Mobile Computing Laboratory 2021 Assignment Report

Tobias Lafer *Technical University of Graz*, Austria

***Abstract*—In the *Mobile Computing*-laboratory, at TU Graz, the students learn in a hands-on way the fundamentals of developing applications for state-of-the-art mobile computing devices. As a final result, the students create an Android-application for either a pre-defined (for beginners) or a self-chosen (for experienced students) task. Due to the missing experience app-development, the pre-defined task was chosen by the author.**
This document describes and discusses the essential ideas and concepts of the developed app as well as the derived performance and the raised issues.
The source-code of the app as well as the data-set and all additional scripts etc. can be found on [1].

I. Introduction

The pre-defined task is divided into two parts: First, activity-monitoring based on a rather simple classifier, like *k-nearest-neighbors (kNN)*, and secondly, phone-position independent activity monitoring through on-device transfer-learning.
In more detail: A set of activities should be chosen by the student and a simple-classifier should be designed to distinguish certain activities via the captured sensor-data on the device. Thereby, the required data-set for training the classifier should be self-captured. For the second part, a pre-captured data-set for similar should be used to design a neural-network based classifier. Using transfer-learning, this classifier should then be trained on a smaller, on-device captured data-set for the current phone-position.
To ease the app-development, a software-library was provided implementing the transfer-learning procedure itself [2] to let the students focus on developing the classifier itself as well as the necessary signal capturing and pre-processing.

II. *kNN*-classification

The *k-nearest-neighbors*-algorithm [3] is a simple but very powerful classification algorithm, based on a majority-voting of the distances between the current individual to classify and a set of labeled reference-individuals. Therby, no formal restrictions on the distance measurement are given (one can even define its own 'distance'), but typically one uses a certain type of *norm* [4], mostly the well-known *euclidean norm* [5]. After calculating the distances, one sorts them in ascending order and counts the classes of the k first distances. Therby, k is an design-parameter which has to be properly chosen. The final class of the individual is then the one with the highest count of this k closest reference-individuals.
Listing 1 shows a *Python*-implementaion of kNN. The classifier implemented in the app is also based on this code.

```
def kNN(features_in, features_ref, labels_ref,
        k, N_classes):

    differences = features_ref - features_in
    euclidian_distances =
        np.linalg.norm(differences, axis=1)
    sorted_indices = np.argsort(euclidian_distances)
    sorted_labels = labels_ref[sorted_indices]

    return k_majority_voting(sorted_labels, N_classes, k)

def k_majority_voting(data, k, N_classes):
    counters = np.zeros((N_classes), dtype=int)

    for l in range(k):
        counters[data[l]] += 1

    sorted_indices = np.argsort(counters)
    return sorted_indices[-1]
```

Listing 1. A simple python-implementation of *kNN*.

III. Activities for the *kNN*-classification

Four well-known body-strength exercises were chosen as activities to be classified. These activities were selected as each of them has a quite unique motion-profile, and were therefore supposed to be easily distinguishable by *kNN*. A sketch of each activity is shown in figures 1 to 4. Thereby, the phone is placed as shown in figure 5.

Fig. 1. Bizeps Curls [6]

Fig. 2. Triceps Curls [7]

Fig. 3. Crunches [8]

Fig. 4. Russian-Twist [9]

IV. Signal processing

The data from both, accelerometer and gyroscope sensors of the phone, are continuously captured in an event-based manner (see [10]). The actual classification is then executed with a period of one second in separate thread. A sketch of the corresponding signal-processing pipeline is shown in figure 6.

1) **Data-Capturing**
The Android operating-system offers classes and routines to handle the event-based capturing of on-device sensor-signals [10]. Thereby, an event is triggered each time one of the registered sensors delivers new values. The derived x-, y- and z- signals per sensor are then

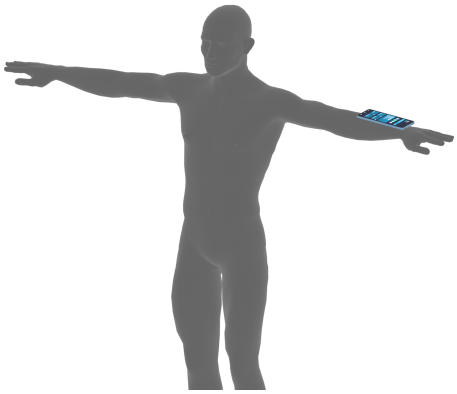

Fig. 5. Phone position as used for the *kNN*-classification task.

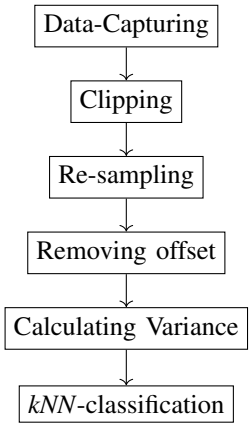

Fig. 6. Outline of signal-processing pipeline.

pushed to fix-sized buffers including their timestamps. If a buffer buffer is full, the oldest values are removed from it when pushing the new ones.

2) **Clipping**
It was found that a time-period of 4 seconds is enough to classify the selected activities. Therefore, the time-series contained in the capturing buffers are clipped to this period.
3) **Resampling**
Next, the clipped time-series are re-sampled to a uniform sampling rate of 50 Hz by linear interpolation between the two neighbor samples. Each sensor-channel then contains exactly 200 samples.
4) **Offset removal**
The activities to monitor are periodic, so only the AC-part of the single timeframes are required.
5) **Variance calculation**
The actual features used for classification are the variances of the single sensor-channels:

$$var\left(\{x_{i,k}\}\right) = \frac{1}{N}\sum_{k=0}^{N-1} x_{i,k}^2 \quad \forall i = 1...6 \qquad (1)$$

6) ***kNN*-classification**
The previously calculated variances are then fed into the *kNN*-algorithm using $k = 10$. The *euclidean norm* is used as distance-measurement.

V. Applying actual data to *kNN*

A set of samples for each activity was captured and split into the required 4 seconds timeframes, each overlaping for 3 seconds. The data-set was separated into a 'training' and a 'validation'-set, with a validation-split of 25 %. The distributions of the single activities in both sub-sets are shown in figures 7 and 8.

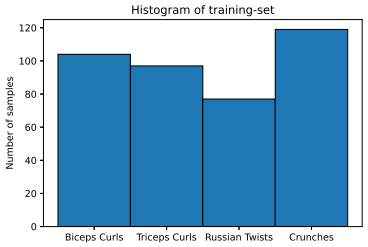

Fig. 7. Activity-histogram of the training-set.

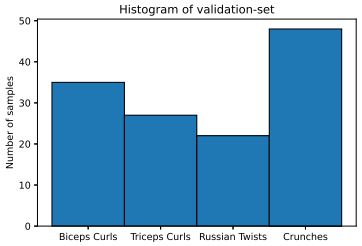

Fig. 8. Activity-histogram of the validation-set.

If an clear distinction between the 4 different classes is actually possible via *kNN* can be investigated by a so-called *pair-plot* [11]. Thereby, a set of sub-plots is generated and set up in way to unroll the, in this case, 6-dimensional feature-vectors in a 2D plot pane. The pair-plots for both, training and validation sets are shown in figures 9 and 10. The off-diagonal plots are of main interest, as these plots show the relationships between one selected feature and all remaining features. If a clear grouping is visible in one of the plots, the corresponding feature-pair can be used for classification.

The performance of the classifier can nicely be visualized by a so-called 'confusion-matrix'. Therby, the samples from the validation-set are consecutively fed into the *kNN*-classifier. The predicted and correct classes are then plotted in a 2D grid. A perfect classifier contains only elements on the main-diagonal grid faces.
Due to the quite clear grouping visible in the pari-plots a clear distinction between the activities can be assumed. The proof of this assumption is given by the confusion-matrix in figure 11, where one can observe that all samples were classified correctly.

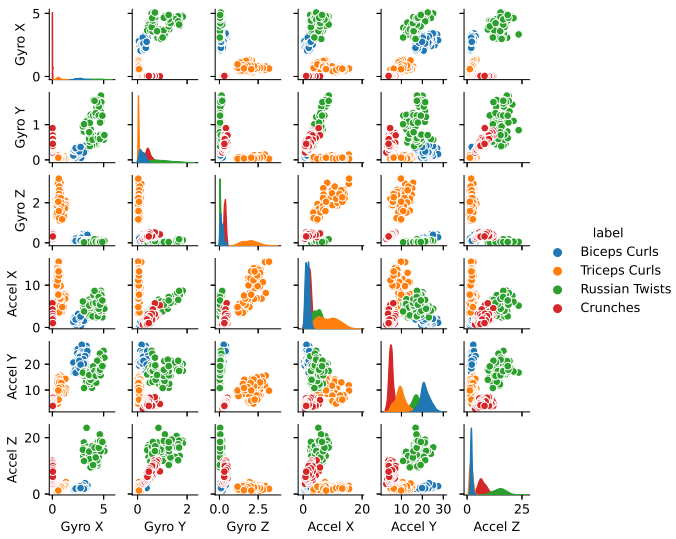

Fig. 9. Pair-plot of the training-set.

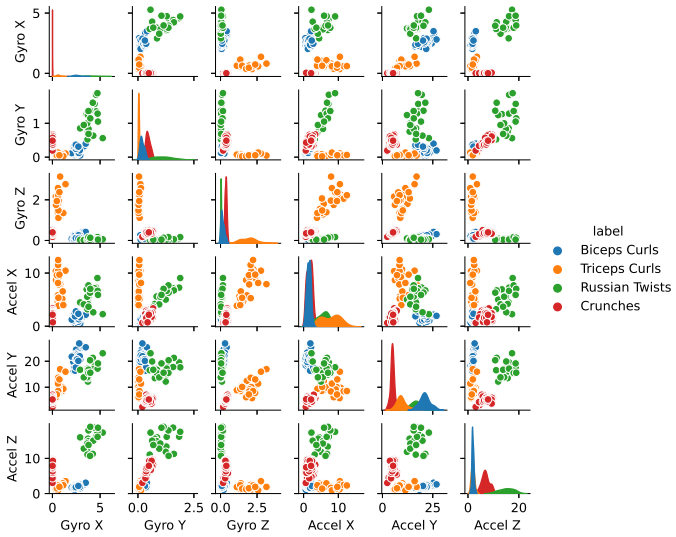

Fig. 10. Pair-plot of the validation-set.

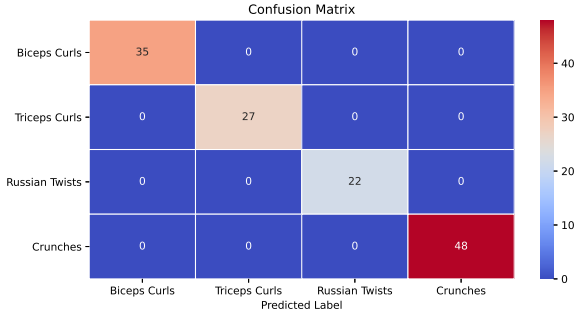

Fig. 11. Confusion-matrix for the *kNN*-classifier.

VI. Transfer-Learning Basics

Changing the phone's position when attempting to classify an activity will typically lead to a massive drop in accuracy, although the underlying activity is the same. *Transfer learning* is one solution to this issue. Thereby, an existing machine-learning model, the so-called *parent-model*, typically trained on a huge data-set, is slightly modified and re-trained on a new but typically smaller data-set. If both, old and new data-sets, are somehow related to each other (e.g. same activities but different phone positions), one could derive similar classification performance, but without designing and training a model from scratch.

In the second part of this assignment a neural-network is used for classification, trained on a large data-set for one dedicated phone-position. Afterwards, the output-layer as well as some of the top hidden layers are removed and a new, so-called *head-model*, is attached to the remaining parts of the parent-model. Throughout the rest of this paper, the remaining parts of the parent in the child-model is called *base-model*. The parameters of the base-model are fixed so that only the head of this new *transfer-learning model* (*TFL*-model) is trained for a new phone-position **on-device**.

VII. Activities, Data-set and Model-architecture

Due to several issues during development and with first phone used in this assignment, a totally different set of activities were used for the second part. Further, no pre-captured data-set was used, but the author captured the data-sets for both, parent and child-models himself. The reasons for this decisions are discussed in detail in section X.

The following activities were chosen to be classified:

- Walking on a plane surface
- Walking Upstairs
- Walking Downstairs
- Running

The parent-model is designed for the phone being placed in the front-left trouser pocket with an orientation as shown in figure 12. Although universally placeable, the phone was placed on the front-right pocket as shown in figure 13 for validating the child-model. Data-sets were captured for both phone-positions and split into training- and validation-sets using again a validation-split of 25 %. The corresponding histograms are shown in figures 14 to 17.

The actual model architectures for both, parent and head-model are shown in figures 18 and 19. The essential part of the parent model is a 2D convolution layer, depicted in more detail in figure 20.

The transfer-learning part utilizes a similar signal pre-processing scheme as for the *kNN*-classifier (see figure 21). Thereby, time-frames with a length of 4 seconds are resampled by linear-interpolation to a sampling-rate of 50 Hz, resulting in a total of $N_{Samples} = 200$ samples of each of the 6 available sensor channels. Each channel is then normalized by

$$\{\hat{x}_{i,k}\} = \frac{1}{max\left(\{|x_{i,k}|\}\right)}\{x_{i,k}\} \quad \forall i = 1...6 \tag{2}$$

The resulting $(N_{Samples} \times 6)$-matrix is then fed into a 2D convolution layer, setting up one 2D convolution kernel for each of the 4 activities. The idea behind this architecture is that each activity is characterized by one or another unique motion pattern in the sensor-channels. As there is one kernel available for each activity, the layer has the ability to derive one optimal kernel for each activity. The pooling-layer following the convolution-layer then just reduces the number of samples fed into the final dense-layers to reduce the number of tunable parameters. By a try-and-error approach, it was found that the higher the number of neurons in this dense-layers, the better the accuracy on the validation set. As the parent-model is not trained on the device directly, a quite high number of neurons was chosen. The head-model for transfer-learning, on the other hand, has the same base-architecture as the head of the parent-model, but utilizes a smaller number of neurons to speed-up on-device training.
All layers except the output-layers of both models use *ReLU*-activation. As the basic task of this model is a classification-task, *softmax*-activations are used for the output layers.

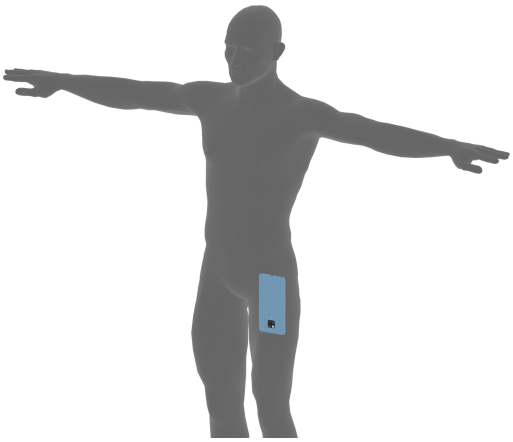

Fig. 12. Phone-position for the parent-model data-set.

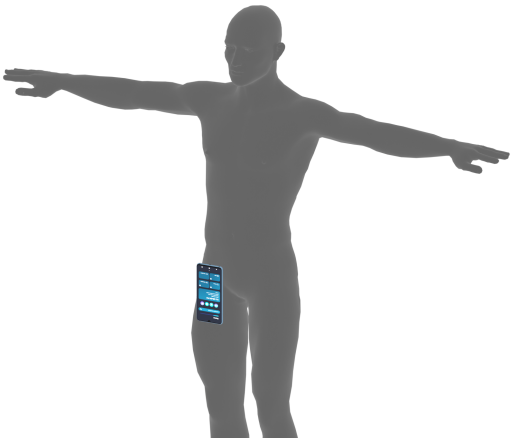

Fig. 13. Phone-position for transfer-learning.

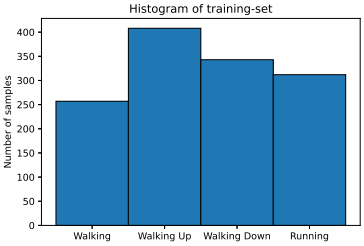

Fig. 14. Activity-histogram of the parent-model training-set.

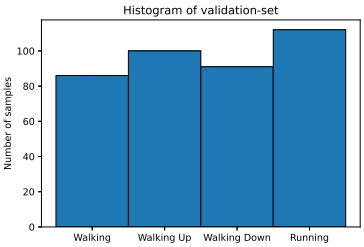

Fig. 15. Activity-histogram of the parent-model validation-set.

VIII. Validating parent- and child-model

Although the child-model is actually trained on-device, an offline-training and validation procedure was set-up during the development process to validate its classification-performance. The confusion-matrices for both, parent- and child-models for the already introduced data-sets (figures 14 to 17) are shown in figures 22 and 23.

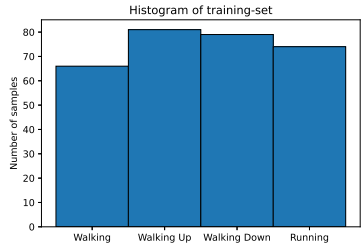

Fig. 16. Activity-histogram of the child-model training-set.

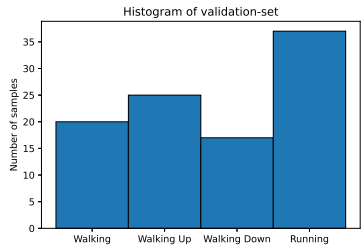

Fig. 17. Activity-histogram of the child-model validation-set.

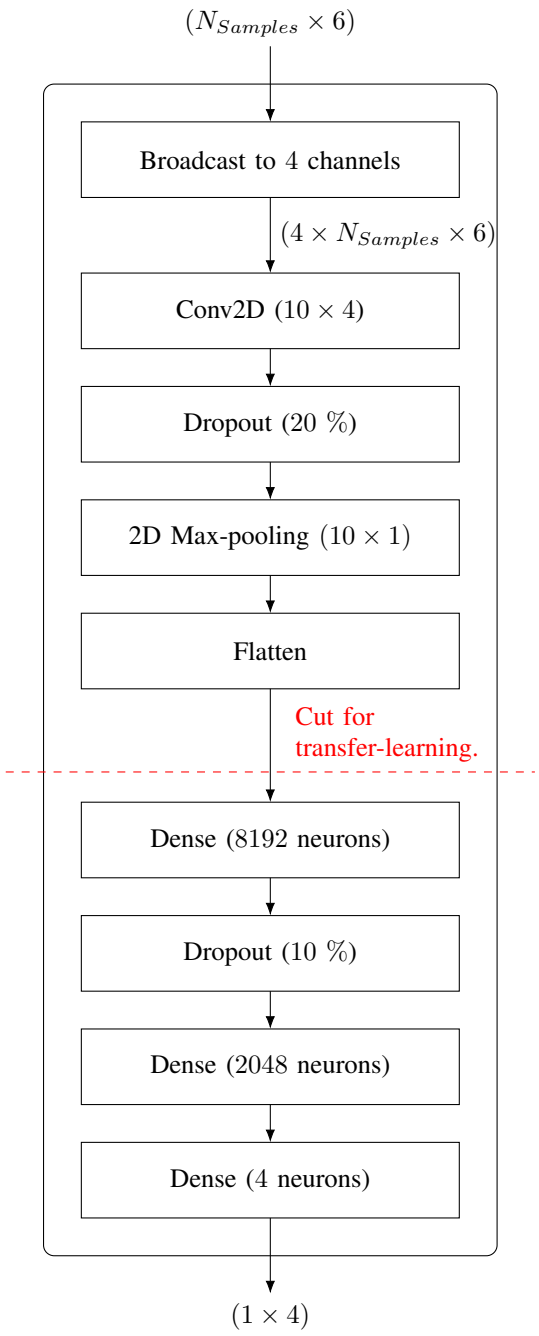

Fig. 18. The architecture of the parent-model. All layers except the last one use *ReLU*-activation. The output-layer uses *softmax*-activation. The architecture of the *Conv2D*-layer is shown in detail in figure 20.

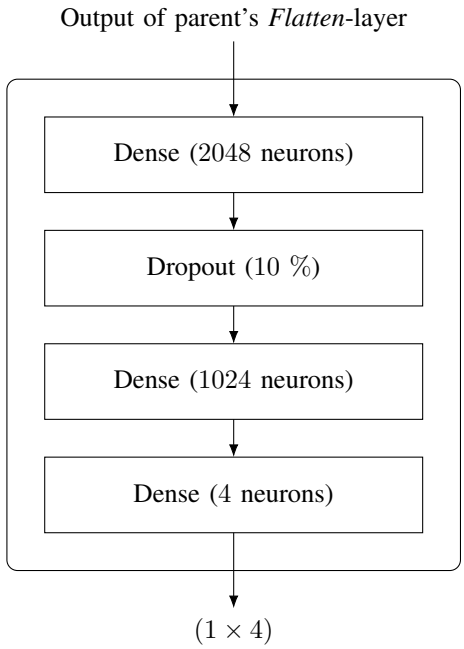

Fig. 19. The architecture of the head-model. All layers except the last one use *ReLU*-activation. The output-layer uses *softmax*-activation.

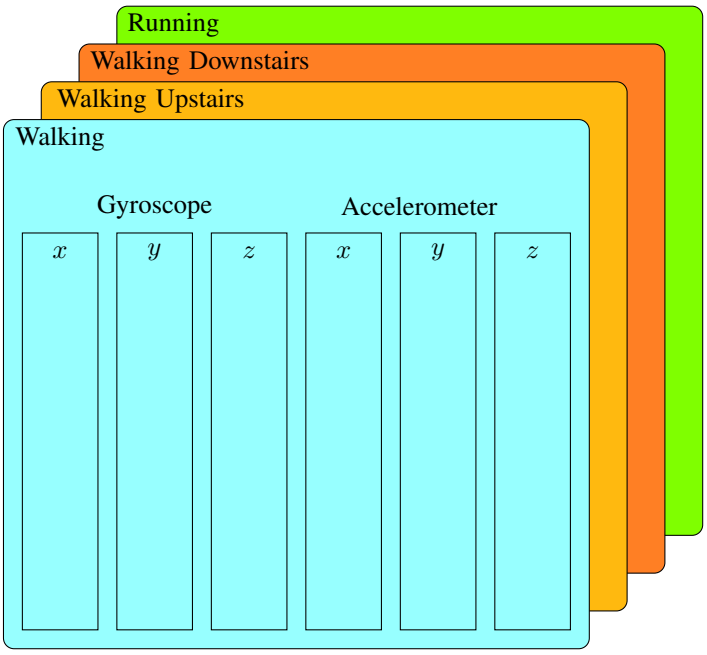

Fig. 20. Architecture of 2D convolution-layer. The sensor-data is interpreted by an $N_{Samples} \times 6$ image. As a 4-seconds timeframe at a sampling-rate of 50Hz is chosen in the pre-processing, one derives $N_{Samples} = 200$. The 2D-convolution is set up to have 4 channels, resulting in an own convolution channel for each activity.

IX. The App UI

This section gives an quick introduction on the UI of the app.
After opening the app, the user observes the UI-window shown in figure24. He can then choose between the 3 actions:

- **CAPTURE DATA (Figure 25)**:
  This button opens a new window for capturing the sensor-data and storing it to a *.csv*-file on the device. Thereby, the selected activity only affects the names of the resulting files, so one can keep the default selection when not

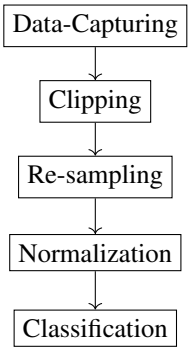

Fig. 21. Outline of transfer-learning signal-processing pipeline.

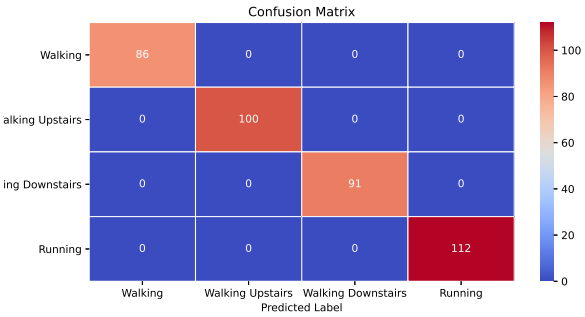

Fig. 22. Confusion-matrix of the parent-model.

caring about the actual name of the output file. To prevent overwriting previous captures, a suffix is attached to the single filenames containing the UTC-timestamp from the start of the capture in milliseconds.

- **ACTIVITY RECOGNITION KNN (Figure 26)**:
  This button opens a window for classifying an activity with the pre-trained *kNN*-classifier. The classification is executed continuously as soon as this window is opened.
- **TRANSFER LEARNING (Figure 27)**:
  This window contains all possible actions for the transfer-learning part of the app. The upper part shows the probabilities of the single classes as well as the class with the currently highest probability. Classification is not started when the window is opened, but has to be manually enabled through the *Inference*-switch.

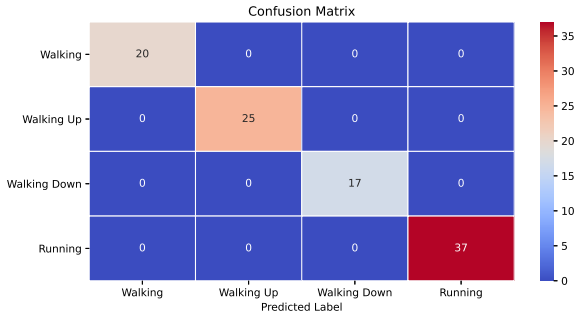

Fig. 23. Confusion-matrix of the child-model.

The lower section contains all actions and information about the transfer-learning samples themselves. Similar as in the data-capturing window, the user can select the proper activity and start the recording via the *Record*-switch.
The table below this switch shows a summary of the captured samples: How much samples were captured and when they were updated. Through the switch with labels *Overwrite* and *Add*, the user can select whether to overwrite the samples for the selected activity after the next capture, or to add new samples.
The *DEBUG* button automatically loads the validation-set shown earlier in this report. This functionality can be used for debugging and validation purposes.
The actual learning procedure is then started via the *Learn*-switch. Thereby, the current loss is additionally printed right next to the switch.
When disabling the learning, the current optimization-epoch is finished, so it is recommended to wait a few seconds until the loss-value is updated to its final value.

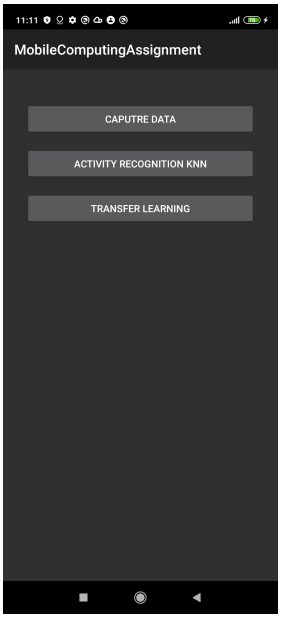

Fig. 24. The main-window, seen by the user when opting the app.

X. Issues, Possible Improvements and Conclusion

The main reason why a different set of activities was chosen for part 1 and part 2 is, that it was originally planned to use a provided data-set for transfer-learning, containing the 6 activities *Walking*, *Walking Upstairs*, *Walking Downstairs*, *Standing*, *Sitting*, *Laying* as well as the 6 activity-transitions *Stand-to-Sit*, *Sit-to-Stand*, *Sit-to-Lay*, *Lay-to-Sit*, *Stand-to-Lay* and *Lay-to-Stand*. Later, the *kNN*-classifier should also be trained and applied to these activities/activity-transitions. Unfortunately, designing and implementing a model for this 12 classes were harder then expected, especially as the originally used phone had a broken accelerometer. As the activities in part 1 are mainly characterized by their axis of

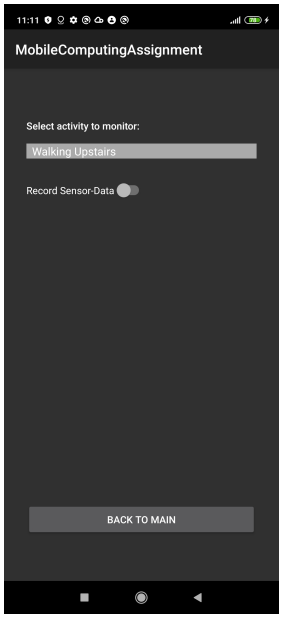

Fig. 25. The UI-window for capturing data.

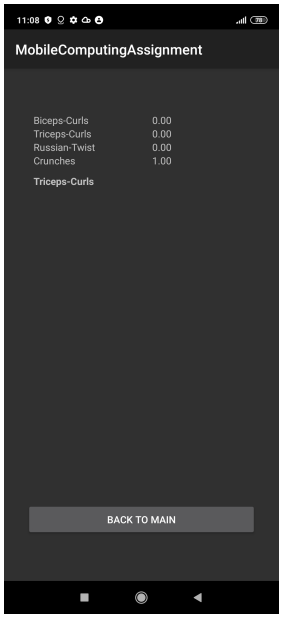

Fig. 26. The UI-window for the *kNN*-classifier.

rotation, a accurate classification was possible, independent of the accelerometer. The broken sensor then became a real problem when testing a first version of the parent-model, which achieved a quite acceptable accuracy on the computer, but had a terrible performance when utilizing it on the phone. To determine whether this issue is data-set of phone-related, an own data-set for the 4 activities then finally implemented in the app was collected. After a quite large time invested in debugging, the accelerometer could finally be recognized as the actual cause of the problem.
After a new phone was obtained, the reduced data-set was re-collected and parent- and child-models were re-designed, implemented and tested, no time was available anymore

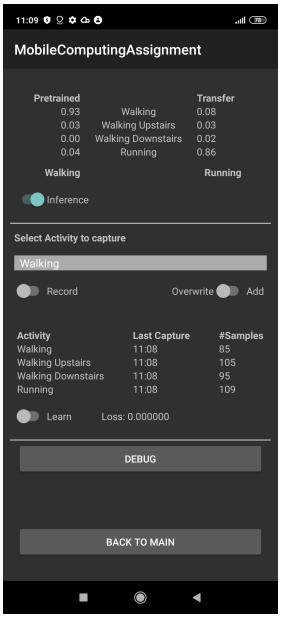

Fig. 27. The UI-window for the transfer-learning part of the assignment.

to switch from the two self-collected data-sets for the two different sets of activities, to the provided data-set with its 6 activities and 6 activity-transition. Instead, stability and performance of the app and the derived results were optimized.

One issue observed while designing and testing the current 4-activity data-set was that the performance of the transfer-learning model significantly depends on the phone itself as well as its position: When wearing 'loose' trousers, the phone starts to jump-around within the pocket in an non-deterministic way for several activities. This, of course, makes it hard for the model to determine the correct activity. This problem especially occurs when putting the phone in one of the back pockets, as these are typically looser than front-pockets. Further, size and weight of phone significantly increased during the past years. This facts further promote the jumping.

To conclude, the classifiers for both parts of this assignment work as expected. One could definitely improve the number of activities to recognize as well as utilizing *kNN*- and *transfer-learning* classifiers to the same activity-set. Further, one could try to train the *transfer-learning* classifier on a data-set captured really by different phones, as this would further generalize the approach. Further, the number of parameters for the *transfer-learning* model might be reduced to improve the training-performance on older phones.

XI. Appendix

References

[1] T. Lafer. (Jun. 27, 2021). "Mobile-computing assignment," [Online]. Available: https://github.com/tlaf0504/mobile_computing_assignment.

[2] Pavel Senchanka, *Example on-device model personalization with TensorFlow Lite*. Dec. 12, 2019. [Online]. Available: https://blog.tensorflow.org/2019/12/example-on-device-model-personalization.html (visited on 06/27/2021).

[3] (Jun. 21, 2021). "K-nearest neighbors algorithm," [Online]. Available: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm.

[4] (Jun. 22, 2021). "Norm (mathematics)," [Online]. Available: https://en.wikipedia.org/wiki/Norm_(mathematics).

[5] (Jun. 22, 2021). "Euclidean distance," [Online]. Available: https://en.wikipedia.org/wiki/Euclidean_distance.

[6] (Jun. 22, 2021), [Online]. Available: https://www.shutterstock.com/de/image-vector/man-doing-standing-dumbbell-bicep-curls-1850250391.

[7] (Jun. 22, 2021), [Online]. Available: https://www.elle.de/armuebungen-model-workout.

[8] (Jun. 22, 2021), [Online]. Available: https://www.shutterstock.com/de/image-vector/man-doing-crunches-abdominals-exercise-flat-1842272014.

[9] (Jun. 22, 2021), [Online]. Available: https://www.elle.de/bauchmuskel-training-abnehmen.

[10] (Jun. 22, 2021). "SensorEventCallback," SensorEventCallback, [Online]. Available: https://developer.android.com/reference/android/hardware/SensorEventCallback#onSensorChanged(android.hardware.SensorEvent).

[11] (Jun. 27, 2021). "Seaborn.pairplot," [Online]. Available: https://seaborn.pydata.org/generated/seaborn.pairplot.html.