

Biomedical Engineering Laboratory Final Project

Human identification using 1-lead ECG signal
final report

Professor: 蔡德明

Student:

Dept. EE

0811562 何祁恩

Problem to solve

The electrocardiography (ECG) signal has been proven as a unique signal for a person (See reference [1]). Similar to fingerprints and retinas. However, the fingerprints and retinas can be easily faked or stolen from the dead body, the ECG signal can only exist in the human body itself, and no one can steal the ECG signal.

In this project, we provide a hardware/software system for using a 1-lead ECG signal to identify people with a deep-learning model.

What's machine learning do in this project? The input of the machine learning model is an ECG signal, and the model will output the prediction result of the ECG signal.

$$f(\text{ECG signal}) = \{0, 1, 2\}$$

Figure 1: machine learning schematic diagram

In brief, in this project, we are going to find a function that maps the given ECG signal to the identification. (0, 1, 2 indicate different people).

Well-defined this problem:

$$ECG\ signal_i\ come\ from\ x \Rightarrow f(ECG\ signal_i) \rightarrow x \in \{0, 1, 2\} [close\ set\ problem]$$

What do we mean by a close-set problem here? In the machine learning region, the model will be trained on a specific set, if the given input is not in that set, the algorithm will force it to put the result into a category (in classification problem) which will be undefined behavior in this project.

Project Pipeline

1. Give the label to each test subject in the set we desired.

$$\left\{ \begin{array}{l} 0: \textit{Coherent} \text{ 何祁恩} \\ 1: \textit{Daniel} \text{ 陳浚維} \\ 2: \textit{Mariia} \text{ 馬莉亞} \end{array} \right\}$$

2. Construct the circuit to the output of the commercial ECG sensor.
 - Amplified the raw ECG signal and level shift to meet the range that can be fed into the Raspberry Pi digital input pin.
 - Sampling the analog ECG signal to the digital signal.
3. Collect enough training data.
4. Data preprocessing.
 - Remove the noise.
 - Cut the multi-cycle ECG signal into 1-cycle.
5. Training the Deep learning model on the preprocessed data.
 - Light/Fast/High predict accuracy
6. Output the result of the prediction and make use of it.
 - Predict results and confidence.
 - Blink the specific LED.
 - GUI windows to show the final result.

Hardware Design

Since the Raspberry pi digital input range is 0 ~ 5 volts, we need to upshift the signal to quantify it into a 10-bit resolution signal. We simply use a non-inverting weighted adder to amplify and level shift the signal.

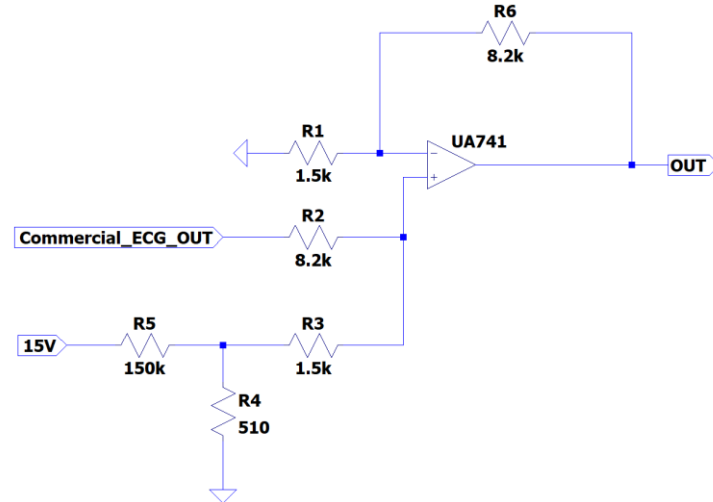


Figure 2: circuit diagram to level shift and amplify the input signal from the analog commercial ECG sensor. (Draw by LTspice)

https://github.com/coherent17/BMElab-Final-Project-ECG-classfier/blob/main/amp_level_shift.asc

$$\begin{aligned}
 V_{out} &= \left(1 + \frac{8.2k}{1.5k}\right) \left(\text{Commercial ECG OUT} \cdot \frac{1.5k}{1.5k + 8.2k} + \left(15 \cdot \frac{510}{150k + 510}\right) \cdot \frac{8.2k}{1.5k + 8.2k} \right) \\
 &\approx (6.5) \cdot (\text{Commercial ECG OUT} \cdot 0.15 + 0.04) \\
 &\approx \mathbf{1} \cdot \text{Commercial ECG OUT} + \mathbf{0.27}
 \end{aligned}$$

Amplify the ECG signal from the commercial ECG sensor by 1 and upshift 0.27 volts. Furthermore, in the specification of the Vernier EKG sensor:

$$\begin{aligned}
 \text{Commercial ECG OUT} &= (1000 \text{ raw ECG signal} + 1) \\
 &\approx \mathbf{1} \cdot (1000 \text{ raw ECG signal} + 1) + \mathbf{0.27} \\
 &\approx \mathbf{1000} \cdot \text{raw ECG signal} + \mathbf{1.27}
 \end{aligned}$$

The total gain and upshift voltage:

$$\begin{cases} \text{gain} = 1000 \left(\frac{V}{V} \right) \\ \text{upshift} = 1.27V \end{cases}$$

Hardware Design continued

After amplifying and level-shifting the raw signal, the amplified signal remains analog, however, the Raspberry pi doesn't support reading the analog input. Thus, we need to sample the amplified signal first. In this project, I choose MCP3008 IC as an analog-to-digital converter (ADC) to convert analog signals from the non-inverting weight adder to digital signals.

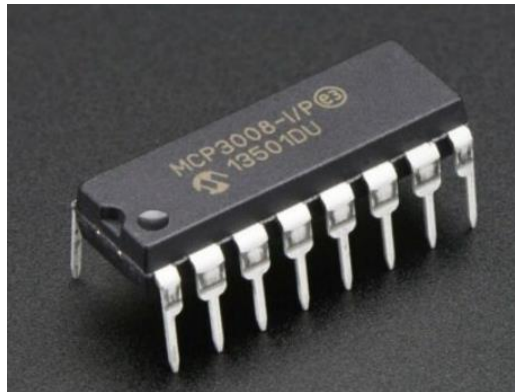


Figure 3: MCP3008 ADC

I use Serial Peripheral Interface (SPI) to communicate between Raspberry Pi General-purpose input/output (GPIO) with the MCP3008 IC, the connections are as follows.

Connect the output of the non-inverting weighted adder to MCP3008 Pin1 (CH0).

MCP3008	Raspberry Pi GPIO
Pin 9 (DGND)	Pin 6 (GND)
Pin 10 (CS/SHDN)	Pin 24 SPI0_CE0 (GPIO8)
Pin 11 (DIN)	Pin 19 SPI0_MISI (GPIO10)
Pin 12 (DOUT)	Pin 21 SPI0_MISO (GPIO9)
Pin 13 (CLK)	Pin 23 SCLK (GPIO11)
Pin 14 (AGND)	Pin 6 (GND)
Pin 15 (VREF)	Pin 1 (3.3V)
Pin 16 (VDD)	Pin 1 (3.3V)

Run the readData.py by typing \$ python3 readData.py on terminal
And wait until the progress bar finish

```
1  import busio
2  import digitalio
3  import board
4  import adafruit_mcp3xxx.mcp3008 as MCP
5  from adafruit_mcp3xxx.analog_in import AnalogIn
6  import time
7  import sys
8  import csv
9  from tqdm import tqdm, trange
10
11 def readData(duration = 1000):
12     spi = busio.SPI(clock=board.SCK, MISO=board.MISO, MOSI=board.MOSI)
13     cs = digitalio.DigitalInOut(board.D22)
14     mcp = MCP.MCP3008(spi, cs)
15     chan = AnalogIn(mcp, MCP.P0)
16
17     ecg_signal = []
18
19     for i in trange(duration):
20         ecg_signal.append(float(str(chan.voltage)))
21         time.sleep(0.01)
22
23     print(ecg_signal)
24
25     #write the ecg signal data to file
26     filename = 'ecg_signal.csv'
27     with open(filename, 'w') as csvfile:
28         csvwriter = csv.writer(csvfile)
29         csvwriter.writerow(ecg_signal)
30
31 if __name__ == '__main__':
32     readData(duration = 1000)
```

Figure 4: readData.py code screenshot

The code is available on my GitHub repository:

<https://github.com/coherent17/BMElab-Final-Project-ECG-classfier/blob/main/readData.py>

This code simply uses the SPI to get the digital ECG signal per 10ms (sampling rate = 100Hz) and store the data point in the “ecg_signal” list. Next, output the result to the “ecg_signal.csv” file for later use.

Both training data collection and demo data collection use the same code, except for the different duration (line 32: default using duration = 1000 * 0.01 = 10 sec) of getting the ECG signal.

For training data, we collect 30 sec per data. As for demo data, to reduce the time without affecting the accuracy, 10 seconds is enough.

The DKK column is the label $\{0, 1, 2\}$: {Coherent, Daniel, Mariia}.

Data exploration

After collecting the data from executing readData.py, let's observe these amplified and level-shift data, and try to figure out if there exists any difference between the ECG signals that come from us. Using python and matplotlib to perform data visualization so that we can have a better understanding of the data.

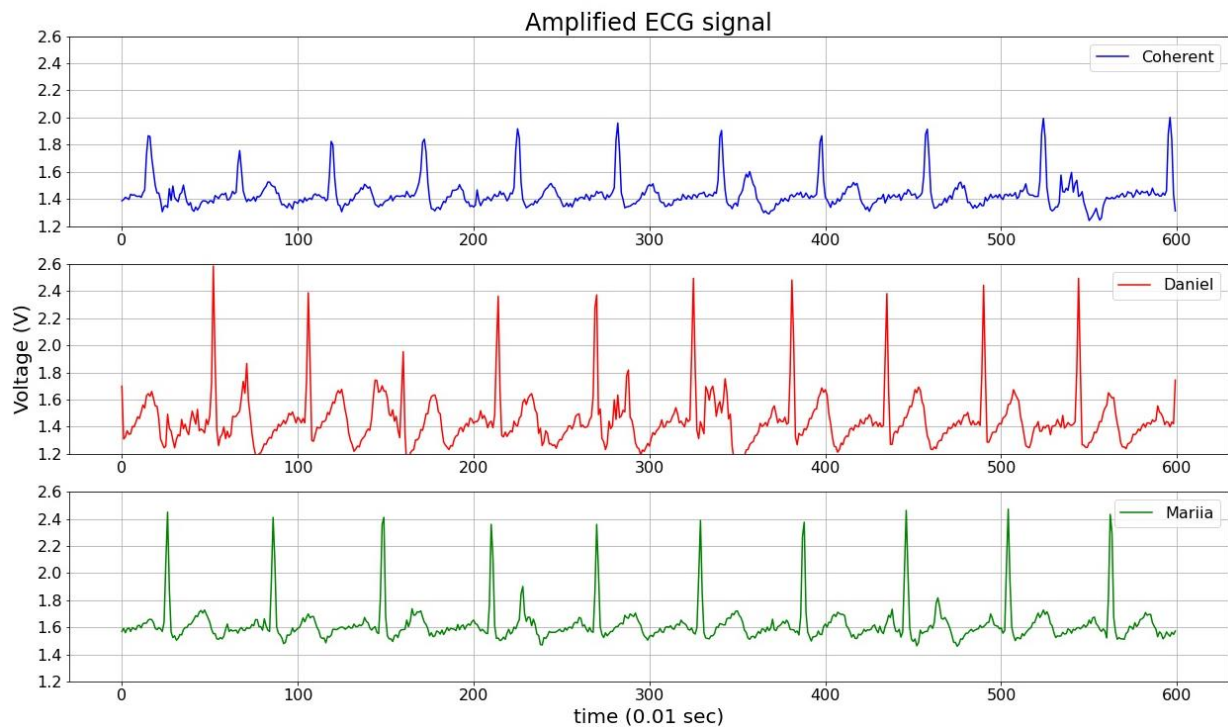


Figure 7: Amplified ECG signal subplot

From row [26 6 36] in 1216_ecg_data.csv

The amplified ECG signal in figure 7 with the blue line comes from **Coherent**, as you may notice in the DKK column in the 1216_ecg_data.csv row 26 (with label 0). Let's see if there exist any features in this plot.

- The R-peak voltage is around 1.8V ~ 1.9V
- The S-peak is around 1.3~1.4V
- The T-peak is around 1.5V

The amplified ECG signal in figure 7 with a red line comes from **Daniel**, as you may notice in the DKK column (with label 1).

- The R-peak voltage is around 2.3V ~ 2.5V
- The S-peak is around 1.2~1.3V
- The T-peak is around 1.7V
- A little not periodic

The amplified ECG signal in figure 7 with the green line comes from **Mariia**, as you may notice in the DKK column (with label 2).

- The R-peak voltage is around $2.4V \sim 2.5V$
- The S-peak is around $1.3V$
- The T-peak is around $1.7V$
- Very periodic

We can have a little conclusion here:

The ECG signal is different from the person to person. The shape and the amplitude are the features of the ECG signal. It's quite easy for us to observe the difference between our ECG signals, however, if the number of test subjects increases, it's not easy for human-being to recognize all of the features by only using observation and simple statistic math. Therefore, rather than just doing some mean, and average in another group, we also build a model using the deep-learning technique to classify the ECG signal between us.

So, what's the main purpose of building this model?

The model takes only 1-cycle of our ECG signal and outputs the classification result of our label.

However, in the real world, the ECG signal isn't a noise-free signal and is not fully periodic. The problem of segmenting the 1-cycle of ECG signal is our first difficulty in this final project.

Furthermore, the noise problem while measuring the ECG signal is another issue. Since we place the electrodes on the hands, the EMG signal may also be included in our tests, these noises should also be removed in the data preprocessing stage.

Data Preprocessing

When training the machine learning model, the most important thing must be data preprocessing. In this part, there are two things we need to finish.

- Segment the ECG signal into 1-cycle
- Remove noise

It's quite a challenge for me to develop an algorithm for data preprocessing. Rather than training the model. I spend most of my time on this part designing a good algorithm.

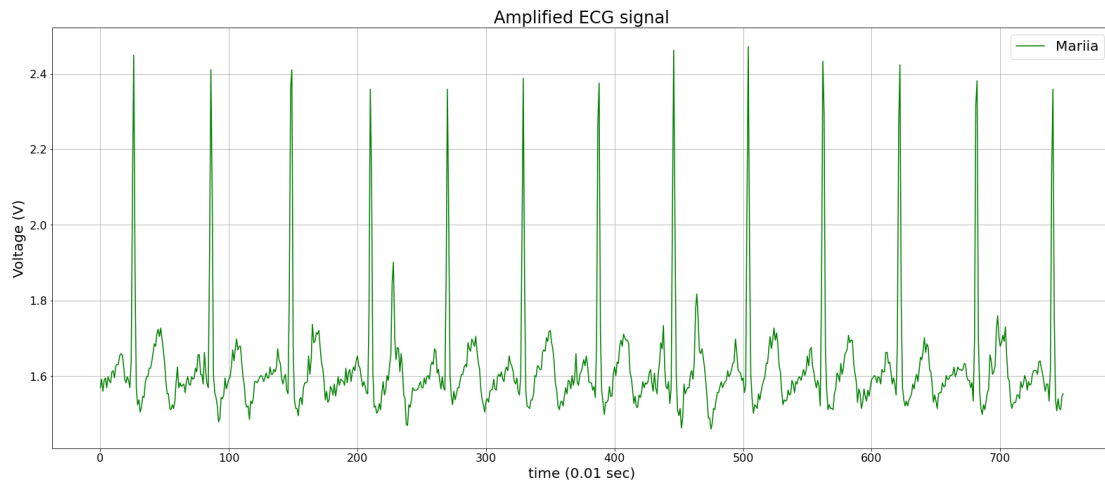


Figure 8: input of data-preprocessing module

From row 36 in 1216_ecg_data.csv

In figure 8, the ECG signal comes from Mariia, I want to design an algorithm to cut the signal into 1-cycle (Figure 9 below) and reject the noise component.

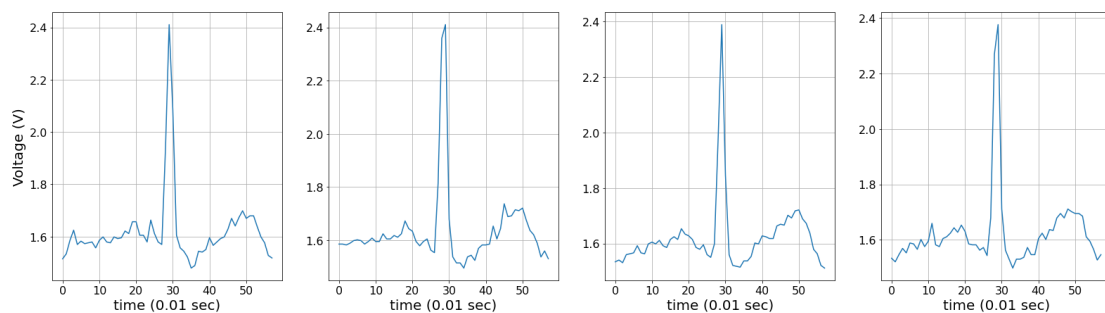


Figure 9: output of the data-preprocessing module

1-cycle amplified ECG signal

Algorithm Version1

Concept: Using alpha-trimmed filter concept to determine the cut period in the figure8 signal, and cut continuously in that specific period.

The alpha-trimmed filter is the filter used in digital image processing. Alpha-trimmed filters can be used to eliminate the extreme value in the given range. When it using in digital image processing, it can eliminate the pepper and salt noise, which is the black-and-white pixels in the image world.

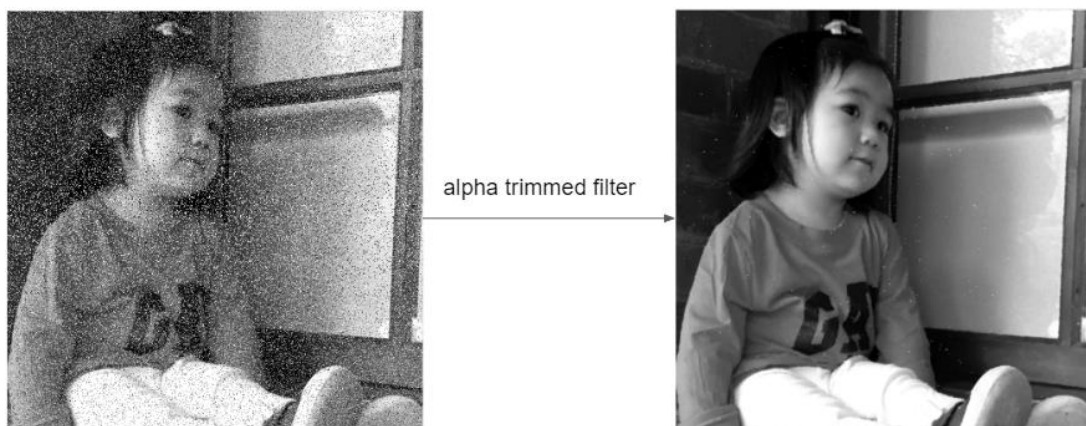


Figure 10: alpha-trimmed filter example

With the kernel moving in the given image, it removes the lowest and highest value and average on the other pixel magnitude. By doing so, we can successfully remove the extreme value in the image.

So, what's the relation between the pepper and salt noise in digital image processing related to our ECG project? You will figure it out later!

Steps in Algorithm version 1:

1. Find the position of the R-peak
2. Perform finite difference and sort
3. Using the alpha-trimmed filter to get the period we want
4. Continuously cut the signal in Figure 8 with approximate offset time, so that it includes all features in the one cut (P, Q, R, S, T peaks).

Example of Algorithm 1

1. Find the position of the R-peak

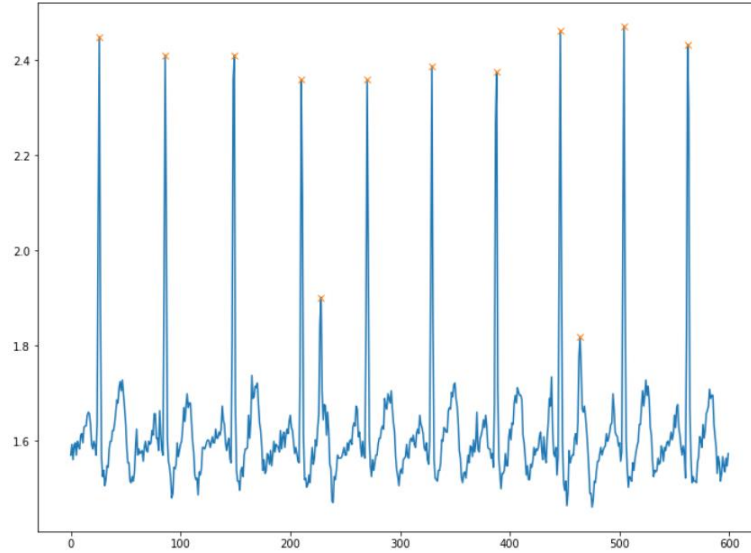


Figure 11: R-peak position example

Simply finding the peak timing position with voltage over $0.5 \cdot \max$ (amplified ECG signal), however, it might also include some noise and other peaks, we can correct it in the next few steps.

$$position = [26, 86, 149, 210, 228, 270, 329, 388, 446, 464, 510, 562]$$

2. Find the finite difference of the position array called position difference, the element in this array is the consecutive R-peak time duration.

$$position_difference = position[i] - position[i-1], \forall i \in length(position)$$

$$position_difference = [60, 63, 61, 18, 42, 59, 58, 18, 40, 58]$$

3. Sort the duration and eliminate the lowest and highest part and average the rest to get the mean value.

$$sort_position_difference = [18, 18, 40, 42, 58, 58, 59, 59, 60, 61, 63]$$

$$alpha_trimmed_filter = [58, 58, 59, 59]$$

$$period = mean([58, 58, 59, 59]) = 58$$

4. Find the offset time and cut continuously with the period calculated in step 3.

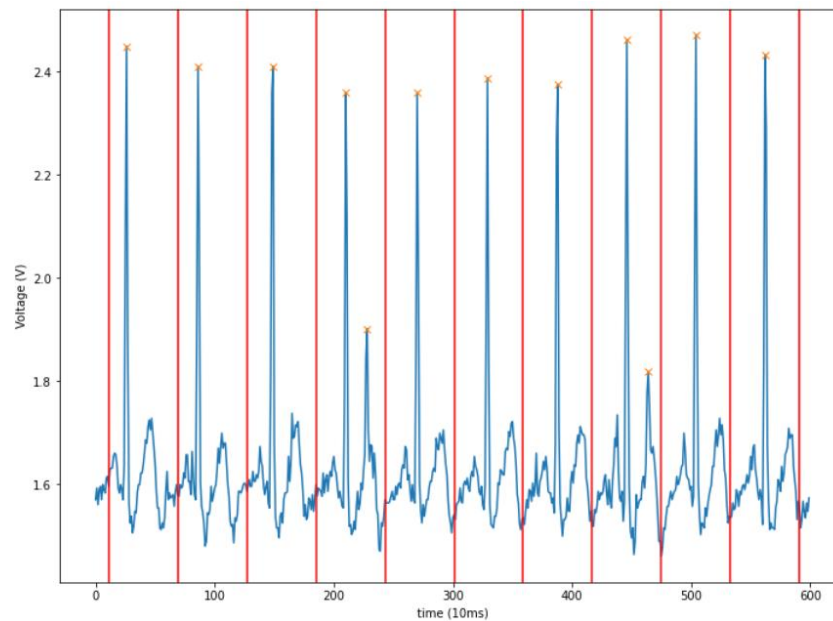


Figure 12: Cut with the same period

It seems to work well, however, notice the leftmost part, if we cut in the same cut, that R-peak not close to 58 might fail to cut the whole cycle in this algorithm, therefore, this algorithm will fail.

Another extreme example is when the test subject is not stable or maybe suffer from an arrhythmia problem, some of the cut will fail.

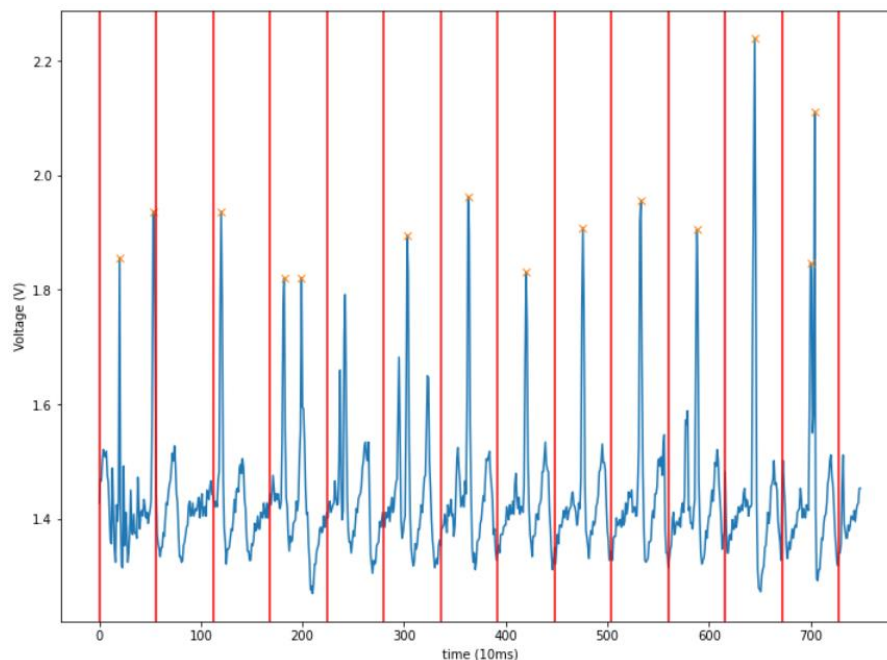


Figure 13: Cut with same period failed extreme case

Instead of using fix cutting period, perhaps a dynamic cutting period is better!

Algorithm 1 Improved

Let's first well-defined a good cycle:

A good cycle should include:

- All peaks (P, Q, R, S, T peaks).
- 3 consecutive R-peaks in the range near the peak we calculated in 3 steps in algorithm version 1 ex: 58 ± 5

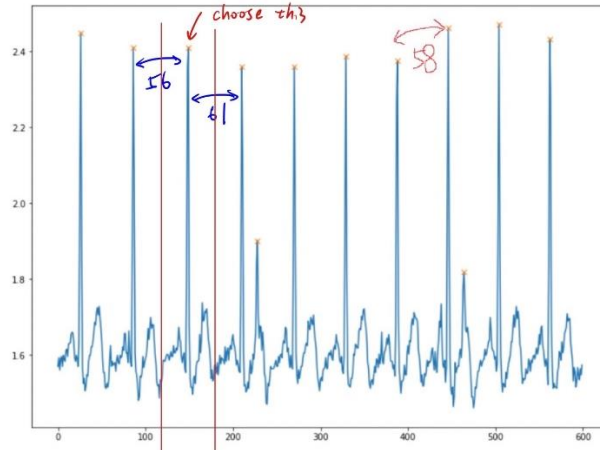


Figure 14: Algorithm 1 improved by dynamic cut

Accept if the consecutive 3 R peaks finite difference durations have the property:

$$period - threshold < duration < period + threshold$$

For example, in the figure14, if the threshold = 10, then we should accept the duration $= 58 \pm 10$. If the 3 consecutive R peaks have 2 durations = 56 and 61, then we should cut the middle cycle from the R-peak position-period/2 to the R-peak position+period/2.

By strictly well-define the good cycle, we can cut the ECG signal with all peaks information included and reject most of the noise by using the alpha-trimmed filter.

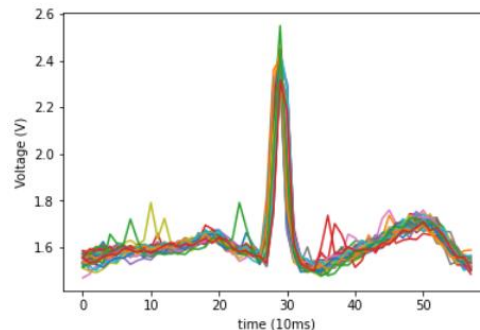


Figure 15: Cut superposition

After the cutting process, all of the ECG signals almost have the same shape and location in the same place. Therefore, our model can converge in a very short period.

Model training:

After well data preprocessing, we are now going to model the training part. Just a quick review what the purpose of this model. The model takes 1 cycle ECG amplified signal as input and outputs the category corresponding to the classified result.

$$f(\text{ECG_Signal}) = \{0, 1, 2\}$$

Figure 16: machine learning schematic diagram

After data preprocessing in the last part, our dataset distribution is as follows:

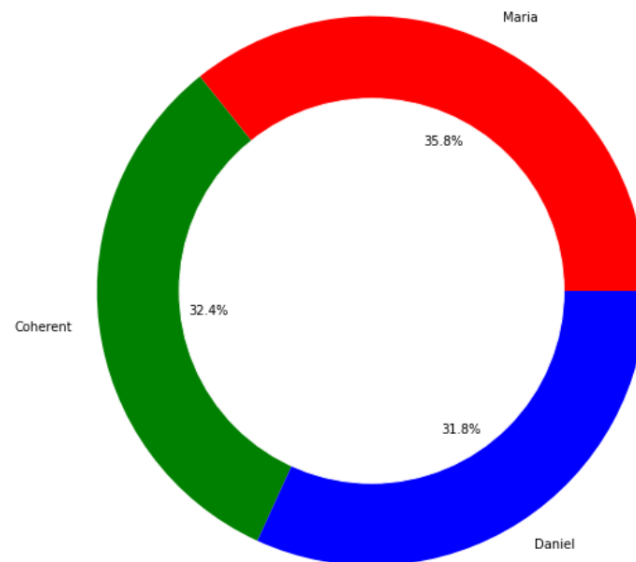


Figure 17: Distribution of the dataset

From figure 17, we can know that the dataset distribution is quite an equilibrium, therefore, if the machine learning model guesses randomly, the accuracy should be at most 33%.

At first, I use the dataset that comes from the algorithm version 1, the output of the dataset contains so many inappropriate cuts, so the model prediction accuracy is always around 40%. After I optimize the algorithm we used to cut the amplified ECG signal, the accuracy improved so much.

I use 70% of the data in the dataset to train the model, and the rest of the 30% for the validation set, try to give the model these un-seen data so that I can tune the model well.

Model Network:

Layer (type)	Output Shape	Param #
inputs_cnn (InputLayer)	[(None, 186, 1)]	0
conv1d (Conv1D)	(None, 181, 64)	448
batch_normalization (Batch Normalization)	(None, 181, 64)	256
max_pooling1d (MaxPooling1D)	(None, 91, 64)	0
conv1d_1 (Conv1D)	(None, 89, 64)	12352
batch_normalization_1 (Batch Normalization)	(None, 89, 64)	256
max_pooling1d_1 (MaxPooling1D)	(None, 45, 64)	0
conv1d_2 (Conv1D)	(None, 43, 64)	12352
batch_normalization_2 (Batch Normalization)	(None, 43, 64)	256
max_pooling1d_2 (MaxPooling1D)	(None, 22, 64)	0
flatten (Flatten)	(None, 1408)	0
dense (Dense)	(None, 64)	90176
dense_1 (Dense)	(None, 32)	2080
main_output (Dense)	(None, 3)	99
Total params: 118,275		
Trainable params: 117,891		
Non-trainable params: 384		

Figure 18: model network

Simple CNN with 3 convolution layers. Using activation function as Relu and optimizer as Adam.

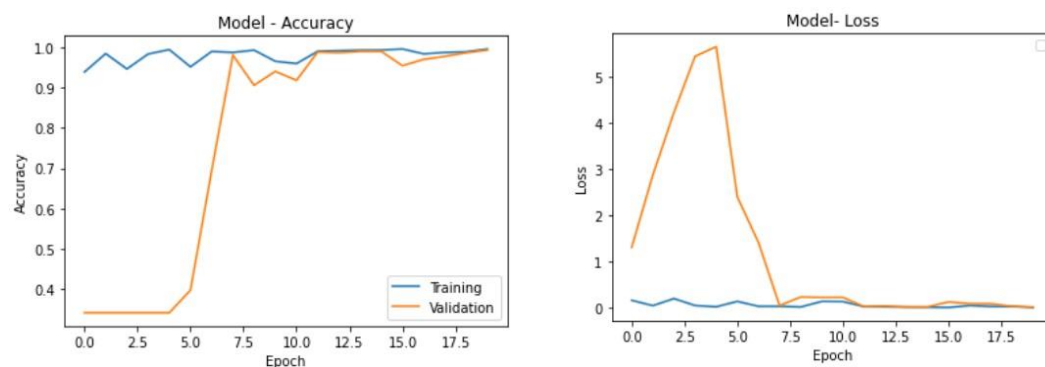


Figure 19: model training loss and accuracy vs epoch

As you can see in figure 19, the accuracy converges to 90% correctness after 6 epochs, and the loss of both training and validation data is low, thus, the model works quite well.

Model Result:

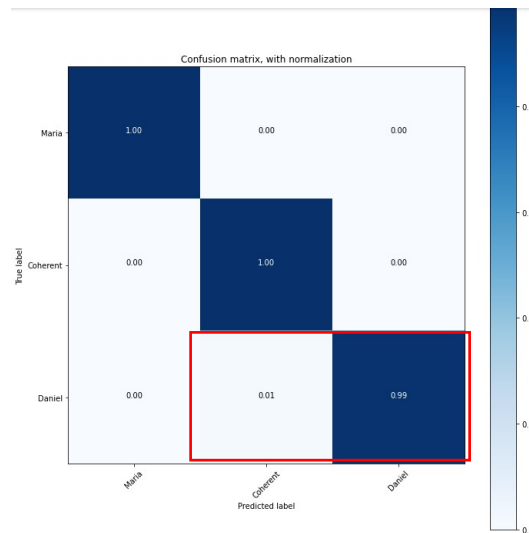


Figure 20: Confusion matrix

In the confusion matrix of the predicted result, we can see that the model has a 1% chance to classify Coherent and Daniel wrong. The result is quite good, but can we keep improving it?

Voting System:

When demoing, we measure the test subject for 10 seconds, thus, we can have a multi-good cycle defined in algorithm 2, therefore, since our model takes 1 cycle of amplified ECG signal as input, why don't we put all of the 1-cycle cuts ECG signal into the model and vote by the predicted result from the model, therefore, there is no way to misclassify.

$$f(\text{ECG}_1) = 0$$

$$f(\text{ECG}_2) = 1$$

$$f(\text{ECG}_3) = 1$$

Figure 21: Voting system example

If there exist 3 good cuts in the measured signal, then we put all of the signals into the model and choose the mode in the prediction result as the final result. In this example, the final result is 1.

Result Display:

After predicting the result, I will output the log file which contains the prediction result after the voting system and the prediction confidence by the model, after that, I will flash a certain LED to represent the classification result and output a window about the result and the confidence.

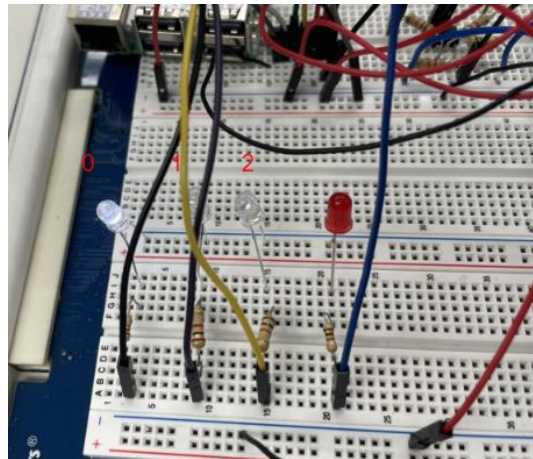


Figure 22: LED shine

If the label 0 LED shine, Coherent is the final result, else if the label 1 LED shine, Daniel is the final result, else if the label 2 LED shine, the result is Mariia.

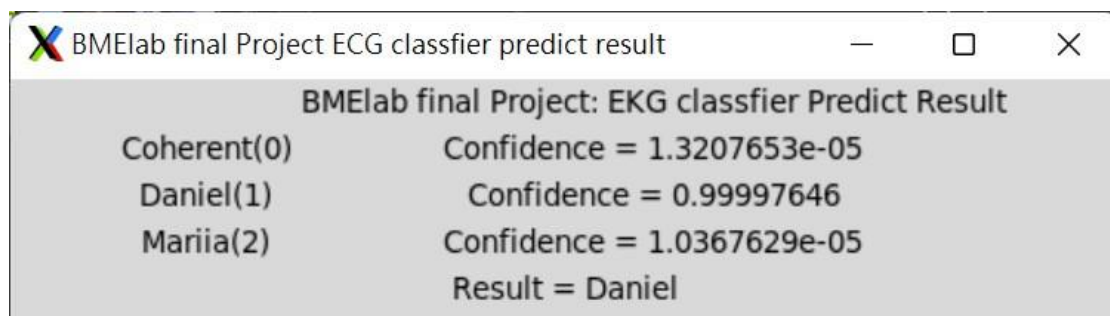


Figure 23: Window to show the final result and confidence

Final Remark:

I had already mentioned in the beginning, our problem is the close-set problem, the model can only be used in the specific pre-trained datasets, however, what if we input TA's ECG signal (not in the set)? The confidence will be close to one-third for each, and therefore, it can perhaps detect the person who is not in this pre-trained set.

Testing Protocol:

1. The test subject sits stable and calms down.
2. Put on the electrode on the arms.

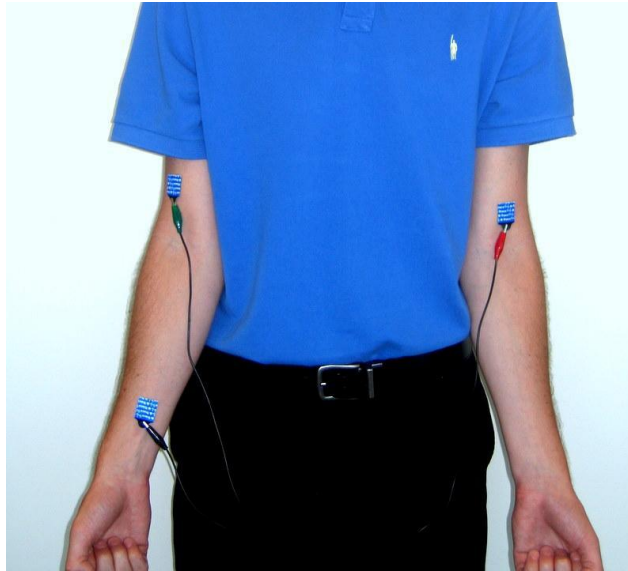


Figure 24: Electrode placement

3. Run readData.py and get the ecg_signal.csv file upload to the google drive
4. Run the model and get the prediction result in the log file
5. Using a log file to flash the specific LED and show the message box

Conclusion:

Using machine learning techniques, we can classify the difference between human ECG by our hardware/software system design.

Feedback:

I spend intractable time on this project, I think I do at least 60% job in this final project, from connecting the Raspberry Pi system with the circuit, data preprocessing, machine learning model, and the data visualization part. There are so many debug nights at home, just willing to design an amazing algorithm to fulfill the thought in my mind. Thanks to the TA and my teammate's help, I can finish this big project. This project let me see the infinite possibility in our EE major. All of the group integrate their major into this project. The results are incredible and astonishing.

Reference

[1] The thesis proved that ECG signals can be used to identify people:

Person identification using ECG signal's symbolic representation and dynamic time warping adaptation

Leila Yousofvand, Abdolhossein Fathi & Fardin Abdali-Mohammadi

<https://link.springer.com/article/10.1007/s11760-018-1351-4>

[2] Serial Peripheral Interface (SPI)

<http://wiki.csie.ncku.edu.tw/embedded/SPI>

[3] MCP3008 datasheet

<https://datasheetspdf.com/pdf/439989/MicrochipTechnology/MCP3008/1>

[4] The connection between Raspberry Pi and MCP3008

<https://www.hackster.io/talof99/analog-input-on-raspberry-pi-with-mcp3008-e64e43>

<https://atceiling.blogspot.com/2014/04/raspberry-pi-mcp3008.html>

[5] Vernier EKG sensor specification

<https://www.vernier.com/product/ekg-sensor/>

[5] Find peak on discrete time signal

https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.find_peaks.html

[6] Alpha-trimmed filter and my implementation

<https://ithelp.ithome.com.tw/m/articles/10272112>

https://github.com/coherent17/Digital-Image-Processing/blob/main/Project3/Project3_report.pdf

[7] Convolutional Neural Network (CNN)

https://www.youtube.com/watch?v=OP5HeXJg2Aw&list=PLJV_el3uVTsMhtt7_Y6sgTHGHP1Vb2P2J&index=9

<https://github.com/dave-fernandes/ECGClassifier>

<https://www.kaggle.com/code/gregoiredc/arrhythmia-on-ecg-classification-using-cnn>

<https://github.com/alinstein/Human-Identification-with-ECG-->


Video Link:

<https://youtu.be/U-uMFqbB7Aw>

Code Link:

<https://github.com/coherent17/BMElab-Final-Project-ECG-classfier>

Code Structure:

	coherent17 update circuit	df59d72 3 hours ago	🕒 26 commits
📁	Data	remove unuse data	3 days ago
📁	Model	connect all file	3 weeks ago
📄	LICENSE	Initial commit	last month
📄	PredictResult.py	add final src	last week
📄	README.md	final version	4 days ago
📄	amp_level_shift.asc	update circuit	3 hours ago
📄	amp_level_shift.log	update circuit	3 hours ago
📄	readData.py	final version	4 days ago

In the Data folder:

📄	1216_ecg_data.csv	finish data preprocessing	last month
📄	1223_ecg_data.csv	add 1223 ecg dataset	3 weeks ago
📄	1227_ecg_data.csv	add ecg data	3 weeks ago
📄	data_preprocessing.ipynb	add preprocessing code	last week
📄	data_preprocessing.py	add preprocessing code	last week
📄	preprocessingData.csv	update data	3 weeks ago

- 1216_ecg_data.csv/1223_ecg_data.csv/1227_ecg_data.csv are the amplified ECG signal measured by Raspberry Pi.
- preprocessingData.csv is the training dataset by executing the data_preprocessing.py

In the Model folder:

📄	best_model.h5	connect all file	3 weeks ago
📄	ecg_class.ipynb	connect all file	3 weeks ago
📄	ecg_class.py	connect all file	3 weeks ago

- best_model.h5: the weight file from the tuned model
- ecg_class.ipynb: code for CNN model

PredictResult.py:

Read the log file(result.csv) produced by the ecg_class.py file and shine the corresponding LED and show the message box.

amp_level_shift.asc:

The spice file simulates the non-inverting weighted adder

readData.py:

Code for reading the data from commercial ECG sensors and storing the data into ecg_signal.csv

If there is any problem with the code, please contact me:

Gmail: mnb51817@gmail.com

GitHub account: coherent17 <https://github.com/coherent17>