PDIoT Coursework 3 (2023-24)

Daniel Cooper(S2026670), Sean Choi(S2101367), Daniel Kim(S2017594)

November 2023

Abstract

RespFit is a mobile application that performs HAR (Human Activity Recognition). It can classify different movements (walking, jogging, shuffle-walking, etc) and respiratory symptoms (normal breathing, coughing, hyperventilating, etc). This application is built for Android devices and uses a RESpeck sensor to collect data from the user. We classify raw data using state-of-the-art neural network models such as CNNs (Convolutional Neural Networks) [16] and LSTM (Long-Short-Term Memory) [12] along with custom pre-processing of data into their magnitudes, derivatives, and other features. Additionally, *RespFit* can handle multiple users - users can register or log on, connect their sensors, perform activity classification, and see their activity history. *RespFit* is also connected to Google's free RealTimeDatabase, which provides a seamless and instant user authentication experience. Overall, *RespFit* is a useful tool for both patients and their carers. With the app's simple design, it is extremely intuitive to navigate through the pages and monitoring one's physical activities has never been easier.

1 Introduction

1.1 Project Aims

Our project aims to perform HAR (Human Activity Recognition). Throughout the years, many researchers have attempted HAR using videos [15] [24] and sensors [5], and we have chosen sensors, namely RESpeck sensors, to perform HAR. Another aspect of our project is to incorporate the HAR models into an Android app. The app allows users to register and log in to their accounts, perform HAR on demand, and see their activity histories with pre-made and custom date filters. We connect a RESpeck sensor using BLE (Bluetooth Low Energy) to the app on an Android device. Furthermore, we aim to develop three models which achieve a threshold LOO (Leave-One-Out) cross-validation accuracy for the three tasks mentioned later.

To achieve these goals, we split these tasks appropriately among the three members of our group. One of us took on the mobile application, integrated the models with the app, and set up the database, while the other two focused on feature engineering and model development.

1.2 Methodology Overview

To perform the real-time classification of tasks as presented in section 1.3 we adopted a method that involved splitting the data into windows of size 50, then exacting a series of features such as the Jerk and Fast Fourier transform from the columns of each window. These data windows are then used by two separate models, one for the classification of the physical activity, and another for the classification of the respiratory symptom. Each of these models is a deep neural network using a combination of sequential layers, such as CNNs (Convolutional Neural Networks) [16]. For the in-app model if the physical model believes that the user is conducting a non-stationary activity then we override the results from the respiratory classifier with normal breathing since this is the only option in these cases. Within the application but not in our offline classification we also perform signal smoothing for the classification of the physical activity, this helps to reduce real-world noise and makes it closer to our offline collected data. More details on the construction of these models and how we developed them can be found in section 4.

1.3 Activities Classification

Below is the list of physical and respiratory activities that our models can predict.

- Physical Activities:
  - Sitting/Standing
  - Lying down on right side
  - Lying down on left side
  - Lying down on stomach
  - Lying down on back
  - Walking
  - Jogging/Running
  - Shuffle Walking
- Respiratory Activities:
  - Normal breathing
  - Coughing (Only for stationary activities)
  - Hyperventilating (Only for stationary activities)
  - Other (Talking/Eating/Laughing/Singing) (Only for stationary activities)

These are then combined into three tasks for the evaluation of our model, these tasks are given by the production of the following classes:

- Task 1:
  - class 0: sitting/standing
  - class 1: lying down on your left side
  - class 2: lying down on your right side
  - class 3: lying down on your back
  - class 4: lying down on your stomach
  - class 5: walking
  - class 6: running/jogging
  - class 7: descending stairs
  - class 8: ascending stairs
  - class 9: shuffle walking
  - class 10: miscellaneous movements
- Task 2:
  - class 0: sitting/standing + breathing normally
  - class 1: lying down on your left side + breathing normally
  - class 2: lying down on your right side + breathing normally
  - class 3: lying down on your back + breathing normally
  - class 4: lying down on your stomach + breathing normally
  - class 5: sitting/standing + coughing
  - class 6: lying down on your left side + coughing
  - class 7: lying down on your right side + coughing
  - class 8: lying down on your back + coughing
  - class 9: lying down on your stomach + coughing
  - class 10: sitting/standing + hyperventilating
  - class 11: lying down on your left side + hyperventilating
  - class 12: lying down on your right side + hyperventilating
  - class 13: lying down on your back + hyperventilating
  - class 14: lying down on your stomach + hyperventilating
- Task 3:
  - class 0: sitting/standing + breathing normally
  - class 1: lying down on your left side + breathing normally
  - class 2: lying down on your right side + breathing normally
  - class 3: lying down on your back + breathing normally
  - class 4: lying down on your stomach + breathing normally

- class 5: sitting/standing + coughing
- class 6: lying down on your left side + coughing
- class 7: lying down on your right side + coughing
- class 8: lying down on your back + coughing
- class 9: lying down on your stomach + coughing
- class 10: sitting/standing + hyperventilating
- class 11: lying down on your left side + hyperventilating
- class 12: lying down on your right side + hyperventilating
- class 13: lying down on your back + hyperventilating
- class 14: lying down on your stomach + hyperventilating
- class 15: sitting/standing + other
- class 16: lying down on your left side + other
- class 17: lying down on your right side + other
- class 18: lying down on your back + other
- class 19: lying down on your stomach + other **Other refers to singing/talking/laughing/eating

2 Literature Review

2.1 Model Explanations

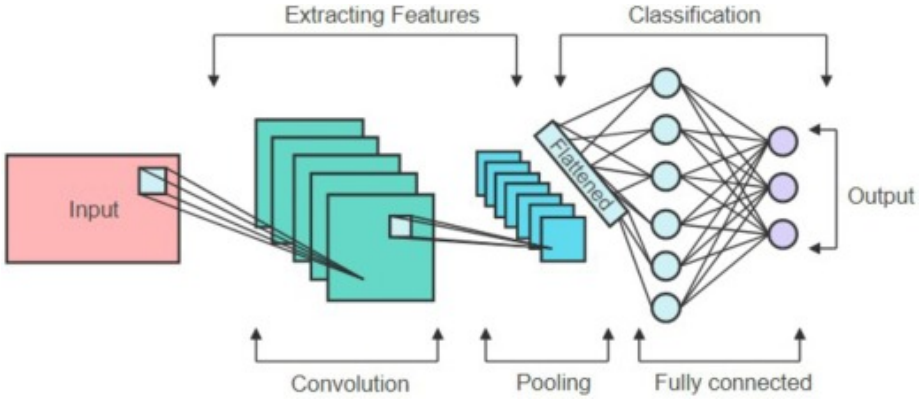


Figure 1: CNNs Model Structure
[14]

Convolutional neural networks(CNNs)1 are made of multiple layers; convolutional layer, pooling layer, activation layer, and fully connected layer. In a

convolutional layer, a kernel(i.e. a filter) traversed the input, such as a 1D signal in small segments. As it moves across the entire input field, it creates the convoluted feature map, which is an interpreted input. In a pooling layer, it produces the most essential elements by distilling the convolved features. Pooling layers reduce the computational power for processing the data. Max pooling and average pooling are examples of it. In an activation layer, activation functions help the network to improve the understanding of complex relationships between features. In a fully connected layer, all features come together for the final decision.[4]

In the case of HAR, when a sensor receives accelerometer and gyroscope signals at 25 Hz (i.e. 25 data points per second), elements vector of features after feature extractions with the received input signal data are fed into the CNN model. Firstly, the convolutional layers read time series signals using a kernel to get the convolved feature. And, the convolved feature is passed on to the next layer which detects more complex features. As the features move into deeper into the network, it can identify even more complex features. [18]

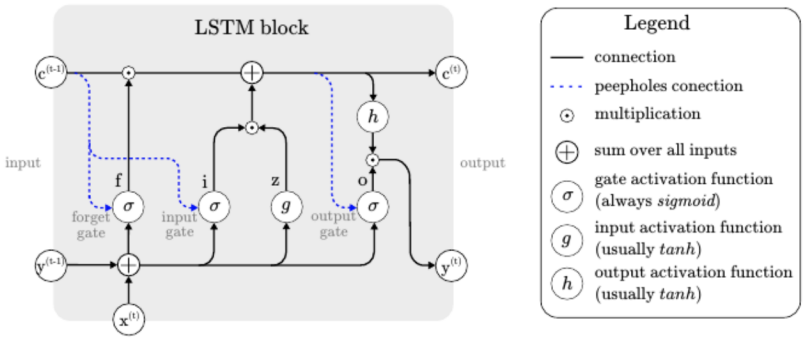


Figure 2: LSTM Model Structure
[26]

Long short-term memory(LSTM) 2 network is a type of recurrent neural network, developed to handle the vanishing gradient problem found in traditional RNNs. The vanishing gradient problem refers to the occurrence where the training process is getting slow as the gradients of the loss function are expected to decrease to zero when the sequence length increases. [28]

A common vanilla LSTM unit is comprised of a cell, an input gate, an output gate, and a forget gate. The cell remembers values over random time intervals and the information flow into and out of the cell is regulated by the three gates. The decision of what information to abandon from a previous state occurs at forget gates with a value between 0 and 1. A (rounded) value of 1 refers to keeping the information, whereas a value of 0 refers to abandoning it. Input gates decide which pieces of new information will be stored in the current state, employing the same systems as forget gates. Output gates regulate the

flow of information from the current state by assigning values between 0 and 1 to determine which piece of data to release, taking into account both the previous and current states. The output-relevant information ensures the LSTM network has useful and long-term dependencies for predictions, both in current and future time steps. [27]

2.2 Related Works

In this chapter, we introduce several papers in the context of Human Activity Recognition(HAR) for our software. Initially, we conducted a survey to encompass methodologies for HAR, and subsequently, we refined these methodologies based on several criteria, including the size of the dataset, the number of classes, creativity, and model performance.

[11] shows a summary of recent state-of-the-art HAR research utilising deep learning. This survey introduced 35 deep learning-based HAR studies conducted between 2014 and 2020. Among these studies, Convolutional Neural Network (CNN) was used in 10 studies, Long short-term memory(LSTM) in 10, Deep belief network(DBN) in 7, Statistically adjusted engineering(SAE) in 5, and other methods in 2. In the case of combined models like CNN + LSTM, we treated each model separately for counting. Most studies using CNN or LSTM demonstrated promising results. Furthermore, the recent [23] survey presents similar findings to the previous [11] survey, with 14 studies from 2018 to 2021. The majority of these studies employed LSTM and LSTM + CNN for HAR tasks with average LSTM and LSTM+CNN model performance exceeding 95%, which is quite impressive. Based on these recent studies, we have chosen to focus on CNN and LSTM models for our research.

Among the numerous papers discussing the use of CNN, LSTM or CNN+LSTM for HAR tasks, we aimed to find the models with similarities to our tasks for optimal performance. First, the dataset size should be sufficient or similar to our instance count. Our dataset consists of 4,448 instances, representing 102 student engaged in 44 activities, including physical activities, stationary activities, and respiratory symptoms. Additionally, the number of classes should align with our task. The study in [13] appears to meet these requirements. For task 1 (physical activities), we have eleven classes, and for task 3 (physical activities + respiratory symptoms), we have twenty classes. In [13], the authors used four datasets, and three of which closely resemble our task environments: OPPORTUNITY, UniMib-SHAR, and PAMAP2. PAMAP2 has 12 classes such as walking, cycling, and rope jumping, whereas OPPORTUNITY and UniMib-SHAR contain 17 classes. Unfortunately, the paper does not provide information on the sample sizes for each dataset. The baseline model proposed in the [13] consists of a 3-layer CNN, and the study presents modified models based on this baseline. In the context of HAR, various CNN-based models yield average accuracy rates of around 90% for PAMAP2, around 80% for OPPORTUNITY, and around 75% for UniMib-SHAR. These results suggest that a layer of CNN model structure could be suitable for our task.

Furthermore, the paper [29]has shown a different model structure using CNN

and LSTM. They employed UCI-HAR, WISDM, and OPPORTUNITY datasets. The UCI-HAR dataset contains 6 activities with around 10,000 instances, while WISDM has 6 activities with around 16,000 instances. OPPORTUNITY includes 17 activities with 52,000 instances. Although these datasets have a higher number of instances than ours, the paper suggested an effective model structure. It demonstrated that CNN can be used in combination with LSTM, where each model can have multiple layers. In [29], a two-layer LSTM followed by a two-layer CNN was suggested. The F1 score of this model reached 95.78%, 95.85 %, and 92.63% on the UCI-HAR, WISDM, and OPPORTUNITY datasets, respectively. This paper inspired us to experiment with various combinations of CNN and LSTM layers, leading to the identification of an optimal model structure.

[2] details the process of generating various features from the raw data. Notably, they have chosen the mean and standard deviation of the non-transformed values, as [7] and [25] have demonstrated their effectiveness in activity recognition. Additionally, they selected median, lower quartile, upper quartile, skewness, and kurtosis for feature extraction. [30] adopted a different approach, extracting features from raw accelerometer signals in the x, y, and z axes. They generated magnitude signals and jerk-xyz signals, including the magnitude of the jerk, which is the derivative of accelerometer signals. These studies highlight the potential for meaningful feature generation from our XYZ accelerometer signals.

HAR data is time-series in nature. To maintain the temporal relationship between signal data points, windows with overlap were created, but the overlap percentage varied across studies. [30] used a 66% overlap for windows, while [29] and [13] used a 50% overlap. In our study, we experimented with 25% up to 150% overlap, finding that a 50% overlap yielded the best performance. A detailed explanation is provided in the results section.

3 Methodology

3.1 Hardware and Firmware

Our application can run on any mobile or tablet devices that support Android which has a minimum SDK version 24 and a minimum compile SDK version 30. Furthermore, we use the RESpeck sensor to perform activity classification which can be used as a plug-and-go hardware. The RESpeck sensor is attached to the lower left-hand side of the user's chest below their ribs 3, and the types of collected data can be seen below. Moreover, we were given the option to use the Nordic Thingy sensor [19] in conjunction with the RESpeck sensor. The Thingy sensor also records X, Y, and Z accelerometer and gyroscope data, as well as magnetic field data. However, we opted to only use the data from the RESpeck sensor as we deemed magnetic field data to not provide us with sufficient information about each activity. Furthermore, this can be disrupted by other electronic devices in the sensor's vicinity.

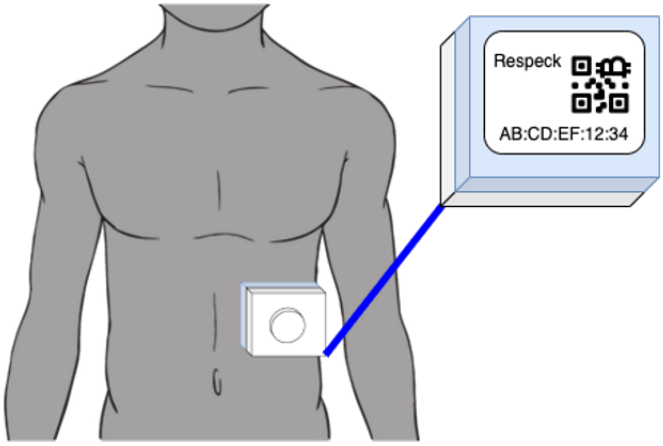


Figure 3: RESpeck sensor location

3.2 Wireless Communications

Our application communicates with the RESpeck sensor through Bluetooth Low Energy, which requires significantly lower power consumption when compared to the standard Bluetooth [6]. We collected data at a frequency of 25Hz, which meant that every second, we took in 25 data points consisting of these values:

- X Accelerometer
- Y Accelerometer
- Z Accelerometer
- X Gyroscope
- Y Gyroscope
- Z Gyroscope

3.3 Activity Detection Algorithm

We use two models for activity detection in our app. The first is a simpler, two-layer CNN (Convolutional Neural Network), and the latter is a combination of different CNNs which result in a decision tree of classifications. Our application follows these processes:

1. Take in raw data.
2. Pre-process into extracted features.
3. Run model one.

4. If the output is a non-stationary activity 1.3, display the activity on-screen.

5. Else, run model two.

6. Take the sub-activity output of model two and combine it with model one to give the final prediction of stationary activity.

A problem we ran into was that by running both models, we were using unnecessary amounts of CPU power. In turn, we found that the user experience deteriorated due to unresponsive and sluggish buttons. Therefore, by adopting this process, we were able to make our application more efficient and responsive (15% vs 9%).

3.4 Mobile Application

We used the skeleton app given to us to develop our application. We then removed code bloat, such as the activity recording page and the onboarding screen. This ensured that we only dealt with essential functionalities. Furthermore, we prioritised an efficient system which ticked all the requirements rather than dedicating our time to complex UI features as we realised that a significant portion of users for this app could be the elderly, who would be less knowledgeable about technology.

When designing the pages for the functionalities, an important metric that we considered was the number of clicks to get to a desired page. In our finished software, we report that it only takes six button clicks for a new user to register, connect their sensors, and start performing activity classification. Furthermore, when a user is already registered and the app has already been connected to a sensor, the user only clicks three times to perform classification.

3.4.1 Database Management

Our application is connected to the Google Firebase Realtime database [10], which uses NoSQL. We decided to use this database over other databases, as we had no prior experience in SQL and felt more familiar with Google's JSON-esque approach to database management. We created two tables in our database. The first table stores user information, such as usernames, passwords, and email addresses, whilst the second stores user activity information. These are shown in 1 and 2.

Key:	User Email
Values:	Username
	Salted and Hashed Password
	Phone Number
	Emergency Phone Number

Table 1: Database storing user information

Key	User Email
Values:	Activity
	Timestamp

Table 2: Database storing user activities

We store users' passwords by salting and then hashing them using the SHA-256 hash function in the jBCrypt [22] library to ensure user privacy and security. Of course, this would not matter in a coursework setting, but we wanted to take a step further and think about our design choices if our application was published to the general public. Thus, we set security to be one of our stretch goals.

For user activities, we stored these in a separate table to avoid a cluttered single table. The individual data we store contains the activity and its respective timestamp. While testing our application, we soon realised that our database would get easily clogged with unnecessary information. For example, if a user starts the model and performs the same activity, our application would continuously write the activity to the database every time.

Our solution came about as we iterated on it. First, we introduced a caching-based system, where we would write to a cache of the current predicted activities stored in-app. When the user left the prediction page, our application would write all the activities in the cache instead of writing them individually. Second, we realized that many redundant activities were being written into the database. Thus, we introduced a check while in the prediction loop, where if the current activity was the same as the previous activity, we would not write that into our cache. This resulted in a more efficient application using less CPU (average CPU usage of 12% without caching vs. 9% with caching and duplicate checking).

3.4.2 Main Page

When the user launches the app, they are greeted with this page 4. We used a simple white, light blue, and purple colour scheme. From here, users can log in if they are registered or create a new account by pressing the `Register` button. We also perform extensive input validation to ensure that we only store clean data in our database.

3.4.3 Hub Page

After the user logs in, they are greeted with this main hub page 5a. They can either view their activity history, watch the live model prediction, or connect their RESpeck sensors to the app. Furthermore, the action button on the top right allows them to go to their profile page, which is shown at 5b. Here, users can change most of their details except for email addresses. This is due to the limitations of the database, and the fact that we would need to transfer over all their existing activities and information to a different dictionary.

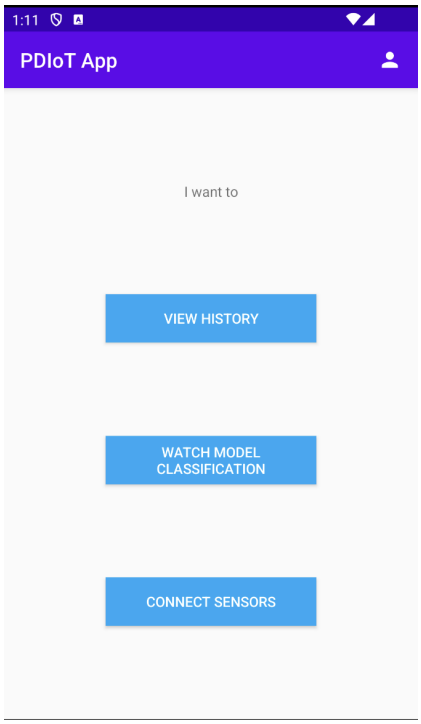
Figure 4: Initial login/register page

3.4.4 Connect Sensor Page

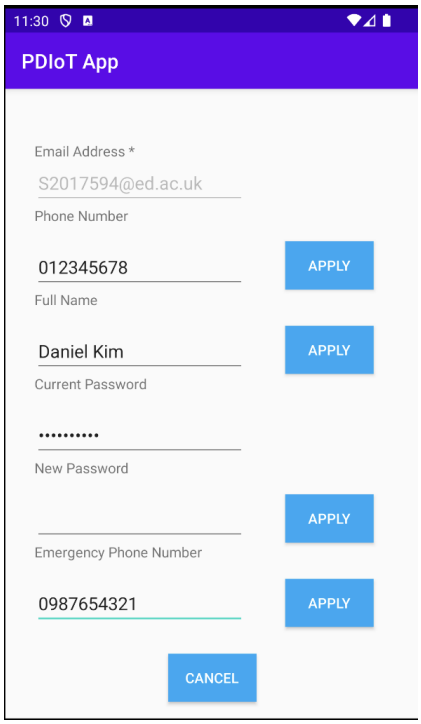
This page was taken from the skeleton app, retaining its prior functionality of connecting RESpeck and Thingy sensors by using either their QR codes or manually inputting their MAC addresses.

3.4.5 Model Prediction Page

Once the user has made an account, altered any incorrect personal details, and connected their RESpeck sensors, they are now ready to start classifying human activities. 7a shows the model classification page. As for the design choice, again, we pursued a simplistic yet functional style. Furthermore, we provided users with even more agency by allowing them to start and stop model recording to their needs. When they press the start button, the connected sensor receives raw data, our application pre-processes them (more on this below), and feeds the extracted features into the models laid out in 4.2.



(a) Hub page



(b) Profile page

3.4.6 View History Page

As mentioned previously, we anticipate that a significant portion of our users would be those who might be less experienced in using technology. Therefore, we decided to make the history page as simple and clutter-free as possible.

First, we offer three pre-made filters (Daily, Weekly, and Monthly) 8a that users can click to view their respective histories. Furthermore, they can fine-tune their search by clicking on the `Advanced` button, where they will be sent to this page 8b.

Our application displays the results from the pre-made filters immediately thanks to caching these values upon start-up. This significantly improved the load times of the history page and decreased the load on the CU by around 2%. Further changes could be made to optimise this page even further, and we discuss this in the later section.

3.5 Software Organisation

3.5.1 Task Management

From the beginning, we realised that we should adopt an agile approach to software development [8]. The agile methodology promotes constant and continuous

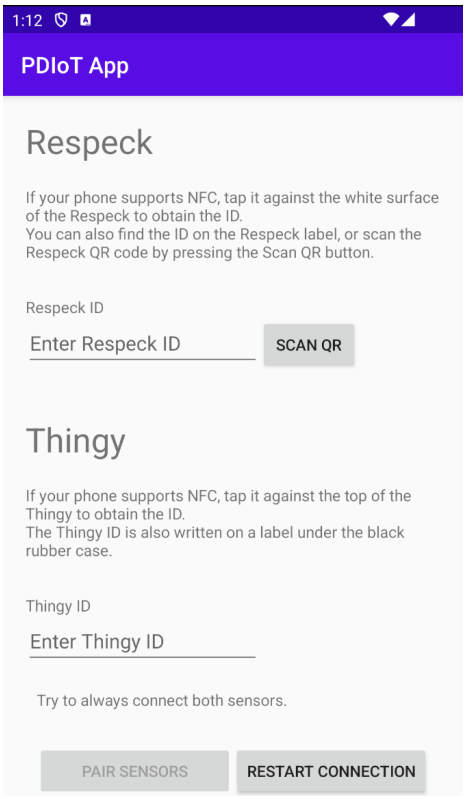


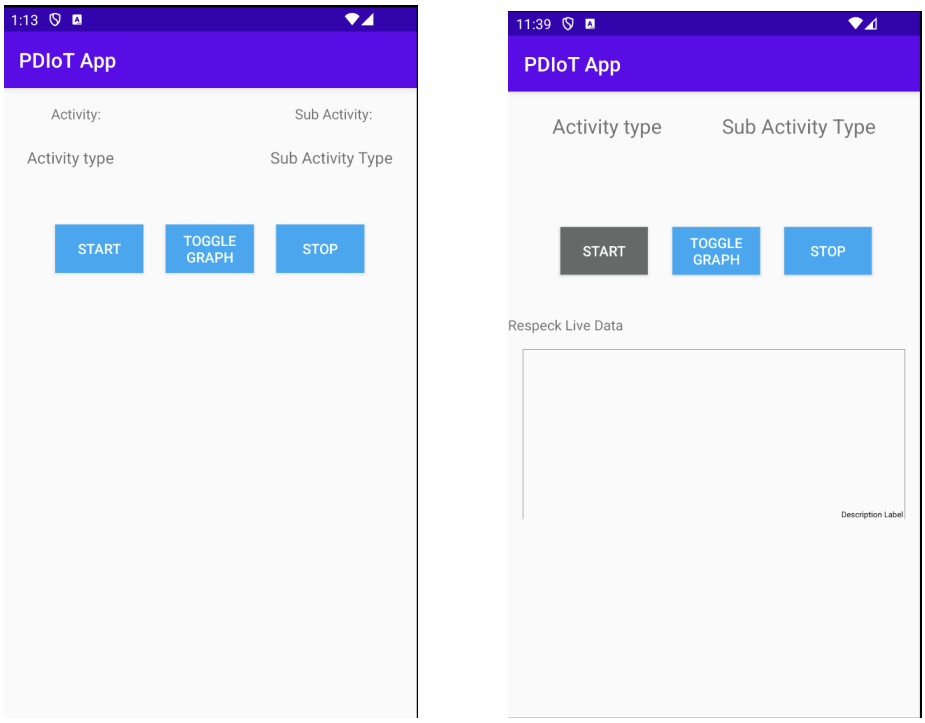
Figure 6: Connect Sensor page

software development through customer communication. Rather than its opposite, the waterfall method, we iterated on a prototype product which we knew had core functionalities and added any optional specifications at every sprint. To keep track of these components, we used Jira [1] to manage our workflow by splitting the major parts of the software into smaller, bite-sized tasks. Below is a screenshot of our Jira page, which shows the overview of our workflow 9, as well as a list of completed and accepted tasks 10.

3.5.2 Efficient Coding

Across our software, we kept efficient code and good coding practices always in mind. In particular, we focused on keeping a loose coupling and high cohesion between different classes or modules. In short, loose coupling between classes shows minimal dependency on each other, whereas high cohesion indicates that certain classes or modules should be grouped. Therefore, we went through multiple refactoring phases throughout development following the agile methodology.

Another broad principle we followed was abstraction. We aimed to abstract



(a) Model Prediction page

(b) Model Prediction page with graph

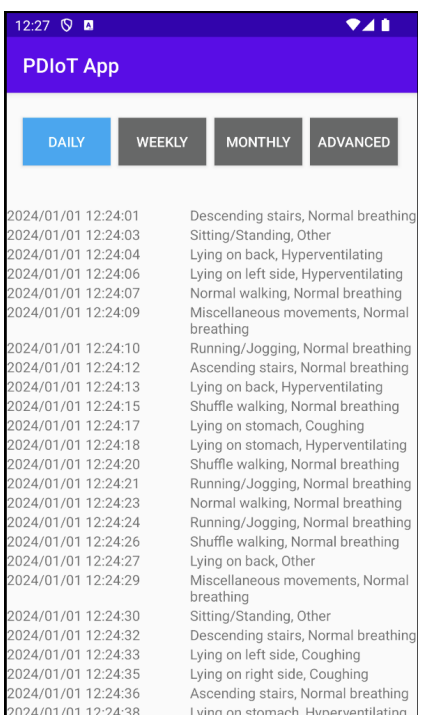
Figure 7: Variations of the Model Prediction page

away as many functions and properties as possible to ensure that classes were abstracted away and only showed the bare necessities. This resulted in us discovering classes which had more than one scope, leading to further refactoring phases.

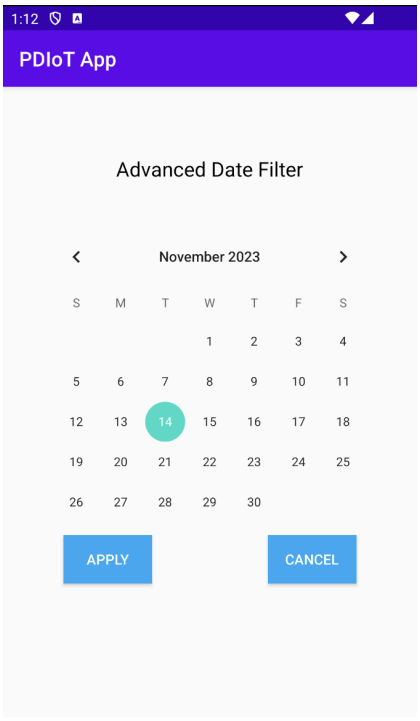
The UML diagram 11 shows the dependencies between classes present in our code. Each class carries out a single duty and utilises some design patterns laid out in *Effective Java* [3], such as Singleton and Repository.

For example, our initial iteration of the history page saw us deal with the database, data filtering, and UI rendering all in the same class `HistoryActivity`. However, this resulted in code that has low cohesion and high coupling. Therefore, we went through multiple refactoring phases. We separated the database functions from `HistoryActivity` into `ActivityDataRepository`, which utilises the Repository design pattern. This design pattern enables data centralisation (in this case, aggregating `ActivityData` objects), and provides an easy way to create functions which handle the data.

Another way that we made the code more optimal was by creating a `UserSession` object using a Singleton design pattern. A Singleton object is only initialised during the lifetime of a running software. The `UserSession` object contains



(a) History page



(b) Advanced History page

Figure 8: History pages

information about the current logged-in user and their previous activities to ensure that the backend correctly writes their predicted data in the database.

3.5.3 Software Flow

Here is a flow chart of our entire application 12. We have four distinct loops - connect sensors 14, user information 13, view history 16, and perform model predictions 15. Out of these, we spent the most time on the model prediction section, as it required integrating the models, backend, and frontend. A crucial component of our software was the model loop 15, where we perform these steps 3.3. Notably, we went through several design iterations when tackling the second and third stages, where we extract features from the raw data and pass them to the models.

Firstly, we tried the most straightforward way - to handle feature extraction using Kotlin within the app. Although this could have been possible had we had enough time, converting the functions used in Python to Kotlin proved to be very difficult, especially since we could not find libraries to perform certain aspects. However, we would have had the most efficient application if we had found success with this method.

h

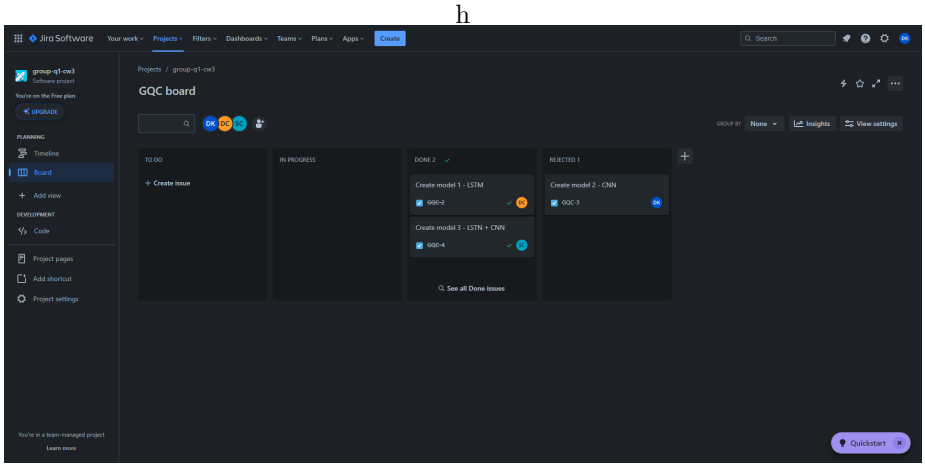


Figure 9: Jira overview

h

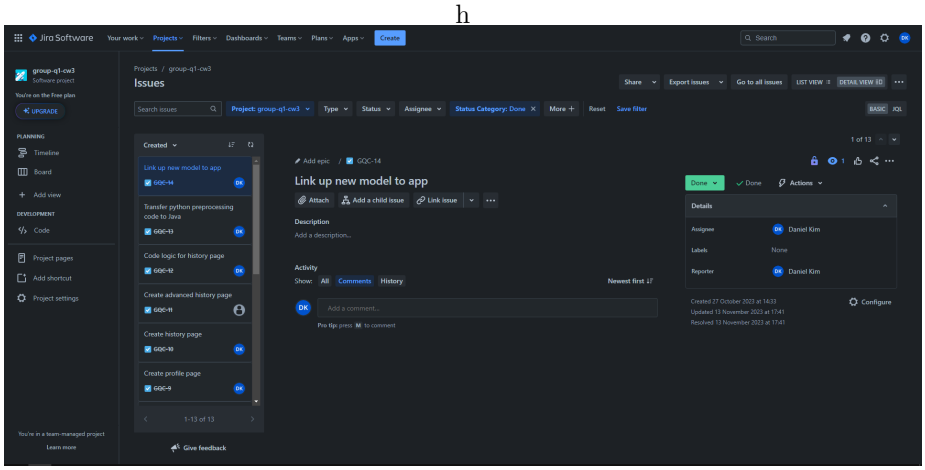


Figure 10: Completed tasks

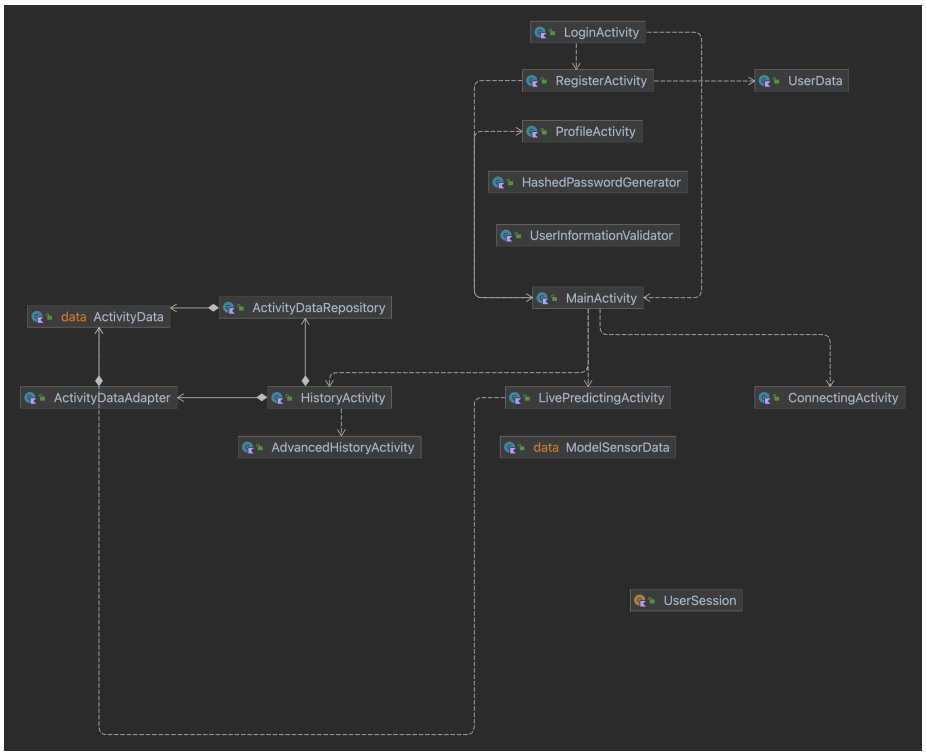


Figure 11: UML diagram

Next, we moved on to the cloud. Using Google's Cloud Computing Platform [10], we managed to host a remote server where we installed the feature extraction Python script. This did give us the result we wanted - we managed to link our app to the server and verified that we were indeed receiving feature packets. However, we found that the network latency between our application and server was too long at 0.2 seconds. Since we wanted to keep the feature extraction latency to < 0.01 seconds in the non-functional specifications, we had to find a better way.

Finally, we arrive at our latest approach which is embedded in the software. This combines the first and second methodologies, where we directly install a Python script into the application itself using the Chaquopy [17] library. This was useful in two ways - it bypasses the network latency problem. We calculated that feature extraction takes less than 0.01 seconds. Moreover, we did not need to write custom functions in Kotlin and instead used existing Python libraries to calculate certain features. However, we found that Chaquopy used a lot of memory. Comparing our implementation to our peers', we noticed that our memory usage was quite high as we had loaded in the Python script on app startup (350MB vs 150MB). Regardless, we believe that this is a fair trade-off between functionality and efficiency. Furthermore, loading the Python

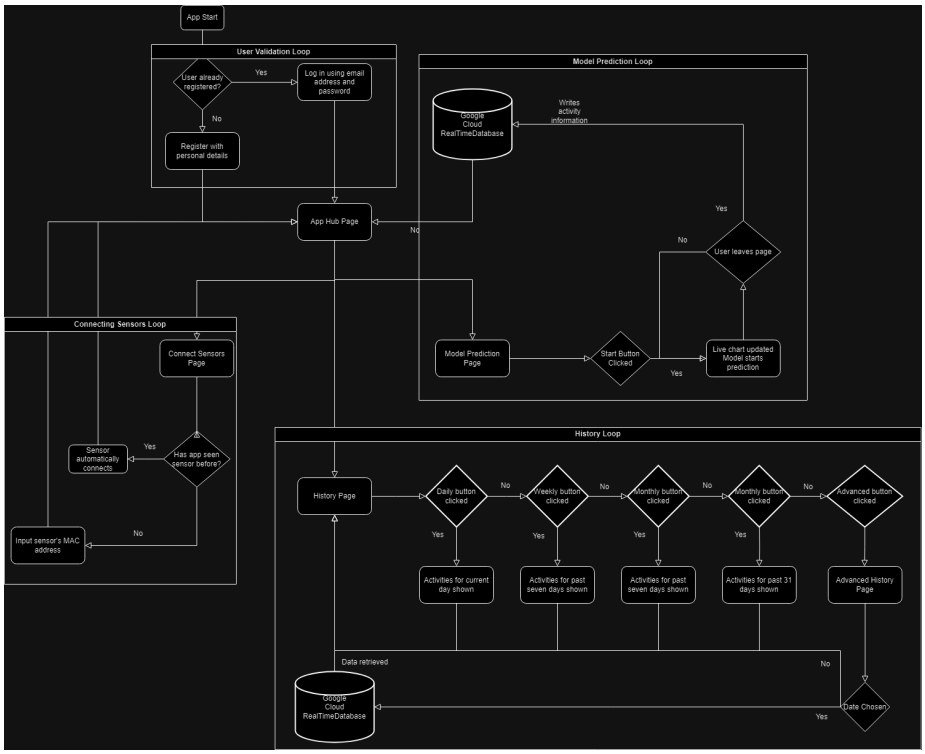


Figure 12: App flow overview

script at app start-up drastically reduced input latency when entering the model prediction page, which led to a better user experience.

3.6 Testing

When testing our models, we took four evaluation metrics into account: Precision, Recall, F1 Score, and accuracy. We chose these metrics, especially the F1 score since they are widely used to compare models' performance while better portraying extreme values of precision and recall. For example, if we simply calculate the mean of precision and recall for any given class, extreme values will be smoothed and will not give an accurate overview of the performance. On the other hand, if we use the F1 score, the use of harmonic means in its formula ensures that these extreme values are also considered, and thus affect the final score more.

3.6.1 Testing on different devices

In our group, we had access to a Samsung (CHECK DEVICE), and we carried out testing on this device. Furthermore, we verified the layout of our front end

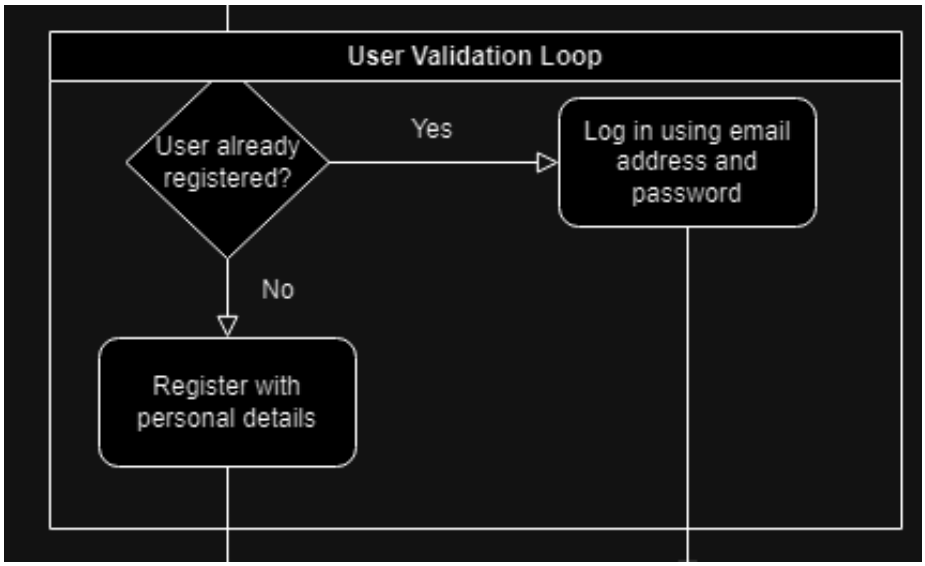


Figure 13: Login loop

using the `Layout Validator` feature in *Android Studio*. Therefore, we confirm that users will be able to have the same experience on our app regardless of their platform (Phone, Tablet, Desktop), thus increasing the accessibility and reach of our application.

3.6.2 Precision

Precision measures how good a model's predictions are. We look at everything the model has classified correctly (TP) out of everything the model has classified ($TP + FP$).

$$Precision = \frac{TP}{TP + FP}$$

3.6.3 Recall

On the other hand, recall looks at the ratio between the number of correct labels against the total number of labels returned. Looking at both metrics, we could theoretically have a model which could either have a precision or recall of 1, but this would come at the cost of the other metric.

$$Recall = \frac{TP}{TP + FN}$$

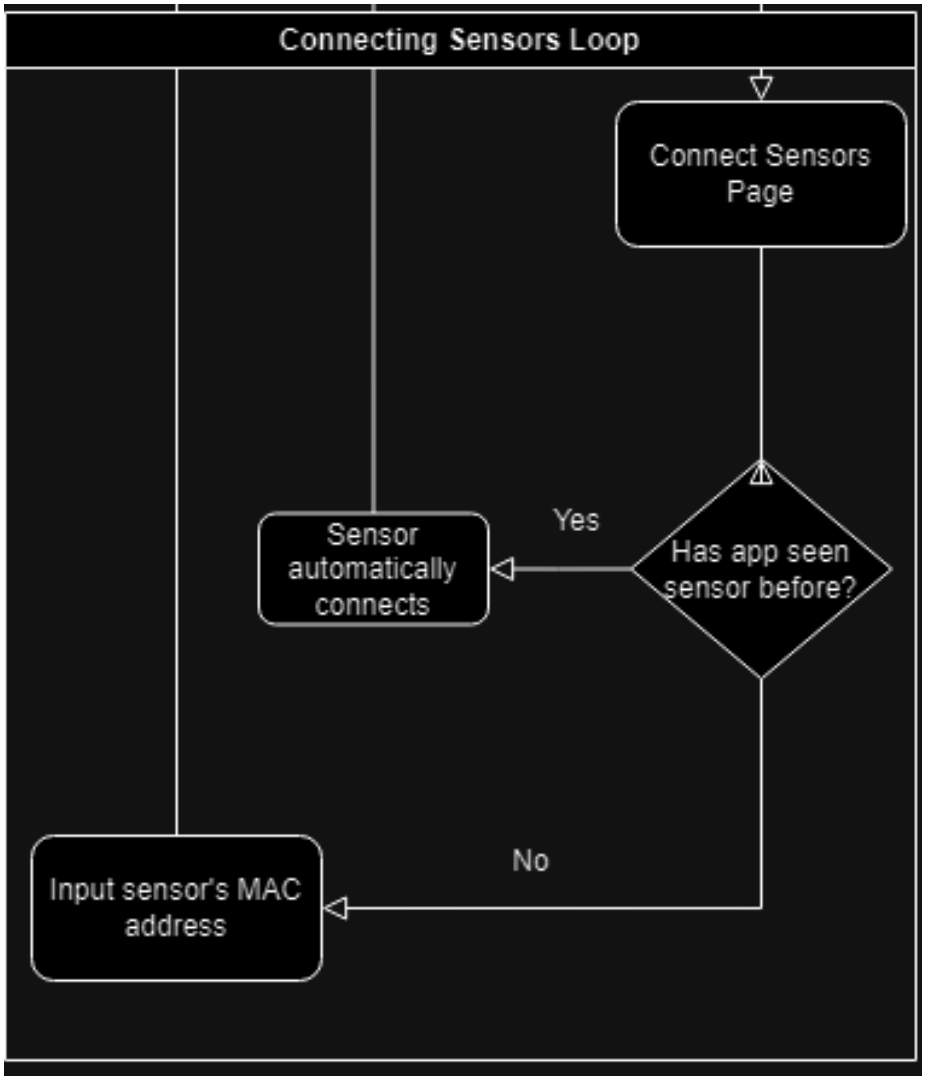


Figure 14: Sensor loop

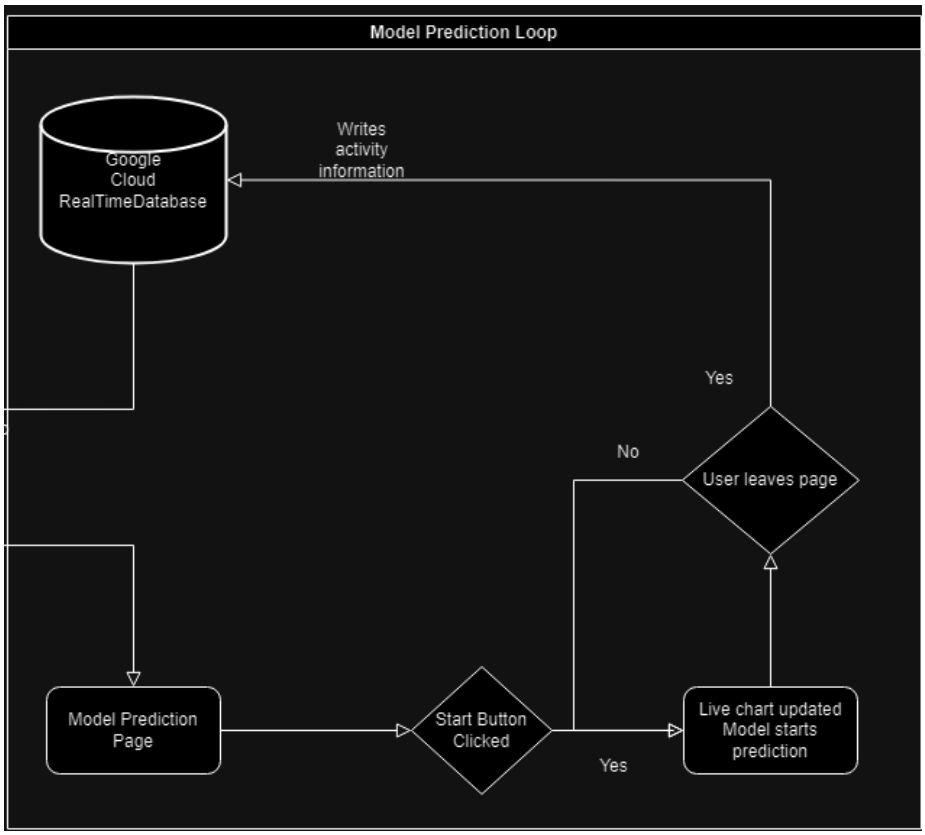


Figure 15: Model loop

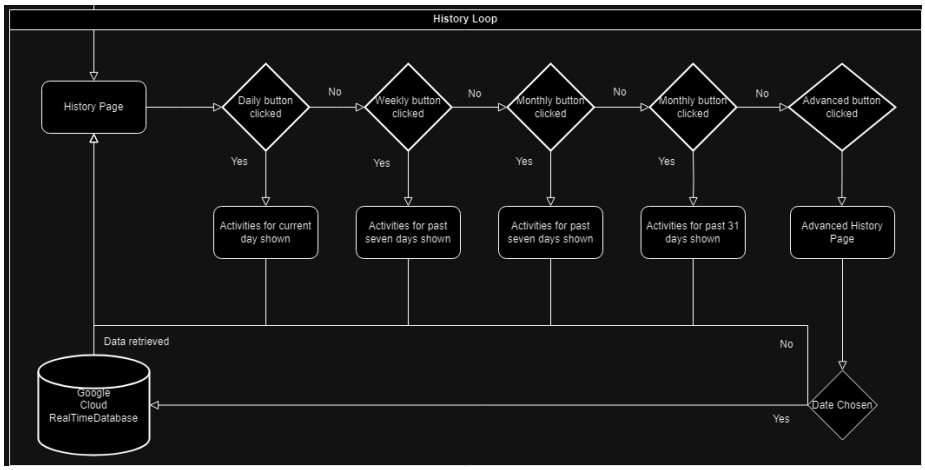


Figure 16: History loop

3.6.4 F1 Score

We use the F1 score to provide an unbiased and more realistic representation of the combination of precision and recall.

$$F1 = \frac{2 * P * R}{P + R}$$

3.6.5 Accuracy

We display our models' accuracy for marking purposes and to enable comparisons between our peers' models. However, using accuracy as a sole metric can sometimes lead to false confidence in a model's capabilities. For example, if a model performs classification on an imbalanced dataset, an accuracy of 95% could still consist of the model getting nearly everything wrong for one class. Therefore, although this task is more well-rounded for all classes, we found it necessary to include more metrics to show that our model performs extremely well.

3.6.6 LOOCV (Leave-One-Out Cross Validation)

We tested our model performances with the Leave-One-Out Cross-Validation(LOOCV). Despite being a computationally expensive approach, it is a reliable and impartial assessment of the model performance. LOOCV takes a single student's data out as a test set and uses the rest of students' data as training set.

4 Results

4.1 Preprocessing and Baselines

A key part of any machine learning system is the preparation and extraction of features performed on the raw data before the model is trained or evaluated on it.

4.1.1 Feature extraction and data exploration

We started by dividing the data files into a series of 50 sample windows using an overlap of 50%. Then, based on the work presented by Zhu et al. [30], we extracted a series of statistical values from each of the three accelerometer axes, including statistics based on the frequency based on the original data, the fast Fourier transform, and the jerk. From each axis, we extracted the mean, median, standard deviation, skewness, kurtosis, range, total energy, signal power, signal variance, root mean square, zero crossing rate, and crest factor; then from the Fourier transform we extracted the dominant frequency, mean frequency, median frequency, and spectral entropy. Finally, we calculate the jerk for each axis and extract its mean, standard deviation, skewness, and kurtosis. This resulted in a total of 57 features, 19 from each axis.

Applying PCA to these features allows us to reduce the windows down to a two-dimensional data point that we can then plot, as seen in the figure 17. We can see two separate clumps, one on the left that contains all the non-stationary activities and another on the right that contains the stationary physical activities. We can see that while the activities involving a large amount of movement are mixed in a complicated way the stationary activities seem to form clear slices, with the lying down activities vertically stacked on top of each other, and the sitting/standing placed away from these lying activities. This suggests that stationary activities can probably be separated by a simple model, while other activities will require a more complicated system. The distance between stationary activities and other activities in the samples also suggests a relatively simple model could be used to separate these two subgroups and then more complicated models could be used on each subgroup to complete the classification. We also performed this procedure by separating based on respiratory symptoms; however, this diagram did not provide any meaningful information and is not included here.

The plot of the number of windows for each of the physical activities and respiratory symptoms also provides some useful information. In figure 18 we can see that there is a large amount of class imbalance when we divide the classifications into two labels. To resolve this we decided to sub-sample the more frequent classes when training the model for use in the app, this will help to improve its real-world performance by removing the class frequency basis from the data. However, to develop our model for evaluation, we did not perform this sub-sampling, since it will be evaluated on unseen data that contain the same class frequency biases.

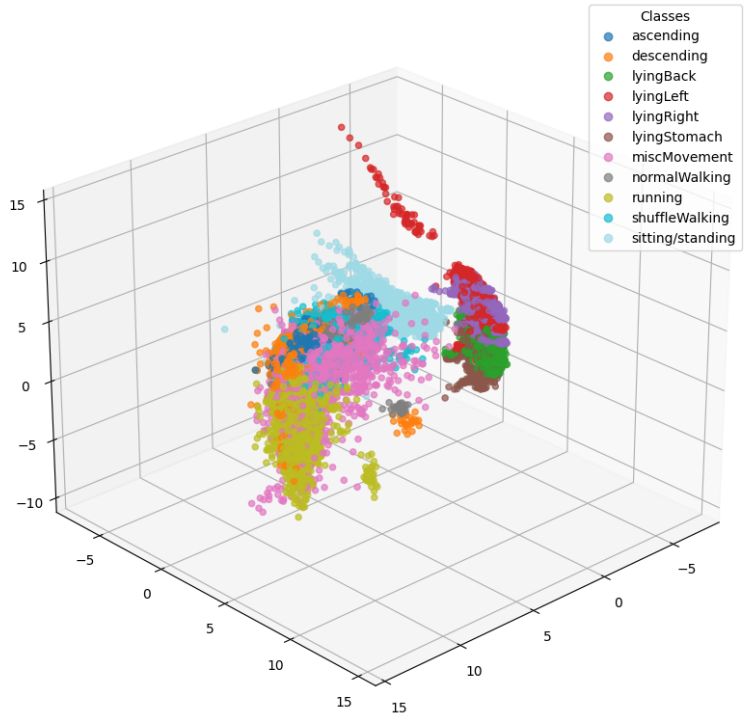


Figure 17: Diagram showing the three principle components after performing PCA to the data, each point represents a window and is coloured by the physical activity performed.

Furthermore, to ensure that all of the selected features are indeed meaningful, we fitted two random forest classifiers, one that predicts a given window's physical activity and another that tries to predict the respiratory symptom (with eating, talking, laughing, and singing combined into one class called "other"). These random forest models attained 89% and 66% accuracy respectively;. In contrast, these accuracies are insufficient for a final model, but they do show that the model has been able to at least partially fit the patterns seen in the data.

The lower accuracy when predicting respiratory symptoms also confirms that this will be a more difficult task than simply predicting the physical action being performed. As a consequence, their importance will give us some insight into the features that are meaningful in making predictions. The importance weights for each feature within the models can be seen in figures 19 and 20. These figures both show a gradual decline in importance; however, we do not see one feature that is unimportant for both models, suggesting that all the

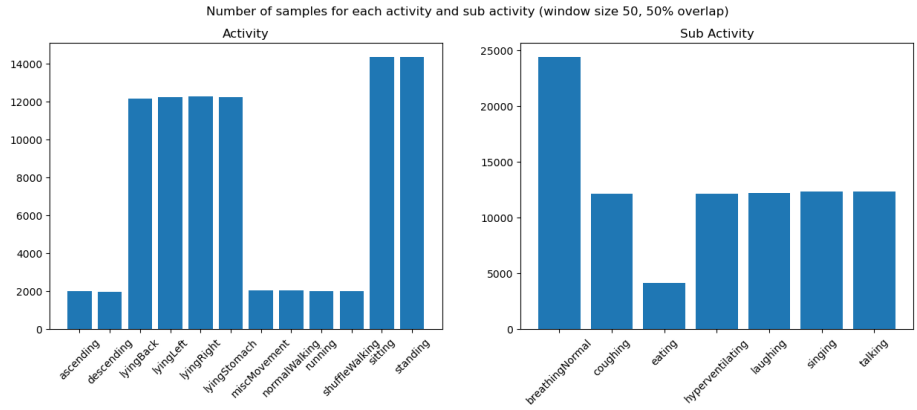


Figure 18: A Diagram showing the number of windows (size 50 with overlap 25) for each of the physical activities(left) and respiratory symptoms(right) that can be formed from the given data.

values we have selected are indeed relevant in making our predictions. We also see a sharper drop in importance for the respiratory model suggesting that our chosen features are insufficient for high accuracy in this task. Instead of spending a large amount of time extracting more features that would likely need subject-specific knowledge, we will proceed by implementing deep learning methods.

Some feature extraction was still completed for these deep learning methods; however, it was focused on producing new columns that preserved the window's shape, as opposed to summary statistics. Since features from the original data, jerk, and fast Fourier transform (FFT) held significance to the random forest models, we decided to include all of these as new columns within the data for our deep learning models. As a result, we augmented the data with 5 new columns per axis, 4 from the FFT (magnitude, real component, imaginary component and angle) as well as a new column containing the jerk. This resulted in each window consisting of a 50x18 data frame. Furthermore, since the activities and respiratory activities define discrete and unordered classes, we decided to onehot encode the labels for our deep learning models.

4.1.2 Choosing a window size

Until now all of our models have been running on a window size of 50 however, this may not be the optimal choice. This window size is a trade-off between the number of samples we can generate from the data, latency in predictions, and the amount of information a model has to make decisions based on. To determine which is likely to be the best for 20 epochs we trained an example LSTM-CNN-based model that consisted of one LSTM layer with 128 units and a CNN layer with 64 units to predict the physical activities - which are the activities that

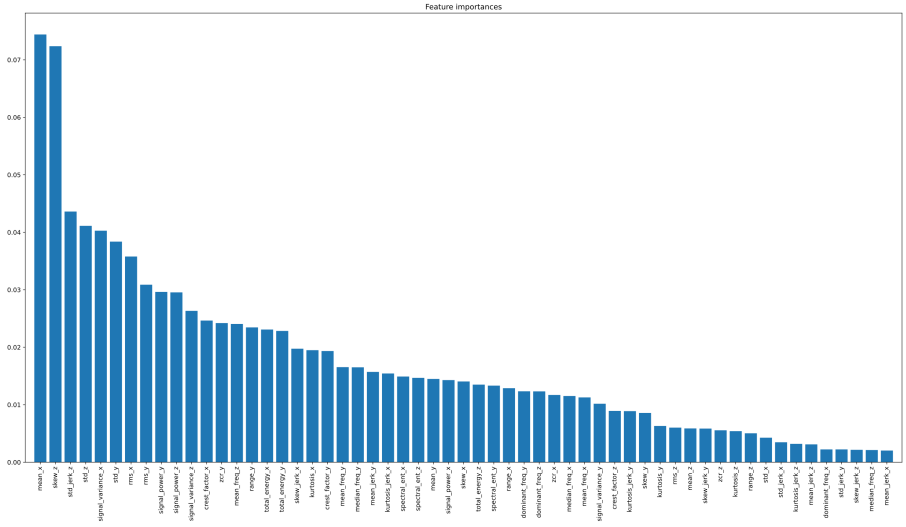


Figure 19: A graph showing the importance weights given by the sklearn random forest when predicting a window's physical activity.

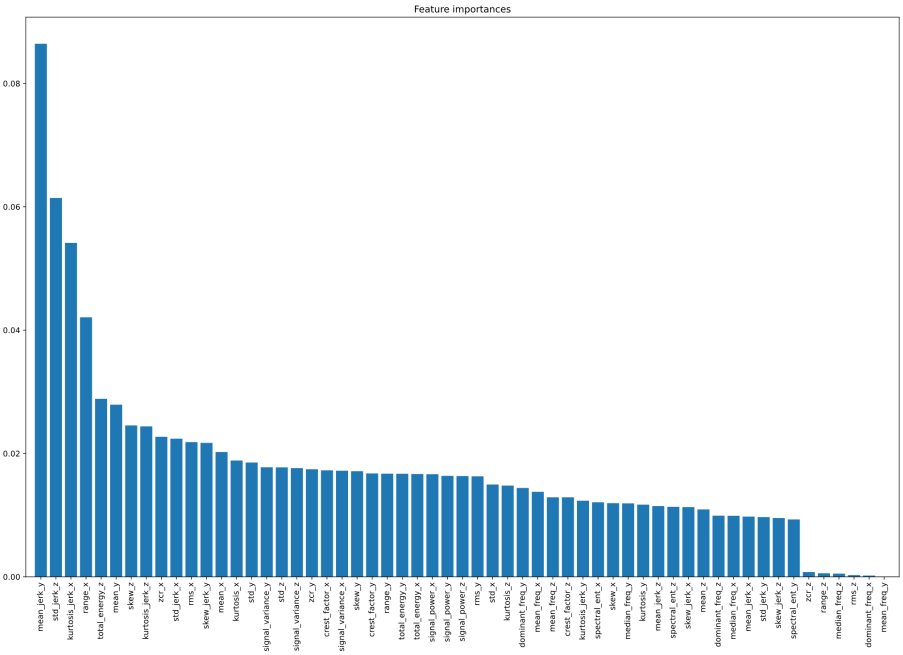


Figure 20: A graph showing the importance weights given by the sklearn random forest when predicting a window's respiratory symptom.

Window Size	train accuracy	val accuracy
25	0.97	0.82
50	0.98	0.84
75	0.98	0.83
100	0.98	0.85
150	0.97	0.83

Table 3: A table showing the average accuracy (2dp) of 3 trails for an LSTM-CNN model for different window sizes, each recorded with a 50% overlap.

are most likely to have longer-term trends, unlike coughs or hyperventilation that are rapid movements. The results of these trials can be seen in Table 3, all windows are created with 50% overlap. Here, we can see that the two best window sizes were 50 and 100, with 100 performing slightly better. However, using a window size of 100 only provides a slight performance advantage of around 1% while doubling the amount of time needed before we can make a completely fresh prediction. To improve the performance of real-time prediction within our app, we decided to proceed with a window size of 50 instead of 100, since this preserves most of the performance while providing a reasonable prediction latency of 2 seconds.

4.1.3 A note on signal smoothing

Within our testing, we also considered applying a low-pass filter over the signal data with a cutoff at 2.25hz before any extra preprocessing was performed based on the process conducted by Georgescu [9] to construct a model capable of classifying similar tasks. This process is included within the submitted version of the app for our physical activity classifier, however following further evaluation we found that it did not improve our predictive capability and added a noticeable amount of computational expense. Therefore, it is no longer included with our models.

4.1.4 Baselines

Following the extraction of features as described in 4.1.1 the windows were split into three sets (train, validation and test) with an 80/10/10 split without shuffling the windows to prevent overlaps causing training data to be contained within the test and validation sets.

We then ran four "out-of-the-box" sklearn models that can be used as baseline comparisons for later neural network-based techniques. For these four baselines, we selected two simple models: logistic regression and k-NN with $k = 10$; as well as two more complicated models: a random forest, and the sklearn Gradient boosting regressor. We trained these methods on the train data set, then evaluated their performance on the test set, with the reported accuracies shown in Figure 4. We performed this procedure for three cases, predicting physical activity for normal breathing (task 1), predicting respiratory symptoms for

Model	Physical Activities	Respiratory Symptoms	Activities & symptoms
K-NN (K = 10)	0.64	0.55	0.47
Logistic Regression	0.85	0.61	0.62
Random Forest	0.89	0.66	0.63
Gradient Boost Regressor	0.93	0.66	-

Table 4: A table showing the accuracy (2dp) of our baseline methods when evaluated on the test data set. We did not evaluate the Gradient Boost regressor for Activities & symptoms due to its excessive run time.

stationary data, and predicting both the physical activity and respiratory symptoms for stationary data (task 3). The results of these baselines confirm our earlier suspicions based on the PCA diagram.

It is clear that the more complicated models do perform better for physical activity recognition (with gradient boost regressor performing very well and attaining 94%) however there is a much smaller improvement for respiratory and combined prediction. Notably, K-NN performed poorly, which further suggests there is a high level of class overlap as we saw in the PCA diagrams.

The logistic regression performed very well for its relatively simple construction and its confusion matrix can be seen in figure 21. This shows that even a simple model can easily classify stationary activities with very high accuracy, attaining nearly 100% accuracy for almost all of these classes. We also see that the model struggles to differentiate between misc movement and other mobile activities, which makes sense considering the wide range of activities involved in misc movements.

In figure 22 we can see the confusion matrix produced by the gradient boost regressor when predicting each window's respiratory symptom. Within this matrix, we see that the symptoms included in task 2 (normal breathing, hyperventilating, and coughing) have a relatively high degree of accuracy, the largest confusion originating from the coughing and hyperventilating classes due to them having similar patterns within the data. The model does, however, seem to struggle when predicting the other category, due to the wide range of activities and thus not having one single pattern to match.

4.2 Model Selection

Using the information about the relative difficulty of predicting each of the classes leads to three possible structures for building our models, shown in Figure 24. Within this subsection, we will describe and evaluate these three structures and then select the best for further tuning and final evaluation. To make this a fair comparison, we used the same generic CNN-based deep neural network within the core components of each structure; a diagram showing the construction of this model can be seen in Figure 23. This ensures that it is not this selection of components that provides performance improvements. The tests were carried out with 64 units in each Conv1D layer and a dropout rate of 0.3.

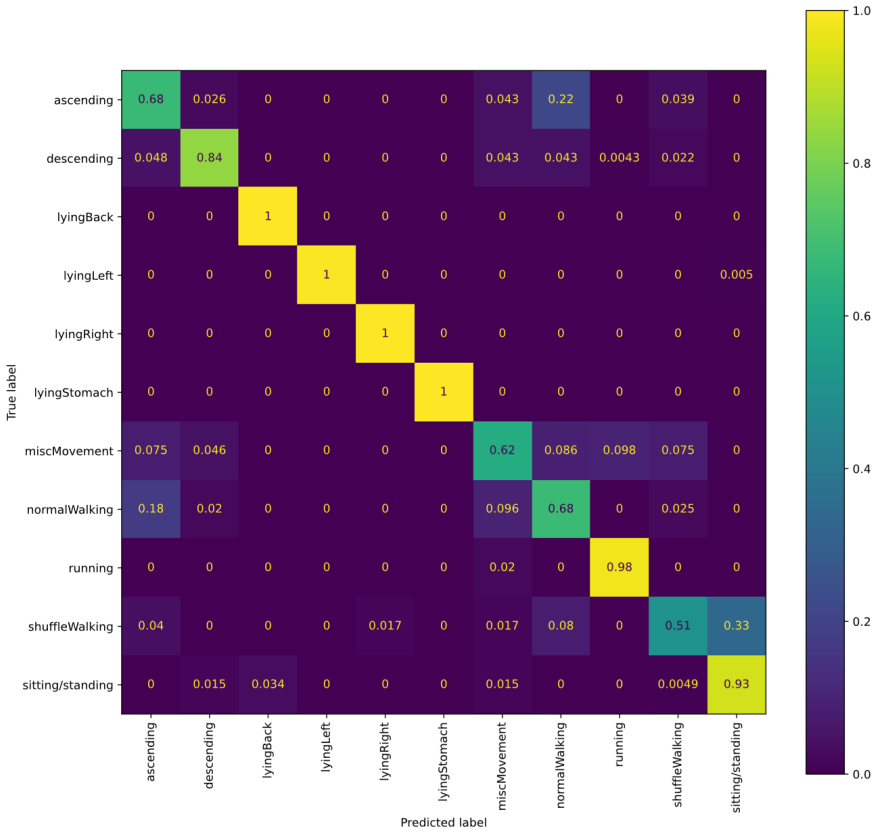


Figure 21: Confusion matrix of a logistic regression classifier when used to conduct task 1.

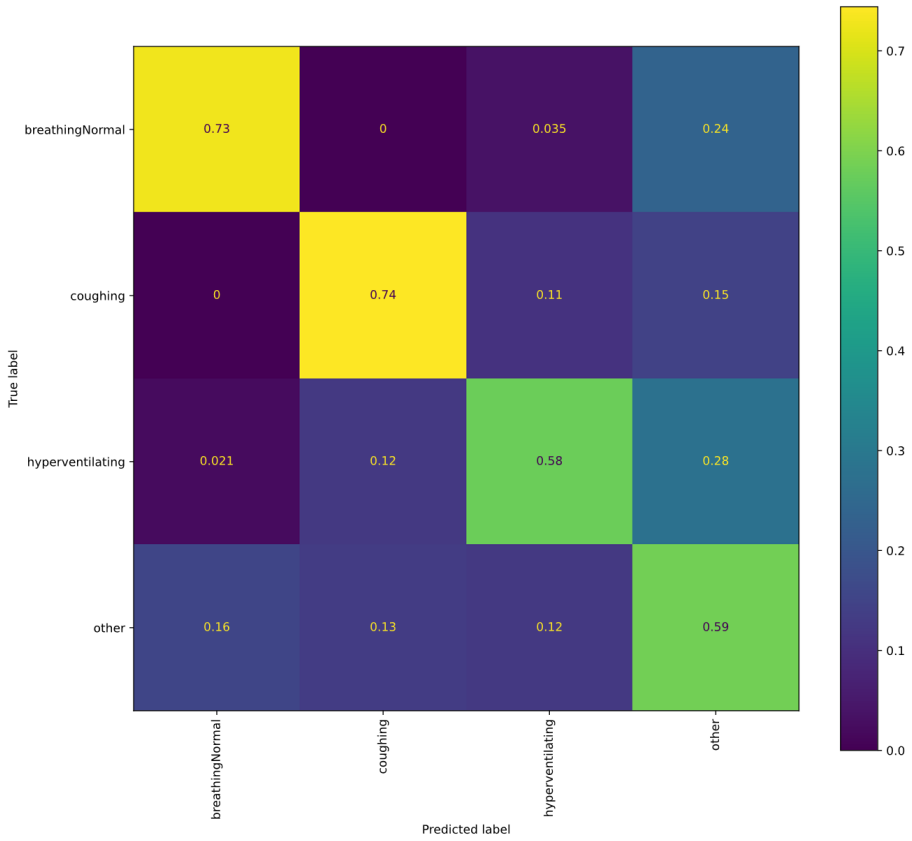


Figure 22: Diagram showing the confusion matrix of the gradient boosting regressor from sklearn when used to predict the respiratory symptoms of each window.

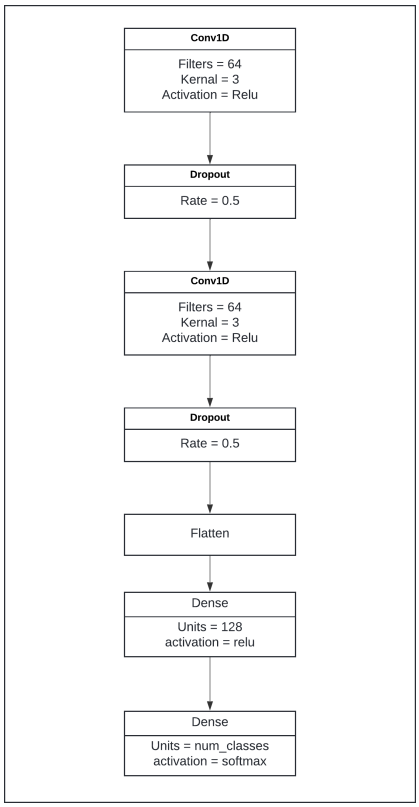


Figure 23: A flow diagram showing the construction of the generic CNN-based deep neural network used for model structure selection.

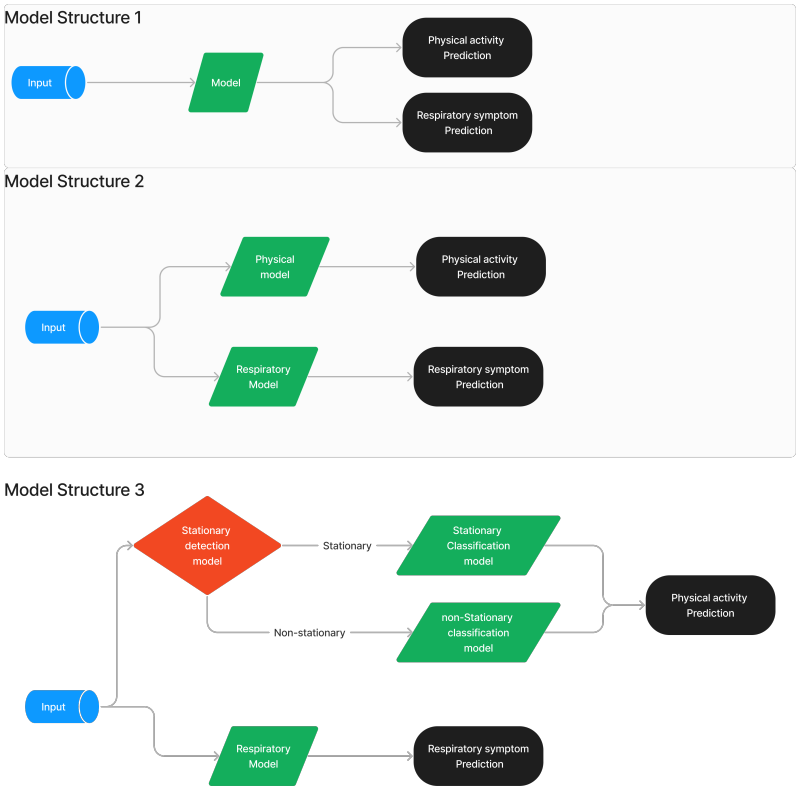


Figure 24: A flow diagram showing the three model structures. The input is assumed to have already been pre-processed.

4.2.1 Model structure 1

The first and simplest structure we considered was to use a single model to predict all the classes within a given task. This means we will have only one of the generic CNN networks to predict both the physical and respiratory symptoms in tasks 2 and 3. This model is also useful for comparing a simple deep network approach to our baselines.

4.2.2 Model structure 2

Using the observation that the relatively simple baseline models were capable of attaining good accuracies for predicting the physical activity of a window we now split the problem into two subparts, the prediction of the physical activity and the prediction of a respiratory symptom. In the case of task 1, this is equivalent to model structure 1 and so only one of the generic Conv1D networks 23 was used. However, for task 2/3 we found that prediction of the physical activity could be performed by a simple two-layer dense network ran for 5 epochs with a 256 batch size. The respiratory symptom was then predicted using the generic two-layer CNN model used in Structure 1.

4.2.3 Model structure 3

Model structure 3 further divides the problem by using four models. One that classifies the respiratory symptoms (as before) and three that combined produce a prediction for the physical activity. To produce a prediction for the physical activity, we first use a binary classifier to decide if the widow represents a stationary activity (sitting/standing, lying on back, lying on right, lying on left, lying on stomach). If the activity is stationary, we send this to a simple classifier and report its classification; otherwise, we send it to another more complicated classifier that is dedicated to differentiating between the nonstationary activities.

This structure then allows a dedicated model to focus on classifying the nonstationary activities that our baselines had a large amount of difficulty telling apart, hopefully improving our accuracy compared to structures 1 and 2. This structure also allows the stationary activities to be classified without needing to evaluate a large neural network.

We used the same model to predict respiratory symptoms as in Structure 2. We also found that two dense layers were sufficient to gain nearly perfect prediction (around 99%) on a test set when used as both the stationary detection model and stationary classifier.

4.2.4 Model structure Comparison

When applying our generic neural network (Figure 23) to task 3 we achieved a test accuracy of 69% after 30 epochs. Even with this simple construction, we do indeed see a marked improvement to any of our baselines that achieved a maximum of 63% accuracy in this task. This shows that neural networks can

provide substantial improvements for this type of classification. Structure 2 on the other hand attained a slightly higher accuracy of 71% in task 3, showing that by classifying the physical activity using a simple dense network and allowing the CNNs to conduct the respiratory symptom prediction we can indeed attain a higher accuracy.

Anecdotally, when implemented within the application where we need to include all the physical activities, not just the stationary ones, in our predictions we found that models that followed structure 1 were less reliable. These systems tended to jump around predictions more violently, often mistaking hyperventilating or coughing for a non-stationary activity. Structure 2 was also able to achieve higher test accuracy in task 3 of 71%.

Structure 3 only differs from Structure 2 in the way that physical activities are predicted in Task 1 and as such we shall compare this structure to the others based on Task 1 performance. When evaluating the models on the test data we attained an accuracy of 92% for both steps 3 and 2. This falls slightly short of the gradient-boosted regressor baseline, however, these models took substantially less time to train and still pose more room for improvement. Since Structure 3 fails to provide any substantial improvement over Structure 2 while increasing the overall complexity of the model, we decided that the best overall model structure was Structure 2 and, as such, we will be required to build and further tune two models, one for physical activity classification and one for respiratory symptom classification.

4.2.5 Further model tuning

Now that the overall model structure has been decided we must tune our model selection for each of the two key components, and as such we will consider three different models and evaluate them on each of the two tasks. Within the literature, we saw a mixture of LSTM, CNN and CNN-LSTM-based models in use for tasks similar to those we are targeting and as such we trialled one of each to evaluate their relative performance. Details for the models evaluated can be seen in Figure 25 and the results can be seen in Table 5.

Model	Task 1 Accuracy	Task 3 Respiratory Accuracy
CNN	0.910	0.760
CNN-LSTM	0.912	0.776
LSTM	0.907	0.687

Table 5: Table showing the test accuracy attained by the three models shown in Figure 25, when applied to task 1 and the respiratory component of task 3.

Here we see that the pure LSTM model performs poorly and, thus may not be appropriate for the tasks we wish to predict. Therefore, we have the CNN and CNN-LSTM models to choose from. For physical activity classification, these two models performed very similarly and thus since the CNN model is substantially simpler, taking much less time to train we have chosen this model

to perform the physical classification for our offline models. However, the CNN-LSMT does gain some performance over the pure CNN model for the prediction of respiratory symptoms.

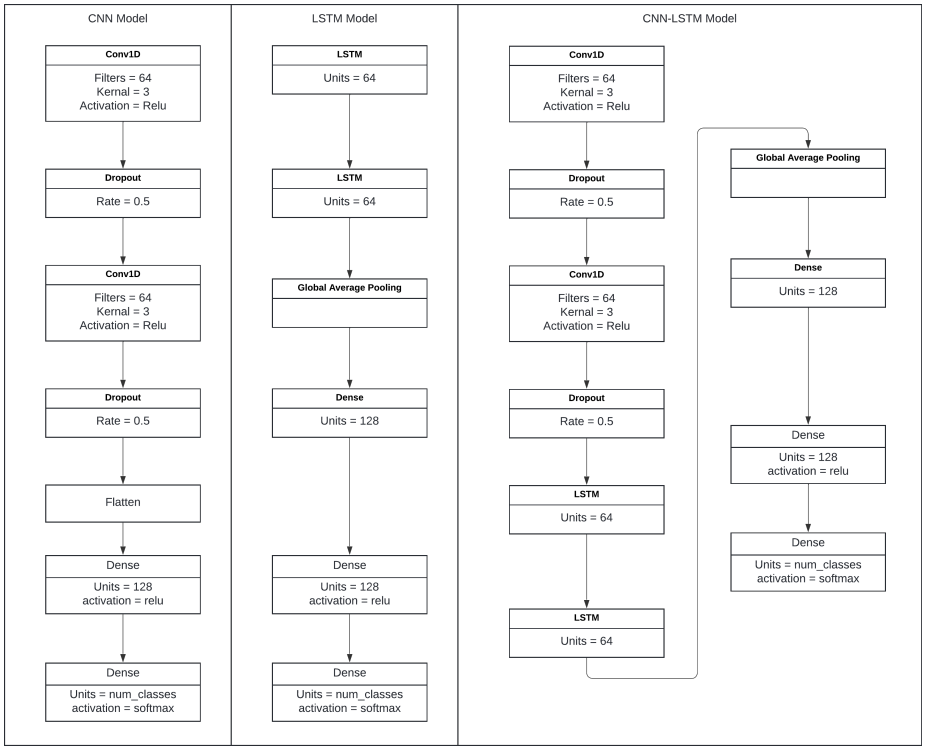


Figure 25: A Diagram showing the three models evaluated. The CNN model is the same as the generic CNN used in selecting the model structure.

In order to further tune these selected models we used a combination of manual testing on the validation set and the hyperband tuner from the keras-tuner package [20] to select the optimal hyperparameters as well as to evaluate the effect of adding extra layers, such as the GlobalAveragePooling layer. This resulted in the two models shown in Figure 26.

The final step in our tuning procedure was to select an optimal batch size and the corresponding number of epochs. To decide this, we plot the training and validation accuracy at each epoch for three batch sizes (64,128,256) for both models, as can be seen in Figure 27. We then picked the combination that provided the lowest validation loss, which for the physical model was a batch size of 256 and with 25 epochs, and for the respiratory model was a batch size of 128 with 30 epochs.

4.3 Final results

The model we selected was evaluated using leave one student out cross-validation (training the model on all but one student and then predicting the classification for the windows generated from the final student). The average of the performance results of this evaluation for each task can be seen in table 6. It is worth noting that we saw a large amount of variance for the models' performance depending on which student was left out, with some students consistently performing notably worse than others irrespective of which model was being evaluated, this variability can be seen in Figure 30. This may suggest that the data gathered by these students is in some way lower quality and could be related to the two Respeck sensors that we discovered were broken after the data were collected (one of which had the axes swapped). These errors in data collection also mean that some activities are not included for all students since some activities are more difficult to classify than others, and this means that the average reported here will be skewed by these removals. For tasks 2 and 3 we used the simplified physical model consisting of only two dense layers since this provides a successful classification of the stationary activities.

Task	Accuracy	Recall	Precision	f1
1	0.926 ± 0.014	0.926 ± 0.014	0.933 ± 0.014	0.918 ± 0.016
2	0.864 ± 0.018	0.864 ± 0.018	0.880 ± 0.020	0.849 ± 0.022
3	0.748 ± 0.018	0.748 ± 0.018	0.763 ± 0.0189	0.726 ± 0.02

Table 6: A table showing the performance statistics calculated as an average of loo cross-validation for each task. Each value is reported with an error bar of two standard errors.

We also evaluated this model using the earlier 80/10/10 train, validation and test split to produce the classification report and confusion matrices seen below. This model retains some of the trends we saw in Section 4.1.4 in the types of miss-classification made, suggesting that there are underlying constraints in the quality of data we have access to and the choice of sensor/ sensor positioning used for classification that may be limiting the performance of our models. We have also not been able to substantially improve our performance of Task 1 compared to baselines, although we have made substantial gains for Task 2/3 in the quality of prediction of respiratory symptoms. The confusion matrices for task 1 and the respiratory component of task 3 can be seen in Figures 28 and 29 respectively. We can see hotspots away from the diagonal for normal walking/ascending and descending/misc movement, these are indeed areas in which we observed common miss-classifications with our real-time classifier once embedded within the application and do represent activities that are close in motion type, this closeness of the movements involved is likely the cause of such hotspots. Without the inclusion of other sensors, it may indeed be difficult for any model to completely remove these errors. We also notice that the respiratory model has started to favor the other classification much more heavily due to the

higher number of these samples within the data, with at least one other activity closely mirroring one of the other three classes.

Using only an accelerometer also increases the difficulty in detecting respiratory symptoms, as opposed to using multiple sensors such as a gyroscope or microphone; as was used in a paper by Pahar et al. [21]. We also saw large and consistent fluctuations in the leave-one-out classification based on which student was left out.

4.4 Adding the gyroscope

To explore the possibility of including multiple sensors within our algorithm, we experimented with including the gyroscope data from the REspeck within the models. We did this by performing the same data extraction processes described for use with deep learning models to the gyroscopic axes and the accelerometer axes as we had previously, then returned our chosen model with kerastuner. This new model using gyroscope data achieved an improved accuracy of $78\% \pm 2\%$ in task 3. This is a reasonable improvement, the average of which is pulled down substantially by two students that had noticeably lower performance at around 40%, while many samples attained over 80%. As a result, we chose to use both sensors within our in-app real-time classifier as well.

5 Conclusions

5.1 Reflections

During this project, we came across various challenges relating to different disciplines. Overall, we were extremely happy with the amount and quality of work that we put out. This was possible due to the dedication of each group member when it came to doing our part in the project.

Firstly looking at the models, we learned that there are different types of neural networks, such as CNNs and LSTMs. It was invaluable to have a hands-on experience and it made us appreciate the technological advancements that have been made in the past decade. Related to the models, we realised that sometimes, a complicated and convoluted model architecture was not always the best. Not only was it confusing for us as engineers to interpret the many layers, but the results were often not up to par with some of the simpler models that we employed.

On the software side, we also learned many valuable lessons. For example, we started using GitHub as our version control as soon as we started working on the application. This meant that we did not have any conflicts between multiple versions and had a smooth development experience when it came to merging changes and creating our branches. Moreover, we found it extremely useful to have code review sessions where we would try and optimise certain aspects of the software. Through these sessions, we managed to cut down a lot

of bloat and use certain design patterns to follow the gold software development standards.

Finally, on the project management side, we believe that we achieved good efficiency and communication. We constantly updated and uploaded tasks, bugs and issues onto our Jira, and we worked as a single unit through constant communication through both in-person meetings and our group chat.

5.2 Future Improvements

5.2.1 UI Improvements

Due to the time constraints of this coursework, we mentioned that we had focused on building simple and intuitive user interfaces, and we believe that we achieved this goal. However, we understand there is room for improvement. For example, we could build an entirely new app from scratch instead of relying on the provided skeleton app. This would allow us to choose an original colour scheme and design new pages rather than relying on pre-made ones.

Furthermore, we could also improve our history page. First, we could add more information about the retrieved activity history. For example, we could display a pie chart showing the percentages of certain activities (e.g. normal breathing vs coughing vs hyperventilating vs other), and warn the emergency phone number if the user seems to be coughing or hyperventilating frequently. Furthermore, a more manageable addition would be to allow selecting a custom date range rather than a single date in the advanced history page. This would help to see the user's activities for a given period, rather than being fixed to just one day.

In conclusion, our user interface provides an efficient and simple user experience by removing any clutter from a user's screen. This ensures that users will be able to use the app regardless of their technical proficiencies.

5.2.2 Software Improvements

On the software side, we believe that we have achieved a good level of cohesion and coupling between classes. However, we could consider trying to optimise our code even further to lower the CPU and memory usage (although this is partly due to running a Python script in-app). We could dedicate more time to developing a Kotlin-native version of the pre-processing script, or try to reduce the time complexity of certain functions.

Next, we could have split the scopes of certain classes even further. Taking `UserSession`, we should have separated code tracking the currently logged-in user and the current activity data cache to ensure that each of these new classes only had a single scope.

Overall, we believe that our code achieves a good level of cohesion and coupling, which is crucial to having reusable and efficient code.

5.2.3 Model Improvements

Though our models did perform well in conducting real-time classification they fall short of attaining the highest bands for performance. This demonstrates space for potential improvements for the models, potentially by utilizing a non-sequential design or an ensemble of models for each component of the tasks. One such model may be using the physical model classification probabilities array as an extra input for the respiratory model. We could also potentially improve our models by integrating other sensors that are placed on other body parts, such as those within a smartwatch, and this would enable both higher accuracy for the physical classification and for different classes (e.g. separate sitting and standing) to be identified.

A further improvement would be to have conducted column randomization for our extracted features. This would allow us to identify if all of our generated features are indeed being used by our final model. If columns are no longer important we could then remove them from the prepossessing and save computation, both in our offline classification and in the app.

References

[1] Atlassian. `https://www.atlassian.com/software/jira`. 2023.

[2] Alejandro Baldominos et al. "A Comparison of Machine Learning and Deep Learning Techniques for Activity Recognition using Mobile Devices". In: *Sensors (Basel, Switzerland)* 19.3 (2019), p. 521.

[3] Joshua Bloch. *Effective java*. Addison-Wesley Professional, 2008.

[4] Jason Brownlee. *Deep Learning Models for Human Activity Recognition*. `https://machinelearningmastery.com/deep-learning-models-for-human-activity-recognition/`. Aug. 2019.

[5] Kaixuan Chen et al. "Deep learning for sensor-based human activity recognition: Overview, challenges, and opportunities". In: *ACM Computing Surveys (CSUR)* 54.4 (2021), pp. 1–40.

[6] Android Developers. `https://developer.android.com/develop/connectivity/bluetooth/ble/ble-overview`. 2023.

[7] D Figo et al. "Preprocessing techniques for context recognition from accelerometer data". In: *Pers. Ubiquitous Comput.* 14 (2010), pp. 645–662.

[8] Martin Fowler, Jim Highsmith, et al. "The agile manifesto". In: *Software development* 9.8 (2001), pp. 28–35.

[9] Teodora Georgescu. *Classification of Coughs using the Wearable RESpeck Monitor*. 2019.

[10] Google. *Google Computing Services*. 2023.

[11] Fuqiang Gu et al. "A Survey on Deep Learning for Human Activity Recognition". In: *Journal Name Here* (Year Here).

[12] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[13] Wenbo Huang et al. "Shallow Convolutional Neural Networks for Human Activity Recognition Using Wearable Sensors". In: *IEEE Transactions on Instrumentation and Measurement* 70 (2021), pp. 1–11.

[14] Walaa N. Ismail et al. "AUTO-HAR: An adaptive human activity recognition framework using an automated CNN architecture design". In: *Heliyon* 9.2 (2023), e13636. DOI: 10.1016/j.heliyon.2023.e13636.

[15] Shian-Ru Ke et al. "A review on video-based human activity recognition". In: *Computers* 2.2 (2013), pp. 88–131.

[16] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), pp. 436–444.

[17] Chaquo Ltd. https://chaquo.com/chaquopy/. 2023.

[18] Manav Mandal. *Introduction to Convolutional Neural Networks (CNN)*. https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/. Nov. 2023.

[19] Nordic. https://infocenter.nordicsemi.com/index.jsp?topic=%2Fug_thingy52%2FUG%2Fthingy52%2Fintro%2Ffrontpage.html. 2019.

[20] Tom O'Malley et al. *Keras Tuner*. https://github.com/keras-team/keras-tuner. 2019.

[21] Madhurananda Pahar et al. "Automatic non-invasive cough detection based on accelerometer and audio signals". In: *Journal of Signal Processing Systems* 94.8 (2022), pp. 821–835.

[22] Nick Provos and David Mazières. https://www.mindrot.org/projects/jBCrypt/. 2015.

[23] Sen Qiu et al. "Multi-sensor information fusion based on machine learning for real applications in human activity recognition: State-of-the-art and research challenges". In: *Information Fusion* 80 (2022), pp. 241–265.

[24] Neil Robertson and Ian Reid. "A general method for human activity recognition in video". In: *Computer Vision and Image Understanding* 104.2-3 (2006), pp. 232–248.

[25] M Shoaib et al. "Fusion of smartphone motion sensors for physical activity recognition". In: *Sensors* 14 (2014), pp. 10146–10176.

[26] Greg Van Houdt, Carlos Mosquera, and Gonzalo Nápoles. "A Review on the Long Short-Term Memory Model". In: *Artificial Intelligence Review* 53 (Dec. 2020). DOI: 10.1007/s10462-020-09838-1.

[27] Wikipedia Contributors. *Long Short-term Memory*. https://en.wikipedia.org/wiki/Long_short-term_memory. Accessed in 2024.

[28] Wikipedia Contributors. *Vanishing Gradient Problem*. https://en.wikipedia.org/wiki/Vanishing_gradient_problem. Accessed in 2024.

[29] Kun Xia, Jianguang Huang, and Hanyu Wang. "LSTM-CNN Architecture for Human Activity Recognition". In: *IEEE Access* PP (2020), pp. 1–1.

[30] Jiadong Zhu, Rubén San-Segundo, and José M Pardo. "Feature extraction for robust physical activity recognition". In: *Human-centric Computing and Information Sciences* 7 (2017), pp. 1–16.

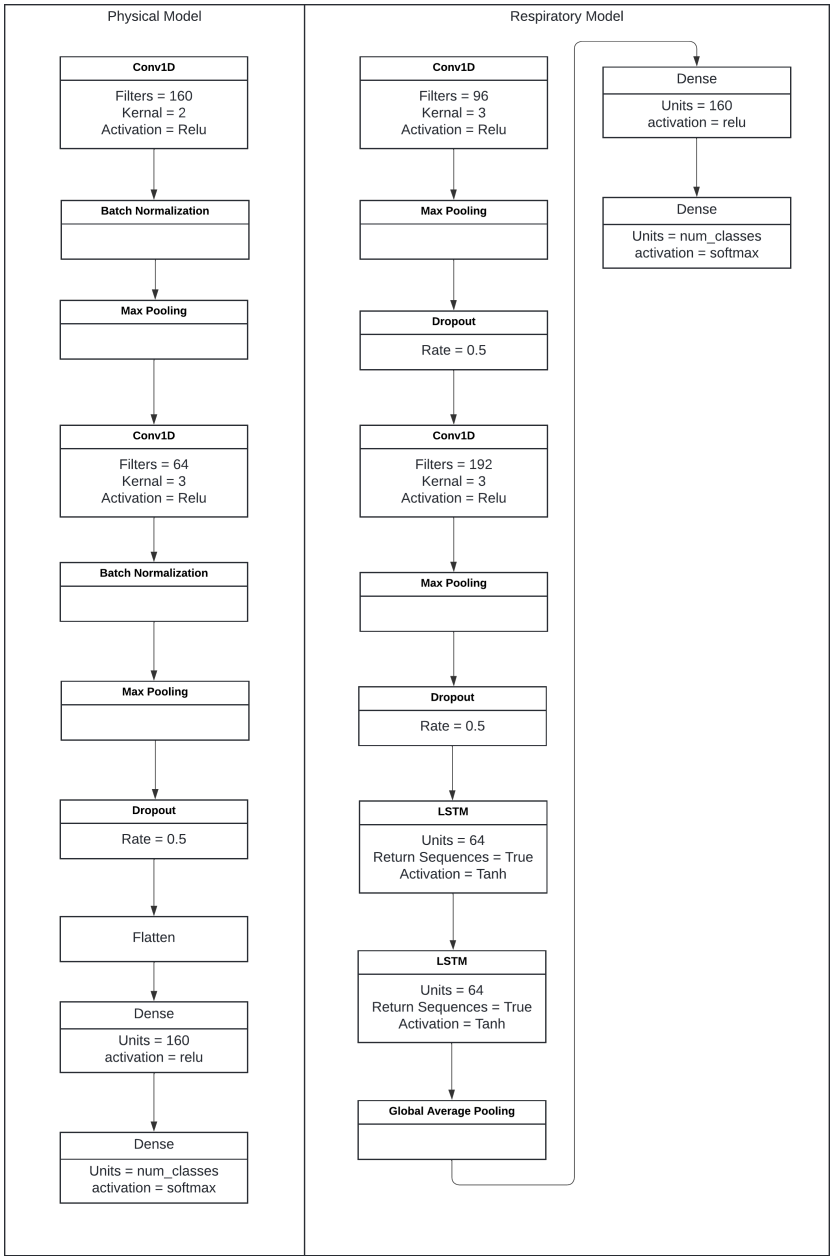


Figure 26: A Diagram showing the construction of our final physical and respiratory classification models.

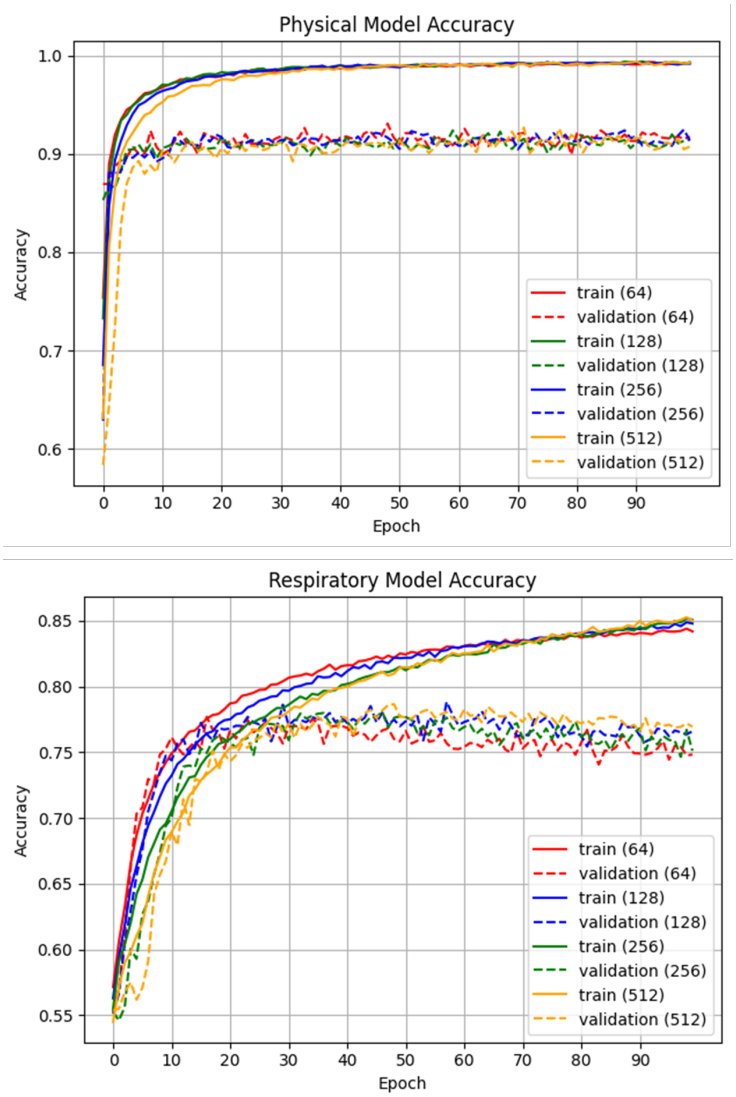


Figure 27: Graphs showing the train and validation accuracy for each.

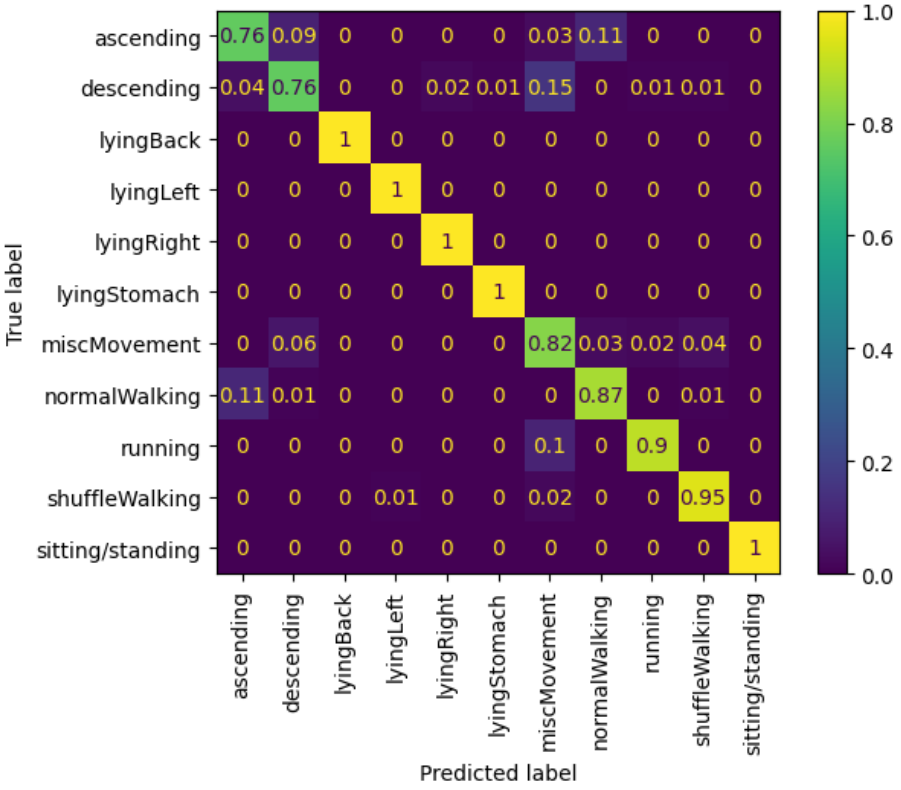


Figure 28: A confusion matrix showing the classification of the test set as predicted by our final task 1 model.

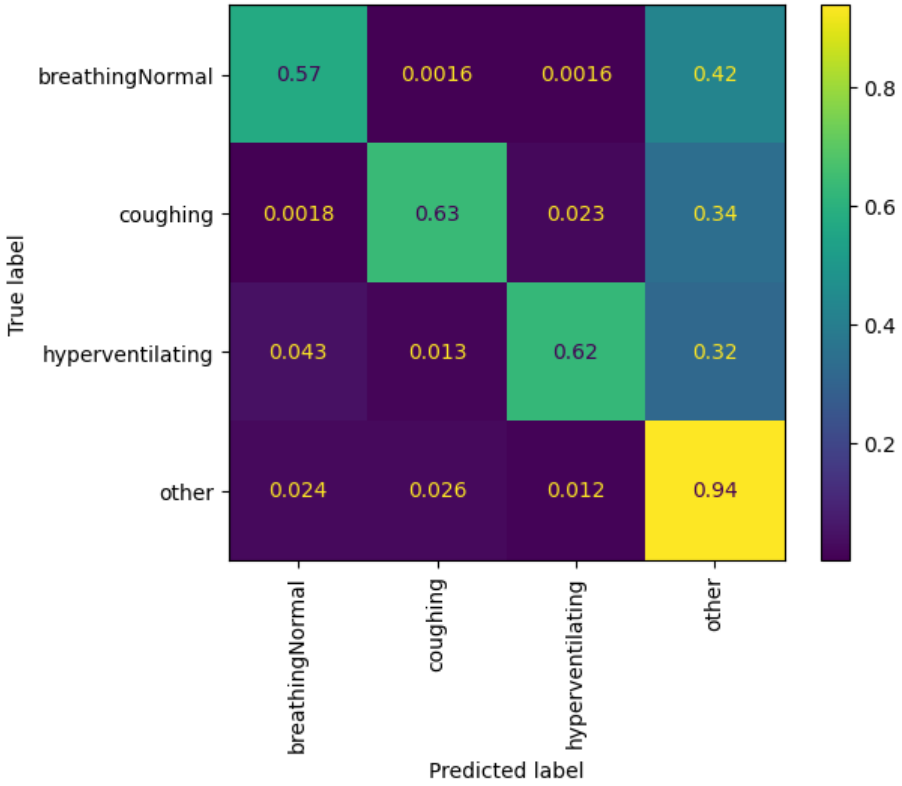


Figure 29: A confusion matrix showing the classification of the test set as predicted by our final task 3 respiratory model.

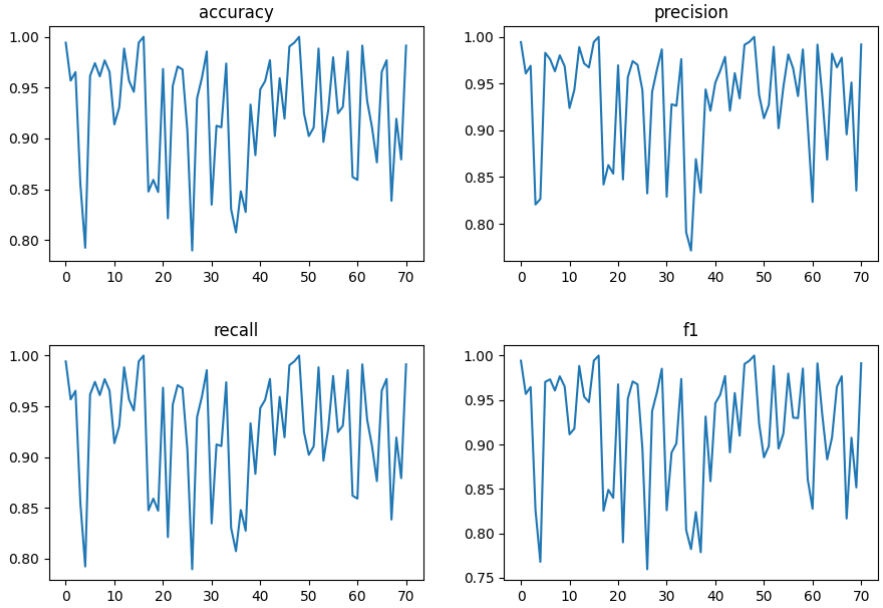


Figure 30: A Figure showing the performance statistics for each iteration of the leave-one-out evaluation. The x-axis represents the student left out for a given iteration.