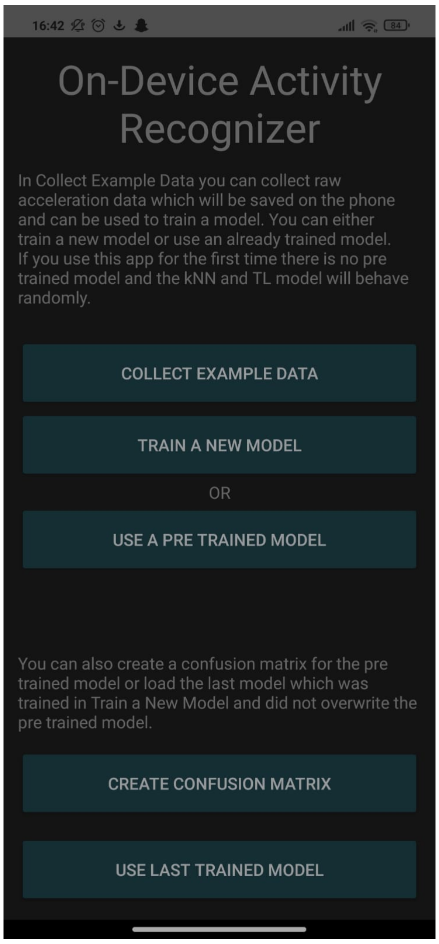
Mobile Computing Report:

On-Device Activity Recognition

Hannes Reindl



On-Device Activity Recognizer

In Collect Example Data you can collect raw acceleration data which will be saved on the phone and can be used to train a model. You can either train a new model or use an already trained model. If you use this app for the first time there is no pre trained model and the kNN and TL model will behave randomly.

You can also create a confusion matrix for the pre trained model or load the last model which was trained in Train a New Model and did not overwrite the pre trained model.

Figure 1. Title screen of the created app

Contents

I. Introduction

For the mobile computing laboratory an app should be coded and designed, which can recognize activities (e.g. walking, standing, going downstairs, going upstairs,...). Furthermore the app should be able to "learn" new activities or relearn an activity if the mobile phone is at another location (e.g. mobile in pocket, mobile on belt, mobile on upper arm, ...). All of this has to be possible using the app alone without changing the code or plugging the phone to a PC. It should also work offline, meaning there is no internet connection required. The activity recognition was done through two different approaches: k-nearest-neighbors and transfer learning.

II. Background and Theory

This section will cover some basics regarding data gathering, k-nearest-neighbors and transfer learning.

A. Data Gathering

In general the accelerometer sensors on the device are used as to get acceleration values which will be the input to both methods (kNN and transfer learning). The figure below depicts an example for what the accelerometer values could look like:

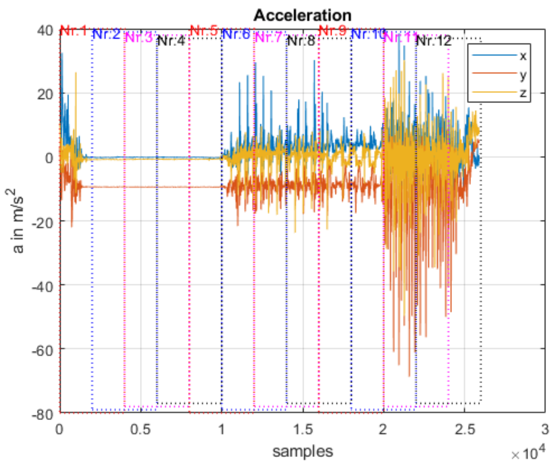


Figure 2. Example for acceleration values

Figure 2 shows three lines, which show the acceleration values for the x-,y- and z-axis over time recorded by my phone. The dotted rectangles shows how the data was segmented for further calculations - this will be discussed later in more detail. To give some insight on how activities look like for an accelerometer, the following paragraphs describe which activity was done at which time.

The first few samples in the figure show the movement of putting my phone into my right pocket. This is not an activity I want to recognize and therefore should be elided. In the final app when the user starts to collect data a timer starts to count down from 3 after which it will start recording data. This allows the user to put the mobile phone at the desired spot before the acceleration values get recorded.

After about $0.25 \cdot 10^4$ samples all three lines are more or less constant up until the $1 \cdot 10^4$ sample. While recording this, I was standing. Furthermore the values for x and z are close to zero while y hovers somewhere around -9.81 m/s^2 which is equal to the acceleration due to gravity g. This reveals two things. First: The measurement values are plausible. Second: The orientation of the phone can be calculated if the coordinate system of the phone is know. This also shows that the same activity can look different, depending on the orientation of the phone. Additionally this also means that an activity can only be learned for one orientation if the raw accelerometer values are used. It should also be noted that activities can look differently if the phone is mounted somewhere else (e.g. from right pocket to right forearm). This is reason why it is important that new activities can be learned **on the device**. During testing with kNN, the app also performed worse by just changing the location and/or orientation of the phone a little (e.g. when using in a pocket, using different pants than it was trained in).

Starting from the $1 \cdot 10^4$ sample up until $2 \cdot 10^4$ the recorded acceleration values show the activity for walking. The values change frequently while some overarching pattern can be seen. Walking is a periodic movement, which can be seen well in the blue lined spikes. This periodicity brings up another important point. Data has to be segmented, because memory is finite and activities done one minute ago should not affect the current result for the activity, but it should also be able to catch at least one fully cycle of walking. A frame period of 2 seconds seems reasonable here. But this raises another problem: It takes 2 seconds to evaluate which activity is done which feels unresponsive. Therefore a sliding frame was used. Figure 2 shows a 50% overlap of frames, which means it evaluates the activity after 1 second. Using sliding frames is reasonable since the world is continuous and changing activities wont be instantaneous. But this also means that for each evaluation some past values will be used, which can cause some delay until the just started activity will be recognized. This will be even more visible for higher overlap percentages and is visible in the final app, which uses an overlap of 95%. Finally we take a look at the samples after $2 \cdot 10^4$ until the end. The activity done there was running. Most of the points said for the walking activity can also be made here. Noteworthy are the bigger peaks due to the rapid movement when running and higher frequency. Just like when walking we repeat the same movement after we step on the same foot again. With all this knowledge the k-nearest-neighbor algorithm can be tackled.

B. k-nearest-neighbors

The k-nearest-neighbor (kNN) algorithm can be used to predict something using new data by comparing it to historical examples. This is done by calculating some features of the given data. Figure 3 below depicts the feature mean value of x- and z-acceleration. Beforehand some data was collected and labeled for some different activities. I don't specify the activities here since the goal of this app is to easily learn new activities and not limit to some specific one.

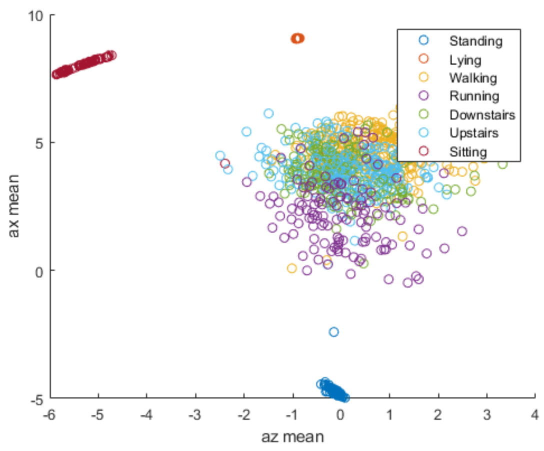


Figure 3. k-nearest-neighbors algorithm

The features get calculated for each frame and each point will be stored. Figure 3 shows how this can look like when plotting the mean features. To predict a new activity the same features get calculated for the new data. If the recorded activity was sitting the features would result in a vector pointing to the top left corner, where all the labeled points for sitting are. The kNN algorithm calculates the euclidean distance to each other point and does a majority vote of the k nearest neighbors. The variable k denotes how many nearest neighbors should be considered for the majority vote. The variable k should also be an odd number since an even number allows for a draw. For the first implementation in python an online tutorial was used.[1]

Which features to use is also quite important. Good features allow for a robust prediction. I found a paper on the website ieeexplore.ieee using kNN and accelerometer values to recognize activities[2] and took inspiration for my features. The features I calculate are:

Minimum:

$$a_{x,min} = \min_{n}(a_x[n]) \tag{1}$$

where $a_x[n]$ is one frame of the acceleration values for the x-axis.
Maximum:

$$a_{x,max} = \max_{n}(a_x[n]) \tag{2}$$

Mean:

$$\bar{a}_x = \frac{1}{N}\sum_{n=0}^{N-1} a_x[n] \tag{3}$$

where N is the amount of samples per frame.

[1]https://machinelearningmastery.com/tutorial-to-implement-k-nearest-neighbors-in-python-from-scratch/

[2]https://ieeexplore.ieee.org/document/9062703

Variance:

$$\sigma_{a_x}^2 = \frac{1}{N-1}\sum_{n=0}^{N-1}(a_x[n] - \bar{a}_x)^2 \tag{4}$$

Signal Magnitude Area:

$$SMA_{a_x} = \frac{1}{N}\sum_{n=0}^{N-1}\left|a_x[n]\right| \tag{5}$$

Energy:

$$E_{a_x} = \frac{1}{N}\sum_{n=0}^{N} a_x^2[n] \tag{6}$$

This was repeated for the y- and z-axis resulting in 18 features. If a new activity needs to be added the stored features just need to be updated with the new extracted features. If a new position of the mobile phone needs to be learned, all old stored features need to be deleted before adding the new ones. Performance metrics using this kNN algorithm can be found in section IV.

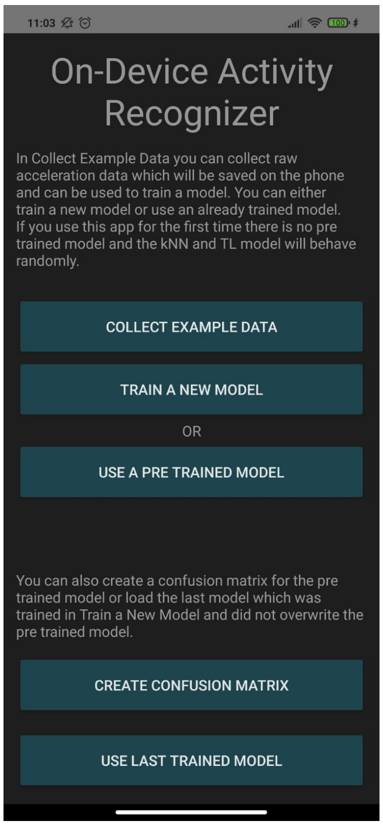
C. Transfer Learning

Creating and training a full neural network can take a lot of time especially with limited computing power like on a mobile device. That is why transfer learning was used which only has to train a fraction of a neural network. The basic premise of transfer learning is to use a base neural network which already learned some overarching patterns regarding the problem and we only retrain the last few layers, which can be modified easily. To put this more into perspective lets take a look at recognizing hand written digits using a neural network. The input layer has the same amount as there are pixels and it takes their corresponding gray values as input. It could be said that the first hidden layer learns to identify straight lines like on a 7-segment display. The second hidden layer learns to identify circles. The last layer combines these found patters to recognize the number. We could now chop off the last layer and change it in such a way, that it can distinguish different shapes(e.g. rectangle, triangle, circle). Since the basic neural network already knows how a straight line or a circle looks like, it is sufficient to retrain only the last layer. Note: This is not really how a NN network behaves but it can be looked it this way, to give some understanding.
For my app I used the provided google colab code resource. First a fully blown NN was trained using the HAPT data. The sampling rate of the HAPT data set is 50 Hz and contains the 3-axial linear acceleration values, where each frame has 200 samples. One frame therefore spans 2 seconds. The rate of gathering acceleration values using the mobile device should also be 50 Hz using the 3 axial linear acceleration values where each frame has 200 samples. It should also be noted, that the HAPT data set recorded the values while the mobile phone was put on the persons belt. Using transfer learning to train the NN the tensorflow package will be used, because it also supports transfer learning. When training the base model only one parameter was changed, namely the `STEP_DISTANCE` from 100 to

10. The `STEP_DISTANCE` influences the overlap between two frames, meaning that after 10 new samples it will be considered as a new frame. This increased the validation accuracy from 72% to about 80 %. The same parameters will be used on the app when gathering data to keep things consistent.

III. Implementation

The figure below shows the start screen after opening the app.



11:03

On-Device Activity Recognizer

In Collect Example Data you can collect raw acceleration data which will be saved on the phone and can be used to train a model. You can either train a new model or use an already trained model. If you use this app for the first time there is no pre trained model and the kNN and TL model will behave randomly.

COLLECT EXAMPLE DATA

TRAIN A NEW MODEL

OR

USE A PRE TRAINED MODEL

You can also create a confusion matrix for the pre trained model or load the last model which was trained in Train a New Model and did not overwrite the pre trained model.

CREATE CONFUSION MATRIX

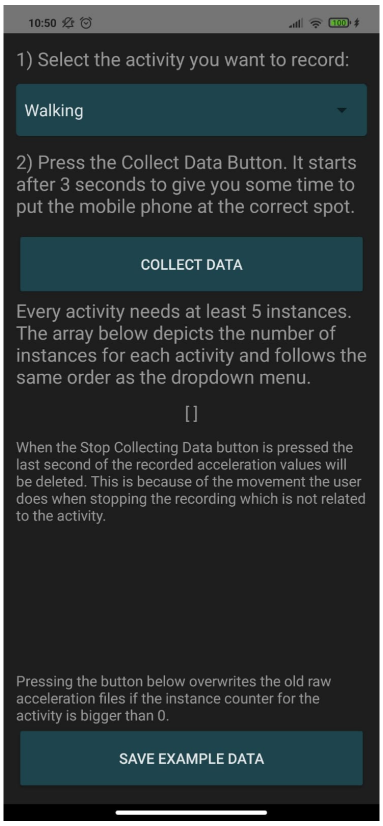
USE LAST TRAINED MODEL

Figure 4. Main Activity

The user can do several things here. The first one is to collect raw acceleration measurement data which can then be used to train a new model **OR** the user loads a pre trained model. A pre trained model only exists if the user saved a model trained in ”train a new model”. The ”collect example data” button leads to the data gathering activity which will be further discussed in section III-A. The activity for ”train a new model” will be discussed in section III-B. The ”use a pre trained model” button loads the pre trained kNN as well as the transfer learning model and starts the prediction activity. In the bottom half of the screen are two additional buttons. The ”create confusion matrix” starts again the data gathering activity, but this time the data will be used to create a confusion matrix (see section III-D). The button ”use last trained model” loads the last trained model in ”train a new model” which did not overwrite the pre trained model and after that starts the prediction activity (see section III-C).

A. *Data Gathering*

The three figures below show the data gathering activity.



10:50

1) Select the activity you want to record:

Walking

2) Press the Collect Data Button. It starts after 3 seconds to give you some time to put the mobile phone at the correct spot.

COLLECT DATA

Every activity needs at least 5 instances. The array below depicts the number of instances for each activity and follows the same order as the dropdown menu.

[]

When the Stop Collecting Data button is pressed the last second of the recorded acceleration values will be deleted. This is because of the movement the user does when stopping the recording which is not related to the activity.

Pressing the button below overwrites the old raw acceleration files if the instance counter for the activity is bigger than 0.

SAVE EXAMPLE DATA

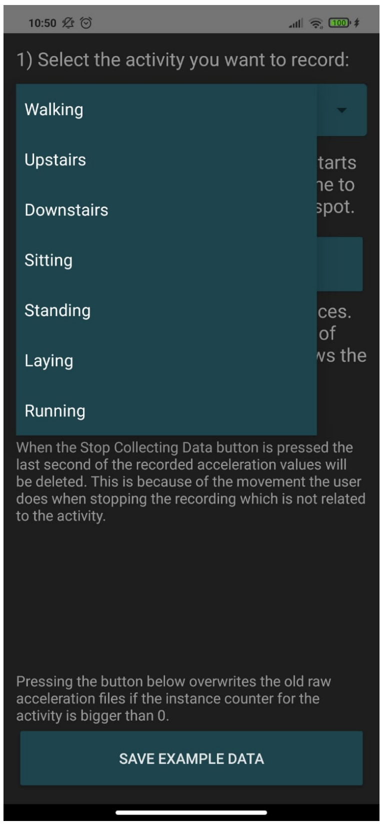
Figure 5. Data Gathering activity

Figure 5 depicts the screen after starting the activity. This activity collects raw acceleration data and labels them. At the top of the screen the drop down menu allows the user to change the current activity which can be seen in figure 6.

When the user presses the ”collect data” button a countdown of 3 seconds starts, which allows the user to put the mobile at the desired spot. During testing I collected data while the phone was in my pocket and it happened quite often that my thigh pressed on the collect data button again or changed the activity in the drop down menu or pressed the back button. All of the buttons on screen are deactivated (even going back) if data gets collected and while changing the drop item is still possible, it has no effect until the data collection is stopped.

To stop the data collection the volume up or volume down button needs to be pressed, which also gets prompted via `Toast.makeText()` if the user presses the ”stop collecting data” button. If the user presses the home or the power button data collection will also be stopped before going to the homescreen/locking the phone.

The array (instance counter) in figure 7 displays how many instances of each activities the user already collected. The

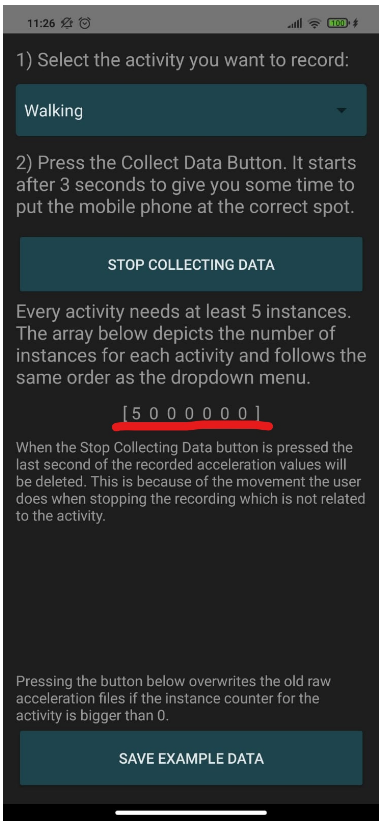


1) Select the activity you want to record:

When the Stop Collecting Data button is pressed the last second of the recorded acceleration values will be deleted. This is because of the movement the user does when stopping the recording which is not related to the activity.

Pressing the button below overwrites the old raw acceleration files if the instance counter for the activity is bigger than 0.

Figure 6. Drop down menu



1) Select the activity you want to record:

2) Press the Collect Data Button. It starts after 3 seconds to give you some time to put the mobile phone at the correct spot.

Every activity needs at least 5 instances. The array below depicts the number of instances for each activity and follows the same order as the dropdown menu.

[5 0 0 0 0 0 0]

When the Stop Collecting Data button is pressed the last second of the recorded acceleration values will be deleted. This is because of the movement the user does when stopping the recording which is not related to the activity.

Pressing the button below overwrites the old raw acceleration files if the instance counter for the activity is bigger than 0.

Figure 7. Instance counter

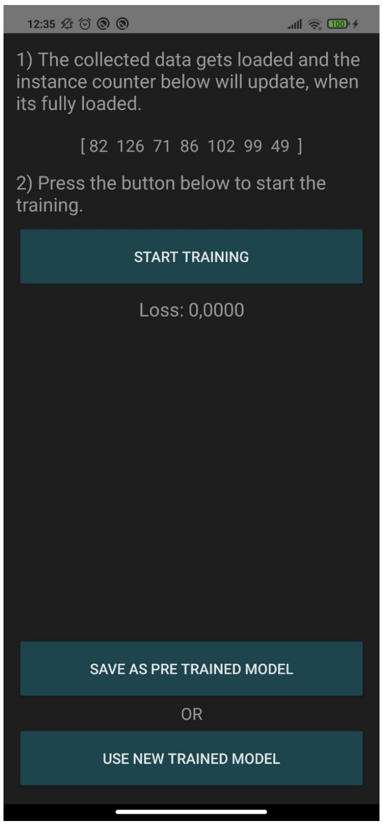
numbers in the instance counter follow the same order as the drop down menu.

When the user stops the data collection, the last second of the recorded activity will be deleted, which can also be seen by a decrease of the instance counter. When trying to stop the data collection it happens that movement which is not related to the activity gets recorded and therefore should be deleted. The trained model can only be as good as the recorded data which is why a lot of thoughts went into this.

Finally the data can be saved onto the internal storage by pressing the button ”save example data”. Each activity has its designated file and the file only gets overwritten if the instance for the activity is higher than 0 (meaning that some data was collected). This way the entire data set does not have to be collected at once. It can be split throughout a day or more. Again to counter act accidental inputs a dialog box prompts when the ”save example data” is pressed, which asks the user if he is sure. If the user pressed the back button if some data was collected, the same dialog box appears.

B. *Training a new Model*

The figure below shows the screen after pressing ”train a new model”.



1) The collected data gets loaded and the instance counter below will update, when its fully loaded.

[82 126 71 86 102 99 49]

2) Press the button below to start the training.

Loss: 0,0000

Figure 8. Train a new model activity

When the activity gets started the raw acceleration data collected in the data gathering activity will be loaded.

After loading the measurements the instance counter will be updated and displays the amount of frames per activity. Following that the ”start training” button can be pressed and the kNN and transfer learning model will be trained, which are discussed further in the sections below. The loss text view shows the current loss of the transfer learning model. After training the model it can be saved as the pre trained model by pressing the corresponding button and the app will return to the main activity (i.e. start screen of the app). If the user taps the ”use new trained model” button the prediction activity gets started using the trained model which will be discussed further in section III-C.

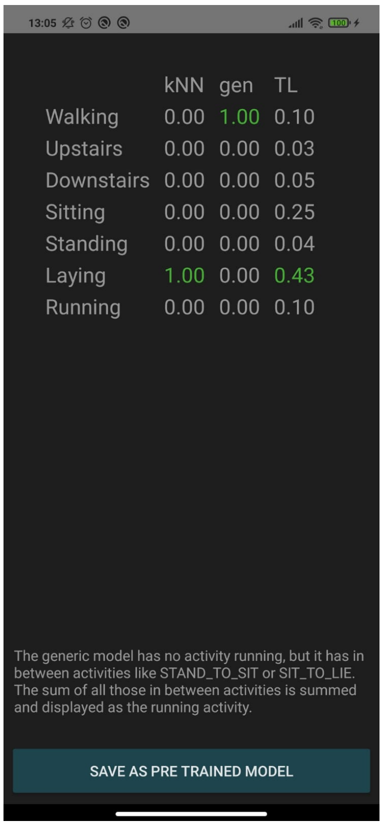
1) k-nearest-neighbors: Training the kNN model is straightforward since only the features for each frame need to be calculated and stored with their corresponding label. For this a kNN class was written in Java which takes care of calculating features, saving the model, loading the model and predicting an activity using acceleration data as input.

2) Generic Model: The generic model is a fully fletched neural network which is already trained using the HAPT data set. To get some predictions the interface provided in the Android Studio Project *Hand Written Digits* and the class `TensforFlowLiteClassifier` was used. The method `recognize` in the `TensforFlowLiteClassifier` class was changed a little to fit my task. During my testing I could not get this working properly. It did not predict things well even if I use the exact same parameters for collecting data as the HAPT data set used, normalized the measurements, changing some hyper parameters.

3) Transfer Learning: The code to train the transfer learning model was taken from the provided Android Studio project *OnDeviceActivityRecognition*. Only the variable `listClasses` in the class `TransferLearningModelWrapper` was changed to fit my activities as well as the new head created in the second google collab resource.

C. Prediction Activity

Figure 9 shows the screen of the prediction activity.
The predicition activity constantly records acceleration data and uses them to predict the activity. The rows show the activity names while the columns are the different methods. The number 1 corresponds to a confidence of 100% and the most probable activity per method is highlighted in green. This screen gets programmatically created, meaning that depending on the number of activities rows get added or deleted as well as the predictions values for kNN and TL. The order of the prediction values for the generic model is fixed since it can not learn new activities. Further the activity Running is not in the generic model but in between movements like going from standing to sitting are in there. The sum of the predictions for the in between activities are displayed in the cell for Running.



13:05

	kNN	gen	TL
Walking	0.00	1.00	0.10
Upstairs	0.00	0.00	0.03
Downstairs	0.00	0.00	0.05
Sitting	0.00	0.00	0.25
Standing	0.00	0.00	0.04
Laying	1.00	0.00	0.43
Running	0.00	0.00	0.10

The generic model has no activity running, but it has in between activities like STAND_TO_SIT or SIT_TO_LIE. The sum of all those in between activities is summed and displayed as the running activity.

Figure 9. Prediction activity

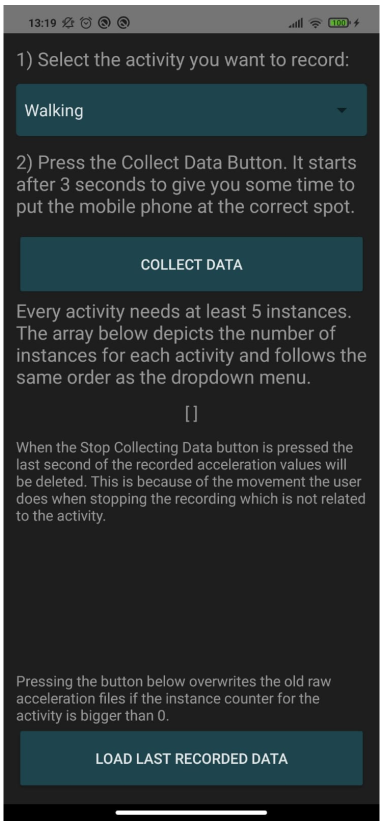
The button ”save as pre trained model” in the bottom allows the user to save a model which started this activity as a one time use only as the pre trained model.

D. Create Confusion Matrix

The three figures below show the important screens to create a confusion matrix.
Figure 10 shows the activity to collect data for the confusion matrix. This is basically the same activity which was used to get raw acceleration measurements to train the model. There is only one slight modification, seen in the change of the button text at the bottom. If no new data is collected, this button starts the activity to calculate the confusion matrix using the last recorded values, which were intended for the confusion matrix.

Figure 11 shows the activity to calculate the confusion matrix. At the top again is the instance counter to display how many instances there are for each activity. Pressing the ”calculate confusion matrix” button uses the raw acceleration data collected before to get the most likely activity from each model and save them in form of a confusion matrix. The button ”load last confusion matrix” simply loads what was the last calculated result for the confusion matrix.

Figure 12 shows the confusion matrix. The buttons below allow the change for which model the confusion matrix should be displayed.


13:19

1) Select the activity you want to record:

Walking

2) Press the Collect Data Button. It starts after 3 seconds to give you some time to put the mobile phone at the correct spot.

COLLECT DATA

Every activity needs at least 5 instances. The array below depicts the number of instances for each activity and follows the same order as the dropdown menu.

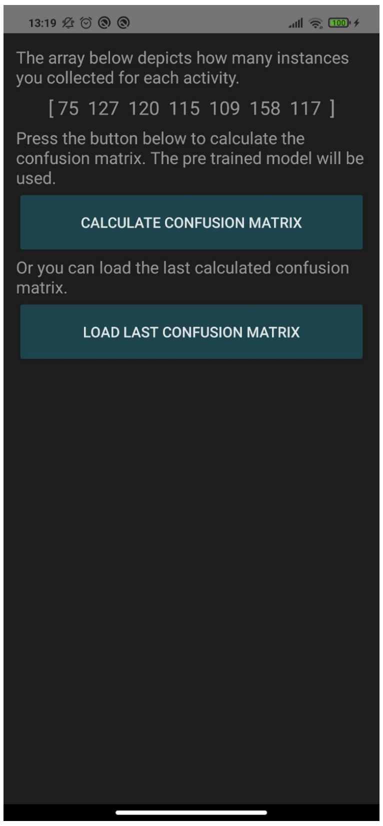
[]

When the Stop Collecting Data button is pressed the last second of the recorded acceleration values will be deleted. This is because of the movement the user does when stopping the recording which is not related to the activity.

Pressing the button below overwrites the old raw acceleration files if the instance counter for the activity is bigger than 0.

LOAD LAST RECORDED DATA

Figure 10. Collect Data activity


13:19

The array below depicts how many instances you collected for each activity.

[75 127 120 115 109 158 117]

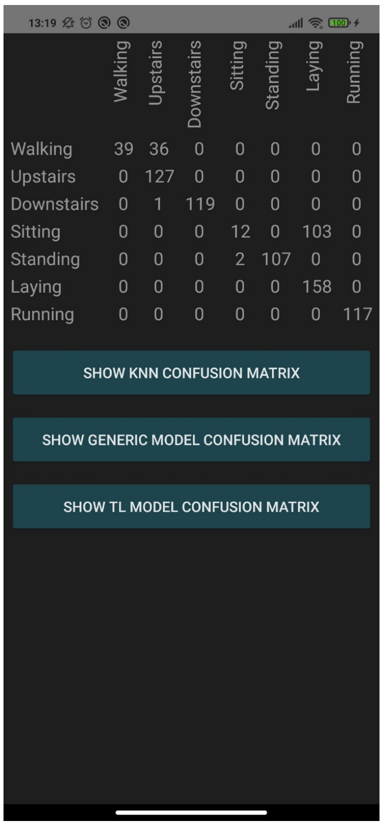
Press the button below to calculate the confusion matrix. The pre trained model will be used.

CALCULATE CONFUSION MATRIX

Or you can load the last calculated confusion matrix.

LOAD LAST CONFUSION MATRIX

Figure 11. Calculate Confusion Matrix activity


13:19

	Walking	Upstairs	Downstairs	Sitting	Standing	Laying	Running
Walking	39	36	0	0	0	0	0
Upstairs	0	127	0	0	0	0	0
Downstairs	0	1	119	0	0	0	0
Sitting	0	0	0	12	0	103	0
Standing	0	0	0	2	107	0	0
Laying	0	0	0	0	0	158	0
Running	0	0	0	0	0	0	117

SHOW KNN CONFUSION MATRIX

SHOW GENERIC MODEL CONFUSION MATRIX

SHOW TL MODEL CONFUSION MATRIX

Figure 12. Show Confusion Matrix activity

IV. Performance

In this section the performance of the app will be discussed for each model at a time.

A. *k-nearest-neighbors*

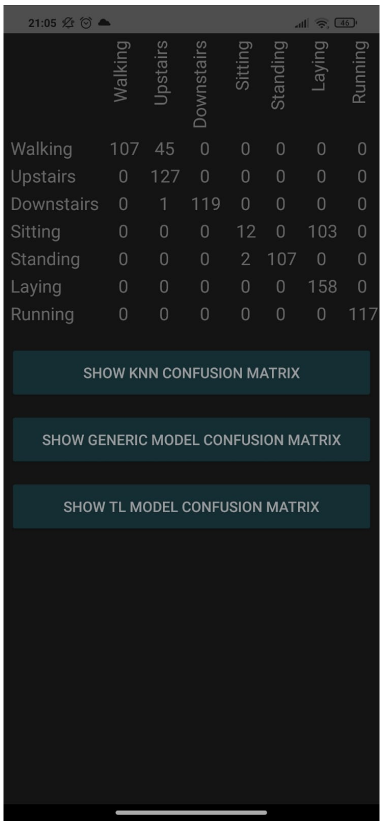
The kNN model performed by far the best of all three models.
Besides the activities walking and sitting every activity was recognized well. The activity sitting can be easily explained since most of the times it missclassifies as laying. Regarding the acceleration measurements sitting and laying is very similar and the output solely depends on a minor difference in the position of the mobile phone. The walking activity often got missclassified as going upstairs which also seems reasonable. I used rather small stairs when training the model for going upstairs so there is not a lot of difference between walking and going upstairs.

B. *Generic Model*

The generic model does not work at all for me.
As already discussed I tried a lot to get this thing going but I did not find a solution.

C. *Transfer Learning*

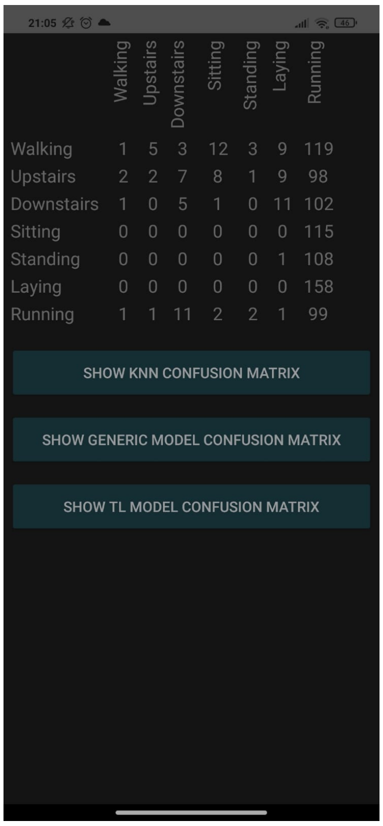
The transfer learning model works better than the generic one, but still not good.
I integrated the model the same way as it was in the given Android Studio Project *OnDeviceActivityRecognition* and



	Walking	Upstairs	Downstairs	Sitting	Standing	Laying	Running
Walking	107	45	0	0	0	0	0
Upstairs	0	127	0	0	0	0	0
Downstairs	0	1	119	0	0	0	0
Sitting	0	0	0	12	0	103	0
Standing	0	0	0	2	107	0	0
Laying	0	0	0	0	0	158	0
Running	0	0	0	0	0	0	117

21:05
SHOW KNN CONFUSION MATRIX
SHOW GENERIC MODEL CONFUSION MATRIX
SHOW TL MODEL CONFUSION MATRIX

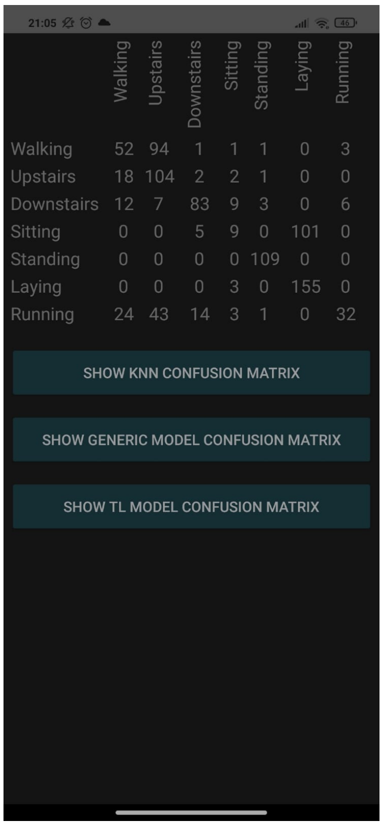
Figure 13. Confusion matrix for kNN



	Walking	Upstairs	Downstairs	Sitting	Standing	Laying	Running
Walking	1	5	3	12	3	9	119
Upstairs	2	2	7	8	1	9	98
Downstairs	1	0	5	1	0	11	102
Sitting	0	0	0	0	0	0	115
Standing	0	0	0	0	0	1	108
Laying	0	0	0	0	0	0	158
Running	1	1	11	2	2	1	99

21:05
SHOW KNN CONFUSION MATRIX
SHOW GENERIC MODEL CONFUSION MATRIX
SHOW TL MODEL CONFUSION MATRIX

Figure 14. Confusion matrix for the generic model



	Walking	Upstairs	Downstairs	Sitting	Standing	Laying	Running
Walking	52	94	1	1	1	0	3
Upstairs	18	104	2	2	1	0	0
Downstairs	12	7	83	9	3	0	6
Sitting	0	0	5	9	0	101	0
Standing	0	0	0	0	109	0	0
Laying	0	0	0	3	0	155	0
Running	24	43	14	3	1	0	32

21:05
SHOW KNN CONFUSION MATRIX
SHOW GENERIC MODEL CONFUSION MATRIX
SHOW TL MODEL CONFUSION MATRIX

Figure 15. Confusion matrix for the transfer learning model

there appear no errors when executing. The loss got as low as 0.05 during training while the training set had about 100 frames per activity. The walking and going upstairs as well as the laying/sitting problem was already discussed in the kNN model. But it also seems that it can not distinguish running from walking/upstairs/downstairs activities well. One solution may be to increase the time a frame spans. Currently a frame spans 2 seconds, which may not be enough. It could also be that the generic model causes some problem.