Mining smartphone generated data for user action recognition – Preliminary assessment

J. Fijalkowski, M. Ganzha, M. Paprzycki, S. Fidanova, I. Lirkov, C. Badica, and M. Ivanovic

Citation: AIP Conference Proceedings **2025**, 090001 (2018); doi: 10.1063/1.5064928
View online: https://doi.org/10.1063/1.5064928
View Table of Contents: http://aip.scitation.org/toc/apc/2025/1
Published by the American Institute of Physics

AIP | Conference Proceedings
Get 30% off all print proceedings!
Enter Promotion Code PDF30 at checkout

Mining Smartphone Generated Data for User Action Recognition – Preliminary Assessment

J. Fijalkowski[1,7,a)], M. Ganzha[1,2], M. Paprzycki[2,6], S. Fidanova[3], I. Lirkov[3], C. Badica[4] and M. Ivanovic[5]

[1]*Faculty of Mathematics and Information Sciences Warsaw University of Technology ul. Koszykowa 75, 00-662 Warsaw, Poland*
[2]*Systems Research Institute, Polish Academy of Sciences, ul. Newelska 6, 01-447 Warsaw, Poland*
[3]*Institute of Information and Communication Technologies, Bulgarian Academy of Sciences, Acad. G. Bonchev, bl. 25A, 1113 Sofia, Bulgaria*
[4]*Software Engineering Department, Faculty of Automatics, Computers and Electronics, 107 Decebal Blvd., Craiova, RO-200440, Romania*
[5]*University of Novi Sad, Faculty of Sciences, Department of Mathematics and Informatics, Trg D. Obradovica 4, Novi Sad, Serbia*
[6]*Warsaw Management Academy, ul. Kaweczynska 36, 03-772 Warsaw, Poland*
[7]*LeanCode sp. z o.o., ul. Mokotowska 46A/25, 00-543 Warsaw, Poland*

[a)]Corresponding author: jakub@codinginfinity.me

Abstract. Smartphones became everyday “companions” of humans. Almost everyone has a smartphone in their pocket, or bag, and use it on daily basis. Modern smartphones are “loaded” with sensors, providing streams of, potentially useful, data. Simultaneously, staying fit, exercising, running, swimming, etc. became fashionable. In this “climate”, employers can try to incentivise their workers, for instance, to use bicycles to come to work. Here, one of interesting questions becomes: are workers *actually* using bicycles, as declared, or do they try to subvert the system and win prizes, while, for instance, using public transport. One of the ways to check this could be to use data from smartphone sensors to determine the mode of transportation that has been used.

This paper presents preliminary results of an attempt at using raw sensor data and deep learning techniques for transportation mode detection, in real-time, directly on smartphone. The work tries to balance sensor power consumption and computational requirements with prediction correctness and response time. In this context, results of application of recurrent neural networks, as well as more traditional approaches, to a set of actual mobility data, are presented. Furthermore, approaches that leverage domain knowledge, in order to make classifiers more reliable and requiring less processing power (and less energy), are considered.

1 INTRODUCTION

Being “fit” is fashionable, and is pushed upon us from “every direction”. Large number of ad campaigns, related to health, promote fitness and physical activities as a mandatory part of, so-called, “good life”. In this context, increasing number of companies promote being fit among their workers. They do it as physical activities improve health, productivity and well-being, thus making workers more efficient, and less likely to go on a sick leave (which directly corresponds to company’s profit).

This trend resulted in introduction of fitness games that try to make employees more “active”. In such games, competition has to be “frictionless” both for the company (as it would introduce unnecessary operational costs) and for the end-users (otherwise, they will not participate). Moreover, competitions (especially, when there are real prizes involved) cause players to consider cheating as a viable option. Hence need for external, unbiased arbitration, to prevent fraud.

In this context, let us consider corporate game, in which workers are incentivised for using bicycles to come to work. Here, the unbiased arbitration concerns “transportation mode detection” (a case of “user action recognition”).

Application of Mathematics in Technical and Natural Sciences
AIP Conf. Proc. 2025, 090001-1–090001-18; https://doi.org/10.1063/1.5064928
Published by AIP Publishing. 978-0-7354-1745-8/$30.00

Obviously, one of the most natural approaches would be to use data generated by smartphones, which almost everyone has, and that are turned on "all of the time".

The idea of using mobile phone-generated data for user action recognition is not new, and has been examined before. Related works concentrate, mostly, on GPS-based recognition, as the location sensor is easily accessible. Using location data to distinguish user transportation behavior is a relatively reliable option, but has many drawbacks that might make it less than optimal [1, 2]. The key problem is the fact that the GPS sensor is, currently, the most energy demanding one; it can require as much as 10 times the energy as other sensors [2]. This results in substantial battery drain that is unacceptable even on phones with huge batteries (and with recent addition of fast charging), as it would reduce the phone "usability" to a couple of hours. There is also a problem with the availability of location data. GPS sensors requires "clear view of the sky" and it seems impossible to gather reliable data when traveling underneath the ground, under overpasses, when in bus or car or in any other situation with obstructed access to satellites. In such situation, GPS receivers provide very poor accuracy with even worse sampling rate (on the scale of fraction of hertz). This would make recognition very inaccurate and with tremendous lag. Also, with current generation of mobile operating systems, it is the most visible sensor. Both iOS and Android display permanent notification that warns the owner about the app that uses the data. This is an expected behaviour, considering how personal the GPS location can be. However, this means that application that uses GS data would generate constant stream of warning, possibly annoying user and stopping her/him from further participation in the program.

The other trend in the transportation mode detection works is accelerometer-based recognition. It uses a triaxial acceleration sensor that is available in most of the modern smartphones. It measures acceleration of the device, which in turn, is related to the movement of the user, and can be used for action recognition [3]. The main problem with this approach lies in the fact that this sensor is constantly affected by the gravity, making it hard to distill only the movement-related information. The acceleration data is also oriented the same way that the device is, which makes it hard to correctly relate the direction of the force with direction of the movement (*e.g.*, taking phone out of bag or pocket and putting it back differently). The acceleration-based detection, despite the gravity and directionality problems, has many advantages. Accelerometer is "cheap" in terms of energy consumption – Android requires that it consumes less than 4mW and suggests that consumption should be less than 0.5mW, when idle. That is an order of magnitude less than the GPS sensor. Moreover, acceleration data is local and plentiful. The sensor does not require any other "source of truth" (*e.g.*, GPS satellites) and therefore can report measurements much more frequently (up to thousands of times per second). That makes the drawbacks much less problematic, as both the gravity and direction can be estimated from the abundance of data.

In current smartphones, accelerometer is not the only local sensor available – there is also a gyroscope, which measures angular velocity of the device, and a magnetometer that measures the magnetic field the device is in (including Earth's magnetic field). They both are quite energy-efficient and report data with high frequency. The three sensors can be used to solve each other problems. This approach has already been proposed in Yu *et al* [2] and gave promising results.

Here, let us note that most of the works, in the transportation mode detection field, were done before the deep learning gained substantial attention. With the advancements in neural networks and improvements in computer hardware, as even smartphones are getting neural processing units. Hence, use of deep learning seems like a viable option for user action recognition.

In this paper we try to develop a simple yet efficient classification method that leverages recurrent neural network's power in analysing time-series data, combined with vastness of data provided by the three local sensors mentioned earlier. Our aim is to create a method that is a suitable option for both real-time detection of user actions (mainly biking) directly on the device and for off-line classification of previously acquired data (*e.g.*, for fraud prevention). We test different data processing techniques, combined with different recurrent architectures, in order to select the most promising approach. We compare the three sensors in tandem to using only specially-processed accelerometer, which can give insights in the real information carried by these sensors. Although data streams acquired from the sensors model continuous processes, we focus only on a window-based classification that can be used as a building block for future sample-wise segmentation algorithms, and not on the segmentation itself. We also investigate how these approaches work for a simplified version of the problem, namely binary classification. The simplified algorithm might be used as an intermediate for targeted activity tracking applications that do not require fully fledged activity recognition. In our case, due to the nature of the corporate game we are developing, we attempt at distinguishing cycling from everything else.

This paper is structured as follows: Section 2 presents more detailed discussion of related work in the field of transportation mode detection. We follow, in Section 3, with summary of works related to the neural networks, and describe tested classifiers, along with their advantages and drawbacks. Next, in Section 4 we describe the dataset used to test our hypotheses, along with selected (pre)processing methods. In Section 5, we report on tests performed for different classifier/data combinations and give insights on problems with each combination. Finally, in Section 6 we summarize our contributions.

2 RELATED WORK

Let us now look, in more details, into recent developments in the area of user action recognition or, more specifically, *transportation mode detection* (TMD). The biggest IT companies, Google and Apple among them, are trying to address this topic by extending their OSes with developer-accessible APIs for transportation mode detection. TMD is also available in most of current smarthwatches or fitness bands. Unfortunately, it is a closed-source software that works as a black box, without any insights on data and algorithms used.

There are also recent papers that deal with TMD. Here, some works, especially in health-related fields, propose to use custom devices to track user actions [4, 5]. However, majority of approaches focus on use of mobile phones, and can be divided into groups based on the data used:

1. GPS and GIS data;
2. GPS and accelerometer;
3. accelerometer, gyroscope and magnetometer;
4. accelerometer-only;
5. other.

Using GPS data *only* is not a feasible approach, as the only general information that can be obtained (average velocity) is not discriminative enough [6, 7]. This is the reason why GPS data needs to be augmented with either external data source (*e.g.*, GIS-based), or by other data (*e.g.*, from an accelerometer). For instance, in [6] and in [1], it was shown how to use bus stop locations, information about tram and train lines, real-time bus locations or bus lines and current position of the device (traveler), to deliver the needed information. Use of such approaches allows to easily distinguish between different transportation modes (specifically, 82% prediction accuracy between walk, car and bus in [6]; and 92% prediction accuracy between stationary state, walk, bike, car, bus and tram that was reported in [1]). However, this class of approaches needs different models in different cities (as GIS data differ) and as for now was tested only on a limited dataset. Nevertheless, having access to historic GPS data of a user, it is quite easy to extend the models to predict future user locations, as shown by Patterson *et al* [6].

The other approach is to combine GPS data with accelerometer, as reported in [3]. This approach is not that popular, but gives promising results (accuracy of about 93% reported in [3]).

The biggest drawback of GPS sensor is its energy demand [2]. This is closely related to physical limitations of GPS systems, as they need to maintain a connection with multiple satellites to *accurately* report location. Moreover, GPS data seems to be insufficient for transportation modes detection when GIS data is not available or when there is a need to reliably perform TMD with more granularity, or between more transportation modes. Current smartphones are also equipped with accelerometer, gyroscope and magnetometer. These sensors report data local to the device (they do not need external source of information) and can report it with very high frequencies without sacrificing battery lifetime, as shown in [2]. The paper reports how to use these sensors to reliably predict transportation mode (with average accuracy of more than 93%), but also points out that some of the data is redundant (for example, gyroscope can be calculated using accelerometer and magnetometer). This approach (using statistical parameters extracted from raw sensor tracks, grouped using sliding window technique) is also used in [8, 9] with comparable results.

As the three sensors seem to be redundant, many works focus on the accelerometer-only-based approach [4, 10, 11]. The triaxial accelerometer data seems to carry enough information for a reliable mode detection. There are different approaches taken to process it:

1. use magnitude of each sample;
2. estimate gravity and use it to process the data.

The first approach, which is more common in literature, uses magnitude of each sample instead of raw data. The data stream is then divided into windows (using a sliding window technique); next, parameters are being extracted (mostly statistical ones); and each window is classified independently. This approach, used in [4, 10, 11] gives satisfactory results in 5-category classification (90% accuracy and higher), but has been tested only on a limited dataset.

The other approach, presented in [10], uses the fact that acceleration readings are sum of two forces (accelerations): a) gravity, and b) motion. By calculating gravity of a window, one is able to separate the samples into vertical and horizontal parts. This gives ability to distinguish between different motorized transportation modes with a satisfactory result. For instance, Hemminki, Nurmi and Tarkoma [10] achieved accuracy of more than 80% when classifying stationary, walk, bus, train, metro and tram modes.

There are also methods that use GSM, Bluetooth or Wi-Fi data to predict user actions, but they tend to have worse accuracy and require much more processing power. For references on different approaches, check [7].

It is worth noting that most the research activities and results presented here use two stage classification. The first part decides whether the user is moving (it is sometimes called *kinematic classifier*) and, if the user is in motion, the second stage classifies transportation mode.

3 CLASSIFIERS

Majority of approaches, discussed in Section 2, use one of classic machine learning classifiers (*e.g.*, naive Bayes, SVM, random forests). There are few works that use neural networks for the TMD task (for instance, see [8]). With the sequential nature of sensor data used in our work, neural networks (esp. recurrent versions of the networks) are of much interest and are integral part of our work.

Neural networks, with the perceptron model introduced by Rosenblatt [12], are currently one of the go-to models for many machine learning tasks. With the evolution of recurrent neural networks (RNNs) and convolutional neural networks (CNNs), they are one of the most promising models for both classification and regression for sequential data (which also include tasks like generation and transformation of this kind of data). Currently, CNNs are not used for sequences, but with the introduction of undecimated fully convolutional neural network (UFCNN; see, [13]), they might be used for TMD (but are out of scope of this work). On the other hand, RNNs are used in a wide variety of sequential tasks, like speech recognition [14, 15, 16] and even music generation [17]. These, combined with current advancements in mobile phone processors, make them ideal candidates for real-time user activity recognition.

Sensor data has sequential nature and often contain temporal dependencies. Recurrent modeling allows the network to classify each timestep separately, using information about the past. It remembers temporal structure of the data and draws conclusions from it. Networks made from simple recurrent neurons, can detect these dependencies, but do that only on very short sequences (tens of timesteps) and *forget* distant (in the sense of timesteps) values [18, 19]. This might be a problem in transportation mode detection as, in our case, single timestep is only about 0.02 s while, from the observations of normal user actions, we have learned that transportation mode changes on the scale of seconds and not milliseconds.

This problem is strictly related to the learning algorithms used to teach the model [18, 20], as they rely on complete gradient calculations and are prone to exploding and vanishing gradients. To fix this problem, Hochreiter and Schmidhuber [19] introduced the Long Short-Term Memory (LSTM) cells, where a single cell is a composition of multiple neurons that are connected in such a way as to prevent the gradient problems and to allow the network to *learn* when to forget. LSTM cells were successfully used in the previously mentioned works about RNN [14, 15, 16, 17]. This makes them viable candidates for user action classification tasks.

The LSTM cells are quite complicated and need substantial amount of processing power to calculate results (both training and inferring processes are thus “expensive”). Cho *et al* recently introduced Gated Recurrent Units (GRU; in [21]) that should have similar properties as LSTM cells, but they require less resources to compute. The comparison between GRU, LSTM and simple recurrent cells can be found in [22].

Recurrent networks, in the simple form, can only use data from **before** currently processed timestep. This is a major limitation, as most of the classes depends both on data before and after the point. This certainly is the case in transportation mode detection, because changes in mode are somewhat rare and most of the time the whole window of data represents the same class. To solve this problem, it is possible to use bidirectional versions of recurrent neural networks, as presented in [23]. It gives the network ability to “look into the future” and make decisions based on data that is after the current sample. It improves network computational power without requiring substantial increase in size.

Currently, the most popular (but not the only one) algorithm used to train recurrent networks is Backpropagation Through Time (BPTT), an extension to the backpropagation algorithm introduced by Rumelhart, Hinton, and Williams [24]. To make gradient effects less problematic, we decided to use truncated version of BPTT that operate on data of constant length.

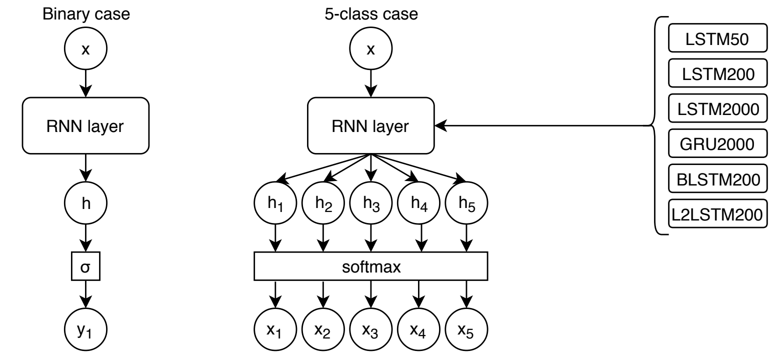

FIGURE 1. Overview of network architecture used in the paper

As one of key contributions of our work, we have decided to test all of so far mentioned recurrent architectures and compare them on dataset that will be described in Section 4. We decided to test LSTM networks thoroughly, assuming that the GRU networks give comparable results (we used them only in some of the tests). We also checked whether using bidirectional version of the LSTM cell of the same size (meaning that we have roughly double the number of neurons in a single cell, compared to the unidirectional version) does improve the classification. RNNs can be stacked, so we tested whether stacking does make difference in this task (this can be directly compared to bidirectional version, as it uses the same number of neurons). We performed both binary (non-bike vs. bike) and multiclass (5 classes – stationary, walk, run, bike, motorized) classification (for a complete description, see next section). Our intention was to do the classification mostly in the recurrent layers, so we used only the smallest possible dense layer to get class probability (meaning single neuron with sigmoid activation for binary classification, and 5-neuron dense layer with softmax activation for 5-class prediction). The network architecture is depicted in Figure 1.

Overall, after thoroughly assessing the set of possibilities, we settled on the following versions of recurrent networks (nevertheless, we plan to substantially increase the number of networks versions when applying to the data that is currently being collected):

1. single LSTM cell of size 50 (abbrev. LSTM50);
2. single LSTM cell of size 200 (abbrev. LSTM200);
3. single LSTM cell of size 2000 (abbrev. LSTM2000);
4. single GRU cell of size 2000 (abbrev. GRU2000) [1];
5. single bidirectional LSTM cell of size 200, with concatenated output (abbrev. BLSTM200);
6. two LSTM200 layers, stacked one on another (abbrev. L2LSTM200).

All of the architectures (except GRU2000) were used both in binary and 5-class classification. Using different sizes of the LSTM layer was to give us insights into how tough the problem is for the RNNs.

4 DATASETS AND DATA PROCESSING

Every year, new and better hardware is installed in mobile phones. Among others, it includes many sensors that constantly gather data about devices' surroundings and users' actions. In this section, we present some of the sensors available in smartphones and select the ones that are important for the TMD. Then, we describe dataset, developed as part of [2], that is used in this paper. Finally we present different data transformations that might give insights into real value of the data and improve, or simplify, TMD algorithms.

[1] Used only for binary classification as they frequently diverged during training.

4.1 Available Sensors

Current smartphones contain many different sensors that are available to developers. Android documentation lists 13 different physical and virtual *sensors*, but it considers only a fraction of sensors actually available in existing devices. Things like GPS, camera, GSM and Wi-Fi antennas are called differently but, for our work, should be treated as sensors, as they enhance sensing capabilities of a phone (moreover, they can be, and were, used in TMD). Availability of sensors varies between mobile phone models, but it is safe to assume that even low-end smartphones have all of the sensors listed in this section. Sensors can be divided into different categories, for instance:

1. motion sensors,
2. environmental sensors,
3. position sensors,
4. location sensors,
5. communication sensors,

and others. This categorization is not a "definitive one", as most sensors can be classified differently based on their usage. For our work, we consider hardware sensors, from the first four categories, as they are easily available and are directly related to the TMD.

Accelerometer; measures acceleration of the device in three different axes – X, Y and Z – that are aligned with phone orientation. It is commonly used to detect orientation of the device, in relation to gravity. The device is exposed to constant gravitational force that is being measured with the movement-related acceleration, making it hard to separate the two accelerations. This is the main limitation of the sensor, but can be turned into advantage when one considers the amount of data that can be gathered (as accelerometer can have very high reporting frequency). Its unit is $\mathrm{m} \cdot \mathrm{s}^{-2}$. This is one of the three sensors considered directly in this work.

Gyroscope; measures angular velocity of the phone, in the three basic axes (oriented as in the accelerometer). The sensor is used to control movement in games. Because of the physical limitations of the device, its zero level is not exactly at 0, but at some unknown value that may change over time. This problem is known as drift, or bias. Drift quickly accumulates when integrating and must be compensated in software. Its unit is $\mathrm{rad} \cdot \mathrm{s}^{-1}$. This is the second sensor considered in our experiments.

Magnetometer (last sensor used in our research); measures the strength and direction (in three basic axes) of the magnetic field that the device is in. It tries to measure the Earth's magnetic field, but is easily skewed by variable fields produced by multiple devices in its surroundings. Its unit is μT.

GPS/GNSS; uses satellites[2] to measure the location of the device, which in turn can be used to estimate device's velocity. This sensor has been commonly used in TMD (see Section 2). Because of the high energy requirements and low reporting frequency, it has been omitted.

Microphone; records the sound. It can be used to aid the transportation mode detection, as the mode strictly relates to the surrounding (*e.g.*, sound of the engine). Its unit is dB. Because constant recording of the sound is considered a privacy violation and is quite hard to process, it is not included in this work.

GSM and Wi-Fi; two communication sensors that are used to wirelessly transmit data from and to the phone. The strength of the signal may change based on users' location and therefore monitoring it might give insights into transportation mode of the user. As noted previously, there are works that use these sensors for exactly that purpose, but because of poor results (compared to other methods), they are not used in our current research.

In the following sections, we use $\mathbf{A}_t, \mathbf{G}_t, \mathbf{M}_t$ as notation for, respectively, accelerometer, gyroscope and magnetometer sample in a single recording session (also called *track*). Throughout the text, bold font denotes a 3-component (x, y, z) vector. Here, t is the index of the sample (timestamp of the sample) in the track. We assume that every sensor reports events at approximately the same time. Throughout the paper, we mostly operate on a single track, so introducing separate notation for cross-track samples is not needed.

The main problem of each selected sensor, is that their measurements are captured in the **phone's coordinate system**. This means the data depends on phone arrangement. In principle, that makes the classification a more difficult task than it really is, as this information is not discriminative enough. The problem with phone orientation is twofold – we do not know the exact location and position of the smartphone, therefore we cannot infer anything based on an exact sample. Also, orientation can change during the recording session, which makes the reasoning even more difficult. To mitigate the problem, we have decided to sacrifice directionality and reduce each sample to its magnitude:

$$A_t = ||\mathbf{A}_t||, G_t = ||\mathbf{G}_t||, M_t = ||\mathbf{M}_t||$$

[2] Currently there are at least three different GNSS systems available to most smartphones – GPS, GLONASS and (still in development) Galileo.

TABLE 1. Amount of data in each category.

	Still	**Walk**	**Run**	**Bike**	**Motorized**
Before	82h (142, 11.5M)	97h (420, 14.0M)	45h (169, 6.2M)	58h (142, 9.2M)	175h (409, 28.1M)
After	49h (97, 8.3M)	57h (352, 9.8M)	27h (115, 4.7M)	46h (121, 7.9M)	156h (388, 26.8M)
After*	50h (98, 8.4M)	61h (368, 10.5M)	28h (117, 4.7M)	46h (121, 7.9M)	159h (391, 27.4M)

In parentheses there is a number of tracks followed by a number of samples (in millions).
* Accelerometer-only

In the reminder of the paper we mostly use the un-directed magnitude, with a few exceptions (as explicitly noted, when applicable).

4.2 Dataset

In our experiments, we use an HTC dataset, obtained from the paper [2]. This dataset consists of raw signal logs of the three sensors – accelerometer, gyroscope and magnetometer – gathered with contemporary Android smartphones by more than 200 participants. Here, note that results presented below have been used to develop an actual application, which is currently being used in Warsaw, Poland (by a substantially larger number of participants). In the near future we will report on results obtained using "our" dataset. The actual HTC data is split into two sets – training and testing – but we don't use this particular split, and (pre)process the data as a whole (splitting only afterwards). The original set contains data concerning 10 different transportation modes: a) still, b) walk, c) run, d) bike, e) motorcycle' f) car, g) bus, h) metro, i) train, and j) High Speed Railway (HSR) that are further divided based on location of the device (*e.g.*, pocket, bag, in-hand, in-stand). The dataset is also divided into different continuous tracks (each consisting of three signal streams) that represent a single recording session. We deliberately abandon the device positioning information to make classification more general. We follow the general TMD trend and consider the latter six modes as a single, "motorized", category (further called **motorized** or **vehicle** category). This decision is particularly relevant to the context of "bicycle use" (*vs.* all other transportation modes) recognition.

Since the data is a raw log of sensor events, it needs preprocessing to be usable by, among others, neural network classifiers. During the preprocessing phase we join the motorized classes into one and abandon pose/location/set information. We have also found out that each HTC data track (and each signal) has slightly different sampling frequency. It is common to have tracks with sampling frequency as high as 200Hz and also tracks that are sampled with a frequency less than 10 Hz. There are also differences between sensors, *e.g.*, accelerometer sampling might be much higher than, for example, magnetometer. To simplify the situation (for the initial set of experiments, reported here) we have decided to filter-out signal streams that are sampled with frequency less than 45Hz and re-sample those that are over 55Hz to about 50Hz. We did not filter-out whole tracks because in some tests, we use only the accelerometer and it tends to be sampled with frequency higher than 50Hz. We summarize the amount of data before and after preprocessing in Table 1.

4.3 Data Transformations

Because of the classification methods used in this paper, using whole tracks (of different lengths) is not a viable way to train classifiers. Having thousands of timesteps (samples) in a single track would prevent the training algorithm (described in Section 3) from teaching the network anything meaningful. It would require too much computing power and would make the calculations extremely fragile to exploding and vanishing gradients (see, also [18]). It also seems that classifying each timestep would be a waste of processing power, as $\frac{1}{50}$ of a second does not contain any significant information (from the perspective of the application we are interested in).

4.3.1 Sliding windows

Therefore, we have decided to apply the well-known concept of a sliding windows [2, 8]. We split a single track into fragments of equal length that overlap each other (in our case – by half). This solves computational requirements and gradient problems of the BPTT, and gives us ability to classify windows and not samples. We decided to test three window lengths:

1. 250 samples (5s);
2. 500 samples (10s);

3. 1000 samples 20s).

This selection is a compromise between fast classification (the classifier can give answer in about 5s since the start of the data stream) that does not require much processing power (250 samples results in substantially less input neurons than 1000 case), and a case that needs much more time to both gather data and process it (1000 samples needs about 20s of the data stream). The 500 sample windows is the middle ground between the other two.

There is also a possibility to split the window further into *frames*. These frames are just a way to group multiple neighboring samples into a single timestep. This approach has been used in [8], but our aim is to test the algorithms with the most general approach, so we treat each and every sample in the window as a separate timestep.

When discussing windows, we use

$$A_t^{(j)}, G_t^{(j)}, M_t^{(j)}$$

to distinguish between full-track and window-track sample. Windows $j-1$ and j, as well as j and $j+1$ overlap.

4.3.2 Gravity

As stated earlier, an accelerometer sample is a sum of two accelerations – gravity and motion. Having them combined makes the data more difficult to interpret (*zero* is at 1 g and not at real 0 $\mathrm{m \cdot s^{-2}}$). Because we use a high frequency sampling of the accelerometer, we tried to estimate gravity. Then, we could use the gravity vector to split the raw sample into horizontal and vertical movements. This idea was first introduced by Mizell [25], but was used in slightly different context. Work reported in [10] used a more sophisticated algorithm to estimate gravity, but because it introduces additional parameters that need to be tweaked, it has not been used in our work.

We adopted the idea from [25] that uses an ordinary mean as a gravity estimator. The main assumption here is that noise and acceleration patterns are not correlated, but this assumption might not be true in long acceleration windows (e.g. in a car). This is the reason why [10] introduced a separate algorithm to deal with this data type. From our observations, one can safely ignore parts of the tracks where the assumption is strictly not true. This means that the gravity Y might be calculated using:

$$\mathbf{Y}^{(i)} = \left(Y_x^{(i)}, Y_y^{(i)}, Y_z^{(i)}\right)$$

$$Y_a^{(i)} = \frac{1}{N}\sum_{t=0}^{N-1} A_a^{(i)}(t), a \in \{x, y, z\}$$

where a is one of the axes.

Having gravity estimated, we separate each sample, in the window, into two separate parts – vertical and horizontal – that are parallel and perpendicular to gravity, respectively. This can be done using simple vector projection onto the gravity vector, as proposed by Mizell [25]:

$$\mathbf{d}^{(i)}(t) = \mathbf{A}^{(i)}(t) - \mathbf{Y}^{(i)}$$

$$\mathbf{V}^{(i)}(t) = \left(\frac{\mathbf{d}^{(i)}(t) \cdot \mathbf{Y}^{(i)}}{\mathbf{Y}^{(i)} \cdot \mathbf{Y}^{(i)}}\right)\mathbf{Y}^{(i)}$$

$$\mathbf{H}^{(i)}(t) = \mathbf{d}^{(i)}(t) - \mathbf{V}^{(i)}(t)$$

where, the “dot operator” • denotes scalar (dot) product of two vectors.

The problem here is that only the vertical vector has a known orientation (parallel to gravity), as it is the result of projection. We can also calculate direction (up/down) of the component using simple dot product. The horizontal part is still oriented the same way as the device. Unfortunately, it seems impossible to reorient it without using sophisticated processing and additional data sources (which we want to avoid). The only meaning the horizontal part has is the magnitude of changes. We can further compute magnitudes of these vectors, preserving the up/down direction of vertical vector in the sign of magnitude:

$$V^{(i)}(t) = \frac{1}{||\mathbf{Y}^{(i)}||}\left(\mathbf{V}^{(i)}(t) \cdot \mathbf{Y}^{(i)}\right)$$

$$H^{(i)}(t) = ||\mathbf{H}^{(i)}(t)||$$

. This way we reduce the three-component vector of raw acceleration that represents the overall force affecting the device, into pair of two separate forces that better reflect real movement of the user.

4.3.3 Parameters

Most of current works in TMD use some combination of window-based parameters, like statistical parameters (mean, variance, standard deviation, *etc.*), frequency- (DFT coefficients, *etc.*) and time-based (integral, double integral, mean crossing rate, *etc.*) parameters. These tend to represent the whole spectrum of information that the data window carries, but abandon the property of time (they aggregate everything into single or multiple numbers). Because we selected recurrent neural networks as the classification algorithm, we must transform the data so that the time dimension is preserved.

As stated earlier, it is possible to aggregate multiple samples into frames. That would allow calculating the same range of parameters per single frame, and use them in every timestep. Because we do not use frames, and operate on single sample level, the only transformation that we apply sample-wise is the magnitude calculation (or magnitude with direction for vertical component). This preserves the time domain and moves the burden of feature extraction to the training phase.

We decided to test different combinations of data (because we do have multiple streams available), as our aim is to validate the claim whether accelerometer data is sufficient for neural networks to perform correct mode of transportation classification. Therefore, we decided to use:

1. magnitude of accelerometer, gyroscope and magnetometer (three floats for every timestep): (A_t, G_t, M_t) (abbrev. **MA**);
2. magnitude of accelerometer only (single float for every timestep): (A_t) (abbrev. **AC**);
3. horizontal and vertical components of accelerometer, but using magnitude for horizontal part and feeding vertical part as-is (four floats for every timestep): $(\mathbf{V}_t, H_t)$ (abbrev. **DR**);
4. horizontal and vertical components of accelerometer, using magnitudes for both of them (two floats for every timestep): (V_t, H_t) (abbrev. **DM**).

Combined with the different windows sizes, this gave us total of 12 different data combinations. With the networks used (6 for binary classification, 5 for the 5-class classification), this gave us 132 different experiments (72 for binary and 60 for 5-class classification) that are thoroughly described in the next section.

4.3.4 Noise in the data

It is considered a good practice to remove noise from the data (e.g. by using low pass filter that leaves only 90% of signal energy), as it cancels sensor jittering. In this paper, we deliberately skip this step and keep the noise in the data. The reason for this is our decision to use neural networks. They are thought to be quite resistant to noise in data and even benefit from it. When data is noisy, networks tend to generalize better, and are not that susceptible to overfitting. The only problem is with test data, as having test set with noise makes the noise effectively part of the meaningful data (although it preserves some noise-related attributes). We believe that for this problem neural networks should be able to correctly recognize noise, and thus we decided not to perform any filtering, even on the test set. Here, note that a comprehensive analysis of role of noise in the data has been presented in [26].

4.3.5 Training and test set

After division and processing phases, we equalize data so that every category has roughly the same amount of samples. This is done for every category by randomly selecting N samples, where N is the number of samples in the least numerous category, provided that the category consists of more than $1.2N$ samples. This makes the classes that have almost equal number of samples, without sacrificing too much data.

Having data equalized, we split it into training and testing (validation) sets by randomly selecting 80% of samples for training and leaving the rest for testing. This process is done separately for every combination of window size and selected data (resulting in 12/24 different splits). We did not test the networks on different splits, as our initial experiments showed that they behave exactly the same on different combinations of data and it is not necessary to do k-fold validation. Information about data sets used in experiments is presented in Tables 2 and 3.

5 EXPERIMENTS AND RESULTS

Let us recall that the main goal of our research was to conduct experiments that would provide a comprehensive overview of performance of recurrent neural networks applied to the transportation mode detection task. Summarizing decisions concerning the experimental setup, we have settled on:

TABLE 2. Final dataset split for 5-class classification

	Still	**Walk**	**Run**	**Bicycle**	**Motorized**
250	36k/9k (50h/12h)	36k/9k (50h/12h)	30k/7k (41h/10h)	36k/9k (49h/13h)	36k/9k (50h/12h)
500	18k/4k (49h/12h)	18k/5k (49h/13h)	15k/4k (41h/10h)	18k/4k (49h/12h)	18k/4k (49h/12h)
1000	9k/2k (49h/12h)	9k/2k (48h/13h)	7k/2k (41h/10h)	9k/2k (49h/12h)	9k/2k (49h/12h)
250*	44k/11k (61h/15h)	44k/11k (61h/15h)	36k/9k (51h/13h)	44k/11k (61h/15h)	44k/11k (61h/15h)
500*	22k/5k (60h/15h)	22k/5k (61h/15h)	18k/5k (50h/13h)	22k/6k (60h/15h)	22k/5k (61h/15h)
1000*	11k/3k (60h/15h)	11k/3k (60h/15h)	9k/2k (49h/13h)	11k/3k (60h/15h)	11k/3k (60h/15h)

The data is given separately for data transformations that rely on all of the sensors (first three rows) or require only accelerometer (latter rows), as the amount differs. We also show amount of data for different window sizes as it differs in sample count. Each set is described as: a) approximate number of samples in training set (in thousands), b) approximate number of samples in test set (in thousands), c) approximate length of training set, and d) approximate length of test set.The number may vary between each split as it was done randomly and independently.

* Accelerometer-only.

TABLE 3. Final dataset split for binary classification

	Non-bike	**Bike**
250	67k/17k (92h/23h)	55k/14k (77h/19h)
500	33k/8k (92h/23h)	28k/7k (77h/19h)
1000	16k/4k (91h/23h)	14k/3k (77h/19h)
250*	60k/15k (84h/21h)	50k/13k (70h/17h)
500*	30k/7k (83h/21h)	25k/6k (69h/17h)
1000*	15k/4k (83h/21h)	12k/3k (69h/17h)

The data described here is the same as in table 2.

* Accelerometer-only.

1. three different window lengths – 250, 500 and 1000 samples each;
2. selection of four different parameters (data types): magnitude of all sensors (**MA**); magnitude of accelerometer (**AC**); decomposed accelerometer, using raw vertical samples (**DR**); and decomposed accelerometer, with magnitudes of both vectors (**DM**);
3. six different network topologies: LSTM with memory cell of size 50 (LSTM50); 200 (LSTM200); 2000 (LSTM2000); GRU with memory cell of size 2000 (GRU2000)[3]; bidirectional LSTM of size 200 (BLSTM200); and two LSTM200 stacked one on another (L2LSTM200).

Every experiment has been conducted both in binary (non-bike and bike) and 5-class classification (still, walk, run, bike, motorized). Each network has been trained three times, using exactly the same data but with different initial weights of the network (to check how stable the training process is). For both LSTM and GRU cells, we used cells with initially zeroed bias (with the exception of bias on the forget gate that was initialized to **1**), random orthogonal matrix [27] for recurrent layer and Glorot uniform initializer [28] for the kernel (input) layer. The dense layer, used to get the class probability, was initialized using the Glorot uniform initializer and had initially zeroed bias. During training, we used dropout with a rate of 0.5 between the recurrent and dense layers. The dense layer had either sigmoid (binary classification) or softmax (5-class classification) activation. The recurrent layers were using default activations provided by Keras [29] (*i.e.*, *tanh* for output activation and (hard) sigmoid for recurrent step activation). For the bidirectional layer, we concatenated the outputs of both forward and backward layers and treated these as an output of the layer.

For the training process we used Adam optimizer [30] (the AMSGrad variant from [31]) with initial learning rate set to 0.001 and with parameters $\beta_1 = 0.9, \beta_2 = 0.999$ and zero decay. We reduced the learning rate by 10, when for 10 epochs the accuracy did not improve. We used binary accuracy as a metric for binary classification and categorical accuracy for 5-class classification. Each network has been trained for 40 epochs[4].

We used Keras [29] library with the Tensorflow [32] backend, and trained the networks on a system with three NVIDIA Tesla P100 cards, using batch size of 256 samples for LSTM2000 and GRU2000 and 1024 for other network types. Note that the training time was not one of parameters that, at this stage, we were particularly interested in. Hence, the information about the system hardware has been provided only for completeness.

[3] This network was tested only in binary classification as it tended to diverge during training.

[4] The training process stabilized after about 20 epochs in most cases.

TABLE 4. The summary of the conducted experiments.

		LSTM50	**LSTM200**	**LSTM2000**	**GRU2000**	**BLSTM200**	**L2LSTM200**
DR	**250**	3/3	3/3	3/3	3/0	3/3	3/3
	500	3/3	3/3	3/3	*/0	3/3	3/3
	1000	3/3	3/3	3/3	*/0	3/3	3/3
DM	**250**	3/3	3/3	3/3	3/0	3/3	3/3
	500	3/3	3/3	3/3	3/0	3/3	3/3
	1000	3/3	3/3	3/3	*/0	3/3	3/3
AC	**250**	3/3	3/3	3/3	3/0	3/3	3/3
	500	3/3	3/3	3/3	*/0	3/3	3/3
	1000	3/3	3/3	3/3	*/0	3/3	3/3
MA	**250**	3/3	3/3	3/3	3/0	3/3	3/3
	500	3/3	3/3	3/3	3/0	3/3	3/3
	1000	3/3	3/3	3/3	*/0	3/3	3/3

X/Y means that we trained X networks for binary classification and Y networks for 5-class classification.

TABLE 5. The best accuracy achieved by every network for binary classification.

		LSTM50	**LSTM200**	**LSTM2000**	**GRU2000**	**BLSTM200**	**L2LSTM200**
DR	**250**	0.944	0.951	0.868	0.959	0.956	0.953
	500	0.889	0.890	0.835	0.000	0.904	0.953
	1000	0.837	0.864	0.906	0.714	0.890	0.903
DM	**250**	0.941	0.950	0.961	0.965	0.952	0.952
	500	0.901	0.905	0.911	0.908	0.914	0.951
	1000	0.861	0.884	0.755	0.763	0.924	0.896
AC	**250**	0.926	0.921	0.853	0.769	0.924	0.929
	500	0.878	0.890	0.850	0.926	0.916	0.928
	1000	0.839	0.884	0.875	*0.573*	0.844	0.903
MA	**250**	0.948	0.964	**0.969**	0.931	0.967	0.967
	500	0.936	0.961	0.942	0.949	0.961	0.964
	1000	0.859	0.894	0.863	0.860	0.923	0.881

In Table 4 we summarize number of trained networks for each configuration of data. GRU2000 layer tends to diverge and we decided to skip the training for the 5-class case. We were also not able to get 3 stable training sessions for some of the binary cases. If the table contains "*" instead of number, it means that at least one of the training session diverged.

Because of the amount of data we have gathered (number of combinations of parameters that we have experimented with), it is impossible to describe every result in details. Instead, we summarize the most significant / interesting findings / conclusions. Hence, we describe the performance of networks when dealing with different window sizes, followed by comparison of network types. Next, we analyze the results from the data point of view and, finally, summarize all of them. Figures 2 to 8 present the training sessions that achieved the best test accuracy.

The best results for binary classification are shown in table 5. The best results for 5-class classification are shown in Table 6.

5.1 Comparison by Window Size and Network Type

The most substantial performance differences occurred for different window sizes. Our experiments have shown that *the bigger the window size, the worse the accuracy*. While some network types performed reasonably well even for the 1000 sample windows (most notably L2LSTM200 and BLSTM200), yet they struggled to achieve as good performance as for the 250 case. Moreover, the windows of 1000 samples were hard to process for the networks with most potential, namely LSTM2000 and GRU2000. GRU2000 networks were tough to train for any window size (they diverged quite rapidly), but even LSTM2000 diverged once (for binary classification of magnitude of accelerometer for window of 1000 samples). Interestingly, the training process is very stable for 250 windows and it seems it could

TABLE 6. The best accuracy achieved by every network for 5-class classification.

		LSTM50	**LSTM200**	**LSTM2000**	**BLSTM200**	**L2LSTM200**
DR	**250**	0.845	0.892	0.932	0.903	0.914
	500	0.769	0.855	0.904	0.880	0.900
	1000	0.754	0.762	0.793	0.771	0.871
DM	**250**	0.841	0.890	0.919	0.897	0.899
	500	0.813	0.865	0.829	0.838	0.865
	1000	0.762	0.766	*0.705*	0.776	0.837
AC	**250**	0.843	0.866	0.893	0.875	0.877
	500	0.806	0.845	0.795	0.816	0.844
	1000	0.778	0.792	0.792	0.742	0.827
MA	**250**	0.895	0.929	**0.953**	0.935	0.930
	500	0.862	0.909	0.896	0.911	0.912
	1000	0.825	0.833	0.883	0.894	0.904

achieve even better performance after more epochs. This is strictly not true for bigger windows – for window of 1000 samples, the training set accuracy is able to differ (in both directions) by more than 0.05 epoch-to-epoch (for test set, it may differ by more than 0.2!). This confirms the previously stated hypotheses that window of 1000 samples is too big and would make the training process unstable.

The results of the training for binary classification of MA data type are shown in Figures 2 and 3. The differences in accuracy, between different window sizes, are mostly the same for different data types. Figure 4 shows that GRU2000 diverged even for window of 500 samples. The stable learning for the 5-class classification has been achieved for small window sizes, as shown in Figure 2.

Figures 2-5 show general trends in network performance when comparing them by network type. Observed trends are mostly the same for every window size (except that, again, the performance is worse for bigger sizes) so, from here on, we consider only windows of 250 samples (unless stated otherwise).

When comparing network types, the difference between binary and 5-class classification is substantial, so we will describe these results separately.

For binary classification, especially for windows of 250 samples, all tested network gave comparable results. Each network type has been able to achieve the accuracy of at least 70% but, on average, the accuracy was greater than 90%. For small windows, the GRU2000 network was learning either really fast (like in Figure 5), or was not able to learn at all (this occurred for AC data type, where it stagnated at about 70% after 5th epoch). This did not occur for LSTM2000, because even if it was not able to rapidly achieve high accuracy, it did not stagnate.

The parameter selection did not matter too much. For every data type, every network (except the above mentioned outliers) was able to achieve at least 90%. The best results were reached by LSTM2000, but even LSTM50 did manage to classify the test set with accuracy of more than 90% (even 95% for MA data type). Moreover, most of the networks achieved 90% accuracy after less than 10 epochs.

The most stable results, across different data, were achieved by the smallest networks, meaning LSTM50, LSTM200, BLSTM200 and L2LSTM200. The two-layer and bidirectional variants were able to reach slightly higher accuracies, but the difference was negligible (less than 1%). The fastest-learning network was GRU2000, but it did so only sporadically.

For 5-class classification, the results were slightly different. The LSTM2000 has been better than any other network, but only for windows of 250 samples. For wider windows, it struggled behind other network types (even LSTM50 in a few cases). On the other hand, LSTM50, except a few cases, was noticeably worse than any other network (the difference was sometimes more than 10%). When dealing with bigger window sizes, LSTM200, BLSTM200 and L2LSTM200 were the most stable ones (and most of the time, one of them was the clear winner), but for smaller window sizes they were not able to catch up to LSTM2000. This can be seen in Figures 3 and 4. There, most of the networks reached the accuracy of about 90% in every case, with the exception of LSTM50 that mostly performed no better than 84%, which still is quite good result.

Performed tests show that, as expected, the network architecture does matter. It seems reasonable to conclude that having a more sophisticated architecture would make the training a more stable and predictable process. It seems like it would also improve the classifier accuracy, without much increase in processing power requirements. We plan to test further architectures in our future work (in particular, for real-world data sets).

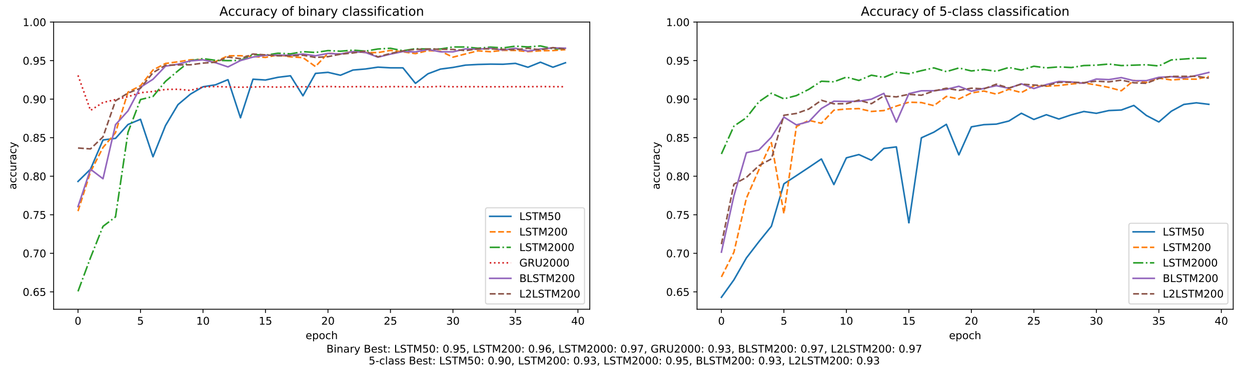

FIGURE 2. Training results for binary/5-class classification of MA, 250 window

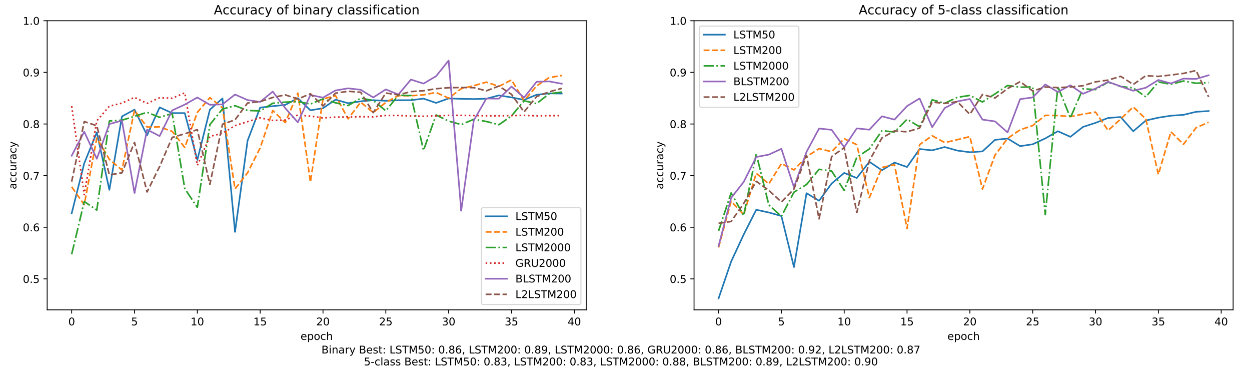

FIGURE 3. Training results for binary/5-class classification of MA, 1000 window

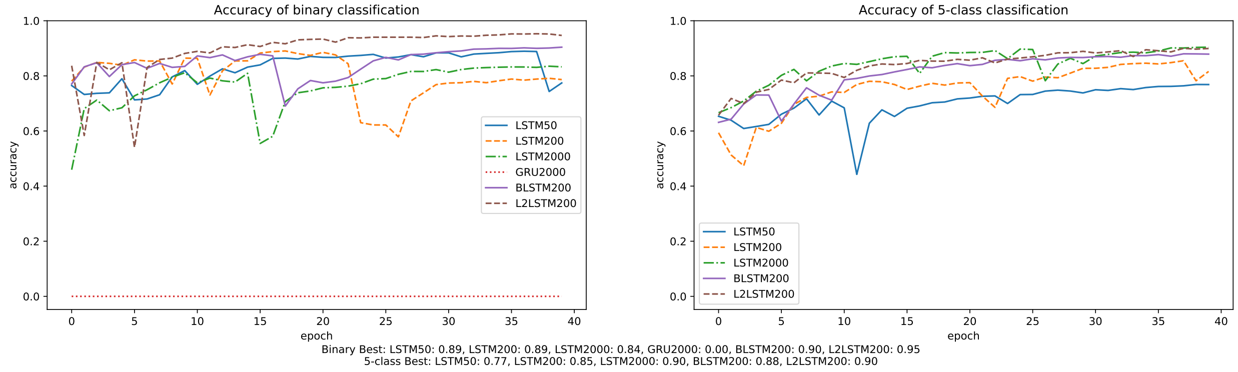

FIGURE 4. Training results for binary/5-class classification of DR, 500 window

Even though there were noticeable differences between network types, we cannot select a single architecture that is the most suitable for the TMD task. The problem lies in the power requirements of each network. LSTM2000 achieved the best accuracy, but can not be used directly on the mobile phone because it would drain battery and would require substantial time to compute, even on high-end smartphones. On the other hand, LSTM50 seems to be too small for off-line processing, where the computational power is not limited. Hence, the desired approach depends on the system being developed and must be selected case-by-case. Furthermore, for the corporate game application (as the one mentioned above) it may be worthy exploring scenario in which different architectures are to be used on the

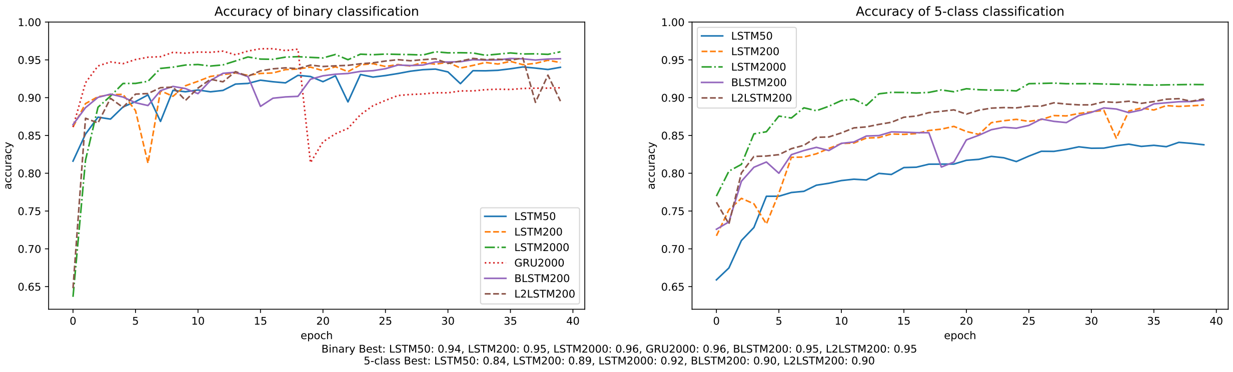

FIGURE 5. Training results for binary/5-class classification of DM, 250 window

smartphone and on the server.

5.2 Comparison of Different Data Types

We have tested four different sets of parameters extracted from the data. Our initial intuition was that the MA data type will give the best results, as it uses three different sensors and therefore contains a lot more information than other parameters. We did not know whether the decomposition of the accelerometer will yield any meaningful differences, yet we supposed so, as it uses domain knowledge to augment the data.

After thorough testing, it seems that our intuitions were correct. The MA data type did achieve the best accuracy for any type of the network and this is consistent for both the binary and 5-class cases classification. This can be seen in Figures 6, 7 and 8. The MA data type is the clear winner in all of the three training sessions, while the AC data type lags behind. The most astounding thing is the binary case with the LSTM2000 network – the MA and DM are on par, yet the latter one has only two thirds the information of the MA when comparing by size, and even less when comparing by meaning. It also seems strange that LSTM2000 performs so well on DM yet fails on DR, while other networks are able to overcome the directionality and perform equally well on both DR and DM. This partly confirms our hypotheses that directionality in data does not help much.

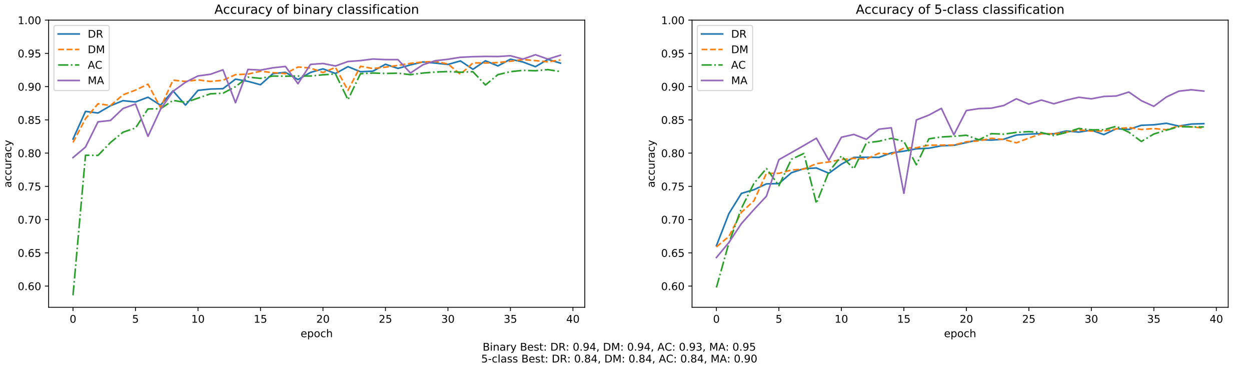

FIGURE 6. Comparison of results for binary/5-class classification with LSTM50, 250 sample window

It is worth noting that there were hardly any differences between accuracy on training and test sets, as shown in Figure 9 (the networks behave exactly the same when trained on different splits, or different parameters). We deliberately have split the data separately for every combination of parameters/window size to check whether the HTC dataset is big enough for neural networks. It seems like the data is astonishingly good, because the test set accuracy mostly follows the training set accuracy (and the network did not use test set during training), but shows symptoms of homogeneity. From this follows also that k-fold validation was not required when testing on this dataset.

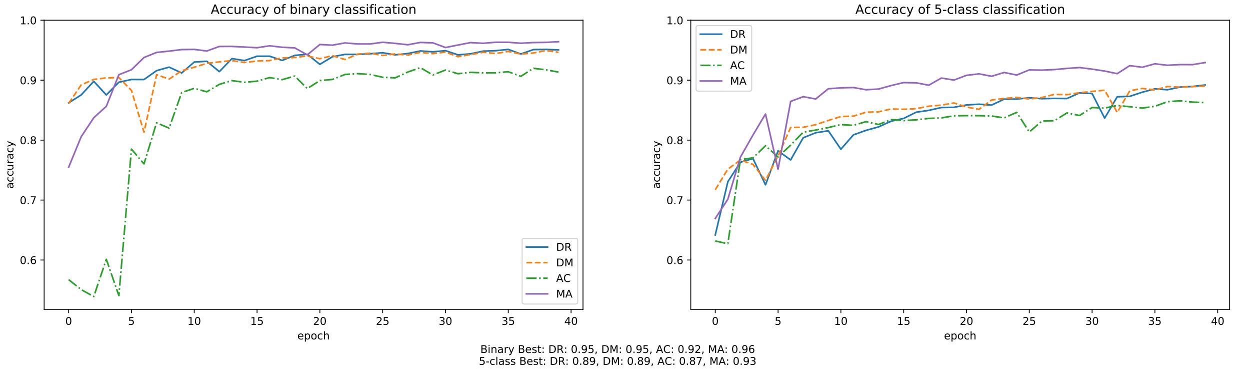

FIGURE 7. Comparison of results for binary/5-class classification with LSTM200, 250 sample window

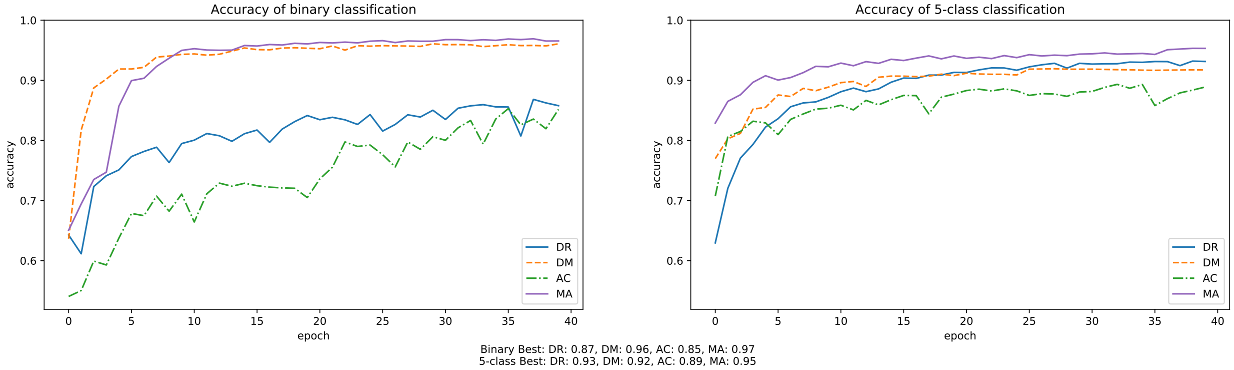

FIGURE 8. Comparison of results for binary/5-class classification with LSTM2000, 250 sample window

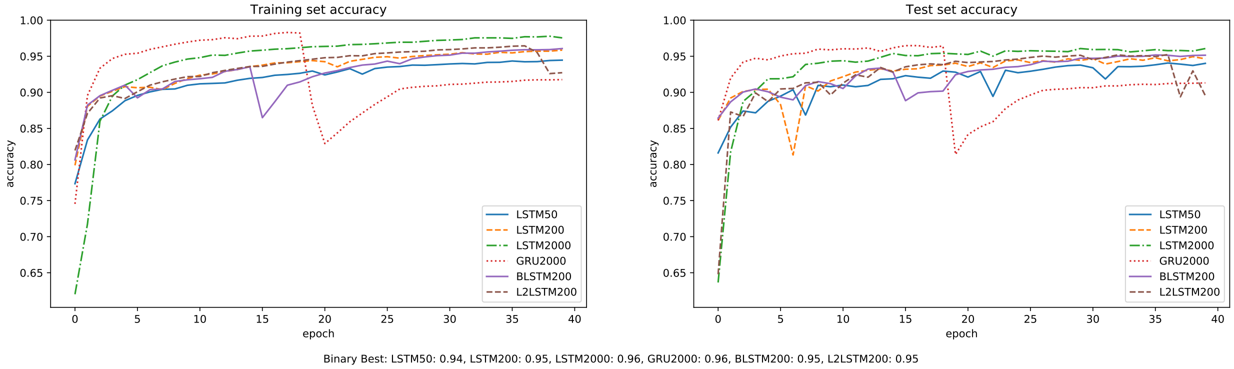

FIGURE 9. Training results for binary classification of DM, 250 window

6 CONCLUSIONS AND FUTURE WORK

The aim of this work was to consider how recent neural network based approaches can be used for transportation mode detection, on the basis of sensor data available from smartphones. In this context, we have described basic sensors that one can use for the TMD. Further, we have gathered current state of the art in TMD and presented a way to adapt recurrent neural networks to this task. We have tested multiple network types, that have been trained with different sets of parameters, to select the most promising ones. We have checked different window sizes and concluded that accumulation of data makes the task harder or even undecidable, as long windows hinder the training process. We have compared different sets of parameters that carry different amount of information and we have shown that accelerometer can be used as the sole data source, without substantial decrease in the accuracy. We have tested different network architectures and selected the ones that can be used for on-line and off-line processing. This gives the general framework for developing more sophisticated algorithms that can be used both on mobile devices and on more powerful servers.

We achieved accuracy of 95.3% for classification of five transportation modes using the LSTM2000 network with three sensors and showed how to reduce the collected data (sensor count) by three, yet still achieving 93.2% accuracy. The same has been achieved for binary classification – LSTM2000 with all of the sensors achieved accuracy of 96.9% and GRU2000 reached 96.5% for accelerometer only. We have also shown that we can sacrifice network size for a small loss in accuracy, which may be useful in case of online processing. It has been shown that, while all three sensors are carrying substantial information, the accelerometer might suffice (and requires much less power to process). We have empirically proven that directionality of the sensor data is only an obstacle for classification.

There are multiple possible improvements / further research directions to be explored in the future. The most promising extensions that we plan to pursue are:

1. Reduce the window size to decrease the likelihood of multimodal tracks even more. Currently, 250 samples per window means that we classify 5 seconds of user movement at once and during that time user might change the means of transport (*e.g.*, stop at the stoplights).
2. Train the networks so that every timestep is classified. We tested the networks so that we have only a single response per window, yet RNNs are able to classify each timestep. This opens up a number of possibilities in aggregating the data and substantially reduces the lag introduced by the network (i.e. the action might be taken on every timestep and not on every 250 steps).
3. Test how the UFCNNs [13] perform in this task and compare their resource footprint to recurrent networks.
4. Test networks on independently gathered dataset, as different people have different styles of movement (*e.g.*, they walk or cycle uniquely), while it seems that the HTC dataset is uniform in this regard.

Our current aim is to test the developed algorithms with a much bigger dataset, that is currently being collected by a system working in Warsaw, Poland. The application is a biking game which aims to encourage people to use bikes, instead of cars or public transport, when commuting (esp. to work). It incentivises employees to compete against each other by ranking them (by some score that relates to the number of bike rides to and from workplace) and awarding them with prizes after every time-limited contest. The results from this paper were used to prepare automatic tracking solution (so that users do not have to remember to turn the app on) and fraud-prevention algorithms. We believe that the new dataset will be much more diverse (at least for some of the transportation modes) and is going to capture the differences in the movement style of different people. We expect that soon we will be able to report on the obtained results.

ACKNOWLEDGEMENT

Work presented in this paper was supported in part by PAS-RAS bilateral project “Semantic foundation of the Internet of Things”, PAS-BAS bilateral project “Practical aspects of scientific computing”, as well as a collaboration agreement between University of Novi Sad, University of Craiova, SRI PAS and Warsaw University of Technology.

REFERENCES

[1] L. Stenneth, O. Wolfson, P. S. Yu, and B. Xu, “Transportation mode detection using mobile phones and GIS information,” in *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, GIS ’11* (ACM, New York, NY, USA, 2011), pp. 54–63.

[2] M.-C. Yu, T. Yu, S.-C. Wang, C.-J. Lin, and E. Y. Chang, “Big data small footprint: The design of a low-power classifier for detecting transportation modes,” in *Proceedings of the VLDB Endowment*, Vol. 7 (VLDB Endowment, 2014), pp. 1429–1440.

[3] S. Reddy, M. Mun, J. Burke, D. Estrin, M. Hansen, and M. Srivastava (2010) Using mobile phones to determine transportation modes, *ACM Transactions on Sensor Networks* **6**, 13, 27p.

[4] M. Shafique and E. Hato (2015) Use of acceleration data for transportation mode prediction, *Transportation* **42**, 163–188.

[5] F. Attal, S. Mohammed, M. Dedabrishvili, F. Chamroukhi, L. Oukhellou, and Y. Amirat (2015) Physical human activity recognition using wearable sensors, *Sensors* **15**, 31314–31338.

[6] D. J. Patterson, L. Liao, D. Fox, and H. Kautz, "Inferring high-level behavior from low-level sensors," in *UbiComp 2003: Ubiquitous Computing*, edited by A. K. Dey, A. Schmidt, and J. F. McCarthy (Springer, Berlin-Heidelberg, 2003), pp. 73–89.

[7] M. Nikolic and M. Bierlaire, "Review of transportation mode detection approaches based on smartphone data," in *Proceedings of the 17th Swiss Transport Research Conference* (2017).

[8] T. H. Vu, L. Dung, and J.-C.Wang, "Transportation mode detection on mobile devices using recurrent nets," in *Proceedings of the 2016 ACM on Multimedia Conference, MM'16* (ACM, New York, NY, USA, 2016), pp. 392–396.

[9] T. Sonderen, "Detection of transportation mode solely using smartphones," (2016).

[10] S. Hemminki, P. Nurmi, and S. Tarkoma, "Accelerometer-based transportation mode detection on smartphones," in *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems, SenSys'13* (ACM, 2013), paper 13, 14p.

[11] P. Siirtola and J. Röning (2012) Recognizing human activities user-independently on smartphones based on accelerometer data, *International Journal of Interactive Multimedia and Artificial Intelligence* **1**, 38–45.

[12] F. Rosenblatt (1958) The perceptron: a probabilistic model for information storage and organization in the brain, *Psychological Review* **65**, 386–408.

[13] R. Mittelman, "Time-series modeling with undecimated fully convolutional neural networks," `arXiv:1508.00317`, 2015.

[14] A. Graves, A. Mohamed, and G. E. Hinton, "Speech recognition with deep recurrent neural networks," `CoRRabs/1303.5778`, 2013, `arXiv:1303.5778`.

[15] A. Graves, "Sequence transduction with recurrent neural networks," `CoRRabs/1211.3711`, 2012, `arXiv:1211.3711`.

[16] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks," in *Proceedings of the 23rd International Conference on Machine Learning, ICML'06* (ACM, New York, NY, USA, 2006), pp. 369–376.

[17] D. Eck and J. Schmidhuber, "Finding temporal structure in music: blues improvisation with LSTM recurrent networks," in *Proceedings of the 12th IEEE Workshop on Neural Networks for Signal Processing*, edited by H. Boulard (Martigny, Valais, Switzerland, 2002), pp. 747–756.

[18] Y. Bengio, P. Simard, and P. Frasconi (1994) Learning long-term dependencies with gradient descent is difficult, *IEEE Transactions on Neural Networks* **5**, 157–166.

[19] S. Hochreiter and J. Schmidhuber (1997) Long short-term memory, *Neural Computation* **9**, 1735–1780.

[20] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," in *A Field Guide to Dynamical Recurrent Networks*, edited by J. F. Kolen and S. C. Kremer (Wiley-IEEE Press, 2001).

[21] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," `CoRRabs/1406.1078`, 2014, `arXiv:1406.1078`.

[22] J. Chung, Ç. Gülçehre,, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," `CoRRabs/1412.3555`, 2014, `arXiv:1412.3555`.

[23] M. Schuster and K. Paliwal (1997) Bidirectional recurrent neural networks, *IEEE Transactions on Signal Processing* **45**, 2673–2681.

[24] D. E. Rumelhart, G. E. Hinton, and R. J.Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1, edited by D. E. Rumelhart, J. L. McClelland, and C. PDP Research Group (MIT Press, Cambridge, MA, USA, 1986), pp. 318–362.

[25] D. Mizell, "Using gravity to estimate accelerometer orientation," in *Proceedings of the 7th IEEE International Symposium on Wearable Computers, ISWC'03* (IEEE Computer Society,Washington, DC, USA, 2003), pp. 252–253.

[26] K.-C. Jim, C. L. Giles, and B. G. Horne (1996) An analysis of noise in recurrent neural networks: convergence and generalization, *IEEE Transactions on Neural Networks* **7**, 1424–1438.

[27] A. M. Saxe, J. L. McClelland, and S. Ganguli, "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks," `CoRRabs/1312.6120`, 2013, `arXiv:1312.6120`.

[28] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, Proceedings of Machine Learning Research*, Vol. 9, edited by Y.W. Teh and M. Titterington (PMLR, Chia Laguna Resort, Sardinia, Italy, 2010), pp. 249–256.

[29] F. Chollet *et al*, Keras, 2015, `https://keras.io`.

[30] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," `CoRRabs/1412.6980`, 2014, `arXiv:1412.6980`.

[31] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of Adam and beyond," in *International Conference on Learning Representations* (2018).

[32] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems, 2015, software available from `tensorflow.org`.