THE UNIVERSITY *of* EDINBURGH

PDIoT Final Report (2024-25)

Human Acticity Recognition with Respeck and Thingy Sensors through an Android App

Abstract

This report presents the development and evaluation of an activity recognition system as part of PDIoT Coursework 3. Our implementation includes an analysis of several machine learning models to achieve the classification of eleven activities - ascending and descending stairs, running, walking, shuffle walking, lying down, right, left and on stomach, sitting/standing and miscellaneous - and four respiratory conditions - coughing, hyperventilating, normal and other -, showing 1D CNNs the most effective. We further provide methods to analyse sleep quality, including the computation of the number of positional changes during sleep or the classification of sleep-wake cycles. The implementation of an Android App is also provided to record data, visualise the real-time classification of activities, and access historical data. The project objectives, methodologies, and results are discussed in detail, with critical analysis and future work proposed.

Group B1

Alioa Iglesias Román	(s2101906)
Julia López Gómez	(s2107370)
Isabel Martínez Barona García	(s2096890)

Friday 17th January, 2025

Contents

Chapter 1

Introduction

1.1 Project Aims and Objectives

The main objective of this project is the design and implementation of an advanced IoT system capable of accurately classifying human activities using sensor data collected from two specialised Inertial Measurement Unit (IMU) sensors: Respeck and Thingy. These sensors provide data streams that serve as inputs to different deep learning models, developed and optimised to achieve the highest possible classification accuracy.

In addition to the core classification system, another goal of the project is the development of a user-friendly and intuitive Android application. This app is designed to be accessible to users of all experience levels, ensuring ease of use. The application functions as both an interface for showing activity classifications and works with the underlying system, providing a simple and practical user experience. It also allows users to directly connect the sensors and record the activities so they can access the activity history for future reference.

The project aims to provide hands-on experience with the entire process of developing an IoT system, covering every stage from data collection to deployment. It starts with gathering raw data from sensors, followed by cleaning and processing the data. Next, the focus goes to designing a robust model for activity classification, using advanced techniques to achieve high accuracy, and finally the implementation of the Android application. Through this comprehensive approach, the project offers very valuable experience that will be highly relevant for future endeavours in the field. The overarching objective is the creation of a comprehensive and impactful solution for Human Activity Recognition, demonstrating the potential of IoT technologies in real-world applications.

1.2 Brief Description of Methods Used

Different methodologies were tested and analysed to perform human activity recognition. For physical activity classification, both Long Short-Term Memory (LSTM) and Convolutional Neural Networks (CNN) were employed. Meanwhile, for the categorisation of social signs, Support Vector Machines (SVMs), Random Forests and CNNs were explored. Across both domains, the 1D-CNN was the superior choice due to its capability to process sequential data and find underlying pat-

terns in temporal data.

For sleep analysis, several features were calculated. For instance, sleep-wake cycles were calculated through unsupervised methods due to the absence of labelled data. Methods such as K-Means, Expectation-Maximisation and Hierarchical Clustering were explored due to their effectiveness in clustering data into categories.

From the sensor data and sleep questionnaire answers, we obtained metrics such as the number of position changes and sleep efficiency of each subject. This, combined with the sleep-wake cycles, allowed us to develop a metric to predict a Sleep Quality Index, providing a quantitative measure for sleep quality.

1.3 Summary of Results

Through the use of CNNs, significant accuracy was achieved in the classification of physical activities. Our model categorised the 11 activities, delivering over 90% accuracy through 5-fold cross-validation for both static and dynamic activities. Similarly, for the classification of social signs, an accuracy of over 80% was obtained for each activity, indicating that meaningful patterns were abstracted from the data. These results show the efficacy of CNNs in handling temporal data and their potential in real-world applications where accurate activity recognition is crucial.

Through the application, we displayed the real-time classification of these activities and achieved an average of 99.09% for physical activities and 84.17% for social signs, proving the effectiveness of our models.

In regards to sleep, reasonable results that resonate with the sleep questionnaire answers were obtained through K-Means and EM algorithms. This, joined with the calculation of overnight orientation changes and sleep efficiency, a Sleep Quality Index was calculated which agreed with the written answers with an accuracy of 67.65%.

Chapter 2

Literature Survey

2.1 Context

The focus of this project is on Human Activity Recognition (HAR), which refers to the ability to interpret human body gestures and movements through sensor data to identify specific activities or actions. HAR is a dynamic research area that employs a variety of machine learning algorithms to recognise both simple and complex activities, such as walking, running and sleeping [21]. Its applications are diverse, ranging from enhancing patient rehabilitation to detecting critical health conditions linked to shifts in activity patterns.

One of the most researched fields of HAR application is healthcare, where systems are commonly deployed in hospitals, rehabilitation centres, or residential environments. In these contexts, HAR devices are used to track elderly people's daily activities, prevent diseases, manage chronic conditions, detect early signs of illness, monitor abnormal conditions for cardiac patients, and perform fall detection, among other uses [28]. Beyond healthcare, HAR has been applied in surveillance systems in public places with the goal of preventing crimes and dangerous activities by real-time detection of human activities [27]. Additionally, HAR has been explored in the Human-Computer Interaction (HCI) domain, particularly in gaming and exergaming, where it enhances user interaction and engagement [3].

Studies on HAR can be broadly categorized based on the type of sensors and data used for the detection of the activity. These fall into two main categories: data collected from wearable sensors or accelerometers or data derived from video frames and images [15]. In recent years, wearable sensors have gained more attention due to their cost-effectiveness, non-location-dependent nature, and ease of use [21]. These sensors typically employ statistical and frequency-based features for activity identification.

The process of HAR involves multiple stages beyond the initial data collection by sensors. These stages include data segmentation, preprocessing, feature extraction, and, finally, the classification of the activity. Each stage plays a critical role in ensuring accurate recognition of human activities and improving the system's overall reliability [2].

2.2 Sensors

Previous studies have explored the use of both the Respeck sensor [12] and the Thingy sensor [8], providing valuable insights into their applications for activity recognition.

[10] highlighted that the type and placement of sensors have a higher impact on the classification accuracy than the number of sensors used. While adding more sensors can improve accuracy, it introduces challenges such as increased calibration complexity, higher storage requirements, greater power consumption, and more demanding communication protocols.

In a related study, [22] compared multiple sensor placement locations and identified that the optimal combination for maximising accuracy involves placing sensors on the right wrist, left hip and chest. This combination was found to reach a balance between accuracy and practicality, emphasising the importance of strategic sensor placement in achieving precise results.

2.3 Traditional Machine Learning Models used

Traditional machine learning methods, such as decision trees, K-nearest neighbours (KNN) [17], support vector machines (SVM) [6], and hidden Markov model (HMM) [23], have been widely used for activity recognition. However, these methods often rely heavily on hand-crafted features and require extensive data preprocessing, which can be time-consuming and computationally expensive [19]. Moreover, the process of feature engineering typically involves dimensionality reduction, which might lead to the loss of critical information inherent in the raw data, limiting the ability of the models to detect complex patterns. Studies in HAR have found that no single discriminative feature universally represents all datasets and applications with high accuracy [5]. As a result, recent research in HAR has focused on deep learning models.

2.4 Deep Learning

Deep learning is a branch of machine learning capable of learning representations directly from raw sensory data. It comprises multiple layers of neural networks to hierarchically represent features, progressing from low-level to high-level abstractions [21]. This approach has revolutionised several research domains, including image and object recognition, natural language processing, machine translation and environmental monitoring. Additionally, deep learning is particularly effective in time-series analysis [11], as it captures temporal dependencies and patterns within sequential data. This, along with the fact that it allows unsupervised learn-

ing, makes it well-suited for various applications such as human activity recognition.

The two main deep learning methods employed in HAR are Convolutional Neural Networks (CNNs), which were presented for activity recognition by [14] and [24], and Long Short-Term Memory (LSTM) networks, which were utilized to recognize human activities using mobile devices in [26].

2.4.1 CNNs

Convolutional Neural Networks (CNNs) are Deep Neural Networks with interconnected structures that have demonstrated exceptional performance in time-series data analysis [4, 1]. Due to their effectiveness, various CNN architectures have been developed and employed for human activity recognition (HAR). [13] proposed a novel HAR framework that utilises a Genetic optimization Algorithm (GA) to automate the selection of the optimal CNN architecture, addressing the challenges of manually designing neural architectures, which is often error-prone and time-consuming. Their work introduced the AUTO-CNN algorithm, which uses a variable-length encoding strategy to identify the most effective CNN architecture for HAR tasks. A key advantage of this approach is that it can determine the optimal depth of the CNN architecture without the need to predefine its maximum length, ensuring flexibility for various HAR applications.

[7] proposed a one-dimensional Convolutional Neural Network (1D CNN) model that employs a divide-and-conquer strategy for classifier learning combined with a test data sharpening technique. Their method involves a two-stage approach using multiple 1D CNN models. In the first stage, a binary classifier is constructed to identify abstract activity categories. In the second stage, two multi-class 1D CNN models are trained to recognise specific individual activities within the previously identified abstract categories. Test data sharpening is also introduced during the prediction phase to improve the activity recognition accuracy further. This approach has proved to be simple and effective, offering ease of implementation once suitable abstract activities for the first stage are identified.

2.4.2 LSTM

Long Short-Term Memory (LSTM) is a specialised type of Recurrent Neural Network (RNN) designed to effectively learn and remember long-term dependencies over long sequences of input information [25]. [29] proposed a stacked LSTM network for recognizing six distinct human behaviours using smartphone sensor data. The architecture includes five LSTM cells, trained end-to-end on raw sensor data. Prior to the stacked LSTM network, a single-layer neural network pre-processes the input data, optimising it for the subsequent LSTM layers. Additionally, multiple

studies have explored the combination of CNNs and LSTMs, showing promising results, as shown in [18] and [31].

2.5 Sleep analysis

For scenarios where labelled data is not available, deep learning proves useful by enabling unsupervised learning, often through clustering methods. The study presented by [16] highlights the limited research on unsupervised learning for Human Activity Recognition and provides a comparative analysis of three clustering techniques: Hierarchical Clustering (HIER), K-means and Gaussian Mixture Model (GMM).

Chapter 3

Methodology

3.1 System Description and Implementation

The system developed is a Human Activity Recognition IoT platform which uses IMU sensors and machine learning algorithms for real-time activity monitoring. The system allows the pairing of two sensors, a RESpeck and a Thingy device, allowing users to observe and record their physical activities through an interactive application interface.

For physical activities, machine learning algorithms (in particular CNNs) process the raw RESpeck data and classify it into one of the eleven different activity classes. It classifies it further using the Thingy sensor if it was categorised as Sitting/Standing in order to separate it into one of the two classes. CNNs are also employed to categorise social signs in real-time, and both results are displayed on the application's live data page. This process is presented in Figure 3.1.

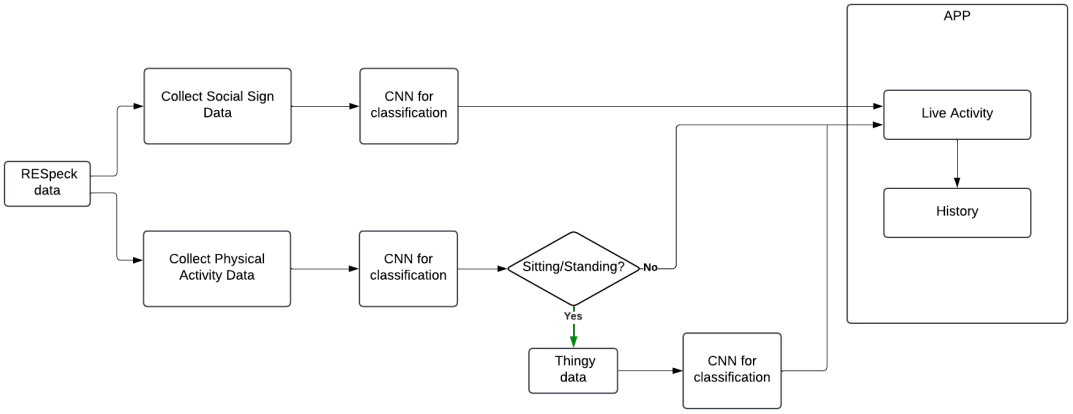


Figure 3.1: Overall system description.

The Android application allows users to watch their live activities and then access them on a *History* tab, showing them the amount of time spent doing each activity and offering insights into their trends over time. In addition, they can record their own activities, which are then saved onto the device.

Further analysis is done with regard to sleep. By calculating the sleep-wake cycles through unsupervised learning, the number of orientation changes during the night

and the sleep efficiency, a Sleep Quality Index was obtained. This can then be compared with the answers provided in the questionnaire in order to corroborate the metric used to assess sleep quality.

3.2 Algorithms and Methods for Activity Recognition

In this section, we evaluate various algorithms suited for classifying physical activities and social signs. As a result, we explain the strengths and weaknesses of each approach and provide a justification for our final decision, which involves 1D-CNNs for both systems.

3.2.1 Physical Activities

To achieve successful activity recognition, three distinct methodologies were employed, leveraging deep learning and machine learning approaches. Each method addressed different classification problems and utilised various configurations of hyperparameters for optimal performance, which will be analysed and compared in Section 4.1.1. These activities were recorded using Thingy and RESpeck sensors.

Implementation Details

The following implementation details were consistent across all methodologies:

- **Windowing:** Data was divided into overlapping windows of fixed size to capture temporal dependencies, with the step size controlling the overlap. This approach allowed effective feature extraction for both sequential and non-sequential models. Windows, and therefore the model's input, were of shape $(1, \text{window size}, 3)$, including *only the accelerometer data from the sensors.*
- **Training and Test Sets:** The dataset was split into training (80%), and test sets (20%).
- **Cross-Validation:** A 5-fold cross-validation was implemented for robust evaluation. This ensured that all data points contributed to both training and validation, providing a comprehensive assessment of model performance.
- **Training Configuration:** Early stopping was employed to prevent overfitting by monitoring the validation loss. Learning rate schedulers were used to adjust the learning rate dynamically during training, promoting convergence.

Method 1: 1D Convolutional Neural Networks (CNNs)

The first approach used sequential 1D Convolutional Neural Networks (CNNs) for classifying eleven physical activities, using exclusively RESpeck data. These activities included ascending stairs, descending stairs, running, walking, shuffle walking, lying down (right, left, on stomach), sitting/standing, and miscellaneous.

The CNN comprised multiple convolutional layers, with the number of layers, kernel sizes and dropout rates optimised through experimentation. The best-performing model included two convolutional layers with kernel sizes of 3 and 10, respectively. Batch normalisation was applied after each layer to stabilise and accelerate training. Dropout rates of 0.1 and 0.2 were used after the convolutional layers, with an additional dropout rate of 0.5 before the dense layer to prevent overfitting. The model was trained using 120 epochs with a batch size of 64, a window size of 100, and a step size of 40, and an Adam Optimiser was chosen.

Additionally, a second 1D-CNN was trained specifically on Thingy sensor data to classify between *sitting* and *standing* activities, as RESpeck data is not enough to distinguish between these two activities. This implementation is detailed in `PDIoT_models_4_sitting_standing.ipynb`.

Method 2: Long Short-Term Memory (LSTM) Networks

The second approach involved implementing LSTM networks to classify the same eleven activities, leveraging their ability to capture temporal dependencies in sequential data.

The LSTM model included two LSTM layers with 64 and 32 units, respectively, followed by dropout layers (0.3). A dense output layer with softmax activation was used for classification.

This methodology is implemented in `PDIoT_models_LSTM.ipynb`.

Method 3: Static vs. Dynamic Activity Classification

A hierarchical classification approach was adopted by first categorising activities into *static* (e.g., sitting, standing, lying down right, left, on back and on stomach) and *dynamic* (e.g., walking, running, ascending and descending stairs, shuffle walking and miscellaneous). Separate models were trained for each category.

A binary classifier distinguished static from dynamic activities using a softmax classifier. Two separate models were trained then for static and dynamic activities, each optimised for its specific subset of activities, using a softmax classifier as well.

These implementations are detailed in `PDIoT_models_staticVSdynamic.ipynb`, `PDIoT_models_static.ipynb`, and `PDIoT_models_dynamic.ipynb`.

3.2.2 Social Signs

This section outlines the methodology employed to classify different social signs. These activities were recorded using the RESpeck sensor, which included breathing normally, coughing, hyperventilating, laughing, singing, and talking. For the purpose of data simplification and better results, the latter three activities -laughing, singing, and talking- were joined into a single category named *Other*. Three different methodologies were used for the classification: Support Vector Machines (SVM), Random Forests, and a 1D Convolutional Neural Network (CNN). Each model was evaluated, considering its strengths and weaknesses, in order to find the most accurate, robust, and computationally efficient model.

Method 1: Support Vector Machine

SVMs are a type of supervised learning model known for their robustness in classification tasks. Its objective is to find a hyperplane in an N-dimensional space that accurately classifies the data points into their respective categories, where N corresponds to the number of features. The hyperplane aims to be positioned in such a way that the distance between the nearest point and the hyperplane is maximised.

However, SVMs typically perform effectively in binary classifications and are known for being computationally intensive. As a result, it does not align well with the task of classifying the social signs into four different classes, which contain a large dataset of sensor data.

Method 2: Random Forests

Random forests, or random decision forests, are an ensemble learning technique usually used for classification or regression tasks. This method constructs multiple decision trees during training and then outputs the class, which is the mode of the classes.

Although they are an appropriate technique to avoid overfitting due to the averaging of multiple trees, therefore smoothing the decision boundary, they can prove to be ineffective in the task of social sign classification. Random Forests ignore temporal dependencies in the data, which are crucial in HAR, where sequential patterns are needed.

Method 3: 1D Convolutional Neural Network

As a result, our primary focus was on employing a 1D Convolutional Neural Network, which is capable of processing sequential data and finding its underlying patterns. We experimented with various architectures in order to determine the optimal structure for the social sign classification. The chosen model consisted of

three convolutional layers with increasing filter sizes: starting with 64 filters in the first, 128 in the second and 256 in the third, allowing the network to process features at different scales, capturing both local and global features.

Then, to avoid overfitting and to make a more robust and generalisable model, several regularisation and normalisation techniques were considered in each layer:

- **L2 Regularisation.** Prevents overfitting by adding a penalty term to the loss function, encouraging model weights to be small, leading to more evenly distributed values.
- **Batch Normalisation.** Normalises the inputs of each layer to have a mean of zero and a variance of one, allowing it to stabilise and accelerate the learning process.
- **Dropout.** Helps prevent overfitting by randomly setting a fraction of the input units to zero at each step during training time, therefore dropping out those units from the network. Consequently, it forces the network to learn more robust features.

Following the convolutional and pooling layers, flattening occurs to transform the multidimensional output of the last pooling layer into a one-dimensional array. This is then fed into the Dense layers, where the first one consists of 128 neurons, following one of 64, where both also include the L2 regularisation and batch normalisation strategy to control overfitting.

For optimisation, we chose the *Adam optimiser*, which adjusts the learning rate for each parameter, handling different scales of parameters more efficiently than others that use a single global learning rate. These methods and the model's configuration can form a generalisable model, capable of distinguishing the different types of social signs.

3.3 Algorithms and Methods for Sleep Analysis

To perform sleep analysis, we evaluate two main methods. We calculate the number of positional changes during the night and obtain the sleep-wake cycles of each subject through unsupervised learning to then derive a Sleep Quality Index that aims to agree with the answers in the sleep questionnaire provided.

3.3.1 Number of Positional Changes

This section outlines the methodology used to compute the number of positional changes during sleep, implemented in `positional_changes.ipynb`. The meth-

ods used rely on analysing transitions between predicted activities over successive windows. The key steps are:

- **Windowing**: The data is reshaped into 4-second windows (of shape (1, 100, 3)), where each window represents a segment of sensor data (accelerometer) over time. This is the window size used for training the model that will be used for the activity prediction. There is no overlapping of windows.
- **Activity Prediction**: The machine learning model predicts the human activity for each window. The best-performing model for static activities is selected (Model 2, see Section 4.1.1). Predictions of miscellaneous or dynamic activities are deleted at this step, as these are not valid activities during sleeping.
- **Transition Detection**: The number of positional changes is calculated as the count of activity transitions between consecutive windows.

Positional changes - the transition detection - are calculated by monitoring the prediction output of the activity classification model over time (in each window) and identifying transitions. We consider variations where the predictions differ between consecutive or multiple adjacent windows, depending on the specific method implemented. The methods are detailed below:

Method 1: Simple Positional Changes

This method counts the transitions between successive predictions. For every window i, if the prediction p_i differs from the prediction of the next window p_{i+1}, it is counted as a positional change. Mathematically, this is defined as:

$$\text{Positional Changes} = \sum_{i=1}^{N-1} \mathbb{1}[p_i \neq p_{i+1}],$$

where N is the total number of windows, and $\mathbb{1}$ is the indicator function (returns 1 when the condition in brackets is met and 0 otherwise).

Method 2: Consecutive Window Stability (Two Windows)

This method refines the calculation by requiring that the transition is stable across two consecutive windows to count as a positional change. For a change from p_i to p_{i+1} to be considered valid, p_{i+1} must equal p_{i+2}:

$$\text{Positional Changes} = \sum_{i=1}^{N-2} \mathbb{1}[p_i \neq p_{i+1} \ \wedge \ p_{i+1} = p_{i+2}].$$

Method 3: Consecutive Window Stability (Three Windows)

This method further extends the requirement to three consecutive windows. A change is counted only if the same prediction is observed over three consecutive windows:

$$\text{Positional Changes} = \sum_{i=1}^{N-3} \mathbb{1}[p_i \neq p_{i+1} \ \wedge \ p_{i+1} = p_{i+2} = p_{i+3}].$$

Method 4: Relative Positional Changes

In this method, the relative number of positional changes is computed as a normalised metric. It divides the number of positional changes from Method 1 by the total number of windows N, providing a proportion that reflects the number of transitions relative to the time of sleep:

$$\text{Relative Positional Changes} = \frac{\text{Positional Changes (Method 1)}}{N}.$$

Implementation Details

The aforementioned methods were implemented in Python using a combination of NumPy for array manipulation and pandas for structured data handling. The results are stored and aggregated for further analysis.

Method 1 proved to be the most effective, as it showed more contrasted results between categories (poor, average, and good quality of sleep).

3.3.2 Sleep-wake Cycles

This section elaborates on the methods used for classifying sleep-wake cycles from the data collected via the RESpeck sensor. The data provided consisted of 48 unlabelled sleeping files, and given the lack of labelled data, unsupervised learning methods were tested. Unsupervised learning allows the discovery of patterns in unlabelled data and is used in applications that include clustering, association, and dimensionality reduction. Clustering is a suitable technique for classifying sleep-wake cycles as it can group the data into sleepiness and periods of wakefulness. We focused on three main algorithms: K-Means, Expectation Maximisation and Hierarchical Clustering.

Method 1: K-Means Clustering

This algorithm partitions data into K non-overlapping clusters, minimising the variance within each cluster. For this task, we applied K-means to segment the sensor data into two main clusters, which represent the sleep and wake states. The

algorithm randomly assigns two points as cluster centres in the initialisation step, and each data point is assigned to the nearest centre point based on the Euclidean Distance (3.1). The cluster centres can then be updated as the mean of all the data points assigned to each cluster, allowing for the iteration of the data point assignments and centroid update steps.

$$\text{Distance}(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} \tag{3.1}$$

Method 2: Expectation-Maximisation (EM) Clustering

EM is a probabilistic clustering technique which assigns data points to clusters but also provides a probability of membership to each cluster. It iterates between two main steps:

1. The Expectation step (**E-step**), which calculates the probabilities of each data point belonging to each cluster based on the current parameter estimates.

2. The Maximisation step (**M-step**) then updates the parameters (mean, variances, and mixture weights) to maximise the likelihood of the data.

Nonetheless, EM can be computationally intensive and is prone to falling into local optima, making it necessary to produce multiple runs with different initialisations.

Method 3: Hierarchical Clustering

This algorithm creates a tree of clusters and can fall into two categories: agglomerative and divisive. For the task, we used *Agglomerative Hierarchical Clustering*, where it begins with n clusters, one for each data point. Then, at each step, the closest pairs are merged until the desired cluster structure is achieved.

However, hierarchical clustering can be computationally expensive, especially for large datasets, as the complexity of merging clusters grows significantly with the number of data points.

Model Feature Selection

These unsupervised learning techniques cluster the data points based on different features. The RESpeck sensor obtained acceleration and gyroscope data, offering insights into the subject's physical movements and orientation. The acceleration data measures the intensity and direction of movement, which gives information on the physical activity level and the subject's orientation, while the gyroscope data measures the angular velocity, providing possible additional data about the orientation and rotational movements.

We attempted to derive a new feature through the gyroscope data in order to produce the subject's breathing rate estimates, which tend to become slower during sleep [30]. This was achieved through the following steps:

1. **Bandpass Filtering:** The z-axis of the gyroscope data, which could capture the torso movements associated with breathing, is filtered. This way, the frequency that corresponds to human breathing (around 0.1 to 0.5 Hz [9]) is isolated, therefore reducing the influence of noise.
2. **Peak Detection:** Using the filtered data, we identify peaks which can constitute one complete breathing cycle.
3. **Calculating Breathing Rate:** To determine the breathing rate, we measure the time intervals between peaks, then calculate the average breathing rate in breaths per minute.

This new property, alongside the acceleration and gyroscope data, can be used as features in the models to test whether it provides deeper insights into sleep patterns.

3.3.3 Sleep Quality Index (SQI)

This section outlines the methodology used to compute the Sleep Quality Index (SQI), as implemented in the provided Python code `sleep_quality_index.ipynb`.

The SQI is computed as a weighted combination of relevant metrics. Three methods were used to compute SQI, differing in the variables/metrics included and the approach to calculating positional changes (absolute or relative).

Each method uses an optimisation process to determine the coefficients and thresholds that maximise the performance of the SQI model. The thresholds represent the specific values of the SQI that categorise sleep quality as poor, average, or good, such that:

$$SQ = \begin{cases} \text{poor} & \text{if } SQI < \text{threshold1}, \\ \text{good} & \text{if } SQI > \text{threshold2}, \\ \text{average} & \text{otherwise.} \end{cases}$$

Method 1: Absolute Positional Changes and Sleep Efficiency

This method calculates SQI using the absolute number of positional changes and sleep efficiency (percentage of time sleeping with respect to the total length of the recording, according to the sleep diaries). The formula is:

$$\text{SQI} = A \cdot \text{Positional Changes} + B \cdot \text{Sleep Efficiency}$$

Method 2: Absolute Positional Changes, Sleep Efficiency, and Number of Awakenings

This method incorporates the number of awakenings calculated from the sleep-wake analysis (Method 2: EM) into the SQI:

$$\text{SQI} = A \cdot \text{Positional Changes} + B \cdot \text{Sleep Efficiency} + C \cdot \text{Number of Awakenings}$$

Method 3: Relative Positional Changes and Sleep Efficiency

This method replaces absolute positional changes with relative positional changes (normalised by the duration of the sleep session):

$$\text{SQI} = A \cdot \text{Relative Positional Changes} + B \cdot \text{Sleep Efficiency}$$

Method 4: Positional Changes with Consecutive Windows and Sleep Efficiency

This method mimics Method 1, but computing the number of positional changes with the requirement of having the same change of activity in two consecutive windows (Method 2 from positional changes, see Section 3.3.1).

$$\text{SQI} = A \cdot \text{Positional Changes (Method 2)} + B \cdot \text{Sleep Efficiency}$$

Optimisation Process

An iterative optimisation process is implemented to optimise the coefficients A, B, and/or C, as well as thresholds for mapping SQI to sleep quality categories (poor, average, good). At each iteration, coefficients were optimised for given thresholds, and then thresholds were optimised, fixing the computed optimal coefficients.

The data from all sleeping files and their classification in the sleeping diaries were used for the training of the parameters. The key steps are:

1. **Initialisation**: Set initial threshold values for the classification categories. 20 and 70 were selected through trial and error.
2. **Coefficient Optimisation**: Use the `differential_evolution` algorithm to optimise coefficients such that the F1-score for sleep quality classification is maximised.
3. **Threshold Optimisation**: For the optimised coefficients, determine the thresholds that best separate the categories (poor, average, good) also using the `differential_evolution` algorithm based on the F1-score of the classification.

4. **Iteration**: Repeat the coefficient and threshold optimisation process for 10 iterations, refining the values in each step.

5. **Final Selection**: Among all solutions achieving the maximum F1-score, select the median coefficients and thresholds as the final optimal values.

Justification of Optimisation Method

The optimisation problem for determining the coefficients (A, B, C) and thresholds for the Sleep Quality Index (SQI) has a non-linear, non-convex nature, and the presence of multiple solutions to achieve the maximum F1 score makes it challenging for standard optimisation techniques to find a minimum. Common algorithms such as BFGS or quadratic programming rely on assumptions like smoothness and convexity, often resulting in convergence to suboptimal local solutions. **Differential Evolution**, on the other hand, proved to be highly effective due to its population-based approach, which explores multiple regions of the solution space simultaneously, and its stochastic sampling, which helps avoid getting stuck in local optima. Its ability to handle non-smooth and discontinuous objective functions made it well-suited to this problem, reliably identifying globally optimal solutions in this complex landscape.

Implementation Details

The optimisation process is implemented in Python, leveraging libraries such as NumPy for array handling, pandas for data organisation, and SciPy for optimisation. The key implementation details include:

- **Optimisation Algorithm**: The `differential_evolution` algorithm is configured with bounds $[-1, 1]$ for A, B, C, and $[0, 100]$ for thresholds, ensuring valid ranges for all parameters.

- **Performance Metric**: The F1-score is used as the optimisation criterion to balance precision and recall in sleep quality classification.

- **Iterations**: The process runs for a fixed number of iterations (10) to ensure stability and refinement of the optimal values.

3.4 Mobile Application Implementation

3.4.1 Tools and Technologies Used

The app is built using Android Studio, which serves as the primary Integrated Development Environment (IDE) for designing, coding and testing. The code is

written in Kotlin, and the project targets devices running Android API level 23 and above, ensuring compatibility with a wide range of Android devices.

The app integrates *Room Database* for efficient local data storage and management, allowing users to store and retrieve activity history. To incorporate and make use of the machine learning models, the app uses *TensorFlow Lite*. Additional relevant libraries include *MPAdnroidChart* for creating visualisations of activity data and *NV-Bluetooth* for facilitating communication with external sensors.

3.4.2 Functions and Features of the App

The app's intuitive layout is designed to align with Nielsen's heuristics [20] from Human-Computer Interaction (HCI), ensuring that individuals with varying technical expertise can navigate and use it effectively. By prioritising principles such as simplicity, visibility and feedback, the app becomes a practical tool for users. This design approach aims to combine advanced technology with everyday usability, making the app accessible and beneficial to a wide audience.

Home Menu

The home menu of the application presents a clean and user-friendly structure that provides quick access to its core functionalities. Each button is paired with an icon to enhance clarity and usability. As shown in Figure 3.2, the menu includes four key options:

- Record data: This feature allows users to gather activity data by connecting to the sensors, specifying the type of activity performed, and selecting the sensor in use.
- Watch live processing: This option enables real-time visualisation of the activity recognition process, offering immediate feedback on the activity being performed along with the display of sensor data graphs. This is essential for building the dataset that users can later access in the 'History' section.
- Connect sensors: This functionality is key for establishing communication between the app and external sensors, Thingy and Respeck, in our case, ensuring seamless data transmission.
- History: This section provides access to previously recorded data, allowing users to review and analyse past activity sessions.

Record Data

This section of the app has the objective of collecting activity data. As illustrated in Figure 3.3, users are required to specify the sensor type, the activity currently

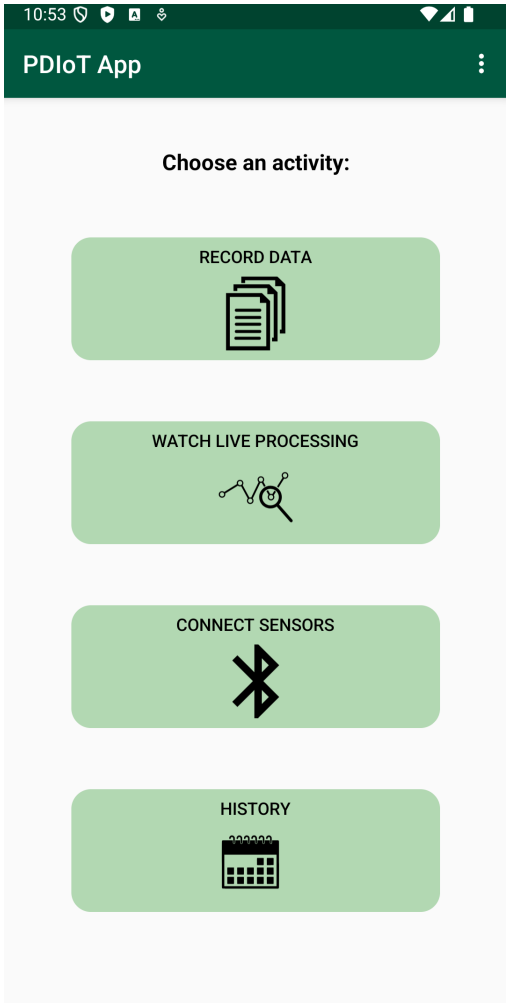


Figure 3.2: Home menu of the Android app

being performed, and the corresponding respiratory condition.

Additionally, it provides real-time sensor data, including accelerometer and gyroscope readings, offering immediate feedback and ensuring proper setup.

Watch Live Processing

As observed in Figure 3.4, this feature of the Android app offers real-time monitoring of the user's current activity and respiratory condition. To enhance readability, the interface includes two icons that visually represent the detected activity and respiratory condition, making the information easily understandable for users. The system's predictions are based on data from the connected sensor(s) at any given time. When both the Thingy and RESpeck sensors are connected, all activities

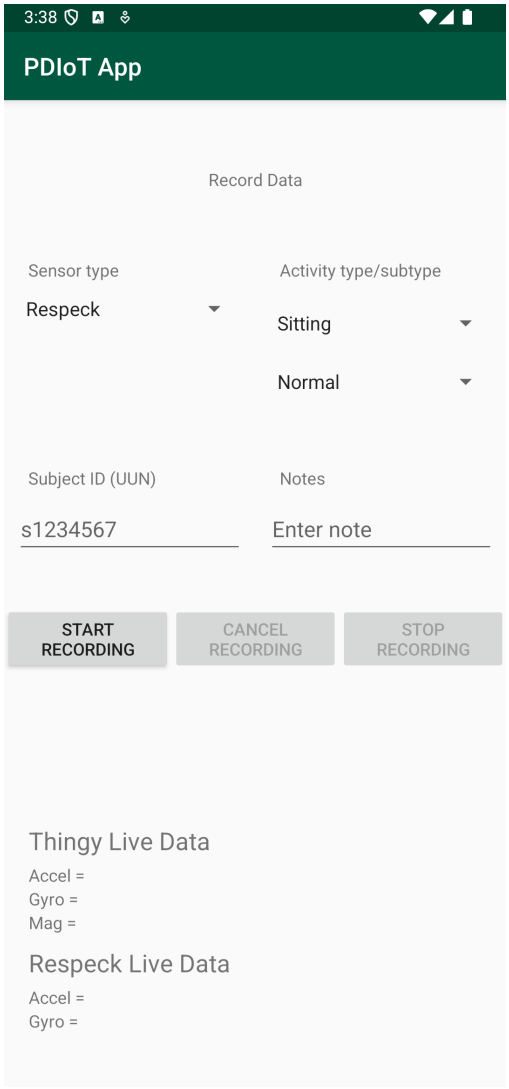


Figure 3.3: 'Record Data' tab

will be identified by the RESpeck except for sitting and standing, as it cannot distinguish between these two states. In such cases, the app utilises data from the Thingy sensor to accurately differentiate between the two and provide a more precise prediction.

Below the activity and respiratory condition indicators, the tab displays real-time sensor data in the form of graphs. The RESpeck Live Data section plots acceleration data across the X, Y and Z axes, offering a visual representation of the sensor's activity. Similarly, the Thingy Live Data section provides corresponding outputs from the Thingy sensor, enabling users to observe dynamic changes in

sensor readings and compare data across devices.

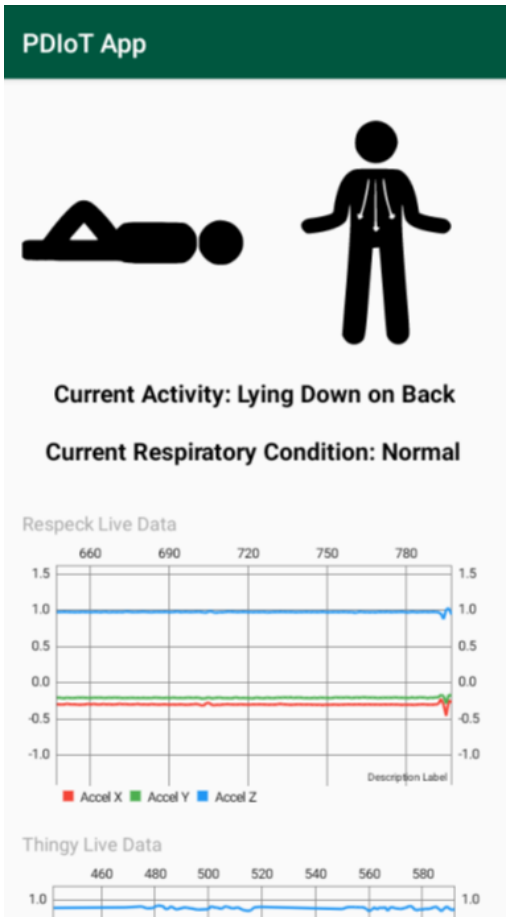


Figure 3.4: 'Watch Live Processing' tab

History

The tab of the app shown in Figure 3.5, referred to as the 'History' feature, allows users to view previously recorded activity data. Using the integrated calendar at the top of the screen, users can select a specific date to access data logged on that day. If recordings exist for the selected date, the app will display the corresponding information.

The first visualisation is a bar graph, which provides an overview of all recorded activities and respiratory conditions. Each bar represents a specific activity or condition, and its value corresponds to the total duration spent on that activity, offering a quick overview of the day's activity distribution.

If the user keeps scrolling, the specific times will be displayed in the form of text,

differentiating between the activities and respiratory conditions, along with the exact minutes and seconds for each activity. Such a level of detail allows users to track daily routines or monitor health-related metrics.

By combining visual summaries with more detailed logs, the app provides an intuitive way to review past activity data for users seeking both quick overviews and in-depth analysis.

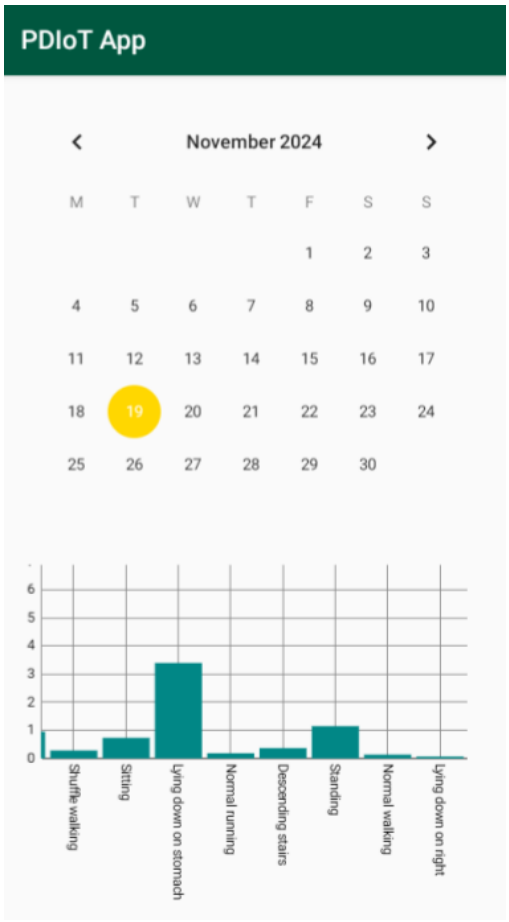


Figure 3.5: 'History' tab.

3.5 Software Organisation

The software organisation of our implementation can be observed in Figure 3.6. The PDIoT App code is organised within the main application directory in pdiotapp/app/src/main/java/com/specknet/pdiotapp, where LiveDataActivity contains most of the core logic and interacts with the machine learning models stored in the assets folder. The history directory implements features for managing and

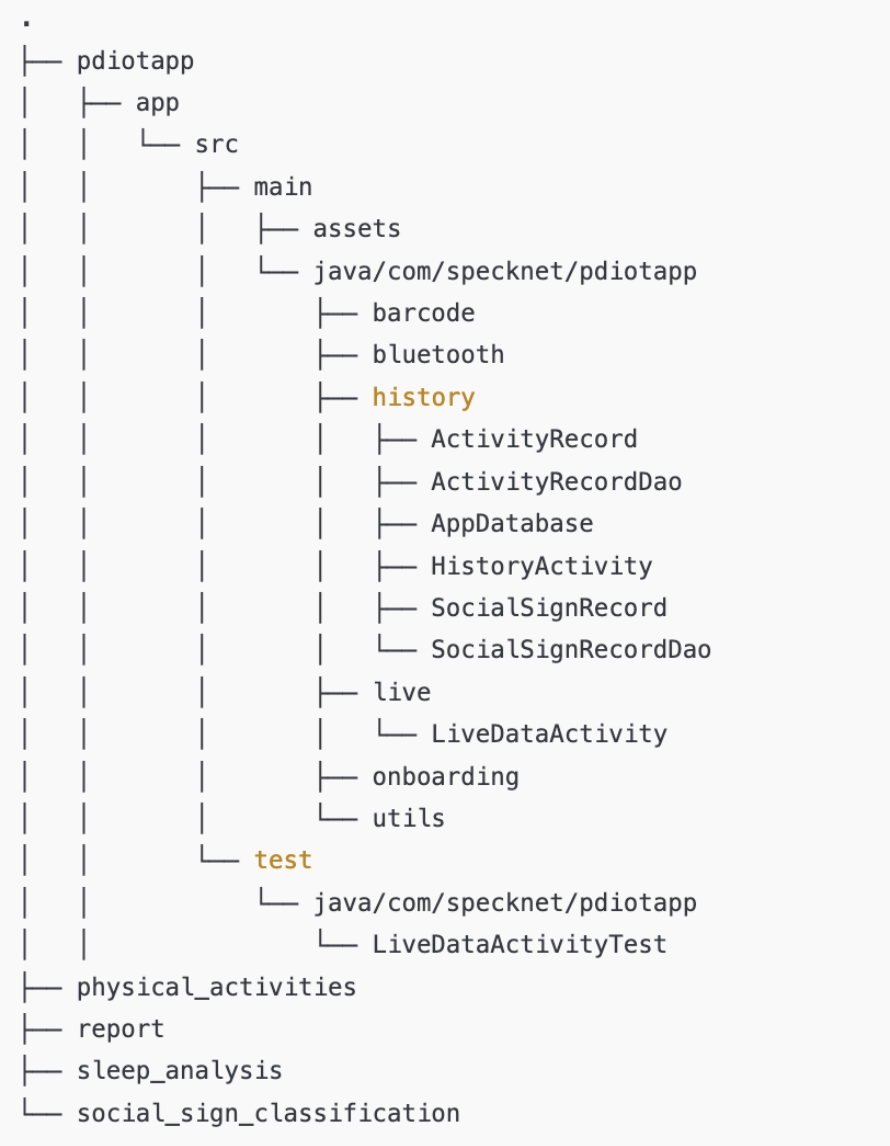


Figure 3.6: Software Directory Organisation of our implementation.

displaying user activity history, while unit tests are located in the LiveDataActivityTest file under test. Additional directories like physical_activities, sleep_analysis, and social_sign_classification contain Python code used for training machine learning models, generating analysis, and creating graphs to support the app's functionality and research.

3.6 Testing

Unit tests were developed and executed for key application functionalities, with a focus on the LiveDataActivity class, which contains the core logic of the app. These tests, implemented in the LiveDataActivityTest class, evaluate the activity's interactions with the machine learning models, database operations, and UI responsiveness under various scenarios.

The tests specifically validate:

- Correct integration and functionality of the machine learning models stored in the assets folder.
- Accurate data handling and management within the history directory, particularly the ActivityRecord, SocialSignRecord, and associated DAOs (Data Access Objects).
- Stability and reliability of key user interactions, including live activity detection.

Chapter 4

Results and Discussion

4.1 Accuracy Results of Machine Learning Models

We present the results from the models described in the previous chapter, evaluating different architectures and comparing the best models.

4.1.1 Physical Activity Classification

This section presents the validation accuracy results for each of the three methods evaluated for activity classification. Each model's architecture is thoroughly described, including the number of layers, window size, step size, kernel sizes, regularisation techniques, dropout rates, batch size, and number of epochs. The results are presented for both dynamic and static activities, followed by an analysis of the best-performing model and its class-wise validation accuracy across folds.

Method 1 (1D CNN): Summary of Models and Performance

Table 4.1 summarises the key characteristics and performance metrics of the five models:

Model	Layers	Window Size	Step Size	Kernel Sizes	Dropout Rates	Epochs	Batch Size	Avg. Accuracy (Static/Dynamic)
Model 1	2	80	40	(3, 15)	0.2, 0.2, 0.5	110	64	96.37% / 90.13%
Model 2	2	100	40	(3, 10)	0.1, 0.2, 0.5	120	64	97.35% / 93.22%
Model 3	1	50	15	(10)	0.2, 0.5	120	64	95.72% / 89.17%
Model 4	2	50	20	(3, 10)	0.1, 0.2, 0.5	50	64	94.79% / 90.48%
Model 5	4	100	50	(3, 3, 3, 3)	0.2, 0.2, 0.1, 0.5	100	64	94.24% / 92.27%

Table 4.1: Model architecture, training parameters, and average validation accuracy across folds for static and dynamic activities.

Model	Precision	Recall	F1 Score	Avg. Test Accuracy (Static/Dynamic)
Model 1	73.57%	72.19%	72.28%	73.95% / 71.33%
Model 2	82.93%	81.83%	81.53%	85.79% / 77.00%
Model 3	73.47%	71.97%	71.79%	78.00% / 66.13%
Model 4	75.87%	72.41%	72.31%	75.01% / 70.85%
Model 5	74.35%	72.25%	72.18%	76.58% / 68.13%

Table 4.2: Test set performance metrics for precision, recall, F1 score, and average test accuracy across static and dynamic activities for all models.

Model 1 (`PDIoT_models_2.ipynb`)**:** This model included two convolutional layers with kernel sizes of 3 and 15, respectively, capturing features at different scales. Batch normalisation was applied after each layer for faster convergence and better generalisation. Regularisation was implemented through dropout rates of 0.2 for the first two layers and 0.5 before the dense layer. The model was trained using 110 epochs with a batch size of 64, a window size of 80, and a step size of 40.

Model 2 (`PDIoT_models_3.ipynb`)**:** This model included two convolutional layers with kernel sizes of 3 and 10, respectively. Batch normalisation was applied after each layer to stabilise and accelerate training. Dropout rates of 0.1 and 0.2 were used after the convolutional layers, with an additional dropout rate of 0.5 before the dense layer to prevent overfitting. The model was trained using 120 epochs with a batch size of 64, a window size of 100, and a step size of 40.

Model 3 (`PDIoT_models_8.ipynb`)**:** This model included one convolutional layer with a kernel size of 10, followed by a pooling layer to reduce dimensionality. Batch normalisation was applied after the convolutional layer to stabilise and accelerate training. Regularisation was implemented through dropout rates of 0.2 and 0.5 for the convolutional and dense layers, respectively. The model was trained using 120 epochs with a batch size of 64, a window size of 50, and a step size of 15.

Model 4 (`PDIoT_models_7.ipynb`)**:** This model included two convolutional layers with kernel sizes of 3 and 10, followed by pooling layers to reduce dimensionality. Batch normalisation was applied after each layer to stabilise and improve training efficiency. Dropout rates of 0.1 and 0.2 were used after the convolutional layers, with an additional dropout rate of 0.5 before the dense layer to prevent overfitting. The model was trained using 50 epochs with a batch size of 64, a window size of 50, and a step size of 20.

Model 5 (`PDIoT_models.ipynb`)**:** This model included four convolutional layers with kernel sizes of 3, 3, 3, and 3, extracting hierarchical features at consistent scales. Batch normalisation was applied after each layer to stabilise and improve training efficiency. Dropout rates of 0.2, 0.2, 0.1, and 0.5 were used to prevent overfitting. The model was trained using 100 epochs with a batch size of 64, a window size of 100, and a step size of 50.

Class-Wise Validation and Test Accuracy of Best-Performing Model

The best-performing model was **Model 2**, which achieved the highest average validation accuracy for both static (97.35%) and dynamic (93.22%) activities. The

class-wise validation accuracy and mean test accuracy for this model are presented in Table 4.3.

Activity Class	**Mean Validation Accuracy (%)**	**Mean Test Accuracy (%)**
Ascending Stairs	93.72	75.63
Descending Stairs	92.02	85.00
Lying Back	100.00	85.83
Lying Left	96.07	76.47
Lying Right	96.67	66.67
Lying Stomach	95.01	100.00
Miscellaneous	80.73	70.83
Normal Walking	98.47	47.06
Running	100.00	100.00
Shuffle Walking	94.40	83.50
Sitting/Standing	99.02	100.00

Table 4.3: Class-wise mean validation accuracy and mean test accuracy for Model 2.

Sitting/Standing Model with Thingy data

The Thingy sensor model for sitting/standing classification uses a single convolutional layer (kernel size 3) with max pooling, dropout (0.2, 0.5), and batch normalisation. It achieved 100% accuracy in both validation and test sets for distinguishing sitting and standing, showcasing exceptional robustness and generalisation.

Analysis

The analysis of the models highlights some clear trends regarding the factors that influence performance. Models with varying kernel sizes, such as Model 2, performed the best as they captured both fine-grained and broad temporal patterns, achieving high validation accuracies for static (97.35%) and dynamic (93.22%) activities. However, excessive depth or overly large kernel sizes, as seen in Model

1 (kernel size 15) and Model 5 (four layers with kernel size 3), led to overfitting, reflected in considerably lower test accuracies. Models with a single convolutional layer, such as Model 3, lacked the feature extraction capacity needed to generalise, particularly for dynamic activities, failing to achieve the required 90% validation accuracy.

Window size also played an essential role in performance. Comparing Models 2 and 4, which shared similar kernel sizes and dropout rates, the larger window size of Model 2 (100) outperformed the smaller window size of Model 4 (50) in validation accuracy, suggesting that a larger temporal context can improve pattern recognition. However, all models struggled with overfitting, as evidenced by lower test accuracies across the board. Despite this, Model 2 demonstrated the best generalisation and the smallest validation-test accuracy gap, making it the most robust model overall. Class-wise analysis showed that static activities like sitting/standing and lying positions consistently performed better than dynamic activities like normal walking and ascending/descending stairs, which are inherently more complex and varied.

Method 2 (LSTM)

The LSTM model, despite using similar metrics as the CNN models, performed significantly worse, particularly in dynamic activities. It achieved a mean validation accuracy of 87.23% for static activities and only 36.26% for dynamic activities, with test set precision, recall, and F1 scores of 41.18%, 43.94%, and 41.62%, respectively. Due to these poor results and inability to generalise, further testing with LSTM was discontinued.

Method 3 (Hierarchical Classification of Static and Dynamic Activities)

The binary model that classified static vs. dynamic activities achieved a validation accuracy of 99.95% and 97.97% for static and dynamic classes, respectively. The model that classified static activities achieved a validation average accuracy of 99.78%; multiplied by the accuracy of static activities in the binary model, this results in an overall mean validation accuracy of $99.95 \cdot 99.78 = 99.73\%$ for static classes. The dynamic model achieved a mean validation accuracy of 89.41%; this produced an overall mean validation dynamic accuracy of $97.97 \cdot 89.41 = 87.59\%$. This was not enough for dynamic classes, and thus, we decided to stick with Method 1.

4.1.2 Social Sign Classification

In this section, we analyse the performance of multiple machine learning models applied to the classification of social signs. The initial models tested were Support

Vector Machines (SVMs) and Random Forests. In Table 4.4, their performance is summarised, demonstrating low overall accuracies, which aligns with the disadvantages described in Section 3.2.2.

Model	**Overall accuracy**	**Precision**	**Recall**	**F1-Score**
Random Forest	50.51%	0.44	0.51	0.36
SVM	52.69%	0.28	0.53	0.36

Table 4.4: Performance summary for Random Forest and SVM models.

As a result, our focus shifted towards 1D Convolutional Neural Networks, which are more effective when handling sequential data. We experimented with different CNN architectures to explore the impact of varying the number of convolutional layers, filter sizes or dropout rates, as well as different regularisation techniques. The performance of these architectures in the test set is summarised in Table 4.5. It is notable that the architectures incorporating repetitive blocks of convolutional layers obtained accuracies of over 60%, outperforming the SVM and Random Forest models. However, it is observed that using higher Dropout rates, such as 0.5 in CNN Model #3, slightly reduced the accuracy. This suggests that even though it is effective at mitigating overfitting, it can also lead to the loss of important data.

Table 4.5: Performance of different CNN architectures on the test set.

CNN	**Details**	**Accuracy**	**Precision**	**Recall**	**F1-Score**
#1	Two Conv1D layers with 64 and 128 filters, each followed by Batch Normalisation and MaxPooling1D. Fully connected Dense layer with 128 units, followed by Dropout (0.4). Output layer with softmax activation. Adam optimiser with lr=0.0001.	63.98%	0.6346	0.6398	0.6314
#2	Three blocks of Conv1D with 64 filters each, followed by MaxPooling1D and Dropout (0.1). Fully connected Dense layer with 64 units, followed by another Dense output layer with softmax activation. Adam optimiser with lr=0.001 and decay.	65.44%	0.6709	0.6544	0.6544
#3	Three Conv1D layers, all with 64 filters, each followed by Batch Normalisation and MaxPooling1D, and higher Dropout (0.5). Fully connected Dense layer with 128 units, followed by Batch Normalisation and higher Dropout (0.5). Adam optimiser with lr=0.001.	61.95%	0.6111	0.6195	0.6077
#4	Three blocks of Conv1D with 64 filters each, followed by MaxPooling1D and Dropout (0.1). Fully connected Dense layer with 64 units, followed by Dense output layer with softmax activation. Adam optimiser with lr=0.001.	63.73%	0.6499	0.6373	0.6353

Consequently, we refined the CNN architecture and produced the final model, which indicated improved performance metrics, displayed in Table 4.6 and described in Section 3.2.2. To validate it, K-Fold (K=5) cross-validation was used, obtaining a more reliable estimate of the model's performance than using a single train/test split. The dataset was, therefore, divided into five folds, with each being used only once as the validation set, while the remaining were used for training. This is especially beneficial when datasets are not large enough, ensuring that all data has the chance of appearing in both training and validation sets. The mean validation accuracy across all folds is displayed in the table, achieving over 80% in all classes. Figure 4.1 displays the mean training and validation accuracies in each fold, indicating consistent improvements as the fold number increases.

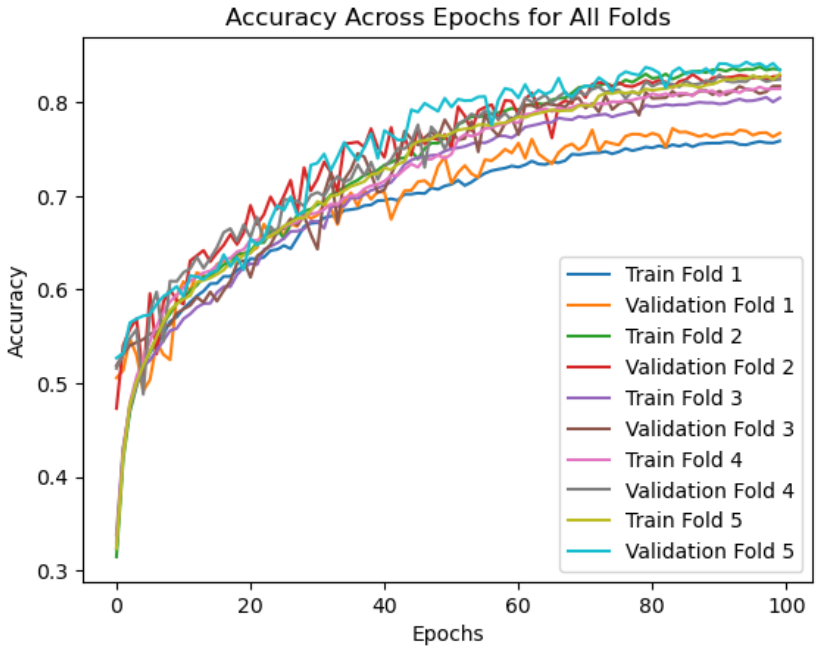


Figure 4.1: Mean training and validation accuracies across the five folds during K-Means Cross-Validation.

Test Set Accuracy	Precision	Recall	F1
75.59%	0.77	0.76	0.76

Table 4.6: Performance metrics for the best CNN architecture for social sign classification in the test set.

Table 4.7 presents the validation accuracy obtained in the last fold. It shows the model classifies with high accuracy the signs of *Coughing*, *Breathing Normally*, and

Class	Accuracy
Breathing Normally	91.53%
Coughing	95.04%
Hyperventilation	93.39%
Other	82.93%

Table 4.7: Mean class validation accuracy across all folds.

Hyperventilation, while the *Other* class had a lower performance, not exceeding 90% accuracy. This is expected as it encompasses a variety of activities, making it more challenging to identify a defining pattern in the data. The confusion matrix for the test set, shown in Figure 4.2, confirms this. Even though it shows a higher count of True Positives for the *Other* category, it is due to its overrepresentation in comparison to the remaining classes. This affects the results, producing misclassifications mostly into *Other*, suggesting that aggregating multiple activities into one can be problematic.

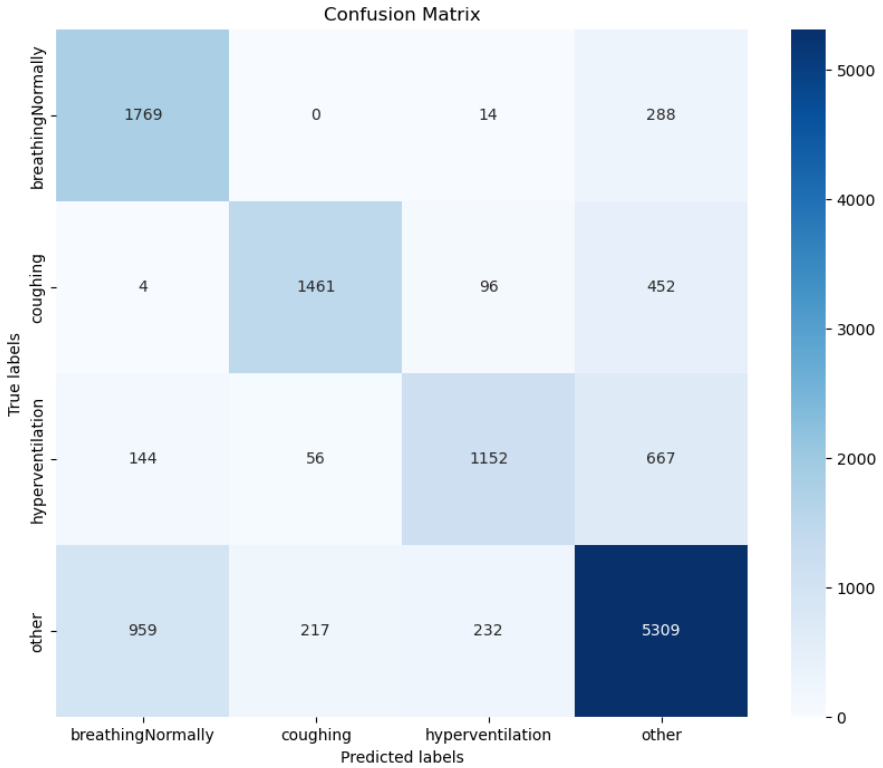


Figure 4.2: Confusion Matrix for the test set.

As a result, we experimented with training the model using all activities without

grouping them into an *Other* category to determine whether it could achieve better results. However, this approach resulted in a decrease in the mean training accuracy to 55.84% and mean validation accuracy to 54.81%. This could be attributed to the increased complexity and variability of the activities.

4.2 Accuracy of Real-Time Classification

To perform real-time classification, each group member performed each activity ten times and monitored the output displayed on the application. For each activity, a five-second interval was allowed to ensure the app had sufficient time to update. In Table 4.8, the real-time classification accuracy is shown for physical activities. Most activities were classified correctly except for walking, which was, on occasions, confused with shuffle walking. For the different social signs, the real-time accuracy dropped, as shown in Table 4.9, especially for the *Other* category, where other signs were grouped together.

Activity Type	**Accuracy**
Walking	90%
Shuffle Walking	100%
Running	100%
Ascending Stairs	100%
Descending Stairs	100%
Miscellaneous	100%
Lying Down Right	100%
Lying Down Left	100%
Lying Down Back	100%
Lying Down on Stomach	100%
Sitting/Standing	100%

Table 4.8: Real-time classification accuracy for physical activities.

Social Sign	**Accuracy**
Breathing Normally	100%
Coughing	83.33%
Hyperventilating	80%
Other	73.33%

Table 4.9: Real-time classification accuracy for social signs.

4.3 Results of Sleep Analysis

This section discusses the outcomes of the number of positional changes of the subjects and the sleep-wake cycle predictions to then create a Sleep Quality Index that can be compared with the questionnaire-based answers.

4.3.1 Number of Positional Changes

This section analyses the results for the number of positional changes computed using the methods described in Section 3.3.1. Methods 2 and 3 yielded identical results; thus, the analysis focuses on Method 2 to avoid redundancy. The results are visualised in Figures 4.3, 4.4, and 4.5.

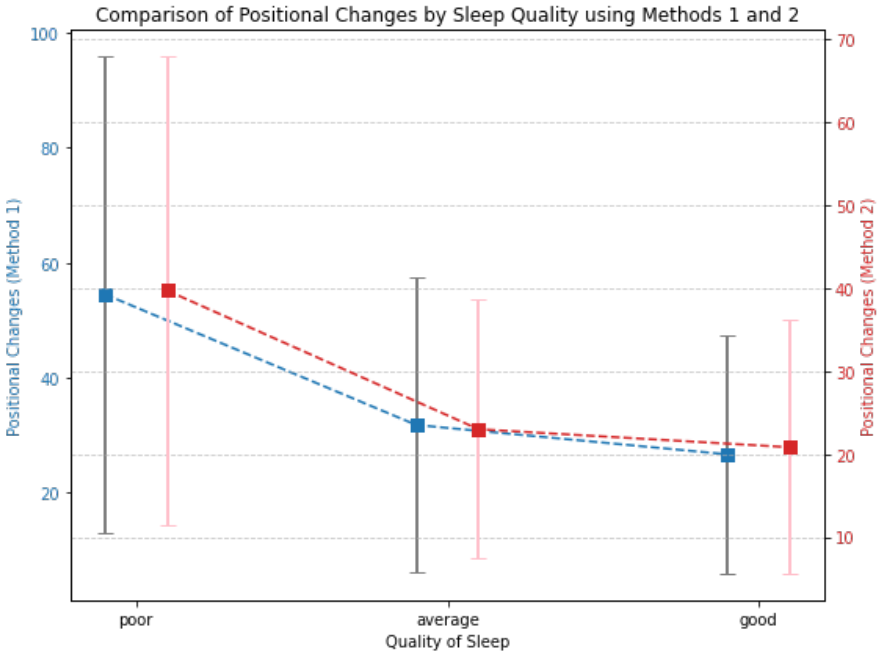


Figure 4.3: Mean and standard deviation of absolute positional changes for Methods 1 and 2 across sleep quality categories.

Figure 4.3 illustrates the mean and standard deviation of the number of positional changes across the sleep quality categories (poor, average, good) for Methods 1 and 2. Both methods show a clear negative correlation between the number of positional changes and sleep quality, with fewer positional changes corresponding to better sleep quality. Method 2 (red curve), which incorporates consecutive window stability, is supposed to reduce noise and variability in the data. The figure shows that Method 2 slightly decreases the distinction between the average and good categories. This might imply that Method 2 may not be as effective in

classifying sleep quality, as the difference between the average and good categories is slightly less pronounced, but the fact that it yields the same results as Method 3 might indicate that it is more stable and robust.

Figure 4.4 compares the absolute number of positional changes (Method 1) with the relative metric (Method 4), which normalises positional changes by the duration of the sleep session (the number of windows in the sleeping data). While normalisation accounts for sleep length differences, it reduces the contrast between categories, particularly between poor and average sleep quality compared to the absolute metric. Additionally, if we observe the standard deviation values, we can see that the good category exceeds the limits of the average category. This suggests that Method 4 might have a worse performance in sleep quality classification. This behaviour is counterintuitive, as it is expected that 10 positional changes in a short nap should not have the same significance as in a full night's sleep. However, in practice, the relative metric diluted the differences between categories, suggesting it is less effective than the absolute metric.

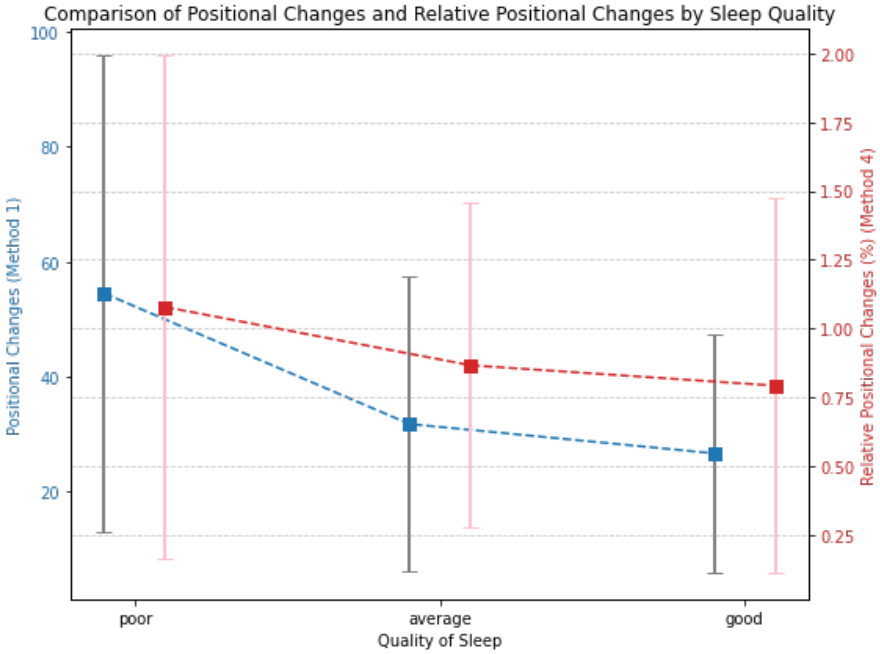
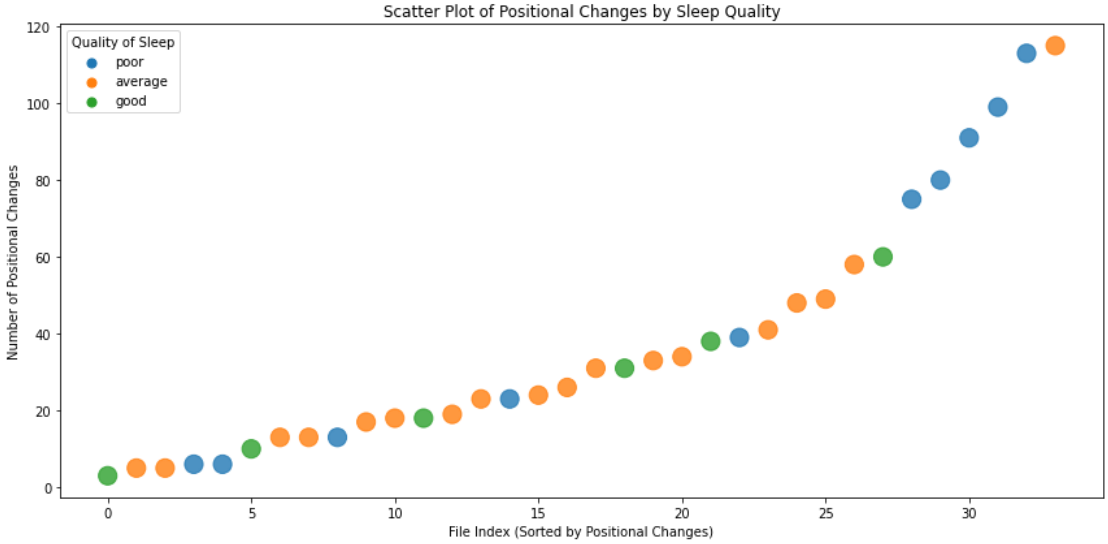
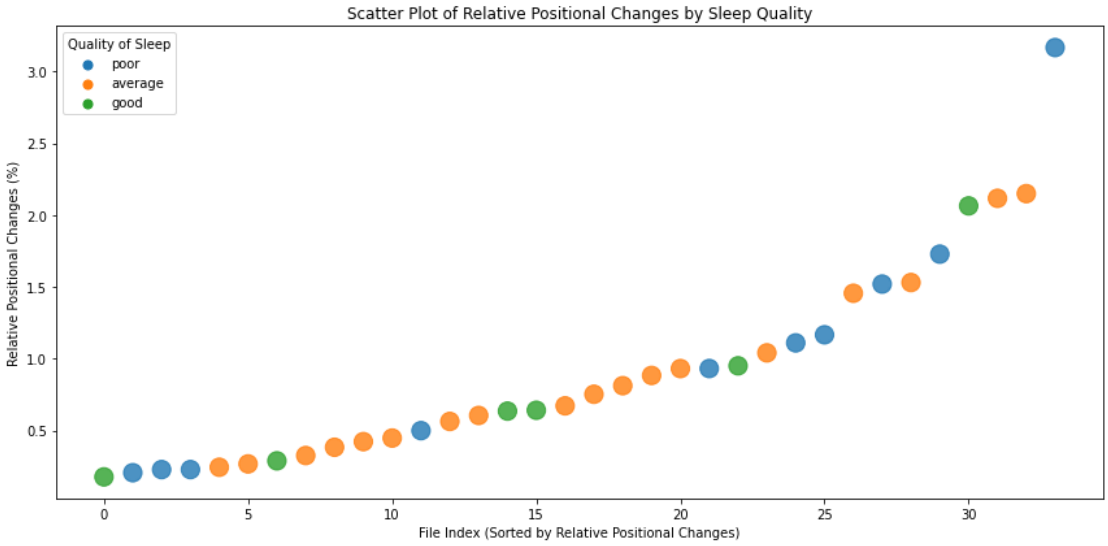


Figure 4.4: Comparison of absolute positional changes (Method 1) and relative positional changes (Method 4) across sleep quality categories.

To further support this observation, Figure 4.5 presents scatter plots comparing absolute (Figure 4.5a) and relative (Figure 4.5b) positional changes in relation to sleep quality categories. In Figure 4.5a, there is a clearer distinction between poor, average, and good sleep quality. Poor sleep classifications tend to accumulate with



(a) Absolute positional changes.



(b) Relative positional changes.

Figure 4.5: Scatter plots comparing absolute and relative positional changes across sleep quality categories.

higher numbers of positional changes, while average quality predominantly appears with mid-range values of changes. Good sleep classifications, on the other hand, start with the lowest number of positional changes and gradually mix with average

classifications toward the middle range.

Conversely, Figure 4.5b demonstrates more overlap among the categories, especially between poor and average sleep quality. In this figure, poor quality does not dominate the higher numbers of relative positional changes, leading to a more generalised overlap compared to the graph showing absolute positional changes. This reinforces the conclusion that absolute metrics provide a stronger signal for distinguishing between sleep quality categories than relative metrics.

Key Observations

- **Consistency Across Methods**: All methods show a negative correlation between the number of positional changes and sleep quality, confirming fewer changes correspond to better sleep.
- **Method 1** demonstrates the clearest separation between sleep categories, making it the most effective for standalone analysis.
- **Method 2** incorporates noise reduction by requiring consecutive window stability. It produces the same values as Method 3, suggesting that this metric is more stable and reliable than Method 1. However, it appears to show a slightly less clear separation between categories.
- **Method 4**, the relative metric. While theoretically appealing for normalising sleep length, it fails to separate categories as effectively as the absolute metric. It introduces overlap, particularly between poor and average sleep quality.

In conclusion, absolute positional changes seem to provide the clearest differentiation between sleep quality categories, as supported by Figures 4.3 and 4.5a. While Method 4 introduces normalisation, it seems to be less indicative of category classification. Overall, every method confirms a strong correlation between the number of positional changes and sleep quality.

4.3.2 Sleep-wake Cycles

In this section, we analyse the results of the different methods described in Section 3.3.2, comparing them to the questionnaire-based answers provided for each file in order to assess their accuracy.

For the method involving K-Means Clustering, different sets of features were used to test their importance. When using all acceleration and gyroscope data, we obtained, as an example, the results displayed in Figure 4.6 a). This concurs with the sleep questionnaire belonging to ID 13, where the sleeping and waking-up times are similar. The subject categorised their sleep quality as poor, which

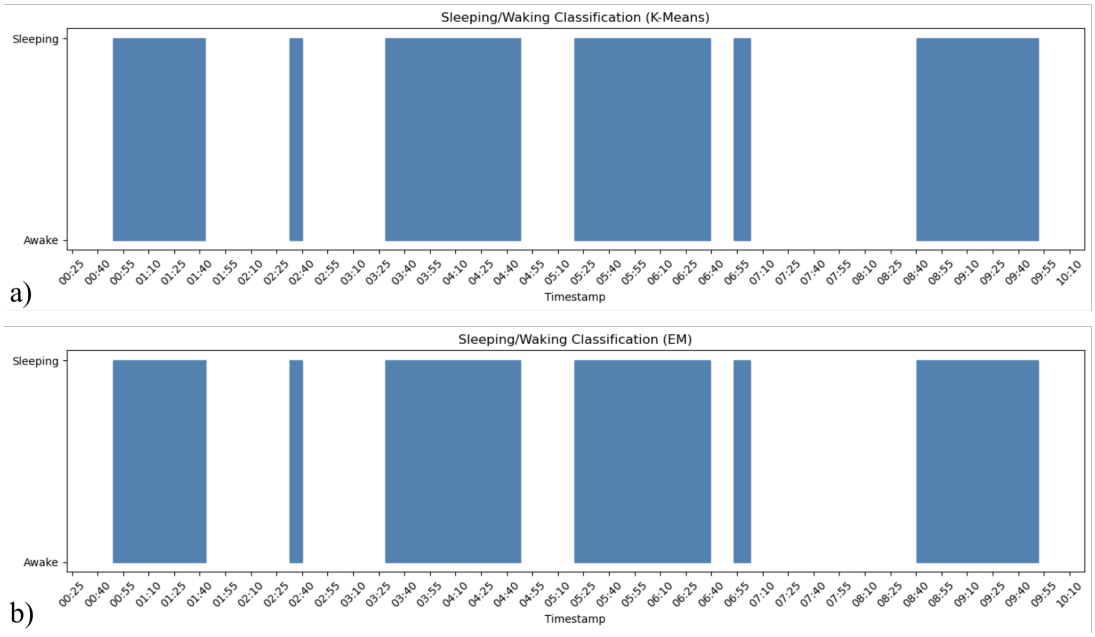


Figure 4.6: Sleep-wake cycles predicted with a) K-Means Clustering and b) EM Clustering.

could explain why the sleep was so fragmented overnight. Using acceleration data alone produced the same results, while using gyroscope data categorised the whole data as being asleep, as displayed in Figure 4.7. Even though we expected the new breathing rate feature to bring better classification outcomes, it produced the same sleep-wake cycles as in Figure 4.6 a). Therefore, we decided not to pursue the idea further, as the operations were computationally expensive.

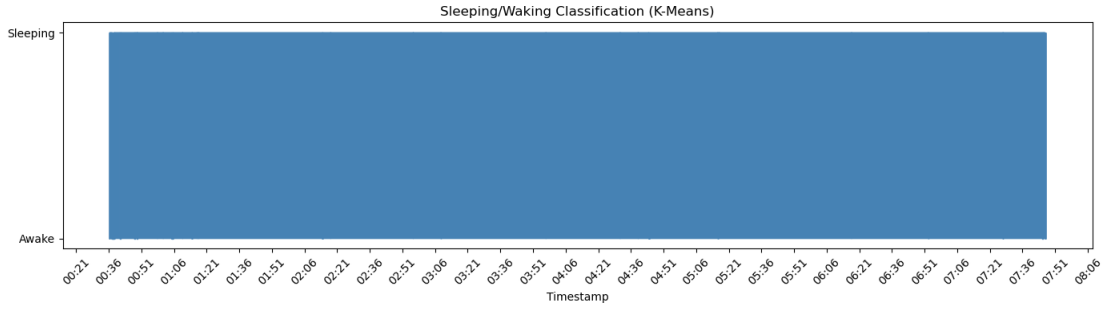


Figure 4.7: K-Means using only gyroscope data

EM Clustering produced the same sleep-wake cycles as the K-Means algorithms in 25 out of the 48 files, an example displayed in Figure 4.6 b). The remaining

files contained similar results in general and comparable waking up and sleeping times. It is not possible to prove whether the sleep-awakenings in the middle are realistic since it is not found in the questionnaire answers.

Hierarchical Clustering, however, produced different results from the EM and K-Means techniques, resulting in more fragmented sleep results, as shown in the example in Figure 4.8, which does not align with reality. This could indicate limitations on this algorithm when modelling sleep patterns.

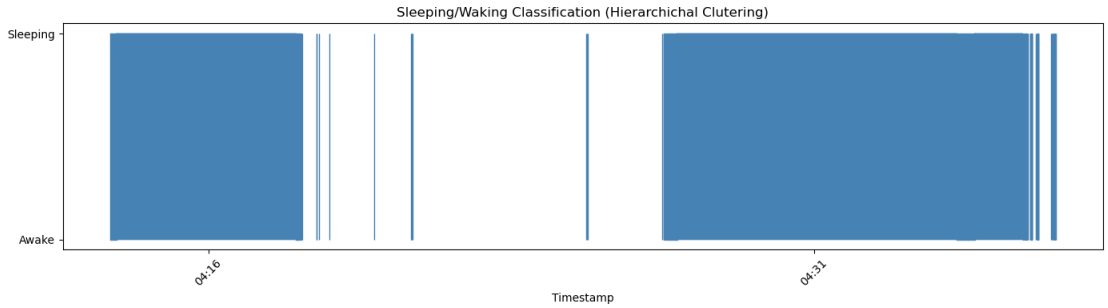


Figure 4.8: Hierarchical Clustering example.

Since both EM and K-Means resulted in similar outputs, it could mean that the essential sleeping patterns are being captured with certain reliability. However, unsupervised learning cannot guarantee accurate results, and since the short sleep interruptions cannot be tested against the sleep questionnaires provided, it is difficult to determine whether the models have predicted precise classifications.

4.3.3 Sleep Quality Index (SQI)

This section presents the results for the Sleep Quality Index (SQI) computed using four methods, as described in Section 3.3.3. These methods are evaluated based on their classification performance for sleep quality categories (poor, average, good). Key performance metrics such as F1 score, accuracy, and the mean and standard deviation of the error are summarised in Tables 4.10 and 4.11, respectively. The features selected for the SQI computation - sleep rate and positional changes - are justified by their observed correlation with sleep quality, as shown in Figure 4.9. This figure also shows that the percentage of sleep time spent at each different position/activity was not representative of sleep quality. We also introduce the number of sleep awakenings to obtain sleep quality, computed through the sleep-wake analysis described in Section 3.3.2.

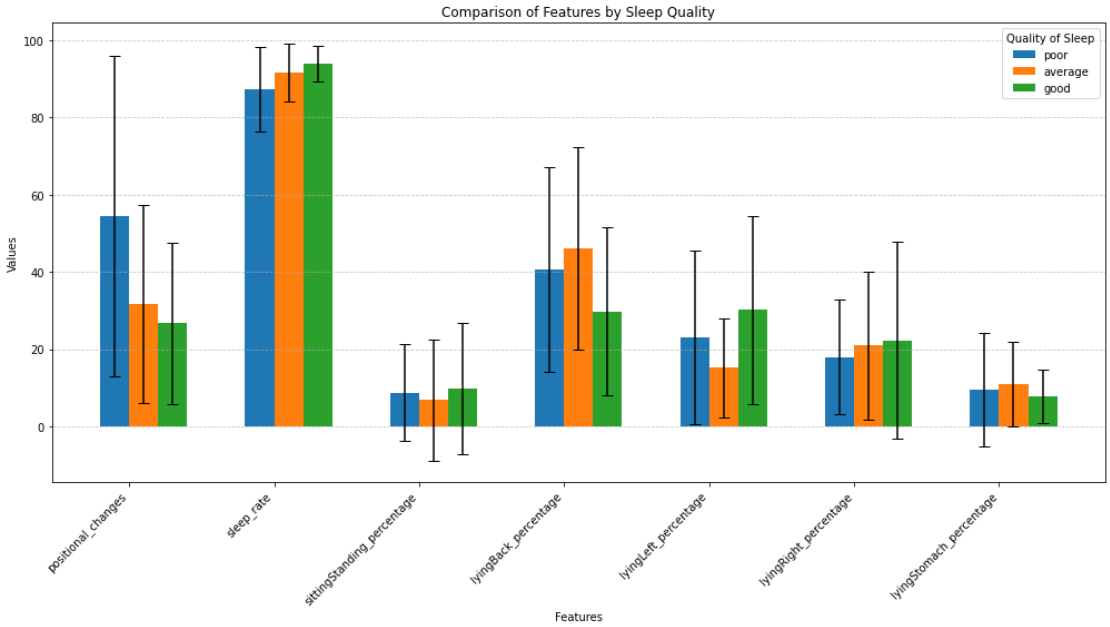


Figure 4.9: Comparison of features by sleep quality. Sleep rate and positional changes show strong correlations with sleep quality, making them suitable for inclusion in the SQI computation.

Performance Comparison

The performance of each method is summarized in Table 4.10. SQI 1 and SQI 2 achieve the highest accuracy, with SQI 2 showing a marginally higher F1 score, indicating that the inclusion of sleep-wake data provides a slight improvement in classification. However, the improvement is minor, suggesting that sleep-wake data is only moderately informative. SQI 3, which uses relative positional changes, has lower accuracy compared to SQI 1 and SQI 2. This aligns with the observations from the positional changes analysis, where relative metrics were less distinctive and thus produced fewer correct predictions. SQI 4 achieves the same high accuracy as SQI 1 and SQI 2 and equal F1 score to SQI 1.

Error Analysis

Table 4.11 summarises the mean and standard error for each method. SQI 3 shows the lowest error, which aligns with the observation that relative positional changes reduce variability. However, its lower accuracy suggests that while relative metrics are more robust, they fail to create clear separations between sleep quality categories. SQI 1 and SQI 2, while achieving higher accuracy, exhibit higher errors due to the variability in absolute positional changes. SQI 4 demonstrates improved

Method	Accuracy (%)	F1 Score
SQI 1: APC + SR	67.65	65.78
SQI 2: APC + SR + SW	67.65	66.12
SQI 3: RPC + SR	35.29	52.44
SQI 4: APC (Method 2) + SR	67.65	65.78

Table 4.10: Performance metrics (accuracy and F1 score) for each SQI method. APC refers to absolute positional changes, RPC to relative positional changes, SR to sleep rate, and SW to sleep awakenings.

balance, achieving the same high accuracy as SQI 1 and SQI 2, but with lower errors (although slightly higher than SQI 3), making it the best overall performer.

Method	Mean Error	Standard Error
SQI 1: APC + SR	31.15	14.39
SQI 2: APC + SR + SW	34.89	16.49
SQI 3: RPC + SR	4.75	4.81
SQI 4: APC (Method 2) + SR	21.38	9.41

Table 4.11: Error analysis (mean and standard error) for each SQI method.

Coefficient and Threshold Analysis

Table 4.12 summarises the coefficients (A, B, C) and thresholds (Threshold 1 -T1- and Threshold 2 -T2-) for each SQI method. The values of A in the best-performing models indicate a negative correlation with positional changes, aligning with the expectation that fewer positional changes are associated with better sleep quality. The positive values of B confirm a direct correlation with sleep efficiency. The negative but smaller values of C suggest a weaker negative correlation with sleep-wake data, making it less influential compared to the other factors.

Method	A	B	C	T1	T2
SQI 1: APC + SR	-0.90	0.83	–	20.78	69.70
SQI 2: APC + SR + SW	-0.99	0.89	-0.45	14.33	69.17
SQI 3: RPC + SR	0.18	0.24	–	21.09	24.16
SQI 4: APC + SR (Method 2)	-0.94	0.84	–	34.56	69.78

Table 4.12: Coefficients and thresholds for each SQI method.

In conclusion:

- **SQI 1 and SQI 2**: Both methods achieve the highest accuracy, with SQI 2 providing a marginally higher F1 score. This suggests that sleep-wake data

is somewhat informative but does not significantly improve overall classification.

- **SQI 3**: Despite lower accuracy, this method demonstrates reduced error, indicating that relative positional changes are more robust but less distinctive for category separation.
- **SQI 4**: Balances accuracy and error, showing the same highest accuracy as SQI 1 and 2 but reducing the error. This makes it as efficient in classification as SQI 1 and 2 but additionally more robust. It demonstrates to be the best-performing model.

4.4 Critical Analysis of Results

The results across all methodologies highlight key strengths and limitations. In physical activity classification, CNN-based models showed strong validation performance but struggled to generalise to the test set, especially in dynamic classes like "Normal Walking" and "Miscellaneous," which are likely impacted by higher variability. In contrast, static activities achieved higher accuracy due to their predictable patterns.

For social sign classification, CNN models demonstrated significant improvements over traditional methods like SVMs and Random Forests. However, even the top CNNs achieved only moderate accuracy, particularly in the "Other" category, which lacks consistent patterns. This also results in weaker real-time accuracy, which can definitely be improved.

In sleep analysis, positional changes and sleep-wake cycle detection revealed trends linking fewer positional changes with better sleep quality. However, normalisation techniques in some methods diluted category distinctions. Clustering-based sleep-wake analysis yielded reasonable results but faced challenges related to the lack of ground truth validation. While absolute metrics proved robust, further refinement is necessary for greater consistency across varying sleep behaviours.

In summary, while the methods showed strengths in capturing sequential data and high validation accuracy, issues with generalisation and variability in real-world data were evident.

4.5 Android App Performance

The performance of the implemented Android application was evaluated with respect to several metrics, including communication and prediction latency, power consumption, CPU usage, and memory usage. Communication latency, defined as

the time required for data transmission between the sensors and the app, averaged 47 ms, while the prediction latency, determined by the machine learning model, was 4 seconds. Power consumption was tested on the provided mobile device, where continuous use of the app for 1 hour consumed approximately 0.91 W.

Memory usage during normal operation was around 55 MB, although this might increase as users record activity data, which requires local storage. Regarding CPU usage, the app maintained an average CPU utilization of 35% while running. These metrics demonstrate the app´s suitability for extended use.

Chapter 5

Conclusions

5.1 Summary of Project and Reflection

This project successfully achieved its main objectives of designing and implementing a system capable of performing human activity recognition and analysing sleep patterns using sensor data. Using deep learning techniques and integrating external sensors, the system attained high accuracy in activity classification. Additionally, a user-friendly Android application was developed, featuring an intuitive interface for recording, processing, and visualising activity data.

1D-CNNs proved to be the most efficient method for both physical activity and social sign classification, and hyperparameters such as a window or kernel size, the number of model layers, or the batch size were essential for enhanced accuracy. Regularisation techniques were also crucial to minimise overfitting and ensure as much generalisation as possible to test and real-time data.

In terms of sleep analysis, EM and K-means showed promising results in determining sleep-wake cycles. This metric, along with the number of positional changes and the sleep efficiency, resulted in a successful computation of a Sleep Quality Index, that achieved over 67% accuracy. Considering that the data was labelled by students and was prone to inconsistencies, we consider this result successful.

As is often the case with large projects, several challenges were encountered during development. One major challenge was selecting and optimising the neural network's hyperparameters to maximise accuracy across all activities. Additionally, ensuring seamless communication between the sensors and the application presented technical difficulties, particularly in managing Bluetooth connections and synchronising data.

A lack of previous experience with Kotlin added another layer of complexity, particularly when integrating the machine learning model into the Android application. This required not only understanding Kotlin's syntax but also debugging issues related to the communication between the model and the app's user interface. However, despite the initial difficulties, the process provided valuable insights into both Kotlin and app development.

On the teamwork front, the importance of clear communication became evident, especially when writing the final report. Ensuring that all team members understood each other's contributions and how all the parts were interconnected was

essential. Teamwork and integration of different perspectives were key to producing a cohesive and comprehensive document.

Overall, the project successfully met its objectives while also providing team members with valuable hands-on experience in app development, machine learning models and team collaboration. The challenges encountered served as learning opportunities, laying the foundation for future improvements.

5.2 Areas for Future Work

While the current project achieves high accuracy in activity recognition, there is still room for further improvement to make the predictions even more precise and reduce the delay in real-time predictions to the minimum. These enhancements could be achieved by exploring the combination of multiple models and fine-tuning their parameters for optimal performance.

In the context of sleep analysis, future work could focus on obtaining a more comprehensive sleep quality index. With access to larger and more diverse datasets, the system could improve its ability to detect sleep patterns and analyse sleep quality. This would not only improve the model itself but would open up possibilities for personalised recommendations to help users improve their sleep routines.

Regarding the application itself, several features could be included to enhance the user experience further. For instance, personalised health plans could be generated based on the recorded data, offering users advice to improve their physical activity or overall lifestyle. Another improvement could be the introduction of goal-setting features, allowing users to set activity or sleep-related targets and track their progress over time. Additionally, a voice assistance feature would allow users to control the app hands-free by using voice commands, such as starting or stopping an activity recording, connecting the sensors or accessing the activity history. The voice assistant could also interact with the users by informing them of their current activity or providing tips based on the recorded data.

These additional features would significantly improve the app's accessibility, usability and overall convenience, making it a more valuable tool for health monitoring and lifestyle management.

Bibliography

[1] Hamed Habibi Aghdam, Elnaz Jahani Heravi, et al. Guide to convolutional neural networks. *New York, NY: Springer*, 10(978-973):51, 2017.

[2] Mona Alzahrani and Salma Kammoun. Human activity recognition: Challenges and process stages. *International Journal of Innovative Research in Computer and Communication Engineering*, 5:1111–1118, 2016.

[3] Ong Chin Ann and Lau Bee Theng. Human activity recognition: A review. In *2014 IEEE international conference on control system, computing and engineering (ICCSCE 2014)*, pages 389–393. IEEE, 2014.

[4] Muhammet Fatih Aslan, Akif Durdu, Kadir Sabanci, and Meryem Afife Mutluer. Cnn and hog based comparison study for complete occlusion handling in human tracking. *Measurement*, 158:107704, 2020.

[5] Nicole A Capela, Edward D Lemaire, and Natalie Baddour. Feature selection for wearable smartphone-based human activity recognition with able bodied, elderly, and stroke patients. *PloS one*, 10(4):e0124414, 2015.

[6] KG Manosha Chathuramali and Ranga Rodrigo. Faster human activity recognition with svm. In *International conference on advances in ICT for emerging regions (ICTer2012)*, pages 197–203. IEEE, 2012.

[7] Heeryon Cho and Sang Min Yoon. Divide and conquer-based 1d cnn human activity recognition using test data sharpening. *Sensors*, 18(4):1055, 2018.

[8] Florenc Demrozi, Cristian Turetta, Philipp H Kindt, Fabio Chiarani, Ruggero Angelo Bacchin, Nicola Valè, Francesco Pascucci, Paola Cesari, Nicola Smania, Stefano Tamburin, et al. A low-cost wireless body area network for human activity recognition in healthy life and medical applications. *IEEE Transactions on Emerging Topics in Computing*, 11(4):839–850, 2023.

[9] B. Erfianto and Achmad Rizal. Imu-based respiratory signal processing using cascade complementary filter method. *J. Sensors*, 2022:1–16, 2022.

[10] Ömer Faruk Ertuğrul and Yılmaz Kaya. Determining the optimal number of body-worn sensors for human activity recognition. *Soft Computing*, 21(17):5053–5060, 2017.

[11] John Cristian Borges Gamboa. Deep learning for time-series analysis. *arXiv preprint arXiv:1701.01887*, 2017.

[12] Martin-Philipp Irsch. Semi-supervised human activity recognition with the wearable respeck sensor using gans.

[13] Walaa N Ismail, Hessah A Alsalamah, Mohammad Mehedi Hassan, and Ebtesam Mohamed. Auto-har: An adaptive human activity recognition framework using an automated cnn architecture design. *Heliyon*, 9(2), 2023.

[14] Wenchao Jiang and Zhaozheng Yin. Human activity recognition using wearable sensors by deep convolutional neural networks. In *Proceedings of the 23rd ACM international conference on Multimedia*, pages 1307–1310, 2015.

[15] Charmi Jobanputra, Jatna Bavishi, and Nishant Doshi. Human activity recognition: A survey. *Procedia Computer Science*, 155:698–703, 2019.

[16] Yongjin Kwon, Kyuchang Kang, and Changseok Bae. Unsupervised learning for human activity recognition using smartphone sensors. *Expert Systems with Applications*, 41(14):6067–6074, 2014.

[17] Saeed Mohsen, Ahmed Elkaseer, and Steffen G Scholz. Human activity recognition using k-nearest neighbor machine learning algorithm. In *Proceedings of the International Conference on Sustainable Design and Manufacturing*, pages 304–313. Springer, 2021.

[18] Ronald Mutegeki and Dong Seog Han. A cnn-lstm approach to human activity recognition. In *2020 international conference on artificial intelligence in information and communication (ICAIIC)*, pages 362–366. IEEE, 2020.

[19] Binh Nguyen, Yves Coelho, Teodiano Bastos, and Sridhar Krishnan. Trends in human activity recognition with focus on machine learning and power requirements. *Machine Learning with Applications*, 5:100072, 2021.

[20] Jakob Nielsen. Enhancing the explanatory power of usability heuristics. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 152–158, 1994.

[21] 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.

[22] Daniel Olguın Olguın and Alex Sandy Pentland. Human activity recognition: Accuracy across common locations for wearable sensors. In *Proceedings of 2006 10th IEEE international symposium on wearable computers, Montreux, Switzerland*, pages 11–14. Citeseer, 2006.

[23] Serafeim Perdikis, Dimitrios Tzovaras, and Michael Gerasimos Strintzis. Recognition of human actions using layered hidden markov models. In *Eurographics (Short Papers)*, pages 79–82, 2008.

[24] Charissa Ann Ronao and Sung-Bae Cho. Human activity recognition with smartphone sensors using deep learning neural networks. *Expert systems with applications*, 59:235–244, 2016.

[25] Chamani Shiranthika, Nilantha Premakumara, Huei-Ling Chiu, Hooman Samani, Chathurangi Shyalika, and Chan-Yun Yang. Human activity recognition using cnn & lstm. In *2020 5th International Conference on Information Technology Research (ICITR)*, pages 1–6. IEEE, 2020.

[26] Dapeng Tao, Yonggang Wen, and Richang Hong. Multicolumn bidirectional long short-term memory for mobile devices-based human activity recognition. *IEEE Internet of Things Journal*, 3(6):1124–1134, 2016.

[27] R. Tripathi, A. S. Jalal, and S. C. Agrawal. Suspicious human activity recognition: a review. *Artificial Intelligence Review*, 50:283 – 339, 2017.

[28] Soe Ye Yint Tun, Samaneh Madanian, and Farhaan Mirza. Internet of things (IoT) applications for elderly care: a reflective review. *Aging Clinical and Experimental Research*, 33(4):855–867, April 2021.

[29] Mohib Ullah, Habib Ullah, Sultan Daud Khan, and Faouzi Alaya Cheikh. Stacked lstm network for human activity recognition using smartphone data. In *2019 8th European workshop on visual information processing (EUVIP)*, pages 175–180. IEEE, 2019.

[30] D. White, J. Weil, and C. Zwillich. Metabolic rate and breathing during sleep. *Journal of applied physiology*, 59 2:384–91, 1985.

[31] Kun Xia, Jianguang Huang, and Hanyu Wang. Lstm-cnn architecture for human activity recognition. *IEEE Access*, 8:56855–56866, 2020.