Real-Time Vehicle and Activity Detection using Smartphone Sensors

Raul Sevillano Gonzalez, Ayoub El Ourrak, Bruno Augusto Casu Pereira De Sousa, Kamran Mehravar
r.sevillanogonzal@studenti.unipi.it, a.elourrak@studenti.unipi.it, b.casupereiradeso@studenti.unipi.it, k.mehravar@studenti.unipi.it

ABSTRACT

In this project we present a framework for the development of a portable Machine Learning model for vehicle and activity detection in mobile devices. From the developed system, a classification model based on a multi-layer perceptron network was designed and trained, using the mobile phone accelerometer data as its input values. With this implementation, the application can detect in real-time the vehicle mode (bus, scooter, or bike) or activity (walking or running) using the extracted frequencies from the accelerometer readings. We also present an efficient data processing algorithm, as well as the use of different data collection techniques, to improve the model accuracy and overall size. The results obtained with the collected data set shows the efficacy of the methods used in the machine learning development, resulting in models with 95% of accuracy after the training. At last, the project provides an Android application, with two distinct modes, to collect accelerometer data for ML training and to test the deployed model in a monitoring mode.

1 Introduction

Context-aware computing is a trend in modern application for smartphones, to identify the user current activity or other relevant information regarding the status of the device. This information is used to provide a better and customizable service for the end users of the software. Many solutions are based on the collection and evaluation of data provided by the may sensors of the phone, such as the camera, microphone, gyroscope, GPS and accelerometer, for example. With the increase in hardware efficiency and computational power, this data can easily e processed and inserted in Machine Learning (ML) models as a way to accurately provide the context information for the high-level applications.

This work focuses on finding a solution that can collect sensor data and provide real-time prediction results in a smartphone application, mainly from the device accelerometer. The system is designed to meet several main goals: it needs low computational resources and low energy consumption, and it needs to be able to respond fast to the changes of travelers' vehicle modes.

For activity recognition, a gradient descent approach was used. For all vehicle modes and driving activities, most of the available research employs the same window size and overlapping ratio. Each driving action, as well as each vehicle mode, has its own cyclic features. These parameter values are picked at random or based on past research. The overall schema for the developed framework in this project is illustrated in Figure 1:

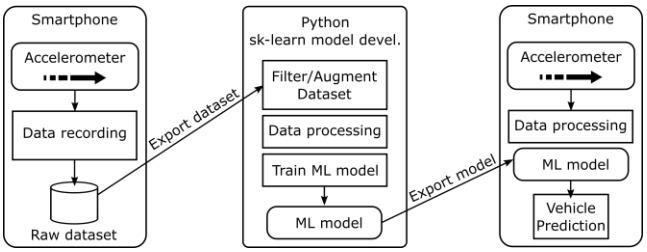

Figure 1. *Model of the vehicle and activity recognition framework*

2 Data Collection

The sensor used for the detection of the vehicle mode is the embedded accelerometer in the smartphone. This sensor was chosen as the analysis of the device movement can provide valuable data for the recognition of different vehicles and activities. The acceleration data is then the preferred parameter to use for the training of the designed neural network (a Multi-layer Perceptron, MLP), as it can be obtained in real-time.

Using the Android platform, the accelerometer data is retrieved with the available APIs, and can be stored in a temporary dataset, to be used later for the ML training. The rate of which the system provides the samples can be defined in the API, and for the current implementation of the vehicle mode detection, the interval between each sample was set to 20 ms. In addition to that, the collection method used for the project was to divide the samples in collection windows, that is, for every time interval the application should collect and store one frame of data. Initially this frame was set to be 5 seconds, providing then a total of 250 accelerometer samplings every window.

Another method for collecting the data, aiming to improve the ML training process, was to overlap the accelerometer samples collected in a time frame. This consists in storing part of

the previous reading in a buffer and using it as the initial parameters for the next window. Multiple values of overlap can be used, and considering this, the developed data collection application allow the customization of the implemented overlapping when performing the collection. For the window size value, however, a manual configuration is required in the code, as this parameter may influence how the data is stored and processed, and thus, must be carefully edited.

The pipeline used in the application for collecting the data is illustrated in Figure 2:

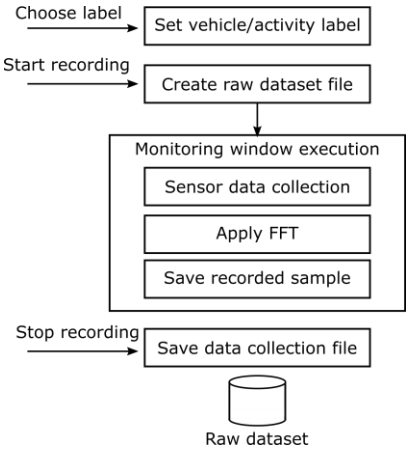

Figure 2. *Pipeline for data collection*

As it is shown in the pipeline, an FFT function is executed on the accelerometer data. This implementation was fundamental for the proper development and training of the presented model for vehicle and activity detection, as data displayed on the time domain cannot be used for the training of a MLP type network. This feature then converts the acquired sensor data from the time domain to the frequency domain, where all the properties of a given signal are maintained on the same axis position.

An example of data collected directly from the accelerometer sensor (provided by the Android API) is shown in Figure 3:

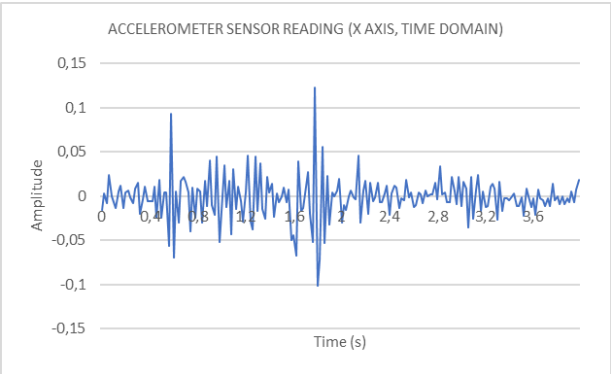

Figure 3. *Accelerometer sample data*

From the plot, it is possible to observe that depending on the time frame that the sensor is triggered, the specific traces of the movement that the phone is doing can be plotted in different parts of the x axis. When training is performed using this type of data, with an MLP, the model does not converge to a predictive network, as from its perspective, the given label could be associated with a signal in any position of the input array.

After the conversion to the frequency domain, it is noticeable that any distinguish feature will be repeated on the same position of the input array, as the signal will produce the same frequency spectrum, even when collected from different time frames. To speed up the FFT computation, the monitoring window for the data collection was then set to 256 samples at 20 ms (5,12 seconds). An example of the accelerometer data converted using the FFT function is illustrated in Figure 4. Only half of the spectrum is used (first 128 values), as the result of the used FFT function is a mirrored image ranging from 0 Hz to 50 Hz.

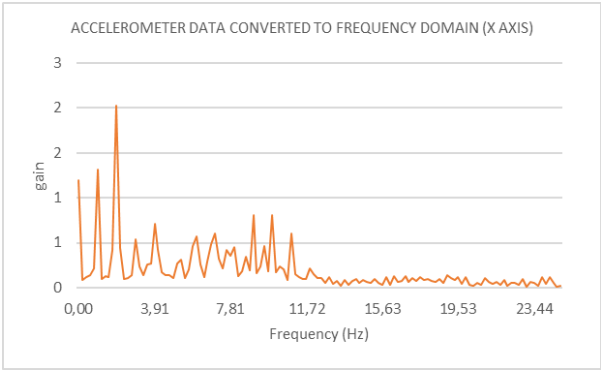

Figure 4. *accelerometer data converted to Frequency Domain*

Another important feature of the data collection performed is that the signal conversion is done for every orientation, that is, the data that is stored contains the frequency spectrum of x, y and z axis. For simplicity, the values were concatenated and converted to a linear scale (the FFT function returns the magnitude values of the samples, using a log scale). The linear correction function used is:

$$gain = 10^{(magnitude/20)}$$

The obtained results of the method for data collection proved to be efficient on the accurate training of th developed ML model. Some example of different frquencies signatures observed in an initial data collection are shown in the plots of Figure 5. It is possible to notice that, when the accelerometer data is converted to frequency, some characteristics are highlited on the spectrum. As an example, the scooter data indicates a higher frequencies in the spectrum, with low amplitude of acceleration, given that the wheels produce this type of oscilation. When observing the walking plot, a different composition of frequencies is noticeable in the spectrum, as the the signature shows a much lower frequency (step frequency) with a higher gain, due to the impact produced by the step.

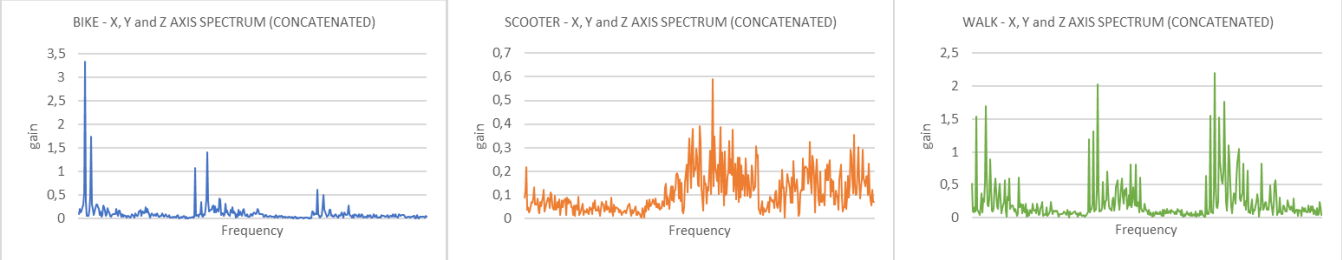

Figure 5. *Examples of vehicles frequency spectrum observed in the data collection*

3 *Preprocessing*

With the available data on the frequency domain, an initial training of the MLP classifier model was executed. The dataset used contained the collected samples mentioned, and the entire sample data was inserted for training, that is 128 frequency points for each orientation axis, resulting an array of 384 columns. From this initial training the overall score of the model was high. However, as the monitoring tests were performed, it was noticed a very poor performance in identifying the activity or vehicle (indicating overfitting). The model flaw was evident as on the monitoring the mobile phone position was changed, as well as the intensity of the activity. This resulted in the misclassification of the performed activity. The following plot show the results of the monitoring using the initially developed model:

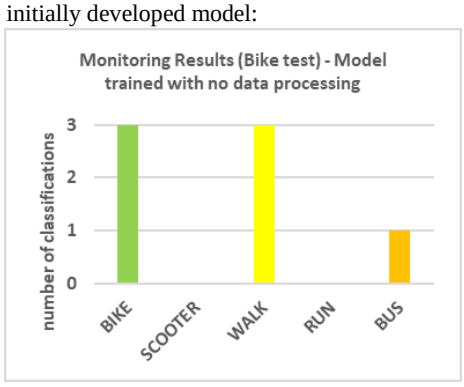

Figure 6. *Monitoring results using initial MLP*

The results show that the model was mistakenly classifying the Bike activity as walking, or even as if the user was inside a bus. Further investigation showed that the cause of this misclassification, was due to the frequency response on the different axis. This effect occurs as the phone is positioned in a different way during the collection, shifting the frequency peak to another axis.

Also, the amplitude of the samples can be significantly different, which makes difficult for a MLP network to learn and perform well.

Figure 7 shows an example of two data collection made in a Bike. On the first sample, the mobile phone was positioned on the pocket of the user, and the result is that the frequency peaks are more predominant on the X axis, with a gain of about 3.

On the second sample, however, the peaks more predominantly on the Z axis, with a lower gain, considering now that the phone was mounted on the Bike.

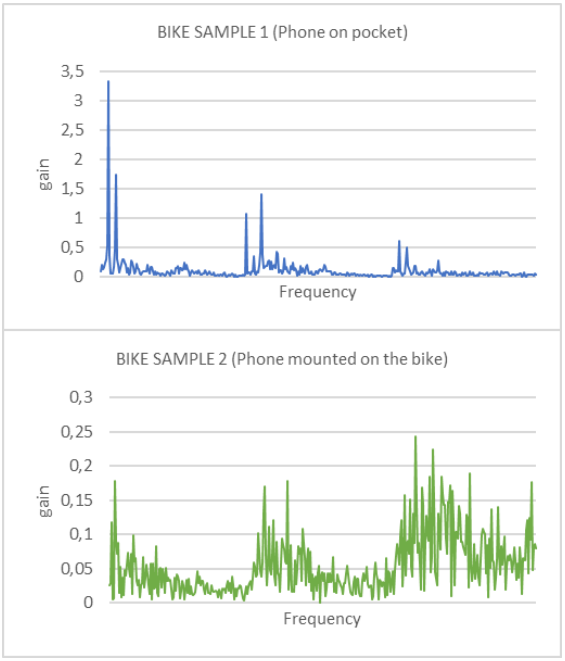

Figure 7. *Examples of different frequency spectrums for the same type of vehicle*

To compensate the axis shifting, the data from all the axis is combined (added) in a single array. Initially this processing was thought to be detrimental to the quality of the data, considering that some information is lost on the procedure. However, as some samples were collected and combined, it was noticed that when adding the data, the predominant frequency (the key frequency to distinguish the vehicle mode) was being highlighted, and even with the change in direction of the phone, the final spectrum was similar between the samples, making it easier for the MLP to identify the desired patterns.

In addition to the axis rotation compensation, the data combination made the final array used for training to be smaller, as now only 128 input values are used for the model development. The reduction of the input array then makes the model overall size to be smaller and more precise, two fundamental characteristics to be

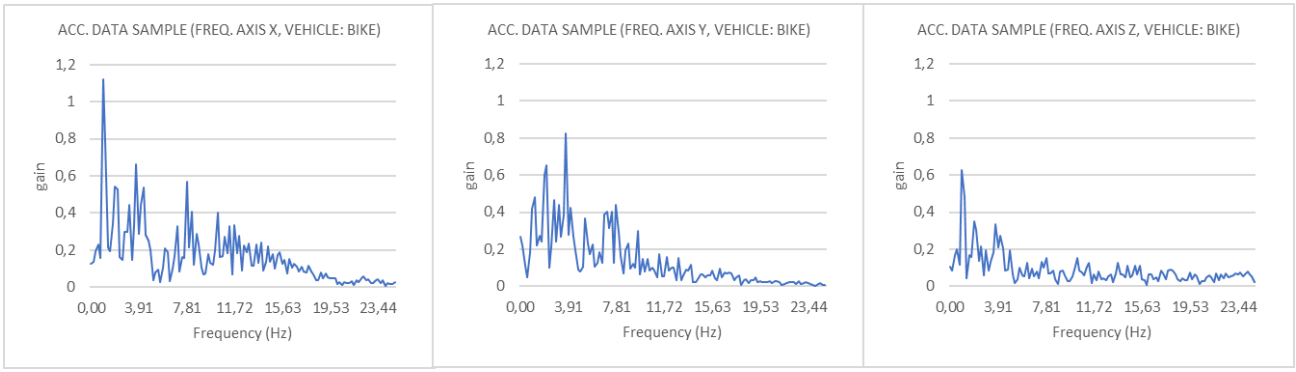

Figure 8. *Frequency data on each orientation axis*

achieved when porting the network to a mobile application. On Figure 8, a sample of data collected is shown on the three different axes. When combined, the result is the spectrum shown in Figure 9. As it can be observed the overall shape of the data was maintained and the predominant characteristics were kept.

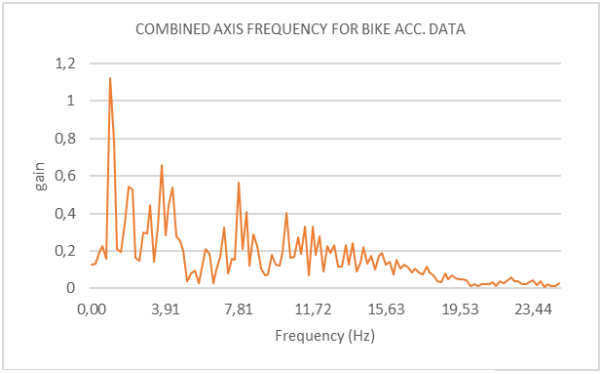

Figure 9. *Combined data*

Complementing the combination of data from the different axis, a second method to improve the data format for ML training was used. This is a known technique used in many applications, and consists in normalizing the data, that is, restricting the gain values of the sample to a fixed range. This transformation compensates for low amplitude samples, that can still contain relevant information for the vehicle identification. The normalization function used is the following:

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Formula 1. *normalization function*

From the linear frequency data obtained from the FFT conversion, the minimum value is always zero for any reading, and the maximum value is computed in real-time for each monitoring window of data reading. This ensures that all data used for the training has a uniform maximum value, and so the MLP training becomes faster and can converge to a more accurate model.

As an example of the effect of the normalization, two signals are shown in Figure 10, on the first plot. Both samples were collected with the phone inside a bus and have a significant amplitude difference. As the normalization function is applied, both signals are now represented on the same scale, as it's shown on the second plot of Figure 10. From this image, it is noticeable that, in spite the amplitude difference, the two signals have similar characteristics on the predominant frequencies, indicating that the information of frequencies distribution is more relevant than the overall gain for each frequency.

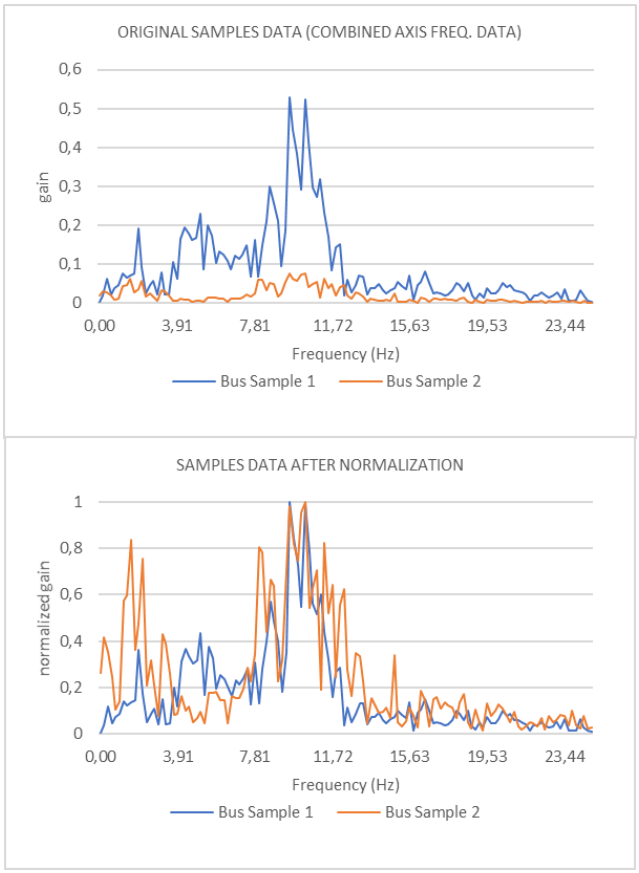

Figure 10. *Original and Normalized sample data*

The performance increase in the developed network will be discussed on the section ML Model development, as all the techniques presented on this section for the data enhancement are used for the MLP training, as well as for the monitoring phase.

4 ML Model development

As we are using scikit-learn library in python instead of using ML-kit android for our machine learning part, we can achieve a lot of advantages, the primary reasons we want to utilize Scikit-Learn instead of ML Kit are that it leads to more generality in our system and that our application will not be confined to Android.

On this project we have used a new concept of machine learning, which is *transfer learning*, and we have only transferred the trained model as a *PMML XML* file to the mobile to use new data according to previous training for our conclusion.

According to our limitations on using client and server communications, and instead of using flask in our application, we have used this methodology:

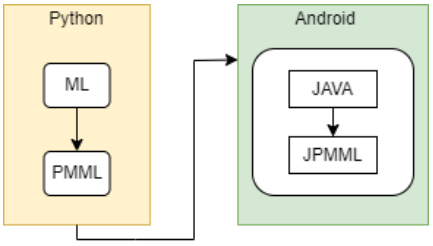

Figure 11. *Network model export method*

Using this implementation, other models can be created and trained according to different application scenarios, being a useful tool to develop and deploy classifiers into the Android platform.

To adapt a MLP to the vehicle and activity detection proposed in the project, the main dataset used to train the model was obtained from different people collecting data. A total of 705, 883, 691, 1040, 364 (total: 3683) samples was obtained for the respective vehicles/activity: Bike, Scooter, Walking, Running and Bus. Noticeably, there is a considerable unbalance in the available individual datasets as acquiring and labeling additional data points was time-consuming and not efficient due to the vehicles being stopped (in that period a lot of noise is introduced to the dataset). However, initial trainings and testing using only the acquired samples were producing highly overfitted models, with very low performance when deployed.

In machine learning (ML), the situation when the model does not generalize well from the training data to unseen data is called overfitting. As we might know, it is one of the trickiest obstacles in applied machine learning. The first step in tackling this problem is to know that our model is overfitting. That is where proper cross-validation comes in.

After identifying the problem, we can prevent it from happening by applying regularization or training with more data. Still, sometimes we might not have additional data to add to our initial dataset. To increase the overall amount of available data, a data augmentation method was used.

Data augmentation isn't just about avoiding overfitting. A big dataset is critical for the performance of both ML and Deep Learning (DL) models in general. However, by supplementing the data we currently have, we can increase the model's performance. This suggests that Data Augmentation can help improve the model's performance. For the current implementation the augmentation process only consists in injecting noise in the samples, to produce new data. Then at the end of this data processing section, the expanded dataset obtained contains a total of 10.475 samples (approximately 2.000 samples for each label).

After the augmentation, a filtering function was used, to remove any sample with very low amplitude, that could be considered as only noise. With the filtering process, the ML dataset used to train the network contained 10.244 samples.

The model developed contains only one layer of 100 neurons (default config from sk-learn MLPClassifier network) and was trained with a maximum of 100 iterations. The result from this training shows a total score of 95.5% of accuracy, considering over 2.000 samples for testing. Results are shown in the following confusion matrix obtained:

PREDICTION	BIKE	SCOOTER	WALK	RUN	BUS
BIKE	0,9242	0,0298	0,0275	0,0093	0,0166
SCOOTER	0,0203	0,9468	0,0085	0,0019	0,0037
WALK	0,0370	0,0000	0,9535	0,0019	0,0111
RUN	0,0055	0,0043	0,0000	0,9870	0,0055
BUS	0,0129	0,0191	0,0106	0,0000	0,9630
	BIKE	SCOOTER	WALK	RUN	BUS
	TRUE LABEL				

Figure 12. *Monitoring Result for walk and bike samples*

The result from the training shows that the model was converging to a good approximation of the real vehicle types, being able to identify patterns in the frequency spectrum provided as input. The final model size had an approximate value of 500KB, being suitable for its target application on mobile devices.

When deployed, the monitoring results were also promising, even when collecting the data in a different manner than when the training data was collected (phone in alternative positions or with slower walk pace, for example). To build the final prediction, the monitoring will check for the highest classification value. On the example provided in Figure 13, a total of 8 windows were collected, then by computing the most frequent result, the monitoring application infers the vehicle or activity that the user is on.

Figure 13. *Monitoring Result for walk and bike samples*

5 *Android application*

The app it's divided into 3 main activities (Select mode, data collection and monitoring) and support night and light mode. On the Main activity just have two buttons (Figure 14) that are used to select between the two modes that app allows to use.

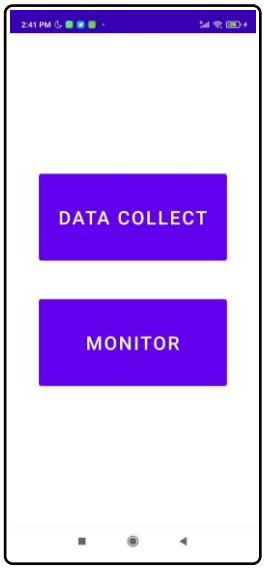

Figure 14. *Main activity window on light mode*

5.1 Data Collection

On this activity we present 5 buttons (Figure 15) that are used to select in which travel mode we going to collect the data. There are 5 different travel modes Bike, Scooter, Walk, Run and Bus, by default the travel mode taken is Bike. This window also allows the user to apply the overlap technique to the data, allowing the user to modify it from none to 75% using a progress bar, default value for overlap is 0%.

After setting this params the user can start recording data from accelerometer by clicking on the record button. This data is recorded on windows of 256 accelerometers records. After having a fully window the FFT algorithm is applied to whole data and magnitude is stored in a csv file on the app files folder of app package in the android storage in the following format: label, UUID, mx0, mx1....mxi, my0, my1.....myi, mz0, mz1.....mzi. Also, each window is stored in a different file depending on the current overlap value. This data will be used later to train a backend model that will be used to predict the mode during monitor phase.

At the left of the record button, we can see a remove button that is just used to remove the current data files that have been recorded in case we have downloaded or just contains fake data. Down these buttons there is a chronometer that shows to the user the amount of time that he has been collecting data. At the bottom there is also a text field that shows the user info about application events like remove files or stop recording.

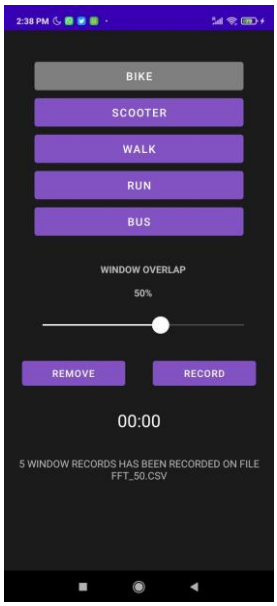

Figure 15. *Data collection window*

5.2 Monitoring

This window is divided into two different states, monitoring and show results. During the monitoring phase we can just see a chronometer and a button (Figure 16, left image), the chronometer starts by clicking on the monitor button and stay there until results are shown. At this point the device starts collecting data from accelerometer sensor and process it on same way as on data collection activity. On this case the sensor just collect data for 6 windows and store it into a temporal csv file on same format as in data collection but without the label and the UUID.

When 6 fully window has been collected and processed, the temporal file is taken as input into the machine learning model, which is automatically install with the application. This model returns as output 6 different results (one for each window) that predict the travel mode for each of the windows. Finally, the results are presented in a plot (Figure 16, right image) showing the probability that each mode has. If user want to check again the predicted travel mode, just need to press the monitor button after the results have been plot.

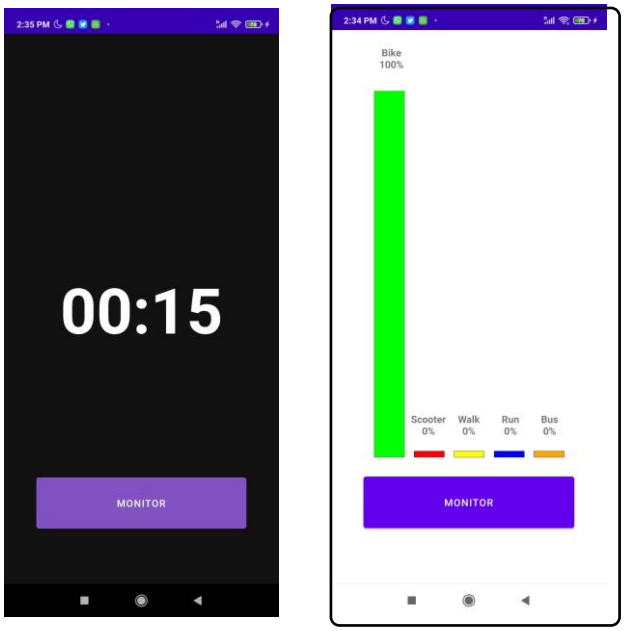

Figure 16. *Monitor window during the collection and result screen*

5.3 Application compatibility and permission

To run the app the minimum SDK version that android device should have been 24.0 version. Also, our application, don't have any permission requirements since it not relies on camera or GPS to run it.

6 Conclusions

On this project we propose a model that predicts the vehicle mode of the user, using the orientation of the smartphone. The data collected from accelerometer is divided in windows of 256 samples and each of this samples is processed using the FFT algorithm to extract the magnitude and frequency. For the monitoring phase just 6 samples are recorder per run while on collection phase records are store until stop button is pressed, this collected is used to train the monitor model. Our model achieves an average accuracy of 84.37% on the travel mode. Moreover, this accuracy could increase if we were able to collect more data to train our classifier and been able to have more data with overlap technique.

REFERENCES

https://github.com/cawfree/android_accelerometer_fft FFT algorithm implementation
https://www.mdpi.com/1424-8220/18/4/1036
https://journals.sagepub.com/doi/abs/10.1177/0361198121990681