DeepMatch: Deep Matching for In-Vehicle Presence Detection in Transportation

Magnus Oplenskedal
Norwegian University of Science and Technology (NTNU)
Trondheim, Norway
Forkbeard Technologies
Oslo, Norway
magnukop@ntnu.no

Amir Taherkordi
University of Oslo
Oslo, Norway
amirhost@ifi.uio.no

Peter Herrmann
Norwegian University of Science and Technology (NTNU)
Trondheim, Norway
herrmann@ntnu.no

ABSTRACT

A key feature of modern public transportation systems is the accurate detection of the *mobile context* of transport vehicles and their passengers. A prominent example is automatic in-vehicle presence detection which allows, *e.g.*, intelligent auto-ticketing of passengers. Most existing solutions, in this field, are based on either using active RFID or Bluetooth Low Energy (BLE) technology, or mobile sensor data analysis. Such techniques suffer from low spatiotemporal accuracy in in-vehicle presence detection. In this paper, we address this issue by proposing a deep learning model and the design of an associated generic distributed framework. Our approach, called DeepMatch, utilizes the smartphone of a passenger to analyze and match the event streams of its own sensors and the event streams of the counterpart sensors in an in-vehicle reference unit. This is achieved through a new learning model architecture using Stacked Convolutional Autoencoders for feature extraction and dimensionality reduction, as well as a dense neural network for stream matching. In this distributed framework, feature extraction and dimensionality reduction is offloaded to the smartphone, while matching is performed in a server, *e.g.*, in the Cloud. In this way, the number of sensor events to be transmitted for matching on the server side will be minimized. We evaluated DeepMatch based on a large dataset taken in real vehicles. The evaluation results show that the statistical accuracy of our approach is 0.978 for in-vehicle presence detection which, as we will argue, is sufficient to be used in, *e.g.*, auto-ticketing systems.

CCS CONCEPTS

• **Software and its engineering → Publish-subscribe / event-based architectures**; • **Computing methodologies → Supervised learning by classification**; • **Human-centered computing → Ubiquitous and mobile devices**.

DEBS '20, July 13–17, 2020, Virtual Event, QC, Canada

ACM ISBN 978-1-4503-8028-7/20/07…$15.00
https://doi.org/10.1145/3401025.3401741

KEYWORDS

Mobile Context, Sensor Event Streams Analysis, Deep Learning, Event Matching, Intelligent Transportation

ACM Reference Format:
Magnus Oplenskedal, Amir Taherkordi, and Peter Herrmann. 2020. DeepMatch: Deep Matching for In-Vehicle Presence Detection in Transportation. In *The 14th ACM International Conference on Distributed and Event-based Systems (DEBS '20), July 13–17, 2020, Virtual Event, QC, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3401025.3401741

1 INTRODUCTION

Within logistics and public transportation, there is a strong need for accurate and intelligent detection of mobile context situations of users, smart devices and vehicles. By *mobile context*, we refer to any kind of information that can be used to characterize spatiotemporal properties of a mobile entity [12]. An example is the position data of a moving vehicle making it possible to find out its current location. Public transportation providers in many countries (*e.g.*, in Northern Europe) are providing smartphone applications for their passengers, in which the user can, *e.g.*, purchase tickets and get route guidance. A key feature to enhance the next generation of these mobile context-aware applications is the integration of information about the presence of a passenger in a vehicle. An advantage of this extension is that it becomes easier to determine the exact flow of passengers between particular places. This makes it possible to plan optimal public transport networks in which passengers are offered rides when they need them and are brought to their destinations without the need to change vehicles often if at all. Further, peak hours can be detected and the supply of vehicles is optimized accordingly.

If the in-vehicle presence detection is highly accurate, we can also make the ticketing of passengers considerably simpler. In existing smartphone applications, the passengers have to remember buying tickets before starting a ride. Moreover, in order to buy the right ticket for the intended trip, they need to have fair knowledge about the ticketing system of the transport provider. In contrast, using a highly accurate in-vehicle presence detection solution, a so-called Be-In/Be-Out (BIBO) system [20], tickets can be automatically issued to the passengers based on the exact duration of their ridings on vehicles. Thus, they can conveniently enter and leave public transport vehicles without having to deal with the transportation provider in advance. If this seamless way of travel

is accepted by many passengers, there is no need for cost intensive ticket checkpoints, ticket machines, and passenger controls anymore.

Existing solutions for in-vehicle presence detection are based on two different approaches. The first group applies communication systems such as Radio Frequency Identification (RFID) or Bluetooth Low Energy (BLE). While travelling, temporary connections are built up between the user's mobile device and certain fixed vehicle equipment through which evidence can be established that the passenger is within the vehicle. Prominent examples for such systems are EasyRide [8] and SEAT [22]. Approaches in the other group use event streams from smartphone sensors to analyze for certain properties. Modern smartphones are provided with a variety of sensors such as magnetometers, accelerometers, gyroscopes, GPS, and barometers which offer unprecedented opportunities to analyze mobile context information from the user's environment. Two prominent solutions for sensor-based analysis are HybridBaro [27] and RideSense [19]. We will argue later that the accuracy of both categories above is still not high enough.

In this paper, we address the accuracy problem and propose a highly autonomous approach that can detect in-vehicle presence with a sufficient degree of precision. A central aspect of our approach is to equip each vehicle with a *reference device* (*e.g.*, an Android phone). This allows us to deduce the presence of passengers in the vehicle based on the match between the stream of events generated by sensors of the reference device and those measured by the sensors of the passengers' smartphones. We propose the system DeepMatch which provides a deep learning model to verify in-vehicle presence. The new learning architecture is based on Stacked Convolutional Autoencoders, used for feature extraction and dimensionality reduction. Matching is provided using a fully connected neural network. DeepMatch is a distributed framework, where feature extraction and dimensionality reduction is offloaded to the users' smartphones, while the matching process is performed in a server, *e.g.*, a cloud server. Using Stacked Convolutional Autoencoders, our model not only learns the most essential features of the sensed data stream (*i.e.*, encoded data), but also finds out which part of the stream can be omitted without deteriorating matching. The part of the model running on a server compares the input encoded data of a smartphone with that of the corresponding reference device and outputs a value indicating the probability of the two data sources being present in the same vehicle. The server can be realized as a centralized unit out of the vehicle. Alternatively, it can be locally installed in the vehicle, *e.g.*, in the reference device, or on a local router in the vehicle following the principle of fog computing [1].

The model is trained on real data traces gathered by a group of people using public transportation in two large Norwegian cities. The evaluation results show that the statistical accuracy, the so-called F1-score, of DeepMatch is 0.978 for in-vehicle presence detection outperforming the two well-known technologies Normalized Correlation by 4%, and Dynamic Time Warping by 16%. This can provide a significant advantage for practical in-vehicle presence detection as we will discuss later.

The rest of this paper is organized as follows. In Section 2, we discuss existing solutions followed the presentation of our in-vehicle presence detection approach in Section 3. In Section 4, the experimental evaluation results are reported. We conclude the paper with a discussion of our future plans in Section 5.

2 RELATED WORK

As mentioned above, existing solutions for in-vehicle presence detection are based on either utilizing communication technologies or analyzing mobile sensor events. These approaches are presented in this section followed by the discussion of some recent works leveraging deep learning for mobile context detection.

2.1 Communication Technology-based Solutions

Early in-vehicle presence detection systems were based on active RFID tags, carried by the passengers, and a single communication unit in the center of vehicles. A contactless, mid-range radio-based identification and communication protocol was used for tracking. One of the first solutions was EasyRide [8], developed by the Swiss Railways Association. Allfa [7] is another RFID-based system, tested for half a year in busses, trams and trains in Dresden, Germany. Due to the weak transmitter strengths of the active tags, it is difficult to guarantee that all of them are detected in the vehicle. As discussed in [8], this affords a vast number of readers in the vehicle, at least one at each door. However, such approaches still suffer from lack of enough detection precision. For instance, Allfa has an accuracy rate of just 68% making it unsuitable for practical use.

Another category is based on BLE. Compared with active RFID approaches using battery-powered tags, BLE-based BIBO systems can utilize smartphones with additional monitoring options, the possibility to measure signal strengths for proximity determination, larger distribution channels, etc. The first BLE-based solution is proposed in [20], while [14] suggests a ticketing system adding a custom profile on top of the BLE to fulfill the payment procedure. In SEAT [22], a BLE-enabled smartphone communicates with devices installed in vehicles to track the journey for automatic pricing. Its main focus is on security, performance, and battery friendliness, but not on the accuracy of the in-vehicle presence detection. The authors of [20] are cautiously optimistic that BLE might work for BIBO systems. However, the chassis of a vehicle does not limit the accessibility of a BLE transmitter which makes it possible that somebody close to it, *e.g.*, a person in another vehicle, is wrongly detected. On the other hand, things in a vehicle may inhibit a BLE connection such that devices in the vehicle are not detected. This is confirmed by the authors of [15] when using BLE for indoor localization. While precise indoor location seems to be more complex than "just" finding out if somebody is in a vehicle, we expect similar accuracy problems.

2.2 Mobile Sensor Event Analytics-based Solutions

Existing work in this field analyzes the data stream of the sensors in user smartphones to detect mobile contexts in transportation. Our extensive experiments and the obtained results, reported in Section 4.4.1, show that the smartphone barometer is the only useful sensor for in-vehicle presence detection since the position and orientation of the phone as well as the movements of its carrier

influence the measurements of other sensors. This is also confirmed by related approaches that mostly focus on analyzing the barometer data. In [21], Sankaran et al. demonstrate that the barometer can be applied to detect user activities of IDLE, WALKING, and VEHICLE at low-power through their context-detection algorithm, using four stages; pre-processing, jump detection, peak detection and walk detection. Likewise, in [25], user activities are classified using the barometer sensor on smartphones. This approach leverages Bayesian networks, decision trees, and RNN as inference models to predict user action, *e.g.*, riding or leaving a cable-car. In [10], the authors demonstrate how the pressure data collected from a smartphone barometer can be utilized to accurately track driving patterns. An expansion is to correlate pressure time-series data sensed by the barometer against topographic elevation data and road maps for a given region. This allows a centralized server to estimate the possible routes through which users have driven. An example is HybridBaro [27], featuring a hybrid algorithm to adaptively utilize GPS data to increase the detection accuracy in flat areas. RoadSphygmo [5] uses the barometer in smartphones to detect traffic congestion. RideSense [19] is aimed to match a passenger's sensor trace against the traces of buses to determine the riding and leaving times.

The important finding of the above approaches is that the barometer can be used as a reliable sensor on smartphones for mobile context detection scenarios. However, they all require continuous sensor event measurements and transmission. Further, while better than RFID and BLE, the accuracy promised by these approaches is still not good enough to fulfill the demands of transportation systems. For example, the accuracy of RideSense for more than 20 hours of traces from five bus lines is between 84 to 98%. As pointed out in Section 4.5, only the uppermost value of 98% would be sufficient for using this technology in practice. Lower levels of accuracies are not acceptable considering the large number of daily trips made through different public transport modes in a city, *e.g.*, 950,000 daily trips in Oslo as an average sized city.

2.3 Mobile Sensor Events and Deep Learning

In some recent works, deep learning has been leveraged to analyze sensor events for detecting mobile contexts. In [26], the authors report on the accuracy of models such as RNN, CNN, various Hybrid models, Restricted Boltzman Machines, and Autoencoders with respect to their ability to classify human activities from body-worn sensors. They conclude that, compared to traditional pattern recognition methods, deep learning reduces the dependency on human-crafted feature extraction and achieves better performance by automatically learning high-level representations of the sensor events. The authors also state that, from a technical viewpoint, there is no model outperforming all the others in general. Thus, they recommend to choose the models based on the requirements of specific scenarios. *DeepSense* [28] uses CNN and RNN to provide an estimation and classification framework for car tracking with motion sensors and human activity recognition. In [30], *DeepSleepNet*, a deep learning framework for automatic sleep stage scoring based on electroencephalogram data, is proposed. The authors show that the model automatically learns features for different datasets without utilizing any hand-engineered features. The model achieves

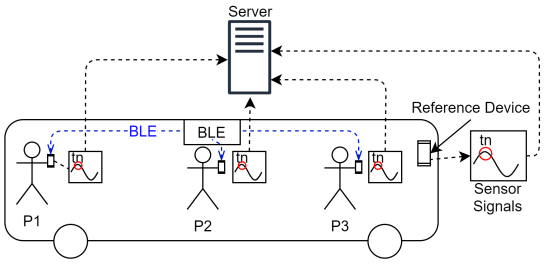

Figure 1: A sample scenario presenting DeepMatch.

an accuracy that is similar to the state-of-the-art methods using hand-engineering. From the ML-based stream matching perspective, StreamLearner [18] is a distributed Complex Event Processing (CEP) system proposed for scalable and low-latency event detection on streaming data that uses neural networks. StreamLearner is mainly designed for systems with multiple event sources causing diverse patterns in the event streams. Its case study is detecting anomalies (*i.e.*, abnormal sequences of sensor events) in smart factories.

The important finding of most works in this category is that deep learning can outperform hand-crafted feature extraction methods when applied to mobile sensor event streams. This can be used to deduce valuable information about the mobile context. We aim to exploit this power of deep learning in DeepMatch to build a model capable of highly accurate in-vehicle presence prediction solely based on sensor event streams. In addition, the limited work carried out on ML-based sensor stream matching, is more focused on the quality of stream matching, *e.g.*, to provide higher throughput.

3 DEEPMATCH

In this section, we first provide an overview of our approach. Then, we explain the hardware and software settings on which DeepMatch is built, followed by the presentation of our mobile stream data analysis and matching approach and a detailed description of the associated design and architecture model.

3.1 Overview

Figure 1 shows a simple scenario that we use to outline our approach. Three passengers are traveling with a bus. Everybody carries a smartphone with an app featuring the DeepMatch learning model. As fixed equipment, the bus is provided with a BLE-transmitter and a reference device (RefDev) that uses the same type of sensors as found on the smartphones. When a passenger enters the bus, the mobile app is awoken by the OS based on detecting the BLE signal. The application then immediately starts to retrieve sensor data. Moreover, it performs feature extraction converting the sensed data to a lower dimensional representation. The compressed version of the data is timestamped and tagged with the ID of the BLE signal awakening the application, before it is transmitted to a remote server. Simultaneously, RefDev is measuring, transforming and transmitting event streams of its own sensors to the same server. Thus, the server receives two sets of data that are compared to infer whether the two sensors are in the same vehicle. For that, a special module carrying out the matching analysis is employed. In

Section 3.4, we explain the learning model used for the matching analysis in detail. If the result of this analysis is that two data sets with the same BLE-transmission ID are collected in the same vehicle, and one of them is produced by RefDev, the person carrying the smartphone producing the other one is assumed to be in the same vehicle.

3.2 Hardware Requirements and System Settings

As indicated above, a RefDev equipped with equal sensors as a typical smartphone, is required to collect the same sensor events with an identical frequency. The RefDev is used to provide a ground truth for the in-vehicle presence detection. Through our empirical experiments, we found out that using only the barometric sensor provides both the best matching accuracy as well as a very low power consumption. This is discussed in Sections 4.4.1 and 4.8. The barometric sensors was initially introduced in smartphones to reduce GPS delay by providing the z coordinate. As described in Section 2, this sensor can also be applied to provide highly accurate contextual information. It guarantees position-independence, resistance to vibrations, and high sensitivity to changes in elevation that are properties of high value to implement the matching process (see [21]). *Position-independence*, *i.e.*, the sensor's ability to provide useful data independently of the sensor's location, is particularly important for underground transportation in tunnels, subways and trains, where for instance GPS is very inaccurate. *Vibration resistance* is important for the ability to measure the movements of the vehicle rather than the movements of the user. With respect to this property, the barometer clearly outmatches the accelerometer and gyroscope sensors, which are often more sensitive to the movements of a user's hands than to those of the vehicle. This results in the fact that DeepMatch with the barometer renders a precision of 97.8% while, with the two other sensors, only around every other matching is correctly detected (see Table 4). Finally, a high *elevation sensitivity* is critical for extracting useful context data in flat terrain, as demonstrated in Section 4.6. In [10], Bo-Jhang H. et al. report that relative pressure sensitivity for the Bosch BMP280 sensor used in the iPhone 6 and Nexus 5 is sensitive to elevation changes of 10 to 20 cm, even better than the specified vertical resolution of about one meter reported by Bosch in [2].

Besides the RefDev, the vehicle is provided with a BLE transmitter that, in contrast to the communication technology-based approaches discussed in Section 2, is not directly used for in-vehicle detection. Instead, its task is to wake up the app in the passenger's smartphone when entering a vehicle as well as to align the data produced by this smartphone with those sensed by the RefDev. Both Android and iOS provide the ability to start "sleeping" applications when a BLE-signal with a pre-defined ID is detected. Thus, our application only turns on and collects the sensor events when the phone is close to a BLE-transmitter registered in the application. Due to the imprecise nature of BLE, a transmitter may not only be readable in its own vehicle but also in its environment. In this case, *e.g.*, in a bus terminal, a smartphone may read several BLE transmitter inputs simultaneously. The IDs of these BLE transmitters are sent together with the collected data to the server. In this way, the service running on the server does not need to compare the user data with those of all RefDevs in the transport network, but only with those related with detected BLE transmitters. This significantly reduces the workload of the server. Further, if we use local servers in the vehicles, *e.g.*, letting RefDev conducting matching, the BLE transmitter can be used to forward the data from the user's smartphones to the server.

Table 1: Example datapoints

Sensor	Value	Timestamp	Trip	Device
Accelerometer	0.117311	3366...	15	75i3...
Magnetometer	21.835773	3366...	15	75i3...
Gyroscope	0.059957	3366...	15	75i3...
Barometer	993.281097	3366...	15	75i3...

If a vehicle enters a *dead spot*, *i.e.*, an area with no cellular network coverage, and we use a central server, the encoded data will be temporarily stored on the device and tagged with timestamps and BLE IDs. When the vehicle leaves the dead spot, the locally stored data will then be transmitted to the server for a delayed in-vehicle presence detection.

3.3 Mobile Data Analysis

The deep learning model of DeepMatch performing the in-vehicle prediction has to be trained based on real sensor events collected from RefDev and passenger devices. In this subsection, we describe how the real sensor events are collected and converted to the training and evaluation datasets used to train the model.

3.3.1 Data Collection and Preprocessing. The sensor events used to train our deep learning model are collected by means of the *DataCollector*, an Android application that we developed for this purpose. The application can be configured to listen to events from any available sensor in the smart device, and to store and timestamp them locally as datapoints, see Table 1. The data from various runs can then be uploaded to a computer running our *Data Analysis* tools. Moreover, the application contains a simple server-client communication protocol using websockets. This allows us to connect several devices and to synchronize their clocks. In this way, the collection of sensor events can be carried out synchronously. The data collection is performed between two stops along the route of a public transportation provider, where all datapoints collected between the two stops are stored as a *Trip*. All trips are registered with a unique *trip* ID, propagated from the server device to all clients.

The sensor framework provided by Android allows developers to determine the sampling rate of each available sensor. The sensors generate events, using this sampling rate as a guideline, usually with a standard deviation of one to two milliseconds. To perform sensor event matching, however, we need a fixed sampling rate across all sensors and devices for a trip. This is achieved through our *Data Analysis* tool by *interpolating* the data collected by each device individually. The interpolation of a trip's data is done by first defining a global start time extracted from the data. Thereafter, this start time is substracted from the timestamps of all datapoints to get a relative timestamp. In the next step, we interpolate the values for each sensor event set with a fixed frequency, and finally remove the original data. With these fixed timestamp and interpolated values,

Table 2: An Example of interpolated data

Timestamp	Accel.	Magneto.	Barom.	Gyrosc.
0 ms	5.62421	21.83577	989.28109	0.05995
20 ms	5.58418	22.83491	989.28610	0.13596
40 ms	5.53032	24.54790	989.27981	0.07716
60 ms	5.67377	25.12537	989.26586	0.08019

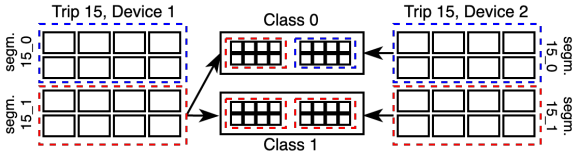

Figure 2: Matching samples created from trip segments.

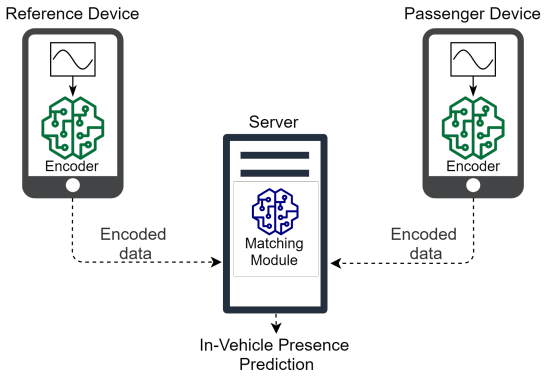

Figure 3: Overview of the DeepMatch model design.

we can now create a new table where the rows represent timestamps and each column contains the value for the given timestamp.

3.3.2 Dataset Creation. An important goal of DeepMatch is to minimize the amount of data needed to perform in-vehicle detection which reduces the amount of data to be transmitted between the devices and the server as well as the number of calculations performed by the server. To this end, we trained our model to perform predictions based on smaller *segments* of the trip data. Our *Data Analysis* tool converts the interpolated data from a trip, shown in Table 2, into *trip segments* by splitting the trip data into smaller segments of a fixed length. Furthermore, all segments are tagged with the ID of the trip they belong to, in addition to a segment number following the naming convention *<trip id>_<segment nr>*, *e.g.*, the first segment of a trip with id 15 becomes 15_0 and the second 15_1. This will be the same for all devices used to gather data for Trip 15. The tool allows us to configure the length of the segments freely to find out which one renders the best matching results. However, when applying the tool to train and use the deep learning model, all segments must have the same length.

The created segments are used to build samples for a *matching dataset*. The samples in this dataset belong to either *Class 1* or *Class 0*. *Class 1* consists of samples from segments with the same trip id and segment number, *i.e.*, sensor events captured by two devices at the same time in the same vehicle. Samples from *Class 0* are created from segments with different trip ids or segment numbers. They represent sensor events not captured at the same time or in the same vehicle, as shown in Figure 2.

3.4 Design and Architecture of the Learning Model

The main goal of the DeepMatch learning model is to perform feature extraction, dimensionality reduction, and matching. As already mentioned, the overall in-vehicle presence detection process will be performed in a distributed fashion that is depicted in Figure 3. The feature extraction and dimensionality reduction take place both in the smartphones of the passengers and the reference devices fixed in the vehicles. They are performed by *Encoder Modules*, which are shown in form of green networks in Figure 3. These encoders reduce the size of the original sensor events stream by a factor of four. In consequence, the bandwidth necessary to transmit the sensor data from the devices to the server will be reduced to a fourth in comparison to sending all the originally sensed data. The main objective of the encoder is to guarantee the preservation of characteristics and features of the data necessary for accurate matching.

The encoder is part of a neural network topology, called *Autoencoder* [6]. It is composed of two parts, an *encoder* and a *decoder*. Autoencoders are used to learn efficient, often lower-dimensional representations of their input through unsupervised training. The encoder maps the autoencoders input to a *latent representation* in *latent space*, *i.e.*, an internal representation of its input. The decoder maps this latent representation to a reconstructed representation of the Autoencoder's original input. The amount of information passed from the encoder to the decoder is typically restricted, forcing the Autoencoder to prioritize the most relevant information in its input. In DeepMatch, we use dimensionality reduction to restrict the encoder in order to achieve the size reduction by the factor four.

The matching predictions are performed on the server by a fully connected deep neural network, called the *Matching* module, depicted as a blue network in Figure 3. To achieve a high in-vehicle presence detection accuracy, this module has to learn and fine-tune the spatiotemporal thresholds to distinguish the samples in *Class 1*, *i.e.*, segments taken in the same vehicle at the same time, from those in *Class 0*, *i.e.*, segments sensed during different trips or at different locations.

The Matching module and the Autoencoder are developed and trained jointly using the architecture shown in Figure 4. Different types of Autoencoders exist. In DeepMatch, we use a *Stacked Convolutional Autoencoder* (CAE) [17] in which the *encoder* is created from *stacks* of alternating convolutional (conv) and maxpool layers. The conv layers are responsible for feature extraction and the maxpool layers for dimensionality reduction.

As previously mentioned, the *decoder* is the part of the Autoencoder responsible to recreate a copy of its input from the latent representation output by the encoder. It is created from stacks of alternating conv and upsample layers. Conv layers are specially suited to detect and extract time-invariant features in sequences,

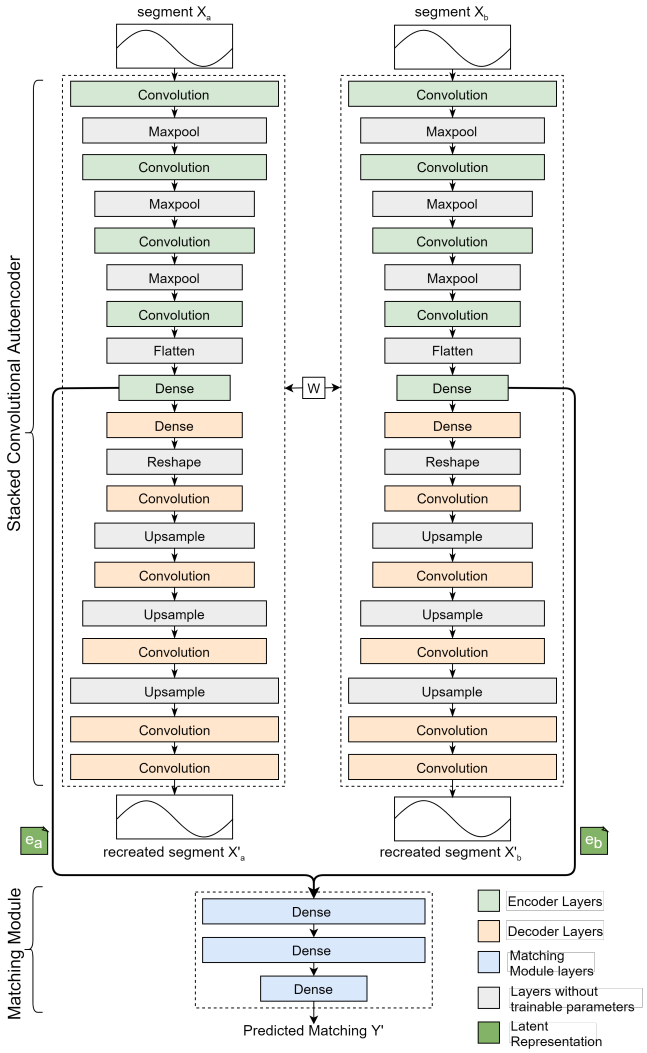

Figure 4: DeepMatch model architecture.

see [16, 17, 23, 26]. The maxpool layers perform dimensionality reduction using the *max* operator. The upsampling layers reverse this process by duplicating each value in its input sequence, *e.g.*, the sequence 1, 2, 3 would become 1, 1, 2, 2, 3, 3.

In Fig. 4, the specifics of our deep model are shown. The green boxes represent the layers of the encoder and the orange ones the layers of the decoder. The grey boxes show layers without trainable parameters and the blue ones the layers of the Matching module. Each convolution layer represents the following three operations sequentially: convolution, rectified linear unit activation (ReLU, *i.e.*, $relu(x) = max(0, x)$) and batch normalization [11]. Every other layer in the encoder is a maxpool layer, using a stride size of 2, and every other layer in the decoder is a upsample layer, with size of 2, doubling the size of their input. The task of the flatten layer is to reshape any N-dimensional input to an 1-dimensional output, whilst the reshape layer reverse this process. In our model, the encoder consists of four convolutional layers, three maxpooling layers, one flatten layer and one dense layer. The decoder consists of five convolutional layers, three upsample layers, one reshape layer and one dense layer. The last part of the learning model in DeepMatch, the Matching module, consists of three consecutive fully connected dense layers, all using ReLU activation and batch normalization.

The DeepMatch model is distributed amongst the server, the reference device, and the passenger devices. The encoder module is embedded in the smartphones and reference devices whilst the Matching module is implemented in the server. The decoder module is only used during training and not in the execution of in-vehicle presence detection.

To train the overall model depicted in Fig. 4, the CAE is duplicated, sharing all trainable parameters W in a network topology known as a *Siamese Architecture*. This architecture has been applied with great success in matching problems like face recognition [13], signature verification [3], and human identification using gait recognition [29]. The Siamese architecture allows the model to accept two sensor data segments at the same time, *e.g.*, segment X_a and X_b. Since the two CAEs share the same weights, the encoder performs an identical mapping of the segments. Therefore, if the segments are matched (*i.e.*, they belong to a sample of *Class 1*), the latent representations e_a and e_b should also be matched. Likewise, e_a and e_b should be different for samples belonging to *Class 0*. Through joint training of both the CAE and the Matching module, the encoder learns to prioritize both, features of the segments that are necessary for the decoder to recreate them, and features needed by the Matching module for matching.

3.5 Model Training

This subsection describes the training routine for the model shown in Fig. 4. We describe two sensor data segments belonging to a matching sample as X_a and X_b (see Section 3.3.2) while the binary label Y refers to the ground truth class of a sample, *i.e.*, $Y = 1$ for *Class 1*, and $Y = 0$ for *Class 0*. Through the encoder layers of the Siamese CAEs, X_a and X_b are mapped to lower-dimensional latent representations e_a and e_b, shown as dark green squares in Fig. 4. Thereafter, we map e_a and e_b through the decoder layers which results in the segment recreations X'_a and X'_b. Finally, we feed X'_a and X'_b to the Matching module which returns the class prediction Y', *i.e.*, $Y' = 1$ if the segments are matched and $Y' = 0$ otherwise.

The goal of the model training, of course, is to reduce the disagreement between the ground truth label Y and the class prediction Y' for as many samples as possible. To achieve that, we also need to reduce the disagreement between the original segments X_a and X_b and the recreated ones X'_a and X'_b. To quantify the disagreements between original and recreated segments, we use *Mean Squared Error*:

$$L = \frac{1}{n}\sum_{t=1}^{n}(X'_a[t] - X_a[t])^2$$

Here, n is the overall time span of segment X_a while $X'_a[t]$ is the recreation of the datapoint $X_a[t] \in X_a$ at the point of time t. As a loss function for the Matching module to quantify disagreements between Y and Y', we apply *Binary Cross Entropy*:

$$L = -Y \cdot log(Y') + (1 - Y) \cdot log(1 - Y')$$

Y' is the predicted label of the sample containing segments X_a and X_b, and Y its ground truth.

The disagreements found by the loss functions described above are used to update the trainable parameters of the model through Stochastic Gradient Descent. We emphasize that the gradients from *both* loss functions are backpropagated to the encoders. This enables the encoders to extract not only the most defining features of its input, but also the features relevant for matching prediction.

3.6 Design Rationale behind the DeepMatch Model

To find out the best model, we conducted hundreds of experiments on various model configurations. Every configuration was evaluated using the performance metrics described in Section 4.1 on the dataset described in Section 4.2. To obtain a useful model architecture, we tried increasing as well as decreasing the number of convolutional layers in the CAEs and swapping the convolutional layers for dense layers. Moreover, we tried multiple variants of the Matching module, using convolutional layers instead of dense layers, varying the size and number of dense layers, and also exchanging the Matching module with a function calculating the Euclidean Distance between the latent representations and using this for matching predictions. We tried stacking convolutional layers as feature extractors instead of using Autoencoders, removing the need for loss calculations between the input and recreated segments. In addition to different model architectures, we tested various hyperparameter settings such as adjusting the number and sizes of filters in each conv layer, and trying various output sizes on the dense layers of the Matching module. From all our experiments, the architecture in Fig. 4, using the hyperparameter settings described in 3.4, rendered the best performance.

All experiments (*i.e.*, training and evaluation) were performed on a desktop PC with an Intel i7 4.00GHz CPU, 16 GB memory, and a Nvidia GTX 1080 GPU. The models were created, trained and evaluated using Google Tensorflow 2.0, version 2.0.0-rc0 [24].

4 EVALUATION

In this section, we first describe the performance metrics chosen to evaluate our learned models. Thereafter, we explain how the data used during training and evaluation was collected and preprocessed. Moreover, we show the performance results from seven sensor modality variations of our model. For that, we investigated not only the barometer but also accelerometer, magnetometer, and gyroscope sensors as well as various combinations. Further, the performance results for DeepMatch 5, DeepMatch 10 and DeepMatch 15 are compared. These variants refer to three different segment sizes of the best sensor modality with lengths of 5, 10 and 15 seconds, respectively. Afterwards, we compare DeepMatch with two well-known baseline methods. The results of evaluating these methods against our datasets are reported and the performance comparison between DeepMatch and those methods is discussed. To further illustrate the accuracy of barometer-based DeepMatch, we look also at the special case of very flat terrain, a worst case scenario when only barometer data is used. In addition, we investigate the execution time overhead of the Matching module carried out on the server. Finally, the battery consumption as well as the CPU and run-time overhead for the passenger smartphones are evaluated.

4.1 Definitions and Metrics for Evaluation

A *positive sample* represents segments belonging to *Class 1*, and a *negative sample* those from *Class 0*. Furthermore, according to the common denominations in binary classification, we define the following terms: *True Positive (TP)*: a correctly classified positive sample; *True Negative (TN)*: a correctly classified negative sample; *False Negative (FN)*: a positive sample wrongly classified as negative; *False Positive (FP)*: a negative sample falsely classified as positive.

The following four metrics are used for evaluation:

- *Precision (PR)*: The ratio of correct positive predictions to the total number of predicted positive samples, *i.e.*, out of all samples classified as positive, how many belong to *Class 1*:

$$PR \triangleq \frac{TP}{TP + FP} \tag{1}$$

- *Recall (RE)*: The ratio of correct positive predictions to the total number of positive samples, *i.e.*, out of all available positive samples in the dataset, how many were correctly classified by the model:

$$RE \triangleq \frac{TP}{TP + FN} \tag{2}$$

- *Accuracy (ACC)*: In a dataset with a 50/50 class distribution, the accuracy describes how good the model is at classifying samples from all classes, *i.e.*, it describes how many of all predictions made are correct:

$$ACC \triangleq \frac{TP + TN}{TP + FP + TN + FN} \tag{3}$$

- *F1-score (F1)*: The harmonic mean between precision and recall. The F1-score is useful in cases where the distribution of classes is not 50/50:

$$F1 \triangleq 2 \cdot \frac{PR \cdot RE}{PR + RE} \tag{4}$$

We plot the results of our evaluation in so-called Receiver Operating Characteristics (ROC)-graphs which describe how good a function and/or a model are at distinguishing between the classes in the dataset. The measurements for the three DeepMatch variants using barometer data and two baseline methods according to these metrics will be discussed in Section 4.4.

4.2 Data Collection and Dataset Creation

The data was collected by volunteers, each carrying one to three Android phones. All phones were connected through the Android application discussed in Section 3.3.1. The following seven Android devices were used: Huawei Nexus 5X, two Huawei Nexus P6, Samsung S8, Sony Z3 Compact, Google Pixel XL and Google Pixel 3a. The data was collected during trips made by public transportation (*i.e.*, trains, subways, busses and trams) in Oslo and Trondheim, two Norwegian cities. In total, we collected 21,252 unique 10 second sensor data segments that consist of events from the magnetometer, accelerometer, gyroscope, and barometer sensors[1]. Following the common practice in machine learning, 70% of the segments were

[1]The datasets will be available via GitHub. In this version of the paper, we do not share the GitHub link due to the double-blind review policy of the DEBS conference.

used for training and 30% for evaluation. Thereafter, matching sets were created separately for both training and evaluation, resulting in a training dataset of 180,408 and an evaluation set of 67,304 unique samples.

The creation of the matching sets was performed separately for the training and evaluation sets to avoid using the same sensor event segments in both phases. In this way, any segment used in the evaluation set has never previously been seen by the model. In both sets, we selected each 50% of the segment pairs from *Class 0* and *Class 1*. Following this approach, we created seven datasets containing data from various sensor modality combinations:

- **A**: Accelerometer
- **M**: Magnetometer
- **B**: Barometer
- **BA**: Barometer and Accelerometer
- **BM**: Barometer and Magnetoer
- **AMG**: Accelerometer, Magnetometer and Gyroscope
- **AMGB**: Accelerometer, Magnetometer, Gyroscope and Barometer

After training the models on these datasets, and evaluating their performance on the evaluation sets, the best performing model and sensor modality were selected for further testing. The next goal was to test how models trained on segments of varying lengths would perform. To do this, we created two additional datasets of 5 and 15 second segments from the best performing sensor modality data.

4.3 Baseline Methods

Two baseline methods were chosen for comparison with DeepMatch: *Normalized Correlation (NORM_CORR)* which calculates the correlation between two sequences by comparing datapoints in the same temporal position, and *Dynamic Time Warping (DTW)* which compares all datapoints in two sequences by warping the temporal dimension to find the best correlation for any datapoint.

Since DTW describes the distance between two sequences, where a large distance equals a small correlation, we inverse the results from this function. The goal is to find a way to classify instances belonging to the two classes in the dataset, using these methods. The assumption is that applying either method on samples belonging to *Class 1*, should provide a large value, while samples belonging to *Class 0* should return a small value. To this end, we used the following equation:

$$c = f(X_a, X_b), \qquad Y' = \begin{cases} 1 & \text{if } c > \alpha \\ 0 & \text{else} \end{cases}$$

The function f represents either of the two baseline methods, and c the result of applying f to the segments X_a and X_b in a sample from the dataset. The delimiting value α is used to classify instances of the two classes from their c values. To find α, we first apply f to all samples in the training set and add the resulting c-values to a sorted array. Thereafter, we search for the optimal delimiting value α, best able to separate instances in the sorted array. If the value c for a sample is larger than the delimiting value α, the sample is assumed to belong to *Class 1*. Otherwise, it should belong to *Class 0*. Optimal α values were searched for both NORM_CORR and DTW using the training set. Then, we evaluated the functions and their corresponding α values on the evaluation set. The results of our experiments are discussed in the following.

Table 3: Confusion matrix for the barometer-based DeepMatch 10

	Predicted Positive	Predicted Negative
Actual positive	33018	634
Actual negative	842	32810

Table 4: Performance comparison various sensor combinations

Model	PR	RE	ACC	F1
DeepMatch 10 A	0.5065	0.9531	0.5122	0.6615
DeepMatch 10 M	0.5064	0.9280	0.5118	0.6553
DeepMatch 10 B	**0.9751**	0.9812	**0.9781**	**0.9781**
DeepMatch 10 BA	0.7332	0.9697	0.8082	0.8350
DeepMatch 10 BM	0.7081	0.9708	0.7853	0.8189
DeepMatch 10 AMG	0.5011	0.9646	0.5020	0.6595
DeepMatch 10 AMGB	0.7079	**0.9892**	0.7905	0.8253

4.4 Experimental Results

During the development of our model, we continuously evaluated our results using the metrics described above. The confusion matrix, *i.e.*, the overall number of TP-, TN-, FN-, and FP-rated samples, for barometer-based DeepMatch 10 is listed in Table 3. The values of the confusion matrices for the learned models and the two baseline models allowed us to compute the outcomes according to the four metrics introduced in Section 4.1. The results are presented in Tables 4 and 5, and discussed below.

4.4.1 Sensor Modality Experiments. Table 4 depicts the results from training DeepMatch on various sensor modality combinations as described in Section 4.2. The numbers show that all models trained on datasets containing barometer data outperform all other models. Moreover, the DeepMatch 10 B, trained on barometer data alone, outperforms all other models. As described in Section 3, the barometer sensor is precise independently of the position of the vehicle. In particular, it is resistant to vibrations and sudden user movements as well as highly sensitive to elevation changes. This makes it perfectly suited to capture the movements of the vehicle rather than the movements of the individual user.

The accelerometer and gyroscope, on the other hand, are more sensitive to the movements of the users. The magnetometer is more sensitive to magnetic objects in the proximity to the user as well as to the power unit of the vehicle than to the movements of the vehicle which makes it also a poor source of data for the model. All these factors impact the performance of the models, and the results in Table 4 show that, with one exception, DeepMatch 10 B renders the best results. That holds particularly for the important *ACC* metric that shows the share of correct versus all matchings. The high *RE* value of the AMGB model indicates that the model correctly classifies most of the positive samples as positive. It seems, however, that it has a bias towards false positives, *i.e.*, classifying also negative

Table 5: Performance comparison of the barometer-based DeepMatch with baseline methods

Model	PR	RE	ACC	F1
DeepMatch 5	0.9408	0.9765	0.9574	0.9583
DeepMatch 10	0.9751	0.9812	**0.9781**	**0.9781**
DeepMatch 15	0.9348	**0.9816**	0.9566	0.9576
NORM_CORR	0.9174	0.9595	0.9393	0.9380
DTW	**0.9810**	0.7350	0.8136	0.8404

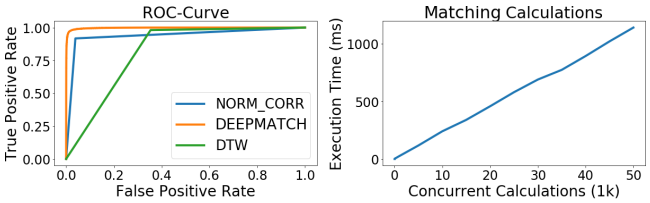

Figure 5: ROC-curve for baseline methods (left) and matching calculations execution time (right).

samples as positive, which results in a low *PR* value. Altogether, limiting ourself to using only the barometer data seems to be the most promising way to conduct in-vehicle presence detection.

4.4.2 Segment size experiments with the barometer-based DeepMatch. From the numbers in Table 5, we can conclude that for all performance metrics, *DeepMatch 10* is outperforming *DeepMatch 5*. This is caused by the difference in segment sizes for the two models, 512 and 256 data points respectively. Thus, the former model has more data to learn from than the latter, which explains the higher quality of its performance. According to this explanation, however, DeepMatch 15 with its 768 data points should outperform the two other models. This is true for RE but not for the other three metrics where it underperforms at least DeepMatch 10. Due to the bad PR value in comparison with the good RE result, the model seems to be biased towards classifying samples as positive which leads to an extended number of false positives. Probably, the composition of 15 seconds long segments of our learning set is non-representative which leads to learning a sub-optimal classifier. Using a larger dataset, we believe DeepMatch 15 would outperform DeepMatch 10. We will follow this up in future experiments.

4.4.3 Baseline Methods. From Table 5, we can see that RE, ACC, and F1 of both baseline methods are lower than the corresponding metrics for the learned DeepMatch models. The sole exception is the metric PR for which DTW gave a better result than both the DeepMatch variants and NORM_CORR. The reason for this is a correlation of DTW to negative samples that we discuss below. That causes the consequence that DTW produces only relatively few false positives which renders the good result for PR. Instead, it generates a significant number of false negatives spoiling the values for the other metrics.

Altogether, the two baseline methods seem to be less suited for in-vehicle presence detection than DeepMatch. For NORM_CORR, we believe this is due to the sensitivity of the function to time-lag between its input sequences, *e.g.*, a passenger sitting a couple of meters behind the RefDev in the vehicle, will experience a lag between the signals which will result in a lower correlation value for positive samples. Therefore, the correlation value for some of the positive samples will be mixed with the correlation value for negative samples resulting in a less optimal delimiter.

The low performance of DTW is most likely caused due to lacking sensitivity to the temporal dimension. DTW warps the temporal dimension between the two sequences to find the shortest distance. This will result in a very high correlation value for some negative samples, making it difficult for the delimiter to separate samples from the two classes. As a result of this, there are relatively few false positives at the expense of many false negatives which explains the discrepancy of DTW's results for the different metrics in Table 5. Similar results can be observed in the ROC-graphs for the models. The left graph in Fig. 5 depicts the ROC-curve for DeepMatch 10, NORM_CORR and DTW. A property of these curves is that, as larger the areas under the curve are, as better the performance of the corresponding model will be. According to that, DeepMatch 10 is better than NORM_CORR and much better than DTW what our RE, ACC and F1 results also reflect.

4.5 Discussion of the Experimental Results

At a first glance, the differences between the accuracies of barometer-based DeepMatch 10 ($ACC = 0.9781$) and the baseline model NORM_CORR ($ACC = 0.9393$) may not seem to be considerable. In practice, however, they may have a great effect. Let us take an auto-ticketing system for city busses. Reflecting short distances of just one or two minutes journey time between two bus stops in an inner city environment, we assume that six in-vehicle prediction runs (*i.e.*, six segments of 10 seconds each) are conducted during this period. To reduce the risk of wrongly billing people who are not riding in a bus but being, *e.g.*, in a car next to it, the bus operator may apply a policy to ticket somebody only if at least five of these six runs predict the user's smartphone being in the bus. The likelihood P_{cr} that our policy correctly detects a passenger can be computed as follows:

$$P_{cr} = ACC^6 + 6 \times ACC^5 (1 - ACC)$$

Thus, taking the ACC value of NORM_CORR, $P_{cr} = 95.31\%$ of all passengers are ticketed on average while the rest travels for free. This system leads to a revenue reduction of nearly 5% which few bus operators would accept. With DeepMatch 10, however, $P_{cr} = 99.32\%$ of the passengers are correctly billed. The loss of revenue of less than one percent seems to be acceptable since it will be easily outweighed by reducing the number of ticket machines and other infrastructure.

Additionally, for the case of wrongly billing non-passengers, DeepMatch 10 has a significant advantage over NORM_CORR. Using the policy mentioned above, the likelihood P_{er} of erroneous ticketing can be calculated by the following formula:

$$P_{er} = (1 - ACC)^6 + 6 \times ACC (1 - ACC)^5$$

That leads to the values $P_{er} = 0.000003\%$ with DeepMatch 10 and $P_{er} = 0.000469\%$ with normal correlation. In the latter case, around 171 people are wrongly billed in a year if we assume a 100,000 non-passengers being checked for in-vehicle presence every day which seems reasonable for a larger city. Thus, more than three such cases

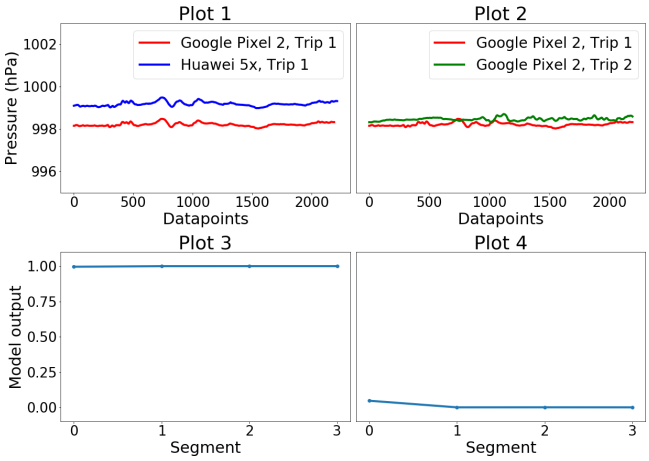

Figure 6: Match prediction tests for trips on flat roads.

arise every week leading to a lot of compensation claims and bad press. In contrast, using DeepMatch 10, only a single person is wrongly billed in a year which seems acceptable.

4.6 Performance in Flat Terrain

In Section 4.4.1, we showed evidence that DeepMatch works best when only barometer data is used. This, however, may cause a problem in level areas. Therefore, to measure the accuracy of DeepMatch in such worst case scenarios, we made different trips in a very flat region in the central district of Trondheim. Some results of these experiments are shown in Fig. 6. Plot 1 shows the pressure measured by two different phones during the same trip, while Plot 2 depicts pressure measurements from two trips using the same phone. The low amplitudes in all curves show the flatness of the area. Plot 3 shows the match prediction of DeepMatch 10 based on the sensor output from the two devices in Plot 1. Plot 4 depicts the match prediction of our model based on the sensor output from the two trips in Plot 2. As indicated by Plot 4, DeepMatch detects dissimilarity at a high accuracy, despite the very similar pressure values measured by Google Pixel 2 in Plot 2.

4.7 Matching Execution Time

To use DeepMatch-based in-vehicle prediction also in real environments, the server needs to be able to do matching calculations from a large number of concurrently travelling passengers. That holds particularly for centralized server structures. The right graph in Fig. 5 shows the execution time of a central server as a function of increasing concurrent calculations. To increase the operational speed of our system, we exploited the feature of Tensorflow models to make several simultaneous predictions on multiple inputs. This resulted in an execution time of 1,140 milliseconds for 50,000 concurrent matching calculations, all running on one desktop equipped with a single GTX 1080 GPU. Since all trips between two stops are far longer than the 1,140 milliseconds, a data center consisting of just 19 of such computers could serve a city like Oslo with its 950,000 daily passengers even if all of them travel at the same time.

Table 6: Android phones used in the tests

Type	Battery capacity	Age
Samsung S8	3000 mAh	2 years
LG Nexus 5X	2700 mAh	3 years
Huawei Nxus 6P	3450 mAh	4 years
Google Pixel 3a	3000 mAh	1 year
Sony Z3 compact	2600 mAh	5 years

Table 7: Battery consumption per hour

Brand	Data collection	Learning	Complete
Samsung	25 mA	26 mA	31 mA
LG	23 mA	24 mA	26 mA
Huawei	22 mA	23 mA	25 mA
Google	16 mA	17 mA	18 mA
Sony	15 mA	18 mA	21 mA

4.8 Battery Consumption on Smartphones

In this subsection, we evaluate the battery consumption of DeepMatch which is crucial for the adoption of our approach in practice. In general, there are three main sources of battery drain in our framework, *i.e.*, collecting barometer data, the encoder module for data processing, and transmitting the processed data to the server.

For our tests, we selected five Android phones from five different manufacturers that are listed in Table 6. To consider age diversity, we used phones that are between one and five years old. Besides the battery capacity, the environment temperature is an important factor that can influence the performance of batteries. Therefore all tests were run in an experimental environment with a temperature of 19° Celsius representing the indoor temperature of typical transportation vehicles. Since barometer-based DeepMatch 10 promises the best overall performance (as discussed above), we consider this version of our model for the battery measurements.

The battery status is collected from the app using the *Batterystats* and *Battery Historian* tools included in the Android framework [9]. These tools provide functionality to extract details on battery consumption for all applications running on the device. In order to ensure that the app can listen to barometer events and process them in intervals of ten seconds, we run the tests in the background with the *wake lock* parameter enabled to keep CPU processing on. Reflecting the above mentioned battery consumption factors, we use the following three scenarios for our experiments:

- *Complete scenario*: All three factors of battery consumption, *i.e.*, the barometer data collection, data processing by the encoder, and data transmission.
- *Learning scenario*: Data collection and data processing.
- *Data collection scenario*: Only barometer data collection.

The results of our tests are depicted in Table 7. The numbers show clearly that for all five devices, DeepMatch influences the battery consumption only marginally. For all phones, the battery usage will be less than 62 mA considering a total travel time of two hours a day. With a battery capacity of 3000 mAh, this equals 2.1%. This value is considerably lower than most smartphone apps, as

Table 8: Run Time and CPU overhead

Brand	CPU	Mean Run Time	Overhead
Samsung	2.3 GHz + 1.7 GHz, Cortex-A53	49 ms	1-2 %
LG	1.4 GHz + 1.8 GHz, 64-Bit Hexa-Core	46 ms	1-2 %
Huawei	2.0 GHz + 1.55 GHz, 64-Bit Octa-Core	52 ms	1-2 %
Google	2.0 GHz + 1.7 GHz, 64-Bit Octa-Core	19 ms	0-1 %
Sony	2.5 GHz Quad-Core, 400 Krait	73 ms	3-4 %

reported in [4]. Therefore, we believe that the battery consumption of DeepMatch is satisfactory.

4.9 Computational Overhead on Smartphones

In this subsection, we evaluate the computational overhead of the feature extraction and dimensionality reduction performed by the barometer-based DeepMatch 10 on smartphones. For these experiments, we used the same smartphones as in the battery consumption analysis. We registered both the run-time and CPU usage of the *encoder* module, when it processed sensor events with intervals of 10 seconds. The results of our tests are depicted in Table 8. The numbers show clearly that, for all phones, the mean run-time and CPU overhead of the encoder is barely noticeable. Even for the oldest model in the tests, the five year old Sony Z3 Compact, the mean run time of the encoder is 73 ms which affords a CPU usage of only three to four percent.

5 CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a machine learning-based approach, called DeepMatch, to address the challenge of in-vehicle presence detection as an important aspect of mobile context. It utilizes the sensor event streams of a smartphone to estimate its presence in a public transport vehicle at the very high accuracy of nearly 98%. DeepMatch is based on utilizing Stacked Convolutional Autoencoders for feature extraction and dimensionality reduction, and a dense neural network for event stream matching. The feature extraction and dimensionality reduction run on the smartphone and the reference device, while the event matching is performed on a server. Through dimensionality reduction, the datapoints are reduced by the factor four such that the bandwidth of the data transfer to the server is considerably reduced without losing the information of the data necessary to perform matching.

Our future plan is to improve DeepMatch 10 with further data gathering and model optimization. During 2020, we will implement a pilot of DeepMatch in a Norwegian city together with a public transportation provider. Moreover, we intend to research on the optimum length of the data segments and the frequency of data gathering (from the reference devices and the smartphones) in order to minimize the amount of data needed for in-vehicle presence detection.

REFERENCES

[1] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. 2012. Fog Computing and its Role in the Internet of Things. In *1st Workshop on Mobile Cloud Computing (MCC)*. ACM, Helsinki, Finland, 13–16.

[2] Bosch. 2020. Bosch BMP280. https://www.bosch-sensortec.com/bst/products/all_products/bmp280. (2020). Accessed: 2020-04-01.

[3] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. 1994. Signature Verification using a "Siamese" Time Delay Neural Network. In *Advances in Neural Information Processing Systems*. Morgan Kaufmann Publishers, San Francisco, CA, USA, 737–744.

[4] Xiaomeng Chen and others. 2015. Smartphone Energy Drain in the Wild: Analysis and Implications. *ACM SIGMETRICS Performance Evaluation Review* 43, 1 (2015), 151–164.

[5] A. Dimri, H. Singh, N. Aggarwal, B. Raman, D. Bansal, and K. K. Ramakrishnan. 2016. RoadSphygmo: Using Barometer for Traffic Congestion Detection. In *8th International Conference on Communication Systems and Networks (COMSNETS)*. IEEE Computer, Bangalore, India, 1–8.

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press, Cambridge, MA, USA, Chapter Autoencoders, 505–528. http://www.deeplearningbook.org.

[7] T. Gründel, H. Lorenz, and K. Ringat. 2006. The ALLFA Ticket in Dresden. Practical Experience of Fare Management Based on Be-In/Be-Out & Automatic Fare Calculation. (2006). IPTS Conference, Seoul, South Korea.

[8] T. Gyger and O. Desjeux. 2001. EasyRide: Active Transponders for a Fare Collection System. *IEEE Micro* 21, 6 (2001), 36–42.

[9] Battery Historian. 2019. Batterystats and Battery Historian. https://developer.android.com/studio/profile/battery-historian. (2019). Accessed: 2019-10-23.

[10] Bo-Jhang Ho, Paul Martin, Prashanth Swaminathan, and Mani Srivastava. 2015. From Pressure to Path: Barometer-based Vehicle Tracking. In *2nd ACM Inter. Conf. on Embedded Systems for Energy-Efficient Built Environments (BuildSys)*. ACM, Seoul, South Korea, 65–74.

[11] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv* cs.LG, arXiv:1502.03167, Article 1502.03167 (2015), 11 pages.

[12] Seungwoo Kang, Youngki Lee, Chulhong Min, Younghyun Ju, Taiwoo Park, Jinwon Lee, Yunseok Rhee, and Junehwa Song. 2010. Orchestrator: An Active Resource Orchestration Framework for Mobile Context Monitoring in Sensor-rich Mobile Environments. In *IEEE International Conference on Pervasive Computing and Communications (PerCom)*. IEEE Computer, Mannheim, Germany, 135–144.

[13] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. 2015. Siamese Neural Networks for One-shot Image Recognition. (2015), 8 pages. https://www.cs.cmu.edu/~rsalakhu/papers/oneshot1.pdf.

[14] Sriharsha Kuchimanchi. 2015. *Bluetooth Low Energy based Ticketing Systems*. Master's thesis. Aalto University, Espoo, Finland.

[15] Andrzej Kwiecień, Michał Maćkowski, Marek Kojder, and Maciej Manczyk. 2015. Reliability of Bluetooth Smart Technology for Indoor Localization System. In *International Conference on Computer Networks (CN) (CCIS 522)*. Springer-Verlag, Br'unow, Poland, 444–454.

[16] Naveen Sai Madiraju, Seid M Sadat, Dimitry Fisher, and Homa Karimabadi. 2018. Deep Temporal Clustering: Fully Unsupervised Learning of Time-domain Features. *arXiv* cs, arXiv:1802.01059, Article 1802.01059 (2018), 11 pages.

[17] Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. 2011. Stacked Convolutional Auto-encoders for Hierarchical Feature Extraction. In *International Conference on Artificial Neural Networks (ICANN) (LNCS 6791)*. Springer-Verlag, Espoo, Finland, 52–59.

[18] Christian Mayer, Ruben Mayer, and Majd Abdo. 2017. StreamLearner: Distributed Incremental Machine Learning on Event Streams: Grand Challenge. In *11th ACM International Conference on Distributed and Event-Based Systems*. ACM, Barcelona, Spain, 298–303.

[19] R. Meng, D. W. Grömling, R. R. Choudhury, and S. Nelakuditi. 2016. RideSense: Towards Ticketless Transportation. In *2016 IEEE Vehicular Networking Conference (VNC)*. IEEE, Columbus, OH, USA, 1–8.

[20] W. Narzt, S. Mayerhofer, O. Weichselbaum, S. Haselböck, and N. Höfler. 2015. Be-In/Be-Out with Bluetooth Low Energy: Implicit Ticketing for Public Transportation Systems. In *IEEE 18th International Conference on Intelligent Transportation Systems*. IEEE, Las Palmas, Spain, 1551–1556.

[21] Kartik Sankaran, Minhui Zhu, Xiang Fa Guo, Akkihebbal L Ananda, Mun Choon Chan, and Li-Shiuan Peh. 2014. Using Mobile Phone Barometer for Low-power Transportation Context Detection. In *12th ACM Conference on Embedded Network Sensor Systems*. ACM, Memphis, TN, USA, 191–205.

[22] C. Sarkar, J. J. Treurniet, S. Narayana, R. V. Prasad, and W. de Boer. 2018. SEAT: Secure Energy-Efficient Automated Public Transport Ticketing System. *IEEE Transactions on Green Communications and Networking* 2, 1 (2018), 222–233.

[23] Akara Supratak, Hao Dong, Chao Wu, and Yike Guo. 2017. DeepSleepNet: A Model for Automatic Sleep Stage Scoring based on Raw Single-channel EEG. *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 25, 11 (2017), 1998–2008.

[24] Tensorflow. 2019. Tensorflow 2.0 RC Tutorials. https://www.tensorflow.org/beta/. (2019). Accessed: 2019-10-23.
[25] Salvatore Vanini, Francesca Faraci, Alan Ferrari, and Silvia Giordano. 2016. Using Barometric Pressure Data to Recognize Vertical Displacement Activities on Smartphones. *Computer Communications* 87 (2016), 37–48.
[26] Jindong Wang, Yiqiang Chen, Shuji Hao, Xiaohui Peng, and Lisha Hu. 2017. Deep Learning for Sensor-based Activity Recognition: A Survey. *Pattern Recognition Letters* 19 (2017), 3–11.
[27] M. Won, A. Mishra, and S. H. Son. 2017. HybridBaro: Mining Driving Routes Using Barometer Sensor of Smartphone. *IEEE Sensors Journal* 17, 19 (2017), 6397–6408.
[28] Shuochao Yao, Shaohan Hu, Yiran Zhao, Aston Zhang, and Tarek Abdelzaher. 2017. Deepsense: A Unified Deep Learning Framework for Time-series Mobile Sensing Data Processing. In *26th International Conference on World Wide Web*. ACM, Perth, Australia, 351–360.
[29] Cheng Zhang, Wu Liu, Huadong Ma, and Huiyuan Fu. 2016. Siamese Neural Network based Gait Recognition for Human Identification. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, Shanghai, China, 2832–2836.
[30] Yi Zheng, Qi Liu, Enhong Chen, Yong Ge, and J. Leon Zhao. 2016. Exploiting Multi-Channels Deep Convolutional Neural Networks for Multivariate Time Series Classification. *Frontiers of Computer Science* 10, 1 (Feb. 2016), 96–112.