COURSEWORK 3

PRINCIPLES AND DESIGN OF IOT SYSTEMS

School of Informatics, University of Edinburgh

Ioana Mihailescu - s1822958
Maria Anelise Ionescu - s1817455
Wassim Jabrane - s1870697

2021-22

Abstract

Human activity recognition (HAR) has been salient in the recent nascence of personalized healthcare technologies by way of continuous monitoring of regular physical activities. Our system consists of an Android Application that collects sensory data on human motions using accelerometer and gyroscope data from an IoT device called Respeck and feeds it to a Convolutional Neural Network model. Furthermore, we give a complete overview of the state-of-the-art techniques and trends in different areas of HAR, reviewing several research papers and articles. We present a comprehensive description of the architecture of our system, the main components, starting with the hardware, detailing the wireless communication and the in-place protocols, and further describing the implementation specifics of the mobile app and the CNN model, including performance reasoning, benchmarks and testing various quantitative methods. To conclude, we report and discuss our findings, including a thorough analysis of the implementation of the CNN model and we consider future improvements of the system.

Contents

1 Introduction

1.1 Project aims

The purpose of this coursework is to use all the Internet of Things and Human Activity Recognition knowledge acquired through the PDIoT course in order to create an end-to-end system that performs on-device, real-time HAR. The system is designed for users of Respeck, an IoT device described in more detail in Section 4.2, Hardware. The real-time HAR refers to a classification of a subset of activities (Sitting/Standing, Walking, Running, Lying Down, and Falling) performed in real-time by an active user. By pairing the sensor to an Android App, called PDIoT App for the purpose of this course, the user can access a basic interface that allows them to register/login, see their current activity as well as view their past data in an intuitive manner. The aim of these features is to encourage the user to maintain a more active lifestyle. To this end the app also displays a notification to the user, reminding them to engage in some physical activity if they have been sitting for too long.

The coursework also focuses on making use of a common dataset, collected as part of the course, for training and testing machine learning algorithms developed for classifying physical activities. These ML techniques were applied to the IMU data displaying the results in the app.

From a learning perspective, through this coursework, we engaged in the different stages in the design and implementation of a complex system, from its specification to the demonstration of a working prototype and evaluation of its performance. This was an opportunity to be exposed to aspects of sensor data analytics using machine learning techniques, mobile application development, user interface design, and system integration and testing.

1.2 Brief description

The current, best CNN model classifies the following subset of activities:

- Falling
- Lying
- Running
- Sitting/Standing
- Walking

Some activities such as Sitting/Standing, Lying, and Falling, performed with different variations were grouped together in one category to provide us with more examples for the model to train on and to reduce complexity.

1.3 Summary of results

The test accuracy achieved on our best CNN model is 96% accuracy. On the analysis side, overall the experiments have shown that training a model on bigger window sizes improved its generalisation performance. We've also noted that the lower performance reported once we evaluate our model with leave-one-out cross-validation could be due to the orientation of how the Respeck is held, according to the example shown in Section 4 where the validation accuracy was low. We also talk more in depth about the metrics performance on our test set, discussing the potential reasoning behind the misclassifications and the measures reported for precision, recall, and f1-score.

1.4 Overview

The rest of this coursework is organized as follows: Section 2 contains the literature survey, a review of the state-of-the-art for human activity recognition (HAR) algorithms, Section 3 presents in detail the methodology of this project, is further divided into a system architecture (3.1), hardware (3.2), wireless communication (3.3), the HAR algorithm (3.4), the mobile application developed (3.5) as well a subsection highlighting the software organisation (3.6). Section 4 presents the results obtained through a critical analysis of the implementation using quantitative methods. Finally, section 5 represents the conclusion of this coursework, exhibiting our reflection on the project together with further extensions and improvements.

2 Literature survey

Human activity recognition systems are developed as part of a framework to enable continuous monitoring of human behaviours in the area of ambient assisted living, sports injury detection, elderly care, rehabilitation, and entertainment and surveillance in smart home environments [10].

The first works on human activity recognition (HAR) date back to the late '90s [5]. However, new methods are being developed to address the challenges HAR is facing, which include: the selection of the attributes to be measured, the construction of a portable, unobtrusive, and inexpensive data acquisition system, the design of feature extraction and inference methods, the collection of data under realistic conditions, the flexibility to support new users without the need of re-training the system, and the implementation in mobile devices meeting energy and processing requirements.

The recognition of human activities has been approached in two different ways, namely using external sensors, that is devices that are fixed in predetermined points of interest and for which the inference of activities depends on the voluntary interaction of the users with the sensors, and wearable sensors which are attached to the user.

Figure 1 [8] summarizes the state-of-the-art methods used for HAR. Traditionally, the activity model needs to be fed with a large set of labeled data using a supervised machine learning algorithm and thus it can learn the hidden patterns during training. However, labeling huge sets of data is inconvenient and often unfeasible. A solution that tackles this challenge is active learning, where the model actively queries the user for labels. Also, transfer learning, a method which allows transferring the knowledge learned from one model to another, can be used to overcome the high computational costs and training times incurred by processing large datasets [4].

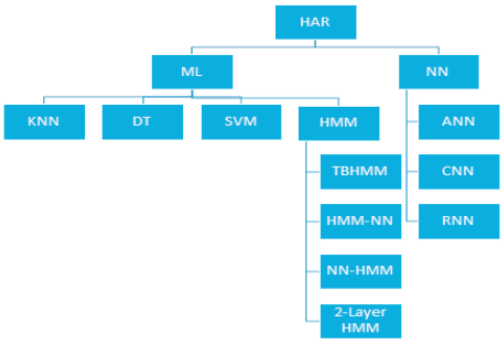

Figure 1: HAR methods. ML: Machine Learning, NN: Neural Networks, KNN: K-Nearest Neighbour, DT: decision Tree, SVM: Support Vector Machine, HMM: Hidden Markov Model, TBHMM: Threshold Based HMM, HMM-NN: HMM-Neural Networks, NN-HMM: Neural Networks-HMM, ANN: Artificial Neural Network, CNN: Convolutional NN, RNN: Recurrent NN

2.1 Machine Learning

A proper classification system has to be capable of learning and tolerating noise. In the recent past, researchers have investigated the human activity recognition problem with a wide variety of ML techniques. These include algorithms such as K-Nearest Neighbours (KNNs), Decision Trees, Support Vector Machines (SVMs) and Hidden Markov Models (HMMs). Traditional techniques such as SVMs and HMMs have been extensively used in the HAR systems,;however, there is a recent shift in the use of machine learning and data mining techniques since the popularity of deep learning.

2.2 Neural Network Techniques

Along with Machine Learning methods, parametric methods such as Artificial Neural Networks (ANNs), Convolutional Neural Network (CNNs), and Naïve Bayesians are also used in the task of HAR. Deep learning methods learn the features directly from the data hierarchically which eliminates the problem of hand-crafted feature approximations. CNNs have been successful in learning complex activities due to their properties of local dependencies and scale invariance [14]. CNNs have found extensive applications in image classification, sentence modelling, speech recognition as well as, more recently, in mobile and wearable sensors based HAR [3].

Long Short-Term Memory (LSTM) networks are a type of recurrent neural network capable of learning order dependence in sequence prediction problems. Ordóñez et al (2016) [11] proposed an activity recognition classifier, which combined deep CNN and LSTM to classify 27 hand gestures and five movements. A CNN-LSTM model was proposed which could extract activity features automatically and classify them with few parameters. Additionally, it was evaluated on three of the most widely used public datasets providing results that show high accuracy and also good generalization ability, and fast convergence speed.

Recurrent neural networks (RNNs) were developed in order to model sequential data such as time series or raw sensor data. They incorporate a temporal layer to capture sequential information and then learn complex changes using the hidden unit of the recurrent cell. LSTM has been designed to overcome these issues that arise in RNN architectures such as vanishing gradients [13]. Fok et al (2018) [6] used RNN with LSTM to analyze dynamic video motion of sports actions and classify different types of actions and their performance with an accuracy of 92.2%. The model could classify different types of human motion with a small number of video frames for the memory and computational efficiency.

2.3 Transfer learning

Transfer learning can be defined as the ability to extend what has been learned in one context to new contexts [2]. Transfer learning has achieved successful applications in many fields, such as computer vision and NLP. Transfer learning approaches in HAR can be categorised along four dimensions: sensor modalities, differences between source and target environments, available types of data labeling, and representation of the knowledge. Morales et al (2016) were the first to explore deep transfer learning within HAR [9]. They used three layers of CNN with one layer of LSTM, fine-tuned the model by freezing different layers to study the transferring between users, locations, and devices. They found out that transfer features of the first layer are possible without affecting performance for the same application domain.

Subasi et al (2020) [16] produced a comparison of several classification techniques, evaluating the performances of the classifiers in terms of total classification accuracy, F-measure, ROC area, and Kappa. Seven activities have been selected for the dataset (elliptical bike, cycling, jogging, jump-up, rowing, running, and walking) and they were measured by a total of nine on-body sensors measuring 3D acceleration, the 3D rate of turn or gyro, 3D magnetic field orientation and 4D quaternions. They used a reduced version of the open-access activity recognition dataset REALDISP (REAListic sensor DISPlacement), breaking it into training and test sets, and then k-fold cross-validation has been applied in order to investigate the accuracy of each model. Figure 2 presents the results.

	Accuracy (%)	*F*-Measure	ROC	Kappa
SVM	99.43	0.994	0.999	0.9933
k-NN	98.92	0.989	1	0.9874
ANN	99.33	0.993	0.993	0.9922
Naïve Bayes	93.97	0.94	0.994	0.9296
RF	99.19	0.992	0.995	0.9906
CART	98.71	0.987	0.995	0.985
C4.5	98.83	0.988	0.996	0.9863
REPTree	98.25	0.983	0.997	0.9796
LADTree	84.05	0.837	0.976	0.8139

Figure 2: Classification results for HAR using different algorithms from Subasi et al.

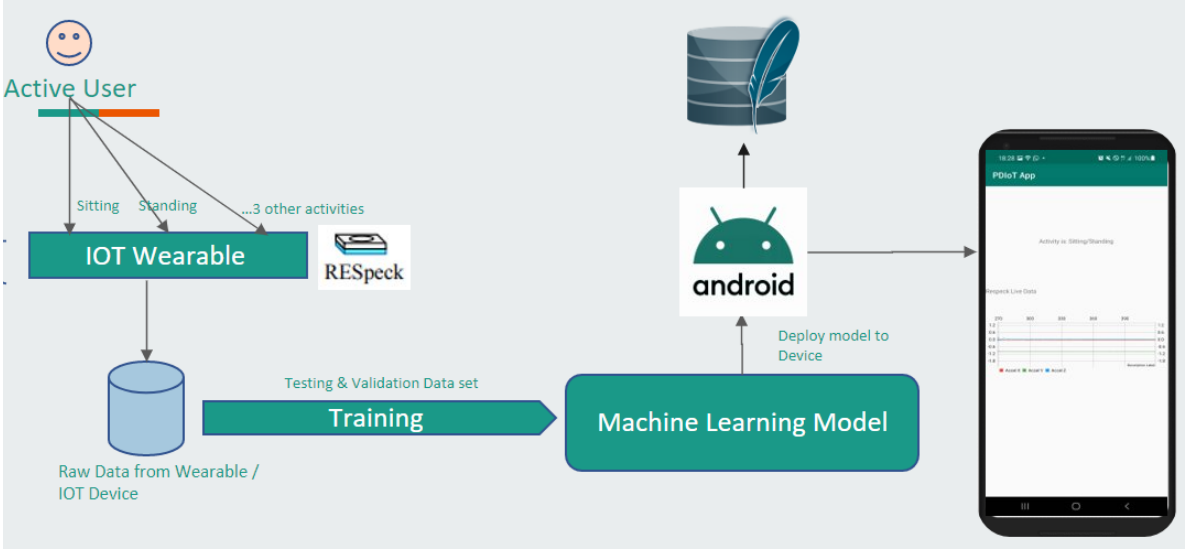

Figure 3: System architecture diagram

3 Methodology

3.1 System architecture

Figure 3 presents the diagram of the system. The interaction begins from an active user wearing the Respeck sensor attached to their lower left rib cage while performing one of the five activities classified by the app. In the first step, the user must establish a connection between the Respeck and the Android App. This is achieved in the "Connect sensors" activity, where the user needs to fill in the Respeck ID. There are multiple choices for connecting the Respeck:

1. NFC pairing if the phone supports NFC
2. Scanning the QR code of the Respeck
3. Manually inputting the ID into the field

The sensor only needs to be paired once as its ID will be remembered by the app whenever it gets started again.

Packets of data from the IoT device are sent over BLE to the connected smartphone. A TensorFlow Lite model trained on data previously recorded and stored as CSV files in the phone's storage was deployed in the current app. All real-time data coming from the connected sensor is passed to this model which predicts the current activity being performed, as well as to local SQLite storage. The stored data is used for analysis over a longer period of time which can be seen in the app under the "Health Data" page.

3.2 Hardware

The RESpeck device (figure 4) is a wearable wireless inertial magnetic unit (IMU) sensor which is attached with a plaster below the rib cage and was developed for measuring the respiratory rate in a simple and unobtrusive way by measuring the rotation of the chest wall using a triaxial accelerometer. The signals are digitised and transmitted by Bluetooth Low Energy (described in section 5.3). The sensor contains an accelerometer that measures the acceleration, observing the change in direction of gravity and a gyroscope. The latter directly measures rotational rate and its values can be drifted by gravity direction or linear accel-

eration and are given in degrees per second about the axis of interest. As the accelerometer struggles to independently resolve lateral orientation or tilt and to find out the location of the user, it is used conjointly with the gyroscope to gather acquire better information about the user's movements while also inferring its position. To record the data for training the model, one had to place the Respeck on their lower left rib cage, attaching it with MeFix tape (to ensure consistency across the dataset), as in figure 5. This choice of placement is beneficial for respiratory as well as activity monitoring purposes - the chest is the best location of the body for the extraction of respiratory signals, while also being very close to the human body centre of mass, which translates into well-defined movement for most activities [7].

Figure 4: Respeck device

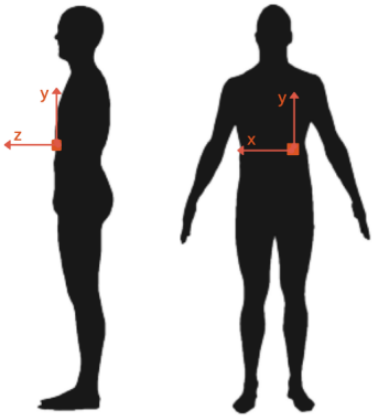

Figure 5: Respeck placement

3.3 Wireless communication

The communication between the Mobile App and Respeck is achieved through BLE (Bluetooth Low Energy), a protocol widely used for establishing wireless communications for low-power devices. BLE constitutes a single-hop solution applicable to a different space of use cases in areas such as healthcare, consumer electronics, smart energy, and security. Internet of Things (IoT) applications rely on BLE for local, energy-efficient data exchange between smartphones and resource-constrained peripherals or edge devices. The Android App used for the purpose of this coursework uses the RXAndroidBLE library which requires Java 8. Paired with it, the Respeck sends packets of accelerometer data at a rate of 25Hz

3.4 HAR Algorithm

Architecture

The HAR model is a CNN model, which consists of layers of convolutional neural network followed by a fully connected network (Dense). The HAR model's architecture is displayed in figure 6.

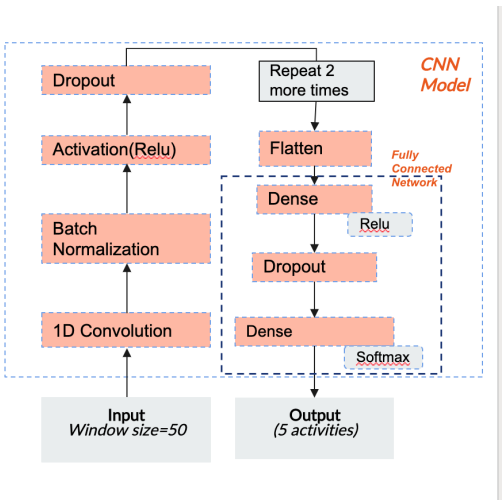

Figure 6: Architecture of the CNN model

Input Layer: The input consists of a row of datapoints with the dimension of (window_size, number_of_features), where window_size is 50 for our final model and number_of_features is 6. The features are the six types of data that the Respeck sensor collects: accelerometer in 3 dimensions and gyroscope in 3 dimensions.

Convolution: It is the mathematical operation that extracts high-level features/patterns from the input passed of dimension (50,6), and the core building block of a CNN.

Batch Normalisation: While the exact effectiveness is still an open research question [15], it is a technique for training very deep neural networks that standardizes the inputs to a layer for each mini-batch. Adding these layers increased the validation accuracy during the stage of designing the model, so they were adopted for the final model.

Activation: Activation layers are the main component of the design of a neural network. The objective of it is to transform the summed weighted input from the node into an output value to be fed to the next hidden layer. activation functions bring non-linearity to solve the complex decision boundary problem. There exist different types of activations, such as tanh. The activations used throughout the model are ReLU, except the final, output layer where a softmax is used to generalise the outputs into different probability distributions where the highest probability is the activity that the model classifies.

Dropout: Dropout is a regularisation technique that randomly drops out a fraction p of hidden units for each mini-batch. The motivation behind them is so that Dropout can reduce the dependence of hidden features, and that helps with preventing overfitting. We also observed higher validation accuracy after adding dropout layers after applying batch normalisation.

Dense Layer: Dense layer is simply a layer that is deeply connected with its previous layer, meaning that each neuron in that layer receives input from all neurons of the preceding one. In essence, the layer performs an affine (linear) transformation from the inputs and the weights of the neurons. The width of the layer of our model is 156 units/neurons, a process found by observing the effects of increasing and decreasing the number of units on the training and validation loss graphs.

Output Layer: The final layer, using the softmax activation function, gives out the result of the classification with 5 dimensions, which are the different activities we are classifying.

Training the model

In addition to the architecture of the model, we discuss the stages of training and validating the model. We report the best test accuracy in the Results section.

Training: The training model uses the clean Respeck dataset provided by the course organisers for all the activities, except falling ones where the original, unclean Respeck data collected by the students are used. This decision was made in order to incorporate more falling data, which was absent in the newer dataset. The learning rate for the Adam optimiser is 0.0005, after being optimised.

Preprocessing: Before passing the data to the model, they are preprocessed by splitting each recording of a certain activity into sliding windows of a specific size (window_size). Additionally, some overlap between each produced window is

introduced, depending on the window size. The appropriate window sizes and overlap parameter were experimented with during the production of our application. For our final model, we found that it was best to use a window size of 50 datapoints (equivalent to 1 seconds under 25 Hz), and an overlap of 50% (step size = 25).

Validation: During the initial stage of training and designing the model, *GroupShuffleSplit* is used to visualise the validation accuracy with a split of 80-20 on training and validation set. In the end, after further optimising the model, we would also use leave-one-subject-out cross-validation (*LOSOXV*), where a subject is left out as a validation set while the rest is used for training, to check the true validation accuracy of the model by combining the average.

3.5 Mobile App

The Android app was developed in Kotlin, using the sample *pdiotapp* provided for the course and following this tutorial for Android development.

The app handles a non-trivial amount of structured data hence it is highly beneficial to store data locally. In other words the user authentication is done using an SQLite Database [1]. SQLite is an Open-Source embedded SQL database engine that comes with Android and provides a relational management structure for storing user defined records. The App contains a registration form and a login page, with inbuilt methods for checking the user against the internal listing and validating the email format and correctness of the password.

Some prerequisites worth mentioning for installing the app are: Android 6.0 or higher, buildToolsVersion of 30.0.1 and min SDK version 23.

3.5.1 App Design

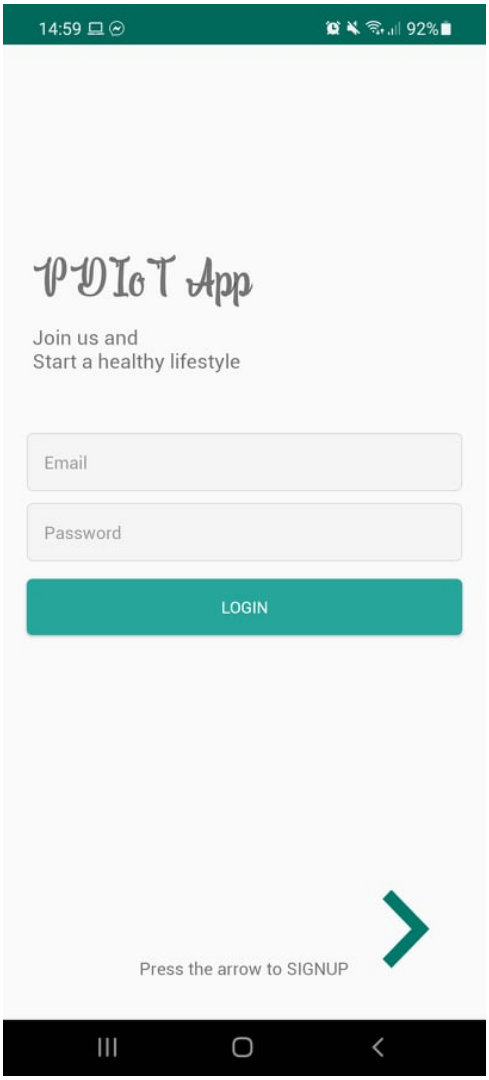

Figure 7: Login page

The screens for the app were designed using the Material Design guidance found here, taking into account the best practices of user interface design, usability being our top priority. The app uses a neutral colour palette, accessible for users with visual disabilities, with dark coloured buttons which make the text more visible through contrast, intuitive button shapes and labels which make the

navigation through the app predictable, a colourful logo, and activity icons which make the app more user-friendly.

Figure 7 presents the Login page of the PDIoT App, which is the first page the user sees when opening the app. This page contains the app name, motto as well as fields for user email and password which need to be filled in in order to access the app. There are two colourful buttons, for accessibility purposes, one for Login and another for Register (at the bottom of the page), for new users.

Figure 8 is the Register page, where new users can insert their full name, email address, and desired password twice in order to check the password is typed incorrectly. All fields must be filled in in order to continue and the two passwords need to match, otherwise an error bar will appear at the bottom of the page with an informative message. If an email address is already assigned to an account an error will pop up informing the user that the email already exists in the database. If all fields are filled in correctly, the user can click on "Register" and successfully create an account.

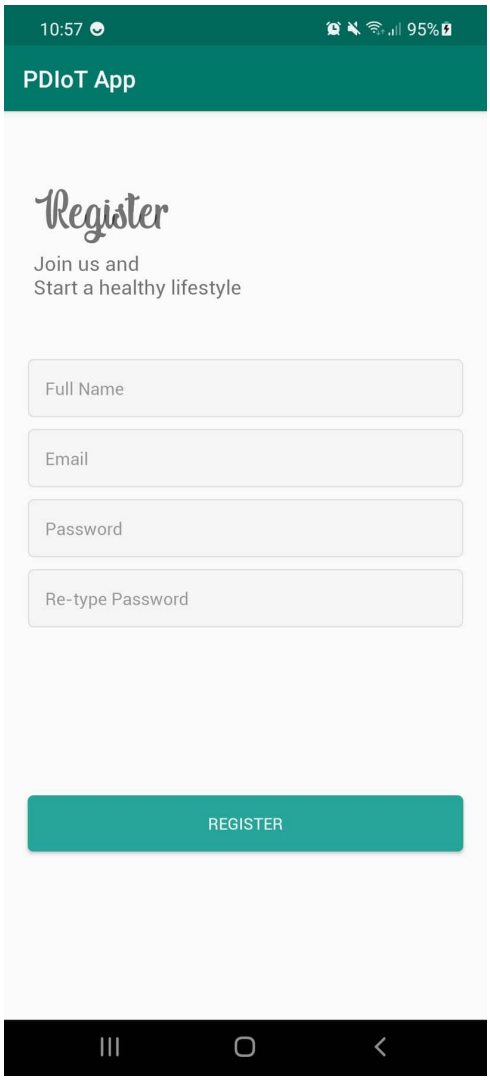

Figure 8: Register page

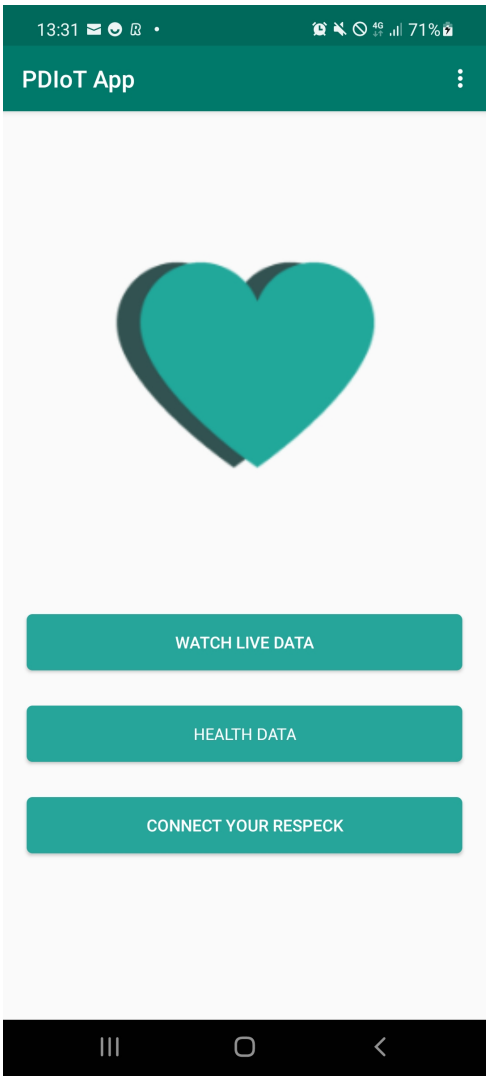

Figure 9: Home page

The home page of the app is displayed in figure 9. The page contains the app logo and its 3 main modules: the real-time physical activity classification (Watch Live Data), past data analysis (Health Data), and a page for pairing the sensor (Connect your Respeck).

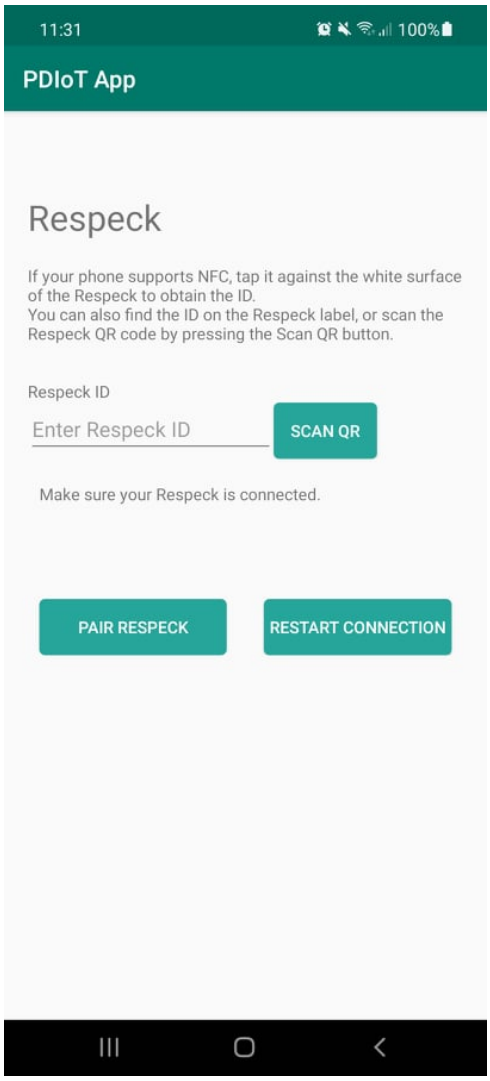

Figure 10: Connect your Respeck page

Figure 10 represents the *Connect your Respeck* page where the user can see a short paragraph containing the instructions on how to achieve this. The pairing can be done through NFC, through typing in manually the Respeck ID, or by scanning the sensor's QR code which can be done by clicking on the button next to the ID field. If the smartphone supports NFC connection and this setting is enabled, an "NFC Enabled" message will shortly appear on the bottom of the page. After entering the Respeck ID through one of the three methods mentioned, the user can click "Pair Respeck" to establish the connection with the sensor. These steps only need to be completed once as the ID information will be remembered by the app. Alternatively, if the user wishes to connect a new sensor, they need to press "Restart Connection" and complete the aforementioned steps once again.

The users of the app can also access an analysis of their past data in the *Health data* module. Figure 11 presents this module which is composed of 5 buttons corresponding to each of the activities classified by the machine learning model. Suggestive icons are used along with the title of each of these activities, for a more user-friendly interface.

In figure 12 a weekly analysis of the current user's Running data can be seen. The data is aggregated by date, in the form of columns measuring the number of seconds spent doing this activity. The date is displayed at the bottom of each column and the total amount of time is written on top of the corresponding column. The y-axis of the plot measures the number of seconds as well, with ticks for milestones. Every column can be expanded by zooming in on it so that the associated text can be easier to read.

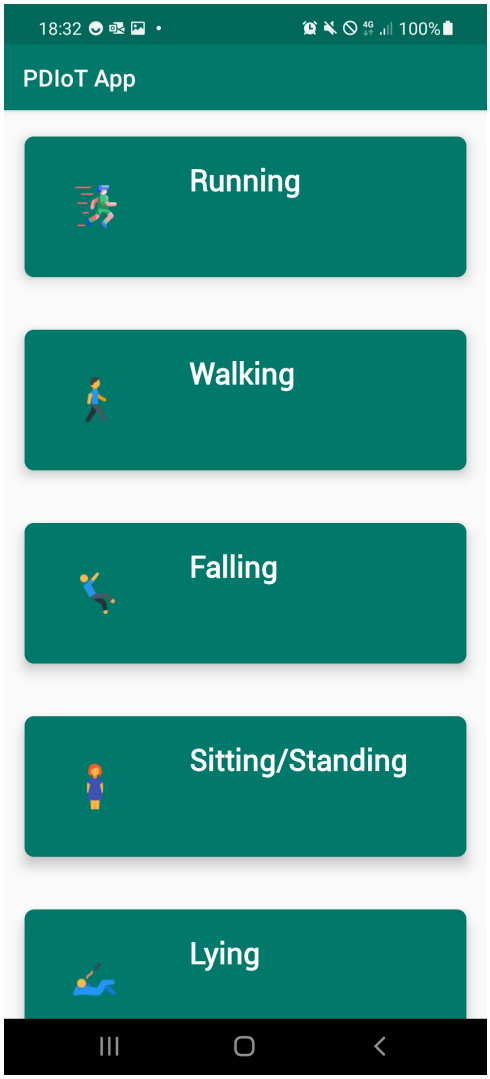

Figure 11: Health data page - menu

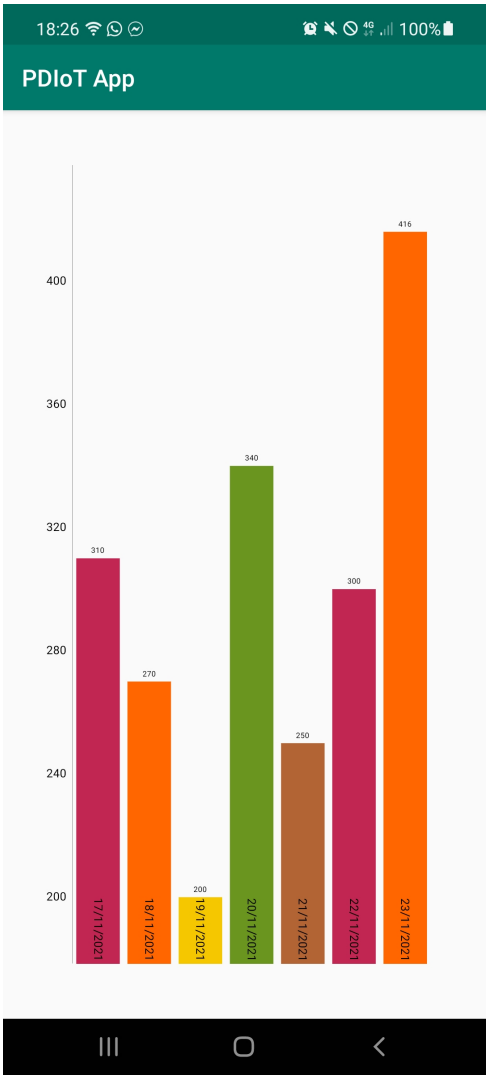

Figure 12: Health data page - Running

Figure 13: Watch Live Data page

Figure 13 contains the *Watch Live Data* page, where the user can see the real-time classification of the activity they are currently performing. On the top of the page some text is used to insert the predicted activity and below it is a visual graph that follows the live changes in the acceleration on the three-axis.

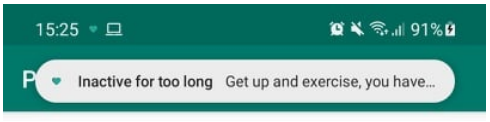

Figure 14: Notification to get active

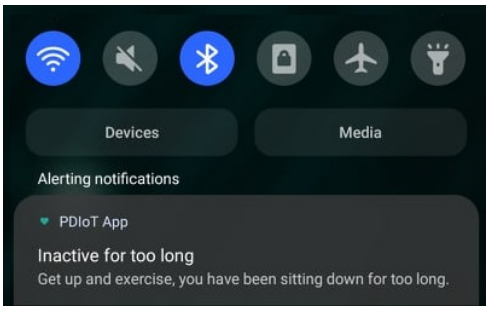

Figure 15: Notification to get active

Figures 14 and 15 showcase the notification message a user receives after being inactive for too long (their current activity for this period being classified as sitting). The notification pops up even when the app is not running, as a reminder for the user to start moving/exercising.

3.5.2 Application Performance

The app performance was measured using the Android phone settings. From the storage point of view, the app itself takes 19.52 MB of memory. Additional storage is used for the storage of data and cache, but both of these can be regularly cleared by the user through the Android settings. The average RAM usage for 3 hours is 53 MB, which is equal to 17.7 MB per hour. Figures 16 and 17 present the app performance as recorded by the Profiler in Android Studio. It can be seen the app is very light from CPU usage and energy consumption perspectives. When the app is running in the foreground, the CPU reaches a rate of 12%, while in the background the value can go below 3%.

Figure 16: App performance in foreground

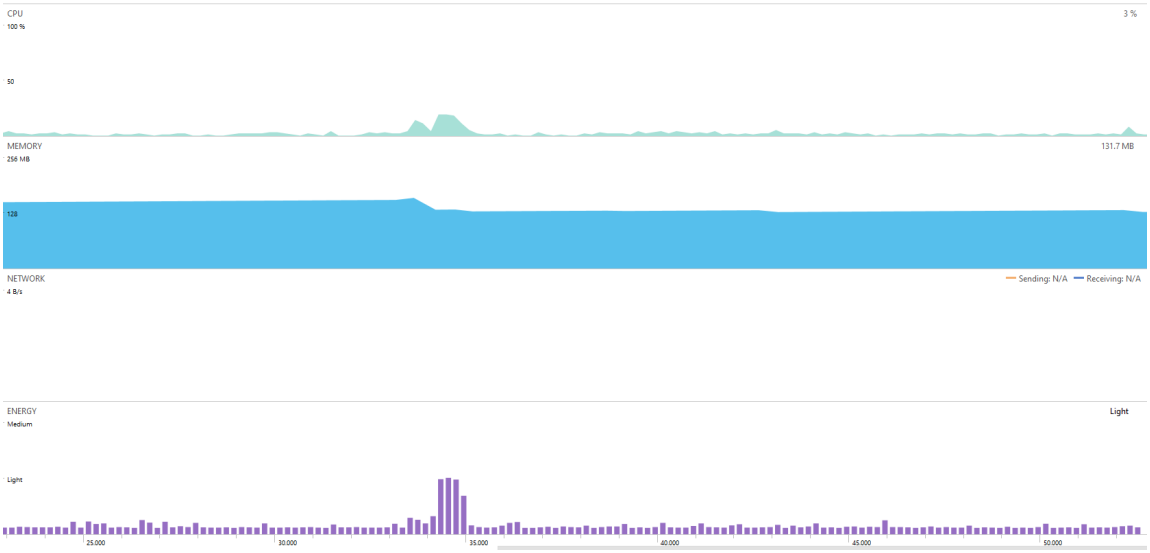

Figure 17: App performance in background

3.6 Software organisation

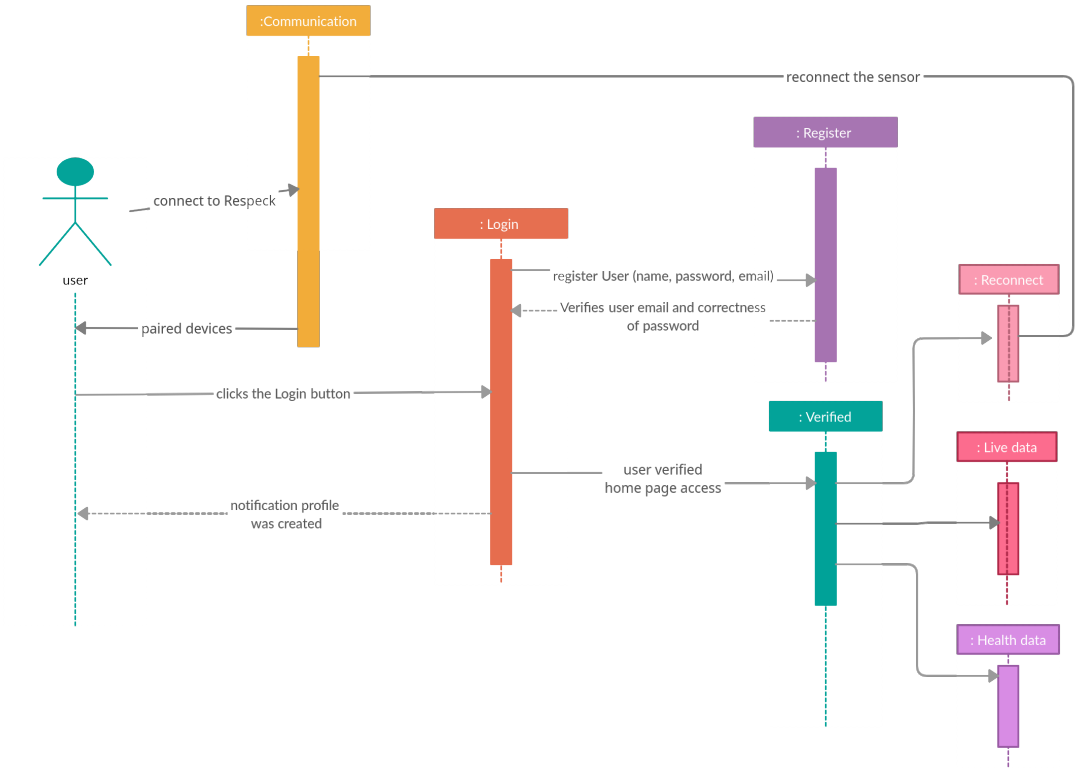

Figure 18: App flow

Figure 18 explicitly describes the flow through the app, a general use case. The user has to login and pair the device and the Respeck only once. However, there are options to connect the devices again or log out and select different credentials.

Implementation details

We have used the PhilJay/MPAndroidChart version 3.1.0 graph and chart Android library [12] to view and model our data. For each of the listed activities in section 1.2, the app is showing a dynamically generated customized graph.

To ensure compatibility with Android 8.0 (API level 26) and higher we needed a channel ID for the NotificationCompact.Builder. However, this is ignored in older versions hence we had to implement pushing notifications for sedentary activity in two different ways based on the Build.VERSION.SDK_INT to make sure the app would work on a wide range of phones. Moreover, it is worth mentioning that we used NotificationCompat APIs from the Android support library as they allowed us to add features available only on newer versions of Android while still providing compatibility back to Android 4.0 (API level 14).

Figure 19 below represents a UML diagram that gives an overview of the core classes and the interactions between them.

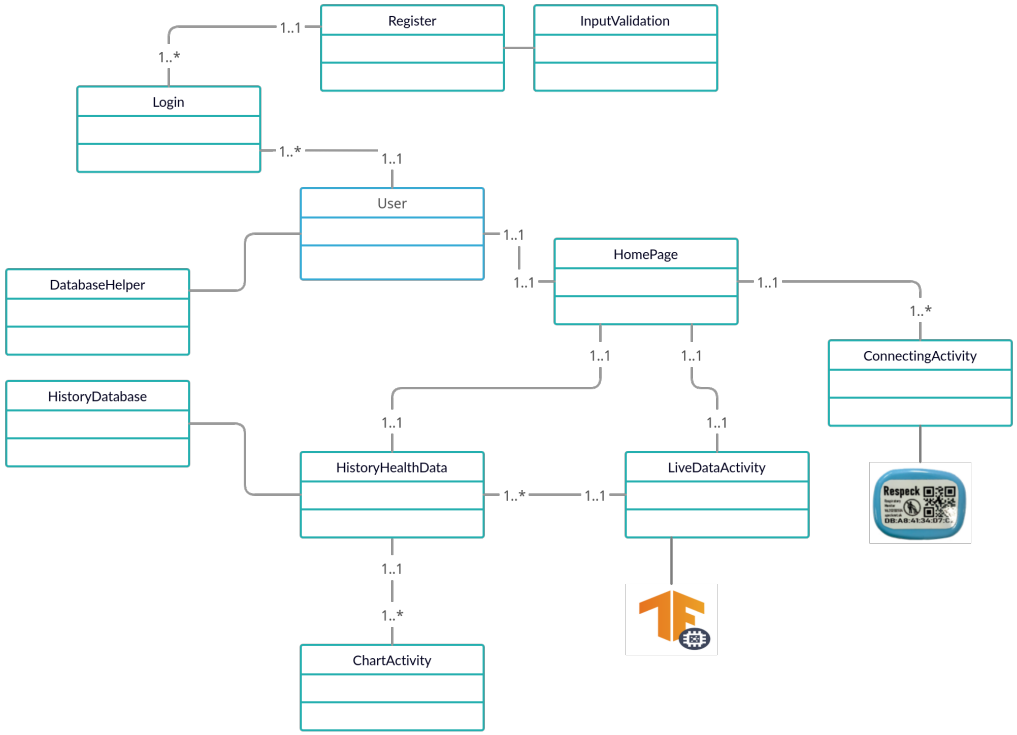

Figure 19: UML diagram - simplified

- The **InputValidation Class** has two main methods: **isInputEditTextEmail** and **isInputEditTextMatches** which ensure the user has filled in a correct input (valid email and typed passwords matching).

- A Fragment represents a reusable portion of the UI, defining and managing its own layout, having its own lifecycle, and handling its own input events. We believed that it will be beneficial to use this over an activity when implementing the register page. This mainly handles the insertion into the SQLITE database, check method **postDataToSQLite**.

- In the **MainActivity** we have used **Shared Preference** for auto-login functionality, the user being redirected to the HomePage.

- **ConnectingActivity** supports the NFC pairing if the phone has enabled NFC, scanning the QR code on the Respeck, or storing the manually inputted IDs (using Shared Preference again). The **BluetoothSpeckService** contains all the methods required to establish smooth wireless communication between the app and the Respeck.

- **LiveDataActivity** uploads the tflite model through **loadModelFile** and uses a **BroadcastReceiver** object to obtain the acceleration on the three axis and gyro data from the sensor. At the point of receiving more than 50 data points, the method **updatePrediction** is outputting the prediction on the

screen and the corresponding graph for the prediction is updated.

- **A RecyclerView** in Android makes efficiently displays large sets of data.
  In **HistoryHealthData** we fetch data from another SQL table created in the **HistoryDatabase** class and define how each item looks like for each of the 5 grouped activities, and the RecyclerView library dynamically creates the elements when they're needed. The functions updateDuration() and updateActivity() handle the writing to the db, updating the time spent on each activity every time a new prediction is made.

- **getScoreList(): ArrayList<DataPointsGraph>** is how the BarCharts are populated (layout created using the MPAndroidChart library). This method takes the data (score in terms of duration per activity type) for the past 7 days from the HistoryDatabase. A new pull request to the db is made every time a user clicks again on the tab, hence the app will be displaying an updated chart.

4 Results

4.1 Benchmarks

While many experiments and efforts were done to reach the final model including the addition of the dropout layer, and optimising the hyperparameters like the probability dropout retention rate, the learning rate, and using other models and designs, we present the most notable ones that were done on CNN. The table presents the different window sizes, and its effect on the validation rate with the GroupShuffleSplit on 80-20 split.

Model	Window	Step size	Val. Acc
1	22	18	93 %
2	25	25	93%
3	50	25	96%

Table 1: CNN Model iterations on different window size. Step size represents the overlap between each sliding window, where overlap is equal to $\frac{windowsize-stepsize}{windowsize}$

Using the model with the best validation accuracy on Group-Shuffle-Split (3rd model shown above), the leave-one-out result on the validation set returned 94% accuracy.

4.2 Critical analysis of the implementation using quantitative methods

Validation set

The main initial motivation behind choosing lower window sizes were to include in our training falling data as most recordings were below 1 second. However, the model's performance suffered overall as later introspection demonstrated; the model trained with higher window sizes gains a better validation accuracy. We deduce that this is due to the CNN model being able to learn more patterns when trained on longer window size. Further recordings should encapsulate longer recording time, despite the stillness in falling (if one were to fall down quickly), as that would allow the model to capture more patterns from the data to generalise well.

Furthermore, as mentioned under Benchmarks, we have gained a lower accuracy in validation set when averaging for every subject. It should be noted however that some subjects were left out of the testing phase because they had missing activities, which made it difficult to validate the model of fixed dimension to changing vali-

dation set size (dimension). However, it should also be noted that the model performed poorly on some subjects. The low performance could be that the validation set contained hard examples.

Upon looking at the data for the subject with a validation accuracy of 72% overall, we observed that the classification report metrics were low for the Lying activity. Analysing the accelerometer data in Respeck on one instance (20 by comparing with other subjects with accuracy close to 100%, we see that the s1541031 plot looks very different from each other. More specifically, between the values 500 - 1500 samples, where the green plot is invertedly mirrored, heading to the other direction compared to the other two plots. This suggests that the data had subjects that have potentially worn the sensor in a different orientation, or perhaps recorded the wrong activity. This allows us to conclude that removing this data would help enhance the model, provided the user wears it correctly. On the other hand, manual calibration of the sensor placement could provide the user more control and understanding of the interface, making the model predictions more accurate and meaningful.

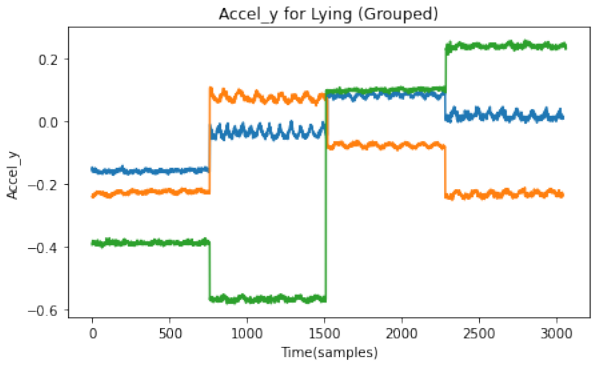

Figure 20: Acceleration data in y-axis for three subjects

In addition, when we performed GroupShuffleSplit for validation, the best performing model seemed to have started overfitting after epoch 4, the phenomenon where the weights become too flexible and change to match identically to the outputs of the training set. This can be shown by the increase in validation loss and decrease in validation accuracy as the model's performance on the training set gets better (higher in accuracy and lower in loss). The results in the benchmarks in Table 1 show the accuracy at the highest peak, but during the final training model on all datasets, it was trained on 10 epochs. Our model may perform better if we train on lower epochs, to avoid overfitting, or apply other known regularisation techniques like L1 or L2 regularisation.

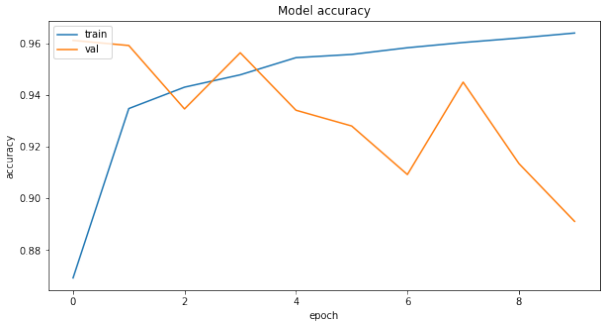

Figure 21: Accuracy Plot for HAR Model on Validation data

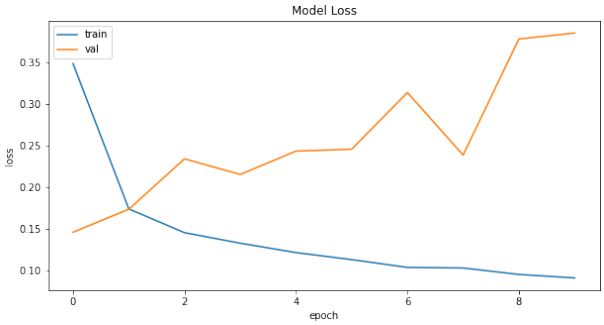

Figure 22: Loss Plot for HAR Model on Validation data

Test set

Observing the confusion matrix on the test set (Figure 23), we can visualise the performance for each activity. Among all activities, the model had the lowest performance in classifying Sitting/Standing, outputting Lying 1% of the time on the test set. However, the model performed well vice-versa on correctly inferring Lying as the correct activity, despite the multiple variations of it on the dataset(Lying on the back, Lying on the left, Lying on the right). The former misclassifications might be due to grouping multiple inclinations of sitting together, such as sitting forward and sitting backward. The increase in the variance of the dataset could be what confuses the model, predicting Lying in some cases as it closely resembles one of the lying positions. However, the model performed well vice-versa.

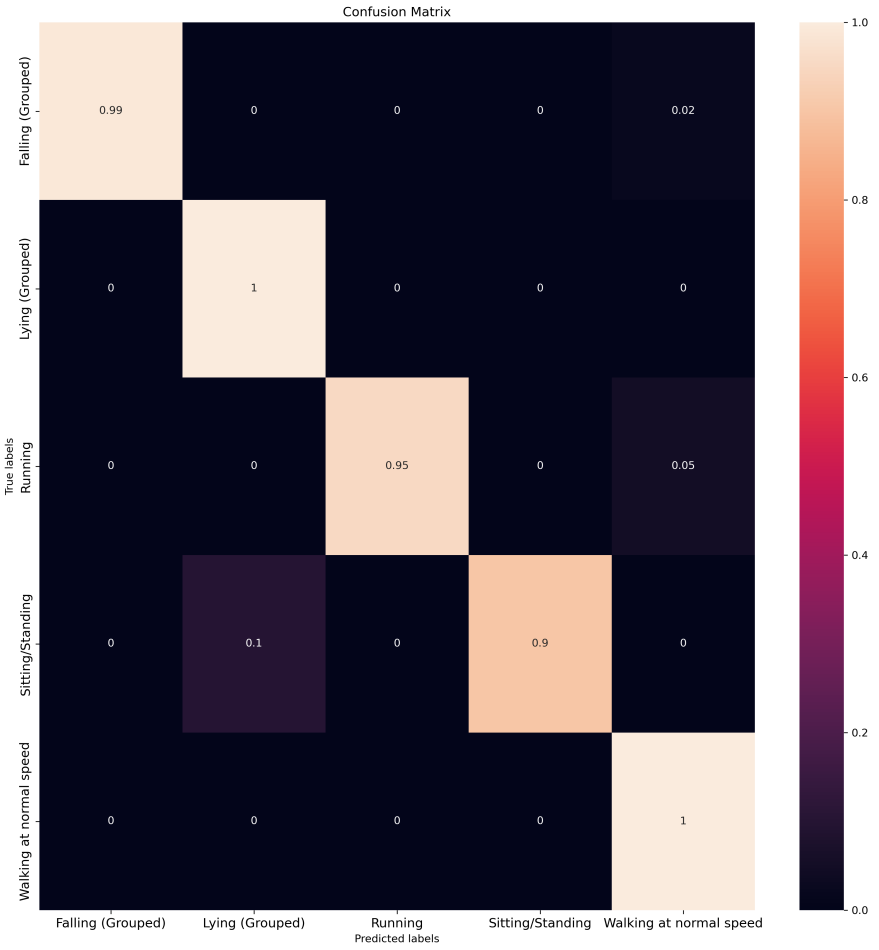

Figure 23: Confusion Matrix for HAR Model on Test data

In addition to the activity Running, the model computed wrong predictions 5% of the time for the test set. This could be attributed to the placement of the Respeck in the lower ribcage, leaving periods of time where the motion of running and walking at normal speed is not distinguishable. The further detailed analysis could investigate the data to confirm the interpretation and possibly indicate any outliers where the subject might have accidentally recorded themselves transitioning from running to walking as they stopped the recording.

Looking at the classification report on the test set (Figure 24), we find further details on recall and precision. Precision is a metric that calculates the quantity of correct positive predictions, and recall is a metric that calculates the proportion of actual positives that was correctly predicted.

$$Precision = \frac{TP}{TP+FP} Recall = \frac{TP}{TP+FN} \quad (1)$$

where TP stands for True Positive, FP for False Positive, and FN for False Negative.

Despite having high accuracy, it is also crucial to look at other metrics since we are dealing with an imbalanced dataset (some activities that were passed to training has a higher number of data points, due to the preprocessing phase of splitting into window sizes with a certain step-size or overlap). The support column in the report is the number of samples of the true response that lie in that class, and we can see how there are more classes in Sitting/Standing compared to Walking at Normal Speed. This imbalance is also due to the effect of merging activities together.

Initially, we observed the accuracy for Lying rounds to 100% accuracy. However, we notice in the report that the precision is low, while the recall is high. This means that the model returns a lot of false positives, and retrieves many results that classify it as Lying (low negative rate). In our current general application, if the sensors were to be used in hospital settings for a particular purpose (i.e new dataset which includes extra breathing activities for patients for health monitoring purposes), a higher recall is necessary. However, for general usage for a user to visualise their activity, it may be more desirable to increase precision.

The Falling activity garnered the best performance metrics on precision and recall, giving the highest F1-score (the harmonic mean of precision and recall).

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * TP + FP + FN} \quad (2)$$

This observation infers that the data for falling activities have different patterns to the other data, and the model was able to successfully train and learn these patterns.

	precision	recall	f1-score	support
0	1.00	0.99	1.00	142
1	0.91	1.00	0.95	231
2	1.00	0.95	0.97	55
3	1.00	0.90	0.95	227
4	0.94	1.00	0.97	58
accuracy			0.96	713
macro avg	0.97	0.97	0.97	713
weighted avg	0.97	0.96	0.96	713

```
Falling (Grouped)..... Accuracy: 0.99, Precision: 1.00, Recall: 0.99, F-score: 1.00
Lying (Grouped)..... Accuracy: 1.00, Precision: 0.91, Recall: 1.00, F-score: 0.95
Running..... Accuracy: 0.95, Precision: 1.00, Recall: 0.95, F-score: 0.97
Sitting/Standing..... Accuracy: 0.90, Precision: 1.00, Recall: 0.90, F-score: 0.95
Walking at normal speed..... Accuracy: 1.00, Precision: 0.94, Recall: 1.00, F-score: 0.97
```

Figure 24: Classification Report for HAR Model on Test data. 0: Falling, 1: Lying, 2: Running, 3: Standing/Sitting, 4:Walking

4.3 Testing

The final test accuracy reaches 96% during offline testing, as demonstrated in 24. When testing the application and the model live, we notice that the latency of updating the graphs can be a bit slow at times. This could be from the model, needing 2 seconds (window size of 50) to output the result.

However, the activity outputted in the application after the small period is classified correctly. Hence, the issue usually arises during the transition from one activity to another. In these, the classification takes time to adjust, with the classification reported as falling. It takes around 1-3 seconds for the model to stabilise.

5 Conclusions

To conclude, the literature review covers the subtleties regarding how human activity recognition has been approached throughout recent history, the brief description of the system summarizes the architecture design and communication between its components and the results section is an explication of the efficiencies and hindrances of changing the variables in the model. The HAR algorithm we adopted, a CNN model classifying a subset of 5 activities, with the data grouped into sliding windows of window size 50 and 50% overlap proved to be efficacious. Furthermore, the Android application developed also provides all the essential features, not least the user is able to create a profile and view its retrograde activity displayed in a user-friendly configuration. Regarding the more complex aspects, the app can push notifications for reminders to exercise and the model accuracy reaches 96%.

Future improvements

The potentially recognizable activities vary in complexity hence successfully classifying a larger type of human movements than our model currently is able to predict, is one key improvement we would like to achieve.

Second of all, performing the computations and the live classification in the cloud would bring a lot of benefits as it allows deep learning models to scale efficiently and at lower costs using GPU processing power.

Third, after experimenting with LSTM models of higher window size (Group-Shuffle-Split of same seed), it achieved 98% validation accuracy and validation loss of 0.08, which is promising. Hence, trying different HAR methods, such as LSTMs would possibly result in achieving higher accuracy and better predictions. The reason this was not included in the report earlier is that the pre-emptive LSTM model is not made into production, and more experiments (including LOSOXV) would be necessary to conclude any further.

In addition, applying a combination of preprocessing techniques could further improve the model accuracy. A further improvement would be to ensure that the transition between one activity to another displays a suitable activity. One idea would be to include the classification of "Movement" activity in the model.

Lastly, we will focus on tailoring the application for a specific purpose, rather than providing general functionalities.

Bibliography

[1] *Sqlite Home Page*.

[2] James P Byrnes. *Cognitive development and learning in instructional contexts*. Allyn and Bacon Boston, 1996.

[3] Kaixuan Chen, Dalin Zhang, Lina Yao, Bin Guo, Zhiwen Yu, and Yunhao Liu. Deep learning for sensor-based human activity recognition: Overview, challenges and opportunities. 01 2020.

[4] Diane Cook, Kyle D. Feuz, and Narayanan C. Krishnan. Transfer learning for activity recognition: a survey. *Knowledge and Information Systems*, 36(3):537–556, June 2013.

[5] F Foerster, M Smeja, and J Fahrenberg. Detection of posture and motion by accelerometry: A validation study in ambulatory monitoring. *Computers in Human Behavior*, 15(5):571–583, 1999.

[6] Wilton W. T. Fok, Louis C. W. Chan, and Carol Chen. Artificial intelligence for sport actions and performance analysis using recurrent neural network (rnn) with long short-term memory (lstm). page 40–44, 2018.

[7] Teodora Georgescu. A remote pulmonary rehabilitation system using the wearable respeck monitor.

[8] Charmi Jobanputra, Jatna Bavishi, and Nishant Doshi. Human activity recognition: A survey, Sep 2019.

[9] Francisco Javier Ordóñez Morales and Daniel Roggen. Deep convolutional feature transfer across mobile activity recognition domains, sensor modalities and locations. In *Proceedings of the 2016 ACM International Symposium on Wearable Computers*. ACM, September 2016.

[10] Henry Friday Nweke, Ying Wah Teh, Mohammed Ali Al-garadi, and Uzoma Rita Alo. Deep learning algorithms for human activity recognition using mobile and wearable sensor networks: State of the art and research challenges. *Expert Systems with Applications*, 105:233–261, 2018.

[11] Francisco Javier Ordóñez and Daniel Roggen. Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition. *Sensors*, 16(1), 2016.

[12] Phil Jay. Chart library for android.

[13] Schalk Pienaar and Reza Malekian. Human activity recognition using lstm-rnn deep neural network architecture. 05 2019.

[14] Sreenivasan Ramasamy Ramamurthy and Nirmalya Roy. Recent trends in machine learning for human activity recognition—a survey. *WIREs Data Mining and Knowledge Discovery*, 8(4), March 2018.

[15] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Mądry. How does batch normalization help optimization? In *Proceedings of the 32nd international conference on neural information processing systems*, pages 2488–2498, 2018.

[16] Abdulhamit Subasi, Kholoud Khateeb, Tayeb Brahimi, and Akila Sarirete. Chapter 5 - human activity recognition using machine learning methods in a smart healthcare environment. In Miltiadis D. Lytras and Akila Sarirete, editors, *Innovation in Health Informatics*, Next Gen Tech Driven Personalized MedSmart Healthcare, pages 123–144. Academic Press, 2020.