Information Fusion 62 (2020) 47–62

Contents lists available at ScienceDirect

Information Fusion

journal homepage: www.elsevier.com/locate/inffus

Classical and deep learning methods for recognizing human activities and modes of transportation with smartphone sensors☆

Martin Gjoreski[a,b,*], Vito Janko[a,b], Gašper Slapničar[a,b], Miha Mlakar[a], Nina Reščič[a,b], Jani Bizjak[a,b], Vid Drobnič[a], Matej Marinko[a], Nejc Mlakar[a], Mitja Luštrek[a,b], Matjaž Gams[a,b]

[a] Jožef Stefan Institute, Department of Intelligent Systems, Ljubljana, Slovenia
[b] Jožef Stefan Postgraduate School, Ljubljana, Slovenia

ARTICLE INFO

Keywords:
Activity recognition
Machine learning
Deep learning
Ensembles
Hidden markov models
Competition

ABSTRACT

The Sussex-Huawei Locomotion-Transportation Recognition Challenge presented a unique opportunity to the activity-recognition community to test their approaches on a large, real-life benchmark dataset with activities different from those typically recognized. The goal of the challenge was to recognize, as accurately as possible, eight locomotion activities (*Still, Walk, Run, Bike, Car, Bus, Train, Subway*) using smartphone sensor data. This paper describes the method we developed to win this challenge, and provides an analysis of the effectiveness of its components. We used complex feature extraction and selection methods to train classical machine learning models. In addition, we trained deep learning models using a novel end-to-end architecture for deep multimodal spectro-temporal fusion. All the models were fused into an ensemble with the final predictions smoothed by a hidden Markov model to account for temporal dependencies of the activities. The presented method achieved an F1 score of 94.9% on the challenge test data. We tested different sampling frequencies, window sizes, feature types, classification models and the importance of stand-alone sensors and their fusion for the task. Finally, we present an energy-efficient smartphone implementation of the method.

1. Introduction

The grand vision of wearable computing is that our devices will know as much as possible about our context in order to provide the best possible service. Which activity is being performed at any given moment is certainly an important part of our context, which is why Activity Recognition (AR) is intensely researched. Most research in AR is focused on the human body, dealing with activities such as walking, sitting and lying. However, transportation studies show that the average commute time is up to 80 minutes a day [1], and thus recognition of transportation modes can be as important as recognition of body-related activities.

The Sussex-Huawei Locomotion-Transportation (SHL) dataset [2] addresses this problem by providing a mixture of both activity types – containing eight different activities: *Still, Walk, Run, Bike, Car, Bus, Train, Subway*. In addition, the SHL dataset provides a unique opportunity for researchers to test their AR approaches against a common, real-life, large-scale benchmark dataset collected over a period of four months, which allows for cross-study comparison and systematic advancement of the research field. To promote the dataset, a competition called the "SHL challenge" was organized. The activities were to be recognized using seven smartphone inertial sensors: accelerometer, gyroscope, magnetometer, linear accelerometer, gravity, orientation and barometer.

This paper offers a detailed description of our approach, which we used to win the SHL challenge by most accurately predicting the activities on an unlabelled dataset. In Section 2 we present the related work and in Section 3 we describe the SHL dataset, including data ordering and splitting. The next four sections present our method, which is schematically shown in Fig. 1. Section 4 starts with how the data was down-sampled and cut into windows, and how we derived additional data streams from the existing ones. Then it shows how we extracted a large body of features from every data stream (altogether calculating 1696 features), and it ends up with the three-step feature selection process that we used to select the best-performing feature subset. Section 5 presents both classical and deep learning (DL) algorithms that we trained for the AR. The classical models used the selected features, while the DL model was trained on the raw data. For the DL we used a novel architecture: an end-to-end multimodal spectro-temporal ResNet (Multi-ResNet). All trained models were combined into an ensemble that we used to generate predictions.

Section 6 describes how we used a hidden Markov model (HMM) to account for the temporal dependencies of sequential activities. Section 7 discusses an energy-efficient smartphone implementation

☆ The first two authors should be regarded as joint first authors.
* Corresponding author.
E-mail address: martin.gjoreski@ijs.si (M. Gjoreski).

https://doi.org/10.1016/j.inffus.2020.04.004
Received 23 April 2019; Received in revised form 15 January 2020; Accepted 12 April 2020
Available online 21 April 2020
1566-2535/© 2020 Elsevier B.V. All rights reserved.

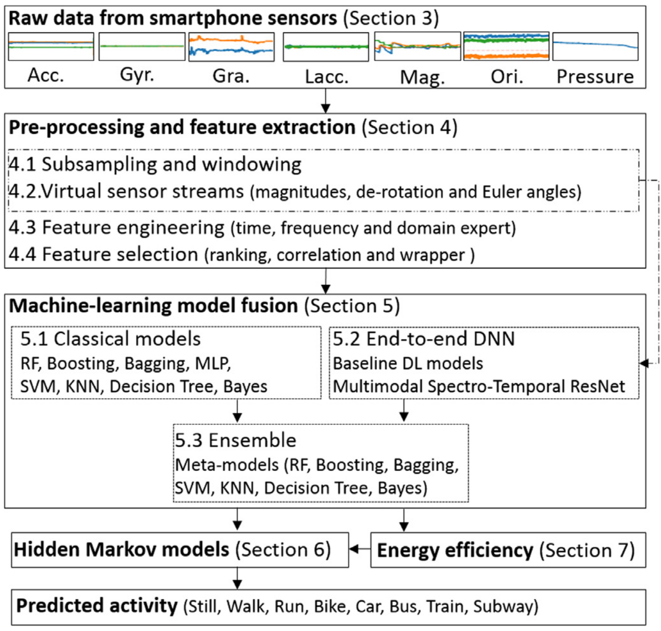
Raw data from smartphone sensors (Section 3)
Acc.
Gyr.
Gra.
Lacc.
Mag.
Ori.
Pressure
Pre-processing and feature extraction (Section 4)
4.1 Subsampling and windowing
4.2.Virtual sensor streams (magnitudes, de-rotation and Euler angles)
4.3 Feature engineering (time, frequency and domain expert)
4.4 Feature selection (ranking, correlation and wrapper)
Machine-learning model fusion (Section 5)
5.1 Classical models
RF, Boosting, Bagging, MLP, SVM, KNN, Decision Tree, Bayes
5.2 End-to-end DNN
Baseline DL models
Multimodal Spectro-Temporal ResNet
5.3 Ensemble
Meta-models (RF, Boosting, Bagging, SVM, KNN, Decision Tree, Bayes)
Hidden Markov models (Section 6)
Energy efficiency (Section 7)
Predicted activity (Still, Walk, Run, Bike, Car, Bus, Train, Subway)

Fig. 1. Scheme of our method.

of our method. It proposes three different ways to increase energy-efficiency while retaining classification accuracy. Section 8 presents the experimental results including an extensive analysis on how different hyper-parameters of our method affect its accuracy. These include selecting which sensors to use, data sampling frequency (e.g., 100 Hz, 50 Hz or 10 Hz), window sizes (e.g., 60 seconds, 20 seconds or 10 seconds), feature types (e.g., time-domain or frequency-domain), feature selection methods (e.g., wrapper or ranking methods), machine learning algorithms (e.g., deep or classical, end-to-end or feature-based models), etc. Finally, in Section 9 we summarize our findings and discuss the limitations of the method and our future work.

The main contributions of the paper are:

- A comprehensive method for AR, which was validated by winning the SHL challenge.
- A novel end-to-end DNN architecture (Multi-ResNet) proven suitable for AR, a domain where DNNs are not yet established.
- An extensive analysis of the method's hyper-parameters, which can facilitate adaptation to other problems.
- Optimization of the method's energy-efficiency for a smartphone application.

In summary, we believe the proposed method is a solid baseline for other authors who attempt to work on the SHL dataset and similar AR problems.

2. Related work

2.1. Classical AR methods

The AR domain has been thoroughly explored in the past using body-worn sensors, ambient sensors and combinations of both. Here, we focus only on the body-worn sensors commonly found in smartphones, since those are most suitable for AR today due to the omnipresence of smartphones. The most frequent AR task is classifying activities in relation to movement, e.g., walking, running, standing still and cycling [3,4]. Most of the multimodal (multi-sensor) solutions use classical ML algorithms (e.g., Random Forest - RF, Support-vector machines - SVM, and k-nearest neighbours - KNN) to build models from features that are extracted from each modality independently. These models can be built using fusion in the feature space, i.e., all features are treated uniformly [5,6]. There also exists fusion in the decision space, where models are built for each feature type with respect to the modality, and the final decision is output by a meta-model [7,8].

2.2. Deep AR methods

In the recent years, many attempts at AR were also made with DL end-to-end architectures [9,10]. The focus was on convolutional networks (CNN), which can automatically capture hierarchical feature representations of the data due to their stacked filtering layers [11]. CNNs, in combination with subsampling layers (e.g., pooling layers) and fully connected layers, are a very powerful end-to-end learning architectures [12–14]. In 2016, Ordonez et al. introduced the DeepConvLSTM [15], an architecture that stacks CNN layers over a multimodal long short-term memory (LSTM) recurrent neural network (RNN) for AR [16]. The LSTM layers allow for the model to learn the temporal dynamics in the input data by utilizing specific gates. Unlike HMMs, which are modelled on the Markovian assumption and have a finite number of hidden states, LSTMs have the advantage of a continuous space memory, which theoretically allows them to base their predictions on arbitrarily long past observations [17]. Most recently, Murahari et al. [18] experimented with attention mechanisms in the DeepConvLSTM, which improved its performance in the AR domain by a few percentage points.

2.3. Transportation recognition methods

The approaches that recognize transportation modes utilize very similar techniques to the standard AR approaches. For example, Martin et al. [19] developed a method for real-time prediction of the transportation mode using smartphone GPS and accelerometer data. They combined dimensionality reduction methods (PCA) and ML algorithms (KNN and RF) to accurately classify five modes of transportation (i.e., walking, biking, car, bus and rail). Fang et al. [20] developed a method for the recognition of transportation and vehicular modes using smartphone accelerometer, magnetometer and gyroscope. Their solution is based on decision trees, KNN and SVM. Reddy et al. [21] utilized a combination of a decision tree followed by a first-order discrete HMM to account for the temporal dependence between the labels. Similarly, Hemminki et al. [22] developed an accelerometer-based transportation recognition method for smartphones capable of recognizing six different modes (i.e., stationary, walking, bus, train, metro and tram). Their solution is based on a combination of AdaBoosting and a discrete HMM.

2.4. SHL Challenge methods

A comprehensive overview of all AR methods competing at the SHL challenge is presented in the summary paper by Wang et al. [23]. Overall, there were nineteen submissions, of which eleven used only classical AR methods, and eight either used only DL or included DL methods. From the top five methods, the teams “S304” and “Confusion Matrix” achieved a similar F1 score of 87.5%. “Confusion Matrix” used an RF model and then smoothed the estimation with majority voting [24]. S304 used a multi-layer perceptron neural network and then smoothed the estimation with an HMM [25]. The third placed team “Tesaguri” achieved an F1 score of 88.8% by applying CNNs to the spectrogram of the sensor data [24]. Our two teams “JSI-Deep” [26] and “JSI-Classic” [27] were the only two teams that achieved an F1 score over 90%. More specifically, JSI-Classic achieved an F1 score of 92.4%, by combining tree-based XGBoost with advanced feature extraction and feature selection techniques. “JSI-Deep” achieved the highest F1 score of 93.9% by using a meta method that utilizes both classical and spectrogram-based end-to-end deep learning methods merged in an ensemble whose final predictions are smoothed by a discrete HMM. This study presents a detailed analysis of the “JSI-Deep” and “JSI-Classic” methods with a few additional novelties including a novel spectro-temporal end-to-end DL

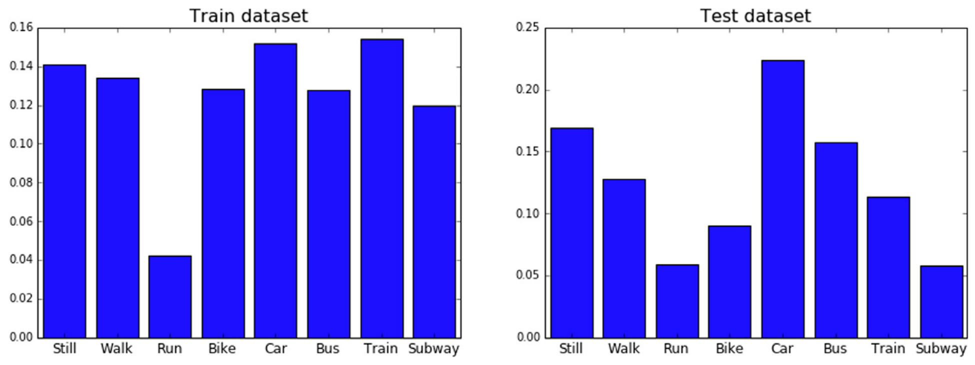

Fig. 2. Activity distribution of the *train* and *test* datasets.

architecture, and a methodology for making our solution more energy-efficient at the cost of a small decrease in accuracy. The "updated JSI-Deep" method presented in this study, achieved an F1 score of 95.2% on the challenge test data.

3. SHL challenge dataset

3.1. General description

The SHL Dataset is one of the largest AR datasets, suitable for a wide range of studies in fields such as transportation recognition, AR, mobility pattern mining, localization, tracking and sensor fusion [2]. It was recorded over a period of 7 months by 3 participants engaging in 8 different activities in a real-life setting in the United Kingdom. A subset of this dataset was used for the SHL Challenge and was subsequently used throughout this work. The dataset for the SHL challenge is available on the challenge website [28] and is described in detail in the baseline paper [29].

All of the data was provided by a single smartphone, worn by the same user in his trouser pocket. Data came in five different modalities: acceleration, magnetic field, angular velocity (measured by gyroscope), air pressure and the phone's current orientation. The acceleration was broken down into two additional data streams: the acceleration as the result of gravity, and the acceleration without the gravity component (linear acceleration). Notably, sensors that could identify the user's location – GPS, Wi-Fi, cell network – were not included.

The competitors were provided with a *train* and a test (*SHLtest*) set. The former came with labelled activities, while the labelling of the latter was the competition's goal. However, by the time of writing of this paper, the *SHLtest* set labels were publicly released, allowing us to present the results on both data sets. All together, there were 82 recorded days (62 for training, 20 for testing). This provided 16,310 minutes, or roughly 272 hours, of data.

There were 8 activities of interest: *Still, Walk, Run, Bike, Car, Bus, Train, Subway*. The activity distribution in the *train* set (Fig. 2) was mostly uniform, with the exception of the *Run* activity, which was (understandably) under-represented. The *SHLtest* set, on the other hand, was skewed towards the *Car, Bus* and *Still* activities. To simulate the competition conditions, we treated this difference in the distributions as an unknown and made no steps to make them more similar.

Example samples of the acceleration and magnetic field data are given in Fig. 3. Both exhibit a periodic and distinct pattern with different magnitudes and frequencies when the user is either walking, running or cycling. Acceleration data also captures the noise present when driving in vehicles. In addition, the magnetic field sometimes wildly oscillates while the user is in the train or subway, which can be potentially exploited for recognizing them.

3.2. Data ordering and data splits

The recording scenario of the SHL dataset was relatively natural, with the test subject moving around the city for several days (as opposed to the scenarios typically performed and recorded in a lab). This resulted in activities that on average lasted a long time (Table 1), with sensible transitions from one to another (Fig. 4). Both properties were exploited for the HMM smoothing (Section 6) and for energy-efficiency optimization (Section 7).

Table 1
Average duration of each activity.

Activity	Still	Walk	Run	Bike	Car	Bus	Train	Subway
Length [mins]	14	12	12	20	60	34	64	34

The data was split into one-minute segments, which were shuffled by the dataset's authors. The order of the one-minute segments was provided for the *train* set, but not for the *SHLtest* set. We assume this was done in order to enforce the use of a window length of less than or equal to one minute when classifying the data.

Our first step was to order the train set using the provided ordering. Then we split it into an internal validation set (*ivalid*), internal test set (*itest*) and internal train set (*itrain*) – using the first 25%, second 25% and last 50% of the *train* set, respectively. The *itrain* set was used to train our models. The *ivalid* set was used to select the appropriate features, sampling frequency and window size, and to train a meta-model. The *itest* set was used to estimate the performance of our model. All three sets were checked to ensure they roughly retained the same activity distribution as the original *train* set. We used the competition's *SHLtest* set for the final evaluation of our method. Note that the ordering of the data before making the split for internal evaluation was a key step, as otherwise two subsequent data samples (which are usually very similar) would frequently be one in the *itrain* set and the other in the *itest* set – leading to model overfitting.

The ordering of the *SHLtest* set was not required for our base model; however, at least locally ordered data was needed for the additional HMM smoothing step in an attempt to further increase the classification performance. Therefore, we had to create a method that could reassemble the one-minute segments into the original order.

Intuitively, data at the end of one segment should closely match the data at the beginning of the next one. A simple procedure can therefore be applied:

step 1: Select a random one-minute segment x.
step 2: Find the one-minute segment y that has the minimum distance from its first sensor reading to the last sensor reading of x. The distance used is the Euclidean distance with different sensors weighted using empirically determined weights.
step 3: If the minimal distance is lower than a predetermined threshold, assume that y follows x and repeat step 2 with y as the new x. If not, assume that the recording was interrupted and start assembling a new sequence of data repeating step 1.

On the *SHLtest* set, we produced 42 sequences of data that turned out (when the ordering was revealed after the competition) to be 99.8% correctly sorted, with roughly 10 instances out of place. The order of

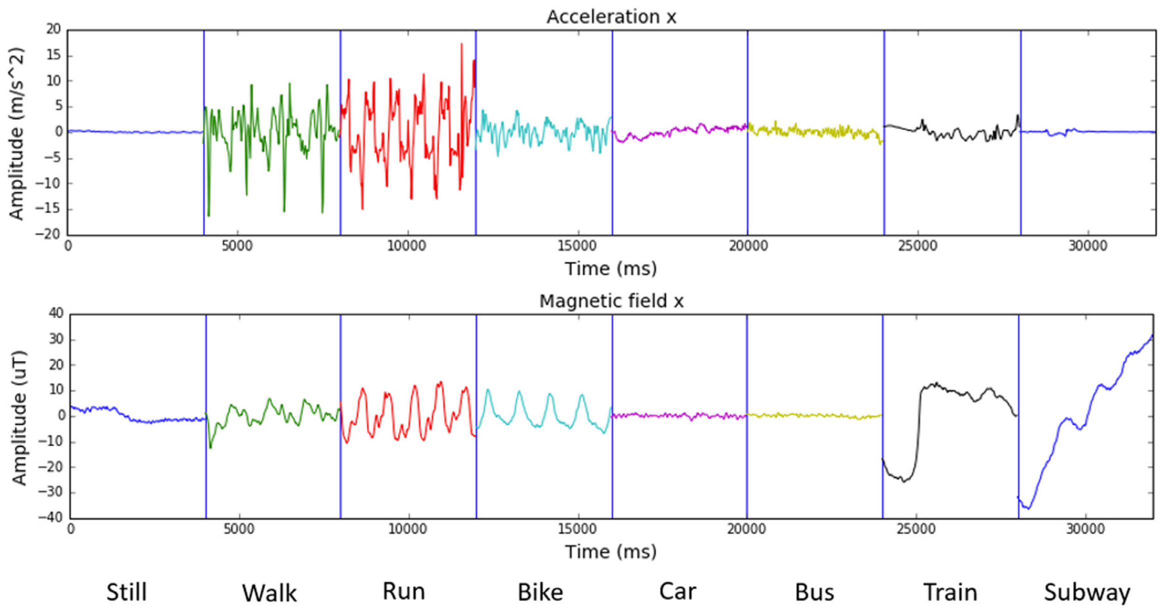

Fig. 3. Acceleration and magnetic field data samples (both taken from the x-axis). A four-second sample is shown for each activity.

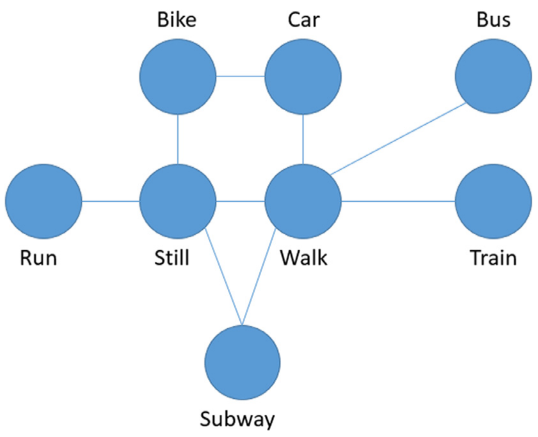

Fig. 4. All activities, connected if the probability of a transition from one to another is more than 30%.

the sequences themselves is largely irrelevant, however, as each had the HMM smoothing applied independently.

4. Pre-processing and feature extraction

This section describes the pre-processing step – windowing, down-sampling and deriving the virtual sensor streams – as well as the extraction of features used for training the classical ML models and their selection process. The DL methods use the raw data instead of features here described.

4.1. Data frequency and window size

The data was originally sampled using a 100 Hz sampling rate for all sensors. All sensor readings and activity labels coincided at the same timestamps. This frequency was previously empirically determined as more than adequate for most AR tasks [30,31]. However, the high sampling frequency presents two practical problems: 1) the further steps (notably the feature extraction) are slow due to the amount of data that needs to be processed; and 2) if this method would later be implemented on a smartphone, the required sampling frequency and the constant processing would shorten the battery life of the device.

To mitigate these issues, we first down-sampled the data. To determine a sensible frequency, we tested different frequencies using two different algorithms: a simple RF using 100 trees and frequency domain features (which were fastest to calculate), and the Multi-ResNet DL model presented in Section 5.2. We chose the RF as the baseline method because it is robust to noise, relatively fast to train and frequently used in the AR domain.

The next decision was to select the appropriate window size, which is used to split the sensor data into windows which are either fed directly to the Multi-ResNet or used for the feature extraction for the classical ML algorithms. Longer windows naturally contain more data and are expected to enable greater classification accuracy. Shorter windows, on the other hand, can detect an activity change faster. We once again used the same algorithms (RF and Multi-ResNet) to test the different window sizes, and repeated these tests with different frequencies. The results of these experiments are shown in Section 8.1.

The activity label was calculated as the majority of the per-sample labels (with the frequency of 100 Hz) in each window. The same procedure was used for testing, calculating the *majority-label accuracy*. This method was used throughout this work, as it is convenient to calculate and is used in most other works in the AR domain. Alternatively, one can compare the model's predictions with the per-sample labels, calculating the *per-sample accuracy* – which was used by the SHL challenge organizers. The majority-label accuracy is in general slightly higher, but the differences are nearly negligible (Section 8.1).

4.2. Virtual sensor streams

The SHL dataset provides 20 different sensor streams, if we are individually counting each axis of the 7 provided sensors. From these original sensor streams it is possible to derive additional sensor streams that are useful for the AR. The subsequent steps treat these derived sensor streams like any of the original ones.

The first derived sensor stream is the magnitude Eq. (1) of the data. It was calculated for all the data coming from tree-axis sensors (acceleration, linear acceleration, gravity, magnetic field and angular velocity).

$$m = \sqrt{x^2 + y^2 + z^2} \tag{1}$$

Second, the orientation data originally presented in the quarternions format $[q_w, q_x, q_y, q_z]$ was converted into the Euler angles – roll, pitch, yaw. While the quarternions have some desirable mathematical properties, the Euler angles are more intuitive and can be individually interpreted (one value for rotation around each axis). To make this transformation, we used the standard formulas for the task:

$$pitch = \arctan\left(\frac{2(q_w q_x + q_y q_z)}{1 - 2(q_x q_x + q_y q_y)}\right) \tag{2}$$

$$roll = \arcsin\left(2(q_w q_y - q_z q_x)\right) \tag{3}$$

$$yaw = \arctan\left(\frac{2(q_w q_z + q_x q_y)}{1 - 2(q_y q_y + q_z q_z)}\right) \tag{4}$$

Third, additional sensor streams were created by rotating the accelerometer and magnetometer data from the phone's coordinate system to the "world" (North-East-Down) coordinate system. This could be useful for determining, for example, if the magnetic field is coming from above or below, as the same axis is always pointed upwards. The transformation was done by multiplying the current values Eq. (6) with the coordinate system change matrix Eq. (5), using quarternions to determine the current orientation [32].

$$R_{NB} = \begin{bmatrix} 1 - 2(q_y^2 + q_z^2) & 2(q_x q_y - q_w q_z) & 2(q_x q_z + q_w q_y) \\ 2(q_x q_y + q_w q_z) & 1 - 2(q_x^2 + q_z^2) & 2(q_y q_z - q_w q_x) \\ 2(q_x q_z - q_w q_y) & 2(q_y q_z + q_w q_x) & 1 - 2(q_x^2 + q_y^2) \end{bmatrix} \tag{5}$$

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}_{world} = R_{NB} \begin{bmatrix} x \\ y \\ z \end{bmatrix}_{sensor} \tag{6}$$

In total we added fourteen derived data streams – five magnitudes, three Euler angles and six streams for rotated acceleration and magnetic field data (three axes each).

4.3. Feature engineering

Features were individually calculated for each data stream, with some features using data from all three axes of the same sensor at the same time.

Features can be roughly split into two categories: frequency-domain and time-domain. In aggregate, a total of 1696 features were computed and used in the subsequent steps.

4.3.1. Frequency-domain features

These features were calculated using the power spectral density (PSD) of the signal, which is based on the fast Fourier transform (FFT). PSD characterizes the frequency content of a given signal and can be estimated using several techniques. The simplest one is to use a periodogram, which is obtained by taking the squared-magnitude of the FFT components. An alternative to a simple periodogram is Welch's method, which is also widely used and commonly considered superior to the periodogram. It computes the average of the periodograms of multiple overlapping segments of the signal to reduce the variance of the PSD. In our work, we opted to use Welch's method to obtain the PSD.

Using PSD is only suitable when the signal is clearly periodic. We chose to test windows of length ranging from 5 to 60 seconds. These were long enough to contain several periods of human motion as well as vehicle vibration. Sample periodical patterns can be clearly seen in Fig. 3 for the activities: Walk, Run, Bike, Car and Bus.

We implemented frequency-domain features as given in related work [33]. The following features were computed.

- *Three largest magnitudes.* Three peaks with the largest magnitude from the PSD were considered. These tell us the dominant frequencies in the signal. Both the magnitude values and the frequencies (in Hz) were taken as features.
- *Energy.* Calculated as the sum of the squared FFT component magnitudes. The energy was then normalized by dividing it with the window length.

$$energy = \frac{1}{N} \sum_{n=0}^{N-1} |x(n)|^2, \tag{7}$$

where $x(n)$ is the n-th FFT component and N is the parameter specifying the number of FFT components to compute.
- *Entropy.* Calculated as the information entropy of the normalized FFT component magnitudes. It helps in discriminating between activities with a similar energy feature.

$$entropy = -\sum_{n=0}^{N-1} x(n) \log(x(n)) \tag{8}$$

- *Binned distribution.* A normalized histogram, which is essentially the distribution of the FFT magnitudes into 10 equal sized bins ranging from 0 Hz to 25 Hz.
- *Skewness and kurtosis.* Calculated on the PSD. Skewness and kurtosis describe the shape of the distribution of the PSD. More precisely, skewness tells us about the symmetry of the distribution while kurtosis tells us about its flatness.

4.3.2. Time-domain features

As the time-domain features, we used the expert features previously used in our other works in similar domains [6,34], including one previously-won competition [3].

A description and analysis of the expert features can be found in our previous paper [6]. In summary, the magnitude data stream provided the information on the intensity of the activity, while the individual axes provided the information on the orientation of the device and subsequently on the position of the user.

Some features come from statistics and describe the intensity and shape of the signal: the mean, variance, Pearsons correlation between axes, their covariance, skewness, kurtosis, quartile values and range between them. Others have a more physics-based interpretation, such as velocity and kinetic energy. The rest came from expert knowledge of the domain: the number and height of peaks in the window, the signal's mean, its sum and squared sum, and the number of times the signal crosses its mean value.

In addition, we used a subset of features from the *tsfresh* library [35] that seemed interesting, but were not included in our set of expert features. These features were: the signal minimum, maximum, the number of times the signal is above/below its mean, the signal's mean change and its different autocorrelations (correlations of the signal with a delayed version of itself, for different delays).

Some of the features were calculated on the unfiltered data streams, while some were calculated on data filtered with either a simple low-pass or band-pass filter. The filters were calculated using Eqs. (9) and (10), where x represents the raw data and y represents the filtered data.

- *Low-pass filter.* Removes the low frequencies from the data. Useful for filtering out, for example, the gravity component of the acceleration.

$$y_i = \alpha x_i + (1 - \alpha) y_{i-1} \tag{9}$$

- *High-pass filter.* Removes the high frequencies from the data. Useful for filtering out noise.

$$y_i = \alpha x_i + \alpha(x_i - x_{i-1}) \tag{10}$$

- *Band-pass filter.* This filter simply applies both low and high pass filters one after another.

4.4. Feature selection

Given the relatively high number of features (1,696), we used a feature selection procedure to remove the features that do not contribute to the accuracy of the model and only increase the odds for overfitting. Calculation of fewer features also contributes to the computational efficiency of our solution and simplifies its implementation. Our feature selection consisted of three steps.

In the first step, the mutual information [36] between each feature and the label was estimated using the *ivalid* set, where larger mutual information means higher dependency between the feature and the label.

After the features were sorted according to this value, correlated features were removed based on the Pearson correlation coefficient [37]. This showed that roughly half of the features were redundant, which was expected due to the number of features and the similarity of the data streams. To make the process computationally feasible, only 100 features were taken at a time, starting with those with the highest mutual information with the label. Correlation was then calculated for each pair. If the correlation was higher than a certain threshold (experimentally determined as 0.8), the feature with lower mutual information was discarded. After that, the next 100 features were added to the remaining set and the correlation between each pair was calculated again.

In the final step, features were selected using a greedy "wrapper" algorithm. An RF model was first trained using only the feature with the highest mutual information. The trained model was used to predict labels for the *ivalid* set, and the prediction accuracy was calculated. Then the second-best feature was added and the model was trained again. If the accuracy on the *ivalid* set was higher than without using this feature, the feature was kept. This procedure was repeated for all remaining features. This strict selection initially led to overfitting to the *ivalid* set (accuracy was much higher compared to the *itest* set), so the condition for keeping a feature was made less strict: the feature was kept if the accuracy did not decrease by more than an experimentally set improvement threshold. Using this rule, overfitting to the *ivalid* set was reduced. The results after each feature selection step are shown in Section 8.2.

5. Classification models

Different classical ML algorithms and an end-to-end DL architecture were trained and tested. After training the individual models, we combined them in an ensemble which further increased the classification performance. In this section we describe the models, and in Section 8.3 and Section 8.4 show the results.

5.1. Classical models

For the classical models, we used the ML algorithms as implemented in the scikit-learn ML toolkit [38]. Each of these algorithms used features as selected in Section 4. The models were tested both with the default hyper-parameter values and with those found using a 2-fold randomized parameter search. The randomized parameter search was performed using the following procedure: for each algorithm, parameter settings were randomly sampled from distributions predefined by an expert. Next, models were built with the specific parameters and then evaluated using a 2-fold validation procedure on the train data. This search was repeated 10 times using different folds. In Section 8.3 we report the averaged results, but we kept the best-performing parameters for the final models. We experimented with the following ML algorithms: Decision Tree [39], RF [40], Naïve Bayes [41], KNN [42], SVM [43], Bagging - using Decision Trees [44], Adaptive Boosting (AdaBoost), Extreme Gradient Boosting (XGB) and Multilayer Perceptron (MLP) [45], label switching both with MLPs [46] and with DTs [47].

5.2. Deep learning models

We propose a deep multimodal spectro-temporal ResNet (Multi-ResNet) and describe it in Section 5.2.2. To compare its performance to simpler DL architectures, we describe and test them as well.

5.2.1. Baseline deep learning models

To provide an end-to-end baseline comparison for our Multi-ResNet, we implemented five DL networks: a network with four CNN layers followed by two fully connected layers to output the final activity prediction (CNN-4); two similar networks with eight and sixteen CNN layers (CNN-8 and CNN-16); one network with two LSTM layers followed by two fully connected layers (LSTM-2); and one ConvLSTM mimicking the architecture proposed by Ordonez et al. [15], CNN-4-LSTM-2. Each CNN layer is followed by a batch normalization layer for reducing internal covariate shift [48], and a ReLU activation layer [49], which speeds up the training process compared to other activation layers (e.g., tanh). To avoid overfitting, L2 regularization and dropout was used for the dense layers. The training of the networks was fully supervised, by back propagating the gradients through all the layers. The parameters were optimized by minimizing the cross-entropy loss function using the Adam optimizer. The models were trained with a learning rate of 10^{-4} and a decay of 10^{-4}. The batch size was set to 1024 and the maximum number of training epochs was set to 75. The models were trained with early stopping on the *ivalid* data.

5.2.2. Multimodal spectro-temporal resnet

Our deep multimodal spectro-temporal ResNet (Multi-ResNet) is depicted in Fig. 5. The structure is based on the idea by He et al. [50] for training very deep end-to-end networks for image recognition by using shortcut (residual) connections. Using the residual networks, Wang et al. [51] proposed an end-to-end unimodal time-series classification network. Our network builds upon Wang's work with two additional novelties, which are key factors for a successful AR system, i.e., multimodal and spectro-temporal information fusion.

For each sensor channel, the network extracts channel-specific spectro-temporal information: the spectral information is extracted by calculating the spectrogram in decibels, i.e., log10 of the amplitude spectrogram, for each input window; the temporal representation is extracted by the residual blocks that contain CNN layers with 1-dimensional filters. The shortcut connections in the residual blocks combat the gradient vanishing problem, i.e., the more layers there are, the smaller learning update each layer receives, thus the harder the training is [30]. For example, for a 20 channel input (3 accelerometer, 3 gyroscope, 3 gravity, 3 linear acceleration, 3 magnetometer, 4 orientation and 1 pressure channel), 4 residual blocks per channel and 3 CNN layers per residual block, the network will have 240 CNN layers (20 x 4 x 3) through which the gradient needs to be propagated, thus we need a mechanism to avoid the gradient-vanishing problem [52]. Each CNN layer is followed by a batch normalization layer, and a ReLU activation layer. Each residual block ends up with an average pooling layer which is used for dimensionality reduction. The output of the channel-specific layers, i.e., channel-specific spectro-temporal information, is then fused by two dense (fully connected) layers. L2 regularization and dropout was used for the dense layers. The final output of the Multi-ResNet is provided by a softmax layer, which represents a class probability for each of the eight different classes. The network was trained using the same learning procedure (loss function, optimizer, learning rate, decay, batch size and early stopping) as the baseline DL models.

The original "JSI-Deep" method that won the SHL challenge used only the spectrogram part of the Multi-ResNet, i.e., the spectrograms were calculated for each sensor channel and were fed into fully connected layers, which provided the final predictions [26]. In this paper, that architecture is referred to as DNN-Spectrogram, in order to provide comparison between the two DNN architectures (Multi-ResNet vs. DNN-Spectrogram).

5.3. Ensemble

Finally, we experimented with an ensemble of models built using a stacking approach. The base models in the stacking ensemble are the classical models and the Multi-ResNet, described in the previous two

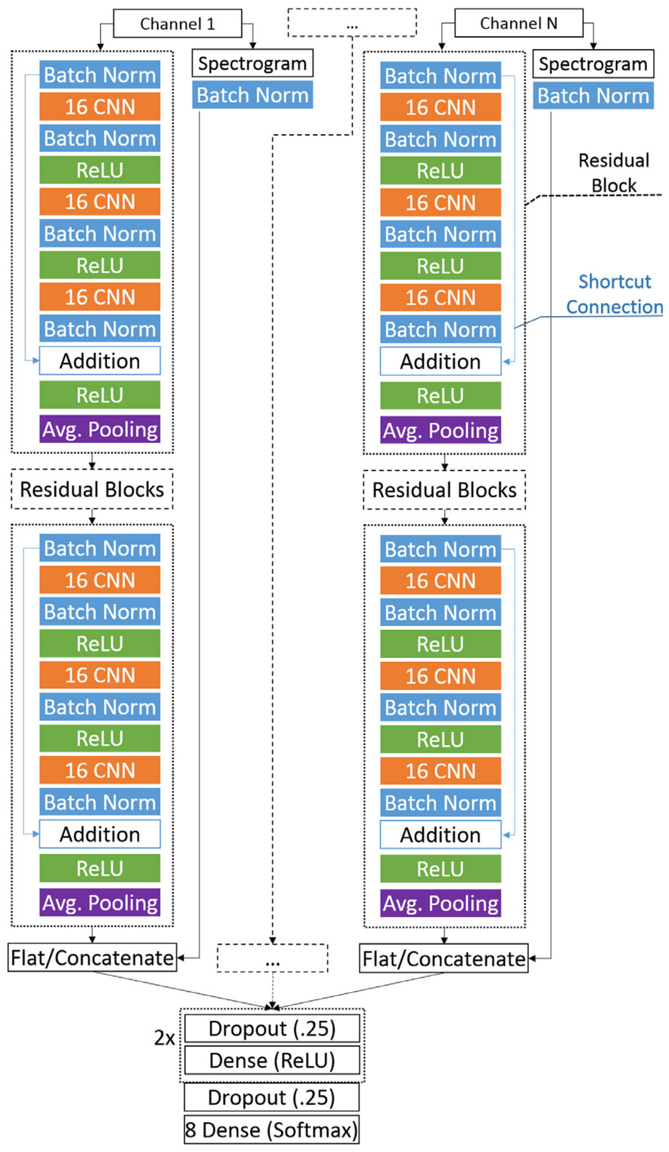

Fig. 5. Proposed Deep Multimodal Spectro-Temporal ResNet (Multi-ResNet).

subsections. The meta learner is a model that takes as input the outputs of base models. We evaluated meta-models built with the different ML algorithms: RF, Gradient Boosting, XGB, AdaBoosting, SVM, KNN, Gaussian Naïve Bayes and Decision Tree, and tuned the hyper-parameters. Each base model was trained on the *itrain* set and each meta-model was trained on the base models' output for the *ivalid* set. The hyper-parameter tuning was again performed using a 2-fold randomized parameter search on the train data.

6. Hidden markov model smoothing

In all experiments so far, all the windows were classified independently from one another. This approach discards all the information on temporal dependencies between them. If a user is currently on a train, for example, but the next window is classified as *Bus*, followed by another *Train* classification, it is far more likely for *Bus* to be a misclassification than a vehicle switch (Fig. 6).

This motivated us to use an extra step after classification, where the temporal information was taken into account. This was done using an HMM model. In this model (a small part of it shown in Fig. 7) the hidden

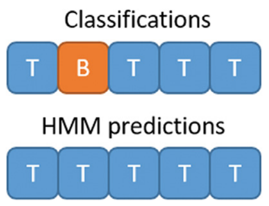

Fig. 6. Top row shows a sequence of *Train* and *Bus* classifications. They are corrected using HMM smoothing into a sequence of only *Train* activities shown below.

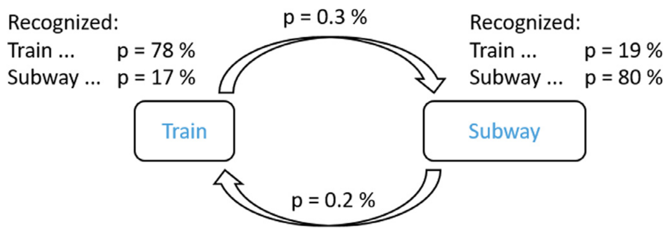

Fig. 7. A small part of the HMM model, where the hidden states represent true activities and the visible states are the recognized activities.

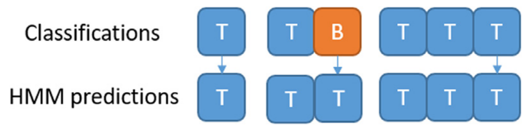

Fig. 8. Iterative HMM predictions. The top row represents the classified activities, and the bottom one represents the activities predicted by the HMM smoothing. Each group represents one step. In each step, the sequence is iteratively lengthened and only the last instance is corrected.

states represent the actual activity, while the visible output represents the classified activities.

The parameters of this model are the transition probabilities between the states and the probabilities of observed emissions in each state. The former can be estimated from the transition matrix of the *itrain* set (matrix of probabilities that one activity is followed by another), while the latter from the confusion matrix of the *ivalid* set.

The HMM smoothing was performed using the Viterbi algorithm in the *hmmlearn* library [53]. This algorithm uses all the available data for "correcting" each instance. From the viewpoint of a single instance, it means that all of its predecessors and successors must be known in order to "correct" it. This is, however, impractical for a real-time monitoring application implemented on a smartphone.

Having the past predictions corresponds to saving predictions as they are made, while having the future predictions means that the HMM smoothing must happen with a delay. To test the usefulness of the HMM smoothing in a practical setting, we tested how different delays (and having no delay – Fig. 8) affect the accuracy of the HMM predictions. To "correct" an instance in these tests, the algorithm was run only on the predictions that happened before the instance and on a number of instances (equal to the delay) that happened after it. The results are shown in Section 8.5.

7. Energy-efficient solutions

The previous sections were focused on how to achieve as high an accuracy as possible using a meta-model in combination with the HMM smoothing. To do so, we used all the available data from every sensor. In practice, however, turning on all the sensors of a smartphone will quickly drain the battery of the device, making the application undesirable from the user's point of view. It would thus be beneficial to find solutions that are almost as good in terms of accuracy, but are much more energy-efficient.

There are four most frequently used methods to reduce the energy consumption: using only a sensor subset, periodically turning the sensors on and off (*duty-cycling*), reducing the sampling frequency and reducing the processing power required for the classification. In this section we briefly describe – and apply on the SHL dataset – a general methodology that can quickly create energy-efficient solutions using the first two methods without any expert knowledge of the domain. This methodology is described in detail in our previous works [54,55]. The last two optimization modalities (sampling frequency and processing power required) are briefly discussed, using simple heuristics, at the end of the section.

Trying to estimate the energy consumption of a smartphone device can be surprisingly difficult, as: 1) it is heavily device-dependent; and 2) energy consumption of different components do not add up linearly. For example: accelerometer and gyroscope active together typically consume less energy than the sum of their individual consumptions. In order to avoid both complications, we simply measured what proportion of the original data we needed for a particular solution. This is a loose estimate of the energy efficiency, as the more data a system needs, the more energy it must consume to both collect and process it. This simplification is also appropriate because all sensors present in this dataset individually consume roughly the same amount of energy. Note, however, that the same methodology could be used with real energy estimates [56,57] if one would be interested in a particular device.

The HMM smoothing cannot be directly integrated into the methodology, so we omitted this post-processing step when searching for the solutions and then used it only on solutions found. Additionally, we used the ensemble without the DL model, as it can be impractical to use it on a smartphone. While recent research [58,59] show that DL models can be adapted to make them more resource efficient and thus more suitable for smartphones, we left such adaptations of our models to future work.

7.1. Choosing sensor subset

Instead of constantly using all five sensors, using only a subset of them would increase the energy-efficiency of the system. The individual performances of all sensors are listed in Table 6. Deriving a table of all possible sensor combinations ($2^5 = 32$) is likewise simple. Doing so reveals that individually the accelerometer is the sensor with by far the highest accuracy, and that the accelerometer with the magnetometer make the best pair. It shows that the contribution of other sensors rapidly falls afterwards, but that they are all required for achieving the highest possible accuracy.

Further improvements can be made by optimizing the sensors for each activity. For example, the magnetometer and barometer might only be useful in vehicles but not while walking or running. We can therefore create an assignment from each activity to a different sensor set, and whenever an activity is classified, the corresponding sensors are turned on. Since the activities last for minutes, the cost of switching a sensor on/off is minimal.

The problem of finding the ideal sensor-subset-to-activity assignment is not trivial. Not only are there $(2^5)^8 \approx 10^{12}$ different assignments, given 5 sensors and 8 activities, but it can be time consuming to test an individual one, as one has to go through the whole dataset, classifying each instance and switching sensor streams and models in the process. Note that trying to circumvent this by testing how a sensor subset works for recognizing an activity, and then aggregating this performance for every activity, makes a poor approximation. In practice, a misclassification can turn on sensors that are inappropriate for the current activity, leading to further errors.

To deal with both problems (the number of assignments and their evaluation) we used the method from our previous work [54]. In short, it uses the *steady-state* of a Markov-chain model to predict how would an assignment behave in terms of both energy and accuracy. To calculate the parameters of the Markov-chain model, it uses the matrix of transition probabilities between the activities and the different models (that use different sensor subset) confusion matrices. It then uses a genetic multi-objective algorithm (NSGAA-II [60]) – the objectives being energy consumption and accuracy – evaluating every assignment with the Markov-chain model. This combination can quickly search through the problem space and efficiently find a set of non-dominated solutions.

Doing so generates an approximation for the Pareto front of solutions, each solution representing a different trade-off between accuracy and energy consumption. From this set of trade-offs, a system designer can pick one that is suitable for his application requirements. Section 8.7 presents the Pareto front for the SHL dataset and some sample solutions from it.

7.2. Duty-cycling

As shown in Table 1, most of the activities are relatively long-lasting. Therefore turning the sensors off when an activity is recognized and then turning them back on a few minutes later could be a way to further optimize the energy consumption of the system. The system simply assumes that the activity has not changed while the sensors were turned off. This process, called *duty-cycling*, can similarly to before, be enhanced by optimizing the cycle length for each activity individually. Intuitively, long lasting activities should have longer cycles, and vice versa. Additionally, if an activity is often followed by a short-lasting activity, the duty-cycle length should also be shorter, so as not to miss the following short activity entirely. Finally, one has to consider the effect of misclassifications, as misclassifying an activity might subsequently cause a wrong duty-cycle length.

This once again introduces two problems: 1) combinatorial complexity, as there are many possible duty-cycle-length-to-activity assignments – l^8, if the maximum cycle length is l; 2) modelling the performance of the system given a duty-cycle-length-to-activity assignment. We solve both problems by using our previous work [55], where we once again use Markov-chain calculus (although in an altogether different way) to determine the effect of an assignment on the system performance and then use the NSGAA-II algorithm to find a set of non-dominated solutions.

In Section 8.7, we show sample solutions where we used the same duty-cycle length for all activities, and we compare them to solutions where activity-specific duty-cycle lengths are used.

7.3. Combination

In our previous works, we used both described methods independently. In this work, we combine them, by first selecting which sensor subset to use for each activity and then running the duty-cycle optimization using this assignment. This requires two choices (one in each step) regarding which solution from the Pareto front to use, but otherwise seamlessly combines the two.

One can combine the selected solution with the optimization of the sampling frequency – which is the third, smaller contributor to the energy consumption of the device. Testing different frequencies (Section 8.1) revealed that using the frequency of 50 Hz loses basically no classification accuracy compared to the frequency of 100 Hz. We thus settled to using it, and made no additional effort at optimizing the frequency, but a reader can find more sophisticated methods in related work [30,61].

Finally, to reduce the power consumption required for classifying a single instance, we compared the processing requirements of different classifiers and their ensembles. Different trade-offs between the classification accuracy and processing power required are shown in Section 8.7. It should be noted, however, that since classifications happen only once each minute, the data processing is not the bottleneck when considering the energy consumption.

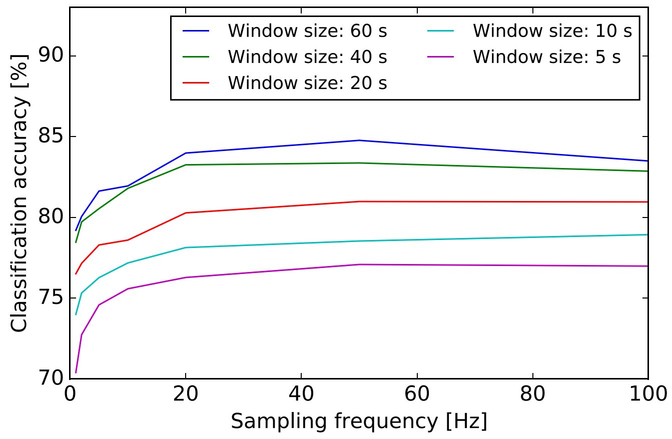

Fig. 9. Accuracy on the *itest* set for different frequencies and for different window sizes, using the RF model.

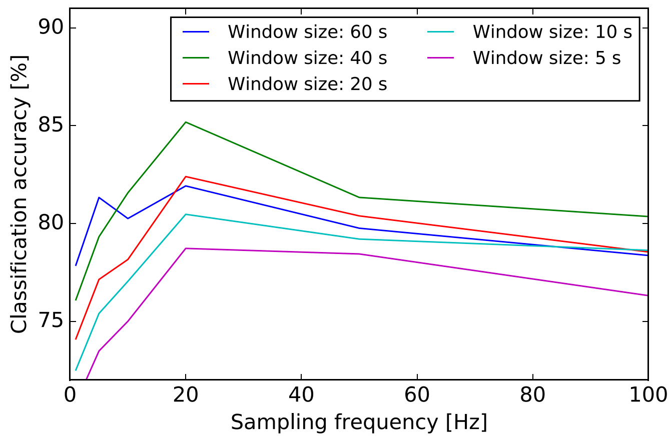

Fig. 10. Accuracy on the *itest* set for different frequencies and for different window sizes, using the Multi-ResNet.

8. Results

For the experimental evaluation, we mostly used classification accuracy as the measure of classification quality, but for the final results we also list the precision, recall, and F1 score:

$$Accuracy = \frac{True\ Positive + True\ Negative}{All\ Test\ Instances} \tag{11}$$

$$F1\ score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{12}$$

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive} \tag{13}$$

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative} \tag{14}$$

When calculating the F1 score for multiple activities, the F1 score is calculated for each activity separately and then averaged over all of them.

8.1. Frequency and window size

We started by testing different sampling frequencies and window sizes. Results using two different models (trained on the *itrain* set and tested on the *itest* set) are shown in Figs. 9 and in 10.

The results for the RF show almost no difference in performance when dropping from a sampling frequency of 100 Hz to a sampling frequency of 20 Hz. At lower frequencies, the performance started to visibly decrease, but surprisingly, even at a sampling frequency of 1 Hz, the accuracy only fell by roughly 5 percentage points compared to the highest value. Given the results in similar domains, it is not surprising that 20 Hz is enough to recognize the human motion, however it is interesting to note that it also seems enough to capture the different vehicle vibrations. Additionally, we can see that larger window sizes increase the accuracy regardless of the frequency.

The results for the Multi-ResNet show that the accuracy reached its peak at a sampling rate of 20 Hz and decreased at lower rates, just as in the RF case. Somewhat curiously, the 40 second window was found the best performing by a significant margin, while it was a close second in the RF case.

For the classical ML models, we selected a sampling rate of 50 Hz and a window size of 60 seconds, to match our submission to the SHL competition. However, these tests show that any other frequency from 20 Hz to 100 Hz would be similarly suitable. For the Multi-ResNet, we continued with the best performing parameter combination, i.e., a sampling rate of 20 Hz and a window size of 40 seconds, since the accuracy significantly decreases for other combinations.

Next, we investigated the difference between having per-sample labels and majority labels. To do so, we calculated the accuracy for both cases for each window size. These tests were done using two frequencies at the opposite end of the spectrum. The results are presented in Table 2, and show that the difference is (sometimes significantly) less than 1%. Results were also unaffected by the selected frequency. Due to the small difference and the reasons explained in Section 4.1, the results reported from here on will continue using the majority labels.

Table 2
Difference in accuracy on the *itest* data using the per-sample labels and the majority labels, using different frequencies and window sizes.

Window length [s]	60	40	20	10	5
Difference [%] at 100 Hz	0.57	0.14	0.04	0.01	0.02
Difference [%] at 1 Hz	0.55	0.17	0.04	0.01	0.01

Lastly, we tested the sensitivity to the choice of the *itrain, itest, ivalid* set. To do so, we tested what happens if we switch them around, e.g., training on the *itest* and testing on the *itrain*. All the experiments from Fig. 9 were repeated for all six possible switches of these three sets. The standard deviation of the results was 1.6% – from which we can conclude that the dataset is quite robust to the choice of the internal train, test, and validation sets. This can probably be attributed to its relatively large size.

8.2. Feature importance and selection

The next area for attention is the importance of the features based on both the sensor stream they were derived from and their type (time-domain, frequency-domain, rotated to world's coordinate system etc.)

We start by training the models on data from only one sensor stream, using both the RF and the Multi-ResNet as before. Results (Table 3) show – somewhat unsurprisingly – that the accelerometer is the best suited for the task, with the gyroscope and magnetometer data in second and third places. For the Multi-ResNet, no sensor stream by itself outperforms their combination – showing the importance of the sensor fusion in this domain. In the RF case, the accelerometer seems to outperform the sensor union, but this ceases to be the case after the feature selection step.

Table 3
Accuracy [%] on the *itest* data achieved using data from only one sensor.

Sensor	Acc	Gyro	Pressure	Mag	Ori	All
RF	86.5	79.5	52.1	72.1	71.3	82.6
Multi-ResNet	81.4	67.2	49.0	67.7	65.9	85.2

Table 4
Accuracy [%] on the *itest* data for different feature subsets using RF.

Features	Mag	Acc
Rotated	76.6	61.6
Un-rotated	76.6	84.3
Un-rotated + Rotated	79.5	86.5
Time	73.7	81.6
Frequency	73.3	83.9
Frequency + Time	79.5	86.5

Table 5
The number of features kept and the accuracy of RF after each step of feature selection on both *itest* and *ivalid* set.

	Features	Accuracy [%]	
		ivalid	itest
All features	1,696	86.9	82.6
Correlation removed	762	87.6	84.8
Wrapper	203	89.5	86.9
Wrapper strict	65	89.8	87.5

Table 6
The number of features selected from each sensor's data stream for the "Wrapper strict" feature subset.

Sensor	Acc	Gyro	Pressure	Mag	Ori
#Selected	30	15	1	7	11

In the next step, we more thoroughly investigate the impact of different feature types, by choosing different subsets of acceleration and magnetic field features. The subsets were made by either splitting the features based on whether they are un-rotated (Section 4.2), or based on whether they are computed in the time or frequency domain. The results are shown in Table 4. The most significant observation, is that no feature subset outperformed the whole set, confirming the usefulness of each feature type. It is worth pointing out that the features in the normalized coordinate system – which are not normally used in AR – consistently improved the accuracy by 2% (while not being good on their own, as there were simply fewer of them). Another interesting observation is that the frequency-domain accelerometer features outperformed the time-domain accelerometer features, even though the latter are more common in AR.

Finally, we present the effect of the feature selection method described in Section 4.4. The correlation threshold was set to 0.8. We tested two different improvement thresholds for the Wrapper step, resulting in "Wrapper" (improvement threshold set to 0.2%) and "Wrapper strict" (improvement threshold set to 0.05%). The results are listed in Table 5.

For the subsequent experiments we continued with the feature subset "Wrapper strict". Table 6 presents more information about this feature subset. From the results, it can be seen that most of the selected features are calculated from the acceleration data; followed by angular velocity and orientation data; next is the magnetometer data; and finally, only one feature is calculated from the pressure data.

8.3. Base models

After choosing the "Wrapper strict" feature set, we proceeded to test different ML algorithms. The results are presented in Table 7. For the classical ML models, the column "Default" presents the models' accuracy with the default scikit-learn parameters, and the column "Tuned" presents the models' accuracy after the 2-fold randomized parameter search. This search was repeated 10 times using different folds, averaging the results. For the end-to-end DL models, the column "Default" presents the models' performance with all sensors as input, and the column "Tuned" presents the models' performance with the top three modalities as input (accelerometer, gyroscope and magnetometer). Multi-ResNet-2 is our DL method with 2 residual blocks per channel, Multi-ResNet-4 is the same method with 4 residual blocks per channel and so on. From the results, it can be seen that most of the classical ensemble models (i.e., RF, Bagging, Gradient Boosting, XGBoosting) performed better compared to the classical single models (i.e., SVM, DT, KNN and MLP). The highest accuracy of 89.4% is achieved by the Gradient Boosting (GradBoost) algorithm. The accuracy of the baseline DL methods is below 77%, which indicates that simple DL methods perform similar to the simple classical ML methods (e.g., DT and NB). In contrast, the accuracy of our DL method (Multi-ResNet) on the same *itest* set is 89.4%, which is basically the same as the best performing classical ML method.

In Fig. 11, we present the learning curves (accuracy with respect to the number of training epoch) for the end-to-end DL models. It can be seen that all of the models except the LSTM achieved training accuracy close to 100%. This indicates that the models converged with respect to the learning phase. However, for all of the baseline models the accu-

Table 7
Accuracy achieved using different classical ML models and end-to-end DL models (including our end-to-end Multi-ResNet model) on the *itest* data. Standard deviation was calculated across ten repetitions of the experiment.

	itest Accuracy [%]		
	Default	Tuned	Std
SVM	81.1	84.1	0.8
DT	72.3	75.4	1.3
NB	76.7	76.7	0
KNN	79.3	79.3	0.2
MLP	82.0	86.7	0.4
RF	87.5	87.7	0.1
Bagging	83.4	87.1	0.4
GradBoost	88.6	89.4	0.3
AdaBoost	46.1	89.1	0.4
XGBoost	88.4	89	0.3
LabelSwitch	82.1	86.8	0.1
CNN-4	60.3	64.5	0.3
CNN-8	58.9	63.1	0.3
CNN-16	58.1	60.2	0.3
LSTM-2	60.1	62.3	0.2
CNN-4-LSTM-2	74.1	76.4	0.4
DNN-Spectrogram	80.1	81.2	0.4
Multi-ResNet-2	80.6	85.4	0.3
Multi-ResNet-4	84.8	89.2	0.2
Multi-ResNet-6	85.2	89.4	0.2

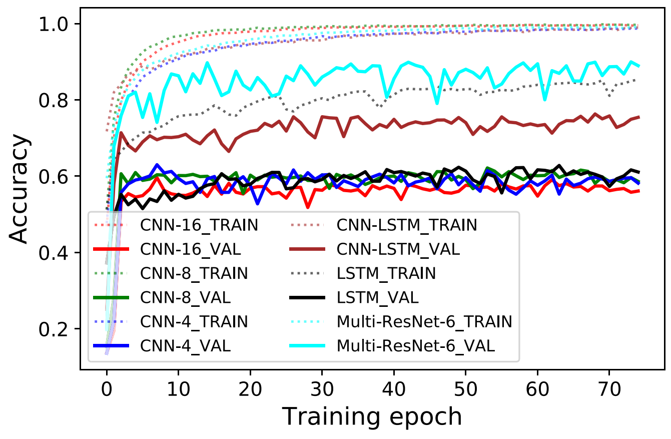

Fig. 11. Learning curves for the end-to-end DL models. The training accuracy (*itrain* data) is presented with dashed lines and the accuracy on the *ivalid* data is presented with solid lines.

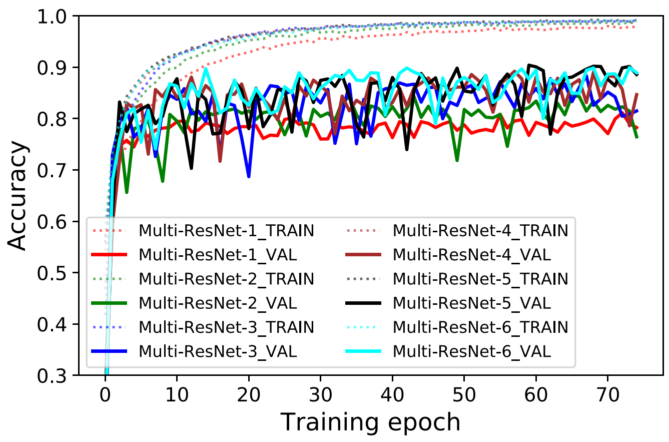

Fig. 12. Learning curves for the Multi-ResNet model with a varying number of residual blocks (from 1 to 6). The training accuracy (*itrain* data) is presented with dashed lines and the accuracy on the *ivalid* data is presented with solid lines.

Table 8
Accuracy achieved using different meta-models on the *itest* set. Standard deviation was calculated across the ten different data splits.

	itest Accuracy [%]	
	Mean	Std
meta-SVM	92.6	0.7
meta-DT	88.0	0.9
meta-NB	91.0	0
meta-KNN	92.1	0.3
meta-RF	93.0	0.1
meta-Bagging	93.0	0.4
meta-GradBoost	93.3	0.3
meta-AdaBoost	93	0.2
meta-XGBoost	93.4	0.3

racy on the *ivalid* data is close to 60%, except for the CNN-LSTM, which achieved accuracy of 75%. This indicates that simple architectures cannot achieve high accuracy. In contrast, our Multi-Resnet outperformed all of the baseline DL models with a margin of at least 10 percentage points.

Finally, in Fig. 12 we present learning curves for the Multi-ResNet with a varying number of residual blocks (from one to six blocks per channel). From the figure it can be seen that the accuracy increases as the number of residual blocks increases. This trend is present up to four residual block per channel. The Multi-ResNet-4, Multi-ResNet-5 and Multi-ResNet-6 perform similarly.

8.4. *Ensemble of models*

All the resulting models from the previous experiment (Table 7) were used as base models in an ensemble. Different meta-models were created using different ML algorithms and were tuned using a 2-fold randomized parameter search. This search was again repeated 10 times using different folds. From the results presented in Table 8, it can be seen that most meta-models (with a few exceptions) have similar accuracy, with the meta-model built with the XGB algorithm having the highest accuracy – 93.4%.

With the meta-XGB being the best meta-model, we analyzed which base models contribute to its performance. In particular, we were interested to find if both ML and DL models were required for the best results. To do so, we tried using only ML models, only DL models (in this case the default and tuned version of the Multi-ResNet), their combination, and only the best model from each category. The results are shown in

Table 9
Accuracy achieved using either (or both) classical ML or DL classifiers as the base of the ensemble on the *itest* set. Either all models of a type were used, or only the best one. Standard deviation was calculated across the ten different data splits.

Models used	*itest* Accuracy [%]	
	Mean	Std
All ML	91.3	0.3
All DL	89.8	0.3
All ML + All DL	93.4	0.3
Best ML + Best DL	93.8	0.2

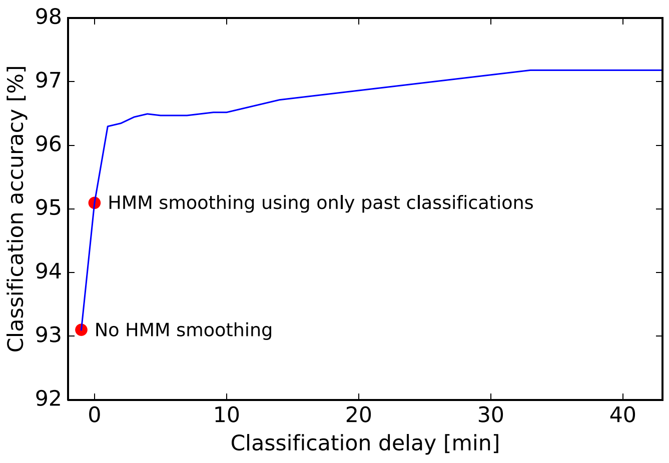

Fig. 13. Accuracy [%] for Markov smoothing applied with different delay intervals on the *itest* data.

Table 10
Different methods for smoothing the data. The test setting with no delay was used for comparison.

No smoothing	HMM	RNN	LSTM	Bi LSTM	GRU
93.4%	95.3%	93.7%	94.2%	93.5%	94.4%

Table 9, and show that having more than one ML model barely improves the performance; however, by combining the ML and DL models (which are substantially different), the accuracy increases significantly.

8.5. *Markov smoothing*

The last step was applying the HMM smoothing. Fig. 13 presents the accuracy with Markov smoothing applied using different delay intervals. The figure shows that even when working with no delay, using only past classifications to “smooth” the current one, the accuracy increased from 93.1% to 95.1%. Small delays slowly increased the accuracy up to its final value of 97.2%. By testing HMM smoothing with predictions made by different classifiers and ensembles, we noticed that the most substantial accuracy gains happen with 5–10 minute delays. Since the SHL Challenge allowed for an offline classification, we used the longest possible delay (using all of the data) for the challenge submission, as this was expected to yield the highest accuracy.

As an alternative to the HMM smoothing, we tested RNN, LSTM, bidirectional LSTM [62] and GRU [63] neural networks to smooth the predictions. These models used past and current predictions as the input and output the “corrected” current prediction. Results using these models are shown in Table 10. They all improved the prediction accuracy, but not to the extent of the HMM model.

Table 11
Accuracy [%] achieved after different steps of the proposed methodology. Test were performed on both internal and final test sets. Unless stated otherwise, each following step uses parameters of the previous one (e.g. all steps after the feature selection step, use the selected features). WL - window length. "/" denotes that ML and DL use different parameters.

	itest		SHL Test	
	ML	DL	ML	DL
WL - 5 sec, RF and frequency features for ML	77.1	78.7	85.1	78.8
WL - 60/40 sec	84.7	85.2	88.2	81.2
Using all features	82.6	/	89.8	/
Feature selection / Sensor stream selection	87.8	89.3	88.4	90.8
Using best classifier	89.4	/	90.9	/
Ensemble (Best ML + Best DL)	93.8		94.7	
Using HMM smoothing	97.2		96.0	

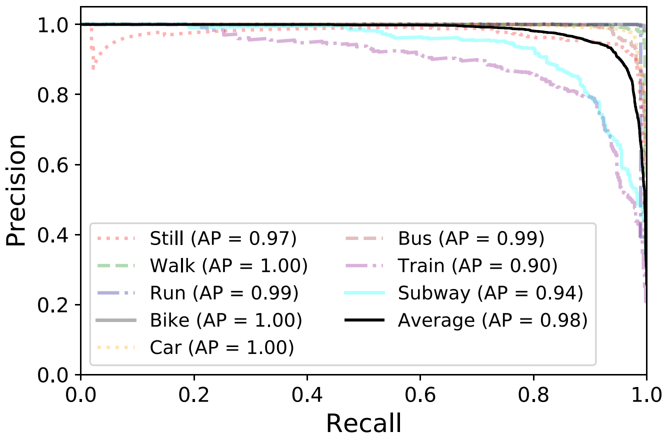

Fig. 14. Precision-recall curve for each of the activities and their average. The AP stands for the average precision, defined as $\sum_n (R_n - R_{n-1}) P_n$, where R_n and P_n are the recall and precision for the n-th decision threshold.

8.6. Experiments on the SHL test set

By the time of writing this paper, the SHL challenge organizers had released the labels for the SHL Test set, thus enabling us to evaluate our methodology on it. Table 11 shows the different steps of our methodology on both the *itest* and the *SHL Test* set. For the *SHL Test*, set we used the same parameters as for the *itest* set, with no additional fitting to it. This table can serve as a summary on how much each step improves the performance. On the *itest* set, every step increased the performance as expected, with the biggest improvements being attributed to the window size (7 percentage points), feature selection (5 percentage points), ensemble (4 percentage points) and the use of HMM smoothing (4 percentage points). In the case of the *SHL Test* set, the starting accuracy was much higher (probably due to different activity distribution), and the role of feature selection was diminished – in fact, using all the features worked best. The sensor stream selection, ensemble and HMM smoothing provided similar benefits. The final results for both test sets are roughly the same.

Table 12 presents the confusion matrix and the precision, recall and F1 score for each class, achieved by this method on the *itest* data. From the confusion matrix, it can be seen that the most problematic activities are the *Train* and *Subway*, i.e., the only two activities that have an F1 score lower than 90%. This is not surprising, since the *Train* and *Subway* are in fact very similar. This also explains why most classifiers worked better on the *SHL Test* set than on the *itest* set, as the former contained proportionally far fewer of these two activities. An alternative display of the model's performance using Precision-Recall curves is presented in Fig. 14. Such curves would be useful to modify the decision threshold

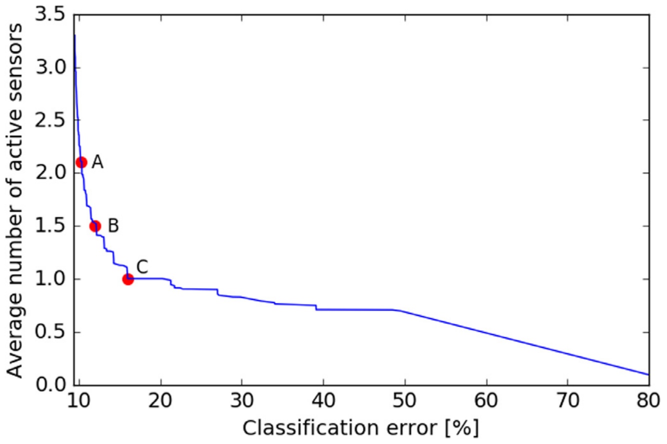

Fig. 15. Different trade-offs between the average number of active sensors and the classification error. Some sample solutions are marked with letters A, B and C.

if we would be particularly interested in correctly detecting a specific activity.

Table 13 presents the final results on the *SHL Test* and compares them to the results of other competitors. To be able to compare these results, we list the results using F1 score, as was used in the competition. The JSI-Deep and JSI-Classic are our methods which were ranked in first and second places at the SHL challenge, and have served as the basis for the development of the method presented in this paper, the meta-XGB-HMM. The last three rows are the third, fourth and the fifth ranked methods at the SHL challenge. From the results, it can be seen that our methods achieved an F1 score that is at least 3.6 percentage points higher, compared to the best performing method from the related work [64]. The meta-XGB-HMM presented in this paper yielded the highest F1 score of 94.9%, which is an increase of 6.1 percentage points compared to the best performing method from the related work.

8.7. Energy efficiency

To explore different trade-offs between energy efficiency and the classification error, we first calculated the Pareto front of the trade-offs that use different sensors for classifying different activities – as described in Section 7.1. The results are shown in Fig. 15. The most interesting solutions seem to lie in the area between one and two sensors used on average. On the same figure, three sample solutions are listed. Note that by not using the DL in the ensemble, the maximum achievable accuracy is 90%.

- Solution A: *Accuracy lost: 0.5%, average number of sensors used: 2.1/5* This solution uses the accelerometer regardless of the activity. It uses the magnetometer in all vehicles and when standing still. In addition, it uses the barometer when walking and on the subway (perhaps to detect the pressure change that happens when the user enters or leaves a subway station). Somewhat peculiarly, it uses the orientation sensor when on the bus or train – potentially to use the rotated magnetometer features that can recognize the orientation of the magnetic field caused by the train.
- Solution B: *Accuracy lost: 2%, average number of sensors used: 1.4/5* The accelerometer is used in all cases. The magnetometer is used when on the subway or train – the two vehicles that are hardest to distinguish from one another. All other sensors are inactive (with the exception of the gyroscope when running).
- Solution C: *Accuracy lost: 7%, average number of sensors used: 1/5* Only the accelerometer is used in all cases.

Table 12
Confusion matrix, precision, recall and F1 score on the *itest* data for the model with highest accuracy, the meta-XGB-HMM model.

	Still	Walk	Run	Bike	Car	Bus	Train	Subway
Still	921	7	0	3	3	11	12	5
Walk	4	718	4	2	1	0	1	1
Run	0	9	328	0	0	0	0	0
Bike	3	0	0	508	0	0	0	0
Car	3	0	1	0	1272	0	0	0
Bus	10	8	1	1	0	876	4	0
Train	30	2	0	0	7	20	514	74
Subway	1	2	0	0	0	0	0	331
Recall [%]	95.7	98.2	97.3	99.4	99.7	97.3	79.4	99.1
Precision [%]	94.8	96.3	98.2	98.8	99.1	96.6	96.8	80.5
F1 score [%]	95.2	97.2	97.8	99.1	99.4	97.0	87.3	88.9

Table 13
Evaluation on the SHL challenge test data. Comparison of F1 scores between our work and the top ranked teams at the challenge.

Name	Algorithm	F1 score
(Our) meta-XGB-HMM	ML + DL + HMM	94.9%
(Our) JSI-Deep [26]	ML + DL + HMM	93.9%
(Our) JSI-Classic [27]	ML (XGBoost)	92.4%
(Rel. work) Tesaguri [64]	DL (spectrogram CNN)	88.8%
(Rel. work) S304 [25]	DL (feature MLP) + HMM	87.5%
(Rel. work) Conf. Matrix [24]	ML (RF)	87.5%

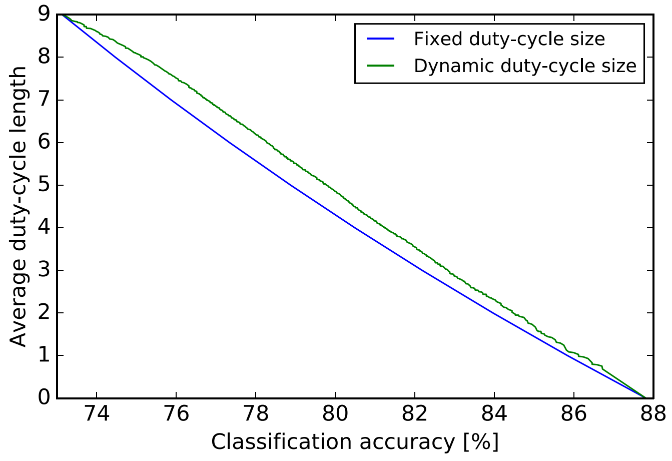

Fig. 16. Different trade-offs between the average duty-cycle length and the classification error on the *itest* data. Dynamic solutions are compared to those with a fixed duty-cycle length.

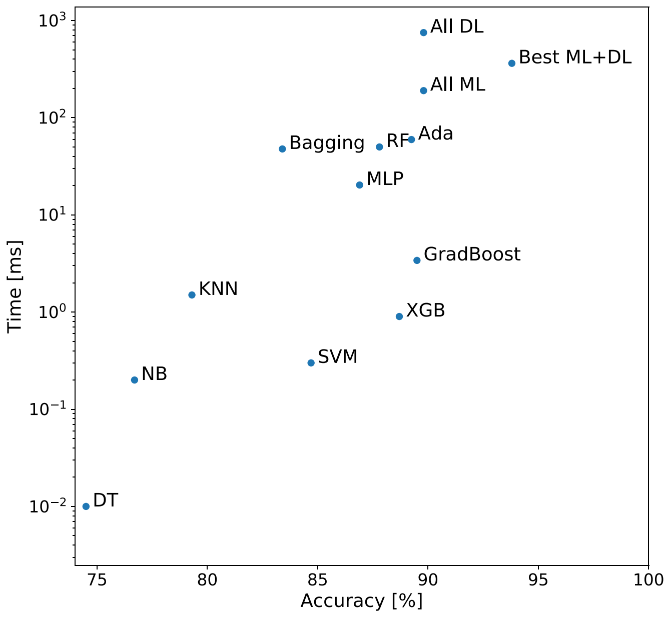

Fig. 17. Time for the classification of a single instance (note the logarithmic scale) for different methods. The "all DL", "all ML" and "best ML + DL" represent the corresponding ensembles.

Table 14
A sample assignment of duty-cycle lengths to each activity.

Activity	Still	Walk	Run	Bike	Car	Bus	Train	Subway
Duty-cycle	2	1	0	1	4	3	2	3

A quick take-away from the presented solutions is that if aiming to be energy efficient, we should always use the accelerometer and potentially use the magnetometer if we suspect the user is in a vehicle.

In the next step we took "Solution B" and applied the duty-cycle methodology on it (Section 7.2). Both fixed and activity-specific duty-cycle lengths were tried and plotted in Fig. 16. Duty-cycle length 0 means that the sensors are always active; duty cycle length 2 means that after a window of sensor readings (1 minute in our case), the sensors turn off for the duration of two windows. We can see that the Pareto front is mostly linear in both cases, decreasing the accuracy for roughly 1.6 percentage points (p.p.) for each additional increment of the average duty-cycle length by 1. However, for the same average duty-cycle length, the activity-specific assignment achieves roughly 1.5 p.p. higher accuracy in all cases.

In all the solutions, the same pattern appears: when in a vehicle, long duty-cycles are used, as these activities have infrequent transitions. When in one of the "hub" activities – *Still, Walk* – a short cycle is used for the opposite reasons. When running, a short duty-cycle is used due to this activity being short on average. An example solution with an average duty-cycle length of 2 is shown in Table 14.

Taking it all together, using the "Solution B", the average duty-cycle length of 2 (Table 14) and the sampling frequency of 50 Hz, we lose roughly 5% (going from 90% to 85%) of the classification accuracy in exchange for using only 5% of all provided data.

As the additional post-processing step, one can use the HMM smoothing on the predictions generated by this solution. Doing so brings the accuracy to 89.9%, roughly 7 p.p. less than the best solution (97.2%) that uses the HMM smoothing.

Another way to minimize energy consumption would be to consider the energy spent on classifying the instances. This energy is largely proportional to the time spent on the task. We thus measured the time required for the classification of a single instance for different methods (the cost of the HMM is independent of the method and not included). All the measurements were performed on a desktop computer, but it is reasonable to assume that the results on a smartphone would be roughly proportional. The deep-learning methods were tested of the CPU (and not the GPU) as related work showed that the GPU does not greatly accelerate the classification time on a smartphone [65].

The results in Fig. 17 show an exponential relationship between the classification accuracy and the required processing time. The XGBoost and GradBoost algorithms seem to present the best trade-offs between the performance and the energy consumption, losing roughly 4 p.p. of accuracy compared to the ensemble, which is three orders of magnitudes slower.

9. Conclusion and discussion

The SHL Dataset is a uniquely large and sensor-rich dataset collected in a real-life setting. It provides an open platform for creating and testing algorithms for AR and similar tasks. By containing activities not common in AR datasets, such as *Train* and *Subway*, it opens new challenges for the AR community. The SHL Challenge was an effective way to jump-start the research on the dataset, yielding the first solutions to the basic locomotion AR problem.

This paper offers a detailed description of our method, which won the SHL challenge by most accurately predicting the activities on an unlabelled dataset. It can thus be considered the reference AR method for the SHL dataset, and a good starting point for similar AR problems. The highlights of the method are the complex pipeline that includes novel pre-processing steps (such as coordinate system rotation), the comprehensive feature set, the complex feature selection method and the novel neural network architecture for deep learning. We also presented how to optimize the energy consumption of our method by adapting sensor settings to each activity. They key contribution, however, might be the extensiveness of the performed experiments, which give insights on the effectiveness of different methods on the SHL dataset and in similar domains.

The following are our key experimental observations: (**Preprocessing**) The ordering of the data before making the split for internal evaluation was a key step, as otherwise the results on the *itest* set would be too good due to overfitting, and would not translate to the *SHL Test* set. (**Window size**) The window size used for the segmentation influences both classical and DL methods similarly, i.e., a higher accuracy is achieved with longer windows. (**Sampling frequency**) The features extracted for the classical ML are less sensitive to the data sampling frequency compared to the automatic features learned by the Multi-ResNet. In both classical ML and DL cases, a frequency of at least 20 Hz was required for most accurate predictions, but surprisingly good results were achieved even with much lower frequencies (e.g. 1 Hz). (**Sensors**) Comparing sensors individually, the accelerometer data provided the highest accuracy for both the classical ML and DL methods. The accelerometer and magnetometer proved to be the best pair. However, all sensors had to be fused together to achieve the best results. (**Feature subgroups**) For the classical ML, using features either in time or frequency domains yielded roughly the same results. Using both feature groups improved the accuracy. The accuracy was further increased by adding features from the accelerometer and magnetometer that were rotated into the world coordinate system. (**Feature selection**) The feature selection results are in line with the per-sensor results - i.e., more features were kept from the sensors for which the sensor-specific models achieved higher accuracy. The three-step feature selection method found a good feature subset that did not overfit to the *ivalid* set. This increased the accuracy by roughly 5 p.p on the *itest* set. Surprisingly, on the *SHL Test* set, using all the features worked better then any tested feature subset. (**Classification models**) As expected, ensemble models (e.g., boosting and RF) performed better than single models (e.g., KNN and DT). The Multi-ResNet, using an end-to-end DL architecture, achieved an accuracy of 89.4%, on a par with the classical ML models – the best of which (GradBoost) also achieved an accuracy of 89.4%. Considering that DL has enjoyed limited success in AR so far, it may be that the research community is still developing appropriate architectures. The majority of the models got significantly better after the hyper-parameter tuning. (**Ensembles**) Building an ensemble that combined the ML and DL models significantly improved the accuracy (from 89.4% to 93.8%.). (**HMM**) The HMM smoothing also worked surprisingly well, improving the accuracy by four percentage points (from 93.8% to 97.2%). Thus, for the type of data where the activities are reasonably long-lasting (in the SHL challenge all activities last more than 12 minutes on average - Table 1), we recommend the use of this post-processing step. (**Energy efficiency**) By combining three different approaches for optimizing the energy-consumption, we found a solution that reduces the amount of data needed – and thus the expected energy to collect and process it – by 95%, while losing only 5% of accuracy.

Regarding the limitations of the presented work, the windowing and frequency experiments for the Multi-ResNet are biased towards the specific DL architecture. Different architectures might find other parameters more suitable. Also, the final and best performing feature subset (”Wrapper strict”) might be biased towards RF, since RF was used to build models for the wrapper feature-selection method. Finally, the SHL challenge data comes from a single smartphone, worn by the same user in his trouser pocket for a period of four months. Thus, the presented analysis is person dependent and the models are person-specific.

Parts of this methodology were successfully used in the past on other datasets (the deep learning component [66,67] and the classical machine learning component [6,34]), which can be an indicator of the generality of the proposed approach. Additionally, we used very similar methodology to win the 2019 version of the SHL competition [68] (this paper was about the competition in 2018). However, in the future, we plan to evaluate the models on additional subjects as the data becomes available, and on additional datasets for AR. Additionally, the Multi-ResNet can be updated with more advanced end-to-end fusion approaches like two-stream network fusion [69] or multimodal subspace clustering [70].

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Martin Gjoreski: Conceptualization, Formal analysis, Investigation, Methodology, Validation, Visualization, Software, Writing - review & editing. **Vito Janko:** Conceptualization, Formal analysis, Investigation, Methodology, Validation, Visualization, Software, Writing - review & editing. **Gašper Slapničar:** Formal analysis, Data curation, Software. **Miha Mlakar:** Formal analysis, Data curation, Software. **Nina Reščič:** Formal analysis, Data curation, Software. **Jani Bizjak:** Formal analysis, Data curation, Software. **Vid Drobnič:** Formal analysis, Data curation, Software. **Matej Marinko:** Formal analysis, Data curation, Software. **Nejc Mlakar:** Formal analysis, Data curation, Software. **Mitja Luštrek:** Conceptualization, Supervision, Writing - review & editing, Project administration, Funding acquisition. **Matjaž Gams:** Conceptualization, Supervision, Writing - review & editing, Project administration, Funding acquisition.

Acknowledgments

The authors acknowledge the financial support from the Slovenian Research Agency (Grant U2-AG-16/0672, 0287 and grant U2-RP-19/0441, 0209).

References

[1] Happiness and Satisfaction with Work Commute, (https://link.springer.com/article/10.1007/s11205-012-0003-2).

[2] H. Gjoreski, M. Ciliberto, L. Wang, F.J. Ordonez Morales, S. Mekki, S. Valentin, D. Roggen, The university of sussex-huawei locomotion and transportation dataset for multimodal analytics with mobile devices, IEEE Access (2018).
[3] S. Kozina, H. Gjoreski, M. Gams, M. Luštrek, Efficient activity recognition and fall detection using accelerometers, in: International Competition on Evaluating AAL Systems Through Competitive Benchmarking, Springer, 2013, pp. 13–23.
[4] D. Roggen, A. Calatroni, M. Rossi, T. Holleczek, K. Förster, G. Tröster, P. Lukowicz, D. Bannach, G. Pirkl, A. Ferscha, et al., Collecting complex activity datasets in highly rich networked sensor environments, in: 2010 Seventh International Conference on Networked Sensing Systems (INSS), IEEE, 2010, pp. 233–240.
[5] V. Janko, B. Cvetković, A. Gradišek, M. Luštrek, B. Štrumbelj, T. Kajtna, E-gibalec: mobile application to monitor and encourage physical activity in schoolchildren, J. Ambient Intell. Smart Environ. 9 (5) (2017) 595–609.
[6] B. Cvetković, R. Szeklicki, V. Janko, P. Lutomski, M. Luštrek, Real-time activity monitoring with a wristband and a smartphone, Inf. Fusion 43 (2018) 77–93.
[7] H. Guo, L. Chen, L. Peng, G. Chen, Wearable sensor based multimodal human activity recognition exploiting the diversity of classifier ensemble, in: Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing, ACM, 2016, pp. 1112–1123.
[8] P. Zappi, T. Stiefmeier, E. Farella, D. Roggen, L. Benini, G. Troster, Activity recognition from on-body sensors by classifier fusion: sensor scalability and robustness, in: 2007 3rd International Conference on Intelligent Sensors, Sensor Networks and Information, IEEE, 2007, pp. 281–286.
[9] H. Gjoreski, J. Bizjak, M. Gjoreski, M. Gams, Comparing deep and classical machine learning methods for human activity recognition using wrist accelerometer, in: Proceedings of the IJCAI 2016 Workshop on Deep Learning for Artificial Intelligence, New York, NY, USA, 10, 2016.
[10] D. Ravi, C. Wong, B. Lo, G.-Z. Yang, A deep learning approach to on-node sensor data analytics for mobile or wearable devices, IEEE J. Biomed. Health Inform. 21 (1) (2017) 56–64.
[11] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, et al., Gradient-based learning applied to document recognition, Proc. IEEE 86 (11) (1998) 2278–2324.
[12] S. Bhattacharya, N.D. Lane, Sparsification and separation of deep learning layers for constrained resource inference on wearables, in: Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM, ACM, 2016, pp. 176–189.
[13] M. Zeng, L.T. Nguyen, B. Yu, O.J. Mengshoel, J. Zhu, P. Wu, J. Zhang, Convolutional neural networks for human activity recognition using mobile sensors, in: 6th International Conference on Mobile Computing, Applications and Services, IEEE, 2014, pp. 197–205.
[14] J. Yang, M.N. Nguyen, P.P. San, X.L. Li, S. Krishnaswamy, Deep convolutional neural networks on multichannel time series for human activity recognition, in: Twenty–Fourth International Joint Conference on Artificial Intelligence, 2015.
[15] F. Ordóñez, D. Roggen, Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition, Sensors 16 (1) (2016) 115.
[16] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780.
[17] K. Krishna, D. Jain, S.V. Mehta, S. Choudhary, An lstm based system for prediction of human activities with durations, Proc. ACM Interact. Mob. Wearable Ubiquit. Technol. 1 (4) (2018) 147.
[18] V.S. Murahari, T. Plötz, On attention models for human activity recognition, in: Proceedings of the 2018 ACM International Symposium on Wearable Computers, ACM, 2018, pp. 100–103.
[19] B. Martin, V. Addona, J. Wolfson, G. Adomavicius, Y. Fan, Methods for real-time prediction of the mode of travel using smartphone-based gps and accelerometer data, Sensors 17 (9) (2017), doi:10.3390/s17092058.
[20] S.-H. Fang, H.-H. Liao, Y.-X. Fei, K.-H. Chen, J.-W. Huang, Y.-D. Lu, Y. Tsao, Transportation modes classification using sensors on smartphones, Sensors 16 (8) (2016) 1324.
[21] S. Reddy, M. Mun, J. Burke, D. Estrin, M. Hansen, M. Srivastava, Using mobile phones to determine transportation modes, ACM Trans. Sensor Netw. (TOSN) 6 (2) (2010) 13.
[22] S. Hemminki, P. Nurmi, S. Tarkoma, Accelerometer-based transportation mode detection on smartphones, in: Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems, in: SenSys '13, ACM, New York, NY, USA, 2013, pp. 13:1–13:14, doi:10.1145/2517351.2517367.
[23] L. Wang, H. Gjoreski, K. Muraom, T. Okita, D. Roggen, Summary of the sussex-huawei locomotion-transportation recognition challenge, in: Proceedings of the 6th International Workshop on Human Activity Sensing Corpus and Applications (HASCA2018), Springer, 2018, pp. 1521–1530.
[24] A.D. Antar, M. Ahmed, M.S. Ishrak, M.A.R. Ahad, A comparative approach to classification of locomotion and transportation modes using smartphone sensor data, in: Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers, in: UbiComp '18, ACM, New York, NY, USA, 2018, pp. 1497–1502, doi:10.1145/3267305.3267516.
[25] P. Widhalm, M. Leodolter, N. Brändle, Top in the lab, flop in the field?: Evaluation of a sensor-based travel activity classifier with the shl dataset, in: Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers, in: UbiComp '18, ACM, New York, NY, USA, 2018, pp. 1479–1487, doi:10.1145/3267305.3267514.
[26] M. Gjoreski, V. Janko, N. Reščič, M. Mlakar, M. Luštrek, J. Bizjak, G. Slapničar, M. Marinko, V. Drobnič, M. Gams, Applying multiple knowledge to sussex-huawei locomotion challenge, in: Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers, in: UbiComp '18, ACM, New York, NY, USA, 2018, pp. 1488–1496, doi:10.1145/3267305.3267515.
[27] V. Janko, N. Reščiç, M. Mlakar, V. Drobnič, M. Gams, G. Slapničar, M. Gjoreski, J. Bizjak, M. Marinko, M. Luštrek, A new frontier for activity recognition: The sussex-huawei locomotion challenge, in: Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers, in: UbiComp '18, ACM, New York, NY, USA, 2018, pp. 1511–1520, doi:10.1145/3267305.3267518.
[28] Sussex-Huawei Locomotion Challenge, (http://www.shl-dataset.org/activity-recognition-challenge).
[29] L. Wang, H. Gjoreski, M. Ciliberto, S. Mekki, S. Valentin, D. Roggen, Benchmarking the shl recognition challenge with classical and deep-learning pipelines, in: Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers, in: UbiComp '18, ACM, New York, NY, USA, 2018, pp. 1626–1635, doi:10.1145/3267305.3267531.
[30] J. Lee, J. Kim, Energy-efficient real-time human activity recognition on smart mobile devices, Mob. Inf. Syst. 2016 (2016). Article ID 2316757, 12 pages
[31] B. Sefen, S. Baumbach, A. Dengel, S. Abdennadher, Human activity recognition, in: Proceedings of the 8th International Conference on Agents and Artificial Intelligence, SCITEPRESS-Science and Technology Publications, Lda, 2016, pp. 488–493.
[32] J.B. Kuipers, et al., Quaternions and rotation sequences, 66, Princeton university press Princeton, 1999.
[33] X. Su, H. Tong, P. Ji, Activity recognition with smartphone sensors, Tsinghua Sci. Technol. 19 (3) (2014) 235–249, doi:10.1109/TST.2014.6838194.
[34] B. Cvetković, V. Janko, M. Luštrek, Demo abstract: Activity recognition and human energy expenditure estimation with a smartphone, in: Pervasive Computing and Communication Workshops (PerCom Workshops), 2015 IEEE International Conference on, IEEE, 2015, pp. 193–195.
[35] tsfresh, (http://tsfresh.readthedocs.io/en/latest/).
[36] Mutual info score, (http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.mutual_info_classif.html).
[37] Pearson correlation coefficient, (https://en.wikipedia.org/wiki/Pearson_correlation_coefficient).
[38] scikit-learn, (https://scikit-learn.org/stable/).
[39] T.K. Ho, Random decision forests, in: Proceedings of 3rd International Conference on Document Analysis and Recognition, 1, IEEE, 1995, pp. 278–282.
[40] J.R. Quinlan, Improved use of continuous attributes in c4. 5, J. Artif. Intell. Res. 4 (1996) 77–90.
[41] S.J. Russell, P. Norvig, Artificial intelligence: A modern approach, Malaysia; Pearson Education Limited, 2016.
[42] D.W. Aha, D. Kibler, M.K. Albert, Instance-based learning algorithms, Mach Learn 6 (1) (1991) 37–66.
[43] J. Shawe-Taylor, N. Cristianini, An introduction to support vector machines and other kernel-based learning methods, 204, Cambridge University Press Cambridge, 2000.
[44] L. Breiman, Bagging predictors, Mach. Learn. 24 (2) (1996) 123–140.
[45] F. Rosenblatt, Principles of neurodynamics. perceptrons and the theory of brain mechanisms, Technical Report, Cornell Aeronautical Laboratory, 1961.
[46] G. Martínez-Muñoz, A. Sánchez-Martínez, D. Hernández-Lobato, A. Suárez, Class-switching neural network ensembles, Neurocomputing 71 (13–15) (2008) 2521–2528.
[47] G. Martínez-Muñoz, A. Suárez, Switching class labels to generate classification ensembles, Pattern Recognit. 38 (10) (2005) 1483–1494.
[48] S. Ioffe, C. Szegedy, Batch normalization: accelerating deep network training by reducing internal covariate shift, arXiv Preprint arXiv:1502.03167 (2015).
[49] V. Nair, G.E. Hinton, Rectified linear units improve restricted boltzmann machines, in: Proceedings of the 27th International Conference on Machine Learning (ICML-10), 2010, pp. 807–814.
[50] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern recognition, 2016, pp. 770–778.
[51] Z. Wang, W. Yan, T. Oates, Time series classification from scratch with deep neural networks: A strong baseline, in: 2017 International Joint Conference on Neural Networks (IJCNN), IEEE, 2017, pp. 1578–1585.
[52] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber, et al., Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, IEEE Press, 2001.
[53] hmmlearn, (https://hmmlearn.readthedocs.io/en/latest/).
[54] V. Janko, M. Luštrek, Energy-efficient data collection for context recognition, in: Proceedings of the 2017 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2017 ACM International Symposium on Wearable Computers, ACM, 2017, pp. 458–463.
[55] V. Janko, M. Luštrek, Choosing duty-cycle parameters for context recognition, in: 2018 14th International Conference on Intelligent Environments (IE), IEEE, 2018, pp. 83–86.
[56] A. Carroll, G. Heiser, et al., An analysis of power consumption in a smartphone., in: USENIX Annual Technical Conference, 14, Boston, MA, 2010, p. 21.
[57] I. Crk, F. Albinali, C. Gniady, J. Hartman, Understanding energy consumption of sensor enabled applications on mobile phones, in: Engineering in Medicine and Biology Society, 2009. EMBC 2009. Annual International Conference of the IEEE, IEEE, 2009, pp. 6885–6888.
[58] N.D. Lane, S. Bhattacharya, A. Mathur, P. Georgiev, C. Forlivesi, F. Kawsar, Squeezing deep learning into mobile and embedded devices, IEEE Pervasive Comput. 16 (3) (2017) 82–88.
[59] M. Tan, Q.V. Le, Efficientnet: rethinking model scaling for convolutional neural networks, arXiv Preprint arXiv:1905.11946 (2019).
[60] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: nsga-ii, IEEE Trans. Evol. Comput. 6 (2) (2002) 182–197.

[61] A. Khan, N. Hammerla, S. Mellor, T. Plötz, Optimising sampling rates for accelerometer-based human activity recognition, Pattern Recognit. Lett. 73 (2016) 33–40.
[62] M. Schuster, K.K. Paliwal, Bidirectional recurrent neural networks, IEEE Trans. Signal Process. 45 (11) (1997) 2673–2681.
[63] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using rnn encoder-decoder for statistical machine translation, arXiv preprint arXiv:1406.1078 (2014).
[64] C. Ito, X. Cao, M. Shuzo, E. Maeda, Application of cnn for human activity recognition with fft spectrogram of acceleration and gyro sensors, in: Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers, in: UbiComp '18, ACM, New York, NY, USA, 2018, pp. 1503–1510, doi:10.1145/3267305.3267517.
[65] S. Richoz, A. Perez-Uribe, P. Birch, D. Roggen, Benchmarking deep classifiers on mobile devices for vision-based transportation recognition, in: Proceedings of the 2019 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2019 ACM International Symposium on Wearable Computers, ACM, 2019, pp. 803–807.
[66] M. Gjoreski, S. Kalabakov, M. Luštrek, H. Gjoreski, Cross-dataset deep transfer learning for activity recognition, in: Proceedings of the 2019 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2019 ACM International Symposium on Wearable Computers, ACM, 2019, pp. 714–718.
[67] G. Slapničar, N. Mlakar, M. Luštrek, Blood pressure estimation from photoplethysmogram using a spectro-temporal deep neural network, Sensors 19 (15) (2019) 3420.
[68] L. Wang, H. Gjoreski, M. Ciliberto, P. Lago, K. Murao, T. Okita, D. Roggen, Summary of the sussex-huawei locomotion-transportation recognition challenge 2019, in: Proceedings of the 2019 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2019 ACM International Symposium on Wearable Computers, ACM, 2019, pp. 849–856.
[69] C. Feichtenhofer, A. Pinz, A. Zisserman, Convolutional two-stream network fusion for video action recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 1933–1941.
[70] M. Abavisani, V.M. Patel, Deep multimodal subspace clustering networks, IEEE J. Sel. Top. Signal Process. 12 (6) (2018) 1601–1614.