PROJECT-II

ECG BASED RECORDING TO SHOW REAL TIME CARDIAC ACTIVITY

Submitted to:

Dr. Muhammad Jawad Khan

Submitted by:

Usman Zaheer

Faisal Javed

Soban Ahmed

Medical Devices & Robotics

Contents

1. Problem Statement:

Design the hardware for ecg based recording to show real time cardiac activity. You have to design a circuit (Ardrino based) and use the recorded data to differentiate between two classes.

In this project, a Arduino based circuit is used to record the ECG. AD8232 ECG sensor is connected with Arduino and the electrodes to get the desired signal. After acquiring the signal, it is preprocessed and then neural network model composed of 01 input layer, 3 hidden layers and 01 output layer is implemented using tensor flow and keras for classification and comparison of normal and abnormal person ECG.

2. Introduction

An electrocardiogram (ECG or EKG) is a recording of the electrical activity of the heart. Heart muscles, like skeletal muscles, are electrically induced to contract. Excitation or activation are terms used to characterize this form of stimulation. At rest, cardiac muscles are electrically charged. In contrast to the outside, the interior of the cell is negatively charged (resting potential). As cardiac muscle cells are electrically activated, they depolarize (change from negative to positive resting potential) and contract. The action potential is a calculation of a single cell's electrical activity. The electrical field changes size and direction as the electrical impulse passes through the heart.. **Heart rhythm.** An ECG can reveal abnormal heart rhythms (arrhythmias). When some part of the heart's electrical system fails, these conditions may grow. Arrhythmias can also be caused by drugs such as beta blockers, cocaine, amphetamines, and over-the-counter cold and allergy medications. **Heart attack.** An ECG may disclose symptoms of a previous heart attack or a current one. The patterns on your ECG can show which part of your heart has been damaged and how serious it is.

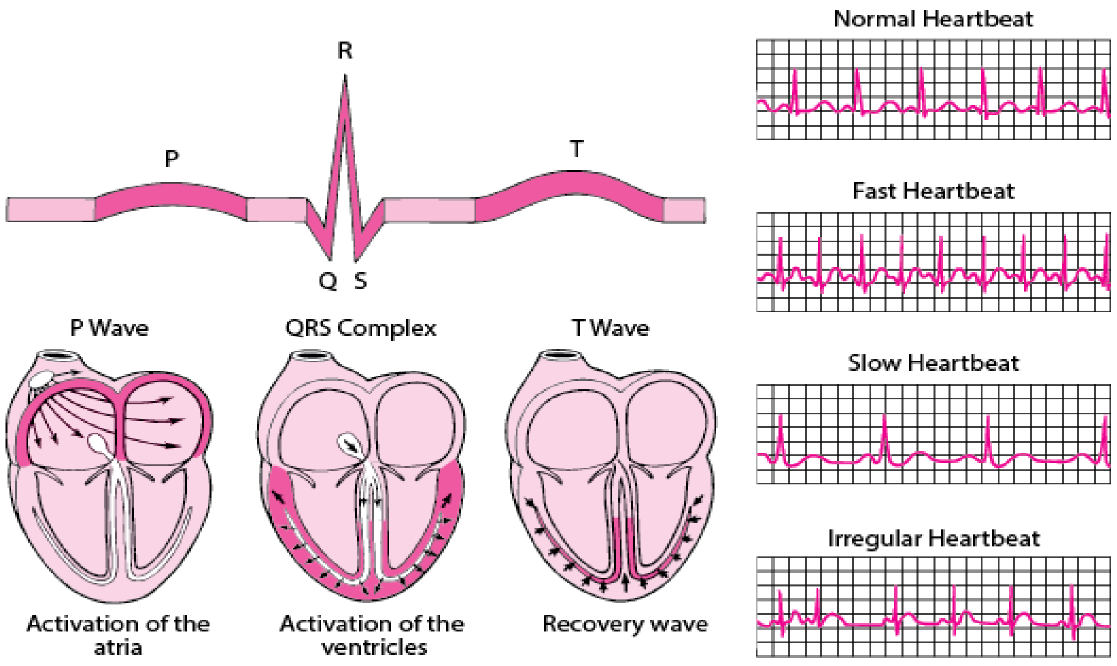


Figure 1: ECG Signal Characteristics

3. Methodology:

The methodology opted for this project is in the following steps:

1) Hardware Design & Selection of Components
2) Assembly of components
3) Data recording
4) Data downloading, reading, and cleaning.
5) Assigning features to x and labels to y.
6) Data splits.
7) Defining the architecture of neural network model.
8) Training on training data split and updating parameters.
9) Metrics and comparison.
10) Results Visualization

4. Hardware Design & Selection of Components

In this project, the following components has been selected for recording of ECG signal.

NAME	ROLE	QUANTITY
ECG SENSOR AD8232	**It is an integrated signal conditioning block for ECG.**	**1**
Arduino Uno	**For interfacing with PC and ECG Sensor and processing the signal**	**1**
Male and Female Communication Cables	**Male and Female Communication Cables for Arduino and ECG Sensor**	**1**
ECG Electrodes	**For acquiring signal**	**3**
Electrodes Connector Cable with 3.5mm Jack	**For Connection between electrodes and ECG Sensor**	**1**

ECG Sensor AD8232:

The AD8232 ECG sensor is a commercial board used to calculate the electrical movement of the human heart. This action can be chart like an Electrocardiogram and the output of this is an analog reading.

Electrocardiograms can be very noisy, so to reduce the noise the AD8232 chip can be used. The working principle of the ECG sensor is like an operational amplifier to help in getting a clear signal from the intervals simply.

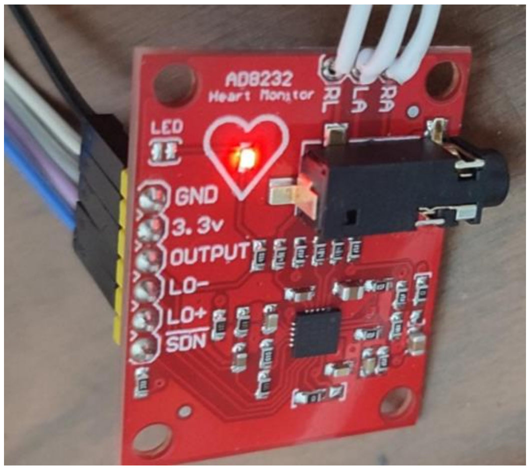


Figure 2: ECG Sensor AD8232

Working of ECG Sensor:

ECG or Electro Cardio Graphy is a method to measure some important parameters of a human heart. It outputs analog values that produce a particular signal. As visible, the signal has a few peaks and important features that are of biological importance. These are marked below.

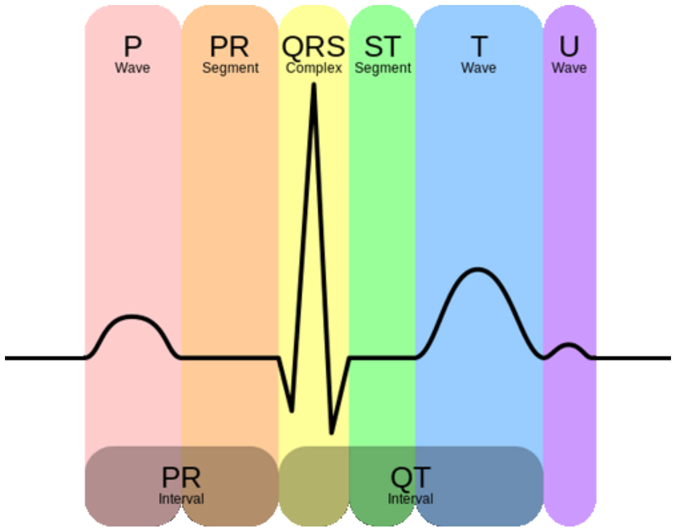


Figure 3: ECG Signal

Each interval has an optimal value range, and deviation from that might be linked to a particular disease. Here are the main parts of an ECG signal.

1. P wave - It is the trailing wave on the left of the QRS complex.
2. QRS complex - It is an impulse generated by ventricular contraction.
3. T wave - It is a leading wave right to the QRS complex.

4. U wave - It is not always observed due to its low peak value.

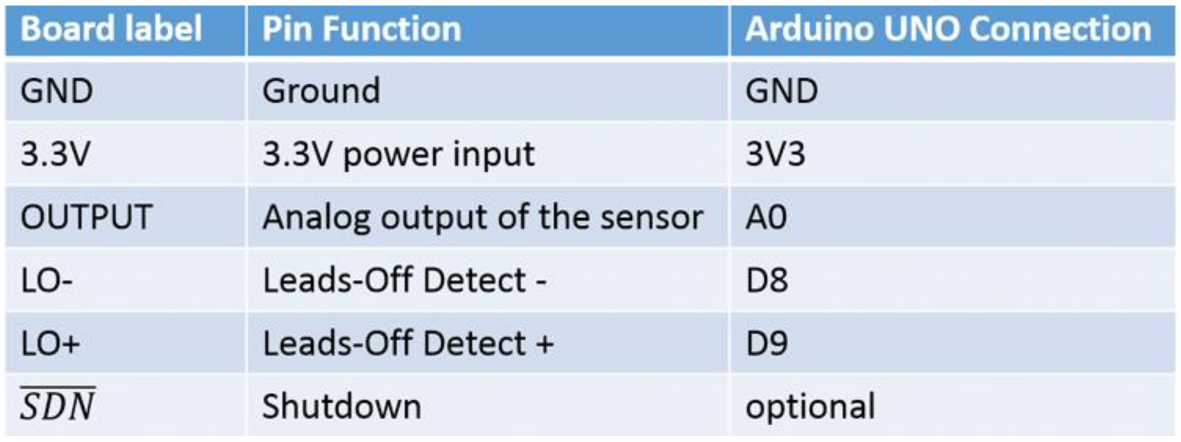
There are many other features as well, but these are the main ones. Based on the shapes of the above features, their interval as well as the interval between them, we can diagnose a lot of cardiac diseases and irregularities. For example:

- Irregular heartbeat or absence of P-wave: Atrial Fibrillation
- Resting Heart Rate of more than 100: Tachyarrhythmia
- Tachyarrhythmia and delta wave: Wolf-Parkinson-White or WPW syndrome
- Sawtooth P wave: Atrial flutter
- Depression of ST-segment: it might indicate Ischemia
- Elevation of ST-segment: it might indicate myocardial Infarction

Hence, ECGs are extremely important for a cardiologist or any doctor for that matter.

5. Assembly of Components and Interfacing with Arduino:

The AD8232 ECG sensor is the most used and available ECG sensor that is affordable and can be used for hobby purposes. It is a 3 lead or single-channel ECG module. The AD8232 from Analog Devices is a 3 lead ECG sensor, which has been converted into various breakouts and modules by Sparkfun and other 3rd party electronics manufacturers. All breakouts generally contain the following pins:



Board label	Pin Function	Arduino UNO Connection
GND	Ground	GND
3.3V	3.3V power input	3V3
OUTPUT	Analog output of the sensor	A0
LO-	Leads-Off Detect -	D8
LO+	Leads-Off Detect +	D9
$\overline{SDN}$	Shutdown	optional

Figure 4: Pins Connection

The shutdown pin is used to send the AD8232 sensor into standby mode, during which it only consumes a current of 200nA. Generally, that mode is not used because the ECG sensor data are to be taken continuously, but we can code in such a manner that the module enters standby mode when electrodes are removed or on a button press. A few possible schematics are shown below.

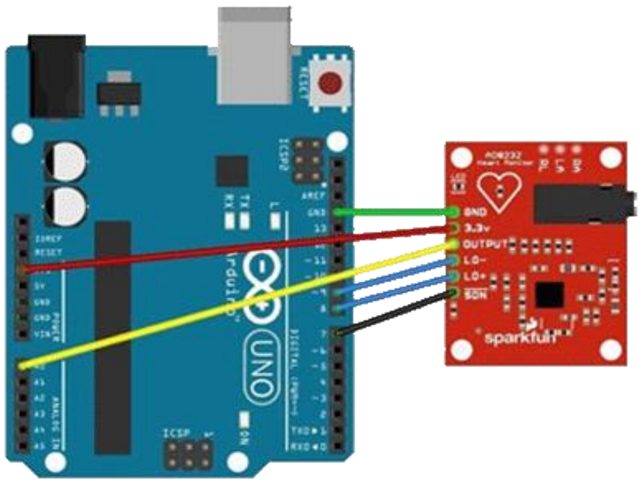


Figure 5: AD8232 interfacing with Arduino

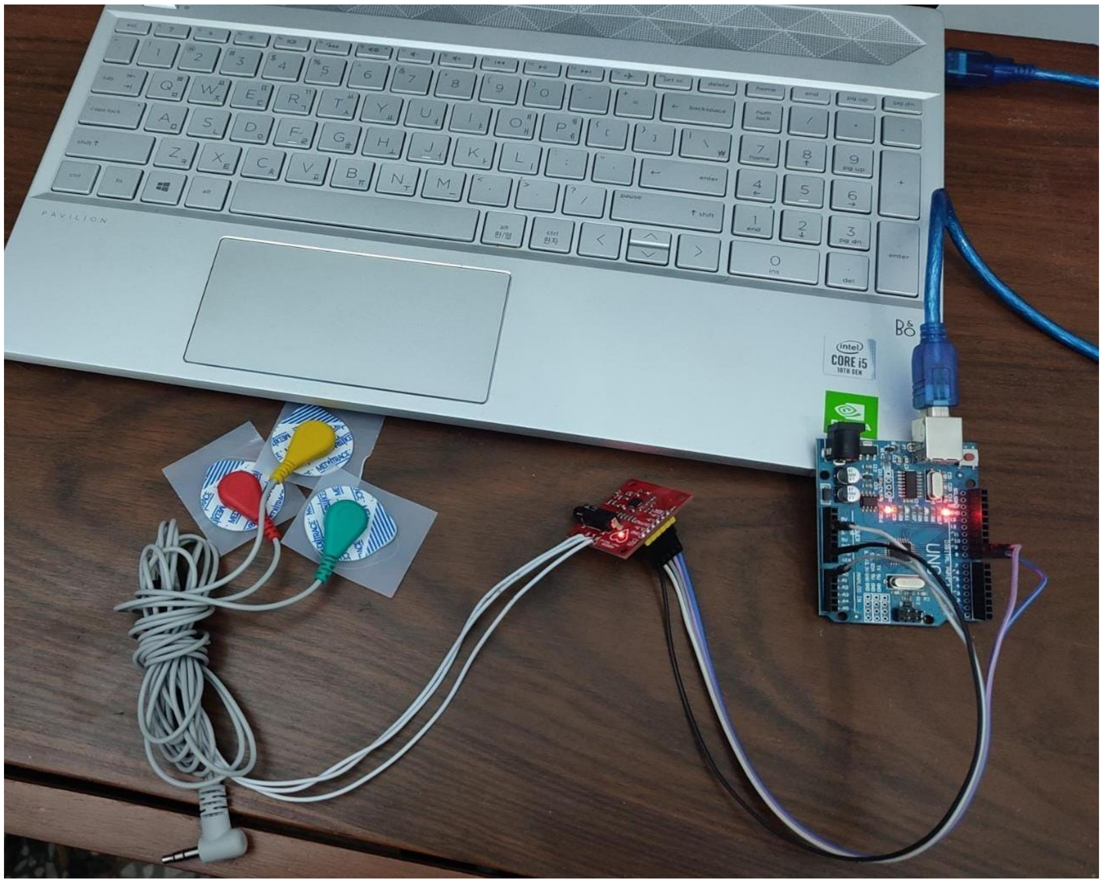


Figure 6: AD8232 with Electrodes

6. Dataset Recording and Electrodes Placement:

For recording ECG Signal, the 3 electrodes are connected to ECG sensor using the 3.5mm jack and their contacts are placed on the body at respective positions.

For a 3-lead system, the electrodes placements are shown in fig below:-

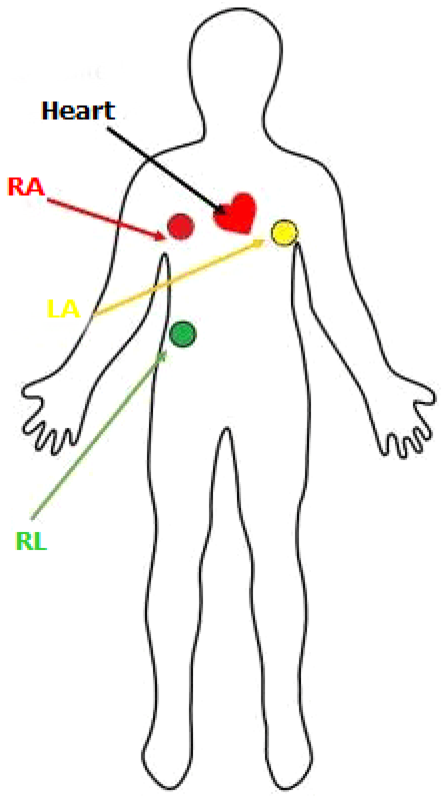


Figure 7: Electrodes placement on body

Electrode Name	**Electrode colour**	**Location**
RA	**Red**	**Right Chest/Arm**
LA	**Yellow**	**Left Chest/Arm**
RL	**Green**	**Right/leg side of Abdomen**

The code for ECG signal generation is written in python and run-on Arduino. Code is available in Annex C. The data is recorded in Arduino PC app and saved as a .txt file. This data saved in the form of txt is loaded in python, preprocessed, and then used for the leaning algorithm

7. Dataset Details:

In this project, 02 types of data are used.

1) ECG data of normal person obtained using our hardware.
2) Abnormal ECG data of a person obtained from “St Petersburg INCART 12-lead Arrhythmia Database”

Both data are saved at a sampling frequency of 250 hz and then the whole data set is divided into a sub- sample of 5secs. From both data’s, the first 30000 datapoints are used and then further resampled so that there are no issues in data mis matching and sample frequencies.

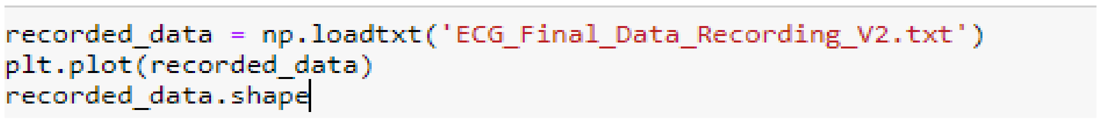
Data Description:

i. Data Gathering

- The normal person ECG data is saved in .txt file ECG_Final_Data_Recording_V2.txt. obtained from our circuit.
- Abnormal person main data file can be downloaded from: https://physionet.org/content/incartdb/1.0.0/ “St Petersburg INCART 12-lead Arrhythmia Database”.
- Data set is downloaded on laptop and folder is created with name “/incart-data/” renamed as Project_data. From this data set, the data of one person is used for this project who have various cardiac abnomrlities.

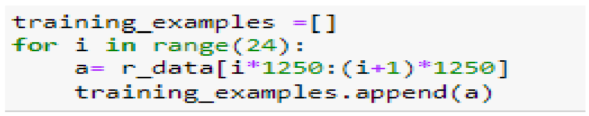
ii. Data Reading

The normal person ECG data is loaded using the numpy library of python as shown below:



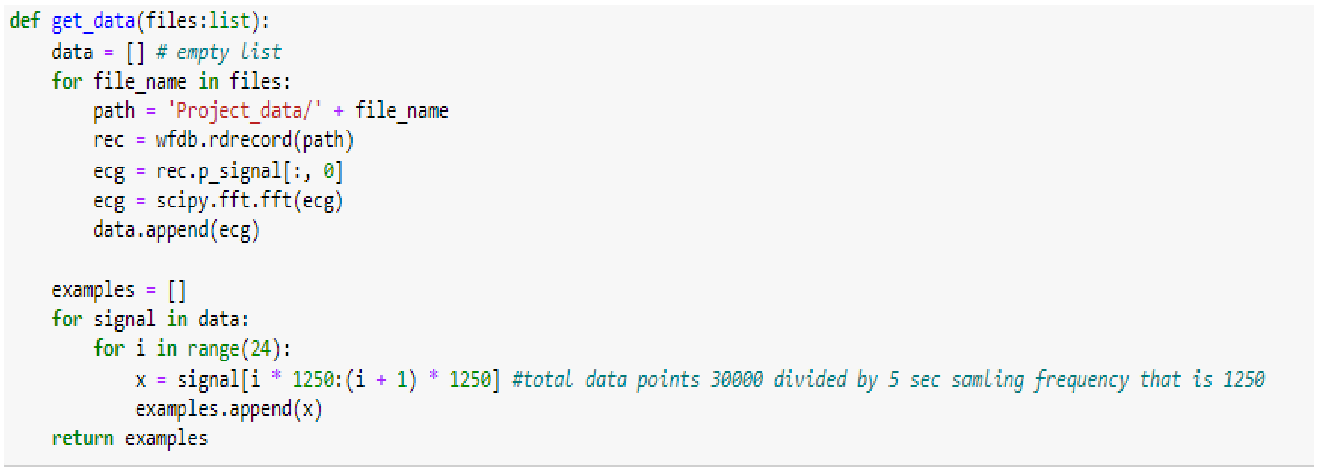
```
recorded_data = np.loadtxt('ECG_Final_Data_Recording_V2.txt')
plt.plot(recorded_data)
recorded_data.shape
```

It is then further divided into subsamples of 5sec for more training examples.



```
training_examples =[]
for i in range(24):
    a= r_data[i*1250:(i+1)*1250]
    training_examples.append(a)
```

The “Get_data” function is created to load abnormal dataset from folder. And samples are created for each 05 seconds. As this data is 12 lead data, in this project only first lead data is used.



```
def get_data(files:list):
    data = [] # empty list
    for file_name in files:
        path = 'Project_data/' + file_name
        rec = wfdb.rdrecord(path)
        ecg = rec.p_signal[:, 0]
        ecg = scipy.fft.fft(ecg)
        data.append(ecg)

    examples = []
    for signal in data:
        for i in range(24):
            x = signal[i * 1250:(i + 1) * 1250] #total data points 30000 divided by 5 sec samling frequency that is 1250
            examples.append(x)
    return examples
```

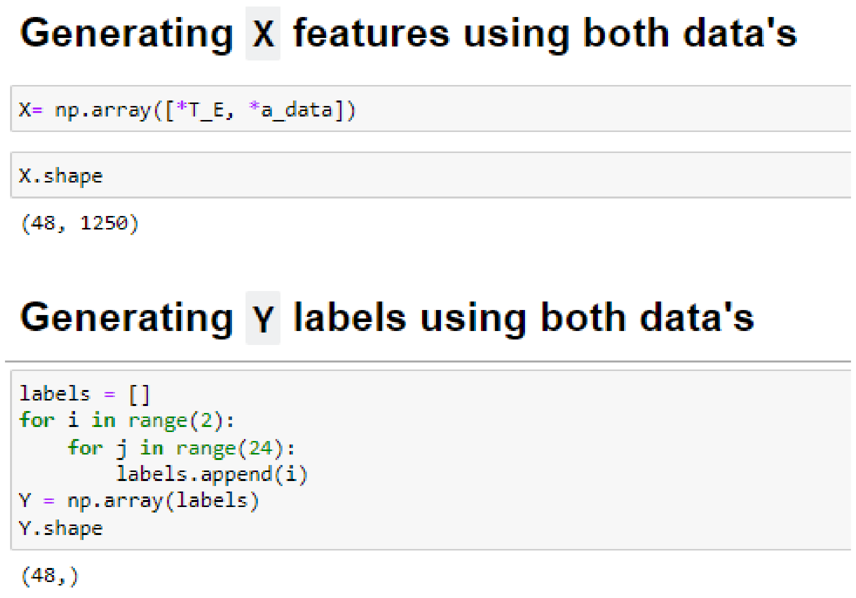
Figure 8: Function to read data from folder

abnormal_data= get_data(['I21'])

Patient I21 has following diseases: PVCs, APCs, atrial couplets, blocked APCs.

8. Assigning features to X and labels to Y

After reading the data features are assigned to X and labels are assigned to Y.



Generating X features using both data's

```
X= np.array([*T_E, *a_data])
```

```
X.shape
```

```
(48, 1250)
```

Generating Y labels using both data's

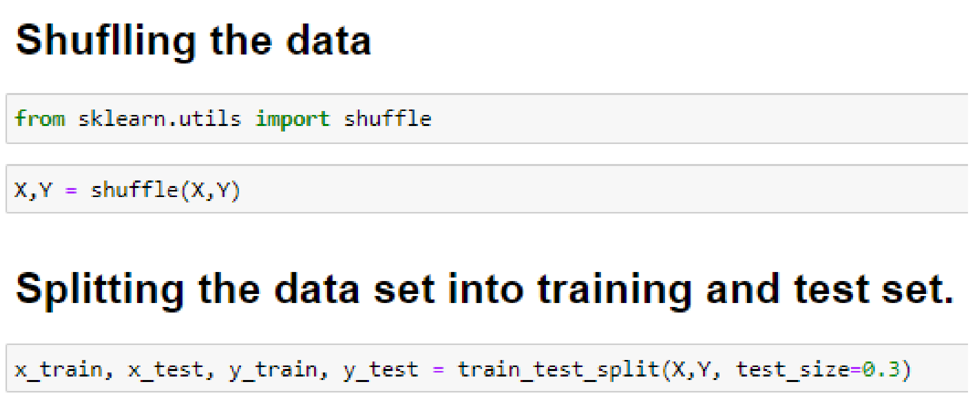
```
labels = []
for i in range(2):
    for j in range(24):
        labels.append(i)
Y = np.array(labels)
Y.shape
```

```
(48,)
```

Figure 9: Making X and Y.

9. Shuffling and Splitting Data

The X and Y are shuffled and then splitted into train and test data set.



Shuflling the data

```
from sklearn.utils import shuffle
```

```
X,Y = shuffle(X,Y)
```

Splitting the data set into training and test set.

```
x_train, x_test, y_train, y_test = train_test_split(X,Y, test_size=0.3)
```

10. Model Details:

i. Architecture of Neural Network Model:

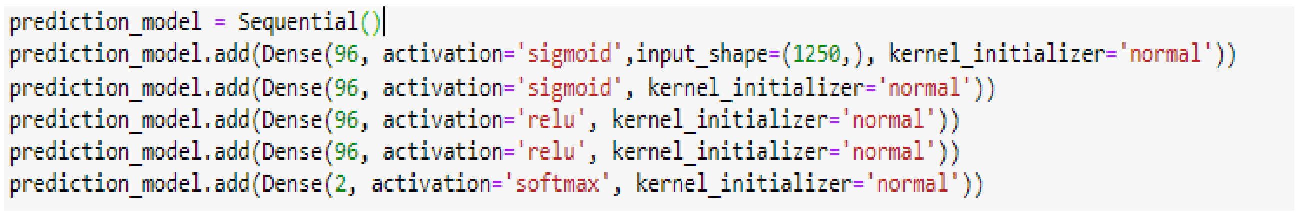
The neural network model used in this case is composed of two architectures:

1) Model 1 without regularization and dropout
2) Model 2 with regularization and dropout

Both models have **01** input layer, **03** hidden layers and **01** output layer.

The activation function in the layers is “**Relu**” and Sigmoid. In the last layer, softmax is used and 02 outputs are acquired because 02 classes are detected in this project. Also, regularization and dropout is also used in Model 2 to prevent overfitting.

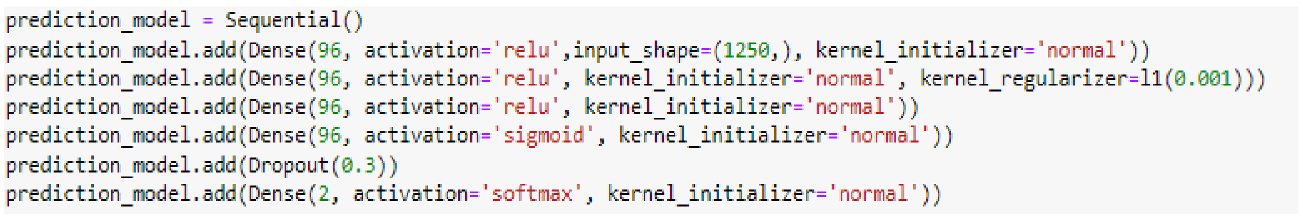
Model 1:



```
prediction_model = Sequential()
prediction_model.add(Dense(96, activation='sigmoid',input_shape=(1250,), kernel_initializer='normal'))
prediction_model.add(Dense(96, activation='sigmoid', kernel_initializer='normal'))
prediction_model.add(Dense(96, activation='relu', kernel_initializer='normal'))
prediction_model.add(Dense(96, activation='relu', kernel_initializer='normal'))
prediction_model.add(Dense(2, activation='softmax', kernel_initializer='normal'))
```

Figure 10: Neural Network Model 1

Model 2:



```
prediction_model = Sequential()
prediction_model.add(Dense(96, activation='relu',input_shape=(1250,), kernel_initializer='normal'))
prediction_model.add(Dense(96, activation='relu', kernel_initializer='normal', kernel_regularizer=l1(0.001)))
prediction_model.add(Dense(96, activation='relu', kernel_initializer='normal'))
prediction_model.add(Dense(96, activation='sigmoid', kernel_initializer='normal'))
prediction_model.add(Dropout(0.3))
prediction_model.add(Dense(2, activation='softmax', kernel_initializer='normal'))
```

Figure 11: Neural Network Model 2

ii. Training the model:

The model 1 and 2 are trained on the training set and accuracy is check on both training and validation data set. The "Adam" optimizer is used in this model and loss is calculated using cross entropy loss method already embedded in tensor flow and keras. The model training details are given below:

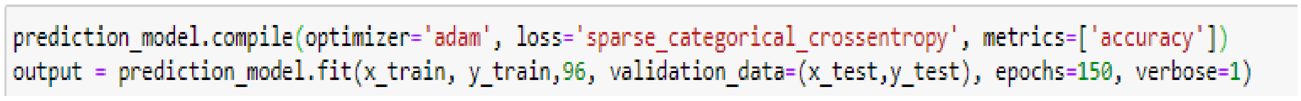
Optimizer= **Adam**

Loss= **Cross Entropy Loss**

Number of Epochs= **150**

Regularization parameter **= 0.001**

Drop out parameter **= 0.3**

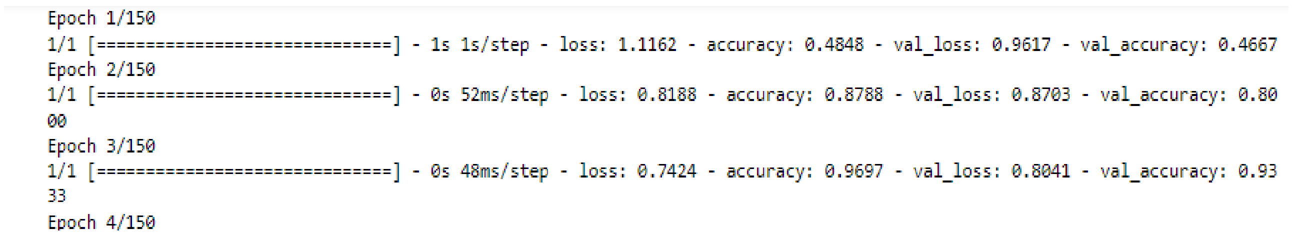


```
prediction_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
output = prediction_model.fit(x_train, y_train,96, validation_data=(x_test,y_test), epochs=150, verbose=1)
```

Figure 12: Model Compiling and Fitting

11. Output of the Model:

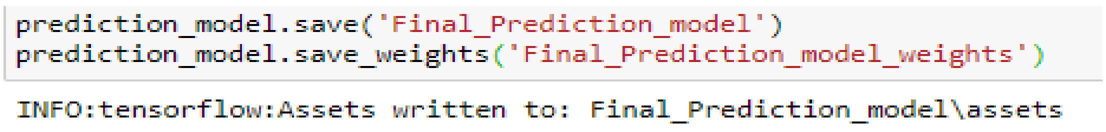
When the model is fitted, the output is given by the model in terms of accuracy, loss and validation accuracy which are plotted to visualize the output.



```
Epoch 1/150
1/1 [==============================] - 1s 1s/step - loss: 1.1162 - accuracy: 0.4848 - val_loss: 0.9617 - val_accuracy: 0.4667
Epoch 2/150
1/1 [==============================] - 0s 52ms/step - loss: 0.8188 - accuracy: 0.8788 - val_loss: 0.8703 - val_accuracy: 0.80
00
Epoch 3/150
1/1 [==============================] - 0s 48ms/step - loss: 0.7424 - accuracy: 0.9697 - val_loss: 0.8041 - val_accuracy: 0.93
33
Epoch 4/150
```

Figure 13: Model Output

The weights of the model are saved using the tensor flow and then used later in the prediction code.



```
prediction_model.save('Final_Prediction_model')
prediction_model.save_weights('Final_Prediction_model_weights')
```

```
INFO:tensorflow:Assets written to: Final_Prediction_model\assets
```

12. Plots/Results

The plots of both models are shown below:

- Number of Epochs Vs Loss:

The loss or cost should decrease as the number of epochs increases. It is shown in the fig:

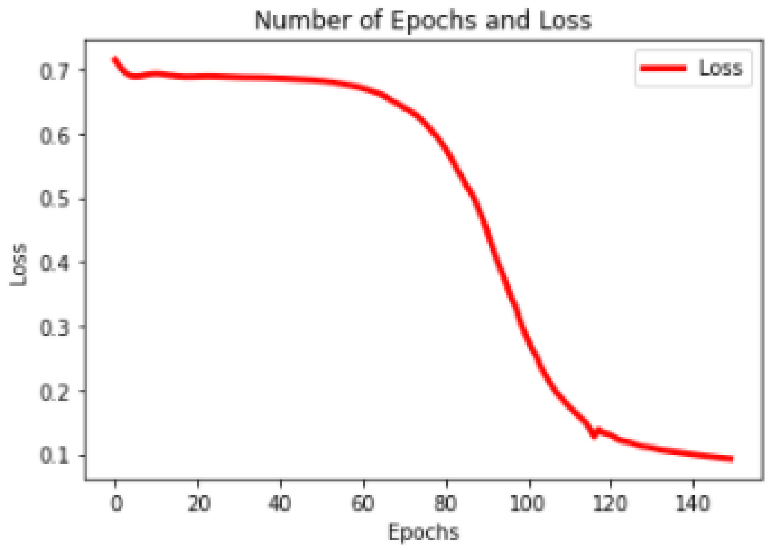


Figure 14: Loss Vs Epoch (Model 1)

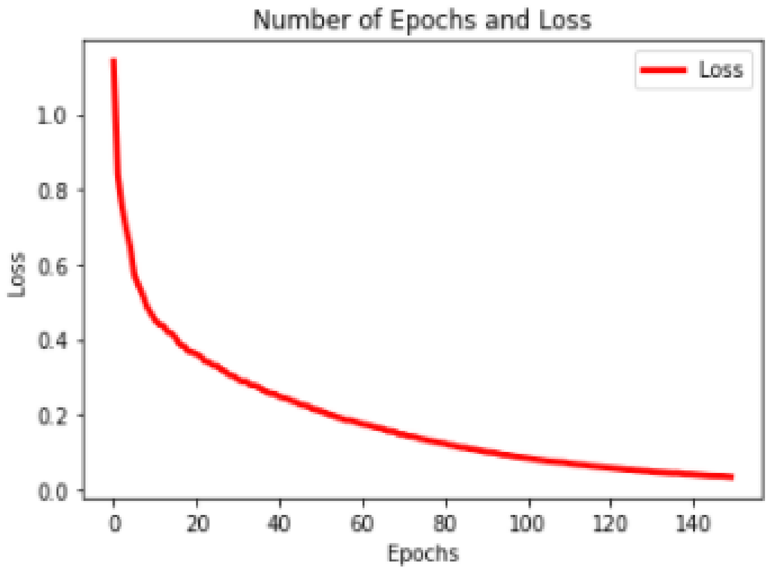


Figure 15: Loss Vs Epoch (Model 2)

- Number of Epochs Vs Validation Set Accuracy:

The graph between number of epochs and validation accuracy is shown in fig:

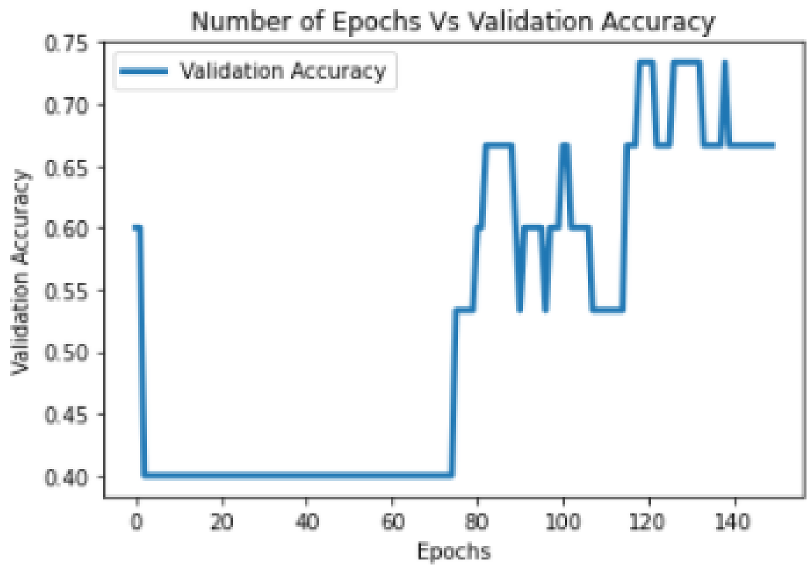


Figure 16: Number of Epochs Vs Validation Accuracy (Model 1)

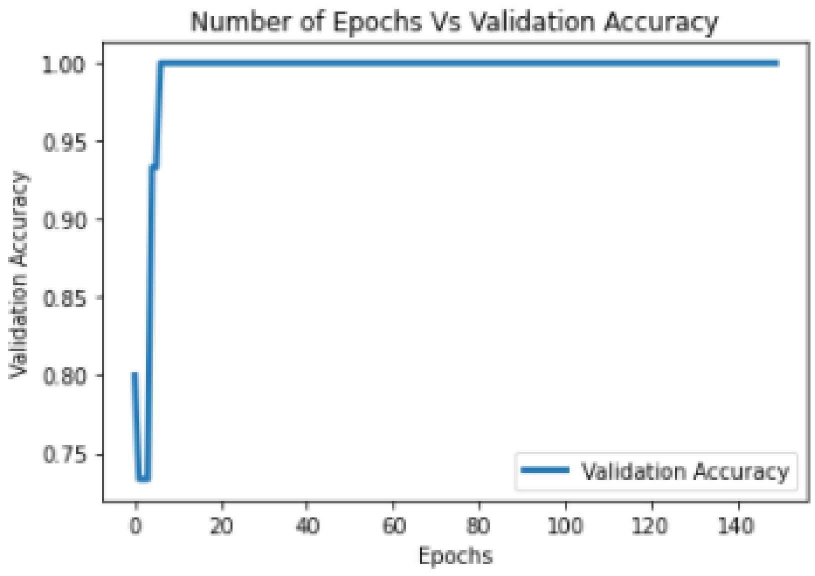


Figure 17: Number of Epochs Vs Validation Accuracy (Model 2)

- Number of Epochs Vs Training Set Accuracy:

The accuracy of this model for 02 classes normal and abnormal is above 95% which is considered as good accuracy.

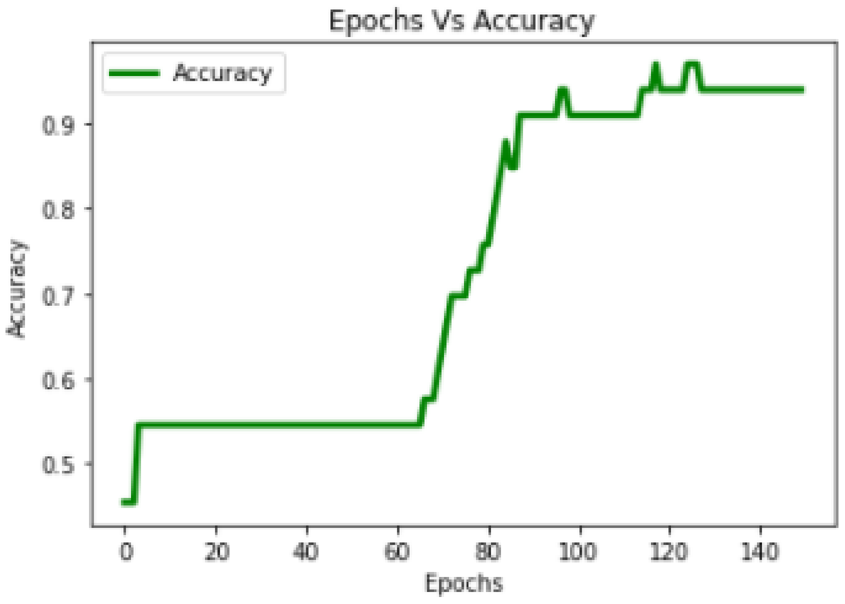


Figure 18: Training Accuracy Vs No of Epochs (Model 1)

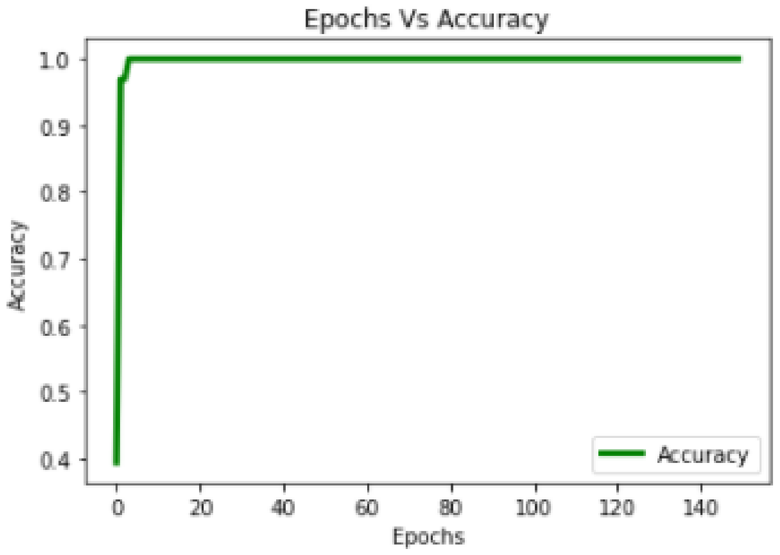


Figure 19: Training Accuracy Vs No of Epochs (Model 2)

13. Conclusion

The problem statement of the project is successfully completed. Training and validation accuracy is above 90% which is quite good and can also be further improved by improving the neural network model or by applying more deep learning algorithms like LSTM+RNN.

In this project, python libraries like wfdb used for reading and analyzing signals has also been used which increased the learning aspect as well. The hardware designing part and code for acquiring signal from ECG signal was challenging but was successfully done by the team.

14. References:

[1] "St Petersburg INCART 12-lead Arrhythmia Database v1.0.0", Physionet.org, 2021. [Online]. Available: https://physionet.org/content/incartdb/1.0.0/. [Accessed: 11- Nov- 2021].

[2] M. Ashfaq Khan and Y. Kim, "Cardiac Arrhythmia Disease Classification Using LSTM Deep Learning Approach", Computers, Materials & Continua, vol. 67, no. 1, pp. 427-443, 2021. Available: 10.32604/cmc.2021.014682 [Accessed 12 November 2021].

[3] "ECG Prediction ~94 Accuracy", Kaggle.com, 2021. [Online]. Available: https://www.kaggle.com/salikhussaini49/ecg-prediction-94-accuracy/data. [Accessed: 09- Dec- 2021].

[4] A. Isin and S. Ozdalili, "Cardiac arrhythmia detection using deep learning", *Procedia Computer Science*, vol. 120, pp. 268-275, 2017. Available: 10.1016/j.procs.2017.11.238 [Accessed 16 December 2021].

[5] "Understating ECG Sensors and How to Program one to Diagnose Various Medical Conditions", Circuit Digest, 2022. [Online]. Available: https://circuitdigest.com/microcontroller-projects/understanding-ecg-sensor-and-program-ad8232-ecg-sensor-with-arduino-to-diagnose-various-medical-conditions. [Accessed: 25- Dec- 2021].

[6] "AD8232 ECG Sensor : Pin Configuration, Features and Its Applications", ElProCus - Electronic Projects for Engineering Students, 2022. [Online]. Available: https://www.elprocus.com/ad8232-ecg-sensor-working-and-its-applications/#:~:text=The%20AD8232%20ECG%20sensor%20is,AD8232%20chip%20can%20be%20used. [Accessed: 15- Dec- 2021].

Annex A

Instructions on running the code:

Complete notebook with all the optimal parameters is provided with the project folder. Just place the data in the root directory of the jupyter notebook and run the book as per given steps.

“Code for MDR_Project-II_ECG_Group”.ipynb is the complete notebook.

Annex B

Instructions on running the code for Arduino:

The code which is run on arduino for aquiring the signal is given below. Just run the code on Arduino application on PC and get the serial plotter for visualizing the signal. For saving signal, the serial plotter is closed and data is saved as a .txt file.

```
/*
 * VARIABLES
 * count: variable to hold count of rr peaks detected in 10 seconds
 * flag: variable that prevents multiple rr peak detections in a single heatbeat
 * hr: HeartRate (initialised to 72)
 * hrv: Heart Rate variability (takes 10-15 seconds to stabilise)
 * instance1: instance when heart beat first time
 * interval: interval between second beat and first beat
 * timer: variable to hold the time after which hr is calculated
 * value: raw sensor value of output pin
 */
long instance1=0, timer;
double hrv =0, hr = 72, interval = 0;
int value = 0, count = 0;
bool flag = 0;
#define shutdown_pin 10
#define threshold 100 // to identify R peak
#define timer_value 10000 // 10 seconds timer to calculate hr

void setup() {
  Serial.begin(9600);
  pinMode(8, INPUT); // Setup for leads off detection LO +
  pinMode(9, INPUT); // Setup for leads off detection LO -
}
void loop() {
  if((digitalRead(8) == 1)||(digitalRead(9) == 1)){
    Serial.println("leads off!");
    digitalWrite(shutdown_pin, LOW); //standby mode
    instance1 = micros();
    timer = millis();
  }
  else {
    digitalWrite(shutdown_pin, HIGH); //normal mode
    value = analogRead(A0);
    value = map(value, 250, 400, 0, 100); //to flatten the ecg values a bit
    if((value > threshold) && (!flag)) {
      count++;
      Serial.println("in");
      flag = 1;
      interval = micros() - instance1; //RR interval
      instance1 = micros();
    }
    else if((value < threshold)) {
      flag = 0;
    }
    if ((millis() - timer) > 10000) {
      hr = count*6;
      timer = millis();
      count = 0;
    }
    hrv = hr/60 - interval/1000000;
    Serial.print(hr);
    Serial.print(",");
    Serial.print(hrv);
    Serial.print(",");
    Serial.println(value);
    delay(1);
  }
}
```