Institute of Technical Informatics

448.066 - Mobile Computing Lab

EmTrack report

By:
Adrian Stokdal Opheim - 12212310

June, 2023

Table of Contents

List of Figures

1 Introduction

The idea of the EmTrack application was to enable day-to-day tracking and recording of CO_2emmission based on the activities of the user. This project is limited to only estimating the user's emission coming from transportation. The app would detect whether the user is walking, biking, driving a car or taking a bus etc. and given the time spent in each transport mode estimate how much $CO_2$is emitted into the atmosphere. The transport mode detection is accomplished by recording sensor data on an android phone and then running this through a pre-trained machine learning model locally on the phone. This ML-model was chosen as an LSTM network due to its performance on time-series-data and previous papers showed great results using this approach.

2 Previous work

There is almost an abundance of papers on the matter of recognizing human-activity by the use of smartphone data as well as a number of papers on estimating the transport mode of the user (Transport Mode Detection - TMD). The one chosen to be the best fit for this project can be found in [1] where the performance of many different algorithms were compared on the Sussex-Huawei Locomotion (SHL) dataset ([4] and [2]). This paper proposed an optimal solution using a Long Short Term Memory network with 64 nodes. The SHL dataset seemed like the largest and most comprehensive dataset of smartphone sensor recordings available therefore this was also used in this project to train the ML model. Another dataset that was considered and tried in the early phases of the project was the dataset created by HTC a few years ago [6]. They also proposed a promising low-power solution using only a handful of features to predict transportation mode. However, newer papers with more modern approaches provided better accuracies for this problem so this approach was not used. Another approach using a temporal convolutional network approach to better extract features from the sensor data was proposed in [5], but the results in terms of accuracy with this approach was lower according to the paper.

3 Implementation

3.1 Android app

The application itself was written in Kotlin for the Android platform. The UI is based on a bottom navigation bar with three different views: *Live*, *Dashboard* and *Record*. The purpose of these views are as follows:

1. *Live* - Shows current detected transport mode along with a confidence level.
2. *Dashboard* - Shows the calculated emission along with a number of statistics around this.
3. *Record* - This is to record labeled training data from the phone. This can be used to further improve the ML model.

The three UI views were created as fragments in the Android framework and its views were inflated based on the actions in the navigation bar. For this project the navigation bar template provided in Android Studio was used and adapted accordingly.

Two helper classes were defined: `TMDModelHelper` and `SensorRecorder`. The former implements helper-functions for the ML model. This includes loading the model from memory, initialization and running inference. The latter takes care of recording the sensor data. This implements the base class `SensorEventListener` provided by Android, which makes it convenient to always run a piece of code upon a change in one sensor value for instance. This happens in the override of the `onSensorChanged()` function. Additionally, there is one main activity which is the starting point of the application. Here, the `SensorRecorder` is started and a runnable is created that checks for

new sensor data every five seconds. If there is new data and enough windows have been recorded it runs inference and the current transport mode is detected. Having the `SensorRecorder` running in the main activity is a must if sensor values have to be consistent between views and fragments. However, also the record fragment need access to the data and variables from the `SensorRecorder`. To get around this, the record fragment gets access to the instance of the `SensorRecorder` through the main activity and reads the data from there. In addition to this a listener to the count of recorded windows was set up to allow the record fragment to only save data if a new window has been created. The inference runnable also processes and saves the output to a JSON-file so that all emission data is available, even if the app is closed and opened again.

After the transport mode has been classfied and stored in the JSON-file, the emitted $CO_2$is calculated by a simple lookup table. Data on how much CO_2each transport mode emmits per kilometer was found in [3]. From this an average speed per transport mode was calculated and average CO_2emission per second was estimated.

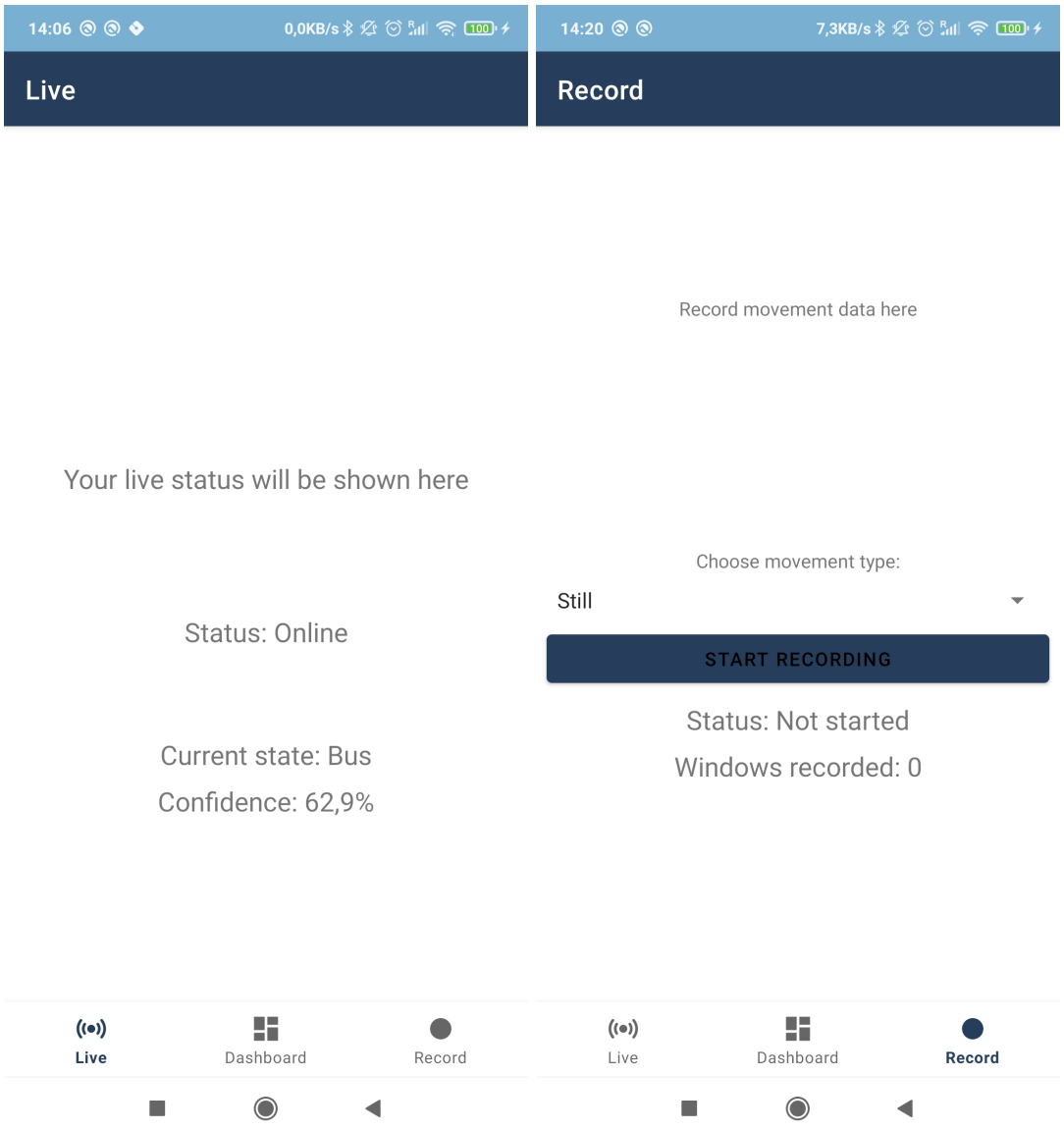

Figure 1: UI of the Live- and record-view.

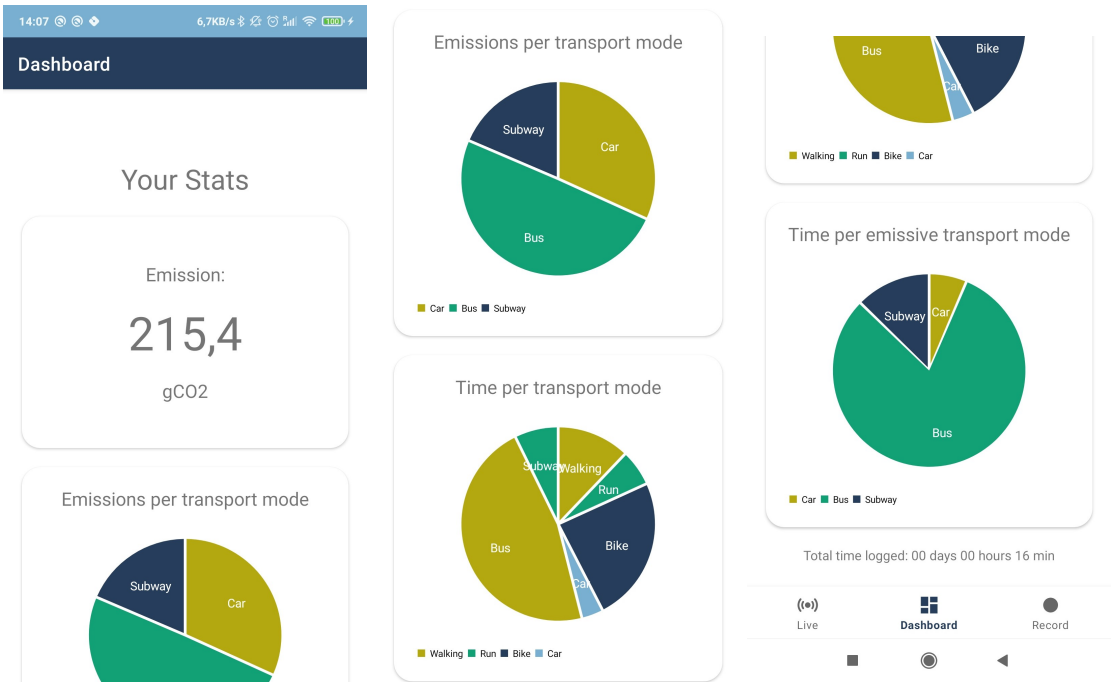

Figure 2: UI of the dashbboard ScrollView.

3.2 Sensor data

In [1] they used all the available sensors in the phone as these are present in the SHL dataset [4]. However for the phone at hand not all of these sensor readings were available and for some of them the data provided in the SHL dataset was inconsistent (0 for large periods of time). Therefore only the following five sensors were used to create input for the ML model:

1. Accelerometer (x, y, z)
2. Gyroscope (x, y, z)
3. Magnetometer (x, y, z)
4. Orientation (w, x, y, z)
5. Linear acceleration (x, y, z)

Additionally, the magnitude of each sensor except for the orientation was calculated and added as a feature. This makes a total of 20 features.

The sensor reading frequency in the phone was set to around 5 Hz. Then, 50 readings are averaged to make up one window of about 10 seconds. The same treatment was given to the data in the SHL dataset (recorded at 100 Hz). First it was downsampled to 5 Hz by only keeping every 20th value, then all rows containing NaN-values or null-class was purged, then the remaining samples was averaged over 50 samples to create a window.

No overlapping of windows was implemented, however the ML model uses 3 consecutive windows to estimate the transport mode.

3.3 Machine learning model

The machine learning model used was a Long Short-Term Memory (LSTM) neural network. TensorFlow was chosen as the machine learning library of choice because of its widespread use

and thorough documentation along with the ability to convert a model into a TensorFlow Lite model that can be used on a mobile platform.

This LSTM network uses 32 nodes and provided sufficient performance when trained on the SHL dataset. In [1] 64 nodes was found to be the best hyperparameter for their problem, however little change in performance was seen when changing to 32 nodes instead. Therefore, the lighter model was prefered. This implementation is a single layer network with an additional means of regularization in a dropout layer randomly dropping 20% of the weights. The Details of the model can be read from the printout of the `model.summary()` function in Figure 3.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 3, 32)             6784

 dropout (Dropout)           (None, 3, 32)             0

 flatten (Flatten)           (None, 96)                0

 dense (Dense)               (None, 8)                 776

=================================================================
Total params: 7,560
Trainable params: 7,560
Non-trainable params: 0
_________________________________________________________________
None
```

Figure 3: Parameters for the machine learning model used.

3.4 Training the model

Two datasets were used in the training process. The SHL dataset and a smaller recorded dataset from my own phone. The total number of samples in the SHL dataset was 141 019, corresponding to $\tilde{3}92$ hours of recording. On the other hand the recorded dataset had only 1065 samples, corresponding to about $\tilde{3}$ hours of recording. The number of labeled samples per class in the SHL dataset can be seen in Figure 4 while the number of samples per class for the recorded dataset can be seen in Figure 5. Notice that no samples of either driving a car or riding a train was recorded.

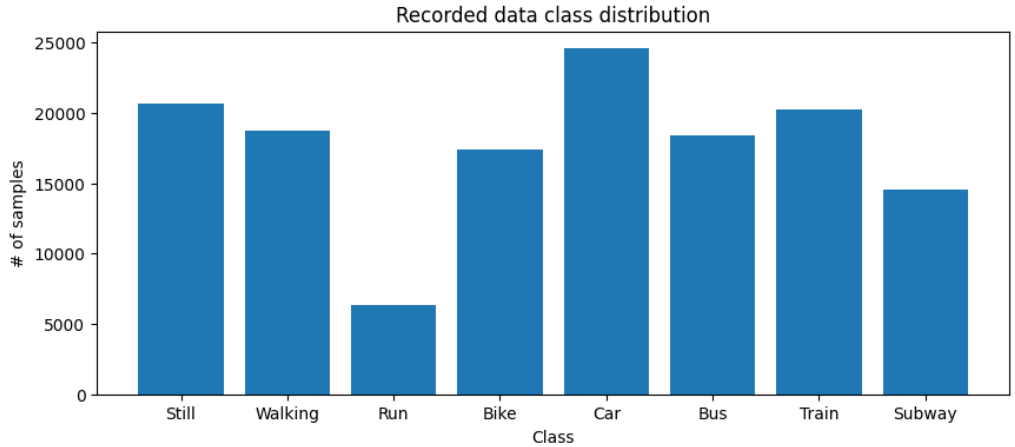

Figure 4: Class distribution of the SHL dataset.

Figure 5: Class distribution of the recorded dataset.

The SHL dataset was used to "pre-train" the model, then the network was re-fitted on the recorded train-set. This means that the model uses its computed weights from the first fit on the SHL dataset and then start from there to optimize the parameters for the recorded dataset. This was done because the performance of the model on recorded data was quite bad after training only on the SHL dataset. In other words, the model didn't generalize well to also work for sensor data recorded on another phone in another location. Ideally, the recorded dataset would be much bigger. Had the network consisted of more layers one could perhaps freeze some layers after the initial training on the SHL dataset. One architecture that might work would perhaps be if convolutional layers was used before an LSTM to encode the most important features and reduce input dimension for the LSTM part. Then, one could freeze the convolutional encoder part after SHL training and only re-fit the LSTM part on the recorded dataset.

The SHL-fit used a batch size of 256, ADAM-optimizer, early-stopping on validation Mean Average Error, categorical cross entropy for its loss function and trained for 200 epochs. During the recorded dataset re-fit the batch size was changed to 64 and it trained for 50 epochs. All other hyperparameters stayed the same. The history of training and validation loss per epoch when training on the SHL dataset can be seen in Figure 6 below.

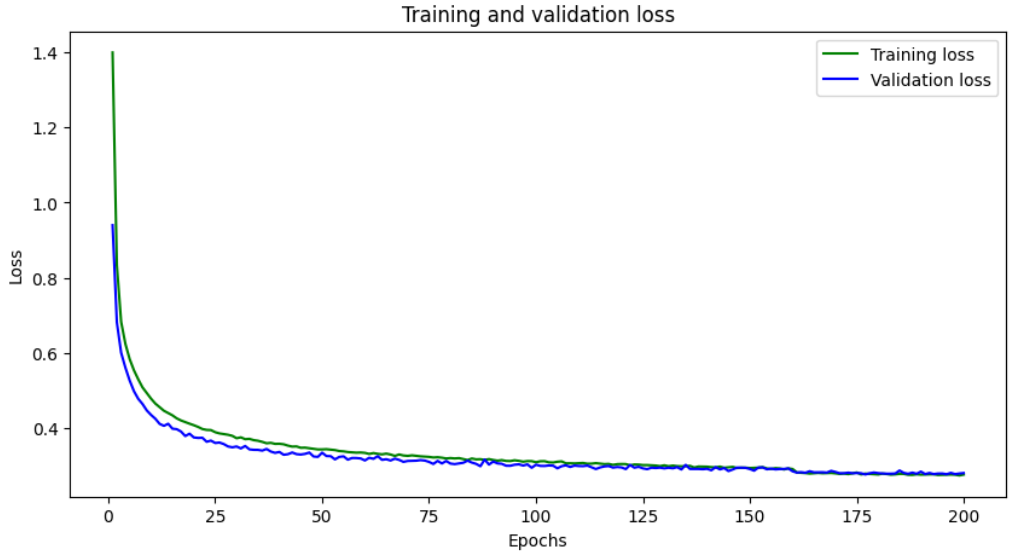

Figure 6: Evolution of Loss per epoch when first training ong the SHL dataset

4 Performance

After training the network on the SHL train-set the test-set performance was quite good, classifying 88.28% of samples correctly. On the other hand, when predicting the entire recorded dataset after fitting only on the SHL dataset the performance was not the best, classifying just 46.57% of samples correctly. However, after splitting the recorded dataset into a train-, validation- and test-set and re-fitting the model on the recorded dataset, the model predicted 99.06% of samples in the recorded test set correctly.

The time it took for the runnable to execute was about 20-25 ms. This includes running inference and storing the data to the JSON-file.

Based on a small number of tests after deploying the re-fitted model to the phone and the app, the model is indeed overfitting to the recorded dataset to some extent. This is reasonable as the recorded dataset is very small an will not cover the variance of data in a day-to-day scenario. Therefore, 99% correct predictions is not to be expected when using the app.

The following confusion matrices show the predicted transport mode against the actual transport mode. If a sample is on the diagonal of this matrix it meas the predicted label matches the true label and this is then a true positive classification. The actual/true label is shown on the vertical axis while the predicted label is shown on the horizontal axis. In Figure 7 the confusion matrix after training the model only on the SHL dataset is shown when predicting the SHL test-set. The second confusion matrix in Figure 8 shows how classifications are distributed when predicting the entire recorded dataset after only training on the SHL dataset. The third and final confusion matrix in Figure 9 shows the distributions of classifications when run on the recorded test-set after re-fitting the model to the recorded train-set.

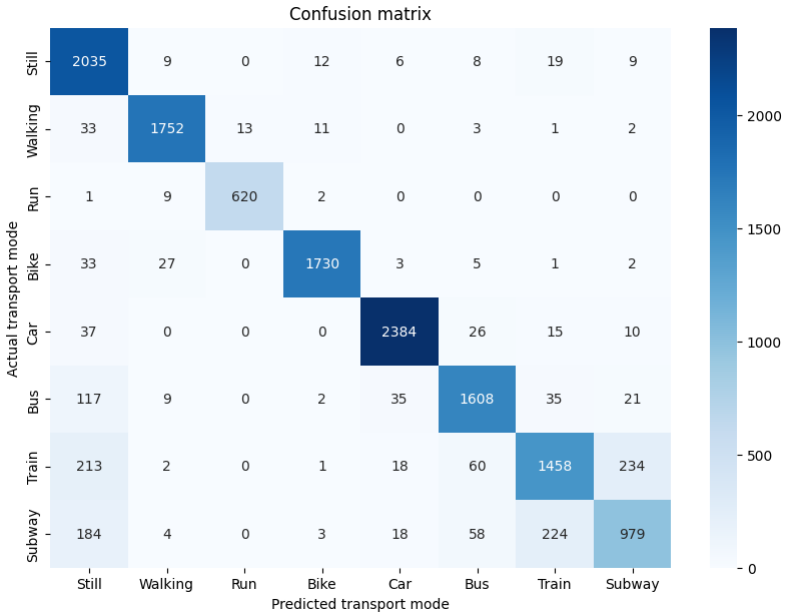

Figure 7: Confusion matrix on SHL test-set after first fit.

5 Limitations & future work

If enough sensor-data from phones is recorded and labeled I believe the machine learning model can be made to perform well enough to actually be useful. However, this requires enormous amounts of data and resources to label all the recordings.

For the calculation of the actual CO_2emission per mode, the method is very primitive. For an

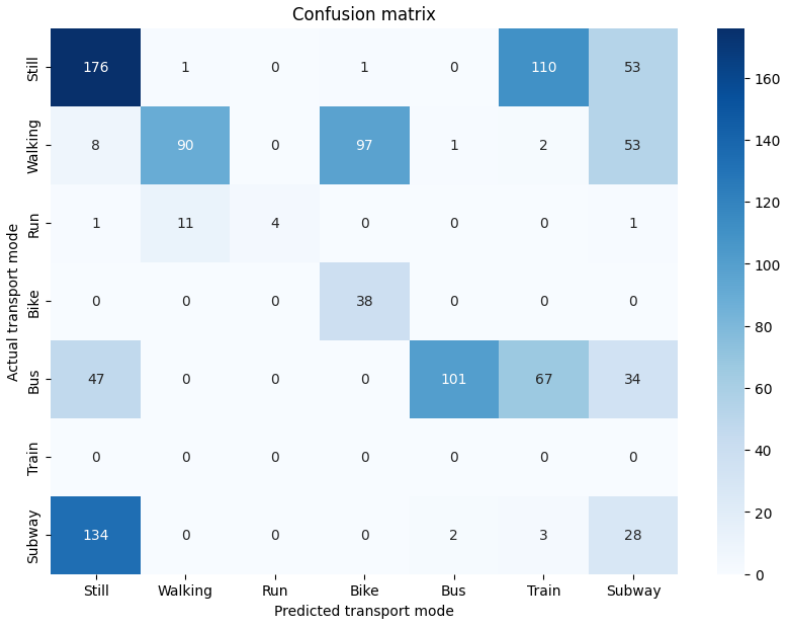

Figure 8: Confusion matrix on entire recorded dataset after first fit.

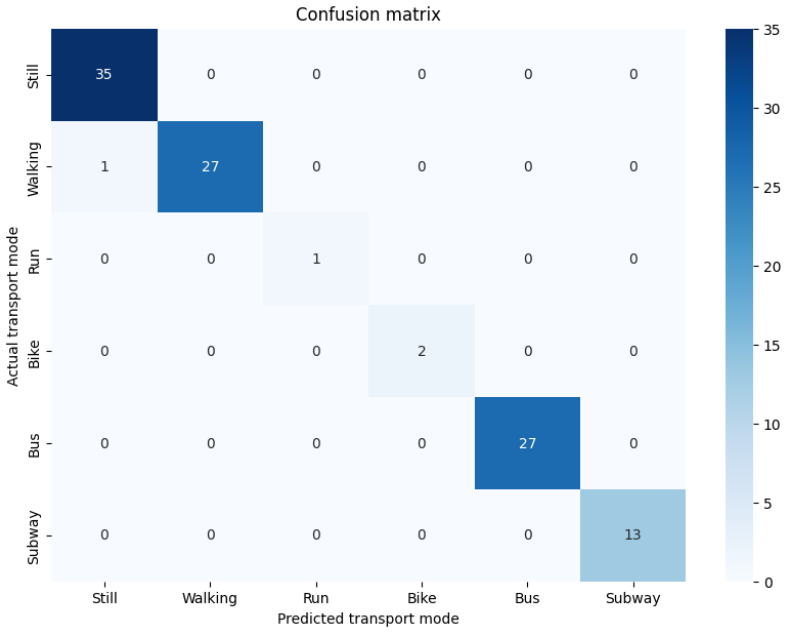

Figure 9: Confusion matrix on recorded test-set after re-fitting to recorded data.

accurate real-life application this will have to be made much more sophisticated in order to get a ballpark accurate estimate of the emmitted greenhouse gasses.

A choice was made not to make use of GNSS data from the phone. Enabling GNSS data can potentially make the estimation of transportation mode many times better. For public transport it might even be a possibility to know which bus or train the user is on by comparing to live timetables like those available through Google Maps for instance. It would also open the door to use the kilometers travelled to estimate the CO_2footprint rather than base this on time. On the downside, GNSS is a power-hungry sensor and it also poses privacy issues in a real-world scenario. These were the reasons it was avoided in the first place.

In the application, the estimated emissions is stored per class in a JSON-file as a single entity. If

this was stored on a per-day basis it would allow interesting statistics of for example when you emitted what. For instance showing if the user tends to walk more and emit less climate gasses in the spring compared to the winter.

6 Conclusion

A functioning mobile application was made that tries to estimate the climate gass emission of the user by obtaining the current transport mode. It does this by running inference on a LSTM neural network that was trained on the SHL dataset along with a recorded dataset from the phone the app was tested on. On paper the performance seems to be very promising, however in real-life the machine learning model does not generalize well enough to be accurate on all unseen data.

Bibliography

[1] Ifigenia Drosouli et al. 'Transportation Mode Detection Using an Optimized Long Short-Term Memory Model on Multimodal Sensor Data'. In: *Entropy* 23.11 (2021). ISSN: 1099-4300. DOI: 10.3390/e23111457. URL: https://www.mdpi.com/1099-4300/23/11/1457.

[2] Hristijan Gjoreski et al. 'A Versatile Annotated Dataset for Multimodal Locomotion Analytics with Mobile Devices'. In: SenSys '17 (2017). DOI: 10.1145/3131672.3136976. URL: https://doi.org/10.1145/3131672.3136976.

[3] Hannah Ritchie. *Which form of transport has the smallest carbon footprint?* URL: https://ourworldindata.org/travel-carbon-footprint (visited on 20th June 2023).

[4] University of Sussex and Huawei. *Sussex-Huawei Locomotion dataset.* URL: http://www.shl-dataset.org/ (visited on 14th June 2023).

[5] Pu Wang and Yongguo Jiang. 'Transportation Mode Detection Using Temporal Convolutional Networks Based on Sensors Integrated into Smartphones'. In: *Sensors* 22.17 (2022). ISSN: 1424-8220. DOI: 10.3390/s22176712. URL: https://www.mdpi.com/1424-8220/22/17/6712.

[6] Meng-Chieh Yu et al. 'Big Data Small Footprint: The Design of A Low-Power Classifier for Detecting Transportation Modes'. In: *PVLDB* 7.13 (2014), pp. 1429–1440.