Ubiquitous Computing Final Project Report

Win Barua // qnk19zxu // 100277378

1. Aims, motivation and background

A real-world problem that many people have is when trying to lead a healthy lifestyle, most people find it very helpful to know how many calories they have burned during a specific workout, as well as having a system that could automatically log what activity they have done. The aim of the system is to take sensor data from user's phone and take a 10 second video of their fingertip with the flash on, from this data the system will recognize what activity they were doing and calculate their heart rate in beats per minute(BPM), then ask some questions about the user, combining these answers with the BPM it will calculate and output the calories burned and the activity they were performing. This splits the system in two parts:

Activity Recognition – I have created an android application to collect as sensor data on each type of activity that I will be detecting and train a Convolutional Neural Network to recognise this data. I did this using Jupyter Notebook to create, test and visualise the model as well as save it in order to import and use it in the main application. These are the activities I will be detecting: Resting, Walking, Jogging, Push-ups, Stair-climber machine, Burpees

BPM Calculation – This takes the user's video and applies some image processing in order to get a graph of the user's pulse and find the peaks in order to calculate the BPM.

For the purposes of this project, I will keep data collection separate from the other two steps, however data processing and feedback/output will work in tandem, this means when using the system, I will be collecting my own sensor data and video and sending it to the main application which will run the BPM calculator on the video and the Neural Network model on the sensor data in order to give the appropriate output. The system could later be applied to a complete standalone application which could analyse live sensor data as well as process the user's fingertip pulse in-app and give the user their feedback, linking data collection, processing and feedback together in a more user-friendly application.

2. Sensing

The sensors needed for this system are the accelerometer on an android phone and the camera sensors with the flash on. The accelerometer allows me to gather the x, y, z values of the phone's acceleration during different movements, therefore different activities will have different levels of activity from the sensor, i.e., "resting" will be mostly flat values around 0 whereas "jogging" will have much higher acceleration values. I will train the model to recognise the patterns from these numbers and what activity they relate to. When first developing the machine learning model I tried to also use gyroscope data however I was unsuccessful in making an accurate enough model as I could only get up to about 55-60% accuracy which would not be enough for the purposes of this project. Trying it out only using accelerometer gave me a much better accuracy of about 80-85%, I did not figure out why this occurred, maybe because of a discrepancy between the two sets of data

or an issue with the construction of the model itself but after doing some research I did not find any helpful resources on this issue. I did not add any barometer data as none of the activities involved changing levels of air pressure, the stair climber activity is a machine which has similar movements to going upstairs however the user is not actually going up and therefore the barometer isn't needed.

The camera sensors are being used to record a video to feed to the BPM calculator module, the video with the flash on will show changes in the colour or intensity which relate to the user's pulse, when blood is pumped to the fingertip, it will go to a darker colour. This concept can be used to plot an oscillating graph similar to a cardiogram, more detail on the methods used will be given in the data processing section. The main advantage of using this sensor is the fact that it does not require any specific hardware apart from the inbuilt camera of my phone and it doesn't need me to produce an android application for this specific purpose, as opposed to using the accelerometer sensor for which I had to create an application on Android Studio to collect and save data to a text file.

3. Data Collection

In order to carry out data collection I have built an android application on Android Studio using Java, it has start and stop buttons to log the activity, it will create a text file with all the relevant sensor data and the label for the activity. There are 6 activities: Resting, Walking, Jogging, Push-ups, Stair-climber, Burpees, for each activity I have collected a total of 10 minutes of data trying to keep it as consistent as possible and making sure all my actions are deliberate, without making and erratic or random movements that could cause issues during the training process, they were collected 2 minutes at a time. For certain activities where I cannot do them for 2 minute at a time I did them 30 seconds at a time and collated the data together into one file, these activities were push ups and burpees. The main reason for choosing 10 minutes of data is because from my research, most activity recognition systems only take use a specific set of activities and push ups, burpees and stair climbers are not included so I suspected that I might have trouble training the model specifically for these activities since I would not have any resource to rely on to find out the best practices or methods specific to those. Furthermore, an activity such as burpees requires a combination of a lot of movement and building a model to understand all of that would be difficult if I did not have a lot of data. After collecting all the separate data, I have manually put everything in one file that would have a sequence of all the sensor data along with the label it corresponds to.

Manual collation of the data should not be a problem as it is easy to do with this data and is only required once at the start of the model training process, when using the end product data only needs to be collected for the activity they are doing on that occasion. The Neural Network is designed and trained on Jupyter Notebook and is saved to the project directory in order to be used by the main driver application.

The **advantage** of collecting data in this way is that it allows me to dictate exactly how I want the data to come out using code since training a machine learning model requires very specific data processing and it is helpful if the raw data is as close as possible to how the data needs to look in order to go into the model. This is why having control over how the data comes out is very important as using any prebuilt app will mean I will have to adjust to how they chose to extract the data. However, the main **disadvantage** is that it requires me to build an app just for this purpose which takes a bit of extra time and effort.

The other piece of data that needed to be collected was the video from the phone's camera with the flash on, this is a simple .mp4 file of 10 seconds of length that should be collected right after or during the exercise if possible, for example, a jogging, resting or walking activity would allow to collect the video while in the middle of it however for the sake of consistency I have collected all the camera data straight after doing the activity, with the fingertip firmly pressed up and covering both the camera and the flash. The reason for only collecting 10 seconds of data is because doing image processing on videos can put quite a load on a computer, it can take a while to extract the video frames and analyse each image, a full application would make use of more powerful computers or servers and could therefore use more lengthy data to give a slightly better accuracy.

The main **strength** of collecting the data straight from the camera is that it does not need any special software, it is also easy to manage the data and the video files and organise them within the project structure. However, a **weakness** I encountered is problems with the file format where it would save it as a .mov file instead of .mp4, which is something I could not change, although this might have been a problem specific to my phone, it did still slow down the development process as converting the videos does take some time.

4. Data Processing and Interpretation

The system has two data processing modules, one for the BPM calculation from the video and one for the training of the machine learning model which will recognise the activity that was performed.

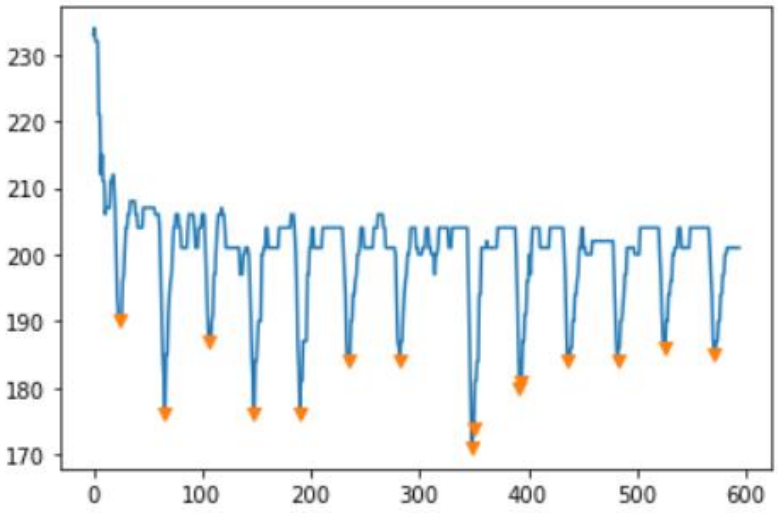
4.1 BPM Calculation

As stated earlier, the idea is to calculate the beats per minute by monitoring the changes in colour of the fingertip when a pulse happens to draw a graph with all the troughs being one pulse. I am using the term trough instead of peak because the colour's value will be going down, and the lowest point per "cycle" is what the program registers as one beat. I will now detail each stage of the process from getting the video to outputting the BPM.

The first stage is to extract each frame from the video using a package called OpenCV also referred to as cv2, it will go through each frame of the video and save it as an image in a specific folder within the project directory, one other method I have tried was using the Pillow Image library however a big problem with this method is that it was very slow when implemented in my frame extractor method and by a significant amount, the tests I have done show that using Pillow took 1084.68 seconds to extract 595 frames whereas using OpenCV only took 19.99 seconds, most 10 second videos I took came out to about 500 to 600 frames.

After extracting the frames, I needed to go through the directory of frames and store each frame as a 2D array in a list of frames, each single entry representing the RBG values of one pixel, this process also took some time, about 15-20 seconds. This gives me a way to essentially store the entire video in a 3D array within my code. However, the RBG colour space was not great for the purposes of monitoring colour intensity changes as depending on the colour all 3 values could change and this meant it wouldn't work when trying to plot the "cardiogram". One of the most used colour models in image processing is the HSV colour space which stands for Hue, Saturation and Value; this was perfect for my use case as the type of colour change that was occurring only changed the Value index of the HSV pixel, meaning I could extract the values of every frame at one specific pixel into a list and plot it onto a graph to show the pulse troughs.

At this point, we have an array which stores the HSV Value amount of a specific pixel in every frame, one thing I noticed is that the video would usually start off with a few frames where it would go very dark before being a more stable which is why I also decided to take out the first 10 indexes of the HSV Values array. From this array I was able to draw a cardiograph which showed the pulses during those 10 seconds, then using the SciPy find_peaks() function I was able to get the list of troughs and draw it onto the graph:



When using the find_peaks() function it will detect every single peak in the graph so I added a rule where it would only acknowledge the peaks that are in the bottom third of the y axis range, this turned out to be a fairly accurate way of detecting the peaks that are significant, usually with an accuracy of ±2. From here I could simply count the number of pulses in these frames and multiply it by 6 to get the BPM.

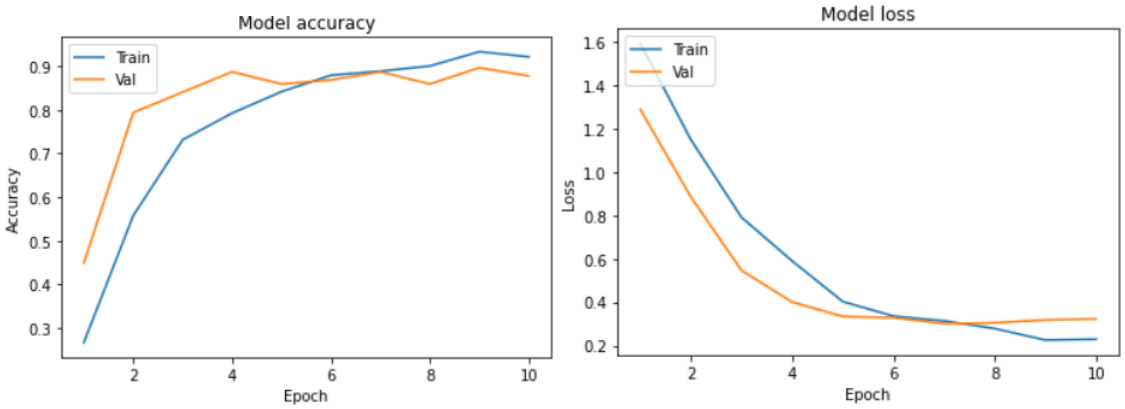
4.2 Activity Recognition

For this aspect of the system, I had to construct a Neural Network model using TensorFlow, the raw input file has 4 columns, 3 of them are the features (accelerometer x, y, z) and a label column, this file puts every separate data collection session in a sequence.

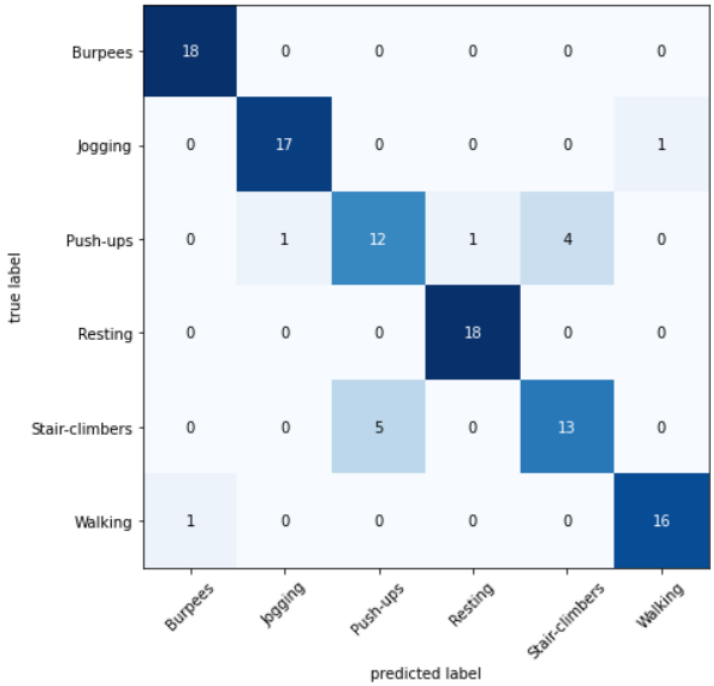
The first step of data processing was to balance the data so that every activity would have the same number of entries and the model would not be heavily biased towards certain activities, I used the Pandas library to give me a count for each label and altered the data to have the same number of entries for each activity as the activity with the least amount of data. I also proceeded to use the Label Encoder to convert all the labels to one-hot-encoding which stores each label as an array of zeros and ones and the corresponding label depends on the index at which the 1 is positioned.

The next stage is to apply feature scaling to the data by standardizing the data which is a type of feature scaling that scales the data uniformly around a mean of 0 with a unit standard deviation. Following this I needed to prepare how my data would be inputted into my model to be trained, sending in one line at a time would not work since each activity is the result of the combination of many lines of data so training the model with single lines would be inaccurate. I chose to send in a sample of the data by sectioning each 80 lines (4 seconds of data at 20Hz) and assigning the most common label to that sample before sending it into the model.

The final step was to separate the data into training and testing data and then create the Neural Network model using a combination of different layers, starting off with a Con2D layer to handle the 2D array going in and using a Flatten layers to input a flat version of the data to a Dense layer, then outputting the weights for the classes. These are the accuracy and loss value graphs given by the model:



As you can see there is some overfitting happening where the training accuracy goes above the actual accuracy(val_accuracy) but overall this is a very well performing model especially for the purposes of building this system.



This is the confusion matrix given by the model, and the model performs mostly as I expected, very accurate for the simpler actions such as resting and walking and it struggles a little bit with stair climber and push ups which are activities that are not often included in these types of projects. However, it did surprisingly well for recognising burpees which I thought the model would struggle with single it involves different types of movements in different directions, on the other hand this could also be a good thing as it makes this activity have very distinct patterns compared to the other ones as it has very fast movements in specific directions which could help the accuracy of the model.

5. System Output and Feedback

The main module of the system combines both the calculate BPM module and activity recognition model and provides more abstraction as it could technically be given to a user along with all the module and a simple instruction file and they could run the main python file from their command line, passing in arguments for the fingertip video file and the sensor data they have gathered and the program will do the rest giving the user constant feedback the current step and ending with the result, with the total calories burned (calculated from BPM) and the activity performed.

This module also does the task of preparing the data to be inputted into the model, which is a similar process to when training the model however it doesn't need any labels or need to be split into training and test data or X and y (features and labels respectively). It also plays the part of displaying questions to the user when the program is finished calculating the BPM in order to calculate the total calories burned, it does this using a simple formula:

```
WOMEN: CB = T * (0.4472*H - 0.1263*W + 0.074*A - 20.4022) / 4.184

MEN:  CB = T * (0.6309*H + 0.1988*W + 0.2017*A - 55.0969) / 4.184
```

Where:

- `CB` is the numbers of calories burned
- `T` is the duration of the exercise
- `H` is the heart rate (BPM)
- `W` is the weight in kilograms
- `A` is the Age

So the program will display a series of questions including their gender, each question will be validated so the questions get asked again if they enter an invalid value and they are not just given an error which wouldn't be user friendly, using the inputs to these questions a function will calculate and return the calories that have been burned.

```
(python390) PS C:\Users\winba\Desktop\UbiqFinal> python .\display_activityandcalories.py .\data\test_vids\Jogging1.mp4
'.\data\test_data\23,04,22,07,23,44,Jogging,accelerometer.txt'
Extracting frames...
Done: extracted 639 frames; Took 24.48 seconds

Analysing frames to calculate BPM...
Done: BPM is  156

Activity recognition started...
Done: Activity = Jogging

Enter your gender (0 = male; 1 = female):  0
Exercise duration in minutes:  30
Weight in kg:  58
Age:  22
You have burned 425 calories

(python390) PS C:\Users\winba\Desktop\UbiqFinal>
```

This screenshot shows how the application will be running if I run it from the command line, it shows each step of the process clearly and shows the results of each step, finally asking the relevant questions to the user so it can calculate the total calories burned. The top line shows how the script will be ran, from the directory of the project folder, the script will be launched with 2 arguments, the first one with the fingertip video and the second with the path of the sensor data, both paths can be relative or absolute.

6. Conclusion

Overall, I think the project would be considered successful in regard to the Aims section, it does very well in outputting the correct activity for the given sensor data, apart from the small issues it has with the push-ups and stair-climber activities as discussed in section 4.2. Also the BPM calculator is very accurate and displays a good value of the user's heart rate and the program does a good job in dealing with different types of input from the user during the questions phase. And the system does well to abstract the complexity of the backend processes by having one module for the user to run using two arguments from which the program will import the modules and function it needs without the intervention of the user or them having to understand how the core of the system works, a simple instruction would be sufficient.

Another strength of the system is that the activity recognition model was able to be constructed with only accelerometer data which significantly increases the efficiency of the system as it has to process less data at a time which could also make the training and prediction times quicker. Also making the development of the data collection app more streamlined since I only had to worry about one sensor for this part of the system, although I did try using the accelerometer with the gyroscope but explained that it did not perform well.

One weakness of the system is that the BPM calculation is very much an estimation as this is not using specialised hardware and is using image processing to get the pulses which cannot have 100% accuracy, most downloadable heart monitor apps which also use the camera and flash have a disclaimer to tell users that the output will not be an exact measure. Having said this, the end-user would normally understand that calorie tracking can never be a completely accurate process because so many variables are involved, which is why this is not a big issue but it is worth mentioning.

A second weakness the system has is the fact that it does take a while to run and complete, especially the calculating BPM module since it's processing images, usually taking about 30 to 40 seconds for the entire thing to finish before getting to the questions phase. This would be something that could be improved given better computer hardware as most distributed applications run their calculations on powerful dedicated computers or servers. Another approach could also be taken where the heart rate calculation is done live using the webcam to analyse visual cues on the face showing a pulse, however this idea would make it very hard to effectively implement the activity recognition aspect alongside heart rate.

APPENDIX

Appendix A - Testing

Time tests when analysing images

100 images – using Pillow

```
# get_image() time test with 100 frames, most real cases will have about 500 frames.
images = []
start = time.time()

print('Getting Images...')
for i in range(100):
    images.append(get_image('frames/frame%d.jpg' % i))

end = time.time()
delta = end - start

print('Done, took %d seconds' % delta)
```

```
Getting Images...
Done, took 49 seconds
```

100 images – using OpenCV

```
# time test with 100 frames, this time using cv2.imread() most real cases will have about 500 frames.
images = []
start = time.time()

print('Getting Images...')
for i in range(100):
    images.append(cv2.imread('frames/frame%d.jpg' % i, 1))

end = time.time()
delta = end - start

print('Done, took %d seconds' % delta)
```

```
Getting Images...
Done, took 0 seconds
```

595 images – using Pillow

```
start = time.time()
imgs = create_frames_list('frames2', 595)
end = time.time()
delta = end - start

print('Took %.2f seconds' % delta)
```

```
Creating frames matrix...
Done
Took 1084.68 seconds
```

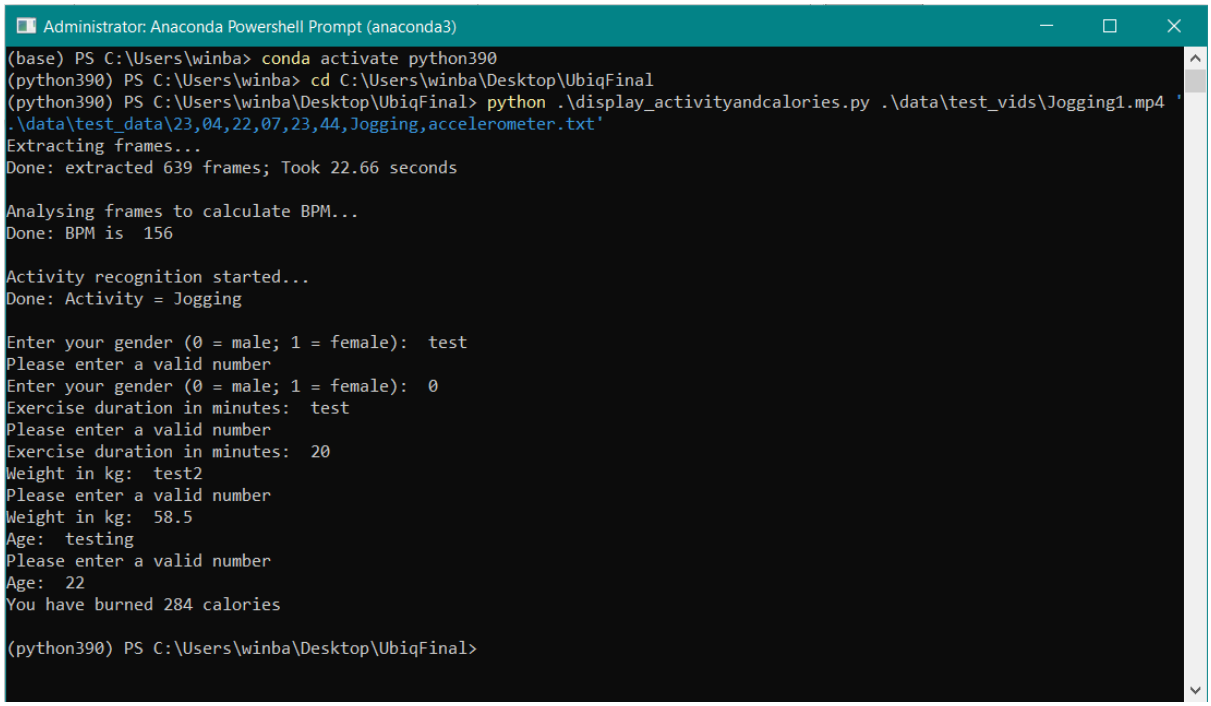
595 images – using OpenCV

```
start = time.time()
imgs = get_frames_arr('frames2', 595)
end = time.time()
delta = end - start

print('Took %.2f seconds' % delta)
```

```
Getting images...
Done
Took 19.99 seconds
```

Test of the finished system, with demonstration of input validation



Appendix B – Code Snippets

```
# Previous function used to get and store image pixels in list, took a very
long time, has been replaced with optimized alternative
# Takes a path to the image that needs to be stored
# Returns image as matrix, get pixels with pixel_values[x][y]
def get_image(image_path):
    """Get a numpy array of an image so that one can access
values[x][y]."""
    image = Image.open(image_path, "r")
    width, height = image.size
    pixel_values = list(image.getdata())
    if image.mode == "RGB":
        channels = 3
    elif image.mode == "L":
        channels = 1
    else:
        print("Unknown mode: %s" % image.mode)
        return None
    pixel_values = np.array(pixel_values).reshape((width, height,
channels))
    return pixel values
```

Appendix C - References

1 - Calories Burned by Heart Rate Calculator (omnicalculator.com)

2 - Image Processing with Python (datacarpentry.org)

3 - Image Manipulation — The Hitchhiker's Guide to Python (python-guide.org)

4 - Extract images from video in Python - GeeksforGeeks

5 - Feature Scaling | Standardization Vs Normalization (analyticsvidhya.com)

6 - https://kgptalkie.com/