MANCHESTER
1824
The University of Manchester

Human Activity Recognition through LSTM RNN and CNN using accelerometer data

Third Year Individual Project – Final Report

April 2020

Talha Jamal
10138573

Supervisor: Dr. Alex Casson

Department of Electrical & Electronic Engineering

Contents

Total word count: 1191

Appendix A:

The purpose of this appendix is to describe how the work for this project got affected due to the ongoing global pandemic caused by Covid-19.

Abstract

This project focuses on creating a Deep Learning model that can accurately identify physical activities of human beings based on the accelerometer data collected from smart-phones or smart-watches.

This can be achieved by creating an LSTM RNN and a CNN model that will be trained on the labelled dataset provided by the International SPHERE competition where participants wore an accelerometer device for approximately 30 minutes and were monitored with cameras to note down their physical activities. The dataset for about 10 people is available with each individual wearing the accelerometer for about 30 minutes – giving us an overall data of almost 5 hours to train the algorithm on. However, the labelled data for 8 people will be used to train the algorithm on and then the labelled data for the last 2 people will be used to test the accuracy of the algorithm. To validate the ability of the algorithm to recognize the physical activity of individuals through accelerometer data, the algorithm will be tested again on another labelled dataset provided by WISDM. The capability of either classifier – the LSTM RNN classifier or the CNN classifier – in terms of correctly predicting the physical human activity based on the accelerometer data will be in terms of its consistency in its accuracy, going from the SPHERE data – which it was trained on – to the WISDM data which was completely unrelated to it.

1. Introduction

1.1. Motivation

The technological revolution has transformed how medical services around the world have treated patients. It has greatly improved the quality of life that most people lead now. The innovation within the hardware devices built by scientists and engineers has allowed for better measurement of data that health services need for a certain health issue to be treated. As this curve of innovation in engineering flattens out with the span of time, it is important that the data these devices produce are utilised in the right manner to improve the efficiency of medical diagnosis. With healthcare's cost rising exponentially, improving the efficiency of the process of medical diagnosis becomes all the more important. One certain physical attribute that is already tracked for in various medical researches is physical human activity.

Research suggests that the physical activities led by people can affect them physically, mentally, socially and cognitively. A sedentary lifestyle can lead to a number of diseases, including but not limited to several forms of cancer, diabetes, hypertension, coronary and cerebrovascular diseases, and obesity etc [1]. Similarly, research establishes a correlation between the amount of physical activity undertaken and mental wellbeing. With 13 percent of the total globe disease burden accounting to mental health, depression is predicted to be the leading cause for it by 2030 [2]. That can be countered with exercise as research suggests that exercise does improve the mental health of people affected by the disease [3].

The diagnosis is pretty clear at this point but till now the means with which psychiatrists and medical practitioners could treat and monitor patients was not ideal. Their diagnosis would mostly depend on the patient's detailed insights into themselves. With a short and limited contact time with each patient, there is a huge deficit in the data that must be needed by the medical practitioner for a solid diagnosis. That vacuum can be filled with data collected from smart watches or phones of people if it can be accurately translated to useful bits. Close to 78% of adults within the UK owning a smart phone [4]. The ease of accessibility makes digital phenotyping a very plausible global diagnosis system. With almost everyone already owning a smart phone, no additional device needs to be bought for diagnosis, thus producing nearly no additional costs to anyone.

With the right Machine Learning algorithms, this data can be processed to transform the vast amounts of data into valuable insights into people's physical, mental and cognitive health.

1.2. Purpose and Objective

The data collected from the accelerometer of the wearable devices is the passive data used for this project. It is possible to establish a correlation and identify human activities through the accelerometer data of the person wearing the wearable device. It measures acceleration along three different axes:

1. X-axis: left to right
2. Y-axis: top to bottom
3. Z-axis: front to back

This acceleration is measured along each above-mentioned axis in units of g where 1 g = 9.81 m/s^2. This data is collected at different rates in different devices. For the purpose of training of the classifier, the supplied data is sampled at 20Hz. However, as different data is sampled at different rates, if a new dataset is used for testing of the algorithm's performance, it will be down sampled to 20 Hz when it will be fed into the algorithm. The aim is to correctly classify the physical activity of people based on the data from the accelerometer that was worn by them at the time.

1.3. Literature Review

Research on human activity recognition has always been of particular interest to many scientists. The research for the creation of such a machine learning algorithm that can identify complex human physical activity in real time setting is still on going. There are various sensors that can be used to identify human activity. Accelerometers are commonly used as they include many features which help in activity classification.

In [5], the authors went beyond extracting the basic statistical (mean, variance, etc) and spatial features (entropy and energy) present within accelerometer data, and used the first order derivative of an acceleration signal to distinguish the variations within the signal. The acceleration data was divided into overlapping frames with features computed for each individual frame, which were then used to classify the activities using classifiers like boosted decision stumps, support vector machine and regularized logistic regression, with the extra feature extraction method producing a 2.5 – 3% increase in the overall classification accuracies.

In [6], the authors proposed a new Haar-like filtering technique to use a difference filter combined with a variable filter to extract features from a 3D acceleration signal. A 1D filter with variable filter and shift width was used as the basis of Haar-like filtering. The resulting feature is the sum of the output Haar-like filtered signals with different shifted values and can be used on one axis only. An additional 1D Haar like filtering is carried out between the feature results from the two axes using the sum of their difference with appropriate shift values. An integral signal is introduced on to the final feature which concludes the overall Haar-like feature to be calculated using one addition, one subtraction and one bit-shift. The results produced an extremely high accuracy of 93.91%, reduced feature extraction costs by 21.22% comparatively, and resulted in a 17.8% less redundantly trained model.

The authors in [7] identified that human activity recognition using computer vision through video surveillance etc causes privacy invasion, and therefore used a single wearable device for their research – a method that can be used universally and hence is more appropriate. Volunteers for data collection wore a Microsoft band 2 on their wrist for sample collection. 1500 and 800 samples for each activity were left for training and validating the model, respectively. Different number of decision tree classifiers were used, with 30 being the number that gave the highest overall accuracy of 90%.

Compared to wearable sensors, researchers in [8] used a smartphone's accelerometer to classify physical activity using a machine learning method called support vector machine and to track it in real time using an android phone (OS). Data from the accelerometer, gyroscope, and accelerometer sensor linearity were collected and pre-processed. Multiple features like mean, standard deviation, and energy of signal were extracted to train the algorithm. For the real time processing of data, an interval of 3-5 seconds was needed for the model to make a prediction. This new model got an accuracy of 93.38% when used on the validation dataset.

The above-mentioned techniques such as support vector machine or random forest classification are not preferred anymore as they need handcrafted features which are time consuming and add external bias to the data. Therefore, nowadays deep learning algorithms like CNNs and RNN are preferred as they do not need hand crafted features and can make use of a sequence of data streams.

Research in [9] compared 9 different machine learning and deep learning algorithms to test their performance when it comes to human activity recognition. With more information fed available, deep learning models were deemed to almost always outperform machine learning models. The research concluded that deep learning models that utilised LSTM networks (explained in section 3.2) performed much better compared to other models.

Researchers in South Korea [10] used a 1D CNN classifier to recognise human activity using a triaxial accelerometer's data collected from user's smartphones. The x, y, and z axis acceleration data were converted vector magnitudes and then fed into the neural network. It was then used to classify the user's physical activity into three categories: walking, running, and staying still. The problem with normal human activity recognition is that the position of the device, e.g. user's hand, user's pocket, user's bag, etc, can have a major impact on the rotational component within the triaxial acceleration data. To keep this to a minimum, the research team transformed the x, y, and z axis data into a single vector magnitude using the Euclidean norm x, y, and z using the following formula [10]:

$$\| a \| = \sqrt{x^2 + y^2 + z^2} \tag{1}$$

This vector magnitude was then transformed into two more further vectors, ten and twenty seconds long and fed as input to the CNN. Their experimentation and evaluation led them to know that in terms of ternary physical activities, the CNN method outperformed the Random Forest Classifier. Better results were observed when the sequence of data fed into the CNN consisted of 'longer' samples in terms of time.

Another research in China [11] compared the performance of a Convolutional Neuron Network with a Support Vector Machine on 3D accelerometer data from a smartphone's sensor to see which one performed better. The 3D accelerometer data was used to train the well-designed CNN whereas 6 different kinds of features were extracted from the accelerometer data before they were used to train the SVM. As this research used an accelerometer sensor that directly gave collected 3D data from a single sensor, it saved a lot of pre-processing time. The 1D CNN model outperformed the SVM as the CNN achieved an accuracy of 91.97% whereas the SVM managed to achieve only 82.27%.

A different way to recognising short-time physical activity was considered in [12], where a CNN was trained with an over-complete pattern library full of activity patterns collected by a wearable device using a method called sliding window. The CNN used the over-complete pattern library to extract the robust features of the data from it and gave a 95.52% accuracy on a validation set.

A different deep learning model commonly used now when working with time series data is LSTM RNN – Long Short-Term Memory Recurrent Neural Networks. Generally, CNNs are more commonly used in deep learning but because people are utilising the benefits of LSTM cells much better now, the use of RNNs is on the rise as well. A research team in South Africa [13] used a LSTM RNN deep neural network architecture to classify 6 different activities based on the WISDM dataset described in section 2. The 6 activities were jogging, sitting, standing, walking, walking upstairs and walking downstairs. The x, y, and z axis were transformed into segments that lasted 10 seconds and then fed into the neural network as input. This ensured that there was ample amount of time to assign the data to the most common activity group found in that interval. It is also an appropriate enough time as in general an activity is performed for a period of time. The data was split using an 80:20 ratio for training and testing, and the remaining training data was split again into training and testing data using an 80:20 ratio – in order to have a validation dataset. Random selections of the training data were sub-sampled, used to train the model and tested on the validation set [13]. The model achieved results up to an accuracy of 90% after experimentation with different parameters.

In [14], researchers looked at producing a multi-layer parallel LSTM RNN model that could classify human activity using smartphone sensor data. Their findings produced better results than traditional machine learning methods and also matched the results produced generally by CNNs as their LSTM model got an accuracy of 94%. However, they also found LSTM RNNs were much less computationally complex than CNNs. The computational complexity of the network decreased with the number of parallel LSTM units. From this it was understood that increasing the number of neurons in a parallel LSTM unit bumped up the performance of the network slightly.

As the number of trainable parameters increase with the bigger data that is available to us, there is also a need of increasing the speed of the process of human activity recognition so that it can be readily available with real time processing. Researchers in [15] did exactly that by utilising a technique called data parallelism – distributing the data across different nodes, performing forward and backward propagation on them in parallel, with the gradients from the backpropagation averaged over all nodes as they are used to update the model of each single node, thus speeding up the whole process – on cloud computing using Kubernetes containers. They trained and ran a LSTM RNN model on online CPUs and GPU and analysed how this improved model learning performance. Their experiments proved how this technique is practical enough, as it reduced computational time by up to 10 times less on an online CPU and GPU using a Kubernetes container, than on a local CPU or GPU.

1.4. Approach and Methodology

Building on the knowledge acquired and researches mentioned in the section 1.3, the focus of this project is to create Deep Learning models that can accurately predict human physical activity through accelerometer data. This will be tested with two different classifiers: an CNN classifier, and a LSTM RNN classifier. The steps for this are threefold:

- Creating two classifiers – a Convolutional Neural Network and a LSTM Recurrent Neural Network for human activity recognition and training them on a labelled dataset from the SPHERE competition.
- Validating the algorithms by testing them on a labelled dataset provided by the SPHERE competition.
- Applying the optimized classifier on WISDM's data to recognize human activities and analysing how each classifier performed when it came to a completely unseen dataset.

The results will then be analysed to see how each model performs overall, their individual interesting features will be discussed, and it will be seen if the original theoretical belief – that LSTM RNN models work better on time series data compared to CNN models – stands.

2. Datasets

2.1. SPHERE dataset

The dataset provided by the SPHERE competition will be used to train the algorithm. The data available was measured on 10 people wearing accelerometers on their wrists performing multiple activities inside a designated location where they were monitored using RGB-Depth cameras. The video footage was used to classify their physical activities during the time they were monitored. Each participant was monitored for almost 30 minutes and so the data available is of approximately 5 hours. The accelerometer measured data at a sampling rate of 20 Hz and in the range of ± 8g. The physical activities are classified into 22 different categories: ascending stairs, descending stairs, jumping, walking with a load, walking, bending, kneeling, lying, sitting, squatting, standing, standing to bending, kneeling to standing, lying to sitting, sitting to lying, sitting to standing, standing to kneeling, standing to sitting, bending to standing, and turning. These 22 different activities come under 3 umbrella terms – ambulation activity (activity that requires continuous movement), static postures (when participants are stationary), and posture to posture transitions. These 22 different categories will be combined into 5 main categories for the purpose of training the model: walking, walking upstairs, walking downstairs, sitting, and standing.

2.2. WISDM dataset

WISDM dataset is collected and provided by the Wireless Sensor Data Mining Lab which had collected the data from 36 different users using triaxial accelerometers. The sampling rate for its measured data is 20 Hz and the range within which the accelerometer measured the values is ± 2g. This dataset classified the physical activities into 6 different ones: walking, walking upstairs, walking downstairs, jogging, sitting and standing. When testing the algorithm on this labelled dataset, the jogging data will be removed as the algorithm will not have been trained on that data so will be unable to predict it from the accelerometer values. The distribution of these classes is as follows:

Activity	Number of samples	Proportion
Walking	424,400	38.6%
Jogging	342,177	31.2%
Walking Upstairs	122,869	11.2%
Walking Downstairs	100,427	9.1%
Sitting	59,939	5.5%
Standing	48,395	4.4%

Table 1. The distribution of Activities within the WISDM dataset

SPHERE's dataset originally contained 22 different activities. They were combined into lesser number of categories in total because there are very few datasets available to validate this model based on those 22 different categories. As different datasets use different activities, the final number of activities that the training dataset should be combined into would depend on the dataset the model is to be validated on. The WISDM dataset contains 6 different activities – walking, jogging, standing, sitting, walking upstairs and walking downstairs. The WISDM dataset is generally considered very reliable when it comes to human activity recognition based on the number of researchers that have used it in their deep learning models. Therefore, it was chosen as the validation dataset and the activities of the training dataset were then to be combined accordingly. As there is no data present for jogging within the SPHERE dataset, the final 5 activities were chosen to be walking, standing, sitting, walking upstairs, and walking downstairs. Figure 1 below shows an example of what the triaxial accelerometer values look like for a physical activity.

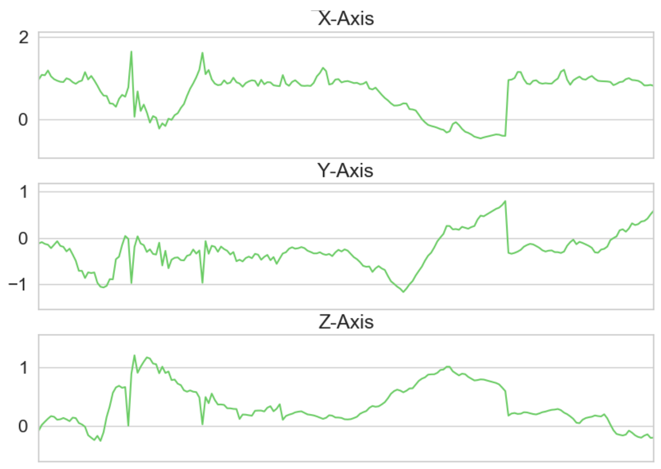


Figure 1. An example to show the accelerometer values of a physical activity

3. Theoretical background

3.1. Convolutional Neural Networks

Convolutional Neural Networks are a type of neural networks that are generally used for the purpose of image recognition. It utilises the concept of using local receptive fields in order to identify certain aspects or features within the image fed into it which then can help it identify it from a group of different images. A CNN has multiple layers – pooling layer, non-linearity layer, convolutional layer and fully connected layer [16]. The name comes from the mathematical concept of convolution where two functions are combined to form a third function – over here the two functions would be the input combined with a filter of weights which are trained using backpropagation (an algorithm that uses gradient descent for supervised learning of neural networks) which then produce a third function: a feature map.

As mentioned earlier, the convolutional layer has parameters (weights) but layers like pooling layer and fully connected layer do not [16]. This greatly improves the performance of convolutional neural networks compared to classic artificial neural networks as this greatly reduces the overall number of parameters within a network. This can allow convolutional neural networks to be used for much larger applications across multiple industries by developers. Structurally, a CNN contains the following three main layers:

- **Convolutional Layer:**
  The main responsibility of this layer is to extract useful features such as edges from the input, and as the network starts to grow deeper, more complex features such as shapes and digits are then identified [17]. It consists of two matrices – where one matrix represents a

selected portion of the fed input data and the second matrix represents a filter/kernel which is known as a set of learnable parameters. The Kernel matrix slides over the selected portion of the input data matrix to produce a final **feature matrix**. Compared to the original input data matrix, this feature matrix has lesser dimensions and has more clear features than it [17]. Simply put, the convolutional filter (kernel) slides over the input data matrix, producing a dot product at every step and the value from that convolution is then represented in the feature matrix. This feature matrix is then used for all further processing.

- **Pooling Layer:**

  The main purpose of this layer is to progressively reduce the size of the feature matrix in order reducing the computation required to process all of the data. Just like features were extracted previously from a much larger matrix, this time those features are 'pooled' together. This is done by extracting the dominant features from within sections of the feature matrix and thus the dimensions are reduced [17].

  This down sampling of the feature matrix to reduce its complexity in the calculations within further layers can be simply understood by the simple concept of reducing the resolution of an object, for instance, a photo – where the important aspects are preserved for the new image to look like the original image [16]. This reduction in number of parameters can help in terms of removing unwanted noise and controlling overfitting.

  Maximum Pooling is the most common method used right now, where the maximum of all values within the region being looked at is taken and assigned to the corresponding index of the final matrix produced by the pooling layer.

- **Fully Connected Layer:**

  After identifying the complex features within the image and then reducing the size of those features, the identification of the images can finally be done. The image is to be flattened into a single column vector which can then be fed into a feed forward neural network where backpropagation at every step is utilised to ensure that over a series of epochs, the neural network can differentiate between dominant and less dominant features within the data and correctly utilise them for classification [17]. By connecting one or more fully connected layers it is possible for the CNN to establish non-linear dependencies as well.

Another important feature within a neural network is the **activation function**. They can be placed within or at the end of a neural network. They allow a neural network to perform non-linear transformation on the linear mapping between the input and the output of the layer it's connected between [18]. In simple words, when applied to an output of 5 different categories, the activation function helps in predicting which category is most probably true and therefore returns a 1 for that category and a 0 for the rest. The activation function has a threshold which, if met, causes it to return a 1, or a 0 otherwise. The three most commonly used activation functions include: sigmoid, Tanh – hyperbolic tangent, and ReLu – Rectified linear units.

Rectified Linear Unit has been very commonly used these days. The reason for that is mostly the simple math behind it which makes it easier to apply it to classifiers with a high number of layers. It's simple logic is shown in the following two equations:

$$If\ x < 0: R(x) = 0 \tag{1}$$

$$if\ x \geq 0: R(x) = x \tag{2}$$

This shows that the gradient of the graph of the function at any point will be either 0 or 1. At no point can the gradient saturate, hence this activation function avoids the vanishing gradient problem. Although the one negative aspect is that if a certain neuron never gets activated for any given input, the gradient will always be zero for it and a dead gradient problem may occur.

The limitation with the rectified linear unit activation function is that it should only be used within the hidden layers of a neural network. Hence, for the last output layer of the convolutional neural network a **Soft-max layer** is used. The softmax function is able to convert the output of the last linear layer of a multi-class neural network into probabilities of which class is most likely to be the correctly predicted one [19]. It produces a probability distribution of the predicted classes – the sum of which should add up to one.

When the three main layers – convolution layer, pooling layer and fully connected layer – are combined with an activation function and a softmax layer, the whole process of taking inputs first into the convolutional layer to getting the output at the output layer is called **forward propagation**. Each individual layer is given the input from the previous layer and processes it as per the activation function and passes each output to the next successive layer. Feed forward networks are important for forward propagation; in order for an output to be generated, it is imperative that no data is fed backwards in the network during the output generation cycle [20]. This takes place at each neuron in a hidden or output layer in two steps: pre-activation phase and activation phase.

At the pre-activation phase, the weighted sum of inputs – the linear transformation of the weights with regards to the inputs is available [20]. This aggregated sum is then taken and based on the activation function the neuron decides if this information is to be passed on or not. This calculated sum of weights is then transmitted to the activation phase where non-linearity is added to the network based on the mathematical function of the activation function.

The neural networks use a random value of weights when they are first initialised. This might seem counter-intuitive, but in supervised learning it is not where you start off in terms of the weight initialisation of the neurons within each layer. If the network is trained enough times and the classifier is robust and pervasive enough, eventually the correct weights are reached. These weights of each neuron within each layer have to be constantly updated and this is done through backward propagation.

Once there is an output from the network in one iteration, the output is compared to the actual value. This is done by the loss function – a generalizable performance metric that enables the user to understand how close the Neural Network's prediction is to the Actual output [21]. The aim of the machine learning program is to always minimise this loss function to zero. The gradient of the loss function is used to optimize the values of the internal weights of the network in order to reduce the value of the loss function [21]. This gradient helps the user understand how much the value of the total error will change if we change the weights of the neuron by any small arbitrary value [21]. A positive gradient would point at reducing the size of the weights as at that randomly initialised weight if you increase the weight size, the total error would increase. A negative gradient would point at increasing the weights as at this current randomly initialised weight, the total error is decreasing. The aim is to get to a gradient of 0 after this process of increasing or reducing the size of the weights. This process of finding a minimum is called gradient descent. Appropriate step sizes need to be taken to ensure that the classifier is able to find a global minimum and not a local minimum. Extensive training and the right epoch sizes can ensure that.

As each error is calculated, and each error's gradient is calculated, this is then sent back to the start of the layers. This backpropagation hence allows the weights to be updated with an appropriate value to ensure the performance of the classifier can be increased. As each weight is updated, it is important to ensure that each change to the weight is minute, as a big change in the weights can cause the behaviour of the neural network to be chaotic [21]. Smaller changes can help in identifying the correct value faster as the trajectory of the neural network can be easily projected and localised. Especially since these neural networks usually have lots of non-linearity, and the current gradient that the network would have is localised at the certain point it is being calculated on, bigger changes to the updated weights would make it very difficult to observe and analyse the results of those updates. The methods using which these weights are updated are called **optimizers**. A commonly used optimizer is the Adam optimizer – which is a variant of stochastic gradient descent.

Adam is an adaptive learning rate optimization algorithm that's been designed specifically for training deep neural networks. The algorithm works particularly well because it finds individual learning rates for each parameter through the power of adaptive learning rate methods. Adam can be considered a combination of RMSprop and Stochastic Gradient Descent with momentum [22]. The central idea of RMSprop is keep the moving average of the squared gradients for each weight. And then we divide the gradient by square root the mean square [23]. In Stochastic Gradient Descent, using random probability, the algorithm tries to minimize a certain cost function by moving in the direction of the steepest descent as defined by the negative of the gradient.

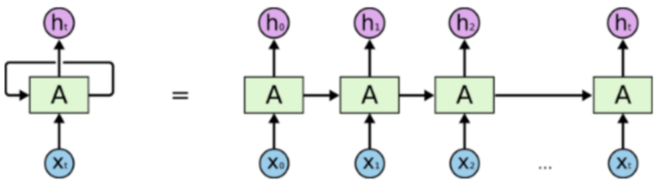
The whole process of forward and backward propagation is then repeated a number of times to get to a point where the neural network is adequately trained on the training data to be able to make predictions on similar information. It takes several iterations for the neural network to reach the right parameters to learn [21]. Each iteration is called an epoch. The number of epochs depends on a number of factors: quality of data, learning rate of the algorithm, complexity of the layers, optimization method used, and even the random initialization of the weights.

3.2. Long Short-Term Memory Recurrent Neural Network

LSTM RNN are a set of Neural Networks whose functions and way of working is inspired by the human brain. A Recurrent Neural Network differentiates from a normal Artificial Neural Network in the way that it specifically makes use of sequential information. If two images of a cat and dog are fed into an artificial neural network, the ANN will recognize a few patterns and predict an image to be of a dog or a cat. At no point did the ANN's prediction depend on previous images. Each prediction of a dog or a cat would be based on the image itself and not the data before it. However, there can be multiple situations where the previous data would be important in terms of making a prediction on the current data. For example, predicting the financial market value of a company's stocks, predicting the next word in a sentence etc. In such situations, a neural network – like the recurrent neural network – is needed where it can have 'memory' to store sequential information and utilize that when making a certain prediction.

In traditional neural networks, it is generally assumed that the input and outputs are independent of each other [24]. But this can become detrimental when, for example, predicting a word in a sentence as mentioned before. The word 'recurrent' is used for this neural network because it performs the same task on every element of a sequence of data, with each output being dependent on the previous data [24]. As can be seen from the following photo where x_t represents the input, A represents the activation function of the hidden layer, and h_t represents the value of the current state.

Figure 2. How an RNN utilizes its 'memory' [24]



There are many different types of RNNs: one to one, one to many, many to one, many to many and bidirectional many to many. Many to one is the kind of RNN where it takes a sequence of multiple data inputs and provides one output of a fixed size. The following photo describes the architecture of a many to one RNN where red arrows represent the inputs and the green and blue arrows represents the outputs of the hidden layers:

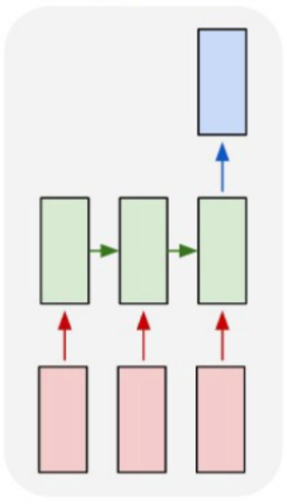


Figure 3. Many to one RNN [24]

The weights and biases applied to the hidden layers of an RNN are always the same in order for it to memorize the information processed through them [24].

In Figure 4 below, the formula to find the current state would be the following equation:

$$h_t = f(h_{t-1}, x_t) \tag{3}$$

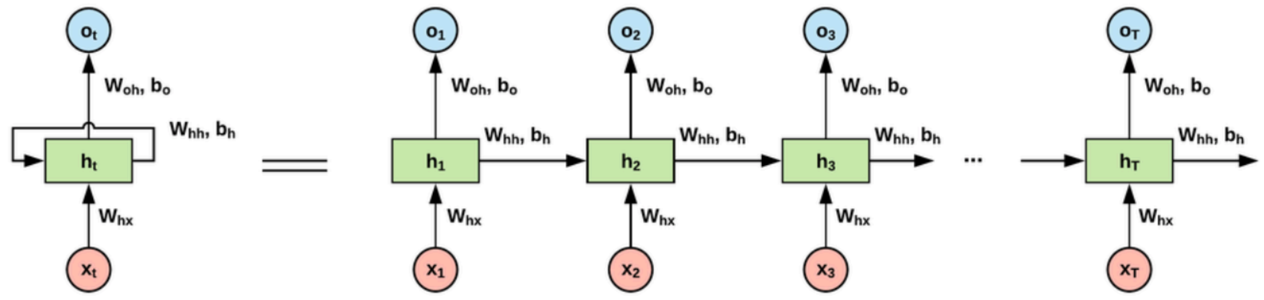


Figure 4. Information retention of an RNN [24]

O_t represents the output state, h_t would be the current time stamp, h_{t-1} would the previous time stamp and x_t is passed as the input to the next stage. The tanh activation function is applied within each candidate cell to determine the internal state value and that is then used to update the hidden state's value. This is described in the following equation:

$$h_t = \tanh\left(W_{hh}h_{t-1} + W_{xh}x_t\right) \tag{4}$$

In equation 4 W, is the weight, with W_{hh} representing the weights from the previous hidden layer, and W_{xh} being the weights at the current input state [24]. The tanh activation function is used because it implements a non-linearity where the output values are within the range -1 to 1. Its non-linear nature allows for multiple LSTM layers to be stacked on top of each other. The output at the final stage is described best through the following equation:

$$y_t = W_{hy}h_t \tag{5}$$

In equation (5) y_t represents the output state and W_{hy} represents the weight at the output state.

The LSTM RNN utilizes backpropagation to improve the value of the weights and biases assigned at each layer during the training process to reach a more accurate output state. The loss function helps to calculate the difference between the current output state predicted and the real output state. Mean squared error is a common loss function used. As mentioned in section 3.1, the gradient of the loss function is used to optimize the values of the internal weights to try and reduce the value of the loss function down to zero. As the weights are the same for all hidden layers in a recurrent neural network, the gradient calculated for each time step can be combined together. The weights of the recurrent neuron and the output dense layer can then be updated together.

There are two kinds of problems one can face during backpropagation: vanishing gradient and exploding gradient [24]. In simple words, **vanishing gradient** is a problem that occurs when the contribution from earlier steps becomes insignificant in the gradient descent step. This can happen when the error between the predicted output and the real output has a partial derivative with respect to the weight that is less than 1. This value is then multiplied with the learning rate – which is already a very insignificant value – causing this final value to be even smaller. Then as the weights are updated using this value, there won't be any significant changes in the weights and thus the vanishing gradient problem is faced. It causes the RNN to forget the long-term dependencies as those value isn't fed to the next iterations. Similarly, exploding gradient is the problem where if the partial derivative of the error with respect to the weights is extremely high, a stupidly high value is assigned to the weights and thus the gradient explodes.

These problems are solved by adding LSTM to RNNs: Long Short-Term Memory. Within the LSTM cell states, long term dependencies and relations are encoded in state vectors and it is this cell state derivative that can prevent the LSTM gradients from vanishing [25]. LSTMs consist of three steps: Forget gate, Input gate, Output gate. This is shown in the image below:

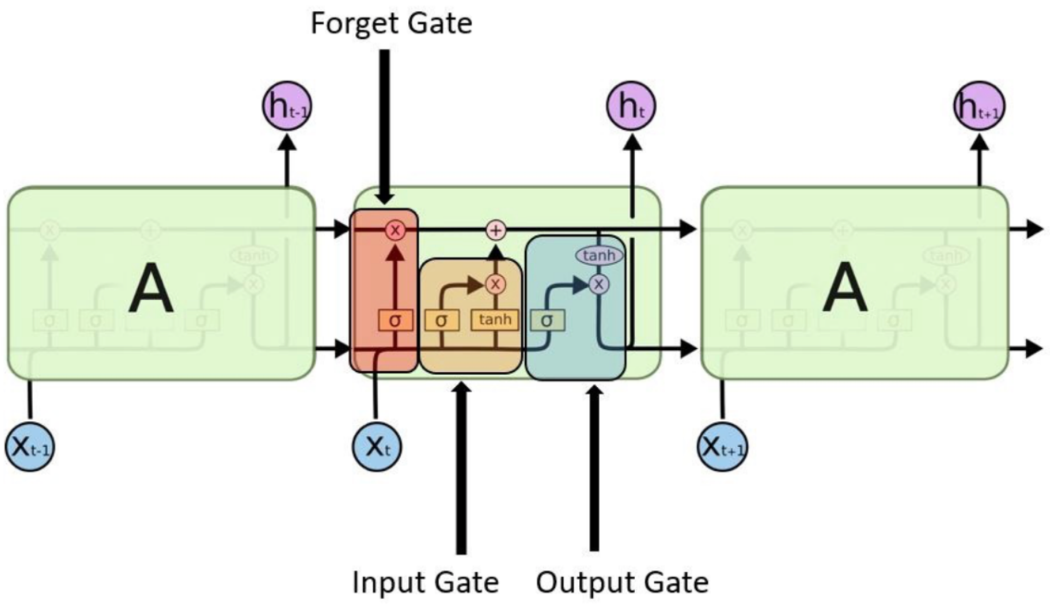


Figure 5. Structure of a LSTM cell

- **Forget Gate:** Essentially, this gate lets the LSTM cell know how much it is supposed to remember and how much it is supposed to forget from the information fed to it [24]. At every time stamp this gate figures out which information is to be left out from the cell. This is done using a **sigmoid function**. It looks at the previous data input and the previous state, and then outputs a value between 0 and 1. 0 meaning remove this information and 1 meaning keep this information. The forget gate's output equation is given as following:

$$f_t = \sigma[W_f.(h_{t-1}, x_t)][25] \quad (6)$$

- **Input Gate:** The input gate is also called the update gate. It decides how much of this unit is then going to be added to the current state. The sigmoid function decides which values to let through by outputting 0 or 1 [24]. The tanh function assigns a weightage to these values on a scale of -1 to 1 based on their level of importance. The input gate's formula is reflected in the following equation:

$$\tanh\,(W_c.\,[h_{t-1}, x_t])\otimes\sigma(W_i.\,h_{t-1}, x_t) \text{ [25]} \qquad (7)$$

- **Output Gate:** This gate is responsible for deciding which part of the current cell is going to make it to the output. The sigmoid function is used to decide which values are going to go through, tanh gives the weightage to these values on a scale of -1 to 1 and then the value from the sigmoid and tanh function are multiplied together. The formula for the output of the output gate can be represented in the following equation:

$$o_t = \sigma(W_o.\,[h_{t-1}, x_t]) \text{ [25]} \qquad (8)$$

The output from the LSTM cell can then be represented by the following equation:

$$h_t = o_t \otimes \tanh\,(c_t) \text{ [25]} \qquad (9)$$

As mentioned previously, the error term within RNNs is a sum of all the gradients calculated within the layers as all weights and biases are the same for all hidden layers. If the derivative of the error with respect to the weight is represented as a sum of sub-gradients through the following equation:

$$\frac{\partial E}{\partial w} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial w} \qquad (10)$$

For the vanishing gradient problem to occur, the sum of these sub-gradients would need to go to zero [25]. To prevent this, the simplest way is to prevent some of these sub-gradients from converging to a value of 0. They must converge to a non-zero value. When an LSTM cell is added to an RNN, the LSTM cell's state vector is added to the error term. If the partial derivatives of the gradient term mentioned in equation (10) is then taken, the partial derivative of the LSTM cell's state vector contains the forget gate's vector of activations which allows the network to better control the gradient value at each iteration using suitable parameter updates of the forget gate [25]. This presence of the forget gate enables the function to decide at each time step which information should not be left out and it can then update the model's parameters accordingly.

For example, if at a certain time t, the gradient of the error, $\frac{\partial E}{\partial w}$, goes to 0, the forget gate's vector of activations in the gradient term along with its additive structure allows the LSTM to find such a parameter update that at time t+1, the gradient of the error, $\frac{\partial E}{\partial w}$, should not be equal to zero. Hence the gradient does not vanish.

In addition to the forget gate's activation function, the shape and form it is present inside the gradient of the error is important as well. After the LSTM cell's state vector is differentiated, it consists of four different additive terms. As these four different terms are now present inside the gradient of the error, at each iteration when these four terms are updated, it is not likely that all four of them will converge to zero. In contrast to a normal RNN's error gradient function which only consists of one term, with LSTM cells added to the RNN, the additive terms make it more unlikely that the function will converge to zero.

In this way, due to the presence of the forget gate's activation function and the additive property of the function, the vanishing gradient problem is avoided. The exploding gradient problem occurs in the same way as the vanishing gradient. When the sub-gradient terms are combined, if their values are higher than one, the exploding gradient problem occurs. However, the LSTM cell solves this problem in the same manner as it solves the vanishing gradient problem.

There are a few more features that are necessary to implement the neural network:

- **Dropout:** At every layer of LSTM, dropout is added to each layer. In simple terms, dropout is leaving out a few units with a set of units during the training phase [26]. Leaving out a few units or neurons means these particular units or neurons are not looked at during forward or backward propagation. These nodes are dropped out with the probability 1-p or kept with the probability p in order for the overall network to be reduced [26]. The incoming and outgoing edges to a dropped-out node are also removed [26].
  Dropout is necessary in order to prevent over-fitting of the network. Overfitting occurs if a model trains 'too well' on the training data. As most of the parameters are occupied by the fully connected layer, the neurons within it develop a co-dependency amongst each other during the training phase [26]. This hampers the individual power of each neuron leading to over-fitting of the training data. Overfitting is detrimental to the model as it learns the pattern and the noise in the training data to such an extent that performance of the model on a new dataset would be negatively impacted.

During the training phase, on every iteration, at each hidden layer a random fraction, p, of nodes and their activation functions will be ignored. In the testing phase, all activations will be used but will be reduced a factor of p to account for the missing activations during the training. The way the inter-dependencies of the neurons are removed can be seen in the following figure:

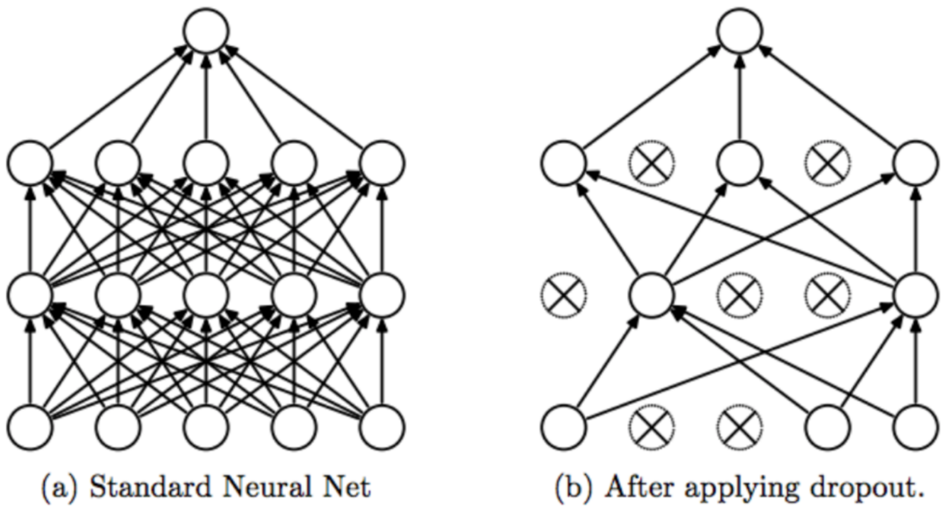


Figure 6. Reduction of inter-dependencies of neurons before and after dropout [27]

The advantages of using dropout is that as mentioned previously, it helps in avoiding overfitting, it enables the neural network to learn more robust features from the training data which might be applicable to different random datasets as well, it also reduces the overall time required to run one epoch.

- **Batch Normalization:**

  Training LSTMs can sometimes become a tedious task. At every iteration the parameters of each layer will change, causing the distribution of the inputs of the other layers to change. This causes the training process to slow down. Batch normalization improves the stability of the network by reducing the covariance shift of the hidden unit values [28]. The output of the previous activation layer is normalized by subtracting the batch mean from it, and then by dividing it by the batch standard deviation [28].

  Although, after this shift takes place, the weights in the next layer will not be optimal [28]. If the loss function in that stage would be minimized by removing this normalization, stochastic gradient descent in that stage will do exactly that. Hence in such a situation the batch normalization may turn out to have been useless.

  To counter this, batch normalization adds two trainable parameters to each layer. Since batch normalization multiplies the output of a layer by the standard deviation and mean, these two are added as two trainable parameters – gamma and beta, respectively. This ensures that SGD is performed on these two parameters, allowing for the denormalization to only take place on these two parameters [28]. This way the stability of the network is not lost by changing all weights.

- **Dense Layer:**
  The dense layer is where the neural network has a neuron for every output expected. So, if there are 5 outputs, there will be 5 neurons in the dense layer. Each neuron will be connected to all the neurons in the layers previously, hence this densely populated connection layer is called a dense layer. This layer has its own weight matrix W, a bias vector b, and the activation of the previous layers.

3.3. Resampling

In order to remove the noise that ordinary analogue data carries due to the relative error of the machine that is recording that set of analogue data, sometimes resampling techniques are applied. Resampling is a process where a discrete signal is converted to a similar representation of the signal itself. Resampling can be needed when, for example, the data provided was measured at a different frequency than what the neural network wants to make predictions on. There are two methods of going about resampling: upsampling and downsampling.

Downsampling is done by the recalculation of the values of a new signal at a lower frequency than the original high frequency signal. The general idea would be to remove samples from within the data whilst maintaining its data integrity. In order to prevent high frequency signals from subjecting the new signal with aliasing distortion, it is important to pass the signal through a low-pass filter as well to remove the appropriate high frequency content [29].

In upsampling, samples are added to the original signal and its length with respect to time is maintained [29]. When increasing the number of samples, there is no issue of aliasing as no extra content has been added – only the sample points have been increased. Upsampling is usually done through interpolation where a value between two known data points is estimated.

3.4. Layer-wise Relevance Propagation

Generally, neural networks are applied to problems using a blackbox method where an input of any sort, for example an image, goes into the blackbox which is the neural network, and then a prediction comes out of that said blackbox [30]. This approach has produced good results for quite a while now, but when expanding the application of these models onto real world problems it is extremely important to understand how and why those decisions are made by the neural network.

This can be imperative for many situations where the stakes are unusually high, and an incorrect decision can cause serious damage to people. For example, deep learning models are nowadays applied on medical scans to predict if a patient has a certain disease or not. There can be many situations during the training process where the accuracy of the neural network's prediction might be very reliable, but it might be picking up a unique thing within the data it was trained on, and so may miss out when that certain type of data is not present within a test scan of a patient who in fact may have the disease. This proves that interpretability of a neural network's prediction, understand why and how it has made that decision, is extremely important [30]. This can help establish trust and validation on deep learning models which can then boost their use in medical diagnosis or autonomous cars etc.

A recent approach to understand the decision making of a neural network has been and is called LRP – Layer-wise Relevance Propagation. It is focused on providing a pixel-wise explanation for non-linear classifier decision [30]. This technique allows for the visualisation of the decision making of each individual pixel of an image when it comes to making a certain decision as heat maps [31]. This enables a researcher to verify that prediction, validate the decision-making process by understanding how that prediction was made using appropriate data, and to observe if there was any new information identified that may be of further interest.

As neural networks are multi-dimensional models where multiple hidden layers are able to have many significant interactions with all input features [32]. As a single input feature can have multiple paths to influence an output, an approach towards the explainability of the decision making of neural networks is needed where that is taken into account. In LRP, the predicted probability of a specific target is deconstructed into a set of relevant scores which are then redistributed onto the neurons of the previous layer [32]. The strength of a neuron's connection – judged by its weight – and the activation of a neuron make up the said relevant score [32]. This relevant score then shows how much an individual neuron was contributing towards the end result at the output neuron based on a certain input feature. And as this technique is applied repeatedly over all layers – output layer to input layer – a relevance score for each neuron is generated.

3.5. Method and Evaluation of Neural Networks

The method for implementing the classification of these activities and the evaluation of that classification is done after the dataset is split into the training set and the testing set. The splitting of this dataset is done in such a way that 80% of the data is used for training of the classifier while 20% of it is used for testing the classification. In order to avoid the overfitting of the model, there will be a separate validation set as well to test the classifier on to understand its general capabilities as well.

There will be two sets of major indicators that will be used to test the ability of the classifier and to evaluate its results:

- The **accuracy** – which gives the result of the correct predictions as a percentage of the total predictions. The formula is defined as:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{3}$$

  Where TP represents true positive – number of positive records that were correctly predicted as positive, TN represents true negative – number of negative records that were correctly predicted as negative, FP represents false positive – number of negative records that were incorrectly predicted as positive, and FN represents false negative – number of positive records that were incorrectly predicted as negative.
- The **confusion matrix** – it is a technique which helps in summarizing the results and the performance of a machine learning classifier. It uses the same 4 values that the accuracy function uses – TP, TN, FP, and FN. A simple 2 x 2 interpretation of a confusion matrix is described in the table below:

		Actual values	
		P	N
Predicted values	P’	True Positive	False Negative
	N’	False Positive	True Negative

Figure 7. A 2 x 2 sample Confusion Matrix

In a larger confusion matrix, the best results are when the left top to right bottom diagonal are the boxes with the most values as they signify the correct predictions.

4. Application of Neural Networks on Labelled Dataset

4.1. Method and Evaluation of CNN

The following diagram describes the architecture and parameters used in the Convolutional Neural Network created for this project:

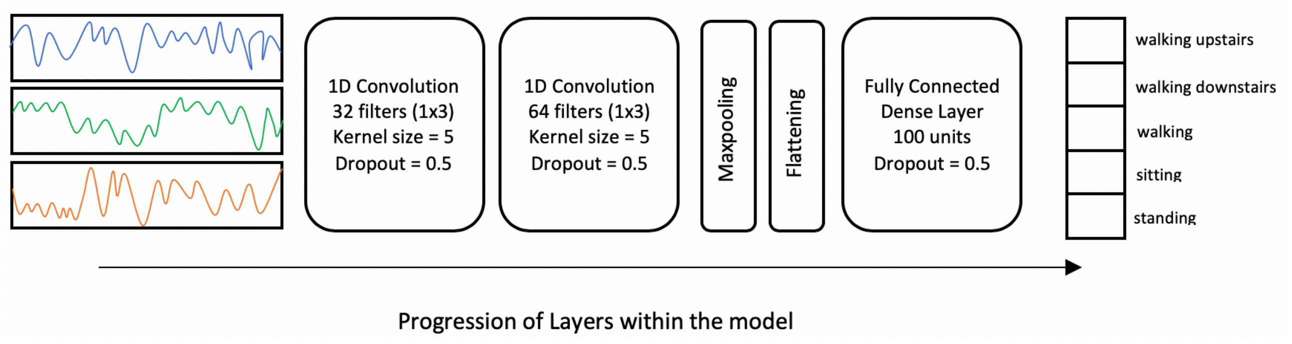


Figure 8. CNN model architecture

The parameters of each layer from figure 8 are described below:

- **Input:** As acceleration from the smart device it was measured from is recorded as time series data, we need to prepare and provide it to the CNN as a 2-dimensional input. The first dimension being time and the second being the acceleration's value. The following diagram shows how this is achieved:

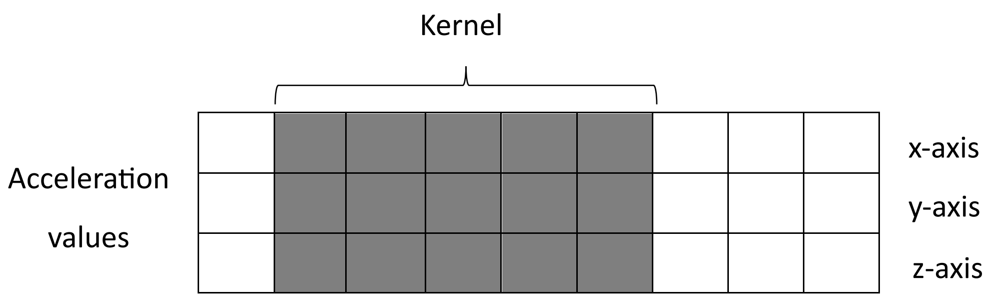


Figure 9. Kernel sliding over input data

- **Convolution Layers:** The acceleration values pre-processed into a 2D structure is then fed into the two convolutional layers. The first convolutional layer has ___ convolutional filters, has a kernel size of 5 and uses the relu activation function. Similarly, the second convolutional layer also has a kernel size of 5, also uses the relu activation function, however, has a convolutional filter of ___. Stride, which refers to how many pixels the kernel shifts by, is 1. A dropout rate of 0.1 is used with the first convolution layer and a dropout rate of 0.2 is used with the second convolution layer.

- **Maxpooling layer:** The pooling layer has a size of 2. This reduces the number of feature maps by ___.
- **Fully connected dense layer:** The output from the Maxpooling layer is supposed to be flattened. It is then fed into a densely connected layer of 100 neurons. A dropout rate of 0.5 is used with this layer to avoid overfitting. Another dense layer representing the output layer is added with only 5 neurons used – representing the 5 output activities the neural network is supposed to classify. At this stage a softmax activation function is used to compute the likelihood of which output activity the data at the current iteration most likely represents.

The programming code for this project was written in python using an IDE called Spyder. An open source library called Keras was used to implement the architecture shown above [33]. A sequential model was used for the implementation as it allows for a model to be created and then layers to be added to it later on. It greatly simplifies creating deep learning models. 1D convolution layers, 1D maxpooling layer, flattening layer, dropout and dense layers were added to the sequential model. The parameters of each of these layers have already been explained previously.

The model was implemented on a 2.3 GHz Dual-Core Intel Core i5 CPU with 8 GB 2133 MHz LPDDR3 memory. The model was trained for 100 epochs with a batch size of 32. The Adam optimizer was used to compile and optimize the model by calculating the gradients of the parameters at each layer, calculating the error between the correct value and predicted value using the sparse categorical cross-entropy loss function, and then finally updating the weights using the default learning rate of 0.001. The sparse categorical cross-entropy function is a normal loss function that calculates the error between the prediction and the correct value but uses the labels instead of a one-hot encoded scheme. The learning rate is a hyperparameter which decides how much to change the weights of the model by at each iteration. The larger it is the bigger the changes.

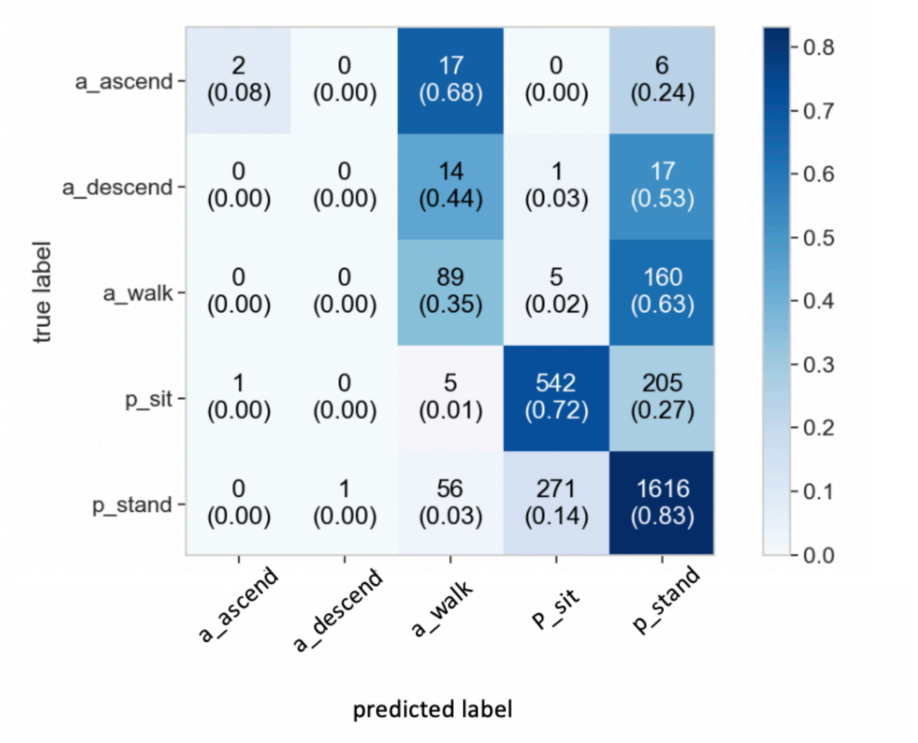
The model was trained on the dataset provided by the SPHERE competition. As mentioned in section 2, it contained data for 10 people. This dataset was split into two parts: 8 people's data was used to train the model and 2 people's data was used to test the model. In this way the model does not get to see the data of the testing set and therefore when that dataset is used for testing, the model will give results that reflect the real-world scenario.

Before the accelerometer values and the activity labels are fed into the CNN, the training dataset is resampled at the same frequency its data was collected at: 20 Hz. The data collected in the real world will have some small errors when it comes to what frequency it is collected at. When the neural network is being trained, it is important that there is no discrepancy in the data it is being trained from. The accelerometer's values were resampled using the interpolation function present in the SciPy library [34]. The interpolation function allows for then any missing values to be filled as well using an extrapolation feature.

As described in section 2.2, the 22 different activities within the SPHERE dataset were combined into 5 different activities. The accelerometer values and the corresponding activity labels were then created into segments and labels. The segments contained the accelerometer values for the x, y and z axis, while the labels contained the corresponding physical activities. To generate these features, a time step of 200 was used and whichever activity was the most common in each time step, the segment was assigned that activity as it's label. A time step of 200 was chosen as it corresponds to 10 seconds of performing an activity. Any physical activity is always performed for a period of time and not for only an instance. Therefore 10 seconds is considered an appropriate length of time for a label to be added to that segment.

The CNN model gets an **accuracy** of up to 75% on the testing dataset. The overall performance of the CNN can be seen in the confusion matrix in figure 10. The confusion matrix shows both, the absolute value and the normalised value of the predictions. Even though the classifier's accuracy is about 75%, the overall performance cannot be considered satisfactory. There is a large discrepancy between the results of each individual activity. The maximum accuracy achieved is for standing activity with 83%. Whereas for walking upstairs and downstairs the classified performed extremely poorly.

There can be many reasons for these misclassifications. One of the important factors is that within our original dataset and our testing dataset, activities like standing and sitting, both of which were classified to a higher accuracy compared to the rest, were more densely present compared to activities like walking, walking upstairs and walking downstairs. In the testing set, there were 1944 samples of standing whereas only 25 samples of walking upstairs.



	a_ascend	a_descend	a_walk	p_sit	p_stand
a_ascend	2 (0.08)	0 (0.00)	17 (0.68)	0 (0.00)	6 (0.24)
a_descend	0 (0.00)	0 (0.00)	14 (0.44)	1 (0.03)	17 (0.53)
a_walk	0 (0.00)	0 (0.00)	89 (0.35)	5 (0.02)	160 (0.63)
p_sit	1 (0.00)	0 (0.00)	5 (0.01)	542 (0.72)	205 (0.27)
p_stand	0 (0.00)	1 (0.00)	56 (0.03)	271 (0.14)	1616 (0.83)

Figure 10. Confusion Matrix of classifier's predictions on SPHERE Testing Set

The interesting feature of these results is that stationary activities like standing and sitting have a much higher accuracy than mobile activities like walking etc. This could be as a result of the fact that when a person is sitting or standing, they perform less movement of their wrists – where the accelerometer was placed. Whereas, when the said person would be moving, their wrist position can change quite a lot, and this can cause the accelerometer values to change drastically – thus not allowing for a pattern to be recognisable. Walking upstairs or downstairs are either predicted as walking or standing – both of which are positions where the wrist positions of people might be similar to not only each other but to walking upstairs or downstairs as well. The positive result over here is that it doesn't classify either of walking upstairs or downstairs as sitting – a position where the posture and wrist position must be very different compared to the rest. Therefore, the classifier can identify a pattern with stationary activities but is unable to do so with mobile activities.

Moving from the testing set to the validation set, the classifier's consistency in performance is key. On the WISDM dataset, the classifier gets an accuracy of 21.5%. From figure 11, we can see that he CNN is only able to track the sitting activity really well – with an accuracy of 98%. Just like it did for the testing set, it identifies walking, walking upstairs and walking downstairs as 'standing' as it perhaps is unable to identify the differences between them. The negative feature over here is that it identifies the standing activity as sitting, which didn't happen with the testing set.

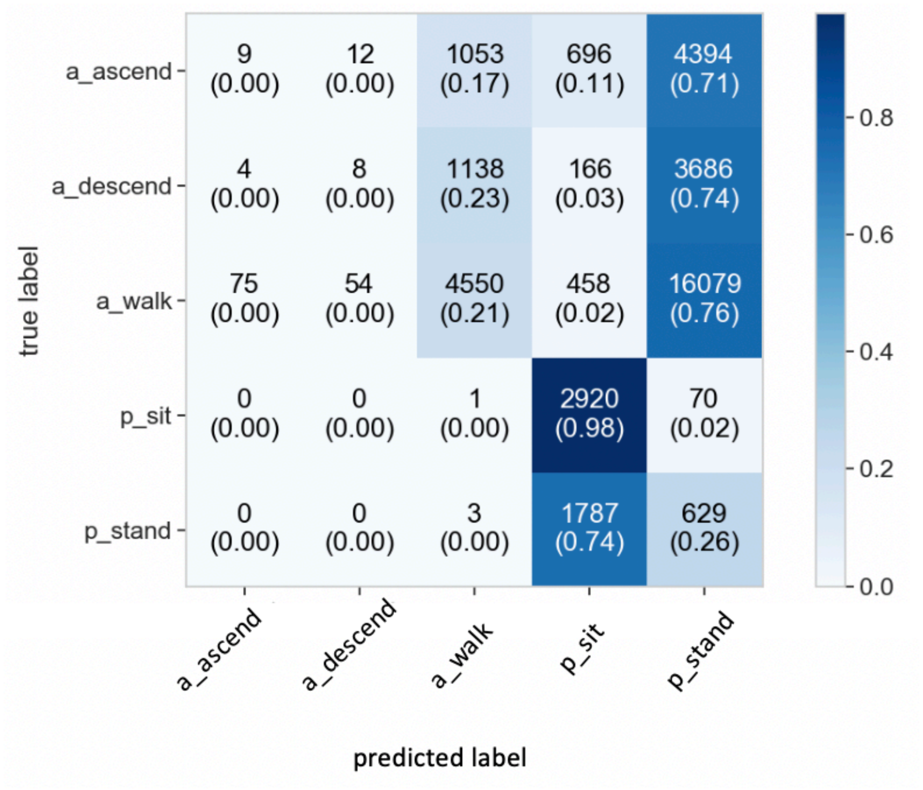


Figure 11. Confusion Matrix of classifier's prediction on validation set: WISDM dataset

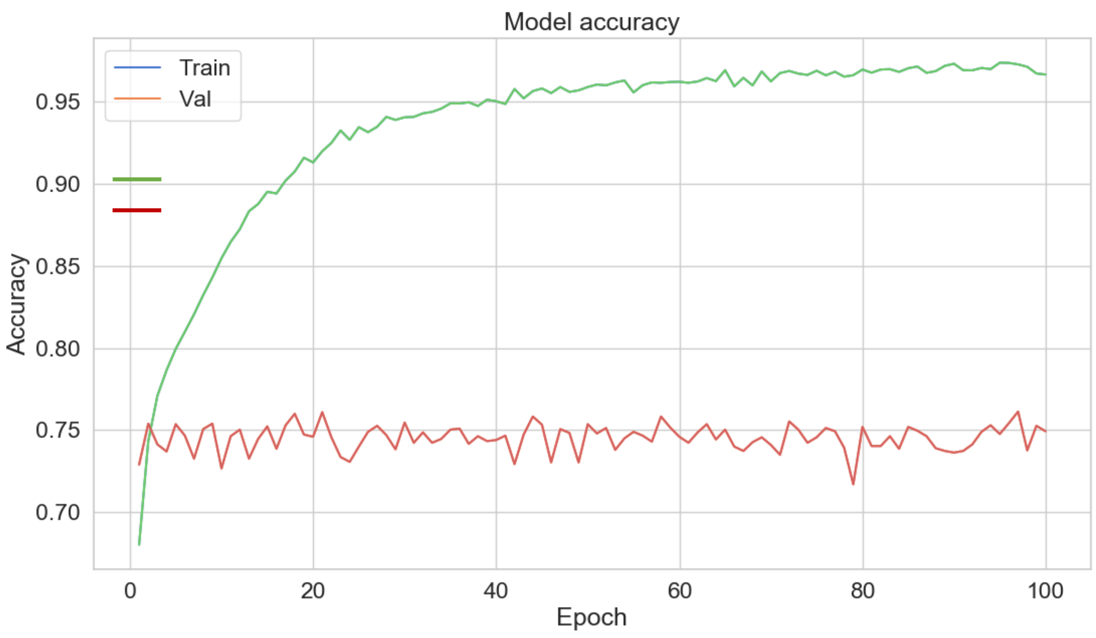


Figure 12. Graph the accuracy of the models against the number of epochs

Figure 12 show how well the model learned on the test data as the number of epochs were increased. The model shows that it learns well over time as the accuracy on the training set increases from 70% to 97% over 100 epochs, although there is not much improvement after 20 epochs. Moreover, the accuracy on the testing set does not improve in this range of epochs. The model seems to be consistent in its accuracy on the testing set as it remains approximately at 75%.

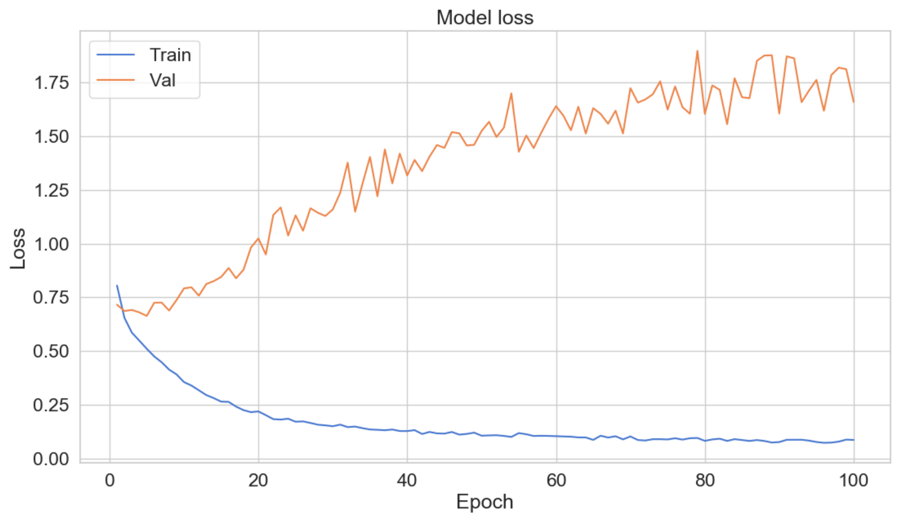


Figure 13. Graph of the loss function of the CNN against number of epochs

Figure 13 shows how the cost function of the model's loss value changes over time. The loss for the training set reduces over time whereas the loss for the testing set increases. This points out to the fact that the model may be being overfitted on the data. A solution to this problem is to increase the value of dropout and to reduce the size of the network by reducing the number of layers. Therefore, the value of dropout was increased from 0.2 to 0.5 and the second convolutional layer was removed. The two solutions were performed individually in order for the effects to be observed. Increasing the dropout reduced the difference in the loss functions of the training and testing data. The overall performance on each activity was not majorly affected either and the model produced similar results. However, when the second convolutional layer was reduced, the overall accuracy of the program did not get affected, however its performance on individual activities was reduced as only the standing activity was predicted to a high accuracy. This points towards the model not being able to learn the patterns of each activity correctly. Hence only the dropout was increased to 0.5, the epochs were reduced from 100 to 20, but the number of convolutional layers were kept the same.

These changes did not drastically change the model's performance on the testing set – as it only improved the overall accuracy from 75% to 76%. However, the model's performance on the validation set improved from 21% to 30%. This improvement on the validation set was reflected in different activities – walking's accuracy improved from 21% to 32%, standing's accuracy improved from 26% to 39%, sitting's accuracy remained the same, while walking upstairs and downstairs had a few samples predicted correctly even though as a percentage it was less than 5% for both.

4.2. Method and Evaluation of LSTM RNN

The stacked architecture of the LSTM used for the multiple classification of Human Activity Recognition can be seen in the following schematic:

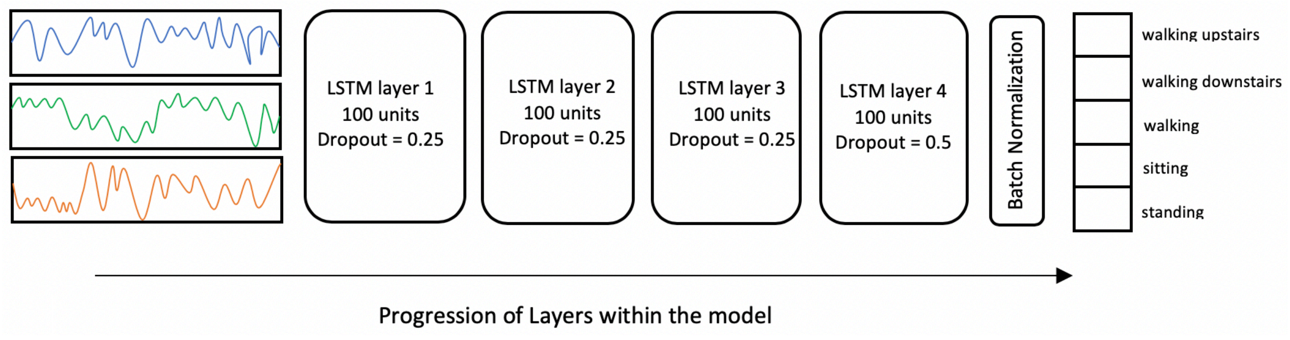


Figure 14. Stacked Architecture of LSTM RNN model used

The parameters used in the above architecture are explained below:

- **Input:** The accelerometer data recorded to train the model is in the form of time series data. It is important for the LSTM within the model that this data is prepared appropriately for it. The 3 axis accelerometer data needs to be reshaped into a parallel 3D structure – as is needed to efficiently build and train the LSTM network. 200 samples were fed into the network to form a segment or a group. The value chosen as the label for a group of sampled data is based on the mode – the most commonly occurring label – within said group. The previous 199 samples would act as memory for the LSTM network.
- **LSTM Layer:** Four LSTM network layers were added to the network. Each layer had 100 units or neurons added to them. Dropout of 25% was added to each individual LSTM layer to prevent overfitting.
- **Batch Normalization:** As mentioned in section 3.2, batch normalization helps in speeding up the training process hence batch normalization is added to the model.
- **Fully connected layer:** A dense layer is added to complete the model. 5 units are added to this layer to represent the 5 different classifications.

This architecture was implemented using Python and Keras – an open source library for building neural networks [33]. A sequential model was used with multiple LSTM, Dense, Dropout and Batch Normalisation layers. The input layer had 100 neurons, with 3 more LSTM layers stacked on to it in order to allow for the establishment of a pattern and dependencies between the input data and the corresponding activities. An output layer with 5 neurons for the 5 different classification was added at the end. The final model is then trained for 100 epochs using the Adam optimizer, with the default learning rate of 0.002 and the loss function of mean squared error. The model is trained offline on a CPU 2.3 GHz processing power, Dual-Core Intel Core i5 with 8GB memory. However, it would take more than 6 to 7 hours to train the LSTM RNN model as the MacBook's CPU was not powerful enough. Therefore, when testing and running the algorithm, Google Cloud Platform's virtual machines were also used to speed up that procedure. The virtual machine had 2 vCPUs with 13 Gb memory and also had 4 NVIDIA Tesla P100 GPUs which boasted its performance. The rest of the procedure in terms of pre-processing, testing, training and validating the classifier was exactly the same as the procedure used for the CNN model to maintain consistency across both models. 8 people's data from the SPHERE competition was used to train the algorithm, 2 people's data was used to test it and then finally the WISDM dataset was used to validate the model.

The LSTM RNN got an accuracy of between 65 and 70% when it was compiled multiple times. The results can be analysed for each individual activity using the confusion matrix from figure 15. The LSTM model has the same trend as the CNN model where it is able to classify stationary activities such as sitting and standing to a good extent but struggles with activities involving lots of motion like walking, walking upstairs and walking downstairs.

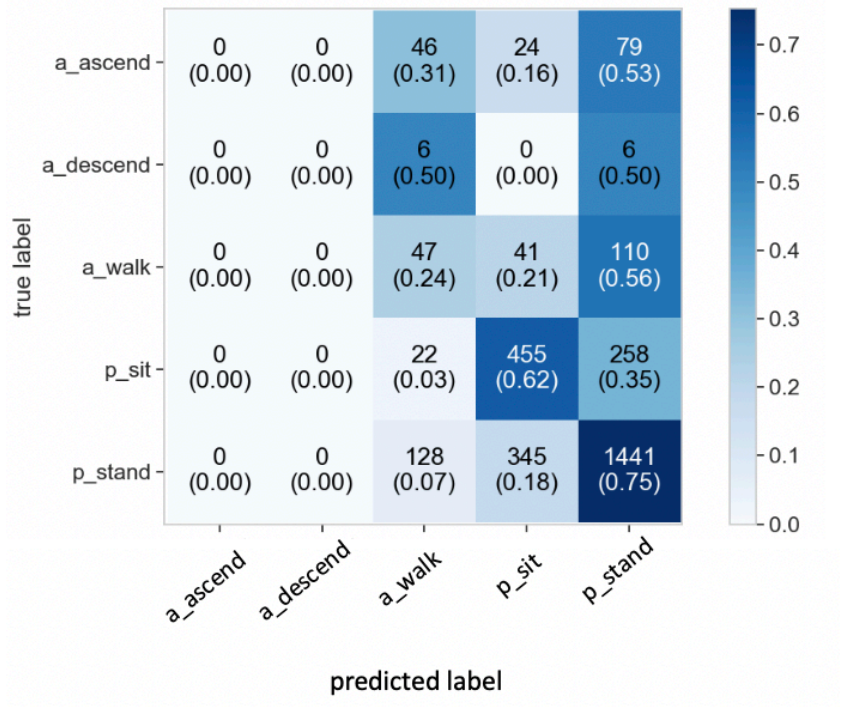


Figure 15. Confusion Matrix of LSTM model's results on the testing set

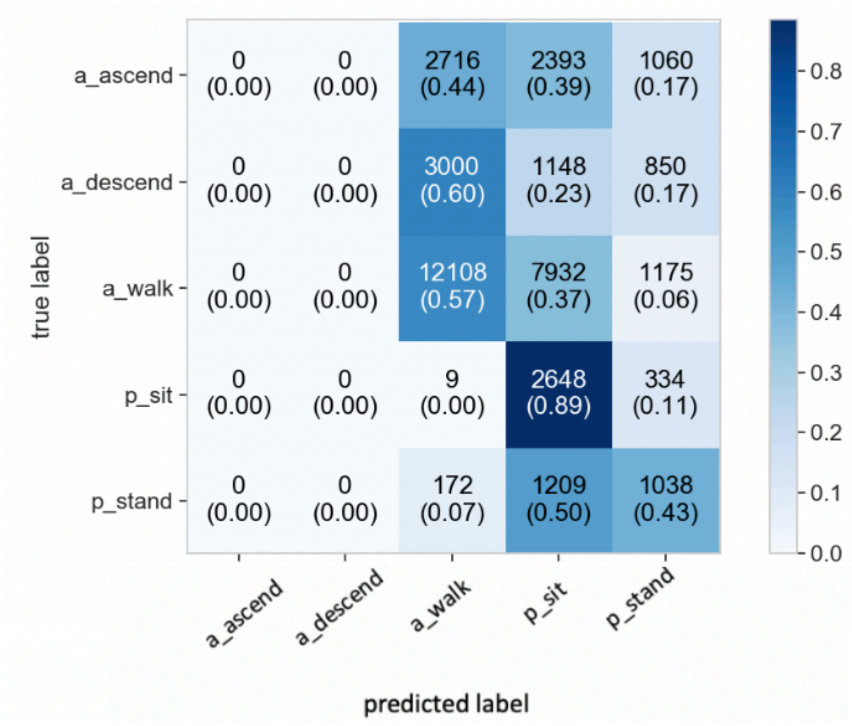


Figure 16. Confusion Matrix of LSTM model's predictions on the validation data.

Contrary to the performance of the CNN on the validation dataset, the LSTM model performed better as it achieved an accuracy between 42 to 50 % on multiple runs. It's performance on individual activities as well is not that different than its performance on them in the testing dataset. From figure 16 we see that the accuracy on walking improved from 24% on the testing set to 57% on the validation set, the accuracy on sitting improved from 62% to 89%, however the accuracy of standing went from 75% to 43%. The model was unable to identify walking upstairs and downstairs at all in both datasets – testing and validation. There can be many different reasons for the results obtained. Compared to the CNN model, the LSTM RNN model has a higher accuracy on the validation dataset. This confirms the theoretical assumption before these models were made that LSTM RNNs are better at classifying time series data the CNNs. However, this might also be the reason that the LSTM model is unable to classify activities like walking upstairs and downstairs. The LSTM model utilises its 'memory' of a certain number of previous samples when making a prediction on the current sample. Walking upstairs and downstairs are activities that are not abundantly present in the training dataset and do not last for a long time either compared to activities like sitting etc. This could be a major factor for why the model is unable to properly learn and classify it. For walking both models perform almost the same with the CNN getting a 98% accuracy and the LSTM RNN getting an 89% accuracy. But for walking and standing the LSTM RNN performs much better than the CNN, with the accuracy of walking going from 21% with the CNN to

57% with the LSTM RNN, and the accuracy of standing going from 26% with the CNN to 43% with the LSTM RNN.

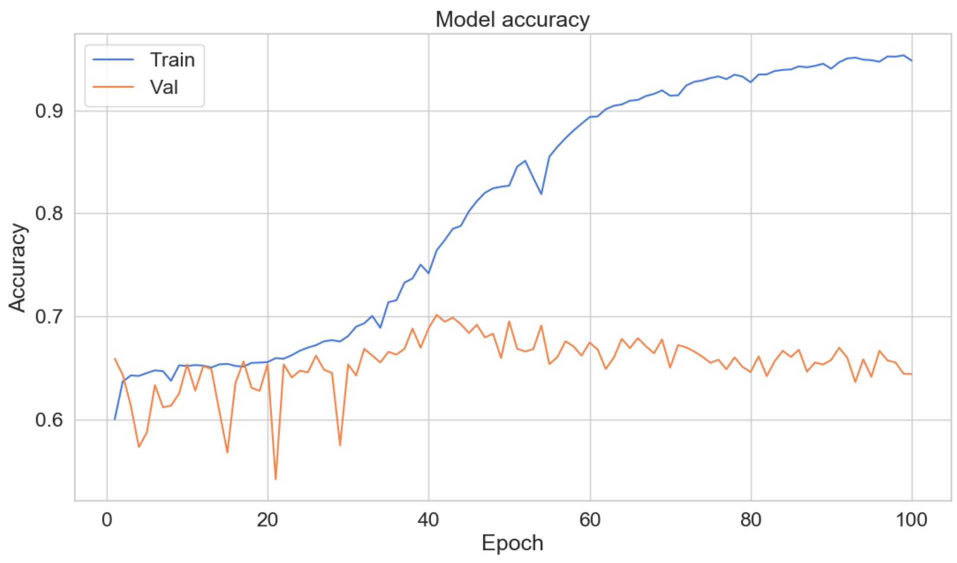


Figure 17. Accuracy of the LSTM model against the number of epochs.

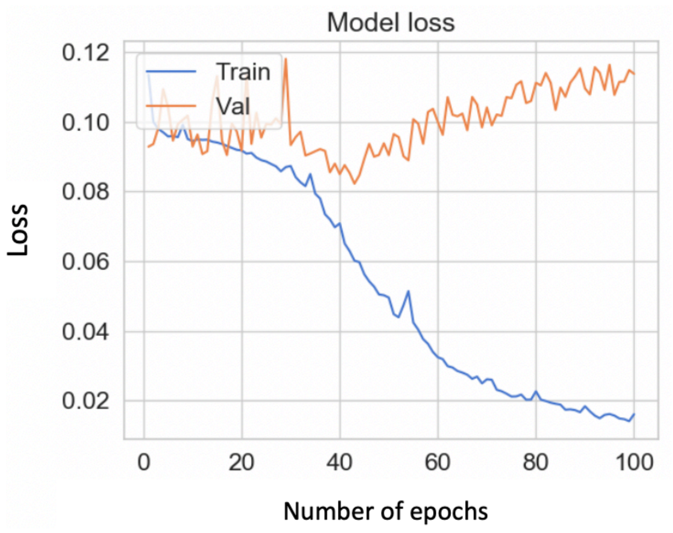


Figure 18. Graph of the loss function of the LSTM model the against number of epochs

Figure 17 shows how the accuracy of the model on the training and testing set changed over the range of epochs. The model's peak accuracy on the training set comes around 40-50 epochs where there is less disparity between the testing set accuracy and the training set accuracy. Therefore, to reduce the disparity between the loss functions and maintain a good accuracy, the number of epochs were reduced to 42 which improved the overall accuracy. Multiple runs must also be performed by increasing the dropout to observe the effects of that on the accuracy. However as one run of the model takes a long time to execute not only on the local computer used but also on a virtual machine on google cloud, time constraints inhibited the completion of this experimentation.

4.3. Analysis of Neural Networks using LRP

The LRP technique, as described in section 3.4, is used to analyse the decision making of a neural network. The innvestigate library was used to analyse the model [35]. Due to the limitation of LRP, it is only possible to apply it to the CNN model. LRP was applied to 2 neurons within the CNN for each outcome from the testing set to understand how well the model followed the value of the accelerometer from each axis. The test set is pretty large and the offline machine that the model is compiled on was deemed not powerful enough to run the analysis on the whole set. Therefore, due to time and CPU resources being an issue, the LRP technique was only applied to two neurons in two limited parts of the testing set: accurately predicted activity of standing (83%) and relatively inaccurately predicted activity of walking (35%). It was only applied to both activities for about 10 continuous samples.

The figure 18 below shows the LRP relevance for a correctly predicted sample of walking using the 200 different values of the accelerometer within the sample. It can be observed that the model does not identify much of a pattern using either x, y or z axis values as the relevance from each of them is extremely low throughout majority of the sample.

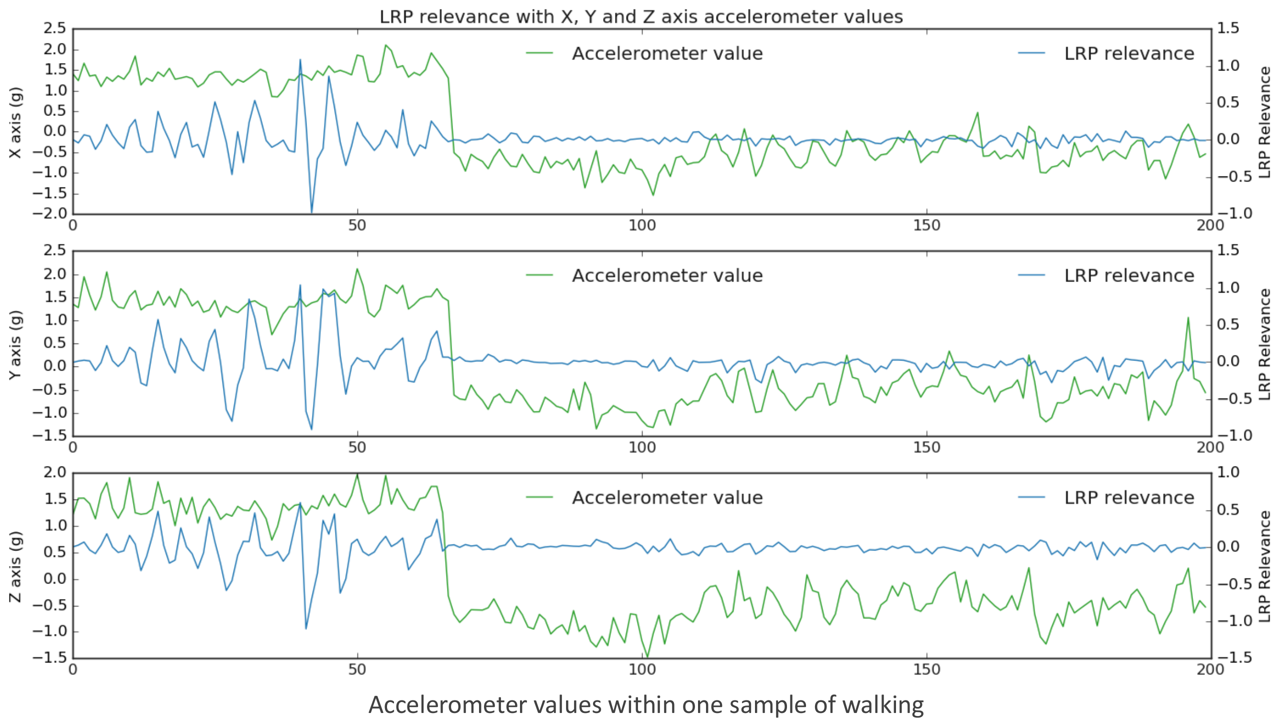


Figure 18. Graph showing the relevance between the accelerometer's values and the predicted outcome

This can now be compared with the figure 19 below which shows the relevance scores for the activity of standing. The LRP relevance scores for each value within the sample is much higher on average than the relevance scores for walking. This is not surprising as there is much less noise in the accelerometer data for standing compared to walking. The stability within standing's data allows the model to learn the patterns much better than on walking's data and hence the higher relevance scores for the LRP.

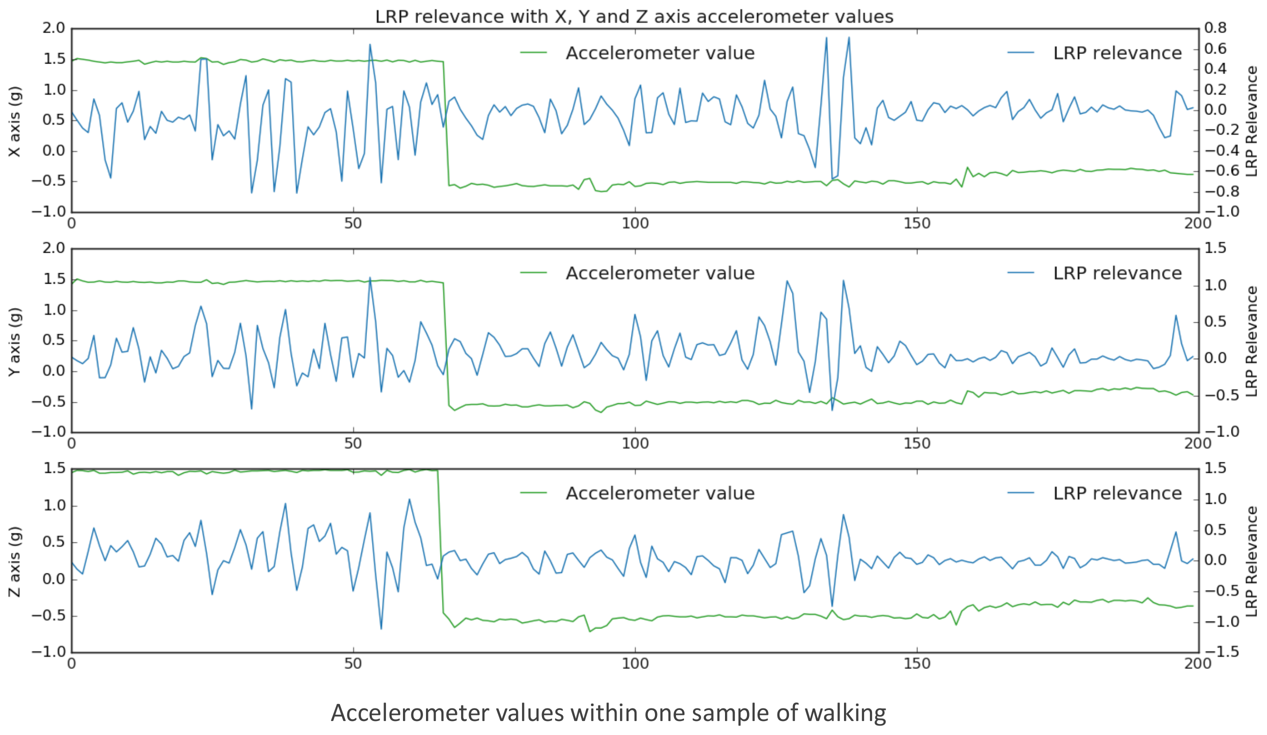


Figure 19. Graph showing the relevance between the accelerometer's values and the predicted outcome of standing.

A lot more analysis can be carried out using LRP. Due to reasons mention at the start of the report in appendix A, it was not possible to continue further analysis. These further steps will be mentioned in section 5.3 as future work that can be carried out.

5. Conclusion and Future Work

5.1. Conclusion

In this Final Year individual project two different classifiers, an LSTM RNN and a 1D CNN, were looked at to see how each of them perform when it comes to Human Activity Recognition using accelerometer data. Both networks were trained on the SPHERE dataset, tested on the SPHERE dataset and then validated using WISDM's dataset. The activities they were trained and tested on were both stationary (sitting and standing) and mobile (walking, walking upstairs, walking downstairs). These activities were chosen as they last for a few seconds and then therefore a correlation between the activity and the accelerometer value may be established by the Neural Networks.

From the results the preconceived notion that Long Short-Term Memory Cell Recurrent Neural Networks perform better on time series data than 1D Convolutional Neural Network got proven further. The results can be summarized below:

- In general, both networks worked to a satisfactory accuracy – up to 70% for the LSTM RNN and 75% for the CNN – when tested on dataset that it was trained on.
- However, the accuracies and the resultant predictions went down when tested on dataset that the neural networks had not been trained on. The LSTM RNN got an accuracy of 50% on the separate dataset whereas the CNN got an accuracy of up to 21% on it. Therefore, the generalisation capability of both networks is weak.

5.2. Limitation

- The classifier is trained on SPHERE's dataset which is sampled at 20 Hz. The WISDM dataset is also sampled at 20 Hz so this is not a problem here, but if other datasets are used for testing and those datasets have values sampled at a higher rate, they will have to be down sampled which will result in a loss of information. This is one of the limitations of this project in terms of future work.
- Another limitation of this work is that the trained and test dataset were both different in terms of the activities they had present in them, with the SPHERE dataset initially having 22 different classification and the WISDM dataset having 6 different classifications.
- Both neural networks need to be fully analysed using techniques like LRP – Layer-wise relevance propagation – to fully understand how or why they are making decisions the way they are. However, as this is very new research in terms of LSTM RNNs, it couldn't be implemented on it. This method will be applied to the CNN and if there are further steps needed and isn't completed, those will be mentioned as steps that were not possible to be implemented considering the time limited schedule of the Individual Project.

5.3. Future Work

- In the future, the datasets that must be chosen for training and testing of the classifiers must be chosen wisely. It is better for training and testing datasets to be as similar as possible in order to understand if the classifier is built and trained properly. However, this will limit the generalisation capability of the classifier.
- The dataset chosen for training should have each activity present in relatively equal proportions. This might not be possible as human beings perform each activity on a different proportion daily, however in terms of training the classifier this has an adverse effect as classifiers are unable to identify the activities that are rarely performed due to lack of training data.

6. References

[1] J. A. Knight, "Physical Inactivity: Associated Diseases and Disorders," 2012. [Online]. Available: http://www.annclinlabsci.org/content/42/3/320.full. [Accessed 2nd October 2019].

[2] M. A. Harris, "The relationship between physical inactivity and mental wellbeing: Findings from a gamification-based community-wide physical activity intervention," 16 January 2018. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5774736/. [Accessed 17 October 2019].

[3] M. E. Donaghy, "Exercise can seriously improve your mental health: Fact or fiction?," 6 March 2007. [Online]. Available: https://www.tandfonline.com/doi/full/10.1080/14038190701395838. [Accessed 21 October 2019].

[4] Ofcom, "A decade of digital dependency by Ofcom," 2 August 2018. [Online]. Available: https://www.ofcom.org.uk/about-ofcom/latest/features-and-news/decade-of-digital-dependency. [Accessed 28 October 2019].

[5] N. C. K. a. S. Panchanathan, "Analysis of low resolution accelerometer data for continuous human activity recognition," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, Las Vegas, NV, 2008.

[6] J. N. a. T. K. Y. Hanai, "Haar-Like Filtering for Human Activity Recognition Using 3D Accelerometer," in *IEEE 13th Digital Signal Processing Workshop and 5th IEEE Signal Processing Education Workshop*, Marco Island, FL, 2009.

[7] W. Y. Y. C. a. Q. L. L. Xu, "Human activity recognition based on random forests," in *13th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery* , Guilin, 2017.

[8] D. N. T. a. D. D. Phan, "Human Activities Recognition in Android Smartphone Using Support Vector Machine," in *7th International Conference on Intelligent Systems, Modelling and Simulation (ISMS)*, Bangkok, 2016.

[9] A. D. P. a. J. H. Shah, “Performance Analysis of Supervised Machine Learning Algorithms to Recognize Human Activity in Ambient Assisted Living Environment,” in *IEEE 16th India Council International Conference (INDICON)*, Rajkot, India, 2019.

[10] S. M. Y. a. H. C. Song-Mi Lee, “Human activity recognition from accelerometer data using Convolutional Neural Network,” IEEE, 2017.

[11] Y. P. Y. Y. a. Y. L. W. Xu, “Human Activity Recognition Based On Convolutional Neural Network,” in *24th International Conference on Pattern Recognition (ICPR)*, Beijing, 2018.

[12] M. S. e. al., “Short-time Activity Recognition With Wearable Sensors Using ConvolutionalNeural Network,” in *ACM SIGGRAPH Conference on Virtual-Reality Continuum and ITS Applications in Industry ACM*, 2016.

[13] S. W. P. a. R. Malekian, “Human Activity Recognition using LSTM-RNN Deep Neural Network Architecture,” IEEE, Pretoria, South Africa., 2019.

[14] J. C. N. Y. a. X. L. T. Yu, “A Multi-Layer Parallel LSTM Network for Human Activity Recognition with Smartphone Sensors,” in *10th International Conference on Wireless Communications and Signal Processing*, Hangzhou, 2018.

[15] T. D. T. N. e. al, “Performance Analysis of Data Parallelism Technique in Machine Learning for Human Activity Recognition Using LSTM,” in *2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, Sydney, Australia, 2019.

[16] T. A. M. a. S. A.-Z. S. Albawi, “Understanding of a convolutional neural network,” 2017. [Online]. Available: https://ieeexplore.ieee.org/document/8308186. [Accessed 4 April 2020].

[17] M. Parekh, “A Brief Guide to Convolutional Neural Network(CNN),” 16th July 2019. [Online]. Available: https://medium.com/nybles/a-brief-guide-to-convolutional-neural-network-cnn-642f47e88ed4. [Accessed 5th April 2020].

[18] A. S. Walia, “Activation functions and it’s types-Which is better?,” [Online]. Available: https://towardsdatascience.com/activation-functions-and-its-types-which-is-better-a9a5310cc8f. [Accessed 5th April 2020].

[19] Uniqtech, “Understand the Softmax Function in Minutes,” 30 January 2018. [Online]. Available: https://medium.com/data-science-bootcamp/understand-the-softmax-function-in-minutes-f3a59641e86d. [Accessed 5 April 2020].

[20] v. luhaniwal, “Forward propagation in neural networks — Simplified math and code version,” 7th May 2019. [Online]. Available: https://towardsdatascience.com/forward-propagation-in-neural-networks-simplified-math-and-code-version-bbcfef6f9250. [Accessed April 7th 2020].

[21] A. MOAWAD, “Neural networks and back-propagation explained in a simple way,” 1 Feb 2018. [Online]. Available: https://medium.com/datathings/neural-networks-and-backpropagation-explained-in-a-simple-way-f540a3611f5e. [Accessed 7th April 2020].

[22] V. Bushaev, “Adam – latest trends in deep learning optimization,” 22 October 2018. [Online]. Available: https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c. [Accessed 4 February 2020].

[23] V. Bushaev, “Understanding RMSprop – faster neural network learning,” 2 September 2018. [Online]. Available: https://towardsdatascience.com/understanding-rmsprop-faster-neural-network-learning-62e116fcf29a. [Accessed 8 February 2020].

[24] P. Gufikandula, “Recurrent Neural Networks and LSTM explained,” 27 March 2019. [Online]. Available: https://medium.com/@purnasaigudikandula/recurrent-neural-networks-and-lstm-explained-7f51c7f6bbb9. [Accessed 2 November 2019].

[25] N. Arbel, “How LSTM networks solve the problem of vanishing gradients,” 21 December 2018. [Online]. Available: https://medium.com/datadriveninvestor/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577. [Accessed 10 April 2020].

[26] A. Budhiraja, “Dropout in (Deep) Machine learning,” 15 December 2015. [Online]. Available: https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5. [Accessed 11 April 2020].

[27] N. e. a. Srivastava, “Dropout: a simple way to prevent neural networks from overfitting,” 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html. [Accessed 11 April 2020].

[28] F. D, “Batch normalization in Neural Networks,” 20 October 2017. [Online]. Available: https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c. [Accessed 12 April 2020].

[29] J. Wren, “How Do I Upsample and Downsample My Data?,” 27 January 2017. [Online]. Available: http://blog.prosig.com/2017/01/27/how-do-i-upsample-and-downsample-my-data/. [Accessed 7th April 2020].

[30] O. -. O. D. Science, “Layer-wise Relevance Propagation Means More Interpretable Deep Learning,” 1 January 2019. [Online]. Available: https://medium.com/@ODSC/layer-wise-relevance-propagation-means-more-interpretable-deep-learning-219ff5158914. [Accessed 7th April 2020].

[31] A. B. G. M. F. K. K.-R. M. W. S. Sebastian Bach, “On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation,” 10 July 2015. [Online]. Available: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0130140#sec001. [Accessed 7th April 2020].

[32] V. T. M. W. a. P. A. F. Y. Yang, “Explaining Therapy Predictions with Layer-Wise Relevance Propagation in Neural Networks,” 26 July 2018. [Online]. Available: https://ieeexplore-ieee-org.manchester.idm.oclc.org/document/8419358. [Accessed 9th April 2020].

[33] F. Chollet, “Keras,” [Online]. Available: https://keras.io.

[34] P. P. E. J. Travis Oliphant. [Online]. Available: https://scipy.org/scipylib/.

[35] M. A. e. al., “"iNNvestigate neural networks",” 13 Aug 2018. [Online]. Available: https://arxiv.org/pdf/1808.04260.pdf. [Accessed 10 April 2020].

[36] W. Koehrsen, “Recurrent Neural Networks by Example in Python,” 5 November 2018. [Online]. Available: https://towardsdatascience.com/recurrent-neural-networks-by-example-in-python-ffd204f99470. [Accessed 2 February 2020].

[37] J. Brownlee, “A Gentle Introduction to the Rectified Linear Unit (ReLU) by Jason Brownlee,” 6 August 2019. [Online]. Available: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/. [Accessed 12 February 2020].