Resource Consumption Analysis of Online Activity Recognition on Mobile Phones and Smartwatches

Muhammad Shoaib*, Ozlem Durmaz Incel†, Hans Scolten*, Paul Havinga*
*Pervasive Systems Research Group, University of Twente, Enschede, the Netherlands
{m.shoaib,hans.scholten,p.j.m.havinga}@utwente.nl
†Department of Computer Engineering, Galatasaray University, Istanbul/Turkey
{odincel}@gsu.edu.tr

***Abstract*—Most of the studies on human activity recognition using smartphones and smartwatches are performed in an offline manner. In such studies, collected data is analyzed in machine learning tools with less focus on the resource consumption of these devices for running an activity recognition system. In this paper, we analyze the resource consumption of human activity recognition on both smartphones and smartwatches, considering six different classifiers, three different sensors, different sampling rates and window sizes. We study the CPU, memory and battery usage with different parameters, where the smartphone is used to recognize seven physical activities and the smartwatch is used to recognize smoking activity. As a result of this analysis, we report that classification function takes a very small amount of CPU time out of total app's CPU time while sensing and feature calculation consume most of it. When an additional sensor is used besides an accelerometer, such as gyroscope, CPU usage increases significantly. Analysis results also show that increasing the window size reduces the resource consumption more than reducing the sampling rate. As a final remark, we observe that a more complex model using only the accelerometer is a better option than using a simple model with both accelerometer and gyroscope when resource usage is to be reduced.**

Keywords*—*Activity recognition, mobile sensors, performance analysis.

I. Introduction

In most of the studies on human activity recognition using the sensors available on smart devices, such as smart phones and smartwatches, a typical methodology is followed: sensor data is collected from device users, if needed data is pre-processed, features are extracted, and classification model is built and its performance is tested using machine learning algorithms on tools, such as Weka [1]. This type of offline analysis of sensor data is termed as offline human activity recognition because the analysis is not performed on the device. Recently, researchers have been moving towards online activity recognition in order to verify the offline results and to analyze the resource consumption of machine learning algorithms on these devices [1]. In online activity recognition systems, the classification of various activities is performed on the device (smartphone or a smartwatch) in real-time. There are a few studies [2]–[6] where the authors have implemented various classifiers on mobile phones and smartwatches. However, in these studies, only a few classifiers are implemented. Moreover, they have used different data sets, data features, platforms, and implementations. They have also used different methods to compute resource consumption. Due to these different experimental setups, it is difficult to compare various aspects of a human activity recognition system in terms of their resource usage when running on mobile and wearable devices. Examples of such aspects include classifiers, feature extraction, sensors, sensor sampling rate, and window size for segmentation.

In order to compare various classifiers for their resource consumption in a similar environment, we developed an online activity recognition framework for both mobile phones and smartwatches. This Android-based framework was presented in [7]. On the framework, various classifiers, data features, sampling rates, and sensors can be evaluated for their resource consumption and recognition performance. In this specific study, we evaluate the resource consumption of six commonly used classifiers on both smartphones and smartwatches. We performed these evaluations for different sensors and their combinations. We summarize our contributions as follows:

- We analyze six commonly used classifiers, naive Bayes, multi-layer perceptron, decision tree, random forest, support vector machine and k-nearest neighbour, for their power (in milliwatt and in percentages using battery-stats), CPU (CPU-time, CPU-time/call, etc.) and memory usage (total and model size).
- We analyze the resource usage with using accelerometer alone, as well as in combination with a gyroscope. We also test the resource consumption with varying sensor sampling rates, and varying window sizes.
- We test the resource usage performance of the classifiers both on a mobile phone trained for recognizing seven physical activities, and on a smart watch trained for recognizing smoking, both in real-time.

This work is supported by Dutch National Program COMMIT in the context of the SWELL project P7 and Galatasaray University Research Fund under Grant Number 15.401.004, by Tubitak with Grant Agreement Number 113E271.

The rest of the paper is organized as follows: In Section II, we present the related work. We describe our methodology in Section III and in Section IV, we present the results of the performance evaluations. Finally, we report the conclusions in Section V.

II. Related Work

Human activity recognition using smartphone sensors has been studied extensively for the last few years [1], [8], [9]. Most of the work in this area is performed offline such that collected data is analyzed in machine learning tools such as WEKA, Scikit-learn, R, and MATLAB. Recently, researchers have been moving towards online activity recognition in order to verify the offline results and to analyze the resource consumption of machine learning algorithms on mobile phones and other wearable devices, such as smartwatches [1], [3], [5], [6], [10]–[12].

In a previous survey paper [1], we reviewed the studies that implement activity recognition systems on mobile phones. Though there are a few studies where the authors have implemented various classifiers on mobile devices, there is still room for improvements. For example, in these studies, only a few classifiers are implemented on different platforms such as Android, IoS, Debian Linux, and Symbian as shown in Table 4 in [1]. Moreover, they have used different data sets, different data features, different experimental setups, and different implementations as presented in Table 1, Table 3, Table 5, Table 12 in [1]. Not all of the studies include an analysis of resource consumption and and those who present an analysis, either focus on one resource, such as battery or use different methods to compute resource consumption. For example, for the battery consumption, two types of measurements are made. In one case, the amount of time a battery lasted was reported, while in another, the power usage was reported in watt-hours per hour. We think that, watt-hour per hour is a better option since it is independent of the battery capacity. The CPU usage was reported in terms of percentages during which the CPU was occupied by the recognition process and memory used was usually reported in MBs (megabytes).

Activity recognition using smartwatch sensors is still relatively new, compared to smartphones. Most of the work using smartwatch sensors is still being done offline [13]–[15]. Only in very recent studies [16]–[18], battery consumption of sensor logging and online activity recognition process on smart watches has been analyzed. However, similar to the studies on mobile phones, different setups and use of different methods make it difficult to compare the results.

Overall, due to these different experimental setups, it is challenging to compare various aspects of a human activity recognition system, including classifiers, feature extraction, sensors, sensor sampling rate, and window size for segmentation, in terms of their resource usage. In order to address this challenge, we explore the parameter space in detail and focus on different metrics in terms of resource usage.

III. Method for Resource Consumption Analysis

The activity recognition process can be divided into various components, such as sensing, feature extraction, training, and classification. This process can be divided into offline and online categories. In online activity recognition, the classification is done on the device (phone or wearable device). However, the training can still be done in two ways: online (on the device) and offline. The training can be very time and resource consuming, that is why it is usually done offline. We have opted for offline training in this work. We use offline training and then port these trained models to the mobile phone and the smartwatch.

We proposed a conceptual framework in [7] for online activity recognition which consists of three main components: Activity recognition (AR) process on a smartphone, AR process on a smartwatch, and a machine learning tool (WEKA) for training models. This framework proposes different modes of operation, such as running the activity recognition process only on phone, only on watch or on both devices. In this paper, we only test the resource consumption of separate modes, only on phone and only on watch. Hence, all the mentioned steps of activity recognition, such as sensing, feature extraction, classification are running on both these devices. However, the training is performed offline on the WEKA tool, where machine learning models are trained offline and then ported to these devices. After training these models, they are serialized in WEKA and stored in the relevant Android Apps where they are de-serialized at the time of their use. This process is described in WEKA documentation [19].

For sensing, we have analyzed the resource usage of classifiers with an accelerometer, in combination with gyroscope and with linear acceleration. For feature extraction, simple features, min, max, mean and standard deviation, are used. Other features can be added on demand. For the classification part, the trained models from WEKA are used to predict the current window of sensor data and map it to the relevant activity. These trained models can be placed in the asset or other folders in our app and they are ready to use. These three modules or parts are implemented as an Android service which runs in the background and it does not need any user interaction.

In our specific use case, we run physical activity recognition process on the phone and smoking recognition process on the watch. For training purpose, we used two data sets, one for the smartphone [20] and one for the smartwatch [21]. We added an additional data of around 5 hours to the smoking data set in order to improve the null or other class, so smoking should not be confused with other activities. This additional data includes various activities such as drinking, eating, walking, biking, washing dishes, cooking, taking part in conversations, inactive (sitting, standing, laying in bed etc.) and others. On both devices, we used a sampling rate of 50 for sensor readings and a window size of 5 seconds for feature extraction.

We trained six classifiers in WEKA: decision tree (DT),

support vector machine (SMO), random forest (RF), multilayer perceptron (MLP), naive Bayes (NB), K-nearest neighbor (KNN). We use these classifiers in their default settings except few changes. For example, KNN was used with 3 neighbors instead of its default value 1. We use an odd number to avoid tie in voting. For the random forest, we used two variants: with 9 trees (RF9) and with 99 trees (RF99).

We used different tools to measure the resource usage. For CPU usage of individual method calls, we used *Android Device Monitor with Trace-View*. For CPU usage of our app, we used *top* and *DUMPSYS cpuinfo* tools. For power consumption, we used an Android application: *POWERTUTOR* and also built-in battery stats. For memory usage, we used *DUMPSYS meminfo* tool. We briefly describe these tools here:

- *Android Device Monitor with Trace-View [22]*: Using this tool, we can track individual methods or functions (such as classification) in our applications for CPU consumption. It is achieved by recording each method's entry and exit point for its CPU consumption. The CPU time reported using this tool should not be taken as an absolute time taken by these function but it is rather used for a comparative purpose which is what we used it for. According to Android documentation [22]: *Don't try to generate absolute timings from the profiler results (such as, "function X takes 2.5 seconds to run"). The times are only useful in relation to other profile output, so you can see if changes have made the code faster or slower relative to a previous profiling run.*
- *DUMPSYS cpuinfo [23]:* It gives detailed information about CPU usage of various running processing both in user and kernel space. It reports the CPU usage per single CPU. We run this command every 3 seconds to record the CPU usage of our app. We take the average of these reported values at the end of each recording session. Each recording session was of ten minutes.
- *TOP [24]*: This command also reports the CPU usage of all running processes. It reports the CPU usage per a single CPU. We ran this command every second for each session where each session lasted for ten minutes. We take an average of these values at the end of each session.
- *POWERTUTOR [25]:* This app measures power in watts per process. Its power model was built on specific phones (HTC G1, HTC G2, and Nexus one), and its results on other phones may be rough [25]. However, it can still report accurate results for relative comparisons because any bias in reported results will be the same for all our measurements. It reports average power usage for the amount of time this app is running.
- *Android Built-in Battery statistics:* This tool tells about the percentage of used battery by running applications.
- *DUMPSYS meminfo [23]*: For memory usage, we used *DUMPSYS meminfo*. It gives us memory use (PSS). We ran this command every second and took an average at the end of each session which lasted ten minutes.

IV. Performance Evaluation

For resource consumption analysis, we used three metrics: CPU usage, memory usage, and power consumption. CPU usage is the percentage of time spent by our app on CPU. Memory usage is measured in terms of PSS (proportional set size). Proportional Set Size is defined as the amount of main memory (RAM) used by a process which contains both the private memory of that process as well as the part which is shared by that process with other processes. For example, if this process shares a specific portion of RAM with one other process, then half of that portion will be counted towards its PSS value. For power consumption, we used Watt as a measure and it is defined as one joule per second. We ran our app for a specific amount of time (ten minutes), repeatedly, and measured all these metrics. We used Samsung Galaxy S2 and LG Watch R in our evaluations.

A. CPU and Memory Usage on Smart Phone

First, we evaluated all six classifiers using accelerometer alone for their resource consumption. Random Forest classifier was used in two variants (with 9 and 99 trees) to see the effect of increasing the number of trees on resource consumption. For a benchmark application, we use *music player* as it is commonly used on smartphones. The results for CPU, memory, and power consumption of our app running these classifiers and the benchmark application are given in Table I(a). These results are the average of multiple recording sessions where each recording session lasts for around ten minutes. The classification function takes a small percentage of the CPU compared to feature extraction and sensing, therefore the difference between various classifiers is not clearly visible in the resource consumption of our app as is shown in Table I(a). There is one exception: KNN classifier. The resource consumption of KNN is higher than all other classifiers, in terms of CPU and power usage. It is higher because KNN searches through the whole training set to find the nearest neighbors which is an expensive task in terms of computations. We can also see that RF99 consumes slightly more CPU than RF9 due to its larger number of trees. As shown in Table I(a), the power consumption shows similar trends as CPU consumption. As for as the memory usage is concerned, Random forest with 99 trees consumes the highest memory. However, overall the memory usage of all these different versions of our app is not very high and shows that memory usage is not a problem for these apps on smartphones and smartwatches. We also present the size of each trained model in Table I(a) and it can be seen that it is very small for all these models.

To zoom in at CPU consumption of each classifier, we used method-tracing with the Trace-View tool. Using this tool, we looked at the CPU time of classification function for each classifier. These measurements were repeated multiple times. The results for zoomed in CPU usage for various classifiers using accelerometer are shown in Table I(b). From this table, we can see a clear difference between the CPU usage of these classifiers. For example,

Table I. RESOURCE CONSUMPTION ANALYSIS OF OUR APP ON PHONE USING ACCELEROMETER

	NB	*MLP*	*DT*	*RF9*	*RF99*	*SMO*	*KNN3*	*Music player as benchmark application*
CPU Usage	4.91%	4.60%	4.59%	4.59%	4.77%	4.69%	7.10%	13.00%
Memory Usage (MB)	10.91	11.30	11.03	11.73	15.19	11.18	12.04	16.00
Power using power tutor(mW)	16.00	16.00	16.00	16.00	17.00	16.00	23.00	45.00
Power using battery stats	9.00%	9.00%	9.00%	9.00%	9.00%	9.00%	10.50%	NA
Model Size (KB)	13	440	36	219	2327	19	412	NA

(a) Average CPU usage, memory usage, and power consumption on smartphone

	CPUtime %	CPUtime	*CPUtime/call*	total calls	total CPUtime	total wall-clock time
NB	***2.80***	1928.95	***26.79***	72	68925.00	360000
MLP	***1.90***	1276.56	***17.73***	72	68392.00	360000
DT	***0.30***	219.25	***3.04***	72	66768.00	360000
RF9	***0.40***	269.39	***3.74***	72	67089.00	360000
RF99	***4.30***	3114.79	***43.26***	72	71668.00	360000
SMO	***1.50***	1003.00	***13.94***	72	68975.00	360000
KNN3	***75.00***	54526.00	***757.30***	72	72232.00	360000

(b) CPU time (milliseconds) of classification function only on smartphone

Table II. RESOURCE CONSUMPTION ANALYSIS OF OUR APP ON SMARTWATCH USING ACCELEROMETER

	NB	**MLP**	**DT**	**RF9**	**RF99**	**SMO**	**KNN3**
CPU	3.67%	3.41%	3.55%	3.48%	3.58%	3.49%	4.65%
Memory (MB)	11.81	12.42	12.41	13.97	28.09	12.27	13.47
Model Size (KB)	13	947	137	947	10397	19	919

(a) Average CPU Usage, and Memory Consumption on Smartwatch

	CPUtime %	CPUtime	*CPUtime/call*	total calls	toal CPUtime	total wall-clock time
NB	***4.90***	2876.00	***57.53***	50	58693.88	252000
MLP	***2.50***	1446.00	***28.92***	50	57840.00	252000
DT	***0.90***	509.00	***10.18***	50	56555.56	252000
RF9	***0.90***	528.00	***10.58***	50	58666.67	252000
RF99	***8.70***	5308.00	***106.17***	50	61011.49	252000
SMO	***2.00***	1118.00	***22.36***	50	55900.00	252000
KNN3	***72.00***	51706.00	***3693.00***	14	71813.89	252000

(b) CPU time (milliseconds) of classification function on Smartwatch

KNN takes a large amount of time compared to all other classifiers. We see that decision tree and random forest with nine trees are light-weight classifiers in terms of prediction.

B. CPU and Memory Usage on Smart Watch

We also evaluated all these scenarios on the smartwatch and observed similar results for classifier comparisons. For smartwatch, the CPU usage, memory usage, and model size of our wearable app running various classifiers are given in Table II(a). We also looked at the CPU usage of the individual classification method for each classifier and observed similar results as on the phone. For example, KNN was taking the highest amount of CPU time among classifiers whereas decision tree and RF9 was taking the smallest fraction of CPU time. These zoomed in CPU times and their percentages are given in Table II(b).

C. CPU Usage with Different Sensors

We also evaluated these classifiers using the combination of the accelerometer and the gyroscope. A comparison of CPU usage using an accelerometer and its combination with the gyroscope are shown in Figure 1. There is a significant increase in CPU usage due to the addition of the gyroscope.

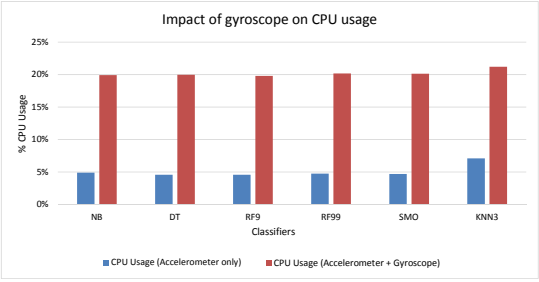
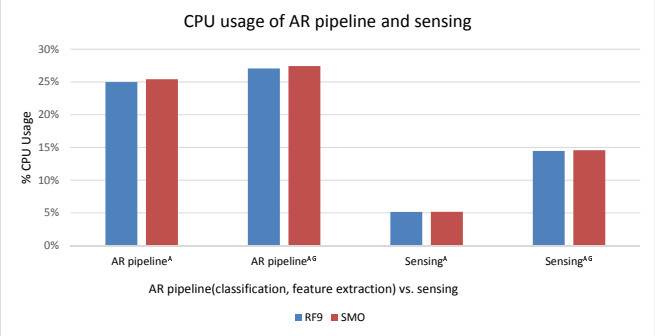


Figure 1. Impact of adding gyroscope (G) with accelerometer (A) on CPU consumption

To see this effect in detail, we zoomed in at the sensing part for accelerometer and gyroscope. We tested this scenario with two classifiers such as support vector machine, and random forest (RF9) out of the six classifiers. These results are shown in Figures 2(a) and 2(b). It can be seen from these figures that the CPU consumption increases significantly, mainly because of the sensing and feature extraction part since the CPU needs to process all changes in gyroscope values and make it available for reading by the Android App. Although we use a sampling rate of 50 samples per second, sensor values are

made available by Android at higher sampling rate over which we do not have any control. Any time gyroscope value changes, *onStatusChanged()* function is called where these sensors values are stored so users can read them according to their own sampling rate. Moreover, the CPU needs to process 16 additional features for the gyroscope. Therefore, we see a higher increase in CPU usage due to sensing compared to the prediction function. Previously we observed that there is a very small increase in CPU usage when we increase the number of trees in a random forest classifier from 9 (CPU usage:4.47%) to 99 (CPU usage:4.61%) as shown Table I(a). It shows that it is better to use a complex training model with accelerometer instead of using a simple model with both accelerometer and gyroscope if the former can provide an acceptable recognition performance.



(a) %CPU time of AR pipeline vs. sensing using accelerometer (A) and its combination with gyroscope (AG)

	CPUTIME_Classification		CPUTIME_Sensing		TOTAL APP CPUTIME		TOTAL APP RUN TIME
	Acc	*Acc + Gyro*	Acc	*Acc + Gyro*	Acc	*Acc + Gyro*	
RF	27902.15	*30217.44*	18574.83	*52061.79*	68212.00	*111641.33*	360000
SMO	28851.93	*31141.51*	18583.68	*52482.64*	68913.67	*113516.67*	360000

(b) CPU time (milliseconds) using accelerometer, and its combination with gyroscope

Figure 2. Impact of adding gyroscope (G) with accelerometer (A) on CPU consumption of prediction and sensing

We also measured the CPU usage of the linear acceleration sensor with respect to an accelerometer to see its resource consumption. For random forest classifier, the average CPU-usage of the linear acceleration sensor was 5.11% whereas that of the accelerometer was 4.49%. We observe a higher CPU usage for the linear acceleration sensor because there is additional processing required to get its values from the accelerometer by removing gravity.

D. Impact of Sampling Rate and Window Size

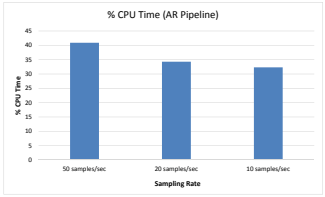
The power consumption of our app can be optimized in many ways. For example, a larger window size can reduce the CPU usage, thereby leading to less battery usage. Similarly, a lower sampling rate can also help in reducing the battery usage. However, these should not compromise the recognition performance. In order to see how these two parameters affect the CPU usage, we evaluated our app with three different window sizes (10, 5, 2 seconds) and three sampling rates (50, 20, 10). We measure the CPU usage of our app on the phone as well as we measure CPU time of the activity recognition pipeline (AR Pipeline with SMO). In our case, this pipeline means storing sensor values in a data structure after every sampling interval, extracting features after each window, and prediction using the trained classifier. These results are shown in Figure 3 for both varying window sizes and sampling rates. We see that CPU usage decreases with increasing window sizes and lower sampling rates. The impact of window size is higher than that of sampling rate on the CPU usage in terms of reduction.

	Win: 2 sec	**Win: 5 sec**	**Win: 10 sec**
CPU TIME (AR Pipline)	39804.13	23638.2	17953.61
% CPU Time (AR Pipeline)	55.31%	40.9%	34.3%
%CPU Usage (APP)	5.44%	4.69%	4.49%

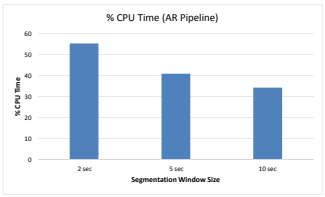
(a) CPU time (milliseconds) for varying window sizes: AR means Activity Recognition

	SR: 50	**SR: 20**	**SR: 10**
CPU TIME (AR Pipline)	23638.2	15139.36	12872.96
% CPU Time (AR Pipeline)	40.9%	34.3%	32.3%
% CPU Usage (APP)	4.69%	3.96%	3.64%

(b) CPU time (milliseconds) for varying Sampling Rates



(c) CPU usage with varying sampling rates



(d) CPU usage with varying window sizes

Figure 3. Impact of sampling rates and window sizes on CPU Usage

Table III. BATTERY USAGE OF OUR ANDROID APP ON SMARTPHONE AND SMARTWATCH (BL: BATTERY LEVEL AFTER ONE HOUR, RH: APPROXIMATE REMAINING HOURS

	Physical activity recognition on smartphone				**Smoking recognition on smartwatch**			
	Our app (DT)		Our app (KNN3)		Our app (DT)		Our app (KNN3)	
	Acc	Acc + Gyro	Acc	Acc + Gyro	Acc	Acc + Gyro	Acc	Acc + Gyro
BL	95%	93%	94%	91%	88%	86%	88%	%86
RH	19	13	16	10	7	6	7	6

E. Battery Consumption on Smart Phone and Smart Watch

Finally, we evaluated how much the smartphone and smartwatch battery last while running our activity recognition app. For this purpose, we ran our app for one hour on smartphone and smartwatch. We only used DT and KNN classifiers. We expect the overall battery consumption of DT to be closer to that of SMO, RF9, RF99, and NB based on our reported CPU usages in Table I(a). We present the battery usage after one hour in different scenarios in Table III. We kept the screen off in all these situations. In the case of idle mode with off screen, the battery level after one hour was 98% for the smartphone and 99% for the smartwatch. The results in

Table III show that smartphones are capable of running an activity recognition system for a reasonable amount of time (14-20 hours). In case of the smartwatch, the results are encouraging, too (6-8 hours) taking into account its low battery capacity (410 mAh). Battery capacities are improving with time so we expect the battery life to improve as well. As for as running trained classifiers for small data sets are concerned, it should not be an issue for a smartphone or a smartwatch. However, if online training is taken into account, then some of the classifiers may not be feasible for these devices such as MLP and random forest with a huge number of trees. Though training these classifiers take a small amount of time on a desktop machine, it can take considerably long on these small devices, thereby leading to a quick drainage of the battery.

V. Conclusions

In this paper, we presented an analysis on the resource usage of various aspects of a activity recognition system running on smart phones and smart watches. In terms of resource consumption, we observed that classification function takes a very small amount of CPU time out of total app CPU time. Most of the CPU is consumed by sensing and feature calculation. We recommend not to use the gyroscope unless it is necessary because due to the addition of the gyroscope CPU usage increases significantly. We observed that a complex model such as MLP or an ensemble model such as RF99 (99 trees) using accelerometer only is a better option than using a simple model such as a decision tree with both accelerometer and gyroscope as for as the resource consumption is concerned. We also observed that impact of window size on resource consumption is higher than the sampling rate. Therefore, increasing window size will lead to more saving in resources compared to decreasing sampling rates. A combination of decreasing sampling rate and increasing window size can be used to reduce the resource consumption. Based on our resource consumption analysis, we conclude that both smartphone and smartwatch are capable of running activity recognition systems for recognizing various activities for a reasonable amount of time. As a future work, we plan to develop a context-aware activity recognition algorithm where sensors, sampling rates, window sizes are decided on demand.

References

[1] M. Shoaib, S. Bosch, O. D. Incel, H. Scholten, and P. J. Havinga, "A survey of online activity recognition using mobile phones," *Sensors*, vol. 15, no. 1, pp. 2059–2085, 2015.

[2] O. Lara and M. Labrador, "A mobile platform for real-time human activity recognition," in *2012 IEEE Consumer Communications and Networking Conference (CCNC)*, Jan. 2012, pp. 667–671.

[3] P. Siirtola and J. Roning, "Ready-to-use activity recognition for smartphones," in *Computational Intelligence and Data Mining (CIDM), 2013 IEEE Symposium on*. IEEE, 2013, pp. 59–64.

[4] C. Schindhelm, "Activity recognition and step detection with smartphones: Towards terminal based indoor positioning system," in *IEEE 23rd International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC)*, Sep. 2012, pp. 2454–2459.

[5] S. Bhattacharya and N. D. Lane, "From smart to deep: Robust activity recognition on smartwatches using deep learning," in *IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*. IEEE, 2016, pp. 1–6.

[6] S. Sen, K. K. Rachuri, A. Mukherji, and A. Misra, "Did you take a break today? detecting playing foosball using your smartwatch," in *IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*. IEEE, 2016, pp. 1–6.

[7] M. Shoaib, "Sitting is the new smoking: online complex human activity recognition with smartphones and wearables," Ph.D. dissertation, 5 2017, cTIT Ph.D. thesis series no. 17-436.

[8] O. D. Lara and M. A. Labrador, "A survey on human activity recognition using wearable sensors," *Communications Surveys & Tutorials, IEEE*, vol. 15, no. 3, pp. 1192–1209, 2013.

[9] O. D. Incel, M. Kose, and C. Ersoy, "A review and taxonomy of activity recognition on mobile phones," *BioNanoScience*, vol. 3, no. 2, pp. 145–171, 2013.

[10] Y. Liang, X. Zhou, Z. Yu, B. Guo, and Y. Yang, "Energy efficient activity recognition based on low resolution accelerometer in smart phones," in *Advances in Grid and Pervasive Computing*. Springer, 2012, pp. 122–136.

[11] V. Stewart, S. Ferguson, J.-X. Peng, and K. Rafferty, "Practical automated activity recognition using standard smartphones," in *Pervasive Computing and Communications Workshops, IEEE International Conference on*, vol. 0. Los Alamitos, CA, USA: IEEE Computer Society, 2012, pp. 229–234.

[12] Z. Yan, V. Subbaraju, D. Chakraborty, A. Misra, and K. Aberer, "Energy-efficient continuous activity recognition on mobile phones: An activity-adaptive approach," in *2012 16th International Symposium on Wearable Computers (ISWC)*, Jun. 2012, pp. 17–24.

[13] M. Gjoreski, H. Gjoreski, M. Luštrek, and M. Gams, "How accurately can your wrist device recognize daily activities and detect falls?" *Sensors*, vol. 16, no. 6, p. 800, 2016.

[14] M. Gjoreski, H. Gjoreski, M. Lustrek, and M. Gams, "Recognizing atomic activities with wrist-worn accelerometer using machine learning," in *Proceedings of the 18th International Multiconference Information Society (IS), Ljubljana, Slovenia*, 2015, pp. 10–11.

[15] F. Attal, S. Mohammed, M. Dedabrishvili, F. Chamroukhi, L. Oukhellou, and Y. Amirat, "Physical human activity recognition using wearable sensors," *Sensors*, vol. 15, no. 12, pp. 31 314–31 338, 2015.

[16] R. Rawassizadeh, M. Tomitsch, M. Nourizadeh, E. Momeni, A. Peery, L. Ulanova, and M. Pazzani, "Energy-efficient integration of continuous context sensing and prediction into smartwatches," *Sensors*, vol. 15, no. 9, pp. 22 616–22 645, 2015.

[17] E. Poyraz and G. Memik, "Analyzing power consumption and characterizing user activities on smartwatches: summary," in *2016 IEEE International Symposium on Workload Characterization (IISWC)*, Sept 2016, pp. 1–2.

[18] X. Liu, T. Chen, F. Qian, Z. Guo, F. X. Lin, X. Wang, and K. Chen, "Characterizing smartwatch usage in the wild," in *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '17. New York, NY, USA: ACM, 2017, pp. 385–398.

[19] "Weka serialization and deserialization," https://weka.wikispaces.com/Serialization, last accessed on 11th Sep 2017.

[20] M. Shoaib, S. Bosch, O. D. Incel, H. Scholten, and P. J. Havinga, "Fusion of smartphone motion sensors for physical activity recognition," *Sensors*, vol. 14, no. 6, pp. 10 146–10 176, 2014.

[21] M. Shoaib, H. Scholten, P. J. Havinga, and O. D. Incel, "A hierarchical lazy smoking detection algorithm using smartwatch sensors," in *IEEE 18th International Conference on e-Health Networking, Applications and Services (Healthcom)*,. IEEE, 2016, pp. 1–6.

[22] "Traceview in android device monitor," https://developer.android.com/studio/profile/traceview.html, last accessed on 11th Sep 2017.

[23] "Dumpsys:," https://source.android.com/devices/tech/debug/dumpsys.html, last accessed on 11th Sep 2017.

[24] "Top command:," http://www.unixtop.org/man.shtml, last accessed on 11th Sep 2017.

[25] "Power tutor:," http://ziyang.eecs.umich.edu/projects/powertutor/, last accessed on 11th Nov 2016.