---------------------------------------------------------------------------------------------------------------------

# Exploring Human Activity Recognition

### Machine Learning Activity Classifier

*Using STMicroelectronics SensorTile, EdgeImpulse, JupyterLab, and more*

---------------------------------------------------------------------------------------------------------------------

Lakshmi Umarale, Srinivas Gogula, Shreya Suriya

[Exploring Human Activity Recognition 1](#_Toc78559725)

[Machine Learning Activity Classifier 1](#_Toc78559726)

[1. Introduction 3](#_Toc78559727)

[What is Human Activity Recognition? 3](#_Toc78559728)

[Our Motivation 4](#_Toc78559729)

[Related Work 4](#_Toc78559730)

[The Purpose 4](#_Toc78559731)

[2. Materials 4](#_Toc78559732)

[2.1WISDM Dataset 4](#_Toc78559733)

[2.2 Software 5](#_Toc78559734)

[2.3 Hardware 6](#_Toc78559735)

[3. Data Acquisition 6](#_Toc78559736)

[4. Modeling 7](#_Toc78559737)

[4.1. Edge Impulse Model 8](#_Toc78559738)

[Step 1: Log the Data 8](#_Toc78559739)

[Step 2: Format the Data 9](#_Toc78559740)

[Step 3: Create an Impulse 9](#_Toc78559741)

[Step 4: Generate Features 10](#_Toc78559742)

[Step 5: Train the data 10](#_Toc78559743)

[Step 6: Test the Model 13](#_Toc78559744)

[4.2. Jupyter Notebook Model 13](#_Toc78559745)

[Step 1: Import the relevant libraries 14](#_Toc78559746)

[Step 2: Load, Inspect and Transform the Accelerometer Data 14](#_Toc78559747)

[Step 3: Splitting the Data 15](#_Toc78559748)

[Step 4: Normalize the Training Data 15](#_Toc78559749)

[Step 5: Reshape Data Into Segments and Prepare for Keras 15](#_Toc78559750)

[Step 6: Training Neural Network Models 15](#_Toc78559751)

[Step 6: Check Against Test Data 17](#_Toc78559752)

[Step 7: Making Predictions 19](#_Toc78559753)

[5. Results and Discussion 20](#_Toc78559754)

[5.1. Edge Impulse Model 20](#_Toc78559755)

[5.2. Jupyter Notebook Model 20](#_Toc78559756)

[6. Conclusion 20](#_Toc78559757)

[7. Future Work 20](#_Toc78559758)

[8. References 21](#_Toc78559759)

# Introduction

## What is Human Activity Recognition?

As technology rapidly advances, we see a greater demand for complex systems in the status quo. Human activity recognition plays a significant role in human-to-human interaction and interpersonal relations as it provides information about a person’s identity, personality, and psychological state. In another light, HAR is the problem of classifying sequences of accelerometer data recorded by specialized harnesses or smartphones into known well-defined movements. This concept is applied in various areas, such as video surveillance, healthcare, and human-computer interaction.

## Our Motivation

We aim to study various models and their accuracy to discover what would be best for the concept of HAR. The overall pursuit of this project is to investigate what software, what features, and more we would need to make the most accurate model, which would successfully classify our activities.

## Related Work

There have been many projects done utilizing Human Activity Recognition Technology. Such is documented in various survey papers. These survey papers introduce the recent advances in automated human activity recognition topic. A few of these works provide a review on different aspects of HAR methods similar to the goal of our project, while most of them look at HAR tasks from a specific point of view. For example, projects have been conducted to classify HAR approaches according to the spatial-temporal characteristics of actions, video segmentation and recognition systems and camera modalities. Likewise, there are also projects that advocate for a taxonomy-based approach and they compare the advantages and limitations of each method.

## The Purpose

The purpose of this project is to dive deeper into the possibilities of Human Activity Recognition to see how accurately we will be able to identify and classify a range of different movements a human may perform into their respective classes. To do this, we created various models built with different software. We trained such models with the WISDM dataset, containing seven different labels of human activity. We then used our data collected from the STM SensorTile and SensorTile box to test our models and assess our accuracy.

# Materials

## 2.1WISDM Dataset

The data set we will be using is the WISDM Dataset. The WISDM lab collects and mines sensor data from wireless devices. Their focus is mainly mining the accelerometer data for tasks like biometric identification and activity recognition, which directly applies to our project. This particular dataset contains data collected through controlled, laboratory conditions.

The data contains 6 labels; Walking, Jogging, Standing, Sitting, Upstairs and Downstairs.

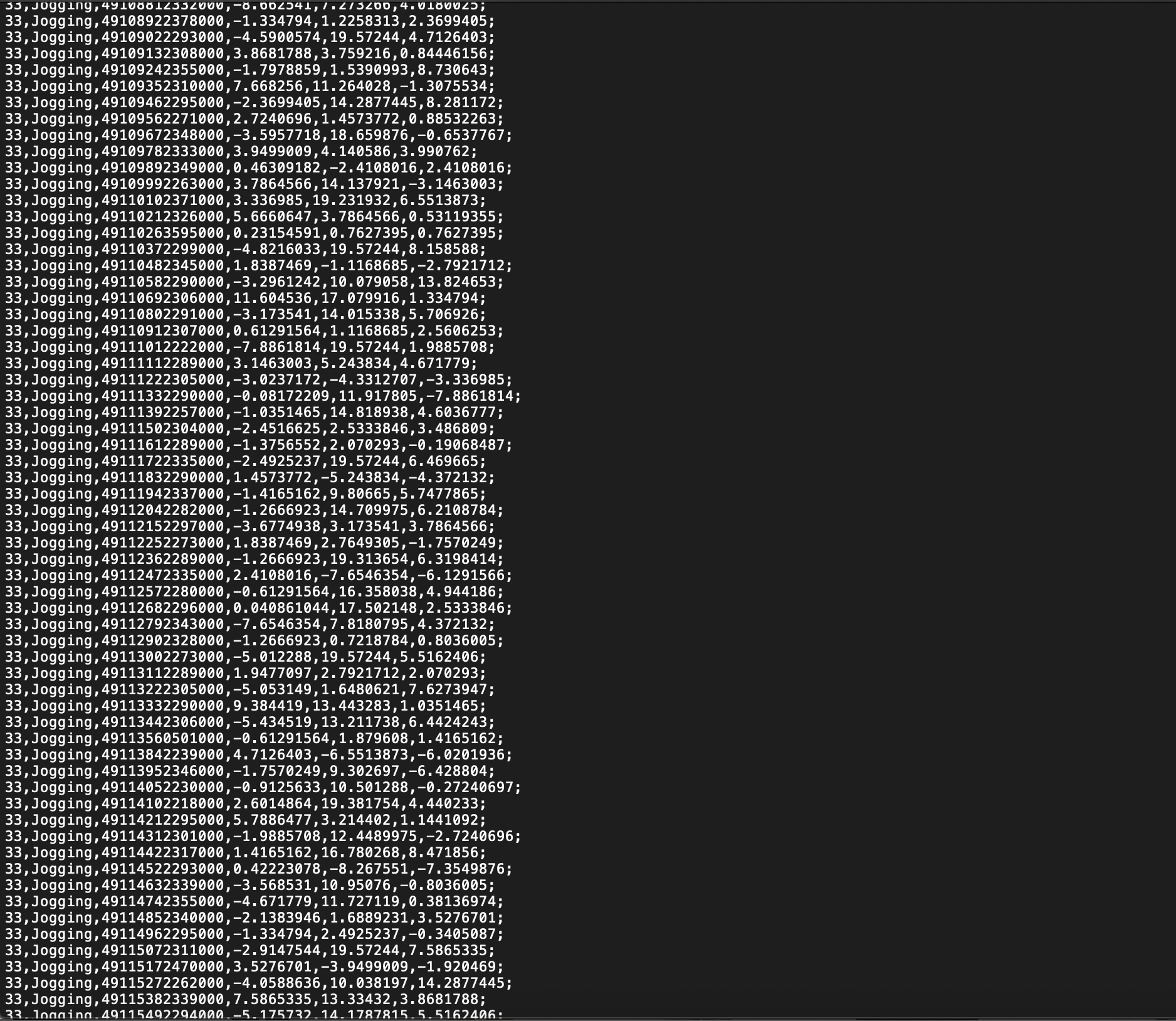


Figure 2‑1WISDM Raw Data

<https://www.cis.fordham.edu/wisdm/dataset.php>

The Raw Time Series Data has 1,098,207 examples, with six attributes. The Transformed Examples Data has 5,424 transformed examples and 46 transformed attributes.

The files that this dataset contains are:



Figure 2‑2 WISDM dataset files

For our project, we will be using the ***WISM\_arv1.1\_raw.txt*** as the file for our training data.  
([WISDM dataset in GitHub](https://github.com/gsreddy99/har/blob/main/WISDM_ar_v1.1_raw.txt))

## 2.2 Software

The below software platforms are used to complete the project:

Data Collection:

* STBLE App

Communication and Sharing:

* Gmail
* GitHub
* Google Docs

Building Models:

* Jupyter Notebook includes python libraries:  
  [Human Activity Recognition Using WISDM dataset](https://github.com/gsreddy99/har/blob/main/HumanActivityRecognitionUsingWisdm.ipynb)
* EdgeImpulse

## 2.3 Hardware

The below hardware is used for data collection:

Data Collection:

* SensorTile
* Micro USB Cable
* Power Bank

Communication and Sharing:

* SmartPhone

Building Models:

* Laptop

# Data Acquisition

To validate our model’s accuracy, we collected test data using the STM SensorTile. SensorTile is a tiny, square-shaped IoT module that packs powerful processing capabilities leveraging an 80 MHz STM32L476JGY microcontroller and Bluetooth low energy connectivity based on BlueNRG-MS network processor as well as a broad spectrum of motion and environmental MEMS sensors, including a digital microphone.

<https://www.st.com/en/evaluation-tools/steval-stlkt01v1.html>

For this project, we focused on Accelerometer data mined from the SensorTile. We logged our acceleration in the X, Y and Z axis using the STBLE App, connected through Bluetooth to the SensorTile. The SensorTile was placed around the waist and secured with a belt, facing upwards to sense all ranges of motion for each activity. Each activity was logged for sets of 40 seconds.

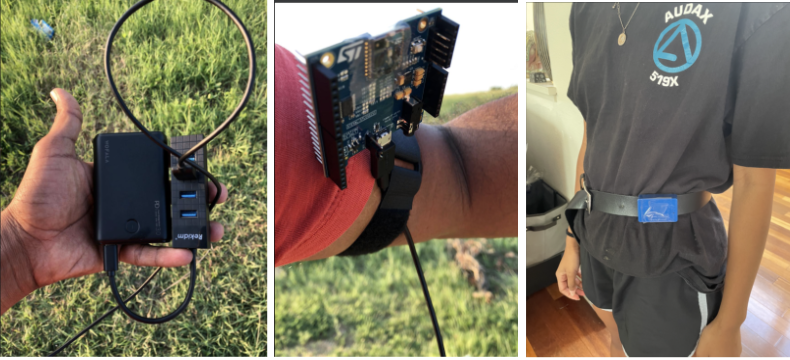


Figure 3‑1 Collecting Accelerometer Test Data

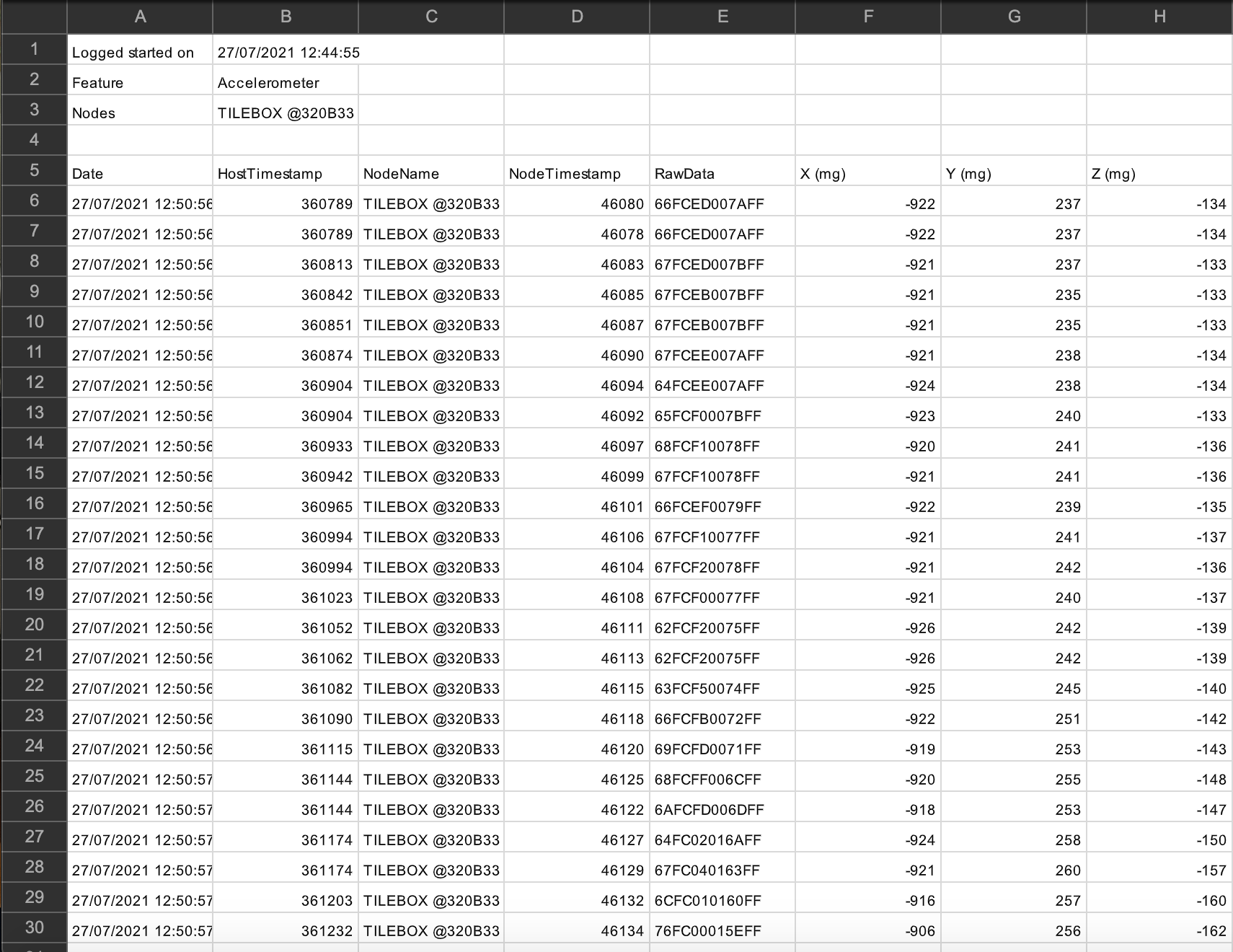
******

Figure 3‑2 Example of Accelerometer Data Format

# Modeling

We created two main models for this project; One using the software Edge Impulse and the other with code in Python, using Jupyter Notebook. We will test these models with data collected from the Sensor Tile (*more on this topic in 3. Data Acquisition*). In this section, we will discuss the steps taken to build each model.

The typical steps for solving a machine learning problem are depicted below and we ran through a very similar process.

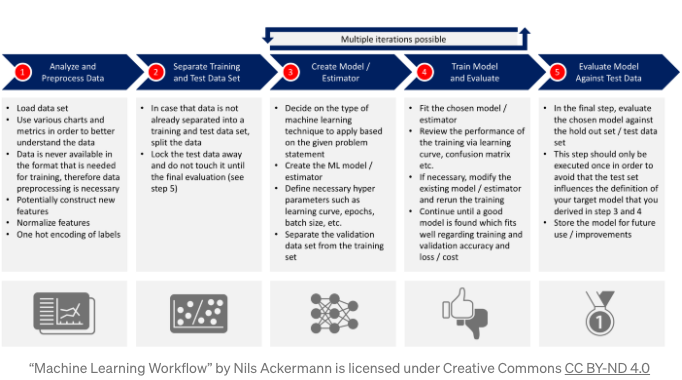


Figure 4‑1 – Machine Learning Workflow

## 4.1. Edge Impulse Model

What is Edge Impulse?

Edge Impulse was designed for software developers, engineers, and domain

experts to solve real problems using machine learning on edge devices without a Ph.D. in machine learning. Using Edge Impulse, users can build a model in real-time using one’s phone’s accelerometer, microphone, or camera to collect data, train machine learning algorithms, and see what happens live on the platform.

We will be using this platform by importing the raw text data file from the

WISDM dataset to the software and setting it as the model’s training data. From there, we will create the model using a neural network, or transfer learning, and test it with our Sensor Tile data to validate the performance.

### Step 1: Log the Data

To collect activity data for Edge Impulse, we selected the feature we wanted to log on to the STBLE App on our smartphones to enable Accelerometer. This allows logging Accelerometer data. Using this format, we performed each activity independently. After collecting each set, we emailed the Accelerometer data to ourselves. The Data is emailed in CSV format, as shown in Figure 3‑2 Example of Accelerometer Data Format.

Text

Description automatically generated

Figure 4‑2 Data collected from Sensor Tile using STBLE app

### Step 2: Format the Data

Next, we downloaded the activity data from the emails sent. However, to upload this data into EdgeImpulse, we must format the data so that EdgeImpulse can analyze it. We formatted the information by deleting all the data except X, Y, and X columns and we populated the Timestamp with values.

Text

Description automatically generated

Figure 4‑3 Formatted dataset for EdgeImpulse

### Step 3: Create an Impulse

After formatting, we uploaded each activity dataset into EdgeImpulse to create an impulse.

Graphical user interface, application, Teams

Description automatically generated

Figure 4‑4 Data Acquisition in EdgeImpulse

### Step 4: Generate Features

We processed the data using Spectral Analysis/RAW Data in EdgeImpulse and generated features from the Activity Data. We then added a deep learning block using Keras classification.

Graphical user interface, application, Teams

Description automatically generated

Figure 4‑5 Feature generation in EdgeImpulse

### Step 5: Train the data

Keras classification was used to classify the data and several improvements were made to improve the performance of the model.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Neurons** | **Drop out layer** | **Flatten Layer** | **Accuracy** | **Change** |
| 50 | None | None | 74.5% |  |
| 50 | Yes | Yes | 76.5% | +2% |
| 100 | None | None | 81.4% | +4.9% |
| 100 | Yes | Yes | 85.3 | +10.8% |

Table 4‑1 Model Structure

Graphical user interface, application, Teams

Description automatically generated

Figure 4‑6 Training performance with default settings in edgeimpulse

Graphical user interface, application, Teams

Description automatically generated

Figure 4‑7 Training performance with dropout and flatten layer

Graphical user interface, application, Teams

Description automatically generated

Figure 4‑8 Training performance with 100 neurons in edgeimpulse

Graphical user interface, application

Description automatically generated

Figure 4‑9 Training performance with 100 neurons, Dropout and Flatten layer in edge impulse

Neural Network settings were modified in edgeimpulse to improve the performance of the training data. 10 percent performance gain was achieved by increasing the neurons in Dense layer from a default setting of 30 to 100 and adding Dropout layer and Flatten layer.

### Step 6: Test the Model

After saving this the Impulse, we began to train the data multiple times with Transfer Learning for high accuracy. To test our model, we used Model Testing using our test data. We then verified the confusion matrix to determine how many samples were predicted correctly and affirmed the training and test accuracy.

Graphical user interface, application, Teams

Description automatically generated

Figure 4‑10 Confusion Matrix using Test Data

We achieved an overall 80.7 % accuracy with the test data. Except Upstairs and Downstairs, all other activities were achieved an accuracy of greater than 64%.

## 4.2. Jupyter Notebook Model

Comparatively to Edge Impulse, we created our Jupyter Notebook Model with code. We programmed this model in Python, while using many different libraries:

|  |  |
| --- | --- |
| python | 3.8.10 |
| tensorflow | 2.5.0 |
| tensorflow.keras | 2.5.0 |
| numpy | 1.19.5 |
| pandas | 1.2.5 |
| scipy | 1.7.0 |
| seaborn | 0.11.1 |

Table 4‑2 Python libraries

Parallel to the Edge Impulse Model, we uploaded the WISDM raw text file as our training data, and also split some of the data into test and validation sets. We then defined models with various neural networks for Human Activity Recognition, and validated the performance of the trained model using a learning curve and confusion matrix. We also tested these models with our Sensor Tile data that we collected for each activity.

### Step 1: Import the relevant libraries

### Step 2: Load, Inspect and Transform the Accelerometer Data

To begin, we downloaded the WISDM dataset, and uploaded it to GitHub

and used it locally. Again, for this project we focused on Accelerometer data.

The next step in the process was to perform Exploratory Data Analysis on the data

set and visualize the accelerometer data.

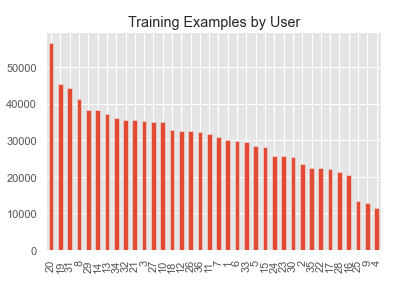
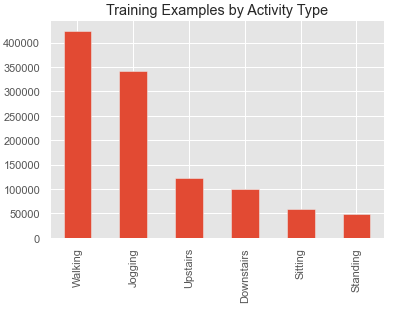


Figure 4‑11 Training Examples by Activity Type/User

As we can see, we have more data for walking and jogging activities than we have for the

other activities. Also, we can see that 36 persons have participated in the experiment.

Next, let’s take a look at the accelerometer data for each of the three axes for all six possible activities. The data is recorded at a sampling rate of 20 Hz (20 values per second). Since we show the first 180 records, each chart below shows a 9 second interval for each of the six activities (calculation: 0.05 \* 180 = 9 seconds).

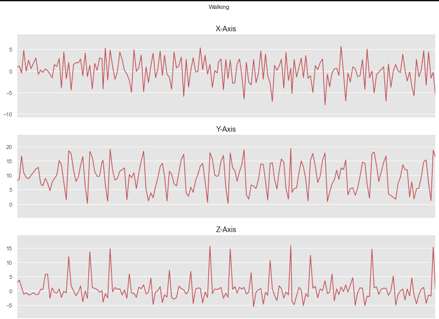
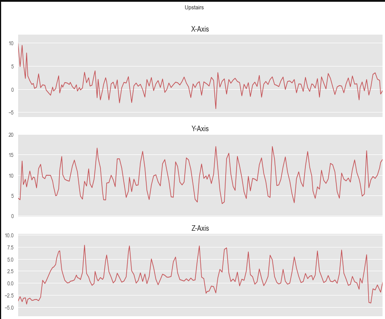
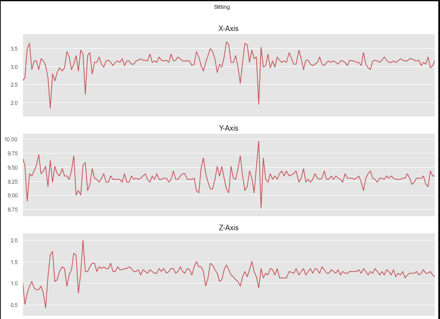
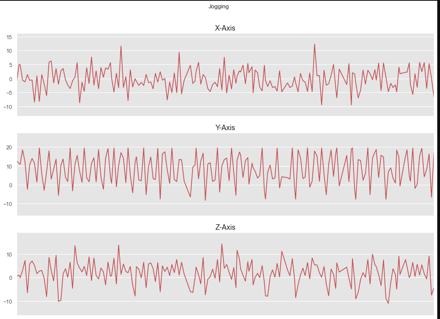
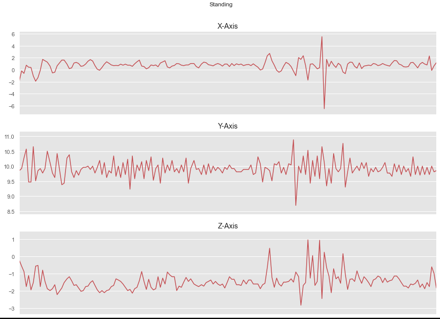
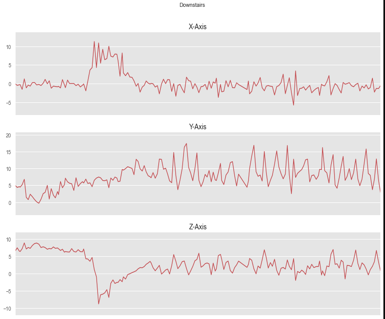


Figure 4‑12 Plot of various activities in the X, Y and Z axis

As expected, there is a higher acceleration for activities such as jogging and walking compared to sitting. We added one more column with the name “ActivityEncoded” to the data frame with the encoded value for each activity. This is needed since the deep neural network cannot work with non-numerical labels. With the Label Encoder, we are able to easily to convert back to the original label text.

### Step 3: Splitting the Data

To build our model using the libraries we imported, we need training data, test data, and validation data. To obtain these sets we split the WISDM dataset into training, validation and test sets. The best splitting approach in our case, was to split the data based on their user IDs. We kept the users with ID 1 to 28 for training the model and users with ID greater than 28 for the test set.

### Step 4: Normalize the Training Data

Next, we normalized the training data (values between 0 and 1) and on top of normalization also applied rounding to the 3 features.

### Step 5: Reshape Data Into Segments and Prepare for Keras

Converted and reformatted accelerometer data into a time-sliced representation

(convert into segments).

In our case, we went with 80 steps. Taking into consideration the 20 Hz sampling rate,   
this equals to 4 second time intervals (calculation: 0.05 \* 80 = 4). Besides reshaping the data, the conversion function also separated the features (x-acceleration, y-acceleration, z-acceleration) and the labels (associated activity).  
   
  
x\_train shape: (20868, 80, 3)

20868 training samples

y\_train shape: (20868,)

For constructing the neural network, we stored the following dimensions-

Number of time periods: This is the number of time periods within one record (since we wanted to have a 4 second time interval, this number is 80 in our case)

Number of sensors: This is 3, since we only use the acceleration over the x, y, and z axis

Number of classes: This is the number of nodes for our output layer (6 – classify six activities) in the neural network.

### Step 6: Training Neural Network Models

We created 3 models with different defined neural networks.

These neural networks include a Deep Neural Network (DNN) in Keras, Convolutional Neural Network (CNN) using TensorFlow, LSTM (Long Short-Term Memory) Neural Network. For each model, we trained the neural network for the Human Activity Recognition data, and validated the performance of the trained neural network against the test data using a learning curve and confusion matrix. Finally, we saved the model and the weights.

Following parameters were used, to compile the models-  
Optimizer – Adam  
Loss Function – Categorical crossentropy was used since we have 6 label classes  
Metrics - Accuracy

Hyperparameters-  
Epochs – 100  
Batch-Size – 128

**EarlyStopping** feature was used to stop training the model when the monitored metric has stopped improving.  
  
**ModelCheckpoint was used to save the** best performing model automatically, by saving its weights to an HDF5 file.

**Deep Neural Network-**

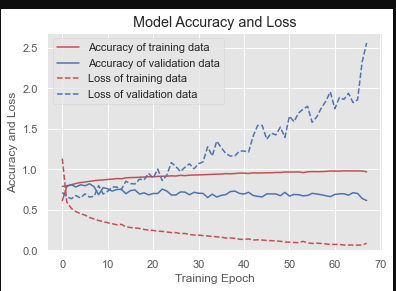
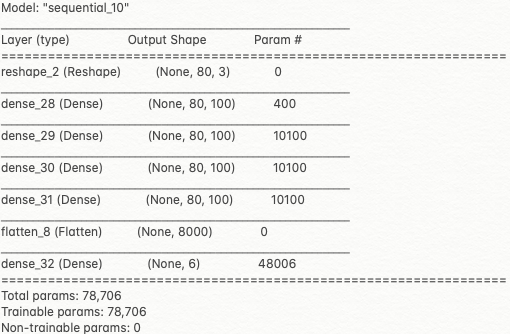


Figure 4‑13 DNN Model and The Model Accuracy/Loss Plot

The validation accuracy was around 67%, which was not great.

**Convolutional Neural Network-**

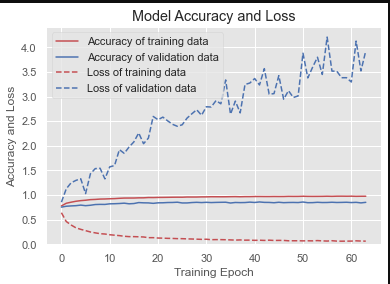
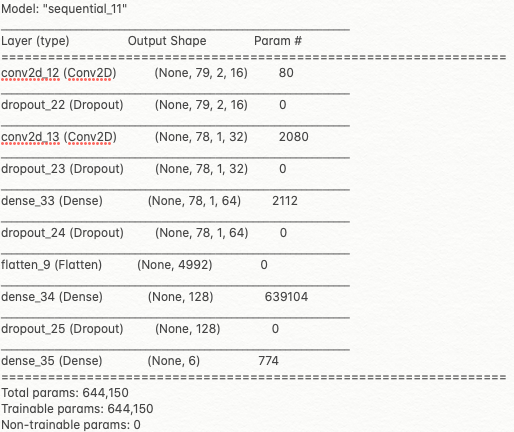


Figure 4‑14 CNN Model and The Model Accuracy/Loss Plot

The validation accuracy was around 85%, which is significantly better than the DNN.

Long Short-Term Memory (Recurrent Neural Network)-  
The benefit of using LSTMs for sequence classification is that they can learn from the

raw time series data directly.

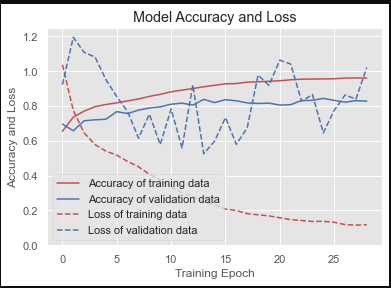
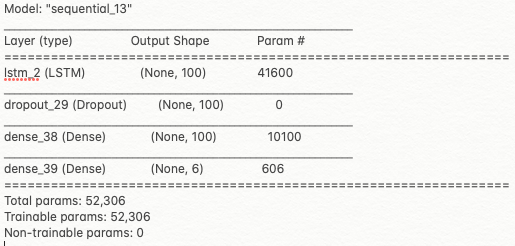


Figure 4‑15 LSTM Model and The Model Accuracy/Loss Plot

The validation accuracy was around 82%, which is significantly better than the DNN.

### Step 6: Check Against Test Data

In our case we checked the performance against the movements of the six users

that the model had not seen.

**DNN-**

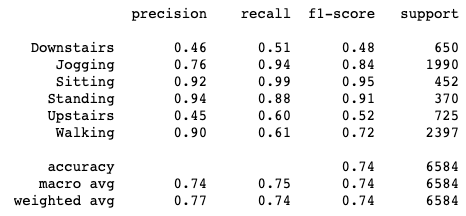
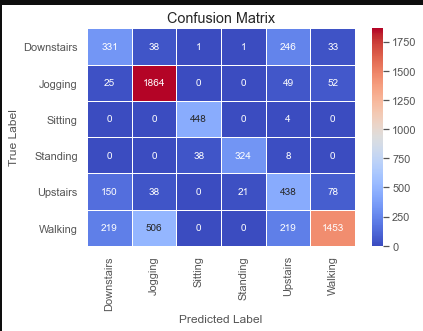


Figure 4‑16 DNN Confusion Matrix and Classification Report

The accuracy on the test data is 74%. This means that our model generalizes well for

persons it has not yet seen. As you can see, the precision of the model is good for predicting jogging , sitting , standing , and walking. This model has problems for clearly identifying upstairs and downstairs activities.

**CNN-**

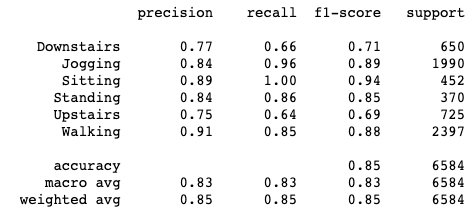
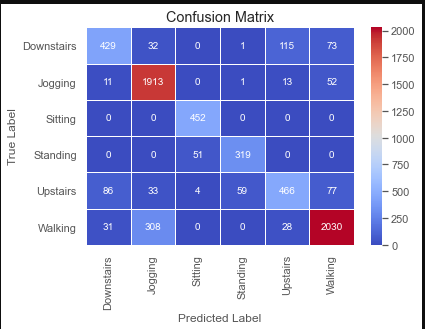


Figure 4‑17 CNN Confusion matrix and Classification Report

The accuracy on the test data is 85%. As you can see, the precision of the model is good

now for predicting downstairs and upstairs as well, compared to DNN model.

**LSTM-**

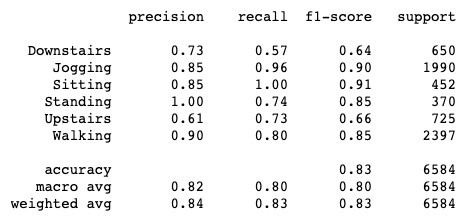
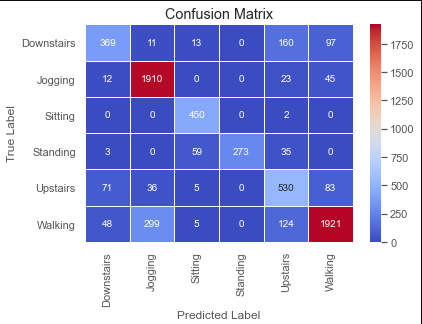


Figure 4‑18 LSTM Confusion Matrix and Classification Report

The accuracy on the test data is 83%. The precision is 100% for standing. Clearly much better than DNN.

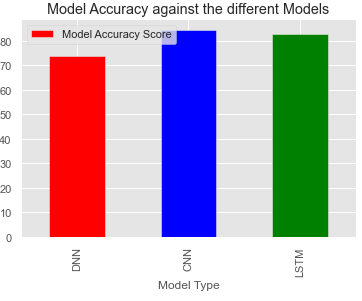


Figure 4‑19 Model Accuracies

### Step 7: Making Predictions

The last step in this process was to visualize the Sensor Tile data collected for all the activities and make predictions.

A helper function was written to parse the sensor tile data to represent the format of the data as in the dataset and to plot the sensor tile data.

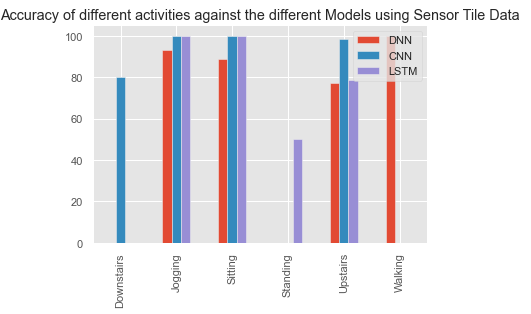


Figure 4‑20 Model Accuracies using Sensor Tile Data

As seen from the chart above, we got very good results for Jogging and Sitting, for all the 3 models.

For Downstairs and Upstairs activity, CNN performed the best.  
With Walking both CNN and LSTM should have done well, since we had high precision, but with the data collected for walking, could only get DNN to classify well.

# Results and Discussion

It is a challenging problem given the large number of observations (tens or

thousands of observations) produced each second, the temporal nature of the observations, and the lack of a clear analytical way to relate the sensor accelerometer data to specific actions in a general way.

Classical approaches to the problem involve hand crafting features from the time

series data based on fixed-size windows and training machine learning models, such as ensembles of decision trees or other classification methods. The difficulty is that this feature engineering requires deep expertise in the field.

Recently, deep learning methods such as recurrent neural networks and

Convolutional neural networks have been shown to provide state-of-the-art results on challenging activity recognition tasks with little or no data feature engineering given their ability to automatically learn higher-order features. Hence, we decided to explore the Deep Learning methods using a 2 folded approach with Edge Impulse and developing the model in python.

## Edge Impulse Model

**<Srinivas adds content here>**

## 5.2. Jupyter Notebook Model

As seen, in4.2. Jupyter Notebook Model, we got better results with CNN and LSTM.

The challenges where we had to collect a lot of Sensor Tile Data for different activities and it might be issues with data acquisition and the positioning of the sensor tile, we were not able to get good predictions always.

# Conclusion

There is of course still great potential for improving the model, by augmenting the data,

or developing further models as described in the next section. For our purpose of showing the end to end process, collecting sensor tile data, developing the model and making predictions, the results were good enough.

# Future Work

Need to improve on the accuracy and do some more research on data acquisition

methods. The movement data recorded was the x, y, and z accelerometer data (linear acceleration).

Plans are also to include the gyroscopic data (angular velocity).

Also augmenting the dataset with statistical features, to get higher accuracy in differentiating the movements.

Research if Transfer Learning can be applied, so that we can improve on the accuracy.

Finally, be able to design HAR applications for healthcare that use wearable sensors.

# References

**References in GitHub-**[HAR Project Proposal](https://github.com/gsreddy99/har/blob/main/HAR_Project_Proposal.docx)[Human Activity Recognition Using WISDM dataset](https://github.com/gsreddy99/har/blob/main/HumanActivityRecognitionUsingWisdm.ipynb)

[WISDM dataset](https://github.com/gsreddy99/har/blob/main/WISDM_ar_v1.1_raw.txt)

**Reference Materials-**

[Human Activity Recognition Tutorial with Keras](https://towardsdatascience.com/human-activity-recognition-har-tutorial-with-keras-and-core-ml-part-1-8c05e365dfa0)

[Kaggle](https://www.kaggle.com/)

[HAR Time Series Classification](https://machinelearningmastery.com/how-to-develop-rnn-models-for-human-activity-recognition-time-series-classification/)

[Load and Explore a standard HAR problem](https://machinelearningmastery.com/how-to-load-and-explore-a-standard-human-activity-recognition-problem/)