For the WISDM dataset, our proposed FNN approach has been compared to the Weka J48 decision tree model, the logistic regression and the multilayer perceptron (FNN) built in [58] itself. While the performance of the proposed FNN was very low (**61%**)



	Out_bike	In_bike	Running	Standing	Treadmill	Walking	Precision
Out_bike	**2913**	0	3	0	7	26	**98.78%**
In_bike	0	**3892**	2	0	0	1	**99.92%**
Running	0	0	**4357**	0	35	78	**97.47%**
Standing	3	0	0	**4491**	9	13	**99.45%**
Treadmill	1	0	39	0	**4283**	53	**97.87%**
Walking	77	0	90	0	57	**4320**	**95.07%**
Recall	**97.29%**	**100.00%**	**97.02%**	**100.00%**	**97.54%**	**96.19%**	

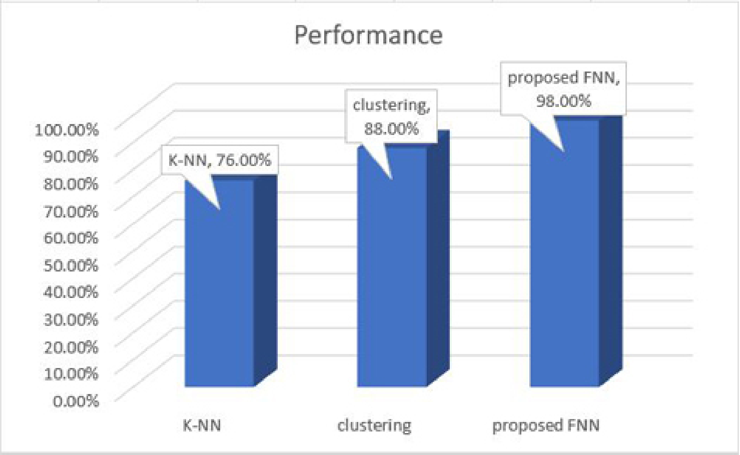
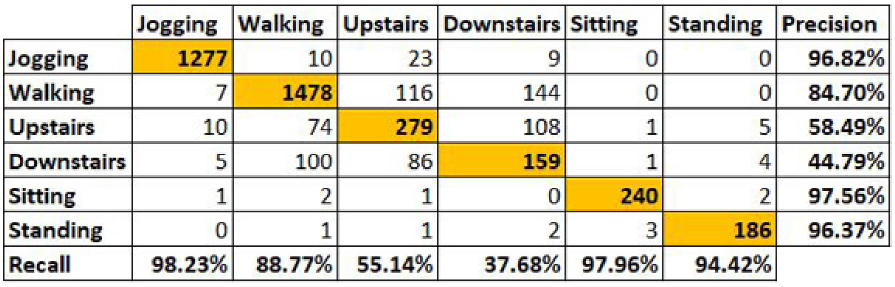


Figure 21: UCF - iPhone dataset: confusion matrix of the proposed FNN approach on top, and comparison with other existing approaches on bottom

on the raw dataset, it had a strong improvement on the transformed dataset, even if it did not outperform the methods proposed by the researchers (Figure 22). However, the complexity of this dataset is not enough to train a performing machine learning model on it. In fact, with the UCI-HAR dataset, which classifies almost the same activities of this one, our proposed FNN had outstanding results on the testing set.



	Jogging	Walking	Upstairs	Downstairs	Sitting	Standing	Precision
Jogging	**1277**	10	23	9	0	0	**96.82%**
Walking	7	**1478**	116	144	0	0	**84.70%**
Upstairs	10	74	**279**	108	1	5	**58.49%**
Downstairs	5	100	86	**159**	1	4	**44.79%**
Sitting	1	2	1	0	**240**	2	**97.56%**
Standing	0	1	1	2	3	**186**	**96.37%**
Recall	**98.23%**	**88.77%**	**55.14%**	**37.68%**	**97.96%**	**94.42%**	

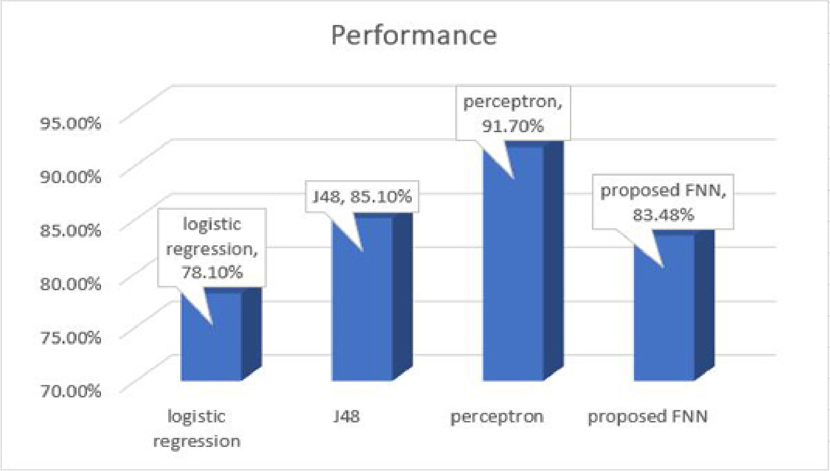
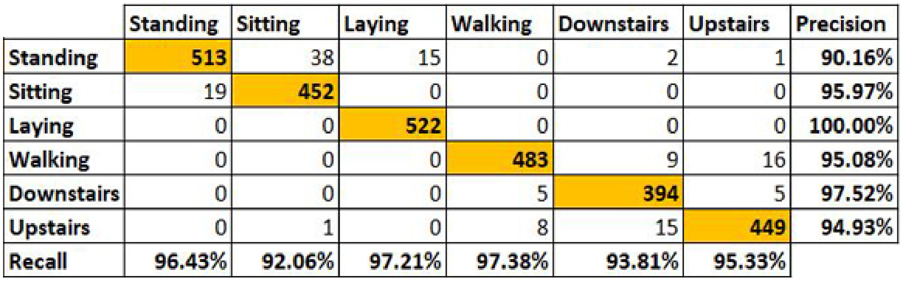


Figure 22: WISDM dataset: confusion matrix of the proposed FNN approach on top, and comparison with other existing approaches on bottom

4. For the UCI - HAR dataset, our proposed FNN approach has been compared to the SVM proposed by the creators of the dataset themselves [59], the convolutional neural network (CNN) proposed in [60], and the sequential extreme learning machine proposed in [61]. The proposed FNN reaches a result near to the best one on this testing set (Figure 23), considering that the classes *upstairs* and *downstairs* are not useful for this research.



	Standing	Sitting	Laying	Walking	Downstairs	Upstairs	Precision
Standing	**513**	38	15	0	2	1	**90.16%**
Sitting	19	**452**	0	0	0	0	**95.97%**
Laying	0	0	**522**	0	0	0	**100.00%**
Walking	0	0	0	**483**	9	16	**95.08%**
Downstairs	0	0	0	5	**394**	5	**97.52%**
Upstairs	0	1	0	8	15	**449**	**94.93%**
Recall	**96.43%**	**92.06%**	**97.21%**	**97.38%**	**93.81%**	**95.33%**	

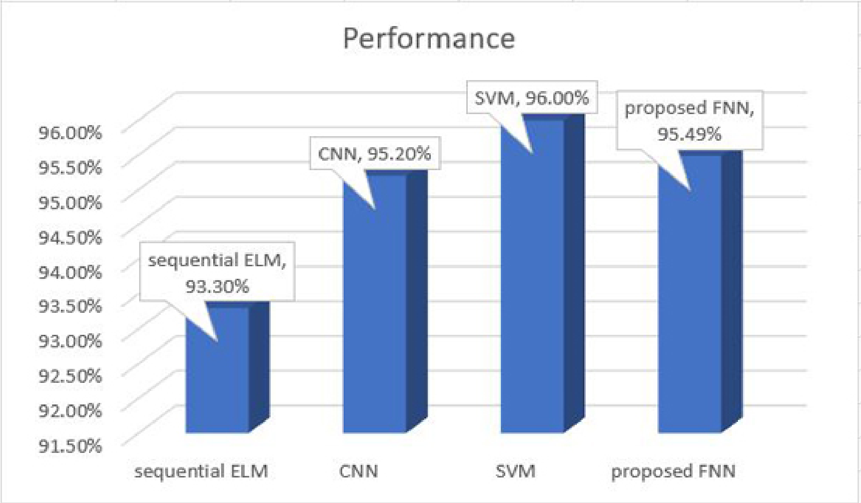


Figure 23: UCI-HAR dataset: confusion matrix of the proposed FNN approach on top, and comparison with other existing approaches on bottom

The difference between the last two performances underlines the fact that our proposed FNN constitutes an optimal approach to activity recognition, but it needs a properly diversified and complex dataset for training an appropriate model. In fact, the performance on the WISDM dataset is affected by the *upstairs* and *downstairs* classes, that in the UCI-HAR dataset are classified with a high accuracy. This means that the WISDM dataset does not contain proper readings for those two classes, and our model is not able to reach great results. It is possible to think that the difference between the two accuracies depends on the different sixth activity classified but, by analyzing the confusion matrices, it can be noted that the two troublesome classes are affected by each other and by the *walking* class, meaning that the main problem is constituted by the dataset itself.

4.3 iOS app experimental implementation

The deep learning model was created on Keras because it is one of the few machine learning frameworks supported by Core ML for exporting and implementing a model in an iOS application. In fact, the model has been exported in `.json` and the weights of the deep learning network in `.h5`, then the library `coremltools` for Python has been used to convert the model and get it with the Core ML extension `.mlmodel`. Finally, using the model in an iOS application is very simple, since it is treated as a class like any other one.

As said in chapter 3, the application gathers magnetic field's and acceleration's samples using `CMDeviceMotion` every 0.2 seconds (i.e., with a frequency of 5 Hz). After having gathered five samples and computed their variance, it uses all the features coming from the various sensors to feed them into the Core ML model, getting as output an array with nine entries, one for each of the target values discussed in chapter 3. The higher is the refresh rate for gathering data, the more iPhone's resources the application will consume. The lower is the refresh rate, the slower the data will be updated. A compromise has been found in this way, in order to reduce the battery consumption.

Following the algorithm designed in chapter 3, several tests on all its facets and their reliability were conducted. If the confidence value is higher than 0.8, the output value is correct almost all the time. In fact, if a value different from the one detected in the previous instant is classified, this has been proved by the experiments that it means an indoor/outdoor switch is going on most of the times, while other times there are simple glitches in recognizing some activities. The most evident example comes from the *automotive* classification, which is always detected with high confidence, except for the stops (at semaphores with the car, at stations with trains and at both with buses) where the application recognizes *indoor_stationary* as an activity with high confidence since the user is still. In this case, the application exploits Core Motion, which during the experiments never switched the detected activity from *automotive* to *stationary* at semaphores or stops and kept storing *automotive* as long as the trips went on, because it is slower in changing its recognition for situations like this one. Hence, the limitations coming from the speediness of recognition of the deep learning model were overcome by interacting with the Core Motion framework. Figure 24 shows an example of what has been explained so far.

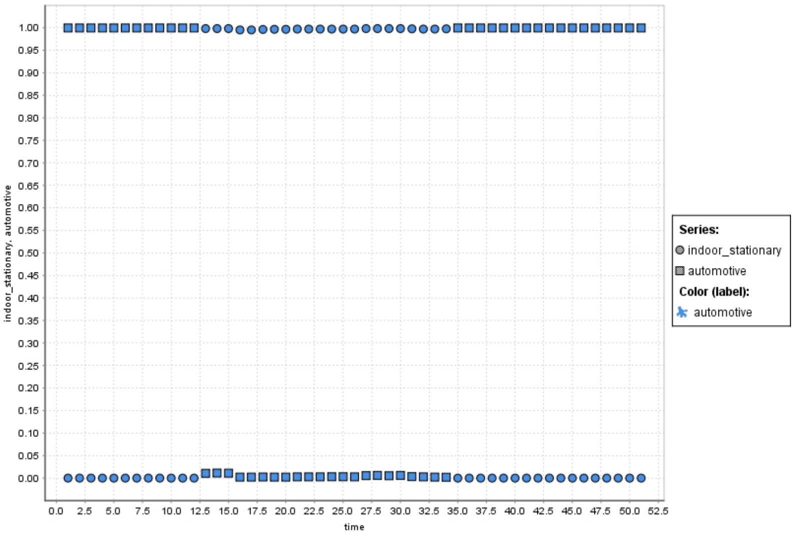


Figure 24: Switch between confidences of automotive and indoor_stationary while stopping, in a 10 seconds window

As it is visible in Figure 24, the detected activities had both high confidence almost instantly, while the Core Motion framework kept detecting automotive, as it is visible by the color of squares and dots (written in the legend).

Otherwise, if the maximum confidence value is less than 0.8, the interpretation of data and experiments is more complicated, and the algorithm for this part was modified following the experimental results. In fact, the activity switch is entirely different from the I/O switch, which is often instantaneous. Its detection, indeed, can last from a minimum of 2 seconds (tested on the switch from *indoor_running* to *indoor_walking*) to a maximum of 8.8 seconds (tested on the switch from *indoor_stationary* to *indoor_walking*), depending on which human activities are involved in it. As Figure 25 and Figure 26 show, it has been tested that during the switch the first activity's confidence slowly decreases while the second highest confidence value keeps increasing, until the two switch themselves. In the algorithm described in chapter 3, a five seconds time window to look for possible behaviors similar to this one was mentioned, and this amount of time has been chosen because it is the window length that gave the best results. The duration of the switch is stored, in order to assign $\frac{1}{3}$ of the time window to the old target value and $\frac{2}{3}$ of it to the new target value. In fact, it emerged from the experiments that this is the proportion minimizing the uncertainty and the loss of accuracy, as shown in the figures by the yellow band, which highlights the actual time windows of the old activity. This approach is the way we handle **delay** during activity recognition in our application.

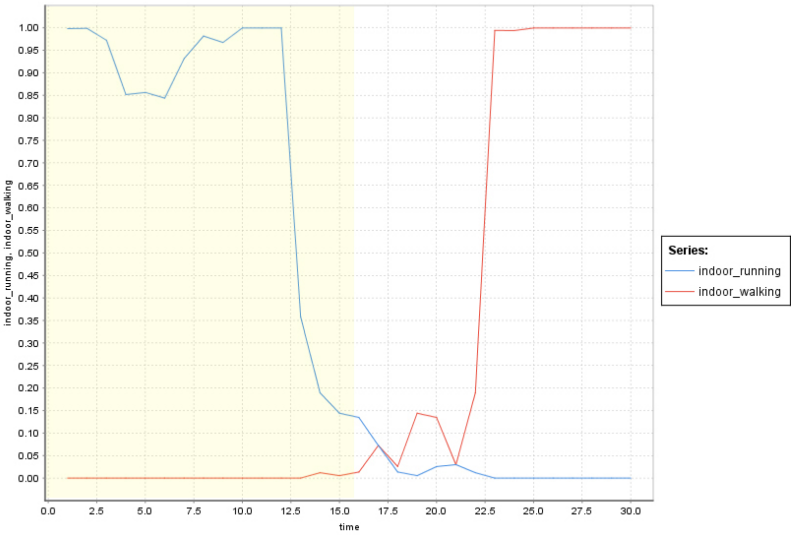


Figure 25: Activity switch from indoor_running to indoor_walking, fastest one with only a 2 seconds switch, with the actual window of indoor_running highlighted

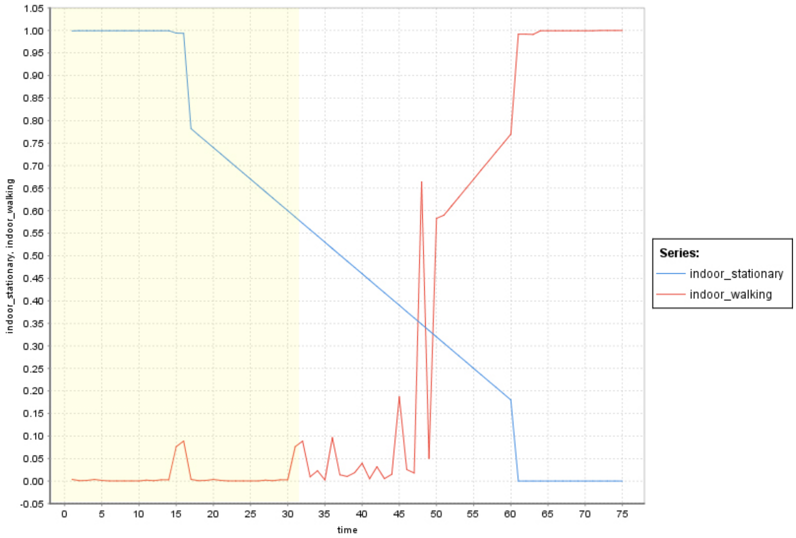


Figure 26: Activity switch from indoor_stationary to indoor_walking, slowest one with 8.8 seconds switch, with the actual window of indoor_stationary highlighted

If there is not an activity switch, the application tries to reinforce its detected value using Core Motion, but this happens hardly ever, and it is the only source of uncertainty in the detection. For Core Motion, the detection delay of the switch between various activities oscillates between a minimum of 1.5 seconds to a maximum of 6 seconds. The efficiency of Core Motion framework has been tested by storing the detected activity while gathering data in the initial phase, to compare it to the actual activity ongoing at that moment. Its performance evaluation is severely influenced by the *indoor_biking* category, since going on an exercise bike is never recognized by Core Motion as *biking*, but as *walking* or *running*, depending on the velocity.

Every confusion matrix in this thesis contains the computed **overall accuracy** written in the rightest cell in the last row, starting from Table II, where it is possible to examine that the overall accuracy of Core Motion framework, in this case, is affected by the biking class recall, since almost all the samples gathered with label *indoor_biking* are classified by it as walking, running or stationary. In Table III a confusion matrix generated excluding the *indoor_biking* category can be observed.

The last classification errors remaining are due to the switches, since the user gathering data almost instantly changed the label, while it took some seconds for Core Motion to detect the new activity. Another factor affecting performances is the M8 motion coprocessor installed on the testing device, which is quite old. In fact, with the latest M11 coprocessor performances would be better and detection delay would be shorter. By looking at Core Motion performances on this same study, we noticed that our real-time activity recognition module (Table XI in section 5.2.1) reaches a performance 6% better than it (only in this specific case). In fact, in this study, the *indoor_biking* category needs to be considered, and therefore the Core Motion performance examined for comparison is the first one.

TABLE II: PERFORMANCE EVALUATION OF THE CORE MOTION FRAMEWORK

		Actual					
		Auto	**Still**	**Walk**	**Run**	**Bike**	**Precision**
Predicted	**Auto**	**8515**	0	58	0	0	**99.32%**
	Still	520	**9408**	404	0	452	**87.24%**
	Walk	291	1785	**22160**	367	3042	**80.16%**
	Run	153	0	166	**4529**	534	**84.15%**
	Bike	0	876	0	0	**9355**	**91.44%**
	Recall	**89.83%**	**77.95%**	**97.24%**	**92.50%**	**69.90%**	***86.97%***

TABLE III: PERFORMANCE EVALUATION OF THE CORE MOTION FRAMEWORK, WITHOUT CONSIDERING THE INDOOR_BIKING CATEGORY

		Actual					
		Auto	**Still**	**Walk**	**Run**	**Bike**	**Precision**
Predicted	**Auto**	**8515**	0	58	0	0	**99.32%**
	Still	520	**9408**	404	0	452	**87.24%**
	Walk	291	1785	**22160**	367	572	**88.02%**
	Run	153	0	166	**4529**	0	**93.42%**
	Bike	0	876	0	0	**9355**	**91.44%**
	Recall	**89.83%**	**77.95%**	**97.24%**	**92.50%**	**90.13%**	***90.71%***

CHAPTER 5

RESULTS OF EXPERIMENTS

The results of the experiments described in chapter 4 are presented in this chapter, both for machine learning model performances after training on Keras (offline classification of the training set without transitions between activities) and for the performances obtained trying the iOS application implementing the Core ML model (real-time with transitions between activities).

5.1 Testing on Deep Learning model

The mean performance of cross validation, on the 10 different folds, was of **97.24%**. Table IV contains the overall performances over each one of the nine output classes of the deep learning model. These are the results obtained by summing the various classifications detected during the training process. On the vertical side there are **predicted** values for the various rows while on the horizontal side there are the **actual** values for the various dataset rows.

A simple explanation of the values in Table IV: precision is also called positive predictive value and indicates how many of the selected items are significant, while recall is also called sensitivity or true-positive rate and indicates how many of the significant items have been selected. In easier words, precision is "how many of the predicted values for a class truly belong to that class" and recall is "how many of the values belonging to a class have been classified correctly". The two measurements are strongly correlated.

TABLE IV: OVERALL PERFORMANCE EVALUATION OF THE DEEP LEARNING MODEL

		Actual									
		Auto	**In bike**	**In run**	**In still**	**In walk**	**Out bike**	**Out run**	**O_still**	**O_walk**	**Precision**
Predicted	**Auto**	**9467**	0	0	0	6	0	0	0	0	**99.94%**
	In bike	0	**2942**	5	0	0	0	0	0	14	**99.36%**
	In run	0	1	**3266**	0	6	0	3	0	2	**99.63%**
	In still	0	50	140	**4679**	167	0	0	3	18	**92.52%**
	In walk	0	11	20	41	**6564**	0	5	3	194	**95.99%**
	Out bike	0	0	0	0	15	**10816**	20	334	187	**95.11%**
	Out run	0	0	0	0	0	0	**1393**	0	9	**99.36%**
	Out still	12	0	1	0	6	57	0	**8165**	347	**95.07%**
	Out walk	0	0	12	0	48	3	31	41	**15405**	**99.13%**
	Recall	**99.87%**	**97.94%**	**94.83%**	**99.13%**	**96.36%**	**99.45%**	**95.94%**	**95.54%**	**95.23%**	***97.24%***

TABLE V: PERFORMANCE EVALUATION OF THE INDOOR/OUTDOOR CLASSIFICATION OF THE DEEP LEARNING MODEL

		Actual		
		Indoor	**Outdoor**	**Precision**
Predicted	**Indoor**	**17892**	242	**98.66%**
	Outdoor	88	**36808**	**99.76%**
	Recall	**99.51%**	**99.35%**	***99.32%***

In previous works, no one had ever researched a method for concurrent classification of semantic location and activity of a user, but only in two separate ways, as explained in chapter 2 by completely dividing the two sections for previous work. By extracting the single indoors/outdoors and activity samples, it is possible to compare the relative performance of the deep learning classifier with some of the results obtained in the literature.

For indoor/outdoor detection, it is possible to extract the confusion matrix in Table V from the previous Table IV, constructed for evaluating the validation set accuracy, considering only the two classes and gathering in them all the various indoor and outdoor values. The *automotive* class has been excluded from the computation. However, the actual values belonging to the various classes but erroneously classified as *automotive* have been counted into the actual number of inputs for each one of the two targets.

In Table VI our performance is compared to other indoor/outdoor detection models created starting from sensors' data on smartphones. We considered the IODetector method with fixed

TABLE VI: COMPARISON BETWEEN DEEP LEARNING MODEL AND LITERATURE ON INDOOR/OUTDOOR DETECTION

Method	**Overall accuracy**
IODetector	61.20%
Co-training (SVM attr.)	93.33%
Ultrasonic (slow latency)	92.70%
GSM (Random Forest)	95.30%
Neural Network	95.40%
Our Deep Learning	**99.32%**

thresholds [5], the correction for IODetector using co-training and separating the attributes via the SVM Attribute Ranking [2], the ultrasonic pings method described at [3], the classification that only uses the GSM sensors to detect near stations, with a Random Forest model [4]. For a complete validation of our method, we also needed to compare it to another study using the same approach (i.e., using a neural network for classification). Since the indoor/outdoor detection is a binomial classification, many researchers prefer not to use a complex neural network. However, there is a study on how to predict the floor of a user for 911 calls with smartphone sensors' data that exploits a neural network for detecting if the user is indoors or outdoors (with some of the same indicators we are using, such as magnetic field intensity or cellular signal intensity, plus GPS properties), first, and then performs other classifications for predicting the floor [56]. Obviously, we were interested only in the indoor/outdoor detection with a feedforward neural network, for comparing their results with our neural network.

From Table VI emerges that our method seems to outperform the other indoor/outdoor detection ones.

TABLE VII: PERFORMANCE EVALUATION OF THE ACTIVITY RECOGNITION OF THE DEEP LEARNING MODEL

		Actual					
		Auto	**Still**	**Walk**	**Run**	**Bike**	**Precision**
Predicted	**Auto**	**9467**	0	6	0	0	**99.94%**
	Still	12	**12848**	538	141	107	**94.15%**
	Walk	0	85	**22211**	68	14	**99.25%**
	Run	0	0	17	**4662**	1	**99.61%**
	Bike	0	334	216	25	**13758**	**95.99%**
	Recall	**99.87%**	**96.84%**	**96.62%**	**95.22%**	**99.12%**	***97.66%***

For activity recognition, we are going to use the same method of comparison, by constructing a more complex confusion matrix (Table VII), capable of listing all the five detected activities, including automotive.

In Table VIII our performance is compared to other activity recognition models created starting from sensors’ data on smartphones, even if they do not recognize the same activities that our model classifies. We considered the Support Vector Machine model used for classifying activities such as ”in car”, ”on foot”, ”tv watching” and ”by bike” [21], the convolutional neural network (CNN) model for classifying ”walking”, ”walking upstairs”, ”walking downstairs”, ”laying”, ”sitting”, ”standing” [22], and the Gaussian Mixture Model (GMM) for classifying ”bus”, ”subway”, ”walking”, ”jogging”, ”staying” [23]. This comparison is used mostly for introducing our method in the broad activity recognition world.

TABLE VIII: COMPARISON BETWEEN DEEP LEARNING MODEL AND LITERATURE ON ACTIVITY RECOGNITION

Method	**Overall accuracy**
Non-linear SVM	99.50%
GMM	95.00%
CNN	94.00%
FNN	95.24%
RNN	96.70%
Our Deep Learning	**97.66%**

For a complete validation of our method, we needed to compare it to another study using the same approach (i.e., using a neural network for classification), in this case too. There are several research studies using different types of neural networks for activity recognition, and three approaches were considered: a convolutional neural network, already cited before, a long short-term memory deep recurrent neural network (RNN) using accelerometer, gyroscope and magnetometer from the UCI-dataset [62], and a feedforward neural network (FNN) using the readings coming from the accelerometer but properly processed with the Fourier Transform and with an appropriate feature subset selection [63].

From Table VIII emerges that our method is really good for activity recognition, for its performances comparable to the other ones.

5.2 Performances of iOS application

In this section, we will analyze the performances, regarding both accuracy and resources consumption, of the real-time classification through the application developed following the algorithm described in chapter 3. Thanks to this algorithm, results similar to dynamic segmentation in real-time with transitions are achieved, by using all the described experimental methods (i.e., switching GPS on exclusively outdoors, detecting magnetic field and acceleration at 5 Hz frequency and other sensors data at 1 Hz, interpreting ambiguous results with the help of Core Motion framework). The overall accuracy results will then be split to check the efficiency of the single modules (indoor/outdoor detection and activity recognition), as done in the previous section for the Deep Learning model.

5.2.1 Accuracy of detection

For performance evaluation, in addition to the methods described before, is also considered the way we manage **delay**. In fact, all activity transitions detected in the list of predictions have been assigned as $\frac{1}{3}$ of the switch time window to the first activity, and $\frac{2}{3}$ to the second activity. The predictions have been collected by a single user in an environment similar to the one where the original dataset has been collected, in Chicago (so excluding California and Detroit).

Table IX contains the overall performances over each one of the nine output classes of the model stored in the iOS application, on the vertical side there are **predicted** values for the various rows while on the horizontal side there are the **actual** values for the various dataset rows.

The performances are lower with respect to the offline Deep Learning model's performances described in the previous section, but this is ordinary, since real-time classification with transitions is more complicated than offline classification. For the *indoor_walking* class, we had awful

TABLE IX: OVERALL PERFORMANCE EVALUATION OF THE REAL-TIME CLASSIFICATION IN THE IOS APPLICATION

		Actual									
		Auto	**In bike**	**In run**	**In still**	**In walk**	**Out bike**	**Out run**	**O_still**	**O_walk**	**Precision**
Predicted	**Auto**	**699**	7	6	7	5	19	0	0	3	**93.70%**
	In bike	0	**416**	27	0	27	0	0	13	0	**86.13%**
	In run	0	41	**1683**	1	23	0	1	0	0	**96.23%**
	In still	14	28	3	**897**	90	0	0	34	2	**83.99%**
	In walk	7	94	4	15	**1032**	35	4	21	82	**79.75%**
	Out bike	0	10	0	2	2	**2651**	23	7	42	**96.86%**
	Out run	0	0	14	0	8	34	**484**	0	15	**87.21%**
	Out still	1	7	0	4	15	40	0	**951**	48	**89.21%**
	Out walk	3	11	0	0	111	116	15	9	**1530**	**85.24%**
	Recall	**96.55%**	**67.75%**	**96.89%**	**96.87%**	**78.60%**	**91.57%**	**91.84%**	**99.03%**	**88.85%**	***88.84%***

TABLE X: PERFORMANCE EVALUATION OF THE INDOOR/OUTDOOR CLASSIFICATION IN REAL-TIME

		Actual		
		Indoor	**Outdoor**	**Precision**
Predicted	**Indoor**	**4406**	214	**95.37%**
	Outdoor	184	**5965**	**96.61%**
	Recall	**95.99%**	**96.54%**	***96.13%***

performances both for the precision and the recall of the class, and it happened because some predictions have been gathered in an open-indoors environment, with high signal strength, high light intensity and low variance of the magnetic field. For this reason, many detections were affected by these characteristics, and the application predicted *outdoor_walking* many times.

The most unsatisfactory performance is the recall for the class *indoor_biking*, since it has the same problem encountered for the Core Motion framework. In fact, many times it was recognized as *indoor_walking* class (another factor affecting the precision of it). Fortunately, it is an activity that is done in a tiny percentage of the day, compared to other activities.

Considering the two different parts as isolated one from each other, we can obtain confusion matrices (Table X and Table XI) for the accuracy both of the indoor/outdoor detection and of the activity recognition, as previously done for the Deep Learning model.

From Table X emerges that the indoor/outdoor accuracy is very similar to the Deep Learning model's one, and it is a high performance.

TABLE XI: PERFORMANCE EVALUATION OF THE ACTIVITY RECOGNITION IN REAL-TIME

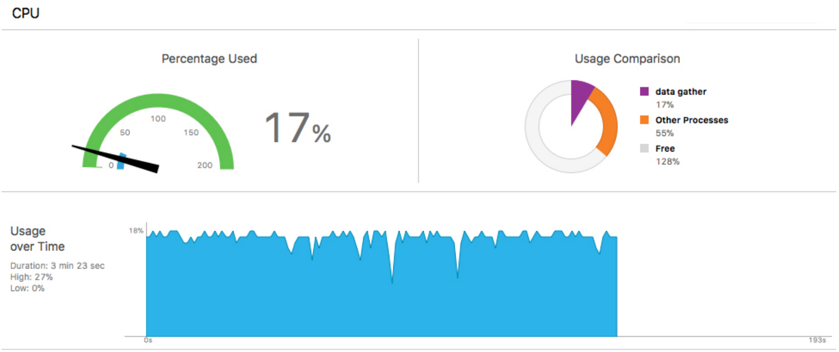
		Actual					
		Auto	**Still**	**Walk**	**Run**	**Bike**	**Precision**
Predicted	**Auto**	**699**	7	8	6	26	**93.70%**
	Still	15	**1886**	155	3	75	**88.38%**
	Walk	10	45	**2755**	23	256	**89.19%**
	Run	0	1	46	**2182**	75	**94.70%**
	Bike	0	22	71	50	**3077**	**95.56%**
	Recall	**96.55%**	**96.17%**	**90.77%**	**96.38%**	**87.69%**	***92.91%***

From Table XI the same problems deriving from the *indoor_biking* class emerge, both for the recall of the *biking* class and for the precision of the *walking* class. The precision of the *stationary* class, instead, is affected by the slow transitions from *stationary* to *walking*, as described in chapter 4. With the accuracy results coming from Table XI, it is possible to compare our approach, with fixed segmentation (time-based) and the exploiting of a specific algorithm to maximize the accuracy of detecting transitions and minimizing the accuracy loss due to delay for their recognition, to the dynamic segmentation approach mentioned in chapter 2 [24] for real-time activity recognition with transitions. Their accuracy on activities and transitions was of **92.9%**. Therefore our approach has the same results.

5.2.2 Resources consumption

The iOS application has a massive impact on device's resources consumption, since it is an application that should run the whole day, gathering data from many sensors at different sampling rates. For this reason, CPU and internal memory are exploited intensively for a long time, and all the precautions listed in the algorithm description are necessary for reducing battery consumption to the minimum.

CPU and memory usage have been tested directly using the XCode debugger while running the application on the device connected to the computer, as it is visible in Figure 27 and Figure 28.


CPU

Percentage Used

17%

Usage Comparison

data gather 17%

Other Processes 55%

Free 128%

Usage over Time

Duration: 3 min 23 sec

High: 27%

Low: 0%

Figure 27: CPU usage during 3.5 minutes of execution of the iOS application

As we can see in Figure 27 and Figure 28, the impact on internal memory is minimal (*data gather* is the temporary name given to the application), while the impact on CPU is

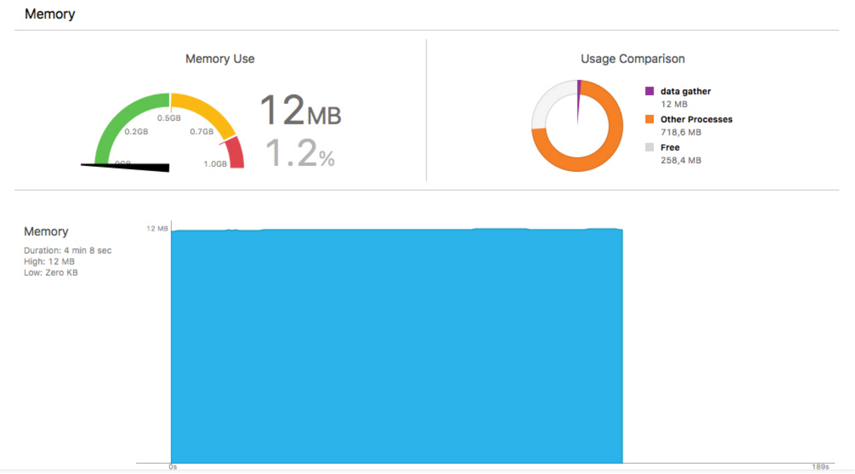


Figure 28: Memory usage during 4 minutes of execution of the iOS application

much higher since iOS creates many threads when gathering data from various sensors and concurrently checking Core Motion framework results. This type of resource usage leads to the analysis of battery consumption described next.

The test on battery consumption has been performed on an iPhone 6 bought in 2014, which mounts the motion coprocessor M8, the CPU A8, 2GB of RAM and, most important of all features, an 1810 mAh battery. For this reason, this first result will be strongly affected by the age and the characteristics of this device.

The application has been tested in two different situations, i.e., 1 hour of running it when the battery was fully charged (100% and 10 minutes waiting before unplugging the wire) and 30 minutes of running it when the battery was not much charged (30%). Both times the application was in use, never in the background (where power consumption is slower than the other situation). A visual representation of the consumption can be seen in Figure 29 and Figure 30, with the light-yellow area symbolizing the indoors time. In the charts, it is possible to see how much the iPhone also consumes without the application, with all the same things activated (also localization with GPS in the second part of testing). This comparison has been done to check the actual application's addition to power consumption.

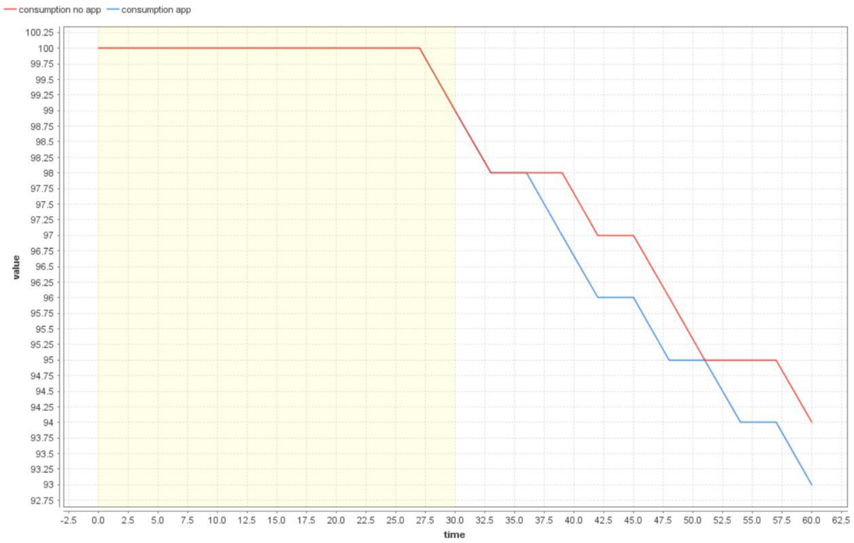


Figure 29: Comparison of battery consumption between a one-hour execution of the iOS application and a one-hour usage of the phone

Figure 30 demonstrates how on an iPhone, with a low-charged battery, any application consumes energy almost twice faster than a state in which the battery is fully charged. In fact,

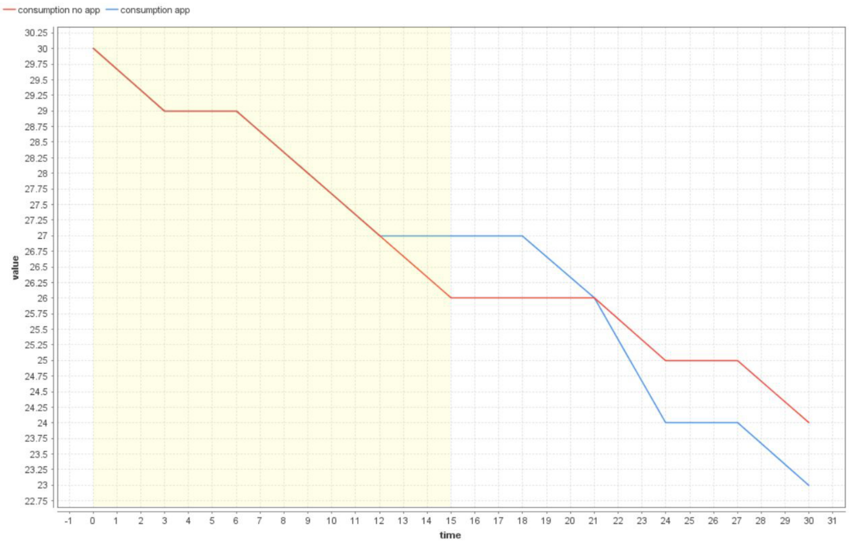


Figure 30: Comparison of battery consumption between a half-hour execution of the iOS application and a half-hour usage of the phone

it is not a problem generated by the iOS application, but rather a characteristic of iOS devices' batteries.

Both tests have been performed for half of the time indoors (i.e., with the GPS updates switched off) and the rest of the time outdoors or in an *automotive* state (i.e., with the GPS updates switched on and screen brightness higher due to the high level of ambient light), and, in fact, the difference between the first half of the graph and the second one is distinctly visible, in both plots. This level of power consumption is then acceptable, since people spend the highest percentage of the day indoors, and so the GPS sensor would be activated less than half of the day. However, more precautions will be studied and introduced in future work, even if the actual addition of this application to power consumption is around 1% or 2% of battery power.

CHAPTER 6

DISCUSSION AND CONCLUSIONS

In this thesis we propose a machine learning method to classify nine types of different activities, collecting and detecting together both semantic location (indoors or outdoors) and activity (stationary, walking, running, biking or automotive) of the user. The results reached are auspicious, also considering that, if looking at the two modules alone, they have an accuracy that can be compared to methods which are much more widespread and have been studied by teams of researchers.

Even considering these optimal results, there are some limitations to our detection that could not be overwhelmed. In fact, the closeness of the iOS system is a considerable limitation to what we want to do, especially considering that, without an illegal private framework for which there is no documentation readily available online, many sensors cannot be accessed, including the ambient light sensor, the cellular signal strength sensor and the battery temperature sensor (this feature could not be derived from anything else, so we did not consider it in our research). The ambient light can be easily derived by screen brightness, whose value is automatically updated basing on the ambient light sensor readings, and so there is a direct connection between the two (even if a limitation is given by the fact that the user must turn on the automatic brightness adjusting). The most substantial limitation we have, instead, is the impossibility to directly access the cellular signal strength sensor, since adapting from a wide range of dBm to only 4 bars of signal is very restrictive and could negatively affect the study.

The other problem we have with our study is the *indoor_biking* class influence on the accuracy of the model. In fact, it has a poor recall, since often is detected as a *walking* or *running* activity, and it negatively affects both precision and recall for the *indoor_walking* class, mostly. However, since the *indoor_biking* activity is sporadic if compared to the eight other ones, on a large scale of detecting activities during the whole 24 hours of the day, its influence will constitute a tiny percentage of the recognition accuracy.

In future work, as well as studying more efficient workarounds for iOS limitations, model and resources consumption testing and evaluation should also be done on a more recent iPhone, like the iPhone 8 Plus (M11 coprocessor, A11 Bionic CPU, 3GB of RAM and a 2675 mAh battery) or the newest iPhone X.

The purpose of this application is to be integrated into a more extensive project called MY-AIR, where the detection of the various activities will be used for computing an estimation of the daily average pollutant intake. In fact, the amount of the pollutant intake depends on various parameters, among which there are personal and clinical information about the user (such as age, gender, breath rate), the level of air pollution in the particular zone of the user (given by NASA satellites) and the various semantic locations and activities detected for the user during the day. The intake depends from all these parameters because there are different levels of pollution indoors, outdoors or in a car, and there are different intake rates depending on what activity the user does in a specific time window (all of these features will be given by our application, for iOS).

CITED LITERATURE

1. Ravindranath, L. S., Newport, C., Balakrishnan, H., and Madden, S.: Improving Wireless Network Performance Using Sensor Hints . In 8th USENIX Symp. on Networked Systems Design and Implementation (NSDI), Boston, MA, March 2011.

2. Radu, V., Katsikouli, P., Sarkar, R., and Marina, M.: A Semi-supervised Learning Approach for Robust Indoor-outdoor Detection with Smartphones, pages 280–294. SenSys ’14. ACM, 2014.

3. Bisio, I., Delfino, A., and Lavagetto, F.: Poster: Detecting if a smartphone is indoors or outdoors with ultrasounds. In MobiSys, 2015.

4. Weiping, W., Qiang, C., Qun, L., Zesen, S., and Wei, C.: Indoor-outdoor detection using a smart phone sensor. Sensors (Basel, Switzerland).

5. Zhou, P., Zheng, Y., Li, Z., Li, M., and Shen, G.: Iodetector: A generic service for indoor outdoor detection. In Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems, SenSys ’12, pages 113–126, New York, NY, USA, 2012. ACM.

6. Su, X., Tong, H., and Ji, P.: Activity recognition with smartphone sensors. Tsinghua Science and Technology, 19(3):235–249, June 2014.

7. Ersoy, C., Incel, O., and Kose, M.: O. Durmaz Incel, M. Kose, C. Ersoy, “A Review and Taxonomy of Activity Recognition on Mobile Phones”, BioNanoScience Journal, Special Issue on Personal Health Systems for Well-being and Lifestyle Change, Volume 3, Issue 2, pp. 145-171, 2013. http://dx.doi.org /10.1007/s12668-013-0088-3. 3:145–171, 06 2013.

8. Eagle, N. and (Sandy) Pentland, A.: Reality mining: Sensing complex social systems. Personal Ubiquitous Comput., 10(4):255–268, March 2006.

9. Anderson, I., Maitland, J., Sherwood, S., Barkhuus, L., Chalmers, M., Hall, M., Brown, B., and Muller, H.: Shakra: Tracking and sharing daily activity levels with unaugmented mobile phones. Mobile Networks and Applications, 12(2):185–199, Jun 2007.

CITED LITERATURE (continued)

10. Krumm, J. and Horvitz, E.: Locadio: Inferring motion and location from wi-fi signal strengths. In MobiQuitous, pages 4–13. IEEE Computer Society, 2004.

11. Y Mun, M., Estrin, D., Burke, J., and Hansen, M.: Parsimonious mobility classification using GSM and WiFi traces. 01 2008.

12. Reddy, S., Mun, M., Burke, J., Estrin, D., Hansen, M., and Srivastava, M.: Using mobile phones to determine transportation modes. ACM Trans. Sen. Netw., 6(2):13:1–13:27, March 2010.

13. Ryder, J., Longstaff, B., Reddy, S., and Estrin, D.: Ambulation: A tool for monitoring mobility patterns over time using mobile phones. 4:927–931, 01 2009.

14. Wu, W., Dasgupta, S., Ramirez, E. E., Peterson, C., and Norman, J. G.: Classification accuracies of physical activities using smartphone motion sensors. J Med Internet Res, 14(5):e130, Oct 2012.

15. Anjum, A. and Ilyas, M. U.: Activity recognition using smartphone sensors. In 2013 IEEE 10th Consumer Communications and Networking Conference (CCNC), pages 914–919, Jan 2013.

16. Martín, H., Bernardos, A. M., Iglesias, J., and Casar, J. R.: Activity logging using lightweight classification techniques in mobile devices. Personal Ubiquitous Comput., 17(4):675–695, April 2013.

17. Yong, C., Sudirman, R., Mahmood, N. H., and Chew, K. M.: Human hand movement analysis using principle component analysis classifier. 284-287:3126–3130, 04 2013.

18. Shoaib, M., Bosch, S., Incel, O. D., Scholten, H., and Havinga, P. J. M.: Fusion of smartphone motion sensors for physical activity recognition. Sensors, 14(6):10146–10176, 2014.

19. Shoaib, M., Bosch, S., Incel, O. D., Scholten, H., and Havinga, P. J.: A survey of online activity recognition using mobile phones. Sensors, 15(1):2059–2085, 2015.

20. Liang, Y., Zhou, X., Yu, Z., and Guo, B.: Energy-efficient motion related activity recognition on mobile devices for pervasive healthcare. Mob. Netw. Appl., 19(3):303–317, June 2014.

CITED LITERATURE (continued)

21. Bugdol, M. D., Mitas, A. W., Grzegorzek, M., Meyer, R., and Wilhelm, C.: Human activity recognition using smartphone sensors. In ITIB, 2016.

22. Ronao, C. A. and Cho, S.-B.: Human activity recognition with smartphone sensors using deep learning neural networks. Expert Syst. Appl., 59(C):235–244, October 2016.

23. Hur, T., Bang, J., Kim, D., Banos, O., and Lee, S.: Smartphone location-independent physical activity recognition based on transportation natural vibration analysis. Sensors, 17(4), 2017.

24. Kozina, S., Luštrek, M., and Gams, M.: Dynamic signal segmentation for activity recognition.

25. Liono, J., K Qin, A., and Salim, F.: Optimal time window for temporal segmentation of sensor streams in multi-activity recognition. 11 2016.

26. Okeyo, G., Chen, L., Wang, H., and Sterritt, R.: Dynamic sensor data segmentation for real-time knowledge-driven activity recognition. Pervasive and Mobile Computing, 10:155 – 172, 2014.

27. Wan, J., O'Grady, M. J., and O'Hare, G. M. P.: Dynamic sensor event segmentation for real-time activity recognition in a smart home context. Personal and Ubiquitous Computing, 19(2):287–301, Feb 2015.

28. Krishnan, N. C. and Cook, D. J.: Activity recognition on streaming sensor data. Pervasive and Mobile Computing, 10:138 – 154, 2014.

29. Apple: IOKit framework. `https://developer.apple.com/documentation/iokit`. [Online; accessed 01/17/2018].

30. Chung, J., Donahoe, M., Schmandt, C., Kim, I.-J., Razavai, P., and Wiseman, M.: Indoor location sensing using geo-magnetism. In Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services, MobiSys '11, pages 141–154, New York, NY, USA, 2011. ACM.

31. Apple: Core Motion framework. `https://developer.apple.com/documentation/coremotion`. [Online; accessed 01/18/2018].

CITED LITERATURE (continued)

32. Lee, J.: How sensor fusion works. https://www.allaboutcircuits.com/technical-articles/how-sensor-fusion-works/. [Online; accessed 02/01/2018].

33. VV.AA.: iPhone gyroscope drift. http://iphonedevsdk.com/forum/iphone-sdk-development/53301-gyro-drift.html. [Online; accessed 03/03/2018].

34. VV.AA.: Detect when an iPhone has been bumped. https://stackoverflow.com/questions/6937867/detect-when-an-iphone-has-been-bumped/6939355\#6939355. [Online; accessed 03/03/2018].

35. National Optical Astronomy Observatory: Recommended light levels (illuminance) for outdoor and indoor venues. https://www.noao.edu/education/QLTkit/ACTIVITY_Documents/Safety/LightLevels_outdoor+indoor.pdf. [Online; accessed 03/15/2018].

36. Locher, B., Piquerez, A., Habermacher, M., Ragettli, M., Röösli, M., Brink, M., Cajochen, C., Vienneau, D., Foraster, M., Müller, U., and Wunderli, J. M.: Differences between outdoor and indoor sound levels for open, tilted, and closed windows. International Journal of Environmental Research and Public Health, 15(1), 2018.

37. Prochazka, A.: AudioKit pod. http://audiokit.io/pod/. [Online; accessed 02/02/2018].

38. VV.AA.: Apple motion coprocessors. https://en.wikipedia.org/wiki/Apple_motion_coprocessors. [Online; accessed 03/01/2018].

39. Apple: CMMotionActivity framework. https://developer.apple.com/documentation/coremotion/cmmotionactivity. [Online; accessed 01/18/2018].

40. VV.AA.: The Art of Core Motion in iOS. http://blog.denivip.ru/index.php/2013/07/the-art-of-core-motion-in-ios/?lang=en. [Online; accessed 02/10/2018].

41. Thiemjarus, S., Henpraserttae, A., and Marukatat, S.: A study on instance-based learning with reduced training prototypes for device-context-independent activity recognition on a mobile phone. 2013 IEEE International Conference on Body Sensor Networks, pages 1–6, 2013.

42. Sun, L., Zhang, D., Li, B., Guo, B., and Li, S.: Activity recognition on an accelerometer embedded mobile phone with varying positions and orientations. In Ubiquitous

CITED LITERATURE (continued)

Intelligence and Computing, eds. Z. Yu, R. Liscano, G. Chen, D. Zhang, and X. Zhou, pages 548–562, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

43. Yin, X.: Leveraging Smartphone Sensor Data for Human Activity Recognition. Master's thesis, The University of Western Ontario, London, Ontario, 2016.

44. Apple: CMPedometer sensor. `https://developer.apple.com/documentation/coremotion/cmpedometerdata`. [Online; accessed 03/17/2018].

45. Apple: Core ML framework. `https://developer.apple.com/documentation/coreml`. [Online; accessed 01/20/2018].

46. Patel, S., Shah, V., and Vala, J.: A survey of outlier detection in data mining. 03 2015.

47. Ramaswamy, S., Rastogi, R., and Shim, K.: Efficient algorithms for mining outliers from large data sets. SIGMOD Rec., 29(2):427–438, May 2000.

48. Russell, S. and Norvig, P.: Artificial Intelligence: A Modern Approach. Upper Saddle River, NJ, USA, Prentice Hall Press, 3rd edition, 2009.

49. Minnaar, A.: Deep Learning Basics: Neural Networks, Backpropagation and Stochastic Gradient Descent. `http://alexminnaar.com/`. [Online; accessed 03/03/2018].

50. Han, Z., Hong, M., and Wang, D.: Signal Processing and Networking for Big Data Applications. Cambridge University Press, 2017.

51. Krizhevsky, A., Sutskever, I., and Hinton, G. E.: Imagenet classification with deep convolutional neural networks. In Advances in Neural Information Processing Systems 25, eds. F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, pages 1097–1105. Curran Associates, Inc., 2012.

52. Glorot, X. and Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the thirteenth international conference on artificial intelligence and statistics, pages 249–256, 2010.

53. Ruder, S.: An overview of gradient descent optimization algorithms. CoRR, abs/1609.04747, 2016.

54. Oxford: Definition of overfitting. `https://en.oxforddictionaries.com/definition/overfitting`. [Online; accessed 02/20/2018].

CITED LITERATURE (continued)

55. VV.AA.: 10 times 10-fold crossvalidation. `https://www.openml.org/a/estimation-procedures/9`. [Online; accessed 03/10/2018].

56. Falcon, W. and Schulzrinne, H.: Predicting floor-level for 911 calls with recurrent neural networks and smartphone sensor data. CoRR, abs/1710.11122, 2017.

57. McCall, C., Reddy, K. K., and Shah, M.: Macro-class selection for hierarchical k-nn classification of inertial sensor data. In PECCS, pages 106–114, 2012.

58. Kwapisz, J. R., Weiss, G. M., and Moore, S. A.: Activity recognition using cell phone accelerometers. SIGKDD Explor. Newsl., 12(2):74–82, March 2011.

59. Anguita, D., Ghio, A., Oneto, L., Parra, X., and Reyes-Ortiz, J. L.: A public domain dataset for human activity recognition using smartphones. In ESANN, 2013.

60. Jiang, W. and Yin, Z.: Human activity recognition using wearable sensors by deep convolutional neural networks. In Proceedings of the 23rd ACM International Conference on Multimedia, MM '15, pages 1307–1310, New York, NY, USA, 2015. ACM.

61. Kumar, R. C., Bharadwaj, S. S., Sumukha, B. N., and George, K.: Human activity recognition in cognitive environments using sequential elm. In 2016 Second International Conference on Cognitive Computing and Information Processing (CCIP), pages 1–6, Aug 2016.

62. Murad, A. and Pyun, J.-Y.: Deep recurrent neural networks for human activity recognition. Sensors, 17(11), 2017.

63. Yang, J.-Y., Wang, J.-S., and Chen, Y.-P.: Using acceleration measurements for activity recognition: An effective learning algorithm for constructing neural classifiers. Pattern Recognition Letters, 29(16):2213 – 2220, 2008.

64. Monna, G. C.: MY-AIR Project: Study on Semantic Location and Activity Recognition Algorithms for iOS Systems. Master's thesis, University of Illinois at Chicago, 5 2018. Equivalent of this same thesis for the other institution involved in the TOP-UIC double degree program.

VITA

NAME	Giovanni Clemente Monna
EDUCATION	
	Master of Science in ”Computer Science”, University of Illinois at Chicago, May 2018, USA
	Specialization Degree in ”Computer Engineering, Software”, Jul 2018, Polytechnic of Turin, Italy
	Bachelor’s Degree in ”Computer Engineering”, Jul 2016, Polytechnic of Turin, Italy
LANGUAGE SKILLS	
Italian	Native speaker
English	Full working proficiency
	2016 - IELTS examination (7.0/9)
	A.Y. 2017/18 One Year of study abroad in Chicago, Illinois
	A.Y. 2016/17 Lessons and exams attended exclusively in English
SCHOLARSHIPS	
Fall 2013	Italian scholarship for deserving former High School students
Spring 2017	Italian scholarship for government employees’ children or orphans
Fall 2017	Italian scholarship for the final project (thesis) at UIC
Fall 2017	Italian scholarship for TOP-UIC students
TECHNICAL SKILLS	
Basic level	Unity3D; C#; Matlab
Average level	CMS (ELGG, Magento, Wordpress); Assembly; MySQL; Data Mining; Neural Network study in Keras
Advanced level	Front-end development (HTML, CSS, JavaScript); Back-end development (Python, PHP); Java; Advanced Algorithms and Data Structures (C, ADTs); iOS development (Objective-C, Swift)

VITA (continued)

WORK EXPERIENCE AND PROJECTS

Oct 2014 - January 2016 Software Developer and platform manager at Tometo Lab S.r.l.

2012 NexTer Maths

Help for Math students in High School, iOS app. Retired from App Store for time lack. Last review at https://sensortower.com/ios/us/giancle-monna/app/nexter-maths-formulary/554451051/

2016 Emergency Quest

Help for elderly people with Alzheimer or dementia. http://www.gmonna.com/EQ/

2017 ARMenus

iOS AR application for showing dishes of restaurant menus in augmented reality. http://www.gmonna.com/ARMenus/

2016-2017 Other experiences:

Tutoring area head for IEEE-HKN MuNu Chapter at Turin Polytechnic. http://www.hknpolito.org