Report on Human Activity Recognition Application

Coursework 3
Principles and Design of IoT Systems
School of Informatics
University of Edinburgh

2022

Abstract

Activity recognition can benefit society greatly, especially when considering human-centric applications such as eldercare and healthcare. This paper focuses on recognizing simple human activities. Recognizing complex activities remains a challenging and active area of research. In this report, we present a sensor-based human activity recognition system integrated in an Android app. We touch upon relevant research that has been done in the field and briefly describe the sensors used in collecting our sample data. We argue the choice of our models, CNN and LSTM, and thoroughly evaluate and analyse the results we obtained. The architecture of our app is also detailed and discussed in depth. At the end of this work, we reflect on what we learnt and provide some possible future expansions of our system.

Table of Contents

Chapter 1

Introduction

Activity recognition is a highly active research area, which, if exploited, can bring great benefits to society, especially when considering human-centric applications such as eldercare and healthcare [24]. Sensor-based systems utilize on-body or ambient sensors to recognise people's motion details or log their activity tracks. Given the privacy issues of video-based HAR systems, sensor-based ones have dominated the market for daily activity monitoring applications [6]. There are many challenges in building such system though, some of them which are described below, with more details given in chapter 2:

- Inter-activity similarity in data output. For example, walking and running have similar patterns, which leads to difficulty in extracting meaningful features to tell them apart.
- Activity patterns are person-dependent. Not only that, but people may change their way of doing an activity over time. This may lead to discrepancies between the training and test data, and draws challenges in clustering patterns that describe the same activity.
- Wearable devices have low computational power. Combined with the fact that, especially for live activity recognition, the requirement is that calculations are done fast in order to be able to keep up with the user's pace, this means that the algorithms must have low computational cost in order to operate effectively.
- Sensory data is noisy and dependent on sensor axis placement. Hence, reliable solutions should be able to adapt their way to interpret the data and be able to distinguish the meaningful parts of data from the rest.

This project's aim is to deliver a real-time, live recognition system that is able to distinguish between five activities: sitting/standing, lying, falling, walking, running. Data has been collected from 49 students, with the help of two sensors, Respeck and Thingy, which are described in chapter 3.1.

The system is integrated as part of an Android app, which offers an intuitive user interface and privacy to the user. It lets the user create their own account, enables them to use the recognition system on their device in real time, and saves past activity

recognition sessions for them to look at it later. Further description of the mobile app's structure is provided in chapter 3.2.

To build the recognition system, we decided to develop two deep learning solutions and then pick the best one. One is a Convolutional Neural Network (CNN), and the other one is a Recurrent Neural Network architecture, namely Long Short Term Memory (LSTM). There are reasons for which we chose these two models, some of which are given in [6], namely:

- Convolutional Neural Networks (CNNs) capture the local connections of multi-modal sensory data, as well as the translational invariance introduced by locality, both of which help achieve accurate activity recognition [13].
- Recurrent neural networks (RNNs) extract the temporal dependencies and incrementally learn information through time intervals, which makes them appropriate for streaming sensory data in human activity recognition systems.

The description of each technique is given in chapter 3.3. In our setup and with the chosen evaluation metrics, both described in chapter 4, CNN gave better results than LSTM on the test data, having 98% accuracy compared to 91%. Further results are shown in chapter 5.

Each team member had a well-defined task: one built the Android app, one focused on developing the CNN model, and one created the LSTM model. This ensured we could work on tasks in parallel, so that development would happen faster. Contact among team mates has been constant through social media channels, and any updates to individual work could be seen and checked by others on the Github repos which were created for each task. Reflections on what we learnt while working on the project, and thoughts on how we can make the current solution better are given in chapter 6.

Chapter 2

Literature Review

The topic of recognising human activity has been researched intensively in the past few years. The methods used have been diverse, although it has been acknowledged that deep learning techniques deliver the better results when compared to traditional machine learning solutions.[36]

[37] Many innovative and pioneering works on the use of accelerometers for human activity recognition were published in the 90s [9]. However, the most cited work that produces satisfactory results by placing multiple sensors in different parts of the body and using different data mining algorithms is [3]. The work concluded that sensors placed on the thigh were the most effective in recognizing different activities, a finding that Kwapisz and colleagues used in order to perform HAR with a single accelerometer embedded in a smartphone [27]. Despite hand-crafting their own features from sensor data, which requires domain knowledge [30] and may lead to loss of information after feature extraction, none of the classifiers effectively distinguished between very similar activities, such as ascending and descending.

[7] The problem of needing to hand-craft features is not unique to activity recognition. For example, in image recognition [2], various types of features need to be extracted when trying to recognize handwriting instead of recognizing faces. However, in recent years, due to the ever-increasing processing power, a large number of deep learning (DL) techniques have been developed and successfully applied to recognition tasks [41] [18]. These techniques allow for the automatic extraction of features without any knowledge of the domain.

[12] The typical workflow for human activity recognition in wearable computing [5] treats individual frames of sensor data as statistically independent, i.e. isolated parts of the data are converted into feature vectors which are then given to the classifier without any more temporal context. However, during modeling, if time contexts that extend beyond frame boundaries are ignored, this can limit recognition performance for more difficult tasks. A method that incorporates the time dependence of sensor data streams would appear to be more appropriate for human activity recognition.

Due to this, recurrent deep learning methods have been gaining popularity in the field. The most prominent model based on the Long Short-Term Memory cell [18] have

been used with great success. In [34], deep recurrent neural networks have been used to identify activity on the Opportunity dataset. The LSTM model was combined with multiple previous CNN layers in a deep network that learns rich sensor representations and can respond very effectively to difficult recognition tasks. Through large-scale experiments by [13], appropriate training procedures have been analyzed for multiple HAR deep learning methods including deep LSTM networks. Most existing methods are based on (variant) sliding window procedures for frame extraction.

Other solutions that have become popular in the field include convolutional neural networks. [36] suggests that CNNs are great at predicting sports activities, energy expenditure and at tracking personal activities. [38] and [37] add that the complexity of derived features increases as the number of convolutional layers increases, but the difference in complexity between adjacent layers decreases with each added layer. [37] details further that convolutional nets achieves very high accuracy on moving activities which were previously perceived to be very difficult to classify, and states they do not require any advanced pre-processing or feature hand-crafting due to CNNs providing a way to automatically extract relevant and robust features. Generally, applying CNNs on Activity Recognition tasks has 2 advantages, according to [47]:

- Local dependencies: The local dependencies of the active signals are captured by CNN. In image recognition tasks, nearby pixels often have strong interrelationships. Similarly, in activity recognition, for a given activity, nearby acceleration readings may be correlated.
- Scale invariance: CNN preserves feature scale invariance. In image recognition, training images may have different scales. In activity recognition, a person may walk at different paces (for example, at different exercise intensities).

Chapter 3

Methodology

3.1 System Description

3.1.1 Hardware

We could use two sensors in this project:

- RESPECK: contains a 3-axis accelerometer and a 3-axis gyroscope. It was supposed to be worn under the left rib-cage, stuck with mefix tape and having the label on the back in standing position. [31] argues that it provides a very good indication of general movement and activity level.
- THINGY: contains the same as RESPECK and a 3-axis magnetometer. It was supposed to be worn in the front right-hand pocket of jeans, with the flickering blue/green light in the up-right position. [3] argues this is the best position for recognising different activities with a sensor. According to the producer's website, it is powered with a rechargeable Li-Po battery with a capacity of 1440 mAh, that can be charged via USB

3.1.2 Wireless Communication

The sensors were connected to an Android phone via Bluetooth and were set to transmit packets of data via Bluetooth Low Energy at a rate of 25Hz. This frequency ensured that components of the analysed signals were captured in great detail.

3.2 Algorithms for Human Activity Recognition

3.2.1 Convolutional Neural Networks (CNN)

This chapter describes convolutional neural networks (CNN)[25], a deep neural network specifically designed for image recognition. This technique exemplifies the importance of deep improvements for information (image) processing.

CNN is more than just a deep neural network with many hidden layers. It is a deep network that mimics how the visual cortex of the brain processes and recognizes images. As a result, even neural network experts often have a hard time understanding this concept when they first encounter it. This is how CNN differs conceptually and operationally from previous neural networks. This section provides a brief introduction to the basic architecture of CNN. An example of it is shown in figure 3.1.

Basically, image recognition is the classification task. However, regardless of the recognition method, directly using the original image for image recognition will output poor results; the image should be processed to compare features. To this end, various techniques for image feature extraction have been developed. CNN includes a feature extractor during training instead of designing it manually. CNN's feature extractors consist of special types of neural networks whose weights are determined through the training process.

When its feature extraction neural network is deeper (containing more layers), CNN produces better image recognition, but at the cost of difficulties in the training process. CNN consists of a neural network that extracts the features of the input image and another neural network that classifies the feature image. The input image enters the feature extraction network. The signals of the extracted features of the image enter the classification neural network. The classification neural network then operates on the features of the image and generates the output.

Feature extraction neural networks [28] consist of stacks of convolutional layers and pooling layer pairs. Convolutional layers, as the name suggests, are the use of convolutional operations [44] to transform images. It can be thought of as a collection of digital filters [1]. Since CNN's primary focus is on images, the operation of convolutional and pooling layers is conceptually in a two-dimensional plane. Convolutional layers gen-

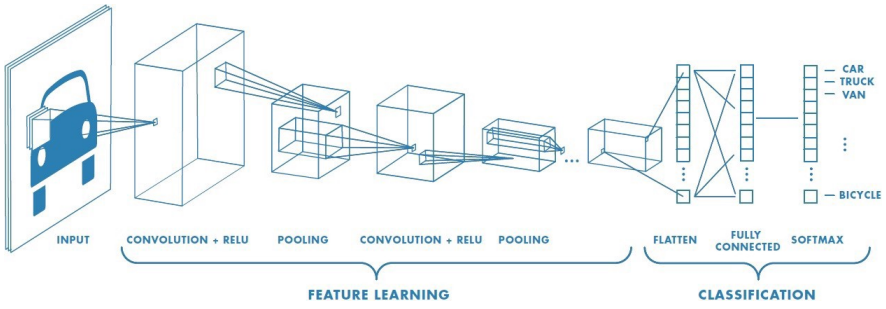

Figure 3.1: Example of a convolutional neural network

erate new images called feature maps. Feature maps highlight the unique features of the original image. Convolutional layers operate very differently than other neural network layers. This layer does not use connection weights and weighted sums. Instead, it contains filters that convert images. We call these filters convolutional filters. The process of inputting an image through a convolutional filter produces a feature map. The features extracted in the convolutional layer are determined by the trained convolutional filter. Therefore, the characteristics of convolutional layer extraction will vary depending on the convolutional filter used. The feature map created by the convolutional filter is processed by an activation function before the layer produces an output.

The activation function of the convolutional layer is the same as that of a normal neural network. Although the ReLU function has been used in most applications, sigmoid and tanh functions are often used as well.[39]

Similar to convolutional layers, pooling layers are responsible for reducing the amount of space for convolutional features. This is to reduce the computing power required to process data by reducing dimensionality. In addition, it is useful for extracting the main features of rotation and position invariance, thus maintaining the process of effective training of the model. There are two types of Pooling: Max Pooling[35] and Average Pooling[43].

Adding a fully connected layer is an (usually) inexpensive way to learn nonlinear combinations of advanced features, as represented by the output of a convolutional layer. The fully connected layer is learning possible nonlinear functions in that space. Now that we have converted the input image into a form suitable for our multi-level perceptron, we will flatten the image into column vectors.

The flattened output is fed to the feed-forward neural network and backpropagation is applied in each training iteration. Over a period of time, the model was able to distinguish between the main features in the image and certain low-level features and classify them using Softmax classification techniques.[29]

In addition to convolutional layers, we can apply batch normalisation[21]to improve learning of our neural network. Batch Normalization is designed to solve the problem of internal covariate shift, where the input distribution of each layer changes with the parameters of the previous layer during training. The authors argue that if there is no batch normalization, the distribution of each layer's inputs can vary significantly due to the stochastic nature of randomly sampling mini-batches from your training set. Batch Normalization achieves to reduce the internal covariate shift through normalization of the layers inputs as the training progresses by normalising the inputs of each layer in the neural network. This normalisation is done by subtracting the mean of the inputs and dividing by their standard deviation. The mean and standard deviation are not extracted from the whole training set but rather from a minibatch making the algorithm a lot faster. Batch normalization addresses the vanishing gradient problem[17] since by normalizing activations throughout the network, it prevents small changes in layer parameters from amplifying as the data propagates through a deep network.

3.2.2 Long Short-Term Memory (LSTM)

A Long short-term memory (LSTM) network is a type of artificial recurrent neural network that not only has the ability to learn, but also has the ability to 'remember' certain values when training on long sequences of data. This trait is very useful for time series classification problems where there can be long time lags between important events. This made it a particularly good choice for human activity recognition. Where many other models would likely fail to learn the importance of the ordering of the data, lstm networks can learn order dependence.

Recurrent neural networks are designed to train on sequential inputs and are also resistant to noise (unimportant or meaningless fluctuations in the inputs) which improves

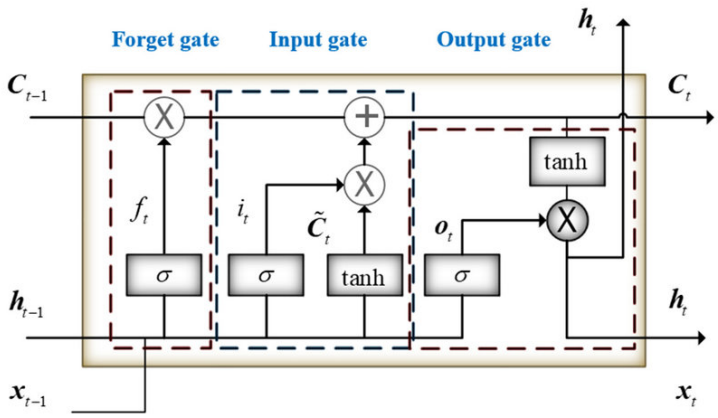

Figure 3.2: Architecture of lstm cell

their accuracy and training time while decreasing model complexity. However, recurrent neural networks struggle to learn when there are time lags much greater than 5-10 time steps between relevant inputs, with the exception of lstm. [11] Lstm networks can recall information even when there are gaps of hundreds or even a thousand time steps between the relevant inputs. It achieves this ability to recall via its 'constant error carousel'.

In a standard rnn, when an input influences the hidden layer its influence can decay exponentially and tend to zero as it cycles through the network. This is known as the 'vanishing gradient problem'. The internal structure of lstm units, shown in figure 3.2, is specifically designed to address the vanishing gradient problem, as they allow gradients to flow through unchanged, enabling them to cycle through the network without decaying. This is known as the 'constant error carousel'. [19] It should be noted that an influence on the network can also grow exponentially and tend to infinity rather than decaying and the structure of lstm units do not prevent this 'exploding gradient problem' from occurring.

A typical lstm unit is known as a cell and contains an input gate, output gate and a forget gate. Inside the input gate, the current state and the previous hidden state are passed to the sigmoid function and returns values between 0 and 1. The current state and previous hidden state are also passed to the tanh function which creates a vector with values between -1 and 1.

The forget gate decides what information is important and what information can be ignored. The current state and previous hidden state are passed to the sigmoid function, returning values between 0 and 1. If part of the old output was important it will return a value closer to 1, and if it was unimportant it will return a value closer to 0.

The output gate is the gate that determines the value of the next hidden state. The current state and previous hidden state are passed into the final sigmoid function. Then the new cell state that was generated from the initial cell state is passed to the tanh function, where the output is multiplied point by point with the output from the final sigmoid function.

After information has passed through the input gate and the forget gate the next step in an lstm unit is to store the information from the new state inside of the cell state. First, the previous cell state will be multiplied with the forget vector. If the result is 0 then the values will be dropped in the cell state. Otherwise, the next step is to take the output value of the input vector and perform point by point addition, updating the cell state of the network.

3.3 Mobile Application

3.3.1 Features & User Interface

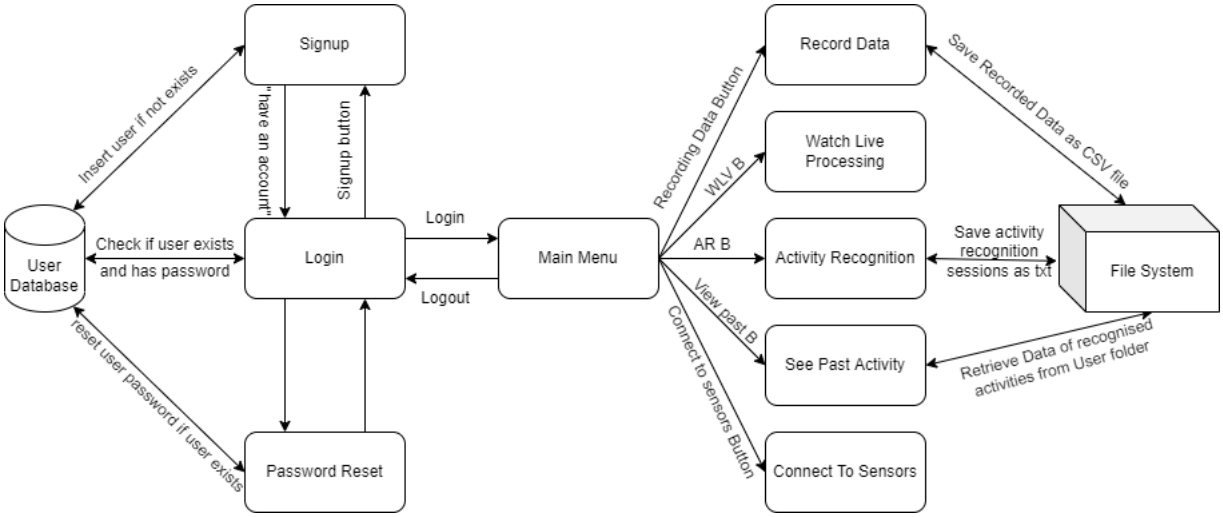

Figure 3.3: System workflow

The main scope of the app is to let the user use it for either fitness tracking or collecting data, with the help of either Respeck sensor or Thingy. Figure 3.1 shows the application workflow and how software communicates with computer resources, which the app uses for storing recordings, as well as the SQLite database, which stores user data. Each rectangle represents a screen of the application, while the arrows indicate how the user can interact with other parts of the system from that screen.

The app has been designed with simplicity and functionality in mind. The buttons have suggestive names to what they are doing and where they are leading the user to, constituting an important pillar of effective UI design [32]. The interface is clean and lightweight, with the spaced-out buttons adding to the app's user friendliness.

Privacy is another principle that was followed when designing the app. When the user enters the app, they are greeted by the login page. If they do not have an account, they can create one by going on the sign-up page and completing the required details. If they have an account, but they have forgotten their password, they can change the password by going on the afferent page and completing the form. Once they enter their correct details, they can login. This will take them to the main menu of the application. From there, they can access the multitude of features the app has to offer. Essential is that the user connects the sensor to the phone and that the app sees it. Once that is done,

they can collect data using either sensor for all sorts of purposes, or they can watch live processed data from the sensor. They can also use the app as a fitness tracker, accessing the ”live recognition” page. The user can select which sensor they want to use or can opt to use both. Once they press the start button, a timer will appear on the screen and below an image and a caption will be shown, indicating what activity the app thinks the user is doing. The user can stop the session by pressing the stop button. Once they do that, they can go in ”View past activity” and see details of their recording. The file is easily recognisable due to its name containing the date and time at which the recording took place. This kind of name format also ensures there are no two files with the same name. Opening the file unveils details such as what sensors were used during the session, the total duration and what activities the user did during the session, and for how long they did each activity. The user can delete any old recording with by long-pressing on it. Logging out is done from the main menu, and the user is driven back to the login page.

3.3.2 Software organisation

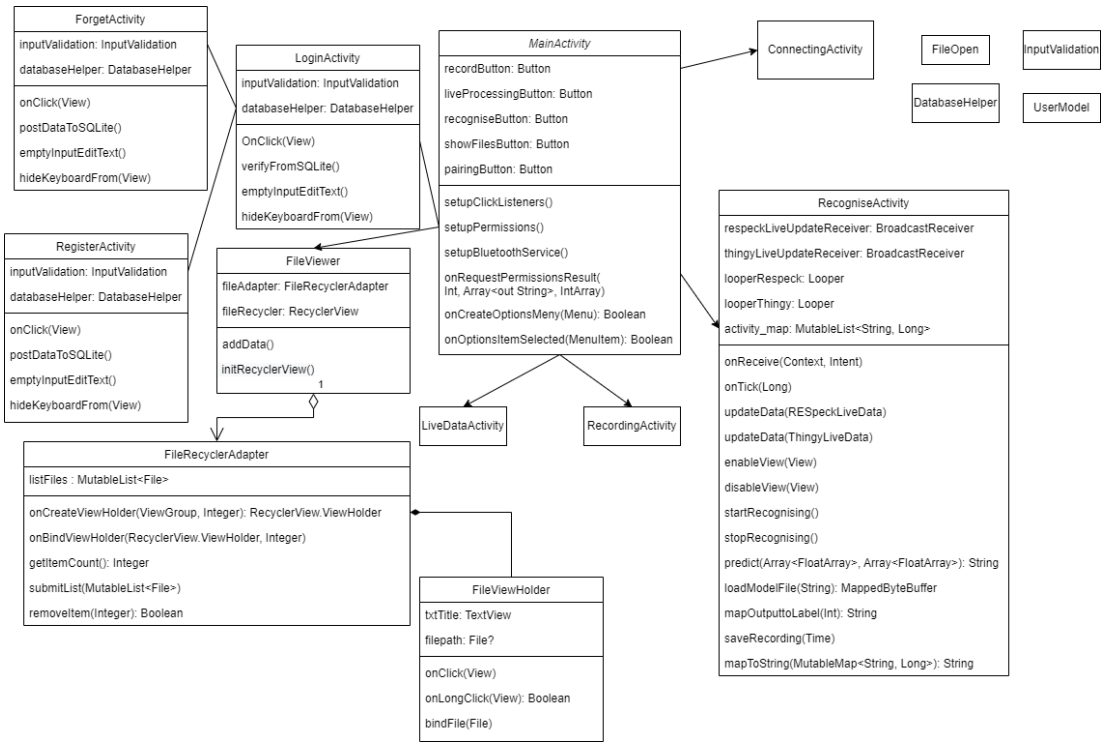

Figure 3.4: UML Diagram of added classes

Figure 3.4 presents a diagram of the newly added components to the received baseline project, as well as the already-existent main components of the project (MainActivity, ConnectingActivity, LiveDataActivity, RecordingActivity). In the top-right corner, enumerated are utility classes which were created to ease communication with the file system (FileOpen) and the SQLite database (DatabaseHelper, UserModel). InputValidation contains functions that help with checking the credentials introduced in any form present in the app.

Centered in the diagram is the MainActivity class, which corresponds to the main menu of the app. To the left of it, LoginActivity, ForgetActivity and RegisterActivity form the login system. FileViewer, FileRecyclerAdapter, FileViewHolder are classes responsible for making possible viewing files of recordings of past activity directly from the app's user interface. To the right of MainActivity, connected is RecogniseActivity, the class responsible for the activity recognition subsystem. The diagram describes only the important variables and functions from each component.

The software was designed to be modular[46], with each class having a well-defined scope. Components have suggestive names with regards to their functionality, making it easier to read and understand. The main components are independent of one another, which helps achieve high maintainability. Moreover, due to its functional design, the code is easily reusable, since each task is split into smaller, simpler sub-tasks, which are sometimes repetitive. Following is a description of all sub-systems:

- The login sub-system has 3 components: LoginActivity, ForgetActivity and RegisterActivity. LoginActivity is responsible for the functionalities present on the login screen, ForgetActivity for letting the user change their account's password, and RegisterActivity for allowing a new customer to create a new account on the app. All components communicate with the SQLite database, storing, editing and accessing user information from there. DatabaseHelper, UserModel and InputValidation come in handily here, ensuring that the communication between the database and the components is consistent.
- The backend for the activity recognition sub-system is written in the RecogniseActivity component. This is a major component, which reuses existent code from other components, in order to shorten development time and benefit from the high quality resource put to disposal [22]. It is capable of recognising what activity the user is doing, and saving recordings in the file system, which are visible only to them.
- Finally, the app is able to provide the user with access to their recordings of activity recognition sessions. The sub-system responsible for that is formed of one main component, FileViewer, and two smaller ones, FileRecyclerAdapter and FileViewHolder. The main component sets up the user interface controls, while the other two make sure that the user can see all files in the folder and that they can open/delete a file with simple commands. Showing the file's contents is facilitated by FileOpen, ensuring data from each file is presented in the same manner to the user.

Chapter 4

Testing

4.1 Environment setup

The app was developed using Android Studio SDK, with the target version being 23, unchanged from the app version we initially received. We tested with 3 different phones:

- Nokia 8: Android 9, Snapdragon 835, 4GB RAM, 64GB memory, battery capacity: 3090mAh
- Huawei P20 Lite: Android 9, Kirin 659, 4GB RAM, 64GB memory, battery capacity: 3000mAh
- Huawei Honor 10: Android 10, Kirin 710, 4GB RAM, 128GB memory, battery capacity: 3400mAh

4.2 Evaluation metrics

One activity can be classified as True Positive (TP), True Negative (TN), False Positive (FP) or False Negative (FN). A true positive is an outcome where the model correctly predicts the positive class. Similarly, a true negative is an outcome where the model correctly predicts the negative class. A false positive is an outcome where the model incorrectly predicts the positive class. And a false negative is an outcome where the model incorrectly predicts the negative class. The definitions below are taken from [36]. The following metrics were used to evaluate our models:

Accuracy provides the overall correctly classified instances. It is the sum of correct classification divide by the total number of classification.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

Precision (Specificity) measures the accuracy and provides the value based on the frac-

tion of the negative instance that are classified as negative.

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

Recall measures the performance of correctly predicted instances as positive instances.

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

F-Measure (Score), F-Measure is primarily applied to unbalanced datasets and provides a geometric average of sensitivity and specificity.

$$F-measure = \frac{2*Precison*Recall}{Precision+Recall}$$

Confusion Matrices: Confusion matrices are important performance measure, and the matrix provides the overall misclassification rate in human activity recognition [14]. Known classes are represented by rows, while columns correspond to prediction classes generated by classifiers. The use of confusion matrices allows the analysis of empty classes commonly found in human activity recognition and further visualization of system recognition performance.

Receiver Operating Characteristics (ROC) Curve: The ROC curve, also known as the exact recall, provides a mechanism for analyzing the true positive rate versus the true negative rate (FPR). However, the ROC curve only applies to detection models because it depends on the number of true and negative classes, and may not apply to unbalanced datasets common in deep learning-based identification of human activity. Metrics such as error rates that display values with accuracy equal to recall, average precision, and area under the curve (AUC) show the overall performance of the classifier and the probability that the positive instance chosen ranks higher than the negative instance [5] [14].

Leave-one-subject-out-cross-validation[45]: Leave-one-subject-out cross validation (LOSOXV) is a cross-validation method that utilizes each individual as a ”test” set. It is a specific type of k-fold cross-validation where the number of folds k equals the number of participants in the dataset. LOSOXV is the most reliable way to test models that contain participant-level data. However, LOSOXV is also the most computationally expensive. LOSOXV is recommended to validate models built on smaller datasets where standard test/training splits may introduce significant bias in the model. If you expect large differences between participants, the LOSOXV method may be the best way to validate your model.

Overfitting[15]: Overfitting is a concept which occurs when a model fits exactly against its training data. When this happens, the algorithm unfortunately cannot perform accurately against unseen data, defeating its purpose. In this case the model also adapts to the noise in the training data and this does not allow the model to predict accurately when it encounters new and unseen data.

Generalization[4]: In machine learning, generalization is a definition that demonstrates the ability of a trained model to classify or predict invisible data. Training a generalized machine learning model often means that it applies to all subsets of unseen data. An example is when we train a model to classify dogs and cats. If the model provides a dataset of images of dogs with only two breeds, it may get good performance. However, when it is also tested by dogs of other breeds, it may get a lower classification score. This problem can lead to the classification of actual dog images as cats from an unseen dataset. Therefore, data diversity is a very important factor in making good predictions. We also define generalisation gap which is the difference between validation loss and training loss.

Cohen's kappa score[42]: Kappa score is a statistic used to evaluate models based on inter-rater reliability. It is a bit more robust than a simple measure of accuracy as it takes into account the chance of an agreement happening by chance.

Matthew's Correlation Coefficient: MCC is a metric used to evaluate models based on the correlation between the predicted class and the actual class. It will only provide a good score if it gives good results in all four of the following categories of confusion matrix proportional to the size of positive and negative items in the dataset: true positive, true negative, false positive and false negative.

Chapter 5

Results and Implementation Analysis

5.1 Convolutional Neural Networks

5.1.1 Model Parameters

For our CNN (convolutional neural network), we decided to use 3 blocks consisting of a convolution layer, batch normalisation and a Relu layer, as can be seen in figure 5.1. Then we apply flattening (to make it possible to classify into different classes) and then apply a fully connected layer. The same decision was taken in a research paper which uses 3 blocks and gets very good results on the classification of 12 activites [20](lying, sitting, standing, walking, running, cycling, nordic walking, ascending stairs, descending stairs, vacuum cleaning, ironing, and rope jumping). We used a kernel size of 3. This seems to be a popular choice (according to towardsdatascience.com smaller odd sized kernel filters perform better in practice.) In addition to that, we used no pooling (the research paper did not use it either[20]). This seems to be a good choice as the use of max pooling would make computations more expensive. This would severely hurt the functionality of the app as the main purpose of it is to provide fast and accurate human activity recognition.

The Relu activation function was chosen over the sigmoid activation function as it is faster and has reduced likelihood of the gradients to vanish[17]. The gradients of sigmoids become increasingly small as the absolute values of x increase causing it to saturate at 0. However, the Relu function does not suffer from that, making Relu better optimized at dealing with vanishing gradients.

We used 1 dimensional convolutional layers instead of 2 dimensional convolutional layers. We chose this to accomodate for our input being a 6 dimensional input (this was for Respeck as we used all of the axis-both accelerometer and gyroscope). An alternative solution would have been to turn our input into a 2 dimensional image but that would make training and testing the model slower as we would be doing 2d convolutions.

We used flattening after the 3 blocks. This was in order to reduce the dimensionality from 2 dimensions into 1 dimension in order to make classifying possible. After flat-

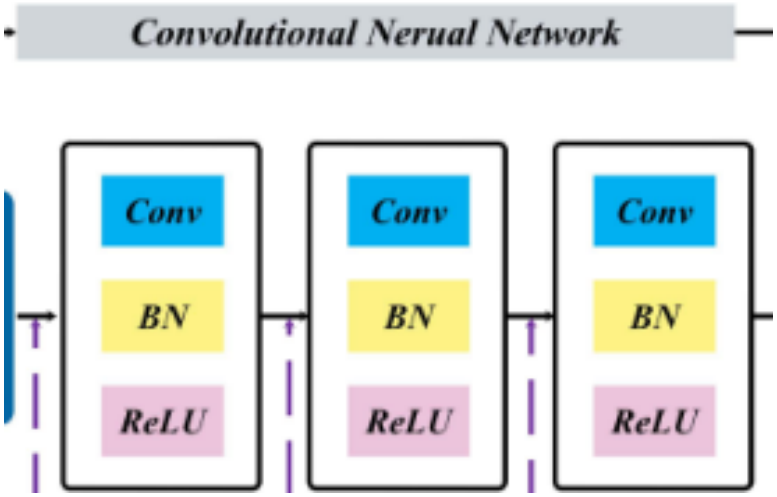

Figure 5.1: CNN Architecture

tening, we apply 2 fully connected layers in order to get the best training for the more advanced features and get the output of network as a 5 dimensional vector (as we are only classifying the subset of activites).

To optimize our training we used the Adam optimizer[26] instead of Stochastic Gradient Descent[23]. Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iteratively based on training data. The reason we picked Adam is because it is faster than stochastic gradient descent and achieves very good results. We used the default configuration of Adam as it has been shown that it works very well on most problems.

As a choice for our loss function, we went with the categorical cross-entropy error function. It works well for multi-class classification and for mutually exclusive classes. It was used along with the softmax activation function as it works better with that activation function. Also, we used a batch size of 200 and we were training for 50 epochs.

To deal with time series data, we used the concept of sliding windows - the most famous approach to dealing with human activity recognition[13]. This was done to divide sensor signals into smaller data segments. One datapoint in isolation can not be used for classification so dividing the data into small windows seems like a good idea. Its simplicity of implementation and lack of pre-processing make the windowing approach ideal for real-time applications. Here, the signal is divided into fixed-size windows, and there is no gap between windows. For some applications, overlap between adjacent windows is tolerable; however, this is rarely used. In our case, we used a window size of 30 and an step size of 15(which means a 50% overlap). A window size of 30 means that each window contains data for a duration of 1-2 seconds. There were a few motivations behind our choice of the window size. First, the falling activities are recorded for a duration of 1-2 seconds so having a larger window size would mean that we would miss all our falling activites or we would have both falling and

another activity in one window which is not desired. In addition to that, using a smaller window size could hinder our classification accuracy as some activities like walking or running could be similar for a small amount of time(less than 1 second). Figure 5.2 illustrates the concept.

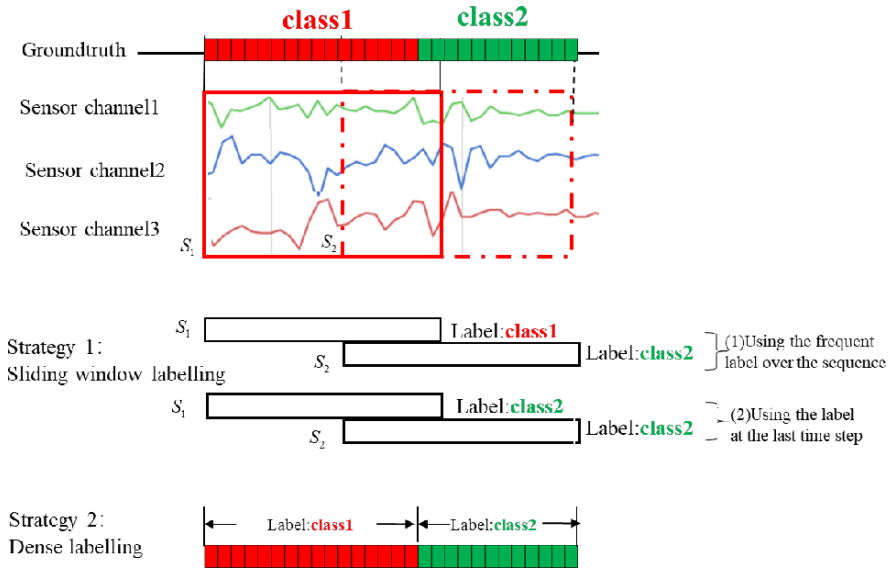

Figure 5.2: Illustration of sliding window concept

5.1.2 Model Performance

To find the optimal parameters (number of layers, batch size, epochs, window size, overlap) we used a hyper-parameter search[8]. Different models with different numbers of blocks were tested and we found that 3 blocks had the best trade-off between accuracy and training time. This was also supported by [20] where a model with 3 blocks (convolution layer, Relu layer and batch normalisation) achieved outstanding results on classifying 12 activities (90%+ accuracy). We also tried different batch sizes and found out that a batch size of 200 (which is not that high) is the sweet point between accuracy and training time. In addition to that, we noticed that training the model with 100 epochs was causing overfitting as there was a big generalisation gap. By decreasing the amount of epochs the generalisation gap became smaller while the accuracy did not go down. Furthermore, different window sizes were considered. As explained before, having a window size that contains data from a 1-2 second interval is ideal in our scenario.

We used Leave-One-Subject-Out-Cross-Validation[45] to estimate and tweak the performance of our model. Using the best model we had, in figure 5.3 we have listed the accuracy's per activity and the overall accuracy. We can notice that walking has an accuracy of 90% while other activities have accuracy over 96%. This can be attributed to the fact that a lot of people might have walked during their running recording or they might have walked during general movement activity recording. In addition to that, we can see that the precision, recall, F1 scores are very good. Sometimes the

accuracy is not a very representative measure of the overall performance of the network. This is usually the case when we are dealing with imbalanced data. Having very good precision, recall and F1 scores assures us that our system will perform good in a real life scenario as those methods will help us in detecting imbalanced data or other problems with our training or validation set. To get the accuracy that we achieved, we needed to do some data pre-processing. We ran the leave one out cross validation on all subjects ids (student's id) and only kept the data from students with a high accuracy. This method seemed to have worked as after removing the bad data, we were achieving much higher accuracy with leave one out cross-validation. The number of students that were kept was 33. The other students were removed as we assumed that they had not collected data correctly. These problems could range from not wearing sensors properly to mislabeling data.

For the specific activities:
Sitting/Standing .....: Accuracy: 0.965, Precision: 0.973, Recall: 0.965, F1-Score: 0.967
Walking .....: Accuracy: 0.906, Precision:0.891, Recall: 0.906, F1-Score: 0.897
Running .....: Accuracy: 0.963, Precision: 0.964, Recall: 0.963, F1-Score: 0.961
Lying down .....: Accuracy: 0.986, Precision: 0.973, Recall: 0.986, F1-Score: 0.979
Falling .....: Accuracy: 0.964, Precision: 0.964, Recall: 0.964, F1-Score: 0.962

Overall Metrics:
Average Testing Accuracy is 0.966, average precision is 0.964, average recall is 0.966,average F1-Score is 0.963

Figure 5.3: Leave One Subject Out Cross Validation average results for CNN

We tested our model during peer review. The model achieved an outstanding 98% accuracy, as shown in figure 5.4. All of activities had an accuracy over 98% while falling had 93% accuracy. This could be attributed to falling being quite difficult to record and data recorded from it could be similar to walking or running. Nevertheless, the results are excellent and meet the advanced criteria of 96%+ accuracy. The confusion matrix is shown in figure 5.5.

****
Classification report
****

	precision	recall	f1-score	support
0	0.98	0.99	0.99	384
1	0.97	0.99	0.98	98
2	0.99	0.99	0.99	94
3	0.97	1.00	0.99	390
4	0.98	0.94	0.96	309
accuracy			0.98	1275
macro avg	0.98	0.98	0.98	1275
weighted avg	0.98	0.98	0.98	1275

Figure 5.4: CNN results on the test set

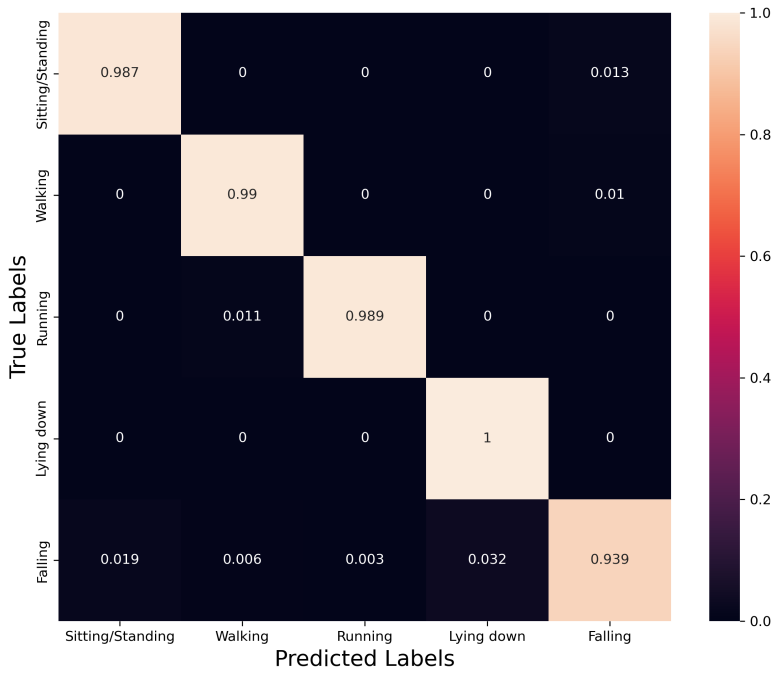

Figure 5.5: CNN confusion matrix on the test set

5.2 Long Short-Term Memory Networks

5.2.1 Model Parameters

For our LSTM model we used a single LSTM layer consisting of 96 units followed by a dropout layer to reduce overfitting. After this, there is a 48 unit densely connected layer with the RELU activation function, followed by a 5 unit densely connected output layer with the softmax activation function. The model has a kernel size of 3, a categorical cross-entropy error function as the loss function and was compiled using the adam optimizer for the same reasons mentioned in the CNN implementation analysis.

The data that was fed into the LSTM network had been pre-processed and turned into sliding windows similar as was discussed in the CNN implementation analysis, however we found that the LSTM network gave better results using a slightly smaller size of sliding window.

The model was trained for 5 epochs as after this point the validation accuracy did not continue to rise but the validation loss did, a clear sign of overfitting. A validation split of 0.1 was used so these values could be monitored throughout the training process. We used a grid search to determine the best values out of a set of given values for number of lstm layers, number of lstm units, dropout rate, batch size and window size. We found the ideal number of lstm layers to be exactly one, the best number of lstm units to be 96, the best dropout rate to be 0.3, the best batch size to be 96, and the best window size to be 26. The model has 44,453 parameters which seems to be in the

range of what is appropriate and realistic for a LSTM model trained to classify human activity.

5.2.2 Model Performance

We can see that based off of our test data, we have an accuracy of 93%, which seems good but not outstanding. Upon further inspection, we see that both the 'Standing' and 'Lying' classes perform very well in terms of both precision and recall. The network does not appear to be as good at classifying 'Falling' as it is at the other two activities but regardless, it is acceptable. However, the network clearly struggles with classifying the Walking and Running activities.

	precision	recall	f1-score	support
Standing	0.98	0.97	0.97	1830
Walking	0.88	0.70	0.78	459
Running	0.91	0.77	0.84	456
Lying	0.97	1.00	0.98	1823
Falling	0.85	0.93	0.89	1333
accuracy			0.93	5901
macro avg	0.92	0.87	0.89	5901
weighted avg	0.93	0.93	0.93	5901

Figure 5.6: LSTM results on test set

Looking at the confusion matrix, the cause of this becomes clear. The falling activity is regularly misclassified as the walking and running activities. On top of this, there is also a less extreme, but still noticeable amount of instances where both the walking and running activities are misclassified as the falling activity. It is clear the network is not as good at telling these activites apart. There could also be some issues in the way in which data was collected by subjects, such as subjects possibly including portions of walking data when completing the running activity data collection.

However it should be noted that despite this, the network maintains a respectable weighted average precision, recall and f1-score of 0.93 and a macro average f1-score of 0.89.

The Cohen kappa score was calculated to be 0.88 indicating a high inter-rater reliability. The Matthews correlation coefficient was also calculated and was 0.88 as well, indicating a strong correlation between the predicted class and the actual class.

When tested on the 'unseen Respeck recordings' data, we saw in figure 5.8 (please see page below for figures) that the network performed slightly worse in some areas and better in other areas compared to our original test data. The overall accuracy and f1 score slightly decreased but the walking class had a modestly improved precision and recall, the running class had a modestly improved precision and a massively improved recall. Additionally, the falling class saw modest improvements to recall and the lying class suffered on recall dropping from a perfect 1 down to 0.86. The overall accuracy is

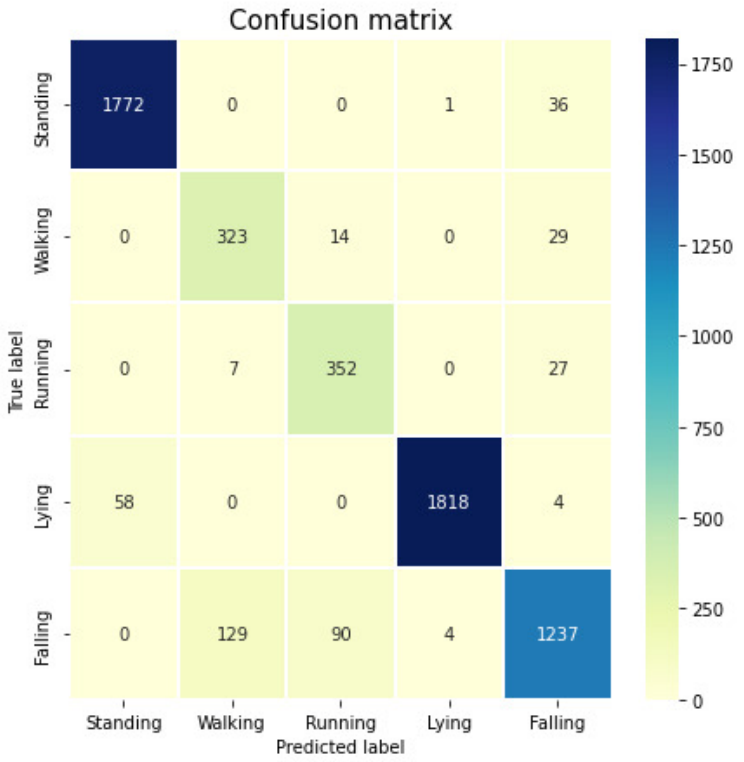

Figure 5.7: LSTM confusion matrix for test set

still 91% with this test set of data, and the weighted average f1 score is 0.91, a solid but achievable performance. Both the results from this test set of data, and the original test set of data indicate that in a real-world scenario it would provide decent performance even if it was not the ideal model for our implementation. The confusion matrix can be seen in figure 5.9.

	precision	recall	f1-score	support
Standing	0.88	0.93	0.90	422
Walking	1.00	0.79	0.88	145
Running	0.81	1.00	0.89	89
Lying	1.00	0.86	0.92	530
Falling	0.83	1.00	0.91	309
accuracy			0.91	1495
macro avg	0.90	0.91	0.90	1495
weighted avg	0.92	0.91	0.91	1495

Figure 5.8: LSTM results on the unseen respeck recordings test set

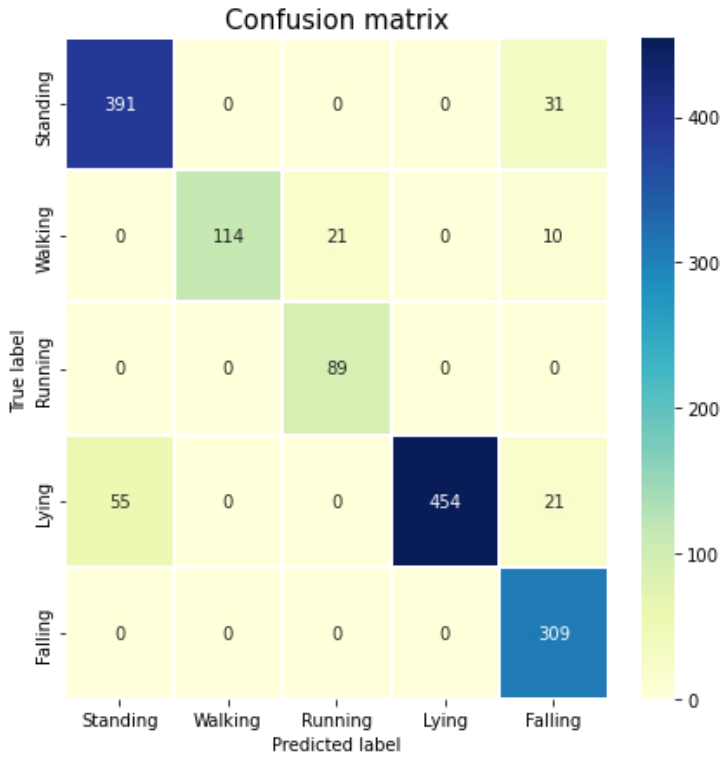

Figure 5.9: LSTM confusion matrix for the unseen respeck recordings test set

Chapter 6

Conclusion

6.1 Reflection

Now, at the end of the course, we have achieved a model with high performance when it comes to classifying human activity recognition and have also built a clean and functional app to use for our implementation. We are very satisfied with what we have achieved but can see scope for further improvement. To reflect on the project we would like to talk about some of the things we felt we learned most about and also some of the challenges we faced.

We have become a lot more comfortable with using machine learning technologies and the libraries associated with them, such as pandas, sklearn and the tensorflow framework. Building a model from scratch and feeding it data to process in real-time, so that the user can see the classification without delay, felt like an achievement to us and was a great learning process. It also increased our awareness of how difficult it is to build a reliable system of industry standard.

We also had to undertake a lot of research when choosing which models to attempt to train, how to best evaluate these models and also how to deal with time-series data among many more things. From learning the state-of-the-art in human activity recognition algorithms to learning how to structure a group technical report, we have learned many things that we had not previously touched on before. Our research skills have definitely improved alongside our technical ones.

We learned how important the process of data collection is and also learned of the challenges faced when using data that has been collected by individuals. Some errors were made during data collection by some subjects which gave us the opportunity to try different methods to solve this issue. Cross-validation was a helpful tool with this, but having a look into the data itself and identifying which subjects could have had the most errors during data collection (sensor oriented wrong way, etc.) and then removing these subjects from the training set led to great gains in performance for our model. It became clear to us how important the quality of data and the pre-processing of such data is when it comes to building a high-performance model.

We also learned skills in android programming and learned to use Kotlin. We applied

real software design and engineering principles when creating the application that our implementation uses. This was our first time developing an application that is modular, with code that can be easily reused for other purposes, and was also our first time implementing a machine learning model into an application.

6.2 Future Work

One way the project may be extended and the implementation improved is a better method of sensor fusion. Our implementation can make use of both the Respeck and Thingy sensors simultaneously but our method of sensor fusion is very rudimentary. We simply introduce a model trained on the Thingy data and then allow both our Respeck model and the Thingy model to classify. The model with the highest confidence in their classification will return their classification and this will output on the app. If a more complex and well-informed implementation of sensor fusion was achieved then it would likely substantially increase the reliability of classifications given by the network.

In addition to that we could further improve our model's performance by employing dropout[40] and Max-out Networks[10]. Max-out networks were proved to work well with Dropout so a combination of both could result in lower generalisation gap[10]. In addition to that, we could also add regularization(namely L2 regularization[33]). Regularization would prevent the model's weights from getting too big and hence helping in the model generalising better. Also, we could employ Residual Connections[16] which would help with the vanishing gradient problem.

Another way in which we could extend our project is by making use of machine learning in the cloud. Machine learning is a very resource-intensive task, especially when working with large datasets. Being able to utilize the large amount of fast GPU's that cloud computing servers use would dramatically increase the speed at which we could run train our model and perform experiments. It would also be especially useful for testing different models (or different parameters for a model, etc.) in parallel, greatly speeding up the process at which we could improve and fine-tune our implementation. Additionally, if we chose to collect more data to expand our training and test sets by a large amount in order to attempt to further improve our model, cloud computing would improve the feasibility of working with these larger datasets.

Bibliography

[1] Andreas Antoniou. *Digital filters*. McGraw Hill New York, NY, USA:, 1993.

[2] Ulas Bagci and Li Bai. A comparison of daubechies and gabor wavelets for classification of mr images. In *2007 IEEE International Conference on Signal Processing and Communications*, pages 676–679. IEEE, 2007.

[3] Ling Bao and Stephen S. Intille. Activity recognition from user-annotated acceleration data. In Alois Ferscha and Friedemann Mattern, editors, *Pervasive Computing*, pages 1–17, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[4] Pietro Barbiero, Giovanni Squillero, and Alberto Tonda. Modeling generalization in machine learning: A methodological and computational study, 2020.

[5] Andreas Bulling, Ulf Blanke, and Bernt Schiele. A tutorial on human activity recognition using body-worn inertial sensors. *ACM Computing Surveys (CSUR)*, 46(3):1–33, 2014.

[6] Kaixuan Chen, Dalin Zhang, Lina Yao, Bin Guo, Zhiwen Yu, and Yunhao Liu. Deep learning for sensor-based human activity recognition: Overview, challenges, and opportunities. *ACM Comput. Surv.*, 54(4), may 2021.

[7] Yuwen Chen, Kunhua Zhong, Ju Zhang, Qilong Sun, Xueliang Zhao, et al. Lstm networks for mobile human activity recognition. In *Proceedings of the 2016 International Conference on Artificial Intelligence: Technologies and Applications, Bangkok, Thailand*, pages 24–25, 2016.

[8] Marc Claesen and Bart De Moor. Hyperparameter search in machine learning. *arXiv preprint arXiv:1502.02127*, 2015.

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

[10] Ian Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. In *International conference on machine learning*, pages 1319–1327. PMLR, 2013.

[11] Alex Graves, Marcus Liwicki, Santiago Fernández, Roman Bertolami, Horst Bunke, and Jürgen Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(5):855–868, 2009.

[12] Yu Guan and Thomas Plötz. Ensembles of deep lstm learners for activity recognition using wearables. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 1(2):1–28, 2017.

[13] Nils Y. Hammerla, Shane Halloran, and Thomas Ploetz. Deep, convolutional, and recurrent models for human activity recognition using wearables, 2016.

[14] Nils Yannick Hammerla. *Activity recognition in naturalistic environments using body-worn sensors*. PhD thesis, Newcastle University, 2015.

[15] Douglas M. Hawkins. The problem of overfitting. *Journal of Chemical Information and Computer Sciences*, 44(1):1–12, 2004. PMID: 14741005.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[17] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

[18] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[19] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997.

[20] Wenbo Huang, Lei Zhang, Wenbin Gao, Fuhong Min, and Jun He. Shallow convolutional neural networks for human activity recognition using wearable sensors. *IEEE Transactions on Instrumentation and Measurement*, 70:1–11, 2021.

[21] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[22] M. Jha and P. Maheshwari. Reusing code for modernization of legacy systems. In *13th IEEE International Workshop on Software Technology and Engineering Practice (STEP'05)*, pages 102–114, 2005.

[23] Nikhil Ketkar. Stochastic gradient descent. In *Deep learning with Python*, pages 113–132. Springer, 2017.

[24] Eunju Kim, Sumi Helal, and Diane Cook. Human activity recognition and pattern discovery. *IEEE Pervasive Computing*, 9(1):48–53, 2010.

[25] Phil Kim. *Convolutional Neural Network*, pages 121–147. Apress, Berkeley, CA, 2017.

[26] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[27] Jennifer R Kwapisz, Gary M Weiss, and Samuel A Moore. Activity recognition using cell phone accelerometers. *ACM SigKDD Explorations Newsletter*, 12(2):74–82, 2011.

[28] M.D. Levine. Feature extraction: A survey. *Proceedings of the IEEE*, 57(8):1391–1407, 1969.

[29] Weiyang Liu, Yandong Wen, Zhiding Yu, and Meng Yang. Large-margin softmax loss for convolutional neural networks. In *ICML*, volume 2, page 7, 2016.

[30] David G Lowe. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1150–1157. Ieee, 1999.

[31] Janek Mann, R Rabinovich, Andrew Bates, S Giavedoni, W MacNee, and DK Arvind. Simultaneous activity and respiratory monitoring using an accelerometer. In *2011 International Conference on Body Sensor Networks*, pages 139–143. IEEE, 2011.

[32] Everett N McKay. *UI is communication: How to design intuitive, user centered interfaces by focusing on effective communication*. Newnes, 2013.

[33] Robert C Moore and John DeNero. L1 and l2 regularization for multiclass hinge loss models. 2011.

[34] Francisco Javier Ordóñez Morales and Daniel Roggen. Deep convolutional feature transfer across mobile activity recognition domains, sensor modalities and locations. In *Proceedings of the 2016 ACM International Symposium on Wearable Computers*, pages 92–99, 2016.

[35] Naila Murray and Florent Perronnin. Generalized max pooling. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

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

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

[38] Charissa Ann Ronaoo and Sung-Bae Cho. Evaluation of deep convolutional neural network architectures for human activity recognition with smartphone sensors. , pages 858–860, 2015.

[39] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. Activation functions in neural networks. *towards data science*, 6(12):310–316, 2017.

[40] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[41] Yichuan Tang, Ruslan Salakhutdinov, and Geoffrey Hinton. Robust boltzmann machines for recognition and denoising. In *2012 IEEE conference on computer vision and pattern recognition*, pages 2264–2271. IEEE, 2012.

[42] Anthony J. Viera and Joanne M. Garrett. Understanding interobserver agreement: the kappa statistic. *Family Medicine*, 37.5:360–363, 2005.

[43] Shuihua Wang, Yongyan Jiang, Xiaoxia Hou, Hong Cheng, and Sidan Du. Cerebral micro-bleed detection based on the convolution neural network with rank based average pooling. *IEEE Access*, 5:16576–16583, 2017.

[44] Eric W Weisstein. Convolution. *https://mathworld. wolfram. com/*, 2003.

[45] Tzu-Tsung Wong. Performance evaluation of classification algorithms by k-fold and leave-one-out cross validation. *Pattern Recognition*, 48(9):2839–2846, 2015.

[46] Stephen S. Yau and Jeffery J.-P. Tsai. A survey of software design techniques. *IEEE Transactions on Software Engineering*, SE-12(6):713–721, 1986.

[47] Ming Zeng, Le T Nguyen, Bo Yu, Ole J Mengshoel, Jiang Zhu, Pang Wu, and Joy Zhang. Convolutional neural networks for human activity recognition using mobile sensors. In *6th international conference on mobile computing, applications and services*, pages 197–205. IEEE, 2014.