School of Computing
DEPARTMENT OF COMPUTER SCIENCE

Find_Fit
Final Report

Prepared For:

Mr Tan Keng Yan, Colin and Mr Boyd Anderson

CS3237 - Introduction to Internet of Things

Member's Name	Matriculation Number
Joel Kweok Zheng Wen	AXXXXXXX
Ahmad Mudaafi'	AXXXXXXX
Tan Jin	AXXXXXXX
Terry Tay	AXXXXXXX

Abstract

Health and fitness is a topic that is important to all. Regardless of one's stage in life, there is a need to incorporate some level of exercise to keep oneself healthy. Tracking these exercises however, has proven to be either a hassle or overly costly. With the advent of the digital age, individuals are now accustomed to having internet-of-things (IoT) devices on their person that could assist them in handling the mundane; such as keeping track of the duration and the intensity of any exercises they've engaged in. Though many solutions are available, full integration has yet to be achieved and users must still actively indicate when they are switching exercises or how many repetition counts they have completed.

Our team looks to take a step towards this ideal by tackling the challenge of identifying the exact exercise being carried out with as little sensory input as possible. The goal is to be able to accurately classify what exercise is being carried out with minimal hardware costs involved. Achieving this would open up a world of possibilities where IoT can passively handle all the boring tasks associated with fitness tracking, leaving us to focus solely on the exercise at hand.

1. Introduction & Problem Statement

Health and fitness remains a relevant topic for a lot of us today. In order for one to stay healthy, consistent exercises have to be done with due diligence. Our group strongly resonated with this topic since all of us incorporate exercises into our daily lives. However, one common problem we all face is that most current applications and fitness trackers do not automatically detect changes in our exercise routines along with repetition counts. For us, this came across as an important feature to allow for a hassle-free exercise experience, where we do not have to interact with technology in the middle of our workouts such as indicating to the application that we have changed exercise or having to manually input our repetition counts post-workout.

As such, our team came together to bring this dream feature into reality, where through minimal sensor input and a well-trained machine learning model, we hope to help take away the mundane task of tracking exercises from users so that they can focus on bringing out the best in themselves.

2. Solution Overview

Broadly speaking, our solution comes in the form of a lightweight mobile application, Find_Fit, along with a small inexpensive sensor. Unlike your typical fitness tracker, Find_Fit is imbued with the ability to predict different forms of exercises based on the current movement of its user. This is done with the aid of the values provided by the sensor that is to be worn on the user's wrist. Find_Fit currently supports three forms of exercises which includes running, push ups and skipping. It also includes support for detecting changes in exercise types and will automatically update its exercise classification accordingly.

3. IoT Architecture

Given that we intend to carry out live classification of the user's exercise, our devised solution must be able to do real-time processing of the data collected. To achieve this, the following IoT architecture was devised.

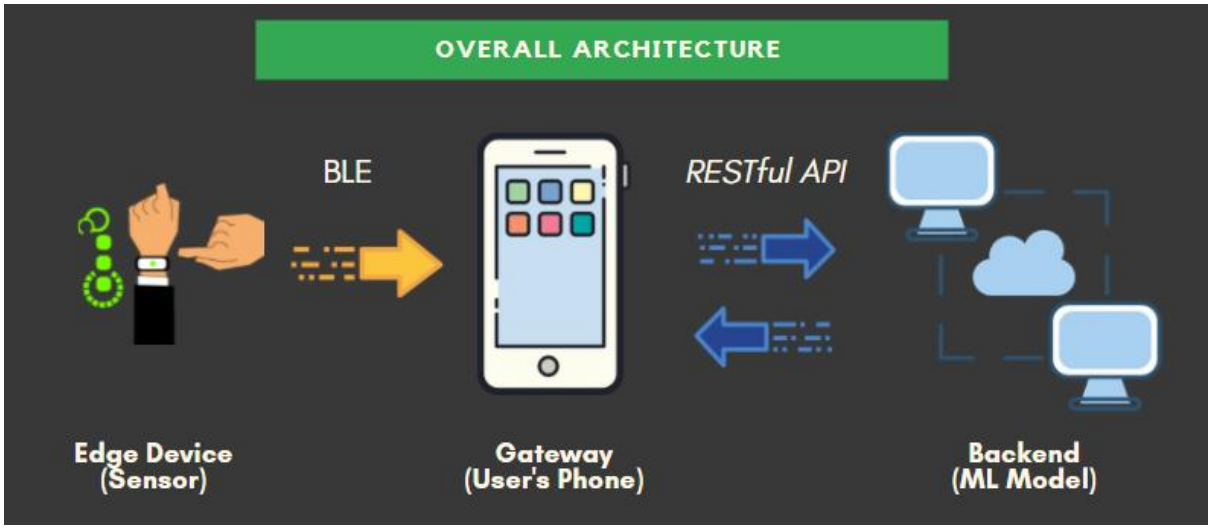

Figure 1. Overall IoT Architecture

As seen in the figure above, the IoT sensor will be mounted on the user's wrist. This sensor will then communicate with the gateway device via Bluetooth Low Energy (BLE), transmitting the relevant data during the session. The gateway device will have our very own iOS application, Find_Fit, that collates the data and sends it to our backend server for classification via the RESTful API. There, the backend server will format the data received appropriately before classifying it using a Machine Learning (ML) model and returning the result to the iOS application.

3.1 Find_Fit - The iOS Application

Coded in Swift, the iOS application on the user's device is the main user interface for Find_Fit. As shown in figure 2, users will be able to see:

(a) Connection status of the sensor
(b) Battery life of the sensor
(c) Start button to begin an exercise session
(d) The time spent in the session
(e) The classified exercise
(f) Stop button to end the current session

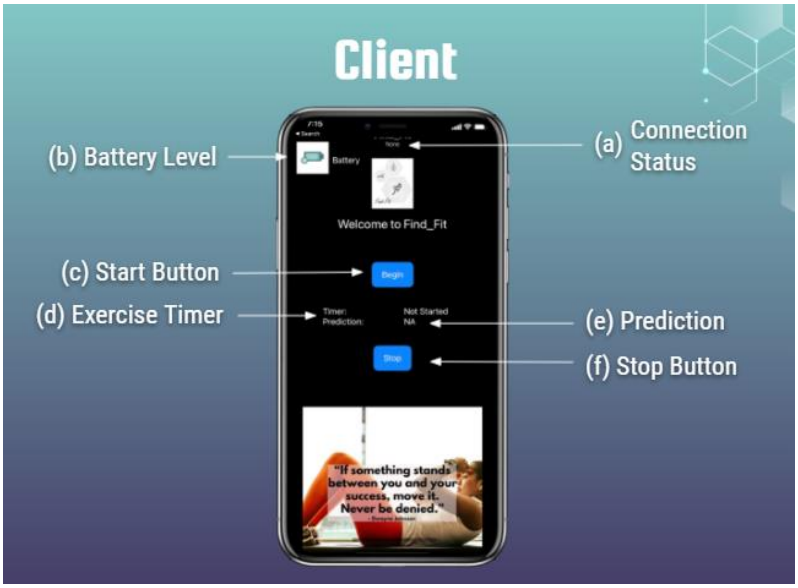

Figure 2. Main User Interface for Find_Fit

Note that while the session is still ongoing, the Find_Fit application will continue to send the most recent data to the Backend Server, displaying the most returned classification to the user. This further improves the user experience as incorrect exercise classifications resulting from edge cases will quickly be corrected.

3.2 Backend Server

The backend server runs on Python using the Flask web framework and is hosted on a Virtual Private Server (VPS) from OVH. The endpoint itself is "api/v1/predict/" which accepts POST requests and returns a JSON object in the response.

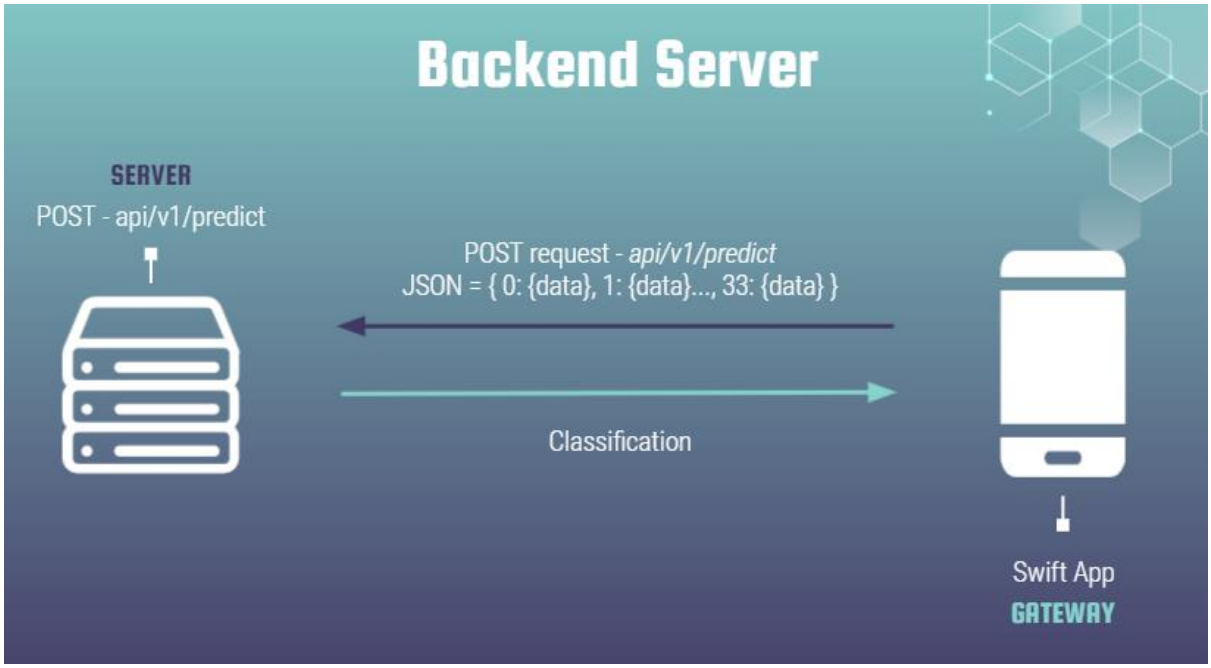

Figure 3. Communication between Gateway and Backend Server

As seen in the figure, the Find_Fit application compiles 10s worth of sensor readings collected at 0.3s intervals and transmits it in JSON format, where each key represents an individual row, via the aforementioned API endpoint. The Backend Server receives this POST request and converts it into a 1x33 numpy array. This numpy array is then passed to the ML model and the classification is returned back to the iOS application to be displayed to the user.

4. The Machine Learning Model

The crux of Find_Fit is its ability to discern whether the user is currently doing push-ups, skipping or running, all from the data provided by the sensor. To do this, we opted to create a Machine Learning (ML) model to assist us in the classification. We tackled this task using three main steps.

4.1 Data Collection

Training an ML model requires a lot of data, and given the timeline of the project, there was an urgent need to begin obtaining as much of this data as fast as possible. To do this, we came up with a simple Python script that allowed us to sample the sensor's accelerometer and gyroscope at a sampling frequency of 3.33hz. The obtained data would then be sent, together with a unique session id, to a PostgreSQL database in the cloud. This allowed each of us to continuously and independently record our own exercise sessions, generating the required data for model training later on.

It should also be noted that the data collected is in the form of a time-series and that the length of each time-series is not standardized. This was done intentionally and will be elaborated more in the following section.

4.2 Data Sanitization and Augmentation

Before the collected data can be used for training our ML models, it must first be appropriately pre-processed. First, we apply sanitization techniques to remove unwanted noise from the dataset. We do this by trimming both the initial and end of each time-series to account for the transition time needed to move from initializing the script to the actual exercise itself. We also chose to drop zero-valued rows which represented periods of extreme idleness.

Next, to augment the dataset, we took advantage of how each time-series had variable length by extracting windows. Initially, we applied a step-wise window to each time-series but quickly moved on to using a rolling window instead. Doing so not only allows to generate much more training points per time-series but also helps the ML model better account for differences in starting and ending points in a given window. After much experimentation, we also found the most success using 10s window intervals.

The above augmentations were sufficient for our current purposes, but we also considered other augmentations as well. The Hamming Window was an interesting alternative to the rolling-window that essentially softens the significance of the edges of a given time-series window, giving more emphasis to the features captured in between. Mutating the time-series using

elongation and compression techniques also seemed promising as they synthetically generated more training points by simulating various exercise speeds.

4.3 Model Experimentation

There were many models we considered, experimented on and attempted to fine-tune. In this process, we often found ourselves returning back to the previous two steps mentioned earlier to either collect more data or better refine our existing dataset. The accuracy scores listed here are the final outcomes of many rounds of data collection, augmentation and iterative parameter fine-tuning.

To create these models, we utilized the Tensorflow library for Python.

4.3.1 Long-Short Term Memory (LSTM) Model - [63% Accuracy]

Initially, we hypothesized that the LSTM model, with its ability to retain contextual information while still being able to focus on the immediate action, would be a good fit for our time-series data at hand. As our problem involved not just classifying an exercise by a single repetition but my many repetitions strung together, it was thought that the LSTM model would be able to effectively recognize individual reps, isolating them in order to classify the overall exercise being done. This however turned out to be a relatively incorrect postulation as we could only achieve up to 63% accuracy no matter how much fine-tuning we attempted to do.

4.3.2 Multi-Layer Perceptron (MLP) Model - [57% Accuracy]

We also attempted a simple MLP model to test and see if going back to the basics would yield any results. This again did not turn out to be the case and we only managed a maximum of 57% accuracy.

4.3.3 Principal Component Analysis (PCA) paired with MLP - [90% Accuracy]

After noting the difficulty in obtaining a good model with the raw time-series data, we hypothesized that extracting the key features and giving them more weight might allow our ML models to better learn the concept. To do this, we decided to apply PCA to extract the top 30 features that would distinguish each exercise from one another. Doing so proved to be immensely worthwhile as we managed to cap out at a 90% accuracy.

4.3.4 Convolutional Neural Network (CNN) - [100% accuracy]

After our success with PCA, we did some further analysis on our present dataset and plotted the results on a 2d-graph. As seen in the figure below, we noted that each individual exercise did look relatively distinct from one another. This inspired us to experiment with CNN models which inherently extract features, especially since we already knew that feature extraction would be very useful prior to model training.

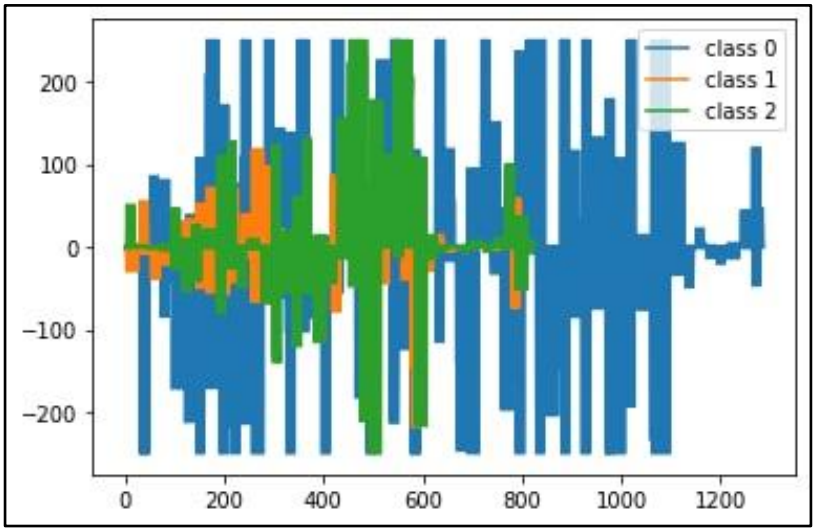

Figure 4 - Distinct 2d plot of Time-Series Data

This hypothesis turned out to be true. Using a combination of Tensorflow callbacks such as ReduceLROnPlateau, we managed to achieve an accuracy score of 100%. This accuracy score is likely to be too good to be true and might indicate either overfitting on the available data or insufficient diversity in the data collected. Nonetheless, with our available data, this CNN model is our best performing classifier.

5. Implementation Details

5.1 Sampling Frequency

When the sensor is connected to the application, the sensor is polled 3 times every 1 second. This means that the application takes in accelerometer and gyroscope data three times in 1 second. The reason for the high frequency rate is because one repetition of an exercise such as a pushup or a skip can be done in less than a second. Hence, to send meaningful data over to the ML model to classify the exercise, it requires a sampling frequency of less than a second.

5.2 Power Management

The high sampling frequency would result in a high battery drain and this is not ideal for the IOT system. Hence we have implemented a few features to ensure that power is optimized. Firstly, we have given the application a 'stop' function. This function not only stops the timer of the application, it also disconnects the application from the sensor. This disconnection also means that the sensor is no longer sending anymore data and helps conserve the sensor battery when the user has finished exercising. Secondly, the application only activates the sensor that is needed during the Bluetooth connection. This is done by the application which specifies the enabling of only the sensor that has the UUID of the movement sensor and the battery. This reduces the power consumption of the sensor as only the relevant sensors are activated.

5.3 Communication Protocol

The main communication protocols that are used are Bluetooth and RESTful API. As highlighted in the earlier section, Bluetooth is used to send the data from the sensor to the phone while RESTful API is used to send the data from phone to server. Flask is used as the framework to support the backend API server with its threaded options set to true to allow for concurrent requests. The response containing the prediction from the ML model is returned via the same communication protocols.

6. Experimental Evaluation

6.1 Experimental Accuracy

After the IOT system was built, the group initially had a poor success rate for the sensor. Out of the 3 exercises, only 1 of the 3 had a consistent classification accuracy.

Subsequently after some experimentation and tinkering with the device, we found that the following adjustments had to be made in order to ensure the device had a higher success rate.

6.1.1 Adjustments during skipping exercise

The original strap band, as seen in figure 5 and 6, that was used to tie the sensor over the wrist had to be removed. The sensor was instead placed at the palm of the user's hand when the

skipping motion was done. This was to ensure the unique hand turning motion is captured more accurately in the sensor. The next adjustment was that the usual hand rotating motions that are used for skipping had to be exaggerated more - the hand had to rotate at more obvious angles and at a quicker pace - in order for skipping to be classified correctly.

Figure 5: Sensor with wrist band components.

Figure 6: Sensor attached to wrist band.

6.1.2 Adjustments during push-ups

The wristband that was used to tie the sensor had to be strapped further up the wrist and had to be bound tightly. This was to ensure that the wristband would not droop down which would affect the level of motion the sensor had and affect the results. After this adjustment was done, pushups were classified more accurately.

6.1.3 Adjustments for running

The running motion of the user needed to be at a jogging pace. If the motion was too fast, the sensor would predict skipping while if the motion was too slow, the sensor would predict push-ups. Thus we found that if the user ran at a moderate pace, the sensor would classify running more accurately.

After these 3 adjustments were made, the accuracy of the IOT system was much higher and the 3 exercises had a much higher classification rate.

6.2 Reasons for discrepancy between success rate of ML model on testing data and experimental success rate

There are 2 main reasons for the differing accuracy rate. First, the amount of data gathered to train the model is still relatively small despite our best efforts. It is possible that the range of data used for model training is not sufficiently diverse, and hence insufficient variance in the exercises was present to allow our model to generalize effectively. In production, user body types would differ and the manner in which they execute the exercises would vary significantly as well. This poor generalization would lead to inconsistent results and poor classification rate. Hence, to facilitate better model generalization, there needs to be more data gathered from a more diverse sample of individuals.

Second, the script used to collect the gyroscope and accelerometer data for training is different from the one used to collect data for classification. Theoretically, though the scripts differ, there should not be a difference as they both poll data from the accelerometer and gyroscope of the same sensor. However, inconsistencies in the way the polling is carried out together with any delays that might occur would affect the data being sent for classification. Pre-processing of data when classifying was also difficult as data was sent in fixed lengths rather than in variable length as done in the data collection phase. Thus there may be incongruencies between the data used for model training and the data piped to the model for classification, leading to a lower than expected actual classification accuracy.

6.3 Battery life estimate

The sensor uses a CR2032 coin-cell battery. When the sensor is switched on and actively sending data, it has a battery life estimate of 600 - 800 hours. This was calculated by switching on the sensor and calculating the amount of time needed for the battery to drop 1% point. 6 to 8 hours was needed to drop 1% point and hence the total battery life would be 600 - 800 hours at 100% battery. No battery life estimate for the sensor's sleep state was calculated as the nature of the application meant that the sensor is either switched on or off. Thus there is no sleep state needed and no battery life estimate for the sleep state.

7. Challenges faced, limitations of the solution and possible future directions

7.1 Challenges Faced

7.1.1 RTOs Implementation

To get RTOs set up, the empty project set up in Code Composite Studio (CCS) does not come with Bluetooth Low Energy (BLE) installed. As such, in order to get bluetooth connectivity up, BLE stack had to be installed. However, there were some difficulties faced during the installation process.

The BLE stack supports only the Windows platform. As most of the developers in this CS3237 project do not use a Windows machine, there was a need to install Windows on a Virtual Machine (VM). After a short time, installation of Windows on VM and BLE stack were successful.

Despite the aforementioned, the development process was not easy. There was a need for a separate ARM compiler which does not ship with the existing tools on hand. To get things working, the team tried to source for documentation and examples. However, the results were limited. In the end, it was decided for the development process of RTOs to be halted. Despite that, there were many learning outcomes that the team picked up during the process.

The team learnt that the CCS platform allows for the modification of the firmware to optimize the power consumption and sample rate. The sample rate can be reduced or increased depending on one's needs. For the unused sensors, they can be switched off. There exists power policies in the RTOs too which one can use to manage power. It was indeed an eye-opening experience.

7.1.2 Limitations of the solution

The final product is running on the default firmware. This means that it is not heavily optimized to meet the level of usage. The mechanism for retrieving the data from the client is through polling from the sensortag. This suffers a bit in terms of power management and performance. If RTOS is used, the mechanism can be switched to listening through notifications.

Next, another aspect of the limitations of the solution is that it is not designed with scalability and production in mind. Although the backend API runs with support for threading on the Flask framework, Flask is typically not suitable for production and it would be more appropriate to serve the request through an application server such as Gunicorn. Thus, if a significant number of users were to start using our application now, performance and reliability may suffer.

7.1.3 Possible future directions

In building our application, the team has considered possible future directions. The first possibility is to implement RTOs for the sensortag. This means that the sensortag can be optimized with better power management and performance. The device can also push notifications to the subscriber upon new sensor readings.

Next, the team considers possible solutions for scalability.

A web socket tends to work better for bi-directional data flow. During predictions, the model will actively predict the type of exercise being performed while the user is doing the specific action. In order to reduce latency and network overhead, the team intends to replace HTTP connections with web sockets.

Last but not least, there is a need to enhance the server for scalability. Instead of running the server on VM in the cloud, the team wants to ship the application as a container in production. This allows for Kubernetes to manage the containers to ensure no downtime. This will increase the reliability of the service. During heavy network traffic, Kubernetes will autoscale the number of containers that the application is running with to support the volume of users in our network. This ensures performance and makes sure that every single user has a good experience.

Lastly, security is an essential part of all applications. We intend to secure the communication between users and our servers using TCP and TLS for privacy purposes. This is to align our application with standard web practices.