Report on a Real-Time Mobile Activity Recognition System for Daily Activities and Social Signals Using Wearable Sensors

Nikita Rameshkumar (s2160176) and Luise Woehlke (s2030860)

Principles and Design of IoT Systems
Coursework 3 (2024-25)
School of Informatics
University of Edinburgh

2025

Abstract

This report presents the design, implementation, and evaluation of a real-time mobile activity recognition system capable of classifying both daily physical activities and social signals on an Android device. Two wearable Internet of Things (IoT) sensors were used to continuously stream tri-axial accelerometer data to an Android application. These signals are processed by two separate TensorFlow Lite models, one for daily activities (such as walking, running, or lying down) and one for social signals (such as coughing, talking, or hyperventilating). The daily activities model is a dual-input Convolutional Neural Network (CNN) that fuses data from both sensors, while the social signals model employs a CNN-LSTM architecture including normalisation and regularisation techniques. The models achieve per-class accuracies of around 99% for daily activities and around 85% for social signals and real-time classification achieves accuracies between 80% and 100% for most classes. The Android application can show the real-time classifications made and the history of classifications made previously. The application is relatively lightweight and has a low communication latency of approximately two seconds. Thus, this report presents a lightweight and accurate activity recognition system with many potential applications for end users or for further research.

Table of Contents

Chapter 1

Introduction

Over the last decade, advancements in wearable and mobile devices have been achieved as well as steep improvements in Machine Learning performance [Lara and Labrador, 2012]. This opens up many possibilities for the field of Human Activities Recognition (HAR) which is concerned with the identification of physical activities a person is performing, often using wearable and mobile Internet of Things (IoT) devices as well as Machine Learning techniques. The present report draws on these advances for the purpose of a real-time activity recognition system using wearable sensors and running on a mobile Android application. We use two Inertial Measurement Units (IMUs) with tri-axial accelerometers (the RESpeck wearable device and the Nordic Thingy 52) and for classification we use TensorFlow Lite models.

1.1 Project Aims and Objectives

Our system is concerned with the accurate recognition of daily physical activities such as walking, lying on one's back, or ascending stairs and social signals such as breathing normally, coughing, or singing. Such recognition has wide-ranging applications from healthcare [Webber and Rojas, 2021] to sports technology [Tessendorf et al., 2011] and VR [LaValle et al., 2014]. Especially healthcare applications should be highlighted here since our system captures medically relevant information such as coughing, and shuffle-walking (relevant for research on Parkinson's Disease).

Our goal is to provide a convenient system using simple wearable sensors that don't interfere with daily life in combination with a lightweight mobile application. Thus, the activity recognition should be easy to run for extended periods of time, gathering valuable data for end users or for research purposes. Further, the Android application should be easy to use with a pleasant user experience and show the classification results in real time. It should also provide a history functionality, allowing the user to see how much time in total was recognized of each daily activity and social signal on any given day. Technical metrics we aimed to optimize for the app were communications latency (that is, the time it takes from performing an activity to the correct classification label being shown in the app), CPU usage, and RAM usage. All this aims to providesan easily usable basic activity recognition application that could be extended to more specialised,

for example medical, applications.

Our technical goal for the activity recognition classifiers was to achieve a high per-class accuracy for all daily activities and social signals. We aimed to achieve a certain standard of quality across classes since users and other researchers rely on this and it would be impractical and unintuitive to have a low accuracy for some specific class. For daily activities we aimed for a per-class accuracy of at least roughly 90%. For social signals, we aimed for a per-class accuracy of roughly 80% because of the relatively higher difficulty of this task.

1.2 Summary of Methods and Results

Data preprocessing included segmenting sensor signals into non-overlapping 2-second windows (50 time steps at 25 Hz), and we used upsampling to address class imbalances in certain categories. For daily activities, we employed a dual-input Convolutional Neural Network (CNN) that processes three-axis accelerometer streams from both sensors simultaneously. This model architecture incorporates convolution, pooling, dropout, and batch normalization layers to capture and combine spatial-temporal features of each sensor stream. For social signals, we experimented with CNNs, unidirectional and bidirectional LSTMs, and Generative Adversarial Networks (GANs). Ultimately, a CNN architecture with a single LSTM layer outperformed alternative designs, achieving stronger results than the GAN-based approaches.

Our final models provide high per-class accuracy for both daily activities and social signals. In 5-fold cross-validation, the daily activities classifier attained mean accuracies for dynamic activities (e.g., walking, running) and for static activities (e.g., lying on one's back) both at roughly 99%. The social signals model yielded 85% average accuracy across classes, with coughing and hyperventilating around 90%. In real-time tests with a fully integrated Android application, static activities like lying on one's side often reached 100% accuracy, and dynamic tasks like walking achieved 80-100%. While social signals proved more challenging (varying between 70 and 100% accuracy), the overall system reliably recognized most activities. The app operates at low latency (roughly 2 seconds for a classification), modest CPU usage (ca. 15%), and moderate RAM consumption (ca. 80 MB), though the power consumption of the phone used increased by about 40%.

Chapter 2

Literature Survey

2.1 Wearable Sensors

Human Activity Recognition (HAR) can be performed using ambient or wearable sensors [Lara and Labrador, 2012, Jordao et al., 2019]. This report focuses on wearable sensors. The literature shows that wearable sensors have the disadvantage of lower-quality collected data and consequently worse activity recognition performance. However, this can be mitigated using data fusion techniques to eliminate noise and data bias before feeding the data into an ML classifier [Charalampous, 2021].

For social signal recognition, wearable sensors could be utilised in several ways by sensing several different indicators:

- Sensors detecting movement, volume, and tissue composition in the abdomen and chest wall [Folke et al., 2003]
- Sensors detecting airflow in the airways [Folke et al., 2003]
- Sensors detecting blood gas (non-invasively) [Folke et al., 2003]
- Sensors detecting coughing sounds [Kim et al., 2019, Korpáš et al., 1996]

In the last decade, the literature has focused in on sensors detecting the movement of the abdomen and chest wall specifically, which is also the approach used in this report. This method interferes the least with the life of the wearer. This is crucial for data collection as it allows wearing the sensors over long periods of time [Charalampous, 2021].

2.2 ML Classifiers

ML classifiers for HAR can either receive human-crafted features as inputs in order to classify activities or they can come up with informative features for classification themselves using deep learning. The former approach is more common and requires human effort to create meaningful domain-specific features which are then fed into a supervised learning classifier [Ravi et al., 2016].

However, advances in deep learning enable automatic detection of the most informative features based on a specific dataset. Deep learning uses multiple layers of processing to learn data representations at multiple levels of abstraction [LeCun et al., 2015]. This can sometimes surpass the human ability to find the best features, even that of experts [Charalampous, 2021, Cleve and Gustafsson, 2019].

Overall, some of the most common ML classifiers used in HAR are [Cleve and Gustafsson, 2019]:

- Support Vector Machines (SVM)
- Artificial Neural Networks
- Decision Trees
- Bayesian Networks
- Instance-Based Learning
- Ensembles of classifiers

Some of these will be explored in more detail for the specific applications of daily activity recognition and social activity recognition in the next sections.

2.3 Daily Activities Recognition

2.3.1 Support Vector Machines

A Support Vector Machine (SVM) is a classifier using "an optimal "hyperplane" that serves to separate (i.e., "classify") observations belonging to one class from another based on [...] features. That hyperplane can then be used to determine the most probable label for unseen data." [Pisner and Schnyer, 2020]. SVMs are popular in daily activity recognition due to their high performance. For example, He and Jin [2009] achieved a classification accuracy of 98% on the activities running, being still, jumping, and walking. They used accelerometry sensors and their features were prepared using Discrete Cosine Transform (DCT) and Principle Component Analysis (PCA).

2.3.2 Artificial Neural Networks

A Convolutional Neural Network (CNN) is "a kind of feedforward neural network that is able to extract features from data with convolution structures. [...] The architecture of CNN is inspired by visual perception. A biological neuron corresponds to an artificial neuron; CNN kernels represent different receptors that can respond to various features; activation functions simulate the function that only neural electric signals exceeding a certain threshold can be transmitted to the next neuron." [Li et al., 2022]. As mentioned, CNNs, as examples of deep learning, can extract features automatically from data. To make use of CNN's high performance in visual tasks, Jiang and Yin [2015] prepared accelerometer and gyroscope data into visual representations (activity images). They managed to achieve a very slightly higher classification accuracy than an SVM at

significantly lower computational complexity. (The daily activities investigated were walking, ascending stairs, descending stairs, sitting, standing, and lying.)

A Recurrent Neural Network (RNN) is "designed to learn sequential or time-varying patterns. A recurrent net is a neural network with feedback (closed loop) connections." [Jain and Medsker, 1999]. A Long Short-Term Memory Network (LSTM) is an extension of RNNs that mitigates the "vanishing gradient problem"—the phenomenon of the influence of each input either decaying or compounding exponentially through the recurrent loops [Bengio et al., 1994]. LSTMs have been used for HAR in order to use the temporally sequential nature of daily activities towards recognizing them. Murad and Pyun [2017] test LSTMs on five different data sets containing a variety of daily activities such as walking, sleeping, closing drawers, and many more. The datasets all use IMUs but differ in sensor numbers and in whether gyroscopes and magnetometers were included. They were able to surpass the classification accuracy of both CNNs and SVMs on all datasets (except that SVMs were not tested for one out of the five). The method also has the advantage over CNNs of allowing inputs of variable length.

2.4 Social Signals Recognition

A Random Forest Classifier is an ensemble learning algorithm that combines multiple decision trees to improve classification accuracy and reduce overfitting. Each tree is trained on a random subset of the data and features, and the final prediction is made by aggregating the majority vote from all the trees [Breiman, 2001]. In a comparison of Random Forest Classifiers, Naive Bayesian Classifiers, Logistic Regression, and SVMs, Georgescu [2019] found Random Forest Classifiers the most accurate for recognizing coughing. They used accelerometers and gyroscopes and achieved a classification accuracy of 94%. However, when trying to classify between more activities (breathing, coughing, laughing, talking, singing, eating, hyperventilating, and swinging back and forth), the accuracy was only 66%. Eating and singing were almost never classified correctly. In another comparison of ML classifiers (with classes coughing, talking, eating, laughing, and breathing), Random Forests in fact came out last with an accuracy short of 30% Charalampous [2021].

In the latter comparison, only accelerometers were used and Generative Adversarial Network classifiers were favored. A Generative Adversarial Network (GAN) uses two neural networks: a generator that creates synthetic data and a discriminator that distinguishes between real and synthetic data. The two networks are trained simultaneously in a competitive process, where the generator improves by trying to fool the discriminator, and the discriminator improves by identifying the real data [Goodfellow et al., 2020]. However, even the best GAN model identified only achieved a moderate classification accuracy of around 55% highlighting the continuing difficulty of social signal recognition.

Chapter 3

Methodology

3.1 System Overview and Implementation

The developed IoT system is designed for the classification and tracking of activities, utilizing two advanced sensors: the RESpeck wearable device and the Nordic Thingy 52 [1]. The RESpeck device features a tri-axial accelerometer and gyroscope, enabling it to capture acceleration and angular velocity in three dimensions. The Nordic Thingy 52, a compact multi-sensor prototyping platform built on the NRF52 SoC, also records linear acceleration along all three axes. For training data collection, the RESpeck was affixed to the bottom-left side of the ribcage with a specific orientation, while the Thingy was placed in the right front pant pocket, oriented accordingly. The data from both sensors was recorded at a frequency of 25 Hz. Using this dataset, two models were trained: one for classifying daily activities and another for recognizing social signals.

Social signals were categorized into the following tasks, split into four classes, with all stationary activities performed for each signal:

1. Breathing normally
2. Coughing
3. Hyperventilating
4. Other social signals (talking, eating, singing, and laughing)

Daily activities included the following tasks split into eleven classes:

1. Sitting and Standing
2. Lying on the left side
3. Lying on the right side
4. Lying on the back
5. Lying on the stomach

[1] Nordic Thingy 52: `https://www.nordicsemi.com/Products/Development-hardware/Nordic-Thingy-52`

6. Walking normally
7. Running/Jogging
8. Ascending stairs
9. Descending stairs
10. Shuffle-walking
11. Miscellaneous movements

An Android application was developed to serve as the interface between the user and the IoT system, facilitating sensor connections and executing the implemented models on live data to deliver real-time results. The app enabled the user to connect to the sensors via Bluetooth and QR code scanning. For accurate tracking, the system required the user to position and orient the devices as during the training phase. As the user performed activities, the app classified them and displayed real-time graphs of acceleration values from both sensors across the three axes. The classified activity label was presented to the user, with the data being recorded and stored. The user could then access historical activity classification data, view specific dates, and examine the durations spent on each physical activity and social signal.

3.2 Software Organisation

The PDIoT app is designed to integrate the various components of the IoT system, encompassing a sensor interface and Bluetooth wireless link, two TensorFlow Lite models (for daily activities and social signals), an Android application, and an SQLite database to record historic classification data. The work flow spans sensor interfacing, machine learning-based classification, real-time processing, and persistent data logging, ensuring a straightforward user experience for real-time monitoring and analysis. This design ensures real-time data acquisition, processing, and visualisation while maintaining robust logging for historical analysis. The software incorporates the following key components.

3.2.1 Sensor Interface and Bluetooth Wireless Link

The sensor interface is responsible for establishing a Bluetooth Low Energy (BLE) connection between the mobile application and the Respeck and Thingy sensors, also representing a primary connection between the user and the IoT system. BLE technology enables wireless communication for low-power devices such as our sensors. This integration ensures that accelerometer data, including x, y, and z values from both sensors (called 'characteristics'), is transmitted wirelessly to the app in real time. The Bluetooth wireless link allows for a low-latency connection, facilitating continuous data streaming. The mobile app constantly polls the sensors to retrieve fresh data (i.e., the relevant characteristics enable 'notifications') while ensuring the synchronization of the two sensor streams, which is crucial for accurate classification. We use the RXAndroidBLE library to handle this in code. The app processes the incoming sensor data efficiently, ensuring no delay between data capture and model inference.

3.2.2 TensorFlow Lite Models

The app utilizes two TensorFlow Lite models for daily activities as well as social signals. TensorFlow Lite provides a lightweight solution for running ML models on mobile devices. Its advantages include low latency and low resource consumption. We use TensorFlow Lite 2.16.1 with the corresponding GPU acceleration library in order to speed up computation by delegating to the GPU. We also use the TensorFlow Lite support library (version 0.4.4) to pre-process data for CNN use.

Both models process real-time data from Respeck and Thingy sensors, enabling simultaneous classification of activities and social signals. The integration of the models into the mobile application is streamlined, with the models being invoked automatically when as multiple sensor data streams in simultaneously. Section 3.3 dives deep into the implementation structure of the model.

3.2.3 Real-Time Data Processing and Classification

A central component system is the real-time data processing and classification pipeline which was implemented in the Android Application.

- Data processing and model classification occur in real time as new data streams in from the sensors.
- Accelerometer data from Respeck (xRespeck, yRespeck, zRespeck) and Thingy (xThingy, yThingy, zThingy) are appended to sliding window arrays.
- Once the sliding window is filled with 50 time steps of live data, the values are flattened into buffers (respeckData and thingyData).
- These buffers are sent as inputs to the TensorFlow Lite models.
- Classification is performed separately for daily activities and social signals.
- The results are immediately passed to the user interface for visualization using classification labels, images and graphs.
- Classification results are sent to the SQLite database to keep track of time and duration of the activities.
- Once the sliding window is filled with 50 newer time steps, it is sent for classification as a non-overlapping window.
- The previous sliding window is discarded and this cycle repeats as data keeps flowing from the sensors attached to the user.

The sliding window mechanism ensures that the live data streams from both the Respeck and Thingy sensors remain synchronized, using only the most recent 50 time steps of data. This approach guarantees that both sensor streams are processed at a consistent rate, with each sensor operating at a frequency of 25Hz.

3.2.4 Android Application

The Android application serves as the central point of integration for all components, providing a user-friendly interface for interacting with the sensors, viewing real-time data, and monitoring activity classifications. The application employs a modular architecture, which separates the concerns of data acquisition, preprocessing, classification, and visualization. The Bluetooth communication module manages the connection between the mobile device and the sensors, ensuring stable and efficient data streaming. The data acquisition layer is integrated with the preprocessing pipeline, which formats and organizes the sensor data into the appropriate structure for model input. The app uses MPAndroidChart to provide real-time visual feedback of the sensor data and classification results, making it easy for users to monitor their activity status. As new sensor data is received, it is passed through the preprocessing module, and real-time classification results are displayed to the user almost instantaneously. The Android application also integrates historic data functionality, allowing previously recorded sensor data and classification results to be accessed and analyzed alongside real-time data, enabling users to track and review past activity trends. The integration of these components within the Android application ensures a seamless and interactive experience for users while maintaining optimal performance. A description of the app from a user and user interface perspective is provided in Section 3.4.

3.2.5 Database

An SQLite database is integrated to persistently store classified data and associated metadata. We use SqLite 2.2.0 with an abstraction provided by the Room library (version 2.4.2) that ensures type safety and the associated Room compiler. The database serves as the key component for storing historic activity records, consisting of the activity name, the start timestamp, and the end timestamp (in milliseconds). This ensures that data is logged continuously as new classifications are made, enabling the user to analyse their activities through visualisation of times and durations. A new database entry is logged every time the real-time classification from one of the TensorFlow Lite models changes and upon exiting the live activity page of the app.

The database integration is designed to be lightweight and database operations are performed on background threads, ensuring minimal impact on the app's performance while providing long-term data storage capabilities. Additionally, the integration of the database with the classification pipeline ensures that newly classified data is logged in real time, allowing users to instantly access their up-to-date historic data. (Placeholder data is provided for evaluating and testing the app.)

Figure 3.1 represents the software organization of the IoT system.

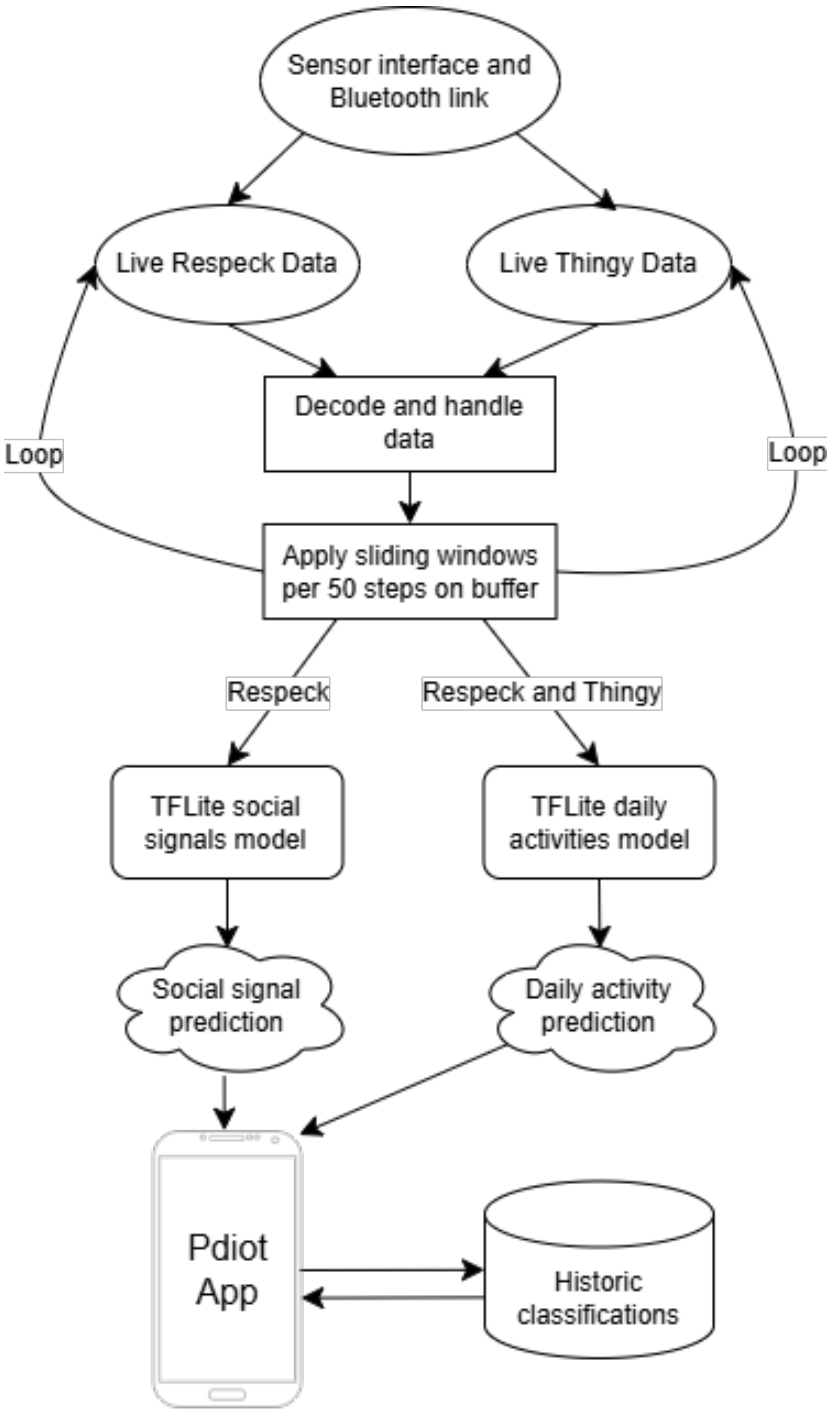

Figure 3.1: Software organization

3.3 Algorithms and Methods Used for Activity Recognition

3.3.1 Dataset and Data Pre-Processing

Separate Machine Learning models were trained to recognize daily activities (sitting, walking, ascending stairs, etc.) and social signals (coughing, hyperventilating, etc.)

The basis for this is a dataset of 3,446 30-second recordings, some with a Respeck and some with a Thingy. The recordings each contain one daily activity and some contain an additional social signal performed at the same time (e.g., sitting and coughing at the same time). The numbers of samples for each daily activity and social signal can be seen in Tables 3.1 and 3.2. (The numbers in the tables don't add up to 3,446 because some recordings contain both a daily activity and a social signal.) 20% of samples were set aside for testing our ML classifiers after training.

Daily Activity	**Number of Samples**
Sitting / standing	629
Walking	248
Shuffle walking	245
Running	247
Lying on left side	313
Lying on right side	309
Lying on back	318
Lying on stomach	318
Ascending stairs	261
Descending stairs	261
Miscellaneous movement	252

Table 3.1: Numbers of 30-second sensor recordings for the eleven daily activity classes

Social Signal	**Number of Samples**
Breathing normally	931
Coughing	723
Hyperventilating	725
Talking / singing / laughing / eating	2,408

Table 3.2: Numbers of 30-second sensor recordings for the four social signal classes

We segmented all data into sliding windows to allow our models to classify manageable chunks as well as for later integration into the classification system with the mobile app which works in real time with short time windows. We experimented with window sizes between 25 and 128 time steps but window size 50 produced the best results. At 25 Hz (due to Bluetooth LE technology), this corresponds to 2 seconds of real time. The step size was 50, meaning the windows don't overlap. We experimented with overlapping windows but the results of these experiments were ambiguous.

3.3.2 Daily Activities Classification

A single TensorFlow Lite model was developed to classify daily activities into the 11 specified classes, using dual input from the Respeck and Thingy sensors, which provide acceleration data across three axes. The model structure consists of two parallel branches, one for the Respeck sensor and another for the Thingy sensor. Each branch processes its respective input data before the results are combined for classification. The thirteen layers of this model include:

Respeck and Thingy Branches

- Layer 1: A 1D convolutional layer with 16 filters, kernel size 5, ReLU activation, and 'same' padding.
- Layer 2: A batch normalization layer to stabilize and speed up the training.
- Layer 3: A max pooling layer with a pool size of 2 to reduce the spatial dimensions.
- Layer 4: A 1D convolutional layer with 64 filters, kernel size 3, ReLU activation, and 'same' padding.
- Layer 5: Another batch normalization layer.
- Layer 6: A max pooling layer with a pool size of 2.
- Layer 7: A 1D convolutional layer with 128 filters, kernel size 3, ReLU activation, and 'same' padding.
- Layer 8: A dropout layer with a rate of 0.3 to prevent overfitting.
- Layer 9: A global average pooling layer to reduce the output to a single vector.

Concatenation

The outputs from both the Respeck and Thingy branches are concatenated to form a unified feature vector. Fully Connected Layers:

- Layer 10: A dense layer with 512 units, ReLU activation, and L2 regularization.
- Layer 11: A dropout layer with a rate of 0.6 to reduce overfitting.
- Layer 12: A dense layer with 256 units, ReLU activation, and L2 regularization.
- Layer 13: The output layer, which uses softmax activation to classify the input into one of the specified classes.

The Convolutional Neural Network is a gradient descent based learning method and suffers from the gradient dispersion problem. [2]. This is mitigated using the ReLU function [Bai, 2022] that is used as one of the final layers. The final TFLite model is compiled using the Adam optimizer with a learning rate of 0.001 and categorical crossentropy as the loss function, optimized for accuracy. Additionally, a learning rate scheduler (ReduceLROnPlateau) is used to adjust the learning rate dynamically during training, improving performance and model convergence.

Prior to model development, there was an inconsistency in the number of data points available for the Respeck and Thingy sensors, resulting in unused data when one sensor provided more data than the other. To address this, up-sampling was applied to ensure no data was discarded. This is a data processing and optimization technique that addresses class imbalance in a dataset by adding data. Up-sampling adds data by using original

[2]Gradient Dispersion Problem: `https://en.wikipedia.org/wiki/Vanishing_gradient_problem`

samples from minority classes until all classes are equal in size. [3] This technique led to an approximate 4% improvement in daily activities model accuracy.

3.3.3 Social Signals Classification

For social signals, we tested both Convolutional Neural Network (CNN) and Generative Adversarial Network (GAN) architectures. The use of GANs was inspired by Charalampous's experiments with a very similar experimental setup to ours [Charalampous, 2021]. We replicated both their Semi-Supervised GAN (S-GAN) and Auxiliary Classifier GAN (AC-GAN) models. However, ultimately our CNN architectures performed better. We will here only go into detail on our final best-performing CNN model. Detailed comparisons between the GAN models and CNN models can be found in Chapter 4 (Results and Discussion).

Our best-performing CNN model uses two convolutional layers, a single LSTM layer, and dropout layers to reduce overfitting. CNNs can already capture local (short) temporal dependencies in data (in this case, in a social signal) but LSTM layers enable recognizing longer dependencies as are likely present in social signals like coughing or eating [Bengio et al., 1994] Moreover, the LSTM layers add parameters and greater parameter numbers allow learning more complex representations and preventing underfitting [LeCun et al., 2015]. To balance out a higher number of parameters, we use Batch Normalisation on each convolutional layer. This can improve training stability and convergence [Li et al., 2022]. Since some of our classes have more samples than others, we use L2 Regularisation on the second convolutional layer in hopes of reducing overfitting to specific classes. The detailed layers are laid out in Table 3.3.

3.4 The Mobile Application From a User's Perspective

The Android app allows the user to view the live processing of sensor data, see the classification results, and view their activity history. The home screen of the app allows selecting the desired action: Record Data, Watch Live Processing, Connect Sensors, or View Activity History (see Figure 3.2).

Under "Connect Sensors", the Respeck and Thingy sensors can be connected using Bluetooth Low Energy (BLE) technology.

Once a sensor is connected, the user can see the real-time data processing and classification under "Watch Live Processing". Initially, the screen will show "Classifying activity..." and "Social Signal: Classifying activity". There is an approximately 2-second delay to collect enough sensor time steps to make classifications. Both classifications are made and displayed in parallel as can be seen in Figure 3.3.

For the classifications displayed, we chose large bold text to visually highlight them and we added an image of the daily activity that is currently predicted to make the page more visual and intuitive. The image changes instantly when the daily activity prediction changes.

[3] Up-sampling: `https://www.ibm.com/think/topics/upsampling`

Layer (type)	Output Shape	Param #
conv1d_3 (Conv1D)	(None, 46, 128)	2,048
batch_normalization (BatchNormalization)	(None, 46, 128)	512
max_pooling1d_2 (MaxPooling1D)	(None, 23, 128)	0
dropout_1 (Dropout)	(None, 23, 128)	0
conv1d_4 (Conv1D)	(None, 19, 256)	164,096
batch_normalization_1 (BatchNormalization)	(None, 19, 256)	1,024
max_pooling1d_3 (MaxPooling1D)	(None, 9, 256)	0
dropout_2 (Dropout)	(None, 9, 256)	0
lstm (LSTM)	(None, 128)	197,120
flatten_1 (Flatten)	(None, 128)	0
dense_2 (Dense)	(None, 128)	16,512
dropout_3 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 4)	516
Total param #		**381,828**
Trainable param #		**381,060**

Table 3.3: Layer-wise details of our best-performing CNN model architecture

The user can scroll down on this page to view the live incoming sensor data from both Respeck and Thingy. The triaxial accelerometer measurements are shown by three lines per graph. Although the Thingy can also record gyroscope data, we do not use this data in our classification and do not show it in the graphs here.

Under "View Activity History", the user can select a date and view the total duration each activity was performed on that day. The individually logged entries for each activity are added together automatically so only the total duration per activity is shown. The page initially shows today's date and another date can be selected by using a button that opens a calendar view (see Figure 3.4).

3.5 Testing

There are a variety of performance metrics used in the evaluation of Machine Learning classifiers. Among the most common are:

- **Precision:** how many of the predicted positive samples are actually positive
- **Recall:** how many of the actual positive samples were correctly predicted
- **F1-score:** harmonic mean of precision and recall

$$\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

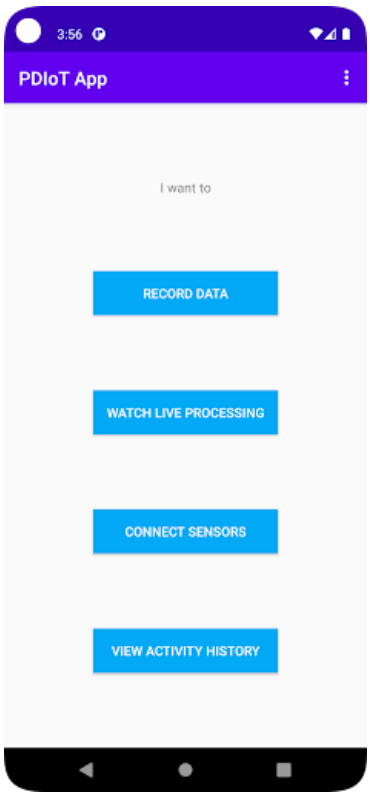

Figure 3.2: The home screen of the app allows selecting the desired action: Record Data, Watch Live Processing, Connect Sensors, or View Activity History

- **Per-class accuracy:** measuring correct predictions for a single class, i.e.

$$\frac{\text{true positives} + \text{true negatives}}{\text{true positives} + \text{true negatives} + \text{false positives} + \text{false negatives}}$$

- **Total (model) accuracy:** ratio of correctly predicted samples to total samples

In the course of refining our models, we used all of the above in some capacity. For example, precision showed us when a model was overpredicting a certain class as in the case of our second CNN architecture described in subsection 4.1.5. We paid special attention to the F1-score, and per-class accuracy. The F1-score was helpful for balancing the tradeoff between precision and recall. However, our ultimate metric was per-class accuracy since each daily activity and social signal should be classified with a certain guarantee of the quality of the predictions. It would be counterintuitive from a user's perspective that they cannot trust the predictions of one specific class.

3.5.1 Daily Activities Model Testing

To evaluate the performance of the model for the dual-input daily activity classification, a methodical and data-driven testing methodology was adopted. While changes were made to the model, a fixed test suite and metric was used to evaluate whether there was an improvement in the model or not. The dataset, comprising all activity classes, was

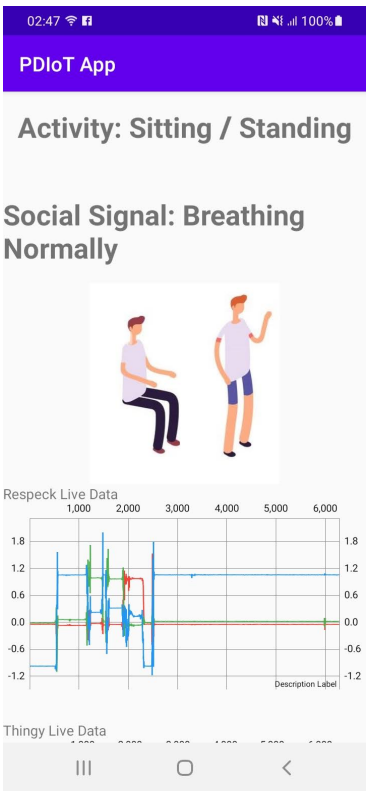

Figure 3.3: The ”Watch Live Processing” page which shows simultaneous live classifications of the daily activity and social signal at the very top

divided into an 80% training set and a 20% test set. This split ensured adequate data for training while preserving a test set for unbiased model evaluation.

Evaluation Metrics during Model Development

The evaluation of the model focused on achieving high accuracy across all activity classes and their sub-categories. A key benchmark was ensuring that the average F1-scores for all 11 classes exceeded 0.90, alongside the averages for two sub-categories of classes: static and dynamic exceeded 0.9 as well. This ensured that static and dynamic activities met the desired thresholds for robust classification allowing for a focused comparison of model performance across activity types.

Static activities included:

1. Sitting and Standing
2. Lying on the left side
3. Lying on the right side
4. Lying on the back
5. Lying on the stomach

Dynamic activities included:

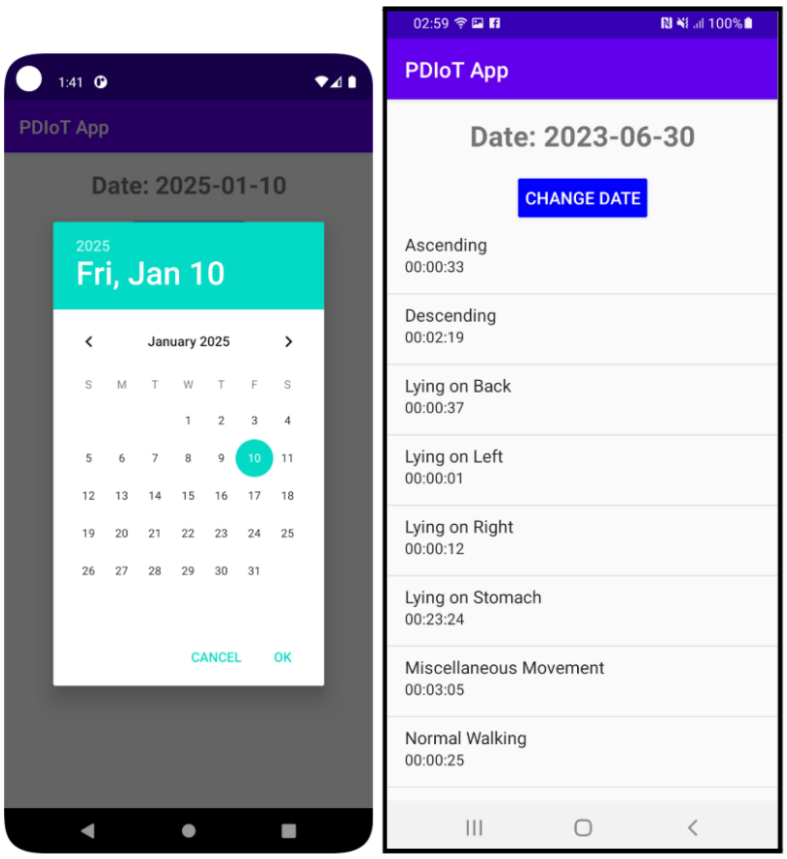

Figure 3.4: The ”View Activity History” page allows selecting a date and viewing the activities performed on that date and their durations

1. Walking normally
2. Running/Jogging
3. Ascending stairs
4. Descending stairs
5. Shuffle-walking
6. Miscellaneous movements.

Overall accuracy, macro averages, and weighted averages were calculated to provide a holistic view of the model’s performance. Additionally, the evaluation also relied on standard performance metrics, including precision, recall, F1-score, and support for each activity class.

Handling Data Imbalance

The dual-input nature of the daily activities model necessitated equal representation of Respeck and Thingy data. To address this, upsampling was applied to both the training and test datasets, ensuring balanced sample sizes between the two inputs.

This approach mitigated the risk of biased model training or evaluation due to data imbalances, particularly for underrepresented activities or sensor streams.

Final Evaluation and Validation

After selecting the top-performing model based on the initial evaluation using metrics like precision, recall, and F1-scores, the model underwent a more rigorous validation process. This included K-fold cross-validation where K=5 to ensure that the model's performance was consistent and generalizable across different subsets of the data. This involved dividing the available data into 5 folds, using one of these folds as a validation set, and training the model on the remaining folds. This process is repeated 5 times, each time using a different fold as the validation set. Finally, the results from each validation step were averaged to produce a more robust estimate of the model's performance. [4] This additional layer of validation reinforced the model's robustness, ensuring that it could accurately classify both static and dynamic activities across diverse conditions. The combination of both development-phase evaluation and cross-validation ensured that the final model achieved high performance with minimal bias and over-fitting.

By combining various methods, such as using an upsampled test set and focusing on achieving F1-scores above 0.90 across all classes and their respective sub-categories, different models were compared and evaluated through this quantitative approach to identify the best-performing model. To further enhance confidence in the robustness of the model's performance, 5-fold cross-validation was implemented, ensuring consistent and reliable results across different data subsets.

3.5.2 Social Signals Model Testing

Testing of the social signal models was similar to that of the daily activities models. A 5-fold cross-validation was used to evaluate our final best-performing model. The results are described in chapter 4. Intermediate and lower-performing models were tested as described in the previous section with a split of 80:20 training:testing data. The only difference was that our per-class accuracy goal for social signals was 80% instead of 90% due to the higher difficulty of the task. During hyperparameter tuning, the best hyperparameter configurations were approximately identified by only comparing total model accuracy. Those fewer identified models were subsequently also tested as described in the previous section.

3.5.3 Overall System Testing

After evaluating the two models on static data and selecting the higher-performing one, the models were integrated into the IoT system, which included the mobile application and sensors that provided a live stream of data. As a result, the overall system integration required thorough testing. This was performed by conducting each task ten times, with a latency of 5 seconds. This latency allowed for slightly more time, incorporating four non-overlapping sliding windows of 50 time steps each for accurate classification. With

[4] K-Fold Cross Validation: https://www.geeksforgeeks.org/cross-validation-machine-learning/

a frequency of 25Hz (25 steps per second), 5 seconds corresponded to 225 steps, and each window required 50 steps, resulting in four non-overlapping windows within the 5-second latency period.

By repeating each activity 10 times, we calculated the accuracy for each class and assessed whether the social signals achieved an average accuracy of 0.80, and if the static and dynamic activities reached an average accuracy of 0.90. Additionally, we measured the system's communication latency, which indicated how quickly the system classified activities on average. We also monitored power consumption, memory usage, and CPU usage across the complex system to evaluate its performance comprehensively.

Chapter 4

Results and Discussion

4.1 Accuracy of Machine Learning Models

Two machine learning models were designed to power the IoT system for classifying daily activities and social signals. Ensuring high accuracy was a critical priority to validate the reliability of these classifications. During the development and testing phases, accuracy metrics were carefully analyzed, with a target of achieving over 90% accuracy for both static and dynamic physical activities, and at least 80% for each social signal class. subsection 4.1.1 and subsection 4.1.3 provide detailed accuracy results of the optimized models while also offering a critical evaluation of earlier models that underperformed.

4.1.1 Daily Activities Classification

Our Best Model

The model that produced the best results through the evaluation methods and metrics explained in subsection 3.5.1 was used for the final IoT system. According to the 5-Fold cross validation metric, for static activities, this model achieved an accuracy of 99.41% and for dynamic activities an average accuracy of 98.53%. Table 4.1 depicts the K-Fold results in detail per-class, for the winning dual input CNN model.

Conversely, during the development phase, the per-class accuracy and the stricter F1-score metric were calculated for each class after dividing the dataset with a 20% test split. The static activities achieved a mean accuracy of 98.65%, while the dynamic activities attained a mean accuracy of 97.26%. The overall accuracy of the final model was 90.98%. Miscellaneous movement and shuffle walking exhibited the lowest F1-scores (73.27% and 76.39% respectively), likely due to the greater variability in the dataset. The miscellaneous movement class encompassed random, unstructured motions, while shuffle walking displayed inconsistency stemming from variations in how the activity was performed. Lying on the stomach and lying on the back achieved the highest F1-scores (98.17% and 97.91%, respectively), attributed to the static nature of these activities and the distinctive, consistent angles at which the sensors were positioned.

Table 4.1: K-Fold Validation Results for Daily Activities

Class	Accuracy	F1-Score
Sitting and Standing	0.9931	0.9804
Lying on Back	0.9950	0.9660
Lying on Left	0.9929	0.9592
Lying on Right	0.9937	0.9585
Lying on Stomach	0.9957	0.9703
Miscellaneous Movement	0.9758	0.8371
Normal Walking	0.9867	0.9227
Running	0.9954	0.9696
Shuffle Walking	0.9793	0.8660
Ascending	0.9874	0.9268
Descending	0.9871	0.9288
Average for Static Activities	**0.9941**	**0.9669**
Average for Dynamic Activities	**0.9853**	**0.9085**

Table 4.2 presents the accuracy metrics on the 20% test set used during the development phase.

Table 4.2: Per class Accuracy on Development Phase 20% Test Set

Class	Accuracy	F1-Score
Sitting and Standing	0.9916	0.9605
Lying on Back	0.9885	0.9791
Lying on Left	0.9751	0.8994
Lying on Right	0.9862	0.9608
Lying on Stomach	0.9910	0.9817
Miscellaneous Movement	0.9613	0.7327
Normal Walking	0.9682	0.9336
Running	0.9910	0.9281
Shuffle Walking	0.9646	0.7630
Ascending	0.9801	0.9020
Descending	0.9701	0.9186
Average for Static Activities	**0.9865**	**0.9561**
Average for Dynamic Activities	**0.9726**	**0.8630**
Model Accuracy	**0.9098**	

4.1.2 Model Comparisons

CNN for Respeck Only

The original CNN model, designed for time-series signal data, served as the baseline for activity classification. The CNN model was trained exclusively on Respeck data to evaluate the contribution of a single sensor to activity recognition. The model achieved a total accuracy of 91.60%, excelling in static activity classification with an accuracy of 98.80%. However, dynamic activities were poorly classified, with an accuracy of

84.15%. The absence of Thingy data, which provides complementary information for dynamic activities, limited the model's ability to capture nuances of movement effectively.

Mix of CNN and LSTM

A hybrid model combining CNN and LSTM layers was evaluated to explore the potential benefits of incorporating temporal dependencies. While the CNN layers extracted spatial features, the LSTM layers aimed to capture long-term dependencies in the data. Despite its promise, the model underperformed with an accuracy of 84.86%. Dynamic classes, which rely heavily on temporal patterns, suffered from imbalanced recall, with some classes as low as 58.25%. Additionally, the model tended to overfit to static activities, indicating that the added complexity did not generalize significantly to dynamic movements.

Addition of Thingy Data and Previous Years' Data

The dataset was expanded to include data from previous years alongside the current year to evaluate the impact of additional variability on model performance. The model's accuracy improved to 90%, demonstrating that the added variability positively influenced the classification results, particularly when combining CNN with the LSTM model.

Further experimentation showed that when using only Respeck data, the CNN model with LSTM achieved an accuracy of 89.52%. However, when both Respeck and Thingy data were incorporated, the LSTM model maintained an accuracy of 90%. In contrast, when the model was combined with Thingy data but without LSTM, it outperformed the others, achieving the highest accuracy of 92.41%. However, the model required more data points as the dynamic activities still had an average accuracy of less than 90%.

Weight Additions

Given that the CNN model, which included resampling and used data from both the Respeck and Thingy sensors across past and current years, achieved less than 90% accuracy for dynamic activities and over 95% accuracy for static activities without the inclusion of an LSTM layer, the addition of weights was explored as a potential means to enhance model performance. Various weighting strategies were tested, where additional weight was assigned to underperforming classes, and reduced weight was given to those performing well. The left column of Table 4.3 illustrates the original model structure and shows the impact of incorporating weights, which led to a decrease in accuracy to 84.21%.

Layer Additions

By removing the weights and adding several new layers, the model structure transitioned from the left to the right column in Table 4.3, resulting in a significant increase in accuracy. These modifications directly contributed to improvements in both static and dynamic activities. The addition of BatchNormalization and Dropout layers, along with

adjustments to the convolutional layers, enhanced the model's ability to generalize, particularly for more challenging dynamic activities. The model's accuracy rose to 90.98%, with dynamic activities achieving an average accuracy of 97.26%, and static activities reaching an average of 98.65%. The F1-scores also showed substantial improvement, reflecting better performance across individual classes, including Sitting and Standing (0.9605), Lying on Back (0.9791), and Normal Walking (0.9336). These changes resulted in a more balanced and robust model, with improved accuracy and F1-scores across both static and dynamic activity categories.

This model was finalised as the winning model.

Old Model Structure	**Final Model Structure**
Respeck Branch:	**Respeck and Thingy Branches:**
Conv1D(64, 3, relu)	Conv1D(16, 5, relu, padding='same')
MaxPooling1D(2)	BatchNormalization()
Conv1D(128, 3, relu)	MaxPooling1D(2)
MaxPooling1D(2)	Conv1D(64, 3, relu, padding='same')
GlobalAveragePooling1D()	BatchNormalization()
	MaxPooling1D(2)
Thingy Branch:	Conv1D(128, 3, relu, padding='same')
Conv1D(64, 3, relu)	Dropout(0.3)
MaxPooling1D(2)	GlobalAveragePooling1D()
Conv1D(128, 3, relu)	
MaxPooling1D(2)	**Fully Connected Layer:**
GlobalAveragePooling1D()	Dense(512, relu, kernel_regularizer=l2(0.001))
Dropout(0.3)	Dropout(0.6)
Dense(128, relu, kernel_regularizer=l2(0.001))	Dense(256, relu, kernel_regularizer=l2(0.001))
Dense(num_classes, softmax)	Dense(num_classes, softmax)
Optimizer:	**Optimizer:**
Adam()	Adam(learning_rate=0.001)
Weights:	**Weights:**
True	False
Accuracy:	**Accuracy:**
0.8421	0.9098

Table 4.3: Comparison of Layers and Accuracy between Previous and Final Model

Figure 4.1 depicts the implementation stages of the model and how changes to the model's format and layers affected its performance. Models highlighted in red did not meet the requirement of achieving an average accuracy exceeding 90% for dynamic activities. The model highlighted in green represents the successful model, which satisfied the criteria by achieving an average accuracy greater than 90% for both static and dynamic activities.

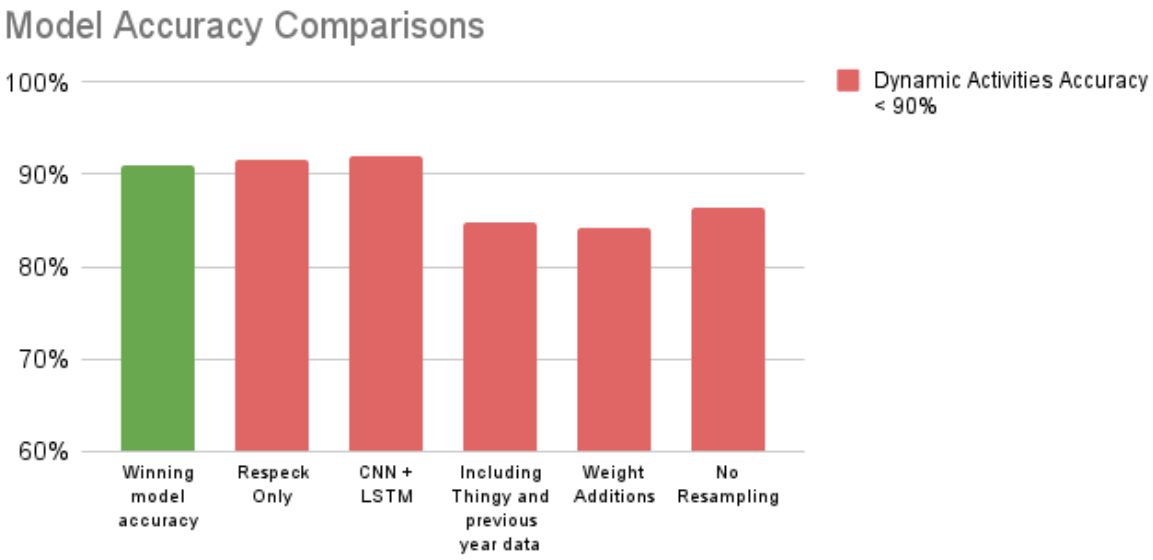

Figure 4.1: Daily Activities Model Accuracy Comparisons

4.1.3 Social Signal Classification

4.1.3.1 Our Best Model

We evaluated our final social signal classifier using 5-fold cross-validation. The mean results are visualised in Figure 4.2 and the detailed results can be seen in Table 4.4. The mean per-class accuracy of three out of the four classes reached over 80%. Only the ”Other social signals” category (laughing, singing, eating, talking) was harder to classify and reached only 74% on average.

Metric	Mean	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Breathing normally accuracy	.84	.83	.82	.84	.86	.85
Coughing accuracy	.91	.92	.91	.91	.91	.91
Hyperventilating accuracy	.89	.89	.89	.89	.88	.88
Other social signals accuracy	.74	.73	.72	.75	.77	.74
Average	.85	.84	.84	.85	.86	.85

Table 4.4: 5-fold cross-validation results for our social signal classifier

The lower accuracy for ”Other social signals” is likely due to the variety of signals in this category, a problem all models we tested struggled with to some extent. We tested a range of other more or less similar CNN models as well as two Generative Adversarial Network (GAN) models. The rest of this section deals with the results of these trials as well as discussion of likely reasons for their relative performance.

4.1.4 Generative Adversarial Network (GAN) Models

We replicated both the Semi-Supervised GAN (S-GAN) and Auxiliary Classifier GAN (AC-GAN) models from Charalampous [2021]. We were able to replicate their accuracy

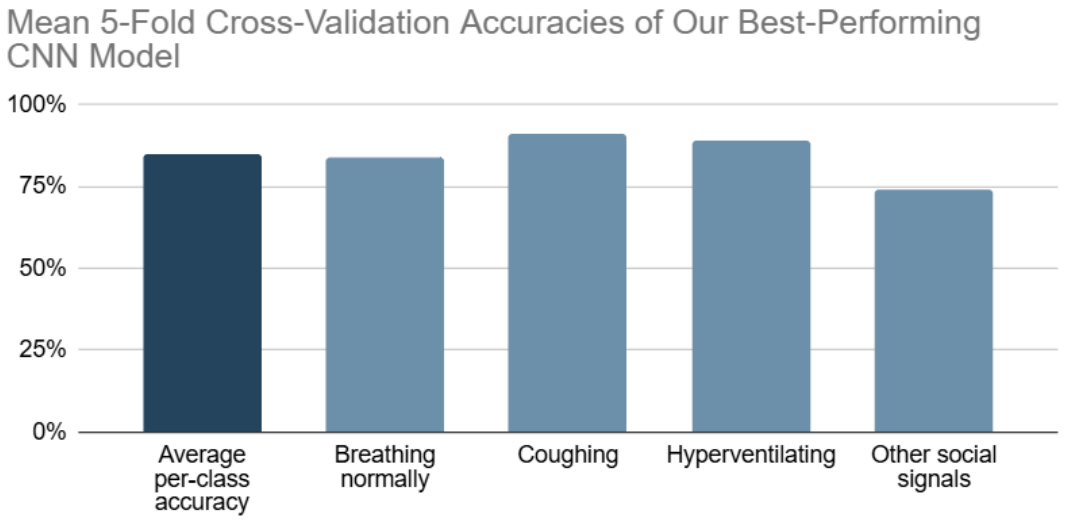

Figure 4.2: Mean 5-Fold Cross-Validation Accuracies of Our Best-Performing CNN Model

of about 55% (although this is only an approximate replication because we used our own dataset for both training and testing). We tried out different window sizes and step sizes but this barely made a difference to accuracy, as can be seen in Table 4.5.

	S-GAN	**AC-GAN**				
Window + step size	128 + 50	128 + 50	72 + 72	72 + 36	24 + 24	24 + 12
Breathing normally	.84	.84	.80	.80	.81	.84
Coughing	.86	.86	.85	.85	.86	.85
Hyperventilating	.85	.85	.83	.83	.85	.84
Other social signals	.59	.59	.58	.58	.56	.60
Average accuracy	.79	.79	.77	.77	.77	.78

Table 4.5: Classification accuracies achieved by S-GAN and AC-GAN models of differing window and step sizes. (Batch size during training was always 128. The number of epochs varied due to the differing time requirements of the training runs.)

These accuracy results, however, are much lower than those we achieved using CNNs. Our best-performing GAN model and our best-performing CNN model are compared in Table 4.6. The average per-class accuracy of the CNN model is a whole 7 percentage points higher. This is substantially driven by the accuracy advantage of about 14 percentage points that the CNN model has in classifying the ”other social signals” category.

An explanation for this may be that GAN models cannot deal with highly diverse classes such as ”other social signals”, which includes talking, eating, laughing, and singing. GAN models generate synthetic samples for each class, however the generated samples may be a sort of mixing/middling between existing talking, eating, laughing, and singing samples, and may not actually resemble any single one of them. A way to test this explanation would be to train an 8-class classifier where ”other social signals” is split up into its four component signals and seeing if the GAN accuracy improves.

	Best GAN model	Best CNN model
Window + step size	24 + 12	50 + 50
Breathing normally	.84	.84
Coughing	.85	.91
Hyperventilating	.84	.89
Other social signals	.60	.74
Average accuracy	.78	.85

Table 4.6: Classification accuracies of our best-performing GAN model and our best-performing CNN model in comparison.

It should also be noted that generating synthetic samples is an approach usually taken because of scarcity in real samples, however, the ”other social signals” category already had about three times as many samples as each of the others.

4.1.5 Convolutional Neural Network (CNN) Variations

For CNNs, we tested three different architectural ideas, several techniques for data and training augmentation, and many choices for specific hyperparameters.

Our first architectural idea was to take a basic CNN and add LSTM layers. As mentioned, LSTMs help the model recognize longer temporal dependencies in the signal and they also add parameter numbers to enable learning more complex representations [Bengio et al., 1994]. Dropout layers were used to prevent overfitting. The detailed layers can be seen in Table 4.7. Training was performed over 50 epochs with a batch size of 32 and automatic halving of the learning rate whenever the validation loss plateaued.

Layer (type)	Output Shape	Param #
conv1d_2 (Conv1D)	(None, 48, 64)	640
max_pooling1d_2 (MaxPooling1D)	(None, 24, 64)	0
dropout_1 (Dropout)	(None, 24, 64)	0
conv1d_3 (Conv1D)	(None, 22, 128)	24,704
max_pooling1d_3 (MaxPooling1D)	(None, 11, 128)	0
dropout_2 (Dropout)	(None, 11, 128)	0
lstm (LSTM)	(None, 11, 100)	91,600
lstm_1 (LSTM)	(None, 50)	30,200
dense_2 (Dense)	(None, 50)	2,550
dropout_3 (Dropout)	(None, 50)	0
dense_3 (Dense)	(None, 4)	204
Total param #		**449,696**
Trainable param #		**149,898**

Table 4.7: Layer-wise details of the CNN-LSTM model.

This model yielded good accuracy results with an average per-class accuracy of 82%. Only ”other social signals” have a relatively lower accuracy than the rest at 67%. The per-class accuracies are visualised in Figure 4.3.

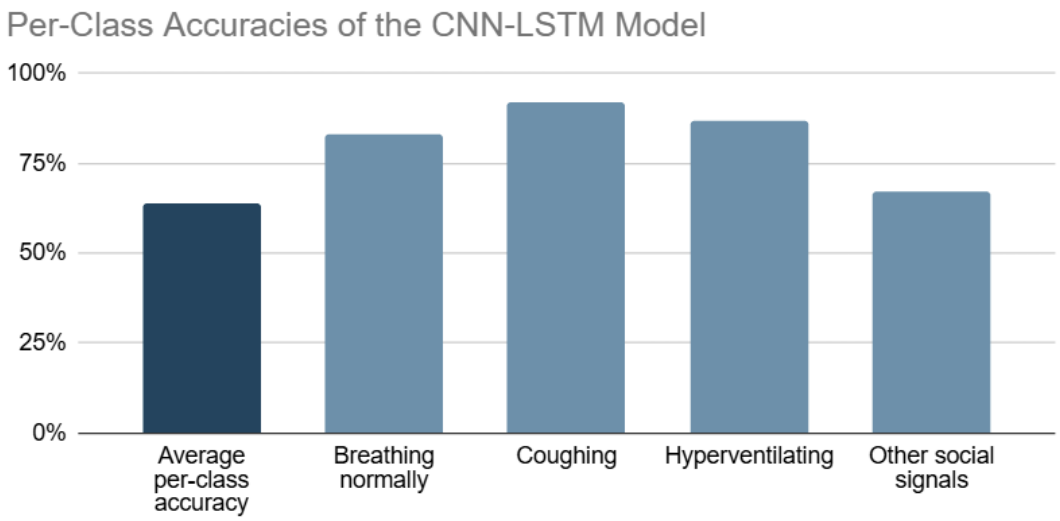

Figure 4.3: Per-Class Accuracies of the CNN-LSTM Model

We tried adding summary features per time window to the data fed into the model. The summary features were: Mean, standard deviation, minimum, maximum, skewness, and kurtosis. The promise of summary features is that they capture global/distributional properties of an entire time window which might otherwise only be understood in a piecemeal fashion by the model [Wang et al., 2017]. Again, we trained for 50 epochs with batch size 32 and we halved the learning rate when the validation loss plateaued. However, we did not achieve a higher per-class accuracy, rather, the average accuracy fell by two percentage points (to 80%). A comparison of the per-class accuracies of the model with and without summary features can be seen in Figure 4.4.

A possible reason for this slight drop in performance might be that the summary features diverted the model's attention away from the piecemeal temporal sequence that it already 'understood' well. If the summary features are uninformative or confusing, this shift in attention would be adverse. One could test this hypothesis by doing dimensionality reduction, e.g. PCA, on the summary features and checking whether the features seem informative or all over the place/confusing.

Next, we tested a similar architecture with bidirectional LSTMs. Bidirectional LSTMs can in theory capture temporal context both forward and backward. The detailed layers are laid out in Table 4.8.

However, this model performed much worse. We tried improving performance by adding class weights to counterbalance the differing numbers of samples per class. However, the model still overpredicted the classes "Breathing normally" and "Coughing", leading to lower accuracies for "Hyperventilating" (58%) and "Other social signals" (51%). Clearly, the larger numbers of parameters introduced by the bidirectional LSTMs did not help much here. The per-class accuracies compared to our CNN with unidirectional LSTMs are visualised in Figure 4.5.

To balance out a higher number of parameters, we tried our third and ultimately best architecture which uses Batch Normalisation on each convolutional layer. This model

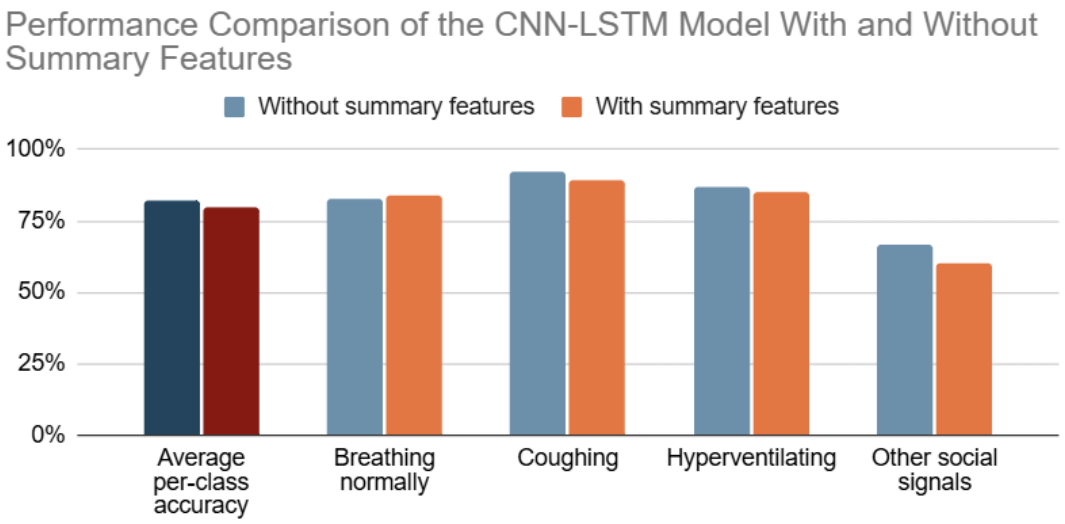

Figure 4.4: Performance Comparison of the CNN-LSTM Model With and Without Summary Features

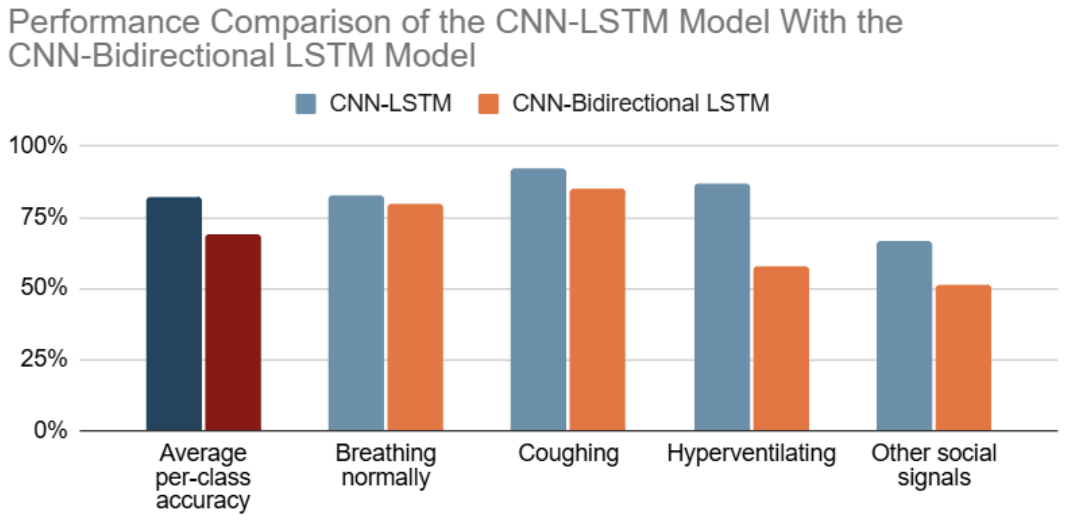

Figure 4.5: Performance Comparison of the CNN-LSTM Model With the CNN-Bidirectional LSTM Model

Layer (type)	Output Shape	Param #
conv1d_4 (Conv1D)	(None, 46, 128)	2,048
max_pooling1d_4 (MaxPooling1D)	(None, 23, 128)	0
dropout_4 (Dropout)	(None, 23, 128)	0
conv1d_5 (Conv1D)	(None, 21, 256)	98,560
max_pooling1d_5 (MaxPooling1D)	(None, 10, 256)	0
dropout_5 (Dropout)	(None, 10, 256)	0
bidirectional (Bidirectional)	(None, 10, 300)	488,400
bidirectional_1 (Bidirectional)	(None, 200)	320,800
dense_4 (Dense)	(None, 100)	20,100
dropout_6 (Dropout)	(None, 100)	0
dense_5 (Dense)	(None, 4)	404
Total param #		**2,790,938**
Trainable param #		**930,312**

Table 4.8: Layer-wise details of the CNN model with bidirectional LSTMs

was described in Section 3.3. We simplified the two bidirectional LSTM layers to one unidirectional one because it did not lower performance. This lowered the parameter number significantly which can reduce susceptibility to overfitting [Zhang et al., 2021]. After tuning the hyperparameters of this final architecture with a random search, we achieved an average per-class accuracy of 85%. A performance comparison of all three of our CNN architectures is shown in Figure 4.6.

The high accuracy results indicate that L2 Regularisation worked better to counteract the class imbalances than class balancing did. We tested adding class weights back in but this did not make a difference for accuracy.

Overall, proper normalisation and regularisation seem to have made the biggest difference to accuracy. High parameter numbers did not necessarily produce higher accuracy and may have led to overfitting. Class imbalances affected performance but could be mitigated using L2 Regularisation.

4.2 Accuracy of Real-Time Classification

We applied these models in real-time for two main purposes: (1) dual-input classification for daily activities using data from both Respeck and Thingy sensors, and (2) social signal classification, which also relied on data from the Respeck sensor. These models ran in parallel on the live PDIoT app, connected to the sensors via Bluetooth, with a real user wearing the sensors. To assess the model's accuracy in a real-world setting, each daily activity and social signal was performed 10 times, with a latency allowance of 5 seconds for classification. The sensor data was recorded from the user at a frequency of 25Hz, and to improve classification accuracy, we used non-overlapping windows of 50 steps, allowing for two windows to be processed per classification. The results from

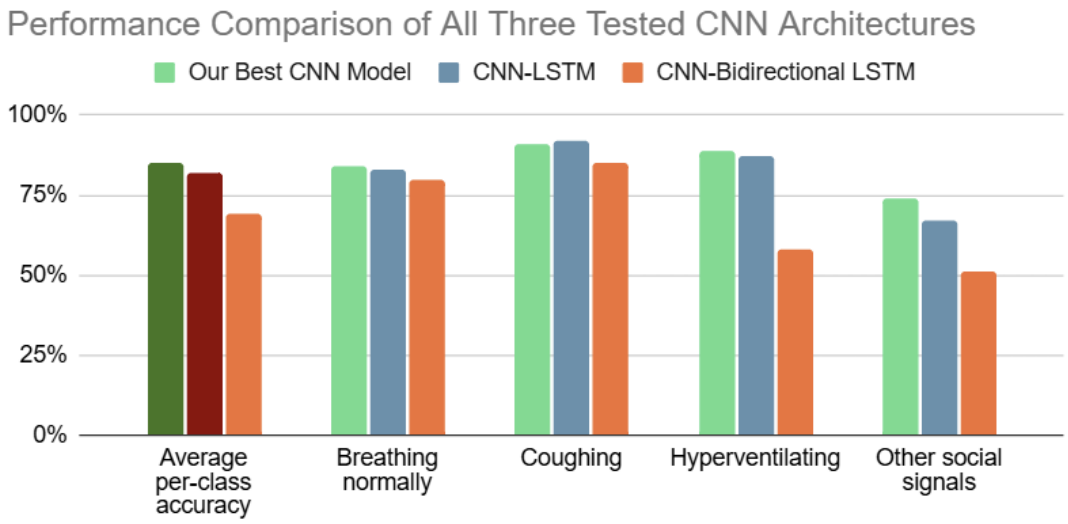

Figure 4.6: Performance Comparison of All Three Tested CNN Architectures

this real-world test are depicted in Table 4.9. This setup allowed us to observe the entire IoT system's performance in real-life conditions, where it successfully identified daily activities and social signals with high accuracy, demonstrating the system's robustness and practical applicability in continuous real-time monitoring.

Stationary Activities	**Accuracy**
Sitting / Standing	90%
Lying on Back	100%
Lying on Left	100%
Lying on Right	100%
Lying on Stomach	100%

Dynamic Activities	**Accuracy**
Miscellaneous Movement	80%
Normal Walking	90%
Running	90%
Shuffle Walking	80%
Ascending	100%
Descending	100%

Social Signals	**Accuracy**
Breathing Normal	80%
Coughing	100%
Hyperventilation	100%
Other	70%

Table 4.9: Results from Real-World Classification using the IoT System

The model showed exceptional performance in classifying static activities, with "Lying on Back," "Lying on Left," "Lying on Right," and "Lying on Stomach" achieving 100% accuracy. It also performed perfectly for dynamic activities like "Ascending" and "Descending." However, the lowest performance was observed with "Other" in the social signals category, which only reached 70% accuracy. Despite this, the model's performance across the majority of activities, especially with 80–100% accuracy for the rest, demonstrates its strong ability to classify real-life data.

4.3 Android App Performance

Communication Latency: Approximately 2 seconds

This is the time it takes to collect a single data window (50 timesteps at 25 Hz). Both classifiers require a full data window to update their classification. Other latencies from app processes are negligible and not noticeable to the app user. The measurement was made empirically with a stopwatch while using the app on a Samsung A20e phone and held up to several retrials.

Power consumption: Increases the phone's power consumption by ca. 40%

This measurement was also made empirically using a Samsung A20e phone. A comparison was made between keeping the phone on for three hours with and without the app running (in the live classification screen).

An increase of 40% in power consumption is substantial and could hinder the use of the app for long stretches of time. Approaches to increase power efficiency could be to quantize our models to reduce their size or to smartly reduce the sampling frequency of the sensors when the classification isn't currently changing.

CPU usage: ca. 15%

In the live classification screen, the app uses ca. 1.5 seconds of CPU time over 10 seconds of wall clock time. This was determined by using the Android Studio Profiler. Several 10-second CPU traces were recorded and their total CPU times were averaged.

15% of CPU time is only a moderate load but it could be further lowered by quantizing models and smartly reducing the sampling frequency of the sensors.

Memory usage (RAM): ca. 80 MB

The app uses relatively little RAM at 80 MB compared to the total RAM of typical smartphones (between 3 and 16 GB). This measurement was made using the Android Studio Profiler. The real-time RAM usage graph was analysed visually to determine the RAM usage stabilises at roughly 80 MB a few seconds after starting the live classification screen.

Chapter 5

Conclusions

5.1 Summary and Reflection

The development of this IoT system for classifying daily activities and social signals involved several key stages. The process began with data collection from Respeck and Thingy sensors, followed by data cleaning to ensure suitable input quality for model training. Two machine learning models were trained on this data, targeting daily activity and classification of social signals, with the goal of accuracy exceeding 90% for static and dynamic activities and 80% for social signals. K-fold testing was also conducted to validate these models, yielding results consistent with or higher than those achieved during development.

The system was integrated into a mobile application, incorporating features such as sensor connectivity, live data visualization through dynamic graphs, real time classification and labeling of actions, and the ability to access historical classification data for specific dates. Real-life testing of the integrated system demonstrated an average accuracy of 98% for daily activities, 90% for dynamic activities, and 85% for social signals, with all classifications operating in real time and within a 2-second latency window. In terms of system performance, the communication latency was measured at 2 seconds, with a 40% increase in power consumption, approximately 1.5 seconds of CPU time per 10 seconds of wall clock time (15%), and a memory usage of approximately 80MB RAM. These metrics indicate the system's ability to operate efficiently under real-world conditions as a robust and reliable IoT system.

5.2 Areas for Future Work

Future improvements to the IoT system could focus on several aspects. Enhancing the classification accuracy of social signals, particularly for the ”Other” category, would be a priority to improve its reliability in diverse real-life scenarios. Expanding the system's functionality to include additional activities and social signals could make it applicable to a broader range of use cases. Implementing overlapping windows or adaptive window sizing could further improve classification performance for dynamic activities.

Additionally, incorporating functional scalability requirements is essential for ensuring the system's adaptability to larger deployments. Derived from a study on large-scale IoT systems, these requirements include explicit control flow, distributed workflows, location transparency, decentralized data flows, and the separation of control, data, and computation [Arellanes and Lau, 2020]. Meeting these requirements would improve the system's scalability, facilitate integration with other IoT platforms, and enable efficient data processing across distributed environments. Finally, transitioning to a cloud-based architecture for data storage and processing could support real-time analytics on a larger scale while enabling seamless integration with existing IoT systems. [Bokefode et al., 2016]

Bibliography

Damian Arellanes and Kung-Kiu Lau. Evaluating iot service composition mechanisms for the scalability of iot systems. *Future Generation Computer Systems*, 108:827–848, 2020. URL https://www.sciencedirect.com/science/article/abs/pii/S0167739X19320291.

Yuhan Bai. Relu-function and derived function review. *SHS Web of Conferences*, 144:02006, 2022. doi: 10.1051/shsconf/202214402006. URL https://www.shs-conferences.org/articles/shsconf/pdf/2022/14/shsconf_stehf2022_02006.pdf. STEHF 2022.

Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994. doi: 10.1109/72.279181.

Jayant D. Bokefode, Avdhut S. Bhise, Prajakta A. Satarkar, and Dattatray G. Modani. Developing a secure cloud storage system for storing iot data by applying role-based encryption. In *Twelfth International Multi-Conference on Information Processing-2016 (IMCIP-2016)*. Elsevier, 2016. URL https://www.sciencedirect.com/science/article/pii/S1877050916310729/pdf?md5=21735dfde75fd0e389983cff514b8e45&pid=1-s2.0-S1877050916310729-main.pdf.

L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, October 2001. doi: 10.1023/A:1010933404324. URL https://doi.org/10.1023/A:1010933404324.

Stylianos Charalampous. Human activity and social signal classification using auxiliary classifier generative adversarial network and transfer learning. Master's thesis, University of Edinburgh, Edinburgh, United Kingdom, 2021. Available at: https://project-archive.inf.ed.ac.uk/ug4/20212442/ug4_proj.pdf.

Oscar Cleve and Sara Gustafsson. Automatic feature extraction for human activity recognitionon the edge, 2019.

Mats Folke, Lars Cernerud, Mats Ekström, et al. Critical review of non-invasive respiratory monitoring in medical care. *Medical & Biological Engineering & Computing*, 41(3):377–383, July 2003. doi: 10.1007/BF02348078. URL https://doi.org/10.1007/BF02348078.

Teodora Georgescu. Classification of coughs using the wearable respeck monitor. Mas-

ter's thesis, University of Edinburgh, Edinburgh, United Kingdom, 2019. Available at: https://project-archive.inf.ed.ac.uk/ug4/20191570/ug4_proj.pdf.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Commun. ACM*, 63(11):139–144, October 2020. ISSN 0001-0782. doi: 10.1145/3422622. URL https://doi.org/10.1145/3422622.

Zhenyu He and Lianwen Jin. Activity recognition from acceleration data based on discrete consine transform and svm. In *2009 IEEE International Conference on Systems, Man and Cybernetics*, pages 5041–5044, 2009. doi: 10.1109/ICSMC.2009.5346042.

L. C. Jain and L. R. Medsker. *Recurrent Neural Networks: Design and Applications*. CRC Press, Inc., USA, 1st edition, 1999. ISBN 0849371813.

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*, MM '15, page 1307–1310, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450334594. doi: 10.1145/2733373.2806333. URL https://doi.org/10.1145/2733373.2806333.

Artur Jordao, Antonio C. Nazare Jr., Jessica Sena, and William Robson Schwartz. Human activity recognition based on wearable sensor data: A standardization of the state-of-the-art, 2019. URL https://arxiv.org/abs/1806.05226.

Mucheol Kim, Ka L. Man, and Nurmamat Helil. Advanced internet of things and big data technology for smart human-care services. *Journal of Sensors*, 2019:3, 2019. URL https://www.proquest.com/scholarly-journals/advanced-internet-things-big-data-technology/docview/2187379877/se-2. Copyright - Copyright © 2019 Mucheol Kim et al. This is an open access article distributed under the Creative Commons Attribution License (the "License"), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited. Notwithstanding the ProQuest Terms and Conditions, you may use this content in accordance with the terms of the License. http://creativecommons.org/licenses/by/4.0; Last updated - 2023-11-28.

J. Korpáš, J. Sadloňová, and M. Vrabec. Analysis of the cough sound: an overview. *Pulmonary Pharmacology*, 9(5):261–268, 1996. ISSN 0952-0600. doi: https://doi.org/10.1006/pulp.1996.0034. URL https://www.sciencedirect.com/science/article/pii/S0952060096900344.

Oscar D Lara and Miguel A Labrador. A survey on human activity recognition using wearable sensors. *IEEE communications surveys & tutorials*, 15(3):1192–1209, 2012.

Steven M LaValle, Anna Yershova, Max Katsev, and Michael Antonov. Head tracking for the oculus rift. In *2014 IEEE international conference on robotics and automation (ICRA)*, pages 187–194. IEEE, 2014.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):

436–444, 2015. doi: 10.1038/nature14539. URL https://doi.org/10.1038/nature14539. Issue date: 2015-05-28.

Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou. A survey of convolutional neural networks: Analysis, applications, and prospects. *IEEE Transactions on Neural Networks and Learning Systems*, 33(12):6999–7019, 2022. doi: 10.1109/TNNLS.2021.3084827.

Abdulmajid Murad and Jae-Young Pyun. Deep recurrent neural networks for human activity recognition. *Sensors*, 17(11), 2017. ISSN 1424-8220. doi: 10.3390/s17112556. URL https://www.mdpi.com/1424-8220/17/11/2556.

Derek A. Pisner and David M. Schnyer. Chapter 6 - support vector machine. In Andrea Mechelli and Sandra Vieira, editors, *Machine Learning*, pages 101–121. Academic Press, 2020. ISBN 978-0-12-815739-8. doi: https://doi.org/10.1016/B978-0-12-815739-8.00006-7. URL https://www.sciencedirect.com/science/article/pii/B9780128157398000067.

Daniele Ravi, Charence Wong, Benny Lo, and Guang-Zhong Yang. Deep learning for human activity recognition: A resource efficient implementation on low-power devices. In *2016 IEEE 13th International Conference on Wearable and Implantable Body Sensor Networks (BSN)*, pages 71–76, 2016. doi: 10.1109/BSN.2016.7516235.

Bernd Tessendorf, Franz Gravenhorst, Bert Arnrich, and Gerhard Tröster. An imu-based sensor network to continuously monitor rowing technique on the water. In *2011 seventh international conference on intelligent sensors, sensor networks and information processing*, pages 253–258. IEEE, 2011.

Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks: A strong baseline. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 1578–1585, 2017. doi: 10.1109/IJCNN.2017.7966039.

Mitchell Webber and Raul Fernandez Rojas. Human activity recognition with accelerometer and gyroscope: A data fusion approach. *IEEE Sensors Journal*, 21(15):16979–16989, 2021.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. *Commun. ACM*, 64(3):107–115, February 2021. ISSN 0001-0782. doi: 10.1145/3446776. URL https://doi.org/10.1145/3446776.