DMDG-S-14: Project Final Report

HSweat

A Wearable Device for Remote Home Gym Workout Training

Aug. 5th, 2020

1. Team Name: *West Coast Harvard*

2. Team Members and Roles

Team Members - ***(Group of Four)***

Suzanne Chong	**Micah Nickerson**	**Nicholas Pesce**	**Lucy Chen Zhang**
suzanne.hj.chong@gmail.com	min021@g.harvard.edu	nicholas.pesce@gmail.com	lucyzhangseattle@gmail.com
650-436-3056	917-586-2003	312-919-7202	206-549-9686
Role: **Lead of Product Design** *User Experience Design/ Data Science*	Role: **Lead of Data Science** *Front End Software Development / Machine Learning*	Role: **Lead of Technology** *Software Engineer / Back end Development*	Role: **Data Science/ User Testing** *Concept Design/ Data Collection/ User testing*

3. Project Repository

Final Github Repository:

https://github.com/mjnickerson/s14groupprojectwch/tree/master

4. Project Goals

- ***Primary*** - It's a difficult time to find a personal trainer or workout at a gym during the COVID-19 pandemic. **Gym access has been severely limited by social distancing, and many gym goers feel being at the gym is unsafe.** In this project, we asked how wearable devices can help people work out more efficiently at home, motivate them to get active, and help them measure success. Our proposed system will act as a digital personal trainer, and guide you on how to work out properly and help you exercise more effectively. **This system will be tailored to the home gym workout, first focusing on pulse sit-ups and v sit-ups.** Our primary goal is to guide you with simple workouts at home so you can build more healthy routines with no expensive gear. **We also want to display this feedback in a simple format, e.g. an mobile app you could download, to tell your performance, which is easy to understand.**

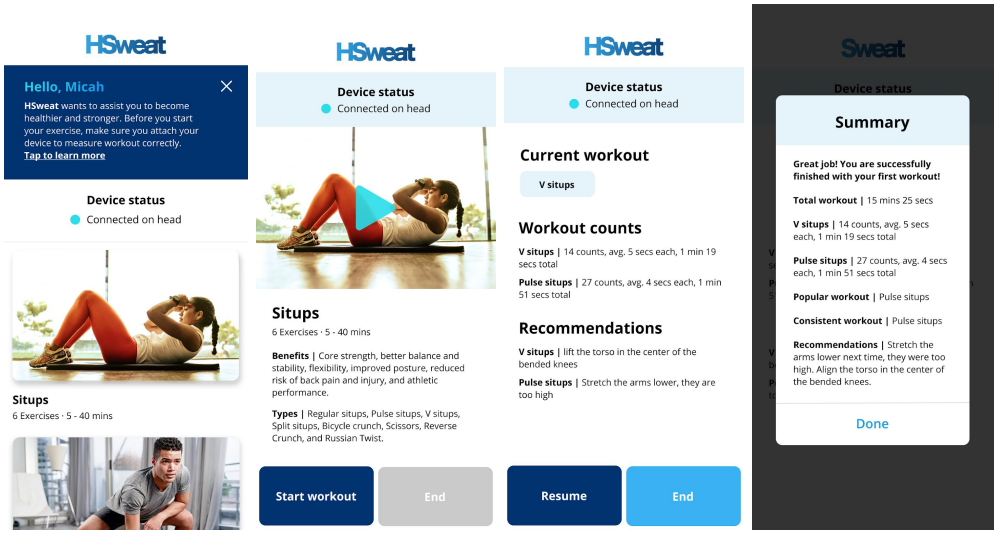


Figure 1: HSweat Application UX Design

- ***Secondary*** - As a next step, we want to use Edge Impulse API to handle data processing and manage model building with in order to make the addition of more exercises as simple and modular as possible, so more workouts can be

added without disrupting the existing models. We will utilize Edge Impulse to deploy the wearable device model into the system as a replaceable module that comprises the data analysis and model prediction. The Edge Impulse platform will be used to collect, host, visualize and process workout data, as well as train our core models.

- ***Tertiary***- We want to provide actionable feedback to the user via their workout data, as well as help the user build a data history. Through the data collected in the application, we want to build users' trust in the quality of measurement and success of the home workout routine, and help them invest in a plan to stay active. We want people can rely on daily, weekly, or monthly data as part of their workout planning, and display this through a web connected display (an app/ web).

5. Product Ecosystem

A. Problem to be solved

The Covid-19 Pandemic has vastly reshaped the world in many negative ways, one of these is the difficulty of getting frequent effective exercise. The closing of many public and private gyms and exercise facilities have increased the difficulty for people worldwide to stay fit and healthy. Having lost access to personal trainers, we are all either relying on a mixture of fragmented information on how to stick to our workouts, (such as online videos, social media, and muscle memory) or investing in expensive subscription and hardware based home screen systems. These **existing solutions are either imprecise, or inaccessible** to all users.

B. Product Solution

We propose a **digital personal trainer:** a cost effective solution that uses wearable sensor technology and any smartphone to determine what exercise the user is doing, and provide visual and haptic feedback on how to improve.

The wearable product concept is named "*Hsweat,* " and is a pair of devices worn together: a **headband with a bluetooth sensor, and an arm mounted phone** (on arm-band). A head and arm band were chosen as they are already accepted locations for wearing functional sports apparel, that do not interfere with the user's ability to workout. Our team determined that introducing a sensor location that did not replace/augment a previously used wearable would inhibit user acceptance and would ultimately get left aside in a home gym.

The headband (BLE sensor) and armband (holder for phone) are to be worn as a set, and **collect two streaming points of data to classify what exercise is being performed.** The armband serves three purposes: it is contact point where the user interacts with the interface, it provides the tethering location for the headband sensor via bluetooth (and required onboard processing), and it also acts as a second accelerometer (internal to the phone). Utilizing a phone as the arm sensor not only reduces the cost of the product, it reduces the number of devices that the user needs to remember to charge, and simply makes it more convenient.

6. Project Scope Completed

For this project, the **arm mounted phone based application has been fully designed and implemented,** and can be accessed via the link adjacent. For this release, it has been **trained to provide feedback on pulse situps and V situps**. Provisions for the headband sensor have been made within the application, but the actual hardware has not been implemented. The app's classification model has been trained to recognize inputs from both head and arm, although it can only be worn on the arm at this time.

The **product application has been made available to the public for open use** through an expo.io shell. It is **supported by a cloud web server** that manages traffic to/from the product, hosts the classification models, and returns classifications back to the app.

7. Data Science

A) Classified Exercise Routines

- **12 home exercise motions have been classified** consisting of
    - different of **pulse sit-ups and V sit-ups**;
    - Sensor on **Head** and **Arm;**
    - Each sit-up/sensor pair has one **correct** method and two **incorrect** methods.
- Based on predicting the correct class, the app will display feedback to the user on whether those exercises are being performed correctly.

B) Data collection

- Accelerometer Data - X, Y and Z Axes (with the null "Z" of 1G calibrated to 10.0);
- 1 Hour 31 min of Training Accelerometer Data;
    - 58% Pulse Sit-ups, in 6 Classes (2 correct, 4 incorrect);
    - 42% V Sit-ups, in 6 Classes (2 correct, 4 incorrect).
- 11 min of Testing Accelerometer Data;
    - 55 sec each, 12 Classes.
- **Method:**
    - Three team members each collected data with an armband and/or headbands;
    - Data was collected directly onto Edge Impulse via it's data ingestion service.

C) Raw Data:

Raw Data Can be accessed here:
https://github.com/mjnickerson/s14groupprojectwch/tree/master/exercise_data

Team's Data specification can be accessed here:

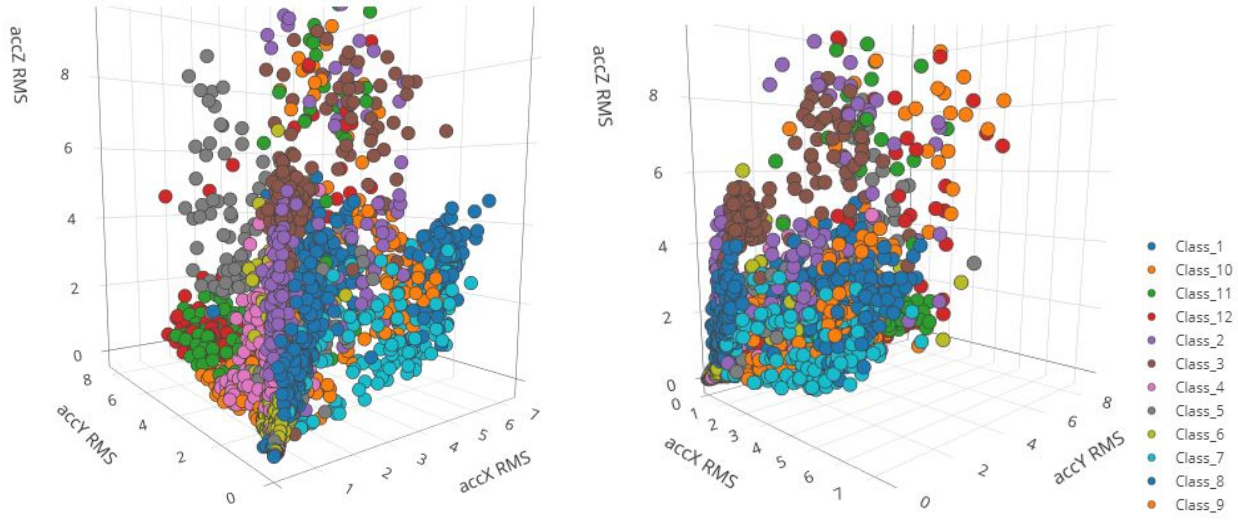
https://docs.google.com/document/d/1pJoaHgUvEgNs17u4p5elDM2PUw1lCjqbkqBajQuA-G4/edit?usp=sharing

D) Exploratory Data Analysis

Raw Data was processed into spectral features and then visualized using three dimensional clustering diagrams below.

Clustering Patterns (Edge Impulse)

Full 12 Class Model



Classification View Classification View-Rotated 90°

Figure 2: Clustering Plots for full "12 Class" Model

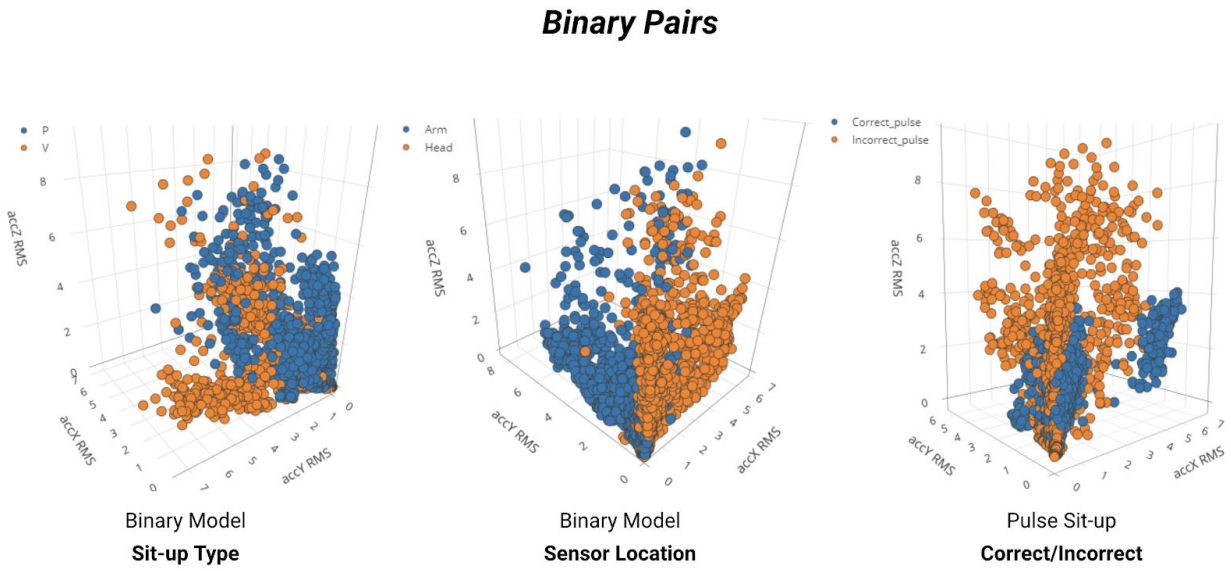


Figure 3: Binary splits for exercise data

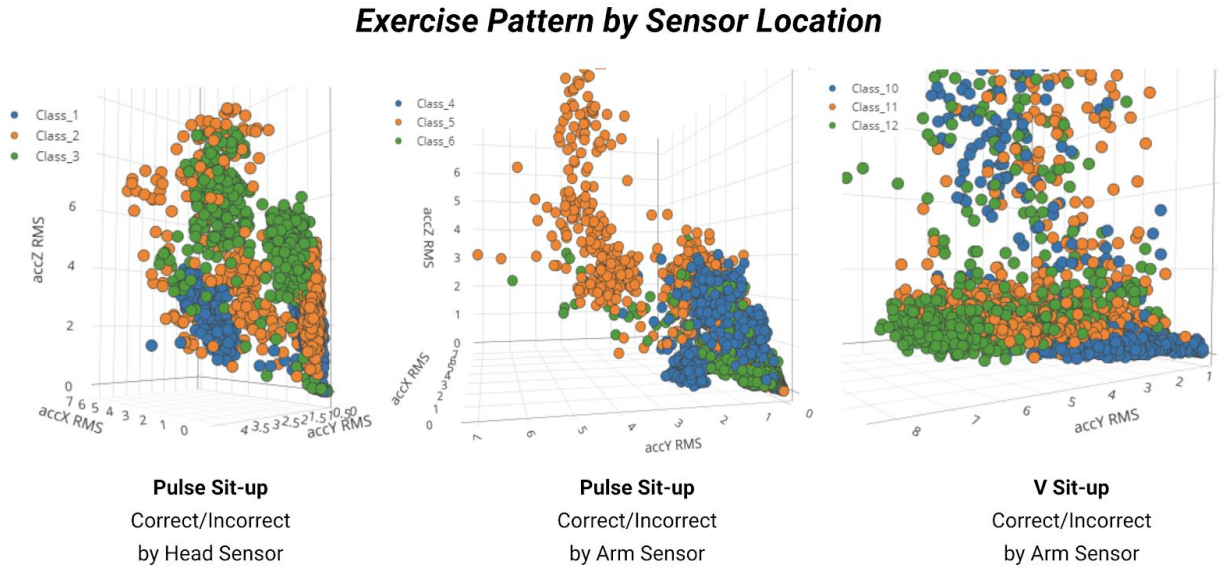


Figure 4: Clustering Plots by Exercise Performance (one correct and two incorrect each)

The 12 class composite model showed well formed clustering for most classes, and so we deemed it acceptable to classify all 12 classes simultaneously. Classes for arm based V-situps were weaker, and showed room for improvement. We also investigated the binary breakdown of each exercise and sensor type, which revealed very strong clustering separations that anticipated very strong binary classification models. Likewise we also

explored smaller models by each exercise/sensor pair, looking at clustering of only correct and incorrect movements. These clustering diagrams also revealed clear clusters will strong predictive ability. Based on our exploratory data analysis, we were satisfied with the amount of data collected, and the spectral features selected, could be used to accurately classify the user's movements.

E) Situp Reps Model (Javascript, onboard device)**:**

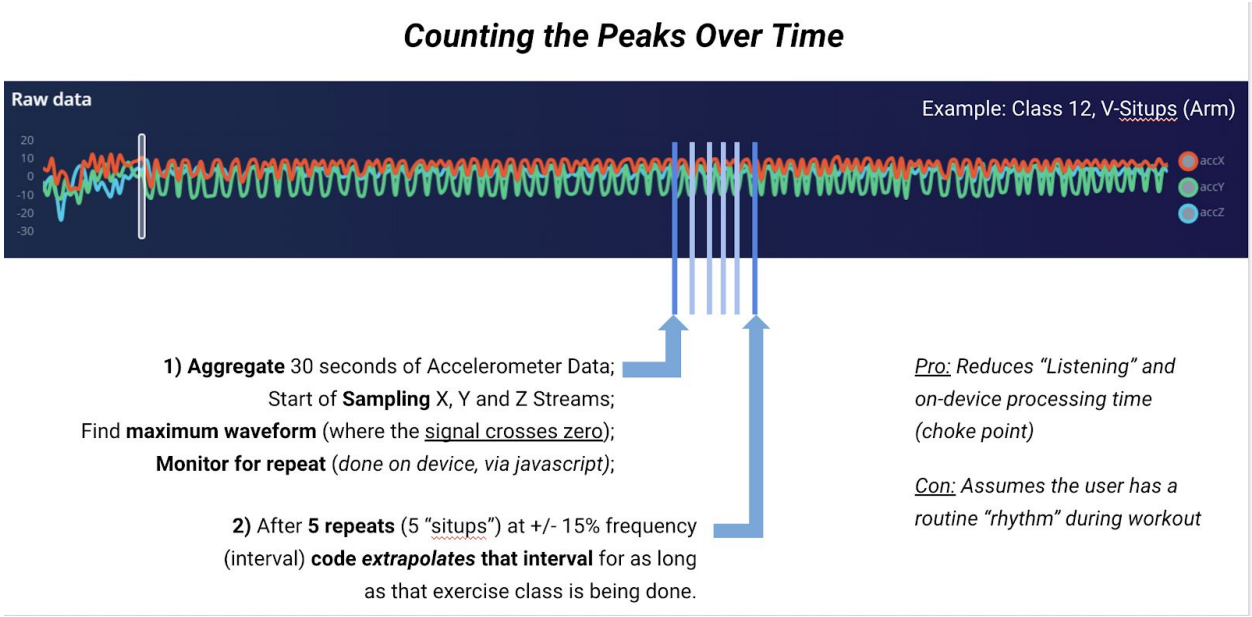


Figure 5: Diagram of Situp Count Algorithm

For counting the total exercises the user is doing, a simple sit-up rep algorithm was implemented. This runs on-board the user's phone (within the app) and aggregates 30 seconds of data, and looks for a consistent interval that the X, Y or Z accelerometer data crosses zero (reverses amplitude or direction). If five repeat occurrences were found, that time difference was set as the "time to do one situp". That interval was extrapolated forward and kept running for as long as the application predicted that exercise was being done. Extrapolating the signal was chosen to reduce the amount of onboard processing time, as it was logical that most users had a standard "rhythm" that would not change over the time of that specific exercise.

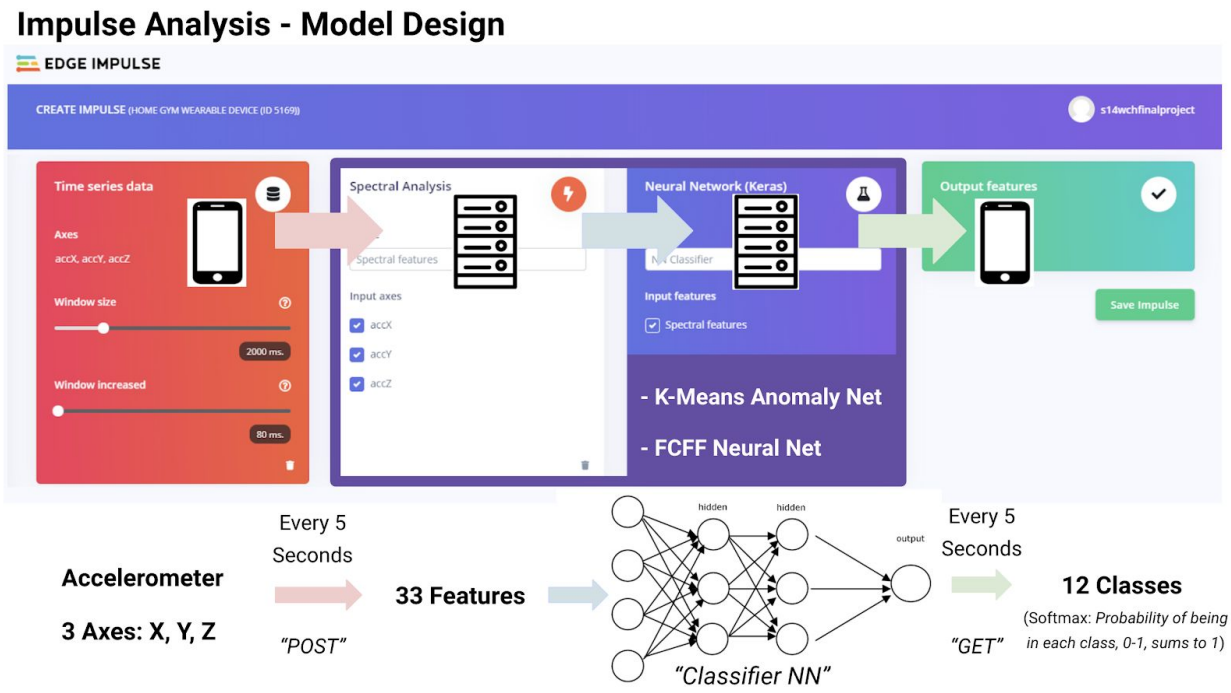


Figure 6: Diagram of Edge Impulse Model Deployment

F) **Classification Model** (Neural Network Classifier)

A keras based fully connected feed forward neural network was chosen as a classifier model, due to its predictive power on large datasets and tunability. Within Edge impulse, raw data was processed and extracted into 11 spectral features for each accelerometer axis. Fully assembled these features created an vector of 33 scalar values as an input to the neural network. It was setup to have two sets of contracting hidden layers, and a softmax output activation. Categorical cross entropy was used as a loss function.

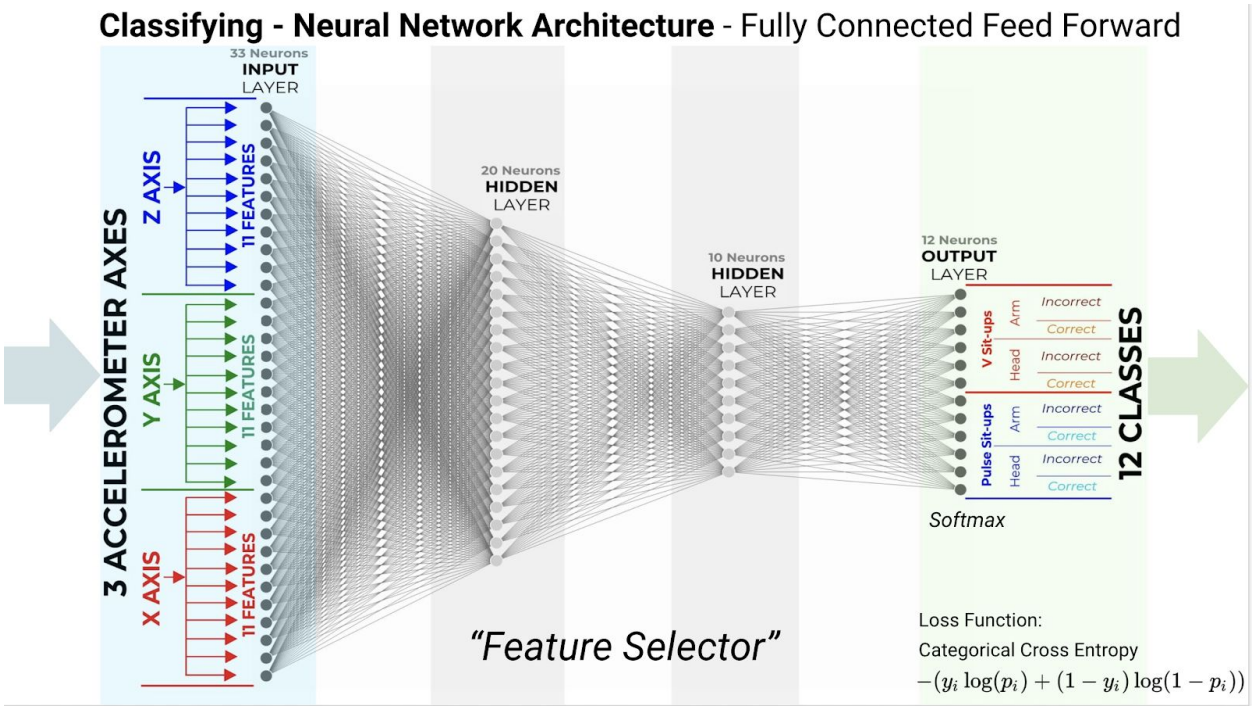


Figure 7: Diagram of Project Fully Connected Feed Forward Neural Network

Because this neural network is very simple (shallow), we can actually draw its complete diagram, seen in figure 7 above. Ultimately, our neural network model acts as a feature selector, automatically up weighting or downweighting the spectral features relative to each output class. Training the network automatically turns off features that are not helpful - highlighting the essential features to classify, and leaving the rest faint.

Prior to classification, K-Means (k-clustering, centroid based) anomaly detection was also implemented to identify and remove non-contributing outliers from the input data.

Please see *Appendix A* for Specific Classification Information

G) Model Performance

Model Performance - Training

ACCURACY 77.1%

Quantized Model

Creating job... OK (ID: 224563)

LOSS 0.83

Copying features from processing blocks...

CLASSES 12

Splitting data into training and validation sets...

Training model...

Training on 53563 inputs, validating on 13391 inputs

Epoch 1/50 - 22s - accuracy: 0.1771 - val_accuracy: 0.2205

Epoch 2/50 - 22s - accuracy: 0.2621 - val_accuracy: 0.2808

.....

Epoch 75/75

1674/1674 - 22s - **accuracy: 0.7843 - val_accuracy: 0.7840**

Converting TensorFlow Lite float32 model...

Converting TensorFlow Lite int8 quantized model...

Confusion matrix

	CLASS_1	CLASS_10	CLASS_11	CLASS_12	CLASS_2	CLASS_3	CLASS_4	CLASS_5	CLASS_6	CLASS_7	CLASS_8	CLASS_9
Class_1	672	4	2	0	0	0	1	2	6	22	1	0
Class_10	0	553	12	2	0	3	14	17	57	15	9	4
Class_11	0	105	486	60	0	0	24	22	6	6	2	7
Class_12	0	34	130	463	0	0	14	19	6	3	3	1
Class_2	9	0	0	0	672	10	25	0	0	0	0	0
Class_3	3	0	0	0	35	718	2	2	0	0	0	0
Class_4	2	25	16	5	21	0	948	10	76	2	2	2
Class_5	2	21	38	4	4	3	41	870	43	43	25	5
Class_6	24	5	9	8	0	0	134	73	874	14	6	9
Class_7	106	4	1	0	0	0	17	16	46	548	0	41
Class_8	4	12	12	10	0	0	27	82	44	19	445	96
Class_9	0	33	10	0	1	1	5	34	79	116	65	348

Figure 8: Model Training and Validation Performance

Model Training Cross Validation performance was 77%. Here you can see the confusion matrix, and the raw model training summary.

Model Performance - Testing

Prediction Accuracy: 37.82% **Class Accuracy: 66.67%**

Actual	Predicted	Sample (Sec)	Anomaly	Correct	Incorrect	Accuracy
Class 1	Class 1	29	18 26.47%	310	68	82.01%
Class 2	Class 2	29	18 8.22%	261	219	54.38%
Class 3	Class 5	29	18 1.61%	0	1115	0.00%
Class 4	Class 4	57	19 12.75%	326	149	68.63%
Class 5	Class 5	57	18 3.51%	424	513	45.25%
Class 6	Class 11	57	18 1.61%	56	1118	4.77%
Class 7	Class 9	29	19 6.19%	225	307	42.29%
Class 8	Class 2	29	19 4.25%	0	447	0.00%
Class 9	Class 9	29	19 4.86%	211	391	35.05%
Class 10	Class 10	57	18 4.58%	165	393	29.57%
Class 11	Class 11	57	17 2.81%	528	604	46.64%
Class 12	Class 12	57	15 2.73%	454	550	45.22%

Exercise Accuracy: 83.33%

	P Situp	V situp
Class 1	378	0
Class 2	480	0
Class 3	569	546
Class 4	462	13
Class 5	818	119
Class 6	451	723
Class 7	3	529
Class 8	447	0
Class 9	36	505
Class 10	53	546
Class 11	94	1038
Class 12	249	755

Testing Confusion Matrix

	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9	Class 10	Class 11	Class 12
Class 1	310	17	1	20	0	30	0	0	0	0	0	0
Class 2	0	261	135	36	34	14	0	0	0	0	0	0
Class 3	1	14	0	83	446	25	17	44	34	73	378	0
Class 4	0	16	0	326	8	112	0	3	6	0	4	0
Class 5	1	105	6	206	424	76	1	75	7	9	19	8
Class 6	1	0	0	67	327	56	57	73	29	149	389	26
Class 7	0	0	0	1	2	0	225	20	270	7	7	0
Class 8	1	242	57	95	8	44	0	0	0	0	0	0
Class 9	0	1	0	1	14	20	36	80	211	166	73	0
Class 10	3	0	1	2	3	44	37	77	153	165	73	0
Class 11	1	0	0	26	31	36	19	8	0	349	528	134
Class 12	0	0	0	0	236	13	18	39	11	33	200	454

Sensor Location Accuracy: 91.67%

	Head	Arm
Class 1	328	50
Class 2	396	84
Class 3	110	1005
Class 4	25	450
Class 5	195	742
Class 6	160	1014
Class 7	328	17
Class 8	300	147
Class 9	328	274
Class 10	271	287
Class 11	28	1104
Class 12	68	936

Figure 9: Model Testing Performance

Predicting what exercise and how it's being done (correct or incorrect) had an accuracy of 66%, which shows room for improvement. However the binary models performed very well - The model that predicts the type of exercise was accurate to 83%, and the model that predicts sensor location was accurate to 92%. Even though the overall prediction accuracy is 38% by raw data, through testing we realized by breaking the problem into a series of steps, we achieve a very high accuracy.

Within the application, classification sensitivity threshold was set at 75% (such that we must be 75% certain in any prediction before assigning that classification). This parameter was adjusted via initial testing, but needs to be experimented with further to find the optimal setting to eliminate false positives and improve the user experience.

H) Future recommendations - Composite model

A divide and conquer approach to models, splitting the models into binary pairs with very high training accuracy. Determining all 12 clusters (or for all exercises, potentially hundreds) simultaneously presents substantial difficulty with class overlap. Splitting the classification process into steps allows class boundaries within root mean square acceleration latent space to be more clearly defined for each step of the model.

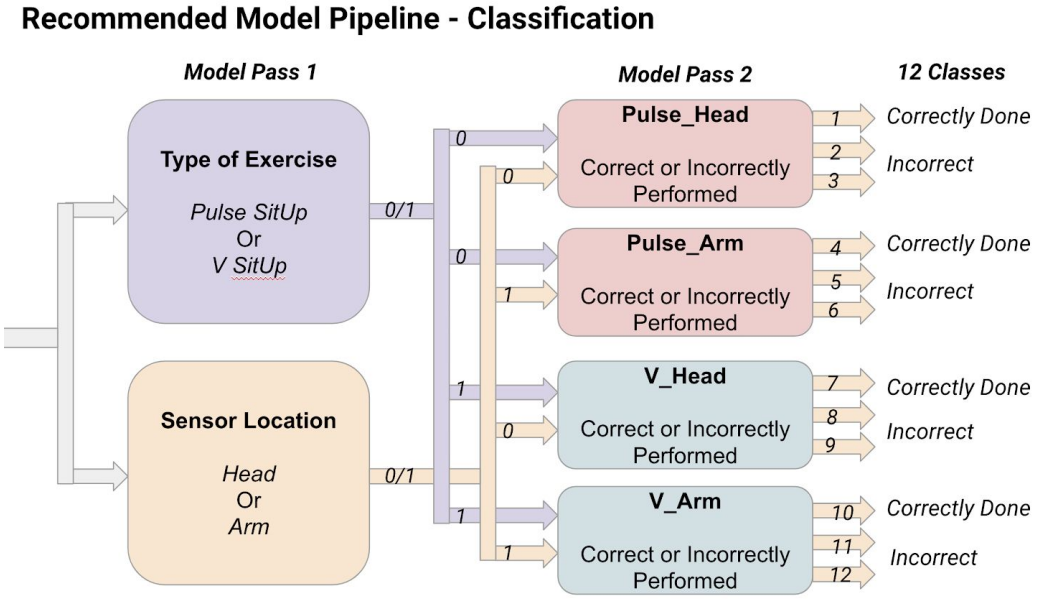


Figure 10: Recommended Improvements for Model Classification "Pipeline"

First a model of which exercise is being performed is run, followed by a model of where the sensor is mounted. These two pieces allow us to "dial into" a model designed for that specific condition - a model designed to determine if that exercise is being done properly. By this procedure, we can more reliably and accurately classify the motion of the user.

But time permitted us from fully testing and implementing this model structure. However for expandability, scalability and modularity, we recommend this model structure be implemented. Training for, and adding classification models for additional exercises would be simplified. Overall model performance would also

8. HSWEAT (Harvard Software Wearable Exercise Application Tool)

This app will help users to define, measure, and summarize their workouts. Users can choose which workout they want to achieve on the home page. Then the app will let them know workout type details, exercise rep counts, total workout duration, and recommended improvement. It features a running timer, and reports exercise durations.

A. Product Design

Key UX Design principles:

Comprehension
Users clearly understand how to use to meet their needs

Transparency
There is no hidden information. We provide useful information at the moment whether you are choosing a workout or reading your workout summary.

Consistency
Build users trust by showing accurate and consistent data-informed design.

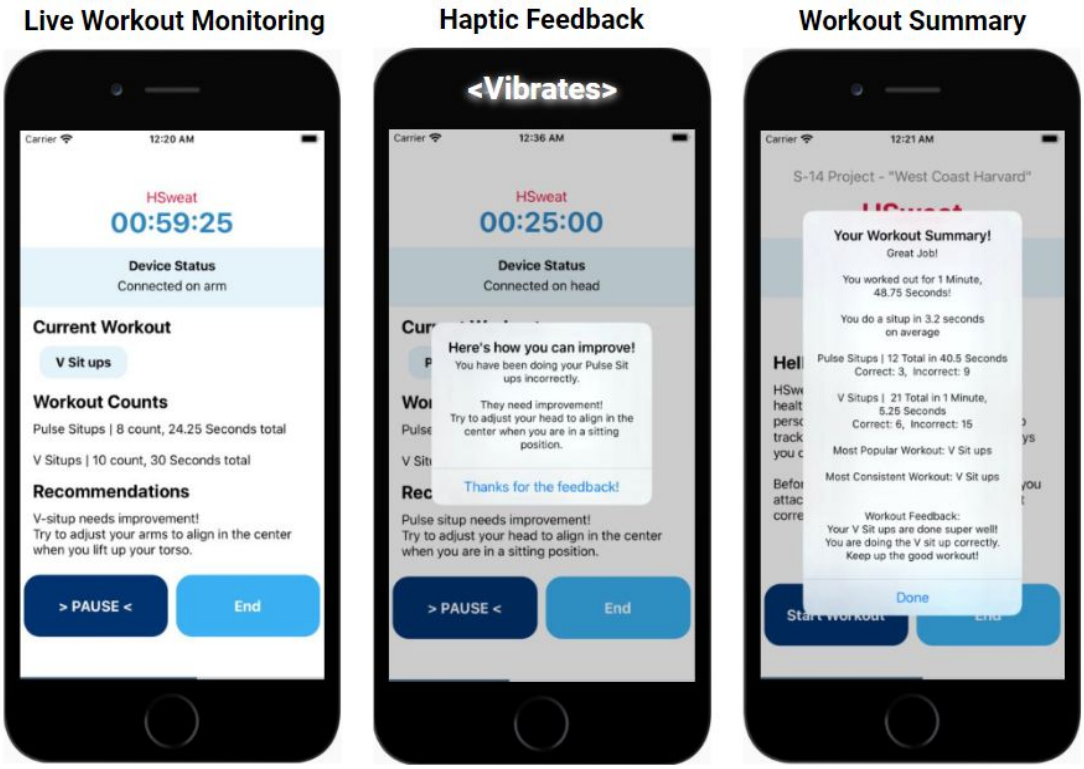
Confirmation
Users feel comfortable to share their data with no privacy issue.

User control
Users know how to control their workout patterns and are able to define areas for improvement.

B. Application - Feature

Our final project deliverable application features a number of functions that support user exercise:

- Browse and choose the best home workout you want to exercise.
- Inform user on benefits and recommendations for each workout.
- Watch and learn the right movement before you start.
- Measure and collect data on how you are doing, how many you have finished correctly, and areas where you need to improve.



- *Figure 11: Screenshots of Project Application Features (User Interface and User Experience)*

Within the user interface, the application informs the user in three key ways:

- *Live Exercise Tracking* - While the user is exercising, the app tracks their motion and analyzes the number of exercises performed (in this case, sit-ups) and whether they are done correctly (with predetermined correct and incorrect patterns). This data is silently collected during exercise, and later made available.

- *Haptic Feedback* - The device will vibrate in a unique pattern if the user consistently (> 5 times) does an exercise wrong, and display improvement recommendations. Since the user will normally be wearing their phone on their arm, this gets their attention and gives them digital trainer feedback.
- *Exercise Summary* - Once the exercise is complete, the user can see a summary of counts of exercises done, average time per rep, total elapsed times, and feedback on how to improve their specific workout.

9. Application - Technology

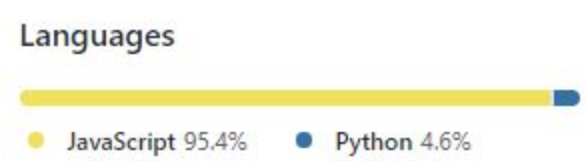
The technology suite deploying the application was selected with a few key criteria:

- to make it as painless as possible for the wearable devices class to access the app. Since the product is not intended to be located in an app store, we strived to make it a one click process to open.
- to support expansion and be as unit based as possible. We strived to make it simple to change out models, classification components or exercise display information
- to make it possible to launch it for beta testing to the general public.

A. React-Native Front End:

The application runs on the snack.expo.io platform, and consists of 1000 lines of javascript code. Access to this platform for any customer is easy. Using a simple QRC code the application loads immediately onto an Android, ro IOS phone or Tablet using the open react-creative platform. Once on the phone the application uses the front end react app to speak with a backend server that contains the modeling software from edge impulse. Ultimately, the goal is a compiled deployment to an app store, which would include user authentication and data hosting; But this was outside of the project scope.

Though the app supports two exercises now, it was built with a modular intent with respect to adding new exercises. All classifications within the app are done via class indexed vectors that make adding new exercises easy. Provided each exercise and exercise characteristic is given a unique class number, the app can rapidly expand



to display new feedback strings, labels and custom display settings.

The app is setup to ingest JSON produced by edge impulse models, but also has provisions for receiving a text file and parsing that text file into classifications.

See *Appendix B* for a technical specification and more details.

B. Server Environment

The server environment is a linux server currently located in Amazon Web Services. This environment runs python, nodejs and flask (web server) in order to handle requests from the react-native application and run the modeling software provided by edgeimpulse.com.

Once the server receives acceleration data from the application is processed through the model software and returns a classification to the app itself.

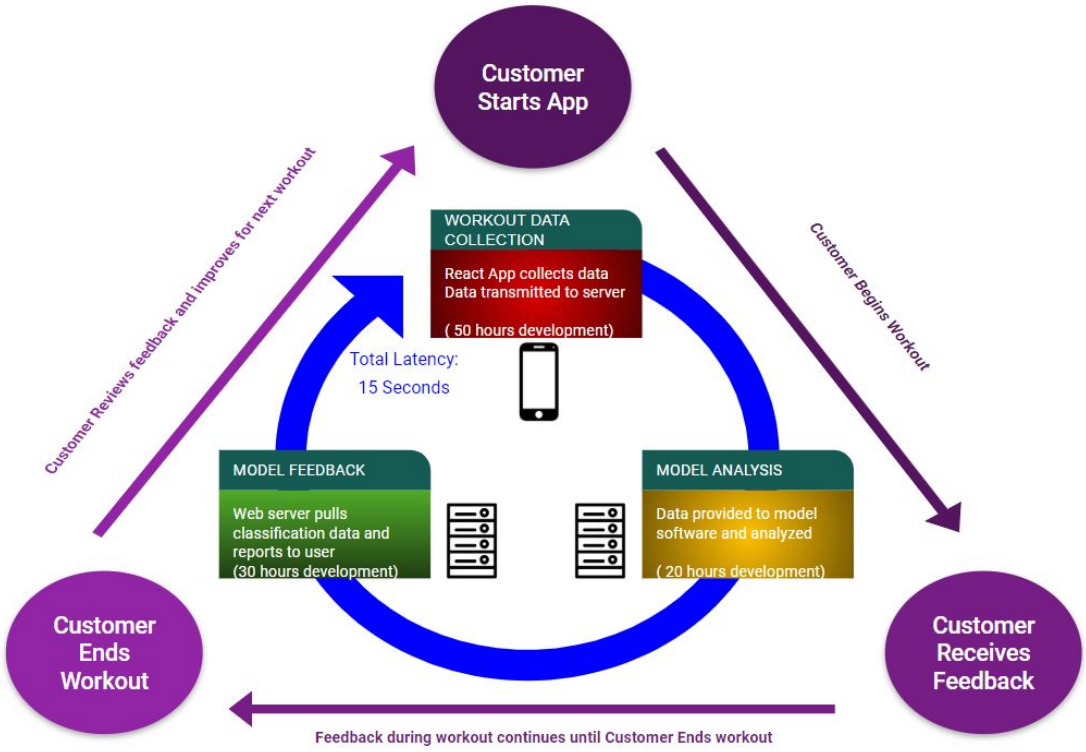


Figure 12: Project Technology Diagram

C. Challenges and Issues

There were several challenges that we ran into in the development and implementation of the technology in this project:

Hardware

- Original design was to use sensor tile with Phone, but multiple BLE issues with the sensor tile prevented us from a more complex implementation and we wanted to prevent using serial cables during collection.

Front End software

- More extensive knowledge of Java Script was needed to truly tweak the React application. Micah, our front end developer, spent countless hours working on making the application work correctly and emulate the designs done by the UX team member.

Edge Impulse

- Edge Impulse runs models in compiled code, and is designed to run on either a physical hardware board, or on a webpage via a "**WebAssembly**" format. This **assembly cannot be installed on a mobile device without considerable difficulty**. It is unclear without a compiled application, downloaded through an app store, if an WebAssembly model can be included in a deployment. Consequently, all Edge Impulse model interactions had to be moved to a server side backend.
- The edge impulse modeling software results did not truly create a correct json table. It created an almost "JSON" table. This ment hours trying to figure out why JSON was "off" and could not be properly sent back from the back end web server to the application. We will be sending Edge Impulse an email detailing our issues so they can fix those issues and hopefully improve their product

D. Back end Software

- No one on the team had extensive experience with Flask, or running a web server

in Amazon Web Services (AWS).

- Flask, the software used to run the web server, required extensive research in order to figure out how we could receive data from the React WebApp and then send the results back to that WebApp.

10. Project Next Steps

Further steps would include constructing a prototype of the headband sensor and integrating it into the system, as well as **deploying a full range of exercise motions** that the app can recognize (approximately 40-50 different types. Hundreds of hours of additional exercise data would need to be collected to train new models for each exercise. The overall neural network classifier can be improved as mentioned in section 8H, splitting the model into a series of binary classifications, to "dial into" the exact exercise being done. The phone based app should ultimately be deployed as a **standalone compiled app in the app store or play store**, and would need all the standard "app furniture" integrated into it, such as user authentication, data hosting, and save states.

11. Contributions by team member

Suzanne Chong

Suzanne Chong **designed the app UX experience and was lead of product design** for the project, created the presentation video, and presentation slide deck. She also collected training data and testing data and came up with 12 classifications. Based on two placements that everyone agreed on, either arm or head, she articulately defined 12 classifications which helped to build edge impulse models. Plus, she spent a lot of time researching which workouts and what kinds of sit-ups would be the most successful to finish the project in scope and given time. After supporting team engineers for collecting data, she designed a full user flow of how users would use the HSweat app usefully. She worked closely with engineers to iterate design to implementation several times.

Micah Nickerson

Micah Nickerson **designed and implemented all front end technology and was lead of data science,** preparing/training edge impulse models (neural networks) for classification. This included coding the application javascript on expo.io, working with UX design to code features for the app, manage continuous integration/continuous deployment of functionality throughout the project, and develop state machines/internal logic within the app. As part of app, he setup the internal timing system that managed the fetch and get, as well as recursive accelerometer data cleaning/classification loops. For project data science, he labeled and analyzed the data, built and trained the core neural network, and implemented a situp counting algorithm. He developed integration of app with backend server, contributed to slide presentation and final report document, and assisted in cutting the presentation video. In total, over 1000 lines of code were successfully deployed as well as a high accuracy classification model.

Nicholas Pesce

Nicholas Pesce **designed and implemented all backend technologies and was lead of technology** involved with the app. This included set up of all backend infrastructure, with AWS, ensuring that the Edge Impulse WebAssembly was properly installed. He was responsible for the Python Flask application, which handles receiving and sending data to the React-native app held on snack.expo.io. His research into Flask sessions allows the whole system to handle multiple independent users at the same time. He also wrote the github documentation (README.md), and co-managed the repository with Micah Nickerson. In addition he also contributed to data collection, slide presentation, video introduction, and authored several sections in the Final Report document.

Lucy Chen Zhang

Lucy **designed the concept** of the home wearable device during the co-vid period which fits customer needs, **implemented the data collection by Edge Impulse model;** cleaned, labeled and tested the data, evaluated the model and gave feedback. Based on previous testing, she transferred the data collection from sensor tile to the edge impulse in order to classify and train the model. She was instrumental in improving the product

by providing significant feedback (data classification model - arm vs head or pulse vs v sit ups) during group discussions, as well as authoring reports for the groups progress. Lucy also smoothed the user flow either in data science as well as in app user experience, do the real case usability testing and shoot the testing procedure as the raw video for product use, the demo video tells the potential client how to use the mobile app and it fulfilled the client-centered interaction needs. She collaborated with team mates to make sure the whole report and procedure followed the logic and present the case in the best way for readers, and constant suggestions on how the product could be improved.

11. References

- Liew, G. Yao, Z. (2017). STMicroelectronics SensorTile Reference Design: Classification of Resistance Training Motion by Machine Learning. Retrieved 7/7/2020 from https://drive.google.com/open?id=1BOEk4zwXd3S4bDC9WduEcDqfpA90bJ3I.
- Staffard, Tiffan CPT (2020). WikiHow.com, How to do Situps. Retrieved from https://www.wikihow.com/Do-Sit-Ups.
- Wikipedia.com (2020). Sit-Up, Retrieved from https://en.wikipedia.org/wiki/Sit-up
- Dixon, Amy (2017). Planks for Beginners; How to do a Plank. Retrieved from https://www.youtube.com/watch?v=ASdvN_XEl_c.
- Wikipedia.com (2020) Plank (exercise, Retrieved from https://en.wikipedia.org/wiki/Plank_(exercise).
- *Martial Arts History, Science, Meditation and Health by Clinet Furr:* Where the head goes the body follows.

Appendix A: Exercise Classification

Class A: Pulse Situp

Class "1": Done Correctly - **Head aligned straight in the center**

Display Message: *"Great job! You are doing the Pulse sit up correctly. Keep up the good workout!"*

ID Label (**One Hot Encoding**): [10000000000000]

Class "2": Done Incorrectly - **Head skewing to the left**

Display Message: *"Wrong situp! Try to adjust your head to align in the center when you are in a sitting position."*

ID Label: [01000000000000]

Class "3": Done Incorrectly - **Head skewing to the right**

Display Message: *"Wrong situp! Try to adjust your head to align in the center when you are in a sitting position."*

ID Label: [00100000000000]

Class "4": Done Correctly - **Arms straight and level, raised to correct height**

Display Message: *"Great job! You are doing the Pulse sit up correctly. Keep up the good workout!"*

ID Label: [00010000000000]

Class "5": Done Incorrectly - **Arms raising too high, not level**

Display Message: *"Wrong situp! Your arms are way too high. Stretch your arms straight when you lift up your torso."*

ID Label: [00001000000000]

Class "6": Done Incorrectly - **Arms not raising high enough, not level**

Display Message: *"Wrong situp! Your arms are way too low. Try to stretch your arms straight when you lift up your torso."*

ID Label: [00000100000000]

Class B: V Situp

Class "7": Done Correctly - **Torso aligned straight to the center**

Display Message: *""Great job! You are doing the V sit up correctly. Keep up the good workout!"*

ID Label: [00000010000000]

Class "8": Done Incorrectly - **Torso skewed to the left**

Display Message: "*Wrong situp! Try to adjust your form to align in the center when you lift up your torso."*

ID Label: [00000001000000]

Class "9": Done Incorrectly - **Torso skewed to the right**

Display Message: "*Wrong situp! Try to adjust your form to align in the center when you lift up your torso."*

ID Label: [00000000100000]

Class "10": Done Correctly - **Arms aligned straight to the center**

Display Message: *Great job! You are doing the V sit up correctly. Keep up the good workout!*

ID Label: [00000000010000]

Class "11": Done Incorrectly - **Arms skewed/aligned to the left**

Display Message: "*Wrong situp! Try to adjust your arms to align in the center when you lift up your torso."*

ID Label: [00000000001000]

Class "12": Done Incorrectly - **Arms skewed/aligned to the right**

Display Message: "*Wrong situp! Try to adjust your arms to align in the center when you lift up your torso."*

ID Label: [0000000000100]

Class C: Technical & Error States

Class "13": Unable to Determine which workout you are doing.

Display Message: *"Umm.... Sorry we're not sure what exercise you're doing - please try again!"*

ID Label: [000000000000010]

Class "14": Null - Blank/No Display.

Intent: Default state for showing a blank screen - Class 14 returns null.

ID Label: [000000000000001]

Class D: Future Exercises

Class "15" -- Class "999"

ID Labels: [0000000000000010000000000000.....000000000000000000]

Slots Reserved

Appendix B: Application Technical Specification

Application Modes:

The app contains two main modes - "**live**" and "**simulation**"**.** These modes are triggered via true/false flags at the header of the program JS code. When the program is running *"live"* (simulations are false), then the app is actively sending, and expecting to receive data to a third party server. When running in *"simulation"* (simulations are true), then the application randomly creates simulated probability return data, and ingests it to simulate the function of the application. The speed of this simulation (rapid_random) can be adjusted to assist in troubleshooting.

Time Intervals based Recursive Processes:

Recursive Interval A: 1/4 Second - X, Y and Z acceleration data is captured from the phone accelerometer and added to a "raw data" vector

Recursive Interval B: 3 Seconds - Data is fetched (GET) from server, and a JSON of predictive classifications received. This JSON is parsed into a vector. If the vector numbers have changed, it is evaluated for the highest probability class, and checked for any ties (in the event of a tie, the first class found is predicted). The probability is then evaluated vs a threshold - are we sufficiently convinced that class is correct? The default threshold is set at 0.75, but can be modified at the header of the script. If the threshold is exceeded, then that class is assigned, otherwise class "13" is returned. At this point, the number of exercises done in that class is incremented, and the elapsed time since, and of the that exercise type is recorded (to calculate intervals).

Recursive Interval C: 6 Seconds - Raw accelerometer data (72 values, consisting of 3 values recorded each ¼ second for 6 seconds) is assembled into a total vector and fetched (POST) to the server to be evaluated. The raw data vector is then cleared for the next round of accelerometer data.

Notes :

- These processes happen and are scheduled independently of one another - the input of one does not directly depend on the output of the other.
- Because of this, stateful variable assignments that are immediately missed in a recursive pass because of mis-timing, will be picked up in the next.

Application States:

"**isExercising**" - *binary flag* - When the user *is exercising*, the elapsed time counter increases, accelerometer data is monitored, data is concatenated into an array, traffic is sent to/from classification, new classifications are displayed, and exercise counts and durations are added to. When the user *is not exercising*, all of the above functions are suspended.

"**Reset**" - *binary flag* - When application is in reset state, home screen is shown, functionality is reduced, and transmissions to/from the server are suspended.

"**_subscribe**" - *binary state*- Indicates that the accelerometer is connected. The app will return all error messages on screen if "_subscribe" is false.

"**situpStreakAlertActive**" - *binary flag* - Indicates that a streak of 5 wrong situps has been found, and vibration/message has been triggered. Exists to prevent any bugs with constant popups, or non-stop vibrating if multiple incorrect exercises are done.

Stateful Variables:

Application contains variables that have a default "error" state, which have new information (new variable states) reassigned to them at each recursive time interval. These as useState() objects, are listed in a block at the top of the script. Key to note, their values are consistently changing, and they must have their value assigned by their own method. Example: exerciseCount is set via setexerciseCount. It cannot have a value assigned to it by normal variable assignment.

Asynchronous Processes:

GET - In recursive interval C, the app async fetches with GET, data from the backend server at *http://ec2-54-162-148-238.compute-1.amazonaws.com:5000/accelerometer.* This asynchronous get does not have an interlock in direct timing to fetch PUT - it will continue to GET the last classification repeatedly even if new data is not transmitted. If server access is terminated during function, GET will default the return classifications to error class 13 (a string of empty return probabilities).

PUT - In recursive interval B, the app async fetches with PUT, data to the backend server at *http://ec2-54-162-148-238.compute-1.amazonaws.com:5000/accelerometer.* This data is an array of concatenated accelerometer values. This asynchronous put does not have an interlock with server function - it will continue to PUT a vector of data even if the web server does not respond.