Activity Recognition using KNN Classification and Transfer Learning

Dietmar Malli

***Abstract*—This report describes the development of a mobile application which enables its users to track their activities using a K-Nearest-Neighbour (KNN) classifier as well as two machine learning based approaches. It also allows them to compare their performance immediately on device by providing graphs depicting the predictions of these algorithms.**

***Index Terms*—MCL, Mobile Computing Laboratory, TU Graz, Android, Transfer Learning, Activity Recognition, KNN Classification, Base Model, Head Model, Machine Learning**

I. Introduction

THE goal of this laboratory was to develop a mobile application which recognizes acitvities performed by the user of a smartphone. During the first part students were introduced to android as development platform as well as how sensors can be read on this platform. Further, a mobile application which recognizes a users activity based on the KNN algorithm had to be implemented. The decision which sensors to use, as well as which features to compute was also left open by the course organization. During the second part the KNN algorithm had to be compared to machine learning approaches. In detail a comparison to a completely pre-trained model, as well as a transfer-learning based model had to be done. This was done by extending the application from the first part of the laboratory.

Throughout the whole lecture it was tried to classify the same six categories of activity introduced by the WIreless Sensor Data Mining (WISDM) dataset[1]:

- sitting
- standing
- walking
- jogging
- going downstairs
- going upstairs

II. Sensor Considerations

At the time of writing, the Android Platform provides a large number of sensors[2]:

- accelerometer including gravity
- ambient temperature
- accelerometer excluding gravity
- current rotation of the phone
- ambient light
- ambient magnetic field
- ambient air pressure
- proximity to screen
- ambient humidity

Manuscript sent June 08, 2020; not yet revised.

For detecting a users activity accelerometer and rotational sensors are obviously the most useful ones. The first idea pursued was to use a combination of rotational sensors as well as acceleration excluding gravity. After taking some measurements this approach was abandoned, because rotation is provided as number which will flip from its maximum value to its minimum at the end of its positive range. As depicted in Fig. 1, this would make feature extraction much more complicated. Using the acceleration sensor without gravity is also a bad idea as this leads to an information loss. If the user has his phone in his pocket while sitting, gravity can be observed on the Z axis. In contrast it will be on the X axis, while he is standing.

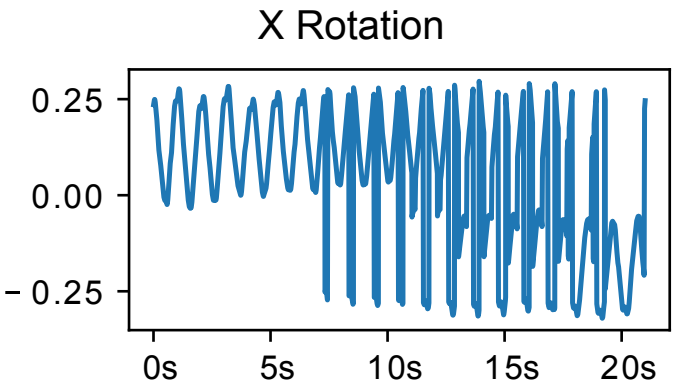

Fig. 1. Examplary rotational sensor output recorded while walking.

This leads to the conclusion, that just using the acceleration sensor which includes gravity will lead to features which are easier to handle.

III. Feature Selection and Windowing

The next task of this laboratory was to record measurements using the phone, transfer this measurements to a computer and define features which can later on be used in a KNN algorithm. To achieve this every class of activity was recorded four times. Three of these recordings were used for training and one to test classification using different features and ways of calculating them. In a first try the following features were calculated per test file:

- minimum of x acceleration
- minimum of y acceleration
- minimum of z acceleration
- maximum of x acceleration
- maximum of y acceleration
- maximum of z acceleration
- mean of x acceleration

- mean of y acceleration
- mean of z acceleration
- standard deviation of x acceleration
- standard deviation of y acceleration
- standard deviation of z acceleration

This sums up to feature vectors whith a size of 12. For the KNN algorithm this means that three test files times six classes lead to a total of 18 samples. For the easier to detect classes like sitting and standing, this is already enough to work reliably. Classes which are harder to separate need a windowing approach which can be seen by comparing Fig. 2 to Fig. 3. Windowing was implemented by calculating one

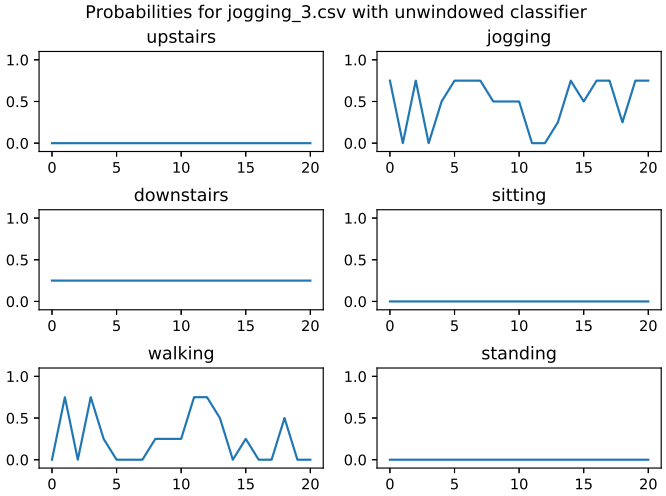

Fig. 2. Classification with features generated without windowing.

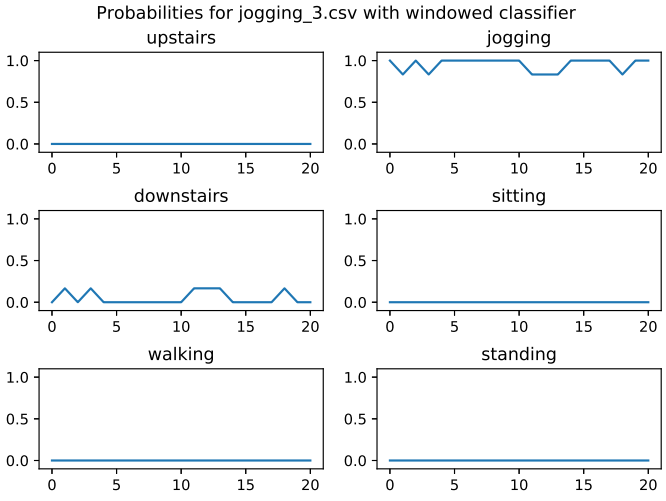

Fig. 3. Classification with features generated with windowing.

feature every 1.2 s over all files. To avoid the bias introduced by longer recordings leading to more features, the minimum amout available in all classes was chosen as a limit. The remaining feature vectors were discarded.

IV. Evaluating the KNN algorithm

In the end of the first part of the laboratory, the KNN algorithm had to be implemented on the smartphone making use of the features precalculated on the computer. A video was recorded to demonstrate that the application is working. The recording was produced making use of the application `OBS-Studio`[3], as it is capable of mixing multiple streams. In detail, the screen of the smartphone as well as a camera stream of the recording device were used as sources.

Fig. 4 shows one frame of the video. This frame shows the probabilities calculated by the KNN algorithm on the smartphone over time while the author of this report is jogging.

Fig. 4. One frame of the proof video demonstrating the KNN algorithm in action. The graph at the bottom depicts the probability of jogging.

V. Developing a Base Model

The course organization introduced the students of this laboratory to machine learning approaches by showing them how to train a model using the publicly available WISDM dataset[1]. Performance of the provided example code is depicted for comparison in Fig. 5. From this starting point

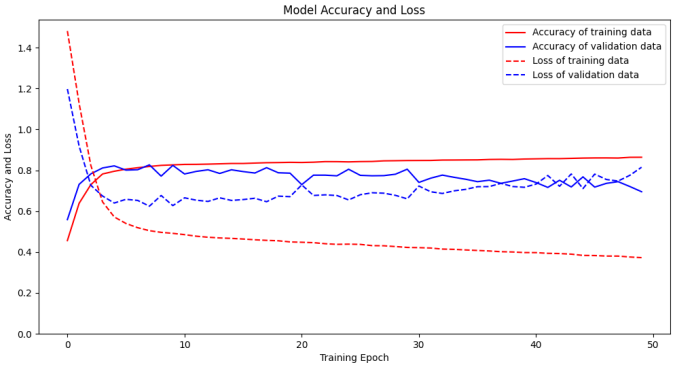

Fig. 5. Performance of the base model example code provided by the course organziation.

a lot of optimizations were tried.

The first idea was to normalize the raw acceleration data in a different way than in the example code, as better normalized input data leads to faster convergence of machine learning models[4]. The results shown in Fig. 6 on the next page were generated using the `StandardScaler` module of `scikit-learn`[5]. Unfortunately the validation accuracy using this method performed slightly worse than the one of the given solution and the validation loss also increased.

The second idea was to split the training data by activities. The code given by the course organization just windows over

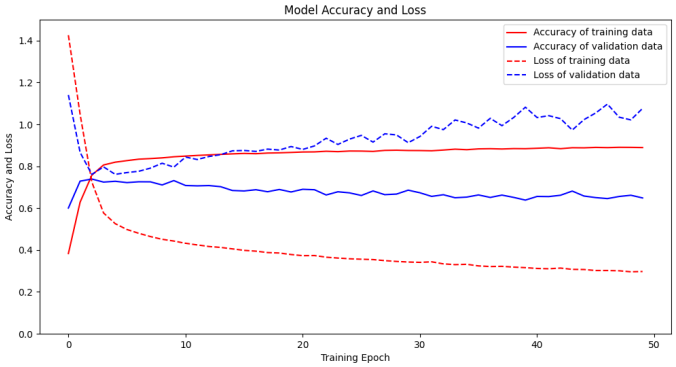

Fig. 6. Performance of the base model with better normalized input data using `StandardScaler` of `scikit-learn`.

all training data and assigns it with the label of the majority of samples. As depicted in Fig. 7, implementing this change made it impossible for the network to converge.

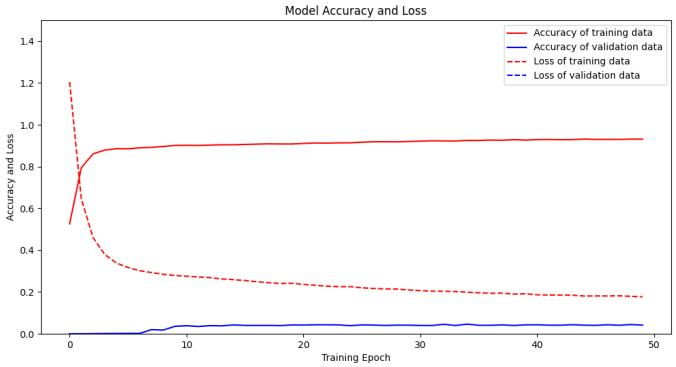

Fig. 7. Performance of the base model with input data split by activities before windowing and labelling.

The third approach was to change the network structure from just a combination of dense layers to:

1) convolutional layer with a height of 64
2) dropout layer with 20 %
3) convolutional layer with a height of 32
4) dropout layer with 20 %
5) convolutional layer with a height of 6
6) global average pooling layer
7) dense layer with softmax activation

Note that the last convolutional layer results in a data cube with the 6 output classes as height. The global average pooling layer makes this convolutional layer learn the required pattern for identifying these classes. This approach was inspired by Martin Görner[6]. It results in much less computational effort than finishing convolutional layers with one or more dense layers. The performance of this model during training is depicted in Fig. 8.

In the end the improved base model's performance was measured using test data which was not shown to it during training or validation. The performance of the authors final model on this test data is depicted in Fig. 10 on the following page and can be compared to the model provided by the course organization in Fig. 9. Accuracy and loss values are compared in table I.

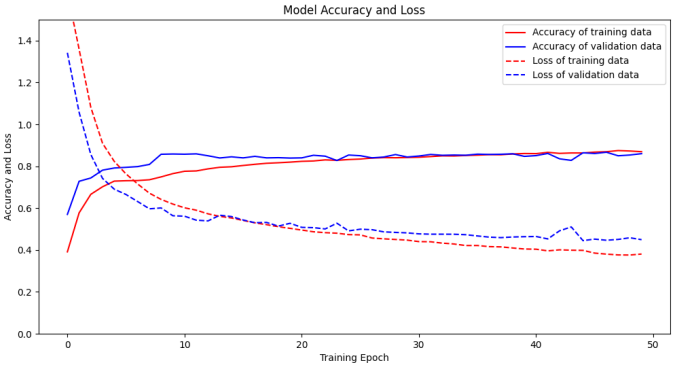

Fig. 8. Performance of the base model with convolutional layers.

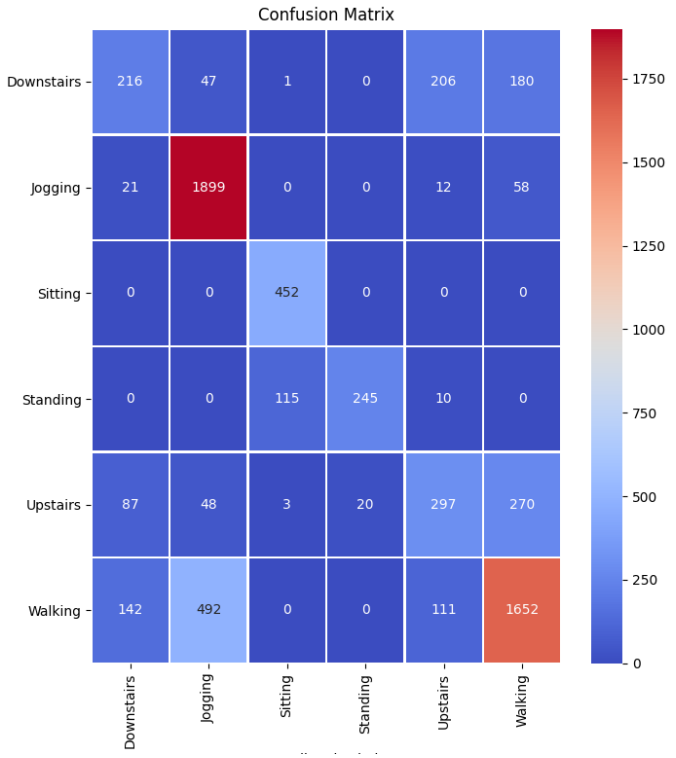

Fig. 9. Confusion matrix showing the performance of the model provided by the course organization.

VI. Creating the Head Model

For transfer learning a pre-trained base model is cut at a certain layer and new untrained layers are appended to it. These layers are trained directly on the target device. The code to do so was initially developed by Aaqib Saeed[7] as an example which is able to identify two separate classes. It was extended to the six classes described in section I on page 1 and adapted to the base model developed in section V on the

	Accuracy	Loss
Course Organization	0.72	1.13
Author	0.77	0.57

Table I
Performance of the authors model compared to the model given by the course organization.

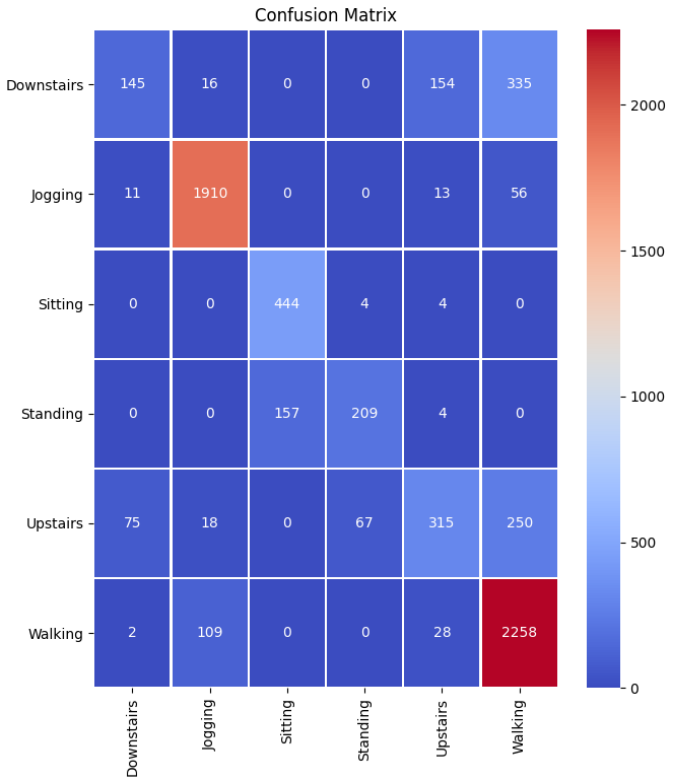

Fig. 10. Confusion matrix showing the performance of the model by the author.

preceding page.

VII. Android Implementation

Most of the code for implementing transfer learning and running `TensorFlow` models on an android device was already developed by Aaqib Saeed[7] and the `TensorfFlow Lite` framework. The graphical user interface (GUI) was implemented from scratch and plots the prediction results of the KNN algorithm, the base model and the transfer learning model over time. A settings frame which is shown in Fig. 11 of the app makes it possible to tell it the current position of the smartphone on the users body.

VIII. Evaluating Machine Learning approaches

The final goal of this laboratory was to compare classification results of the KNN algorithm with the base model and the transfer learning model. This was again done using a video recording as described in section IV on page 2.

Fig. 12 shows one frame of this recording in which the author of this report is standing. KNN as well as the transfer learning model would correctly classify standing, as the smartphone is lieing in an angle similar to being in the pocket where the KNN measurements were taken. Unfortunately the base model thinks the user is going upstairs.

Fig. 13 on the following page shows a screenshot which was taken while the user was sitting with the phone in its pocket. This situation also displays, that KNN and the transfer learning model trained to this location of the phone are working. In contrast Fig. 14 on the next page shows a screenshot which

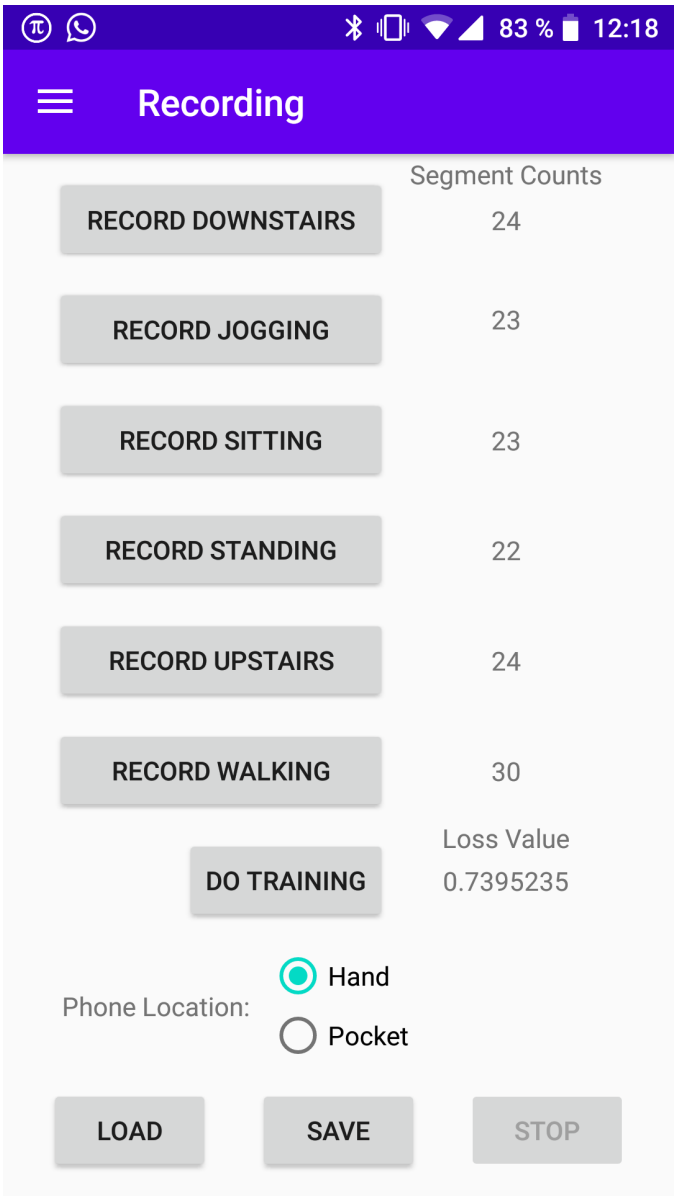

Fig. 11. Screenshot of the settings frame of the android application.

Fig. 12. One frame of the proof video demonstrating the comparison of all implemented algorithms.

was taken while the user is walking and has the phone attached to its hand like in Fig. 12. In this situation the KNN algorithm performs worse, because it was trained with the phone in the pocket. The transfer learning model was trained on device with the phone attached to the users hand. This is why it is able to correctly classify walking in this situation.

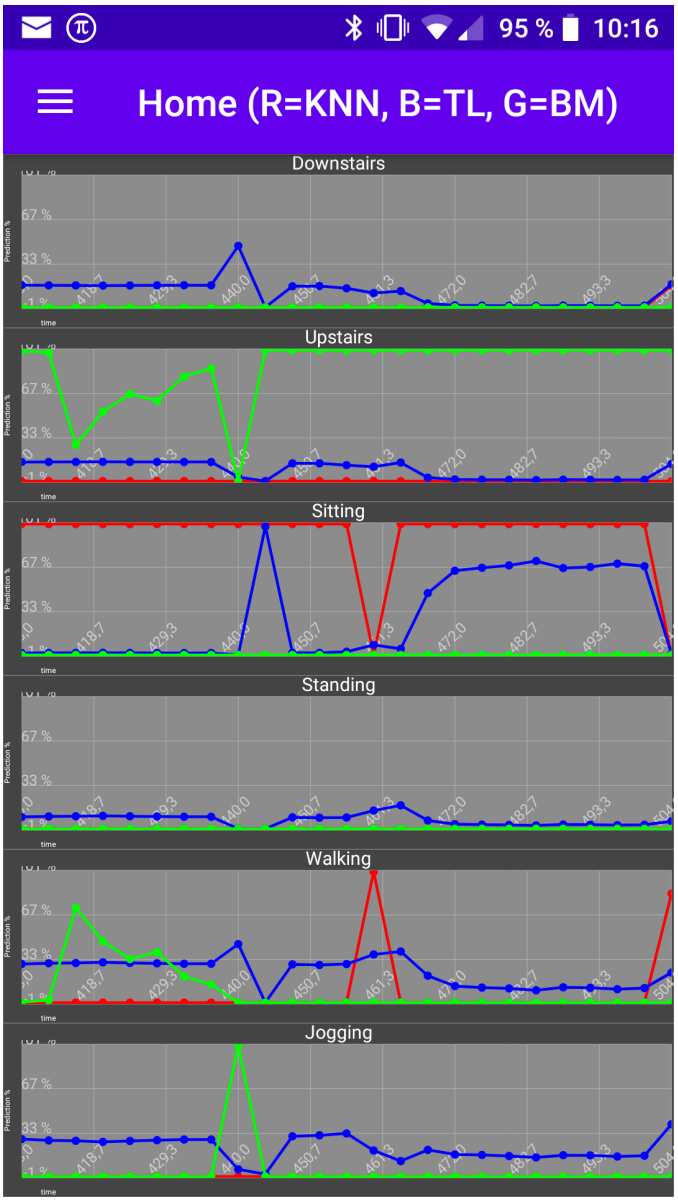

Fig. 13. Screenshot of the application taken while the user was sitting with the phone in its pocket.

To have a quantitative estimation of the apps performance, a logging functionality was implemented in the end. This made it possible to record classification results of all three methods into a file. One file per class and smartphone position was recorded and later on evaluated on a PC. This led to the results depicted in table II.

IX. Conclusion

As shown in section VIII on the preceding page, the KNN classifier worked better than expected as long as the phone is

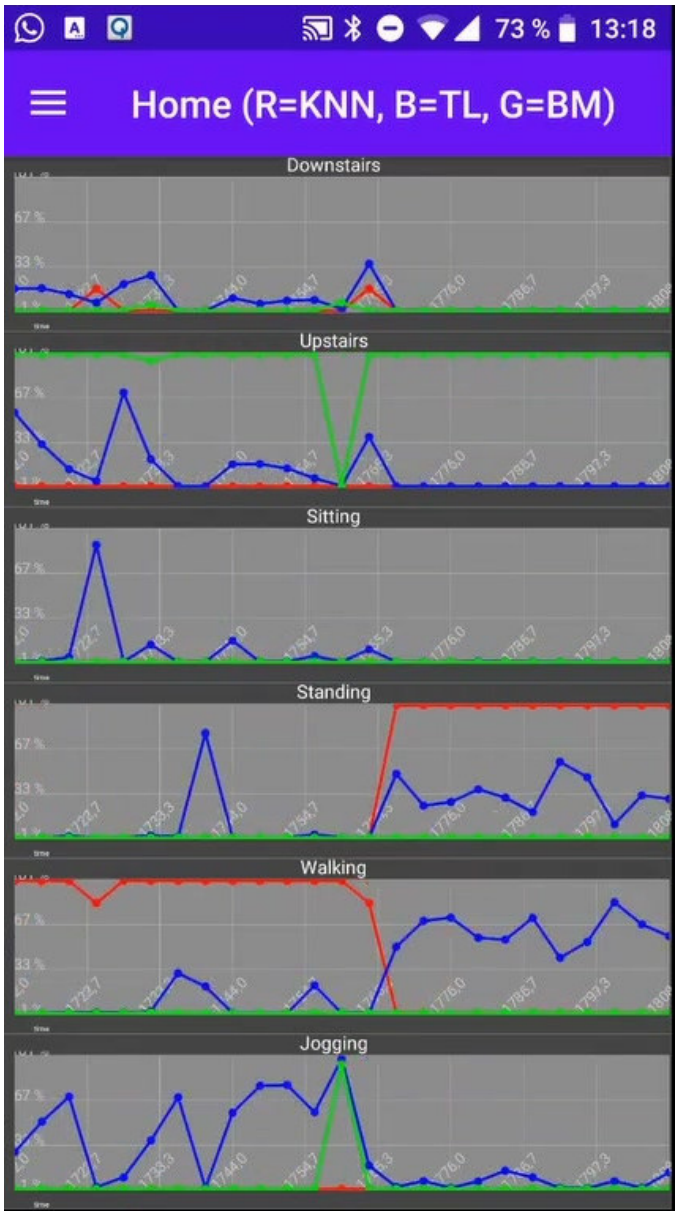

Fig. 14. Screenshot of the application taken while the user was walking with the phone attached to its hand.

	KNN	TL	BM
downstairs, phone on hand	0.0	65.5	0.0
jogging, phone on hand	0.0	34.6	88.8
sitting, phone on hand	0.0	0.0	0.0
standing, phone on hand	100.0	62.4	0.0
upstairs, phone on hand	0.0	39.2	0.0
walking, phone on hand	0.0	28.5	0.0
downstairs, phone in pocket	34.7	0.0	0.0
jogging, phone in pocket	12.0	41.1	94.1
sitting, phone in pocket	100.0	100.0	0.0
standing, phone in pocket	100.0	0.0	0.0
upstairs, phone in pocket	0.0	0.0	0.0
walking, phone in pocket	88.0	100.0	0.0

Table II
Classification performance of all three algorithms in percent.

at the position at which it was during the initial recordings. The base model pre trained by the WISDM dataset did not provide good results in this case. The transfer learning based approach made it possible to train the neuronal network to a new smartphone position directly on the device.

APPENDIX
DEVELOPMENT REPOSITORY

All data created during this laboratory was initially versioned at https://git.tugraz.at/MobileComputingLab.git using the credentials provided to the system at https://git.tugraz.at/ to prevent plagiarism blames. It is now publicly available at https://github.com/didi1357.

ACKNOWLEDGMENT

The author would like to thank the whole Mobile Computing Laboratory team for their time and dedication.

REFERENCES

[1] J. R. Kwapisz, G. M. Weiss, and S. A. Moore, "Activity Recognition using Cell Phone Accelerometers", *Proceedings of the Fourth International Workshop on Knowledge Discovery from Sensor Data*, 2010.

[2] G. LLC. (Dec. 27, 2019). Sensors Overview, [Online]. Available: https://developer.android.com/guide/topics/sensors/sensors_overview.html (visited on 05/31/2020).

[3] H. Bailey. (2020). Open Broadcaster Software, [Online]. Available: https : / / obsproject . com (visited on 06/03/2020).

[4] U. Jaitley. (Oct. 8, 2018). Why Data Normalization is necessary for Machine Learning models, [Online]. Available: https://medium.com/@urvashilluniya/why-data-normalization-is-necessary-for-machine-learning-models-681b65a05029 (visited on 05/31/2020).

[5] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python", *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[6] M. Görner. (Jan. 28, 2019). TensorFlow, Keras and deep learning, without a PhD, [Online]. Available: https://codelabs.developers.google.com/codelabs/cloud-tensorflow-mnist (visited on 05/31/2020).

[7] A. Saeed. (2020). On-device Activity Recognition, [Online]. Available: https://github.com/aqibsaeed/on-device-activity-recognition (visited on 05/31/2020).

Dietmar Malli was born in Deutschlandsberg, Styria, Austria in 1995. He went to local schools there and is working for the company mse elektronik GmbH in Graz since 2012. Currently he is studying Information and Computer Engineering at the Technical University Graz where he also earned a Bachelor's degree in this program of study.