CA400 Final Year Computer Applications Project

Document Type: Technical Specification

Project Title: Human Activity Recognition using wrist-mounted sensors based on Symbolic Aggregate Approximation and Machine Vision

Student: Shane Creedon

Student ID: 15337356

Supervisor: Tomas Ward

Date of Completion: 17/05/2019

1 Abstract

Human activity recognition (HAR) is an active area of research concerned with the classification of human motion. Cameras are the gold standard used in this area, but they are proven to have scalability and privacy issues. HAR studies have also been conducted with wearable devices consisting of inertial sensors. Perhaps the most common wearable, smart watches, comprising of inertial and optical sensors, allow for scalable, non-obtrusive studies. We are seeking to simplify this wearable approach further by determining if wrist-mounted optical sensing, usually used for heart rate determination, can also provide useful data for relevant activity recognition. If successful, this could eliminate the need for the inertial sensor, and so simplify the technological requirements in wearable HAR.

This project is an attempt to detect greater levels of accuracy among human activity recognition using only wrist-mounted PPG sensors combined with machine vision techniques. The project uses the techniques of **Symbolic Aggregate Approximation (SAX)** and **Bitmap Generation** combined with **convolutional neural networks** to bring about activity recognition. This is a somewhat novel approach to activity recognition and has been the basis of this project in an effort of research for the HAR field.

The project consists of three distinguishable parts:

1. **The Web Application to learn about the project specifics and download the desktop software.**
2. **The Desktop Software which enables activity recognition (Client).**
3. **The Processing Server which enables bitmap generation and activity prediction to clients.**

The website offers the ability to learn about the research that I gained via my blog and also learn about the individual techniques that I applied when developing the project. There is also the ability to review posts by other users in a real-time form page. In addition, the desktop software which contains the prime motivations and integrations of the project requirements, is downloadable via a download button present. The website was built using HTML, CSS, Javascript with Pythons Django web framework on the backend. PostgreSQL has been used on behalf of database storage.

The desktop software offers all of the requirements specified in the functional specification and have further features that extend past the requirements. The desktop software was built using Python and PyQt5 (Python desktop application building library). The Symbolic Aggregate Approximation (SAX) module was one I write and is also present in the client-side software. Additionally, in order to communicate with the processing server, I am using MQTT (Message-Queueing-Telemetry-Transport Protocol) which is a publish/subscribe communication standard among sensors and servers.

The server encapsulates the ability to generation bitmaps and feed such bitmaps into the trained model. We can then use the MQTT network to return the results back to the respective client, that made the activity recognition request.

All of the technical details will be explained more below for each section.

2 Motivation

The primary reason behind wanting to get involved with a project as such as this, was primarily due to my desire to learn about Machine Learning. Prior to my 4th and final year in DCU, I felt my technical skills had improved greatly in almost all areas. However, one area that I had no real prowess in was Machine Learning, Deep Learning, Data Science, all of these fields. Therefore, on behalf of my 4th year project I wanted to get involved in a machine learning concept.

I initially came up with the idea to base a project around Machine Learning Classification of skeletal x-ray images on the hand, wrist and lower arm. There were datasets online that had images referring to different breakages of the hand, wrist or lower arm. However, after speaking to my current supervisor, who at the time was not my supervisor, he suggested I involve myself more with sensors, and so we crafted the idea to perform Human Activity Recognition (HAR) using sensors on the wrist.

With respect to the notion that the project is now complete, I feel I have learned an amazing amount from it. My motivation to understand how Machine Learning technologies work has been realised. The project's course helped also in learning countless techniques and abilities which I can now apply in the future.

3 Research

Among the research that I undertook to facilitate and shape the project, I began understanding what **Symbolic Aggregate Approximation or SAX** was. SAX enables a time-series of data to be converted into a string of semantic characters which hold the meaning of the time-series in what letters the string has. https://www.cs.ucr.edu/~eamonn/SAX.htm.

This technique gave us the ability to produce a string of characters which we could then use to generate bitmaps or images, specifically with the jpeg extension as compression did not matter for our prediction algorithm. To understand this, I did further research on why lossy or lossless images did not matter. It turned out since our images were just a sequence of pixels in a 2D array that were independent in terms of overall image context, that the compressed nature of them did not matter. Therefore, we used jpegs to save space and CPU resources given that jpegs are lossy by nature.

My next and perhaps most fruitful area of research was in relation to the actual machine learning portion of the project. How we would be training a model to effectively and accurately predict what activity a specific individual was performing. Through discussion with my supervisor, I knew this would be no easy feat. More specifically, one of the biggest challenges was the fact that the arduino PPG kit was separate to our training data in terms of how the data was recorded. Additionally, Individual human motion artefacts or movement specifics that each person is characterised by is difficult for a model to accurately converge to. Therefore, it was important to have different human test subjects when it came to the training of the model. However, this led to another challenge we faced, coming from the differences between the PPG datasets online and the actual arduino PPG.

Researching about how to correctly and adequately instantiate a predictive neural network model with respect to the correct amount of training and testing data was also crucial. The model was split in a 80% / 20% among training/testing data. I was also suggested to perform k-fold cross-validation to get more correct predictive accuracy. This technique involves splitting our dataset into k pools where by 1 randomly selected pool acts as the test data and the remaining k-1 pools act as the training data. We repeat this process k times, and average out all of the accuracies produced.

The model was trained initially with images of size 128x128 which evaluated to a predictive accuracy of roughly 74%. This was a brilliant milestone for the project however there was a big problem with this approach. An image of size 128x128 requires precisely 64 seconds of data. This is too far outside of the scope of what we're trying to achieve. Through discussion with my supervisor, we converged on 32x32 sized images, which required only 4 seconds of data to thoroughly make a prediction. Now, with the larger image, the accuracy score was drastically higher but we could not forgo the 64 second data requirement.

To get around this, we used (as briefly previously discussed) transfer learning in the form of Google's Inception v3.0 model. I had to do a lot of require on the topic and had many interactions with members of DCU's Insight Centre that my supervisor introduced me to throughout the life-cycle of the project. I was sent code fragments and Github project repos from said Insight members to review code implementations involving Google's Inception v3.0 model. I then took this knowledge and applied it to my model which I will speak more about in the implementation section of this document.

About half-way through the project, I was suggested by my supervisor to look into MQTT or Message Queueing Telemetry Transport protocol, which is a powerful communication mechanism between sensors and machines (M2M) and is highly popularised in the Internet of Things (IoT) community. Naturally, I engaged in my research of the topic and began drafting plans about how to incorporate it into the project. I came up with the thin-client / fat-server approach, where by the bulk of the processing required is done server-side. This was an important aspect of the system as we didn't want to over-burden a client's system. The ability to convert activity strings into bitmaps as well as the model have its home on the server-side.

MQTT allows clients to communicate with the server using a publish/subscribe architecture. Clients contain the SAX module code, and can generate an activity string from any connected wrist-mounted arduino PPG data or from a previously recorded PPG recording file, in the form of a csv. This activity string is then **published** to a particular **topic** which the server is **subscribed** to. The network transmission is converted to **Base64** prior to any publishing procedure. In this way, the client and server can communicate effectively and efficiently.

The remaining research I performed involved refining any parameters that could potentially improve predictive accuracy or perhaps implementation details about any of the above mentioned methods. I spent some time researching how to adequately build a Django-based Web Application. Additionally, a large devotion of my time went into understanding how to build a desktop application. This was another learning outcome that I wanted to achieve alongside understanding the basis for machine learning algorithms and techniques. I wanted to stay with Python as my core technology for the project, especially given all the machine learning packages Python has acquainted with it. Therefore, I decided to build the desktop application using PyQt5 which ultimately was exactly what I needed for the application's requirements.
https://pypi.org/project/PyQt5/

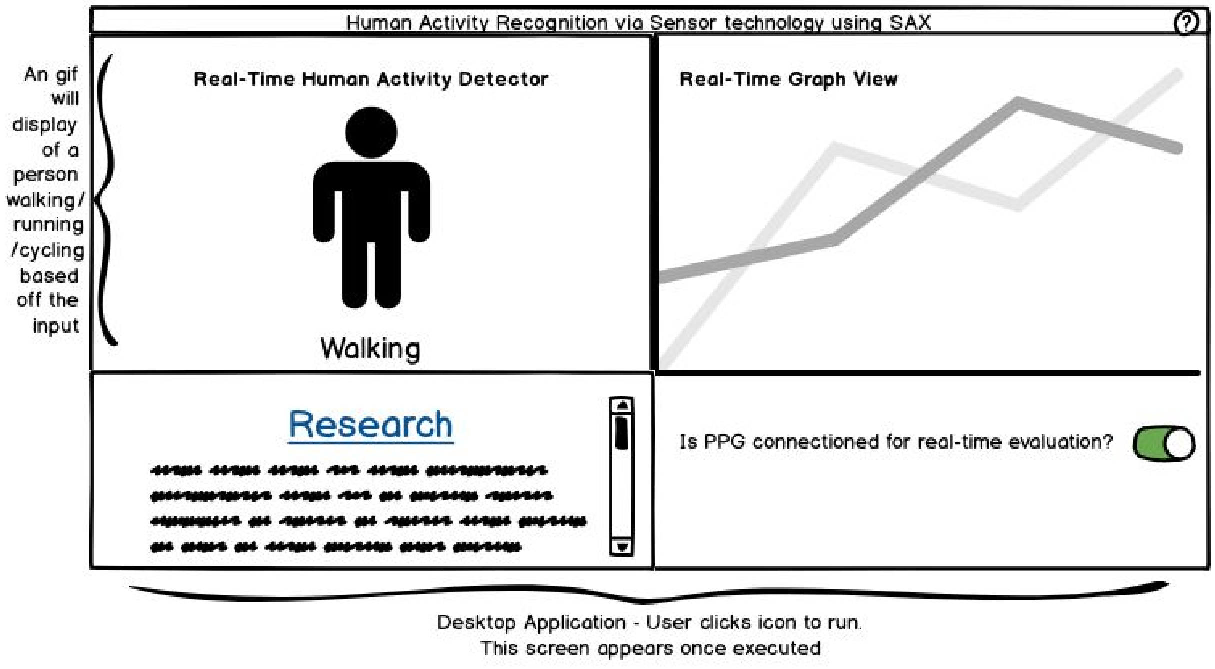
4 Design

Website Design

As previously mentioned, I decided to use Django's Web Framework to explain to people what the project is about and how they can use the project, over a web application. I decided upon Django in terms of Design to stay coherent with the underlying Python technology being used throughout the project as a whole, as well as all of the other advantages of Django such as clean design, security, maintainability etc. The website is now hosted using **Nginx** and **Gunicorn** at https://www.projectactivityrecognition.ml. The design of the website was simple enough to map out internally and had a simple purpose.

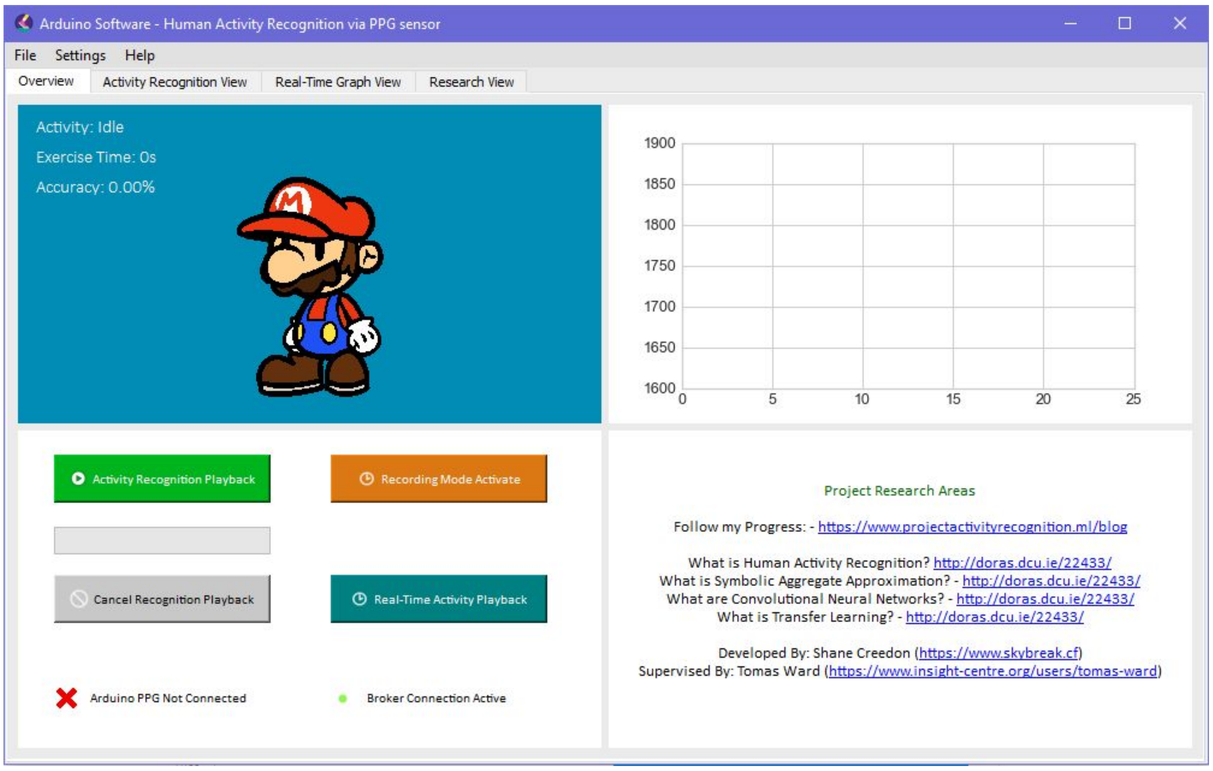
Desktop Application Design

The desktop application design initially began as a single window pane with several sub-panes within. This can be seen below from an **early mock-up diagram** I built.


Human Activity Recognition via Sensor technology using SAX
?
An gif will display of a person walking/ running /cycling based off the input
Real-Time Human Activity Detector
Walking
Real-Time Graph View
Research
Is PPG connectioned for real-time evaluation?
Desktop Application - User clicks icon to run. This screen appears once executed

The design for the most part has stayed the same, however certain aspects of it have certainly evolved. These changes in terms of design can be seen below:

As can be seen in terms of design, it stays relatively faithful to what was initially proposed several months ago. However, notable differences can be drawn. Firstly, the 'controller pane' with all the buttons has been introduced to let the user perform different actions. The window also has several different tabs that can be selected to display a different view. Additionally, there are 3 menu bar options to facilitate usability of the application: **File, Settings and Help.**


Arduino Software - Human Activity Recognition via PPG sensor
File
Settings
Help
Overview
Activity Recognition View
Real-Time Graph View
Research View
1900
1850
1800
1750
1700
1650
1600
0
5
10
15
20
25
Activity Recognition Playback
Recording Mode Activate
Cancel Recognition Playback
Real-Time Activity Playback

Activity: Idle
Exercise Time: 0s
Accuracy: 0.00%

Arduino PPG Not Connected
Broker Connection Active

Project Research Areas

Follow my Progress: - https://www.projectactivityrecognition.ml/blog

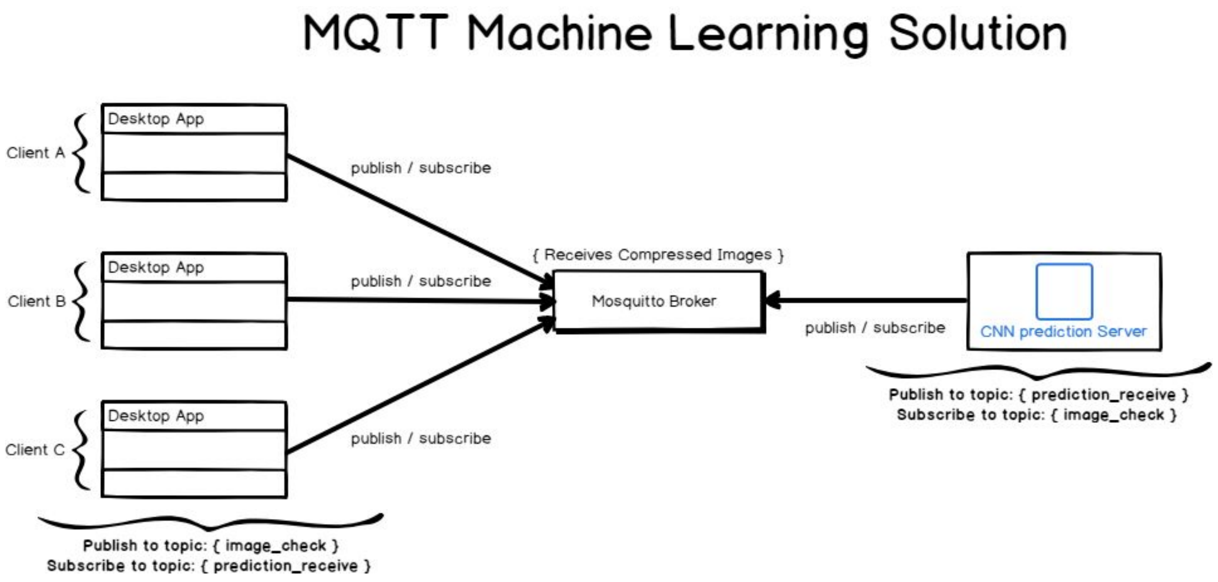
What is Human Activity Recognition? http://doras.dcu.ie/22433/
What is Symbolic Aggregate Approximation? - http://doras.dcu.ie/22433/
What are Convolutional Neural Networks? - http://doras.dcu.ie/22433/
What is Transfer Learning? - http://doras.dcu.ie/22433/

Developed By: Shane Creedon (https://www.skybreak.cf)
Supervised By: Tomas Ward (https://www.insight-centre.org/users/tomas-ward)

In terms of design, I feel the components fit together a lot more nicely and coherently. There is clear visual comprehension among whether or not an arduino PPG device is currently connected and whether or not activity recognition is currently occuring.

MQTT architecture Design

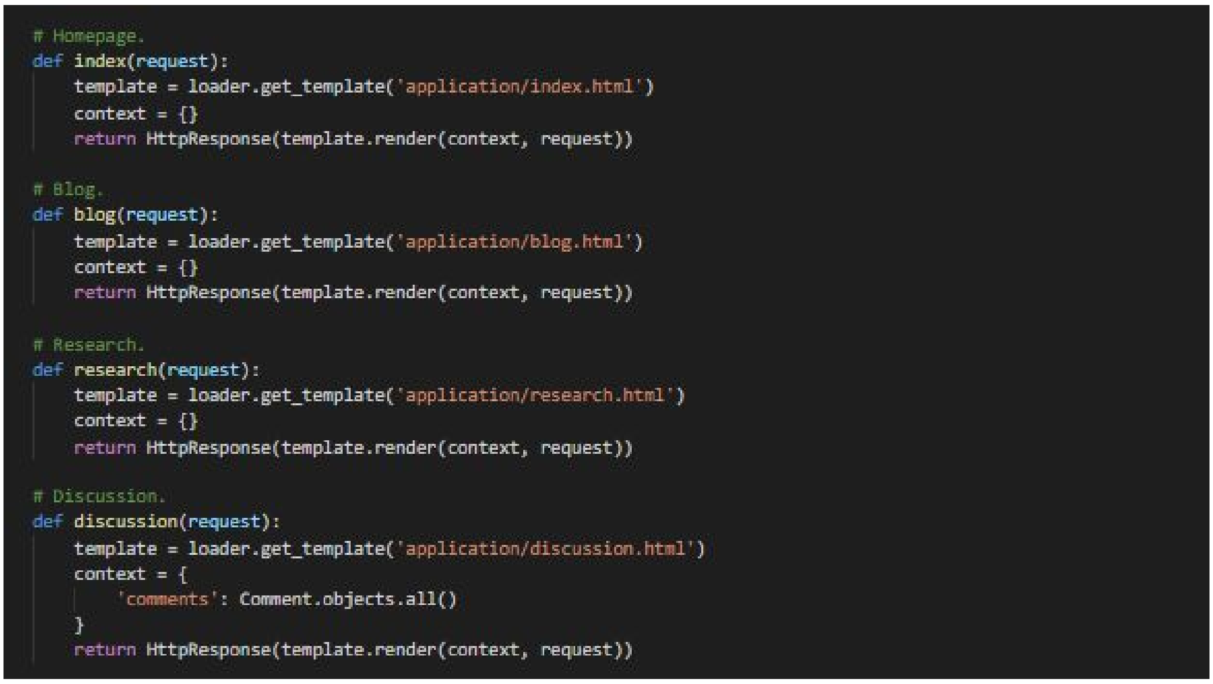
The MQTT architecture design came down to having a thin-client / fat-server architecture where by the server does the majority of the processing. This design makes sense from an efficiency standpoint as we don't want to burden our clients with CPU intensity or memory problems. The server would be located in the cloud perhaps via IaaS functionality.


MQTT Machine Learning Solution
Client A
Desktop App
Client B
Desktop App
Client C
Desktop App
publish / subscribe
publish / subscribe
publish / subscribe
{ Receives Compressed Images }
Mosquitto Broker
publish / subscribe
CNN prediction Server
Publish to topic: { prediction_receive }
Subscribe to topic: { image_check }
Publish to topic: { image_check }
Subscribe to topic: { prediction_receive }

5 Implementation

Website Implementation

As aforementioned, the website was built using the Django web framework, which involves **views** as to how django find and loads relevant pages for the client.



```python
# Homepage.
def index(request):
    template = loader.get_template('application/index.html')
    context = {}
    return HttpResponse(template.render(context, request))

# Blog.
def blog(request):
    template = loader.get_template('application/blog.html')
    context = {}
    return HttpResponse(template.render(context, request))

# Research.
def research(request):
    template = loader.get_template('application/research.html')
    context = {}
    return HttpResponse(template.render(context, request))

# Discussion.
def discussion(request):
    template = loader.get_template('application/discussion.html')
    context = {
        'comments': Comment.objects.all()
    }
    return HttpResponse(template.render(context, request))
```

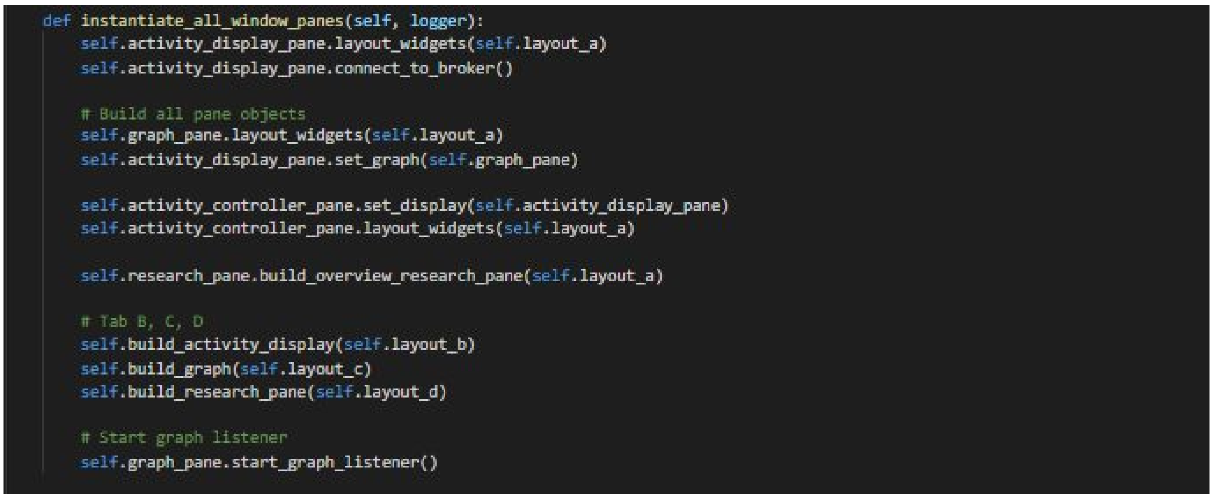
As can be seen above, the implementation of individual views is quite simple. Templates are rendered based on requests and contexts can be applied based on the view in question, such as **discussion**. Each individual template is built using HTML and CSS. One of the main goals of the website was to ensure responsivity. I achieved this using flexible design such as using **percentages(%)** in combination with **media queries**. The new and ambitious **CSS Grids** was also used to aid in the design.

Desktop Implementation

The desktop application was built using PyQt5. The desktop application is split into 4 different 'views' or 'perspectives' as it were. These 4 being:

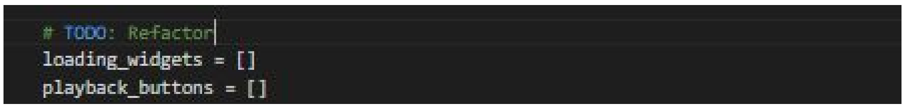
1. Overview
2. Display / Activity Recognition View
3. Graph View
4. Research View

The overview is a summarised version of the 4 views with less detail. The code for which can be seen below.



```python
def instantiate_all_window_panes(self, logger):
    self.activity_display_pane.layout_widgets(self.layout_a)
    self.activity_display_pane.connect_to_broker()

    # Build all pane objects
    self.graph_pane.layout_widgets(self.layout_a)
    self.activity_display_pane.set_graph(self.graph_pane)

    self.activity_controller_pane.set_display(self.activity_display_pane)
    self.activity_controller_pane.layout_widgets(self.layout_a)

    self.research_pane.build_overview_research_pane(self.layout_a)

    # Tab B, C, D
    self.build_activity_display(self.layout_b)
    self.build_graph(self.layout_c)
    self.build_research_pane(self.layout_d)

    # Start graph listener
    self.graph_pane.start_graph_listener()
```

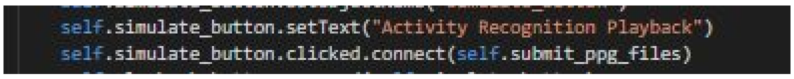
I tried to ascertain a certain degree of clean code smells throughout the implementation of the project with comments reflecting certain components. I also added comments throughout to reflect areas that needed re-work or refactoring. I did this using the **TODO** comment notion in Python. As can be seen below:



```python
# TODO: Refactor
loading_widgets = []
playback_buttons = []
```

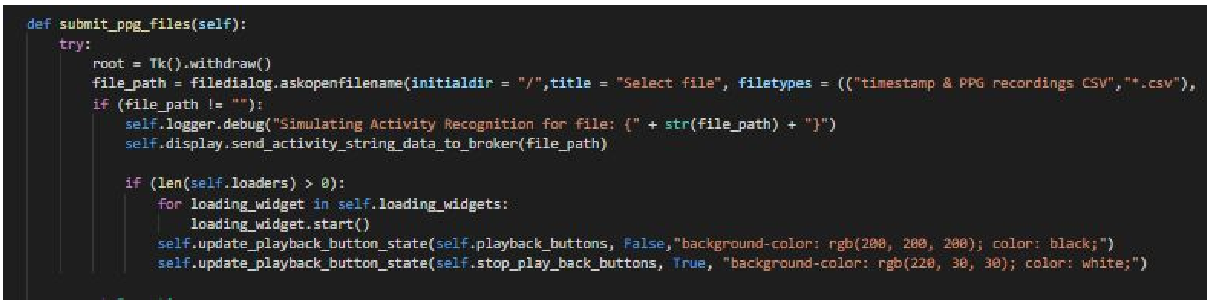
I wanted to compartmentalise each individual window component that the client would be interacting with and this was simply done using unique classes for each. The buttons found in the **controller** pane are connected directly to the functionality found in the display window pane and the graph window pane. The research window pane is entirely separate/independent. For example: When the **Activity Recognition Playback** function is pressed/clicked on, this will send a message to the **display** component which will spin up a thread which will encapsulate the logic behind converting the time-series data into an activity string using SAX. This string is converted to Base64 and subsequently **published** using the MQTT network protocol to the server. The server is listening to that particular topic and can accept the request and perform the relevant requirement. In this case, start a feedback loop to that client where by the string sent over is broken up into windows, and each window is individually fed into the neural network, and the classification is returned to the client. Code for this will be shown below:

Button clicked - Calls function self.submit_ppg_files



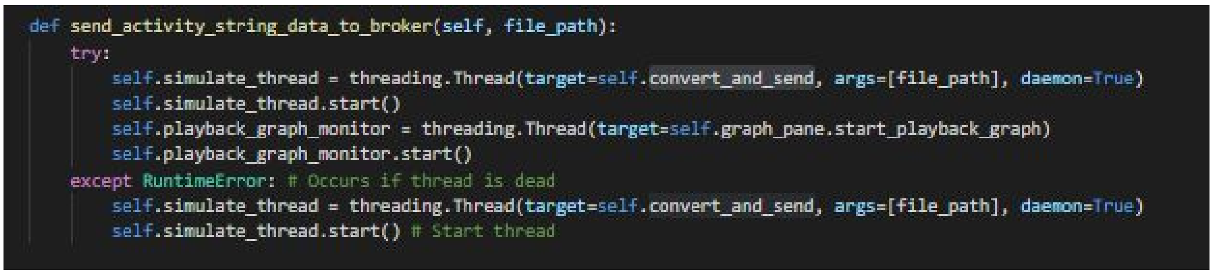
```python
self.simulate_button.setText("Activity Recognition Playback")
self.simulate_button.clicked.connect(self.submit_ppg_files)
```

Relevant function that is called - Users can select the csv file containing the PPG recordings and upload to the model. The display component is required to be interacted with for this to work.



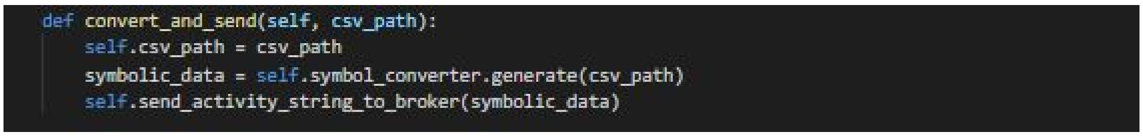
```python
def submit_ppg_files(self):
    try:
        root = Tk().withdraw()
        file_path = filedialog.askopenfilename(initialdir = "/",title = "Select file", filetypes = (("timestamp & PPG recordings CSV","*.csv"),
        if (file_path != ""):
            self.logger.debug("Simulating Activity Recognition for file: {" + str(file_path) + "}")
            self.display.send_activity_string_data_to_broker(file_path)

            if (len(self.loaders) > 0):
                for loading_widget in self.loading_widgets:
                    loading_widget.start()
                self.update_playback_button_state(self.playback_buttons, False,"background-color: rgb(200, 200, 200); color: black;")
                self.update_playback_button_state(self.stop_play_back_buttons, True, "background-color: rgb(220, 30, 30); color: white;")
```

Spin up threads of execution.



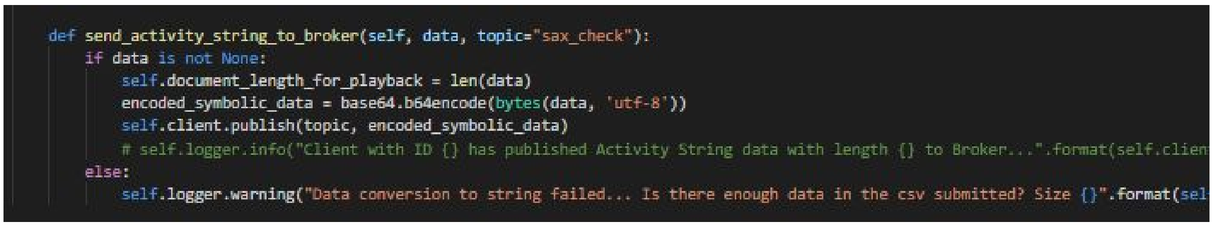
```python
def send_activity_string_data_to_broker(self, file_path):
    try:
        self.simulate_thread = threading.Thread(target=self.convert_and_send, args=[file_path], daemon=True)
        self.simulate_thread.start()
        self.playback_graph_monitor = threading.Thread(target=self.graph_pane.start_playback_graph)
        self.playback_graph_monitor.start()
    except RuntimeError: # Occurs if thread is dead
        self.simulate_thread = threading.Thread(target=self.convert_and_send, args=[file_path], daemon=True)
        self.simulate_thread.start() # Start thread
```

Display Component - Create Activity String using SAX and call publishing function.



```python
def convert_and_send(self, csv_path):
    self.csv_path = csv_path
    symbolic_data = self.symbol_converter.generate(csv_path)
    self.send_activity_string_to_broker(symbolic_data)
```

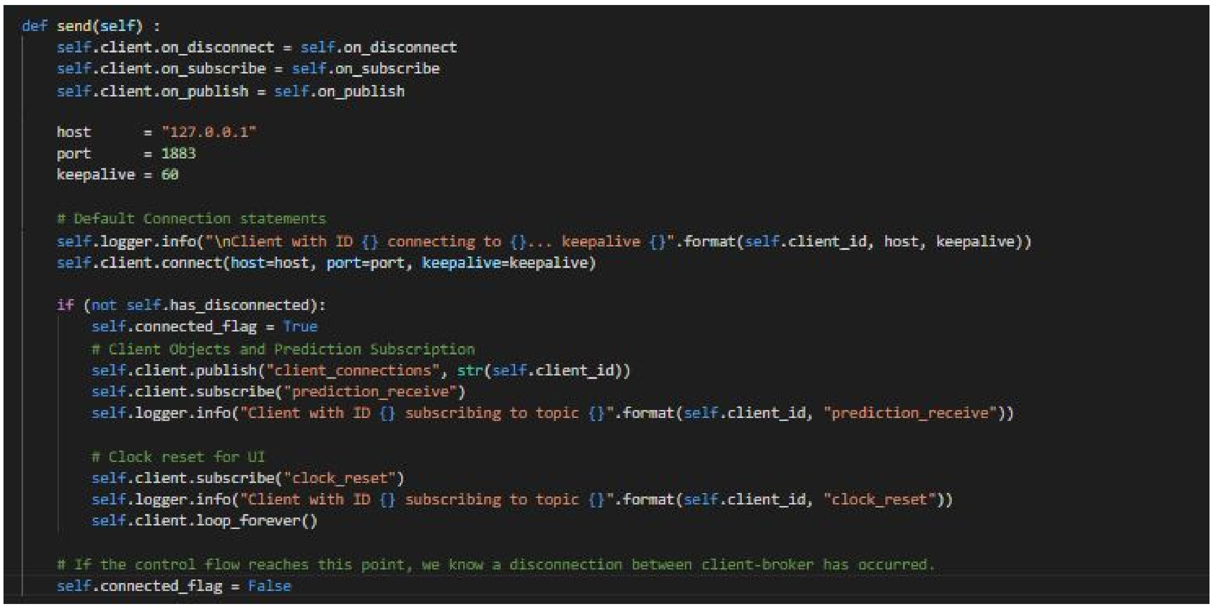
Convert data to base64 and publish it over the MQTT network on a particular topic.



```python
def send_activity_string_to_broker(self, data, topic="sax_check"):
    if data is not None:
        self.document_length_for_playback = len(data)
        encoded_symbolic_data = base64.b64encode(bytes(data, 'utf-8'))
        self.client.publish(topic, encoded_symbolic_data)
        # self.logger.info("Client with ID {} has published Activity String data with length {} to Broker...".format(self.clien
    else:
        self.logger.warning("Data conversion to string failed... Is there enough data in the csv submitted? Size {}".format(sel
```

MQTT Implementation

MQTT was able to be utilised thanks to the available Paho Library: https://pypi.org/project/paho-mqtt/. This library offered a series of functions to allow each publish/subscribe implementation. The difficult part was initialising a **broker.** The broker is the basically the infrastructure that sits between relevant clients and servers. This can be seen above in the MQTT architecture diagram. Specifically **Mosquitto** https://mosquitto.org/ was used to do this.



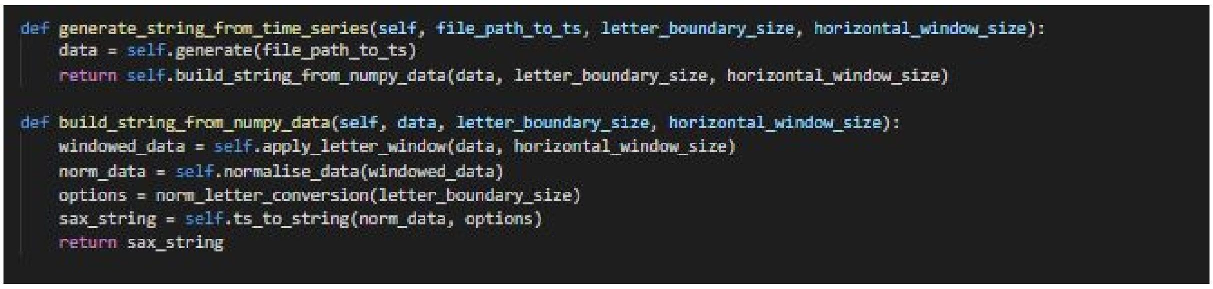
```python
def send(self) :
    self.client.on_disconnect = self.on_disconnect
    self.client.on_subscribe = self.on_subscribe
    self.client.on_publish = self.on_publish

    host      = "127.0.0.1"
    port      = 1883
    keepalive = 60

    # Default Connection statements
    self.logger.info("\nClient with ID {} connecting to {}... keepalive {}".format(self.client_id, host, keepalive))
    self.client.connect(host=host, port=port, keepalive=keepalive)

    if (not self.has_disconnected):
        self.connected_flag = True
        # Client Objects and Prediction Subscription
        self.client.publish("client_connections", str(self.client_id))
        self.client.subscribe("prediction_receive")
        self.logger.info("Client with ID {} subscribing to topic {}".format(self.client_id, "prediction_receive"))

        # Clock reset for UI
        self.client.subscribe("clock_reset")
        self.logger.info("Client with ID {} subscribing to topic {}".format(self.client_id, "clock_reset"))
        self.client.loop_forever()

    # If the control flow reaches this point, we know a disconnection between client-broker has occurred.
    self.connected_flag = False
```

The code snippet above shows the basics of the MQTT protocol in action. It requires a relevant host id with a port number associated with it. The application is set up such that once a client opens the desktop application, they are automatically subscribed and connected to the broker at hand, given it is available.

As can be seen above, clients subscribe to a topic beknownst as **prediction_receive** which is a topic where all relevant predictions for that particular client will pass through. The client publishes a message to the **client_connections** topic such that the server knows when a client has connected and can add them to a hashmap/dictionary of client_ids. This data structure is crucial for keeping track of which clients are currently connected.

Symbolic Aggregate Approximation Implementation

Initially, I found a library online specifically for Python that enabled SAX application. However, due to certain library characteristics combined with my supervisor suggesting to me that I should build my own SAX module, I did just that. I built my own SAX library from scratch involving a series of steps in relation to conversion between vector representation and string representation.



```
def generate_string_from_time_series(self, file_path_to_ts, letter_boundary_size, horizontal_window_size):
    data = self.generate(file_path_to_ts)
    return self.build_string_from_numpy_data(data, letter_boundary_size, horizontal_window_size)

def build_string_from_numpy_data(self, data, letter_boundary_size, horizontal_window_size):
    windowed_data = self.apply_letter_window(data, horizontal_window_size)
    norm_data = self.normalise_data(windowed_data)
    options = norm_letter_conversion(letter_boundary_size)
    sax_string = self.ts_to_string(norm_data, options)
    return sax_string
```

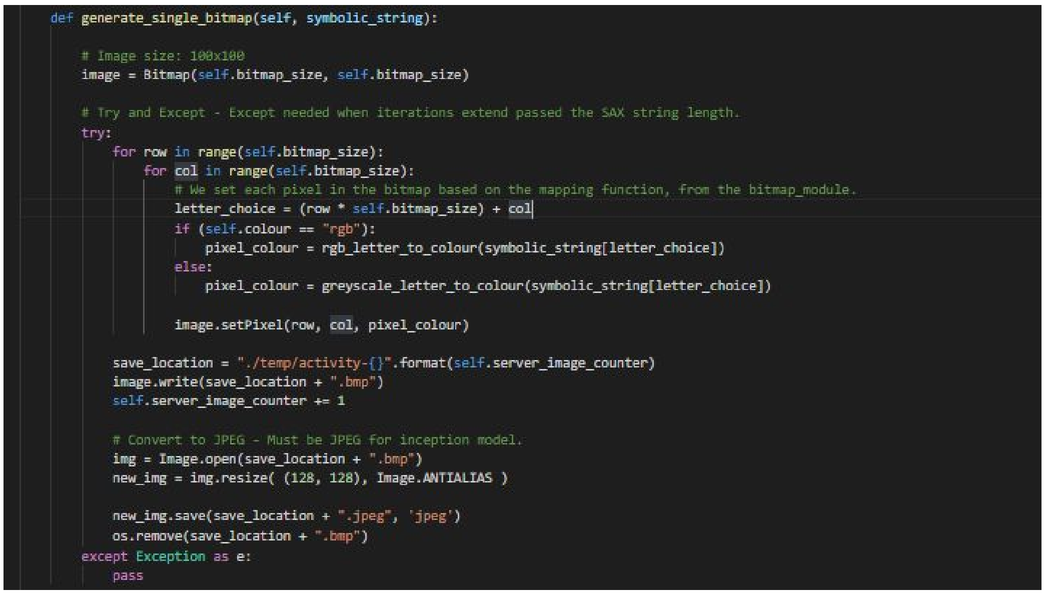
The sequence of steps involved in the SAX process can be seen above. The steps are:

1. Convert Time-Series dataset to numpy array
2. Apply a letter window - This step allows a horizontal window sized to be placed across all data points and then averaging these values to get an overall result. Therefore, as a natural result of the method, the amount of available time-series data decreases, but the generalisation towards which activity it is might improve. So, there is a trade-off here in terms of effectiveness and the project explored that.
3. Apply normalisation to the data.
4. Convert the normalised data to an activity string
5. Return the string

Bitmap Generation Implementation:

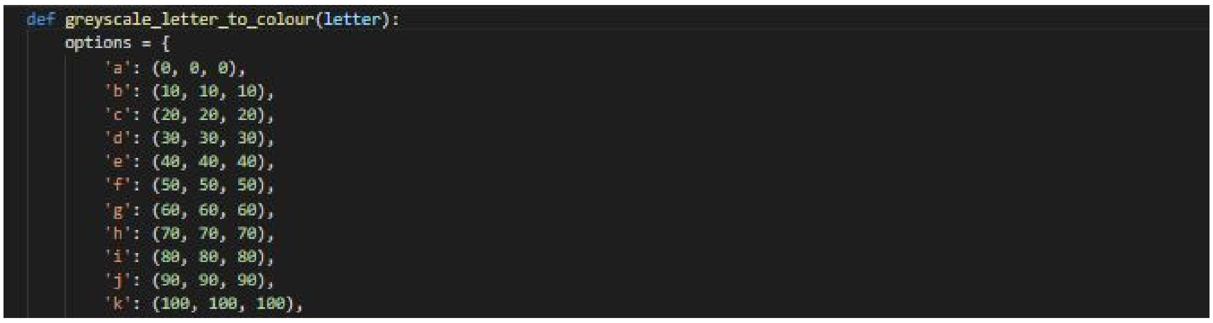
I make use of the activity string generated from Symbolic Aggregate Approximation to build a set of images of size 32x32. The reason for this sizing was explained previously but the primary reason is that images of 32x32 only require 4 seconds of time-series data.

The bitmap generation is quite intrinsic to the whole process and has a large dependency on how strong the predictive outcome of the model is. Many factors are involved such as image size, the amount of images, the colour of the images, whether the data in the images is normalised, etc. This project places an emphasis on varying these parameters to try achieve the highest possible model accuracy.



```python
def generate_single_bitmap(self, symbolic_string):

    # Image size: 100x100
    image = Bitmap(self.bitmap_size, self.bitmap_size)

    # Try and Except - Except needed when iterations extend passed the SAX string length.
    try:
        for row in range(self.bitmap_size):
            for col in range(self.bitmap_size):
                # We set each pixel in the bitmap based on the mapping function, from the bitmap_module.
                letter_choice = (row * self.bitmap_size) + col
                if (self.colour == "rgb"):
                    pixel_colour = rgb_letter_to_colour(symbolic_string[letter_choice])
                else:
                    pixel_colour = greyscale_letter_to_colour(symbolic_string[letter_choice])

                image.setPixel(row, col, pixel_colour)

        save_location = "./temp/activity-{}".format(self.server_image_counter)
        image.write(save_location + ".bmp")
        self.server_image_counter += 1

        # Convert to JPEG - Must be JPEG for inception model.
        img = Image.open(save_location + ".bmp")
        new_img = img.resize( (128, 128), Image.ANTIALIAS )

        new_img.save(save_location + ".jpeg", 'jpeg')
        os.remove(save_location + ".bmp")
    except Exception as e:
        pass
```

The above code snippet shows the basic creation of a Bitmap object. Each bitmap object is set to a particular size, in our case 32x32. Every pixel in that bitmap is initially set to 0, however through the method shown above, each pixel is individually set to the correct colour based off the SAX activity string. How do I achieve this?



```python
def greyscale_letter_to_colour(letter):
    options = {
        'a': (0, 0, 0),
        'b': (10, 10, 10),
        'c': (20, 20, 20),
        'd': (30, 30, 30),
        'e': (40, 40, 40),
        'f': (50, 50, 50),
        'g': (60, 60, 60),
        'h': (70, 70, 70),
        'i': (80, 80, 80),
        'j': (90, 90, 90),
        'k': (100, 100, 100),
```

I have an organised mapping for each letter to a particular 3-value tuple representing the colour of that letter in greyscale. I also have another variant of the above function for RGB colours, but grey-scale is generally prefered in the machine learning community.

Model Implementation and Transfer Learning

I use Google's Inception V3 model (https://cloud.google.com/tpu/docs/inception-v3-advanced) to craft the predictive model. Every image that was previously generated from my bitmap module is fed into the V3 model and the penultimate layer of the model is retrained. This yields high accuracy results in most cases.

6 Testing

Testing is a crucial element to any software project, and this project is no different. To ensure the necessary quality and robustness that every project strives for, I made strong use of several popular testing techniques. These techniques are discussed below:

Continuous Testing via Continuous Integration

For my 3rd year project, we did not make use of Continuous Testing or Continuous Integration. I knew that was a mistake and I wanted to place an emphasis on it this year. I looked into several of the popular CI automated testing tools like TravisCI, Jenkins, Teamcity but I eventually settled with Gitlab's CI/CD platform due to the simplicity of the process (Editing a .yml file) and ensuring an available 'runner' is available for automation. Gitlab was already being used for the project repository so it was an easy choice for me at the inception of the project.

Throughout the life-cycle of the project, I was continuously aware of the state of all the builds of my project but particularly the most recent one. Gitlab makes it extremely easy to view why and how a particular build may have failed. In addition, I was also notified when any of my builds failed, which I found extremely helpful when testing my system.

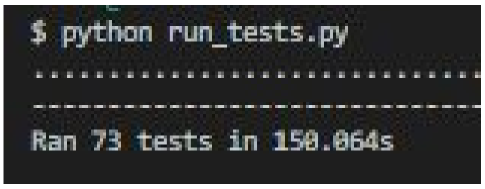
For each new commit I push to the repo, a new build would run and all my test cases would evaluate. This form form regression testing was particularly useful for me to locate any newly introduced bugs that may have broken previously working functionality.

I was focused on adapting elements from the very so popular 'Test-Driven Development' such that I would write initially failing tests prior to the implementation of a particular unit and then modify those tests to pass with the minimal amount of necessary code. I found this particular method very engaging in regards to testing my software. It pushed testing to the forefront of my mind and added robustness and quality to my software.

Unit Testing

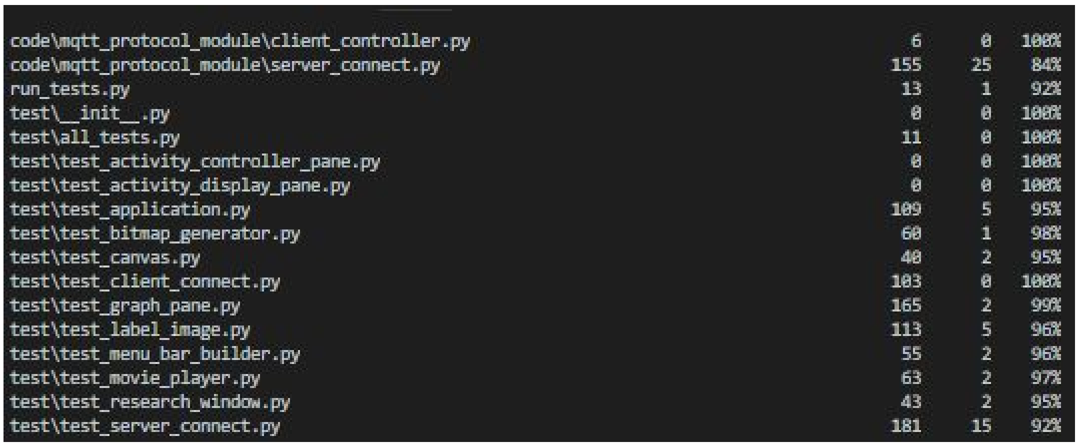
Unit testing was the least arbitrary mode of testing I engaged in with my project. For each method of my desktop application I had a unit test that was written in reference to it. In total, I had over **70** unit tests written for my project suite. Often, I would have to spin up threads to emphasis the time-domain of the unit such that perhaps a certain feature only activated after X seconds.

Figure - Over 70 unit tests



```
$ python run_tests.py
..............................
------------------------------
Ran 73 tests in 150.064s
```

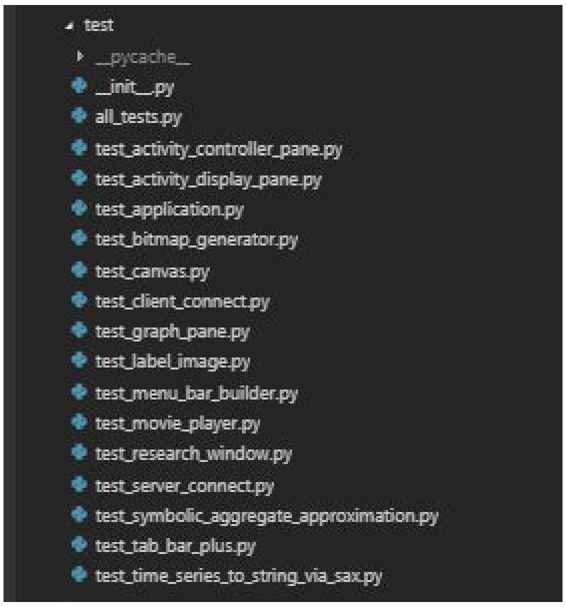
Fig - Coverage of the system



```
code\mqtt_protocol_module\client_controller.py                          6     0   100%
code\mqtt_protocol_module\server_connect.py                           155    25    84%
run_tests.py                                                           13     1    92%
test\__init__.py                                                        0     0   100%
test\all_tests.py                                                      11     0   100%
test\test_activity_controller_pane.py                                   0     0   100%
test\test_activity_display_pane.py                                      0     0   100%
test\test_application.py                                              109     5    95%
test\test_bitmap_generator.py                                          60     1    98%
test\test_canvas.py                                                    40     2    95%
test\test_client_connect.py                                           103     0   100%
test\test_graph_pane.py                                               165     2    99%
test\test_label_image.py                                              113     5    96%
test\test_menu_bar_builder.py                                          55     2    96%
test\test_movie_player.py                                              63     2    97%
test\test_research_window.py                                           43     2    95%
test\test_server_connect.py                                           181    15    92%
```

The coverage report above shows how I emphasized my unit tests. Given the above 3 columns on the right hand side refer to: **Total Statements in File, Total Missed Statements, Overall Coverage Percentage(%)**, it is clear my unit tests cover a large percentage of possibilities and statements within the desktop application software components.

I performed unit testing manually via simple regression testing and also via Gitlabs automated testing through CI pushes. To perform such tests, I initially used Pythons **Unittest** package. However, to receive better feedback from my tests and understand to what degree they cover my system, I downloaded the **nose** package, through which I was able to inspect my coverage in different areas and apply changes to those areas when necessary.



```
test
  __pycache__
  __init__.py
  all_tests.py
  test_activity_controller_pane.py
  test_activity_display_pane.py
  test_application.py
  test_bitmap_generator.py
  test_canvas.py
  test_client_connect.py
  test_graph_pane.py
  test_label_image.py
  test_menu_bar_builder.py
  test_movie_player.py
  test_research_window.py
  test_server_connect.py
  test_symbolic_aggregate_approximation.py
  test_tab_bar_plus.py
  test_time_series_to_string_via_sax.py
```

I have each component set in its own class and each component is made up of a series of unit tests.

Integration Testing

I wrote several integration tests to evaluate the connection between two particular components of my software. In particular, I wrote tests to evaluate the conjunction between the tabbed nature of the desktop application combined with the widgets present on each of those tabs.

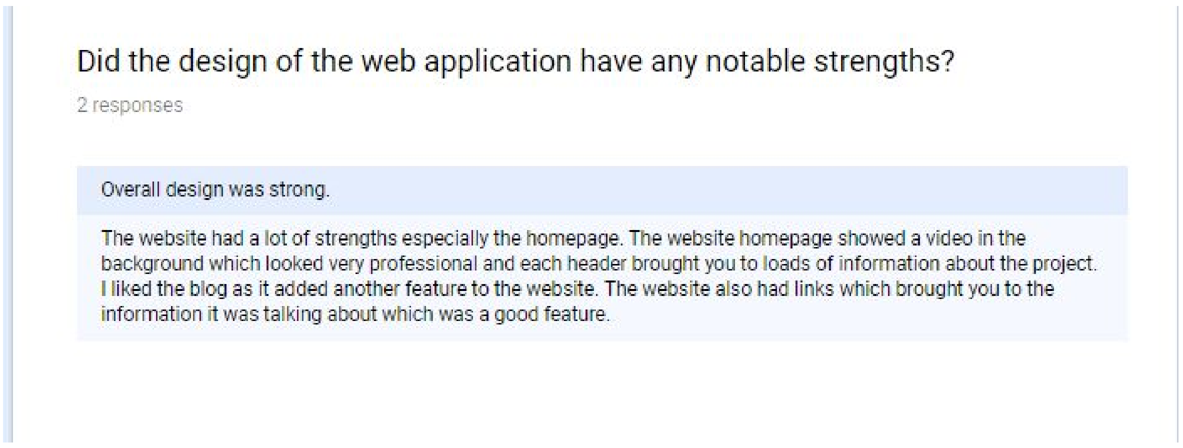
User Testing

In relation to the ethics form I submitted several months ago, I performed user testing by means of two distinct surveys. One based on the Web Application and the other based on the Desktop application. Questions in relation to the accessibility, usability, quality and accuracy of the applications were posed and answered.

I managed to perform to surveying around DCU campus with regards to the majority of the people I aimed to achieve. I managed to survey 7 people around DCU campus with respect to their view of the application. Their micro-voltage data was recorded and compared against other users in terms of activity recognition.

The remaining 3 people I surveyed were friends and family whom I instructed to offer a reasonably justified and fair opinion based on their external view of the application. Both surveys for my user testing can be found in **docs/ethics**

Fig - An example of a response from an anonymised individual:



Did the design of the web application have any notable strengths?

2 responses

Overall design was strong.

The website had a lot of strengths especially the homepage. The website homepage showed a video in the background which looked very professional and each header brought you to loads of information about the project. I liked the blog as it added another feature to the website. The website also had links which brought you to the information it was talking about which was a good feature.

7 Results

The project achieved its goal of being capable of detecting activity recognition. The degree to which that activity was recognised correctly is adequate in the majority of cases, however my particular application struggles with recognising biking due to the notable similarities between Low Resistance Bike and High Resistance Bike.

In addition to the mostly successful affirmation of activity recognition, the dedicated architecture the project was molded into is something I am very proud of. To utilise the MQTT pub/sub distributed communication protocol was a very interesting and enjoyable problem to solve.

As compared to the human activity recognition approach adopted my DCU (train on graph images), this model falls short. Perhaps in reference to how I implemented it but more likely due to the nature of the SAX process. DCU achieved an accuracy of roughly 88% while my model is probably hovering at around 60%. With more data, more time and a greater understanding (Which I now have), I feel such an accuracy can be improved.

The primary problems I faced throughout the development of this project revolved around achieving a higher model accuracy. With the limited data I had, it was a tough procedure, and eventually converged at around

60% as spoken above. The development of the desktop and web application also had problems but they were minor in nature and were quickly refined and solved.

Machine Learning is a very interesting field of expertise and I am happy I embarked on this journey.