References

[1] M. H. Alomari, A. AbuBaker, A. Turani, A. M. Baniyounes, and A. Manasreh, “EEG Mouse: A Machine Learning-Based Brain Computer Interface,” *Int. J. Adv. Comput. Sci. Appl.*, vol. 5, no. 4, 2014.

[2] T. Publication{“id”:3775225, first_name“:”TJPRC, last_name“:”Publication, page_name“:”TJPRCPublication, domain_name“:”independent, “display_name”:"TJPRC Publication", url“:”http://independent.academia.edu/TJPRCPublication, “is_analytics_public”:false, photo“:”/images/s65_no_pic_borderless.gif, “interests”:{“top”:[], and “count”:0}}, “BRAIN CONTROLLED WHEELCHAIR FOR DISABLED.” [Online]. Available: https://www.academia.edu/6986487/BRAIN_CONTROLLED_WHEELCHAIR_FOR_DISABLED. [Accessed: 12-Aug-2014].

[3] R. Bogacz, U. Markowska-Kaczmar, and A. Kozik, “Blinking artefact recognition in EEG signal using artificial neural network,” in *Proc. of 4 th Conference on Neural Networks and Their Applications, Zakopane (Poland)*, 1999.

[4] “Community: The Mastermind Project-MIND CONTROLLED ROBOTS USING EEG - National Instruments.” [Online]. Available: https://decibel.ni.com/content/docs/DOC-24767. [Accessed: 12-Aug-2014].

[5] F. Ebrahimi, M. Mikaeili, E. Estrada, and H. Nazeran, “Automatic sleep stage classification based on EEG signals by using neural networks and wavelet packet coefficients,” in *Engineering in Medicine and Biology Society, 2008. EMBS 2008. 30th Annual International Conference of the IEEE*, 2008, pp. 1151–1154.

[6] C.-K. A. Lim and W. C. Chia, “Analysis of Single-Electrode EEG Rhythms Using MATLAB to Elicit Correlation with Cognitive Stress.”

[7] E. Niedermeyer and F. H. Lopes da Silva, *Electroencephalography: basic principles, clinical applications, and related fields*. Philadelphia: Lippincott Williams & Wilkins, 2005.

[8] G. V. Sridhar and P. M. Rao, “A Neural Network Approach for EEG Classification in BCI.”

[9] R. Singla and N. Sharma, “Function Classification of EEG Signals Based on ANN.”

[10] W. SA\LABUN, “Processing and spectral analysis of the raw EEG signal from the MindWave,” *Przeglad Elektrotechniczny*, vol. 90, pp. 169–174, 2014.

[11] G. Navalyal and R. D. Gavas, “A Dynamic Approach to Foster Cognitive Computing using the Laws of Thought.”

[12] S. Rodrak and Y. Wongsawat, “Dept. of Biomed. Eng., Mahidol Univ., Bangkok, Thailand,” in *Signal & Information Processing Association Annual Summit and Conference (APSIPA ASC), 2012 Asia-Pacific*, 2012, pp. 1–4.

[13] N.-H. Liu, C.-Y. Chiang, and H.-C. Chu, “Recognizing the Degree of Human Attention Using EEG Signals from Mobile Sensors,” *Sensors*, vol. 13, no. 8, pp. 10273–10286, Aug. 2013.

[14] “855-classification-of-electroencephalogram-using-artificial-neural-networks.pdf.” .

[15] E. A. Larsen, “Classification of EEG Signals in a Brain-Computer Interface System,” 2011.

[16] https://en.wikipedia.org/wiki/10-20_system_(EEG)

Appendix A

Data Acquisition using NeuroSky Headset

A.1 Introduction

It all start with acquiring data from NeuroSky headset. We need to find a way to get the EEG data from NeuroSky headset. We need to understand the formats and conventions used by NeuroSky headset first.

A.2 NeuroSky Mobile Mindwave Headset

ThinkGear is the technology inside every NeuroSky product or partner product that enables the device to interface with the wearers' brainwaves. It includes:

- The sensor that touches the forehead,
- The contact and reference points located on the ear pad, and
- The onboard chip that processes all of the data.
- Both the raw brainwaves and the eSense Meters are calculated on the ThinkGear™ chip. The calculated values are output by the ThinkGear chip, through the headset, to a PC.

Types of data output from ThinkGear chips:

- Raw sampled wave values (128Hz or 512Hz, depending on hardware)
- Signal poor quality metrics
- EEG band power values for delta, theta, alpha, beta, and gamma

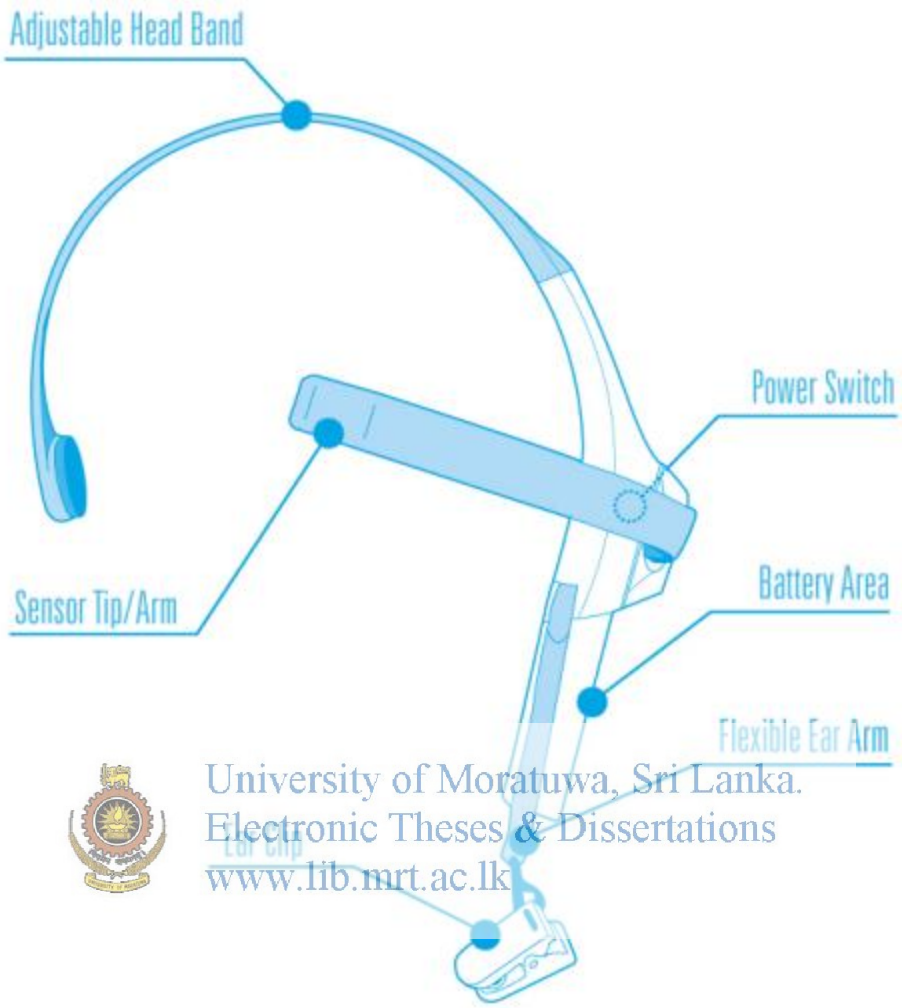
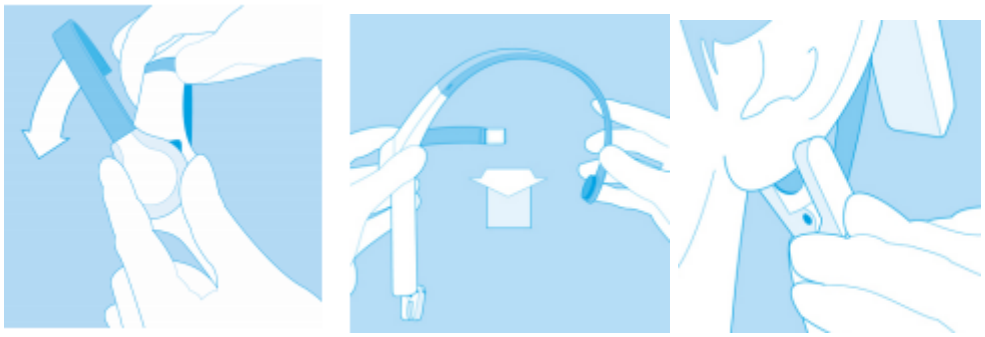


Figure A.1: NeuroSky Headset

To power on the MindWave Mobile headset, slide the switch to the ON (middle) position. When held past the ON position for 3 seconds and then released, the headset will enter Bluetooth pairing mode. If instead the switch is held past the ON position for 6 seconds, the headset's pairing memory will be cleared. While the MindWave Mobile headset is powered on, the LED light on the side of the headset will be turned on. If the MindWave has a low battery, the LED light will flash to indicate low battery status. To turn the MindWave Mobile off, slide the switch back to the OFF position.

A.3 Fitting NeuroSky Mobile Mindwave Headset

- Orient the MindWave with the forehead Sensor Arm on your left hand side. Rotate the Sensor Arm from its base by about 90 degrees. It can be rotated slightly more if necessary to get proper fit and comfort.



- The overhead band of the MindWave is adjustable and can be extended to fit various sizes. Put on the MindWave. If the sensor does not make contact with the forehead or if the fit is not comfortable, remove the MindWave to readjust the overhead band and the forehead Sensor Arm.

- Make sure the two metal contacts on the inside of both sides of the ear clip make skin contact with your earlobe or ear. Move any hair or obstructions (such as jewelry) out of the way. Readjust the ear clip as necessary to make proper contact with the skin of your ear. You may need to squeeze the ear clip against your ear for a few seconds.

A.4 ThinkGear Technology

The electrical signals emitted by neurons generating in the brain. These patterns and frequencies of these electrical signals can be measured by placing a sensor on the scalp. The Mind Tools line of headset products contain NeuroSky ThinkGear™ technology, which measures the analog electrical signals, commonly referred to as brainwaves, and processes them into digital signals. The ThinkGear technology then makes those measurements and signals available to games and applications. The table below gives a general synopsis of

some of the commonly-recognized frequencies that tend to be generated by different types of activity in the brain:

Brainwave Type	Frequency range	Mental states and conditions
Delta	0.1Hz to 3Hz	Deep, dreamless sleep, non-REM sleep, unconscious
Theta	4Hz to 7Hz	Intuitive, creative, recall, fantasy, imaginary, dream
Alpha	8Hz to 12Hz	Relaxed (but not drowsy) tranquil, conscious
Low Beta	12Hz to 15Hz	Formerly SMR, relaxed yet focused, integrated
Midrange Beta	16Hz to 20Hz	Thinking, aware of self & surroundings
High Beta	21Hz to 30Hz	Alertness, agitation

Figure A.2: Brainwave Types, frequencies and mental conditions map

ThinkGear Data Types

POOR_SIGNAL/SENSOR_STATUS

This is integer value provides an indication of how good or how poor the bio-signal is at the sensor. This value is typically output by all ThinkGear hardware devices once per second. This is an extremely important value for any app using ThinkGear sensor hardware to always read, understand, and handle. Depending on the use cases for your app and users, your app may need to alter the way it uses other data values depending on the current value of POOR_SIGNAL/SIGNAL_STATUS.

For example, if this value is indicating that the bio-sensor is not currently contacting the subject, then any received RAW_DATA or EEG_POWER values during that time should be treated as noise not from a human subject, and possibly discarded based on the needs of the app.

Poor signal may be caused by a number of different things. In order of severity, they are:

- Sensor, ground, or reference electrodes not being on a person's head/body
- Poor contact of the sensor, ground, or reference electrodes to a person's skin
- Excessive motion of the wearer

• Excessive environmental electrostatic noise (some environments have strong electric signals or static electricity buildup in the person wearing the sensor).

• Excessive biometric noise (i.e. unwanted EMG, EKG/ECG, EOG, EEG, etc. signals)

RAW_DATA

This data type supplies the raw sample values acquired at the bio-sensor. The sampling rate (and therefore output rate), possible range of values, and interpretations of those values (conversion from raw units to volt) for this data type are dependent on the hardware characteristics of the ThinkGear hardware device performing the sampling. You must refer to the documented development specs of each type of ThinkGear hardware that your app will support for details.

As an example, the majority of □inkGear devices sample at 512Hz, with a possible value range of -32768 to 32767.

As another example, to convert TGAT-based EEG sensor values (such as TGAT, TGAM, MindWave Mobile, MindWave, MindSet) to voltage values, use the following conversion:

(rawValue * (1.8/4096)) / 2000

EEG_POWER

This Data Value represents the current magnitude of 8 commonly-recognized types of EEG frequency bands.

The eight EEG powers are: delta (0.5 - 2.75Hz), theta (3.5 - 6.75Hz), low-alpha (7.5 - 9.25Hz), high-alpha (10 - 11.75Hz), low-beta (13 - 16.75Hz), high-beta (18 - 29.75Hz), low-gamma (31 - 39.75Hz), and mid-gamma (41 - 49.75Hz).

These values have no units and are only meaningful for comparison to the values for the other frequency bands within a sample. By default, output of this Data Value is enabled, and it is output approximately once a second.

A.5 ThinkGear Communication Driver

As per this research we are using ThinkGear Communication Driver shipped with NeuroSky Headset. It does not have fancy API like the .NET version. So we have listen to a port where the data are transferred and manipulate the data.

A.6 Android Application Development with NeuroSky MindWave Mobile

- Open the Settings app on the Android device
- Navigate to Wireless and network and enable Bluetooth if not already enabled
- Go to Bluetooth settings
- Power on the MindWave Mobile
- MindWave Mobile will show up in the list of devices
- Touch MindWave Mobile and pairing will complete automatically

Appendix B

FocusGain Application

B.1 Introduction

This will walk you through the procedure involve with developing Android application for the NeuroSky headset. This is the final output of the project, we named the application as "FocusGain"

B.2 FocusGain Android Application Design

Figure B.1: FocusGain Android Application

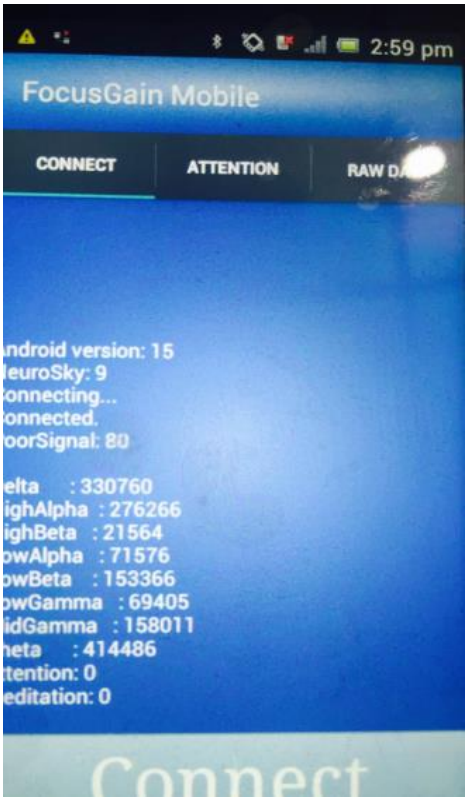


Figure B.2: FocusGain Android Application – Connect Pane

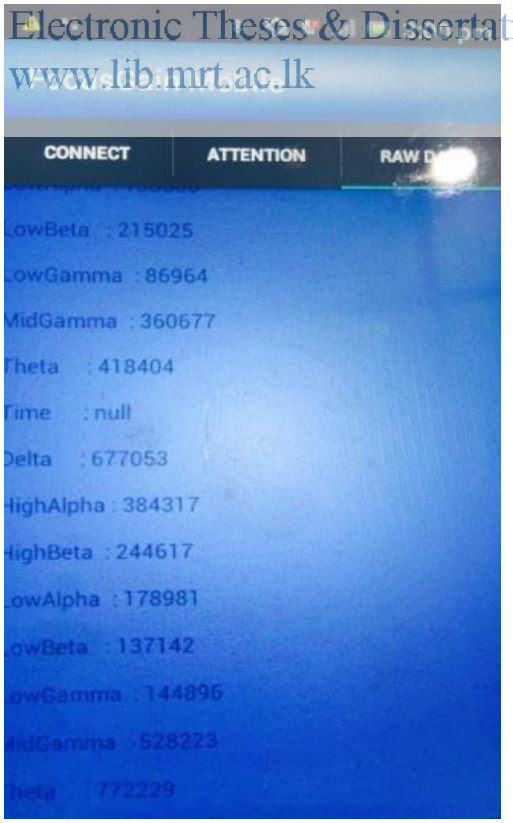


Figure B.3: FocusGain Android Application – Raw Data Pane

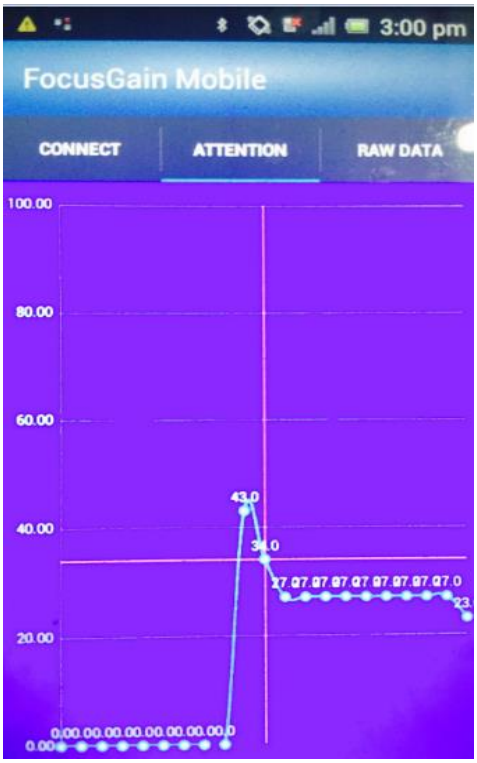


Figure B.4: FocusGain Android Application – Attention Monitoring

Appendix C

Code - FocusGain Application

C.1 Introduction

This section will include code used to develop this application.

C.2 Code for ThinkGear Connection

ThinkGearSocketClient.java

```
package focusgain.eeg.thinkgear;


import java.io.IOException;
import java.net.InetSocketAddress;
import java.nio.CharBuffer;
import java.nio.channels.SocketChannel;
import java.nio.charset.Charset;
import java.nio.charset.CharsetEncoder;
import java.util.Scanner;
import javax.swing.JOptionPane;
import org.apache.log4j.Logger;

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
/**
 *
 * @author Charitha Senarathne
 */
public class ThinkGearSocketClient
{

    /**
     * Logger for this class
     */
    private static final Logger logger = Logger.getLogger(ThinkGearSocketClient.class);

    public static final String DEFAULT_HOST = "127.0.0.1";
    public static final int DEFAULT_PORT = 13854;
```

```
private static ThinkGearSocketClient INSTANCE = null;

private String host;
private int port;
private boolean connected;
SocketChannel channel;
Scanner in;

/**
 * Default constructor using Thinkgear default host/port
 */
private ThinkGearSocketClient()
{

   this.host = DEFAULT_HOST;
   this.port = DEFAULT_PORT;
   this.connected = false;

}

/**
 * Constructor
 *
 * @param host
 * @param port
 */
public ThinkGearSocketClient(String host, int port)
{

   this.host = host;
   this.port = port;
   this.connected = false;

}

public static ThinkGearSocketClient getInstance()
{
   if(INSTANCE == null)
   {
      INSTANCE = new ThinkGearSocketClient();
   }

   return INSTANCE;
}
```

```
    public String getHost()
    {
        return host;
    }

    public void setHost(String host)
    {
        this.host = host;
    }

    public int getPort()
    {
        return port;
    }

    public void setPort(int port)
    {
        this.port = port;
    }

    public boolean isConnected()
    {
        return this.connected;
    }

    public void connect(boolean enableRawData) throws IOException
    {

        if (!this.connected)
        {
            logger.debug("connect() - Starting new connection...");
            this.channel = SocketChannel.open(new InetSocketAddress(this.host, this.port));

            CharsetEncoder enc = Charset.forName("US-ASCII").newEncoder();
            //String jsonCommand = "{\"timestamp\":true,\"enableRawOutput\":" +
Boolean.toString(enableRawData) + ", \"format\": \"Json\"}\n";
            String jsonCommand = "{\"timestamp\":true,\"enableRawOutput\": false, \"format\":
\"Json\"}\n";
            this.channel.write(enc.encode(CharBuffer.wrap(jsonCommand)));

            this.in = new Scanner(channel);
            this.connected = true;

            System.out.println("Connected");
        } else
```

```
        {
            logger.debug("connect() - Already connected...");
            System.out.println("Already connected");
        }

    }

    public void startRecording() throws IOException
    {
        if (this.connected)
        {
            CharsetEncoder enc = Charset.forName("US-ASCII").newEncoder();

            String jsonCommand =
"{\"startRecording\":{\"rawEeg\":true,\"poorSignalLevel\":true,\"eSense\":true,\"eegPower\":
true,\"blinkStrength\":true},\"applicationName\":\"ExampleApp\"}\n";
            this.channel.write(enc.encode(CharBuffer.wrap(jsonCommand)));

            JOptionPane.showMessageDialog(null, "Start Recording");

        } else
        {
            logger.debug("startRecording() - Not connected...");
        }
    }

    public void stopRecording() throws IOException
    {
        if (this.connected)
        {
            CharsetEncoder enc = Charset.forName("US-ASCII").newEncoder();

            String jsonCommand = "{\"stopRecording\":\"ExampleApp\"}\n";
            this.channel.write(enc.encode(CharBuffer.wrap(jsonCommand)));

            JOptionPane.showMessageDialog(null, "Stop Recording");

        } else
        {
            logger.debug("stopRecording() - Not connected...");
        }
    }

    public void cancelRecording() throws IOException
    {
```

```
        if (this.connected)
        {
            CharsetEncoder enc = Charset.forName("US-ASCII").newEncoder();

            String jsonCommand = "{\"cancelRecording\":\"ExampleApp\"}\n";
            this.channel.write(enc.encode(CharBuffer.wrap(jsonCommand)));

        } else
        {
            logger.debug("cancelRecording() - Not connected...");
        }
    }

    public boolean isDataAvailable()
    {
        if (this.connected)
        {
            return this.in.hasNextLine();
        } else
        {
            return false;
        }
    }

    public String getData()
    {
        return this.in.nextLine();
    }

    public void close() throws IOException
    {

        if (this.connected)
        {
            logger.debug("close() - Closing connection...");
            this.in.close();
            this.channel.close();
            this.connected = false;
        }
    }

}
```

C.3 Code for EEG Data Collector

EEGDataCollector.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package focusgain.eeg.read;

import focusgain.eeg.thinkgear.ThinkGearSocketClient;
import focusgain.eeg.ui.EEGDataCollectorUI;
import focusgain.eeg.ui.EEGVisualizer;
import static focusgain.eeg.ui.FocusGainUI.eegDataTxtArea;
import focusgain.eeg.utilities.TimerUtil;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JOptionPane;

/**
 *
 * @author Chartha Senarathne
 */
public class EEGDataCollector
{

   public static void initialiazeConnection()
   {

      new Thread(new Runnable()
      {
         @Override
         public void run()
         {
            while (!(EEGDataCollector.initialize()))
            {
               try
               {
```

```
                Thread.sleep(10 * 1000);

            } catch (InterruptedException ex)
            {

Logger.getLogger(EEGDataCollectorUI.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
      }
    }).start();
  }

  private static boolean initialize()
  {
    ThinkGearSocketClient client = ThinkGearSocketClient.getInstance();
    try
    {
      client.connect(EEGDataCollectorUI.rawDataCheckBox.isSelected());

    } catch (IOException ex)
    {
      return false;
    }
    return true;
  }

  public static void startSession()
  {
    TimerUtil.startTimer();
    EEGDataCollectorUI.eegDataTxtArea.append("Starting Session\n");
    writeToFile("EEG Sample Data for Attention");

    ThinkGearSocketClient client = ThinkGearSocketClient.getInstance();

    if (client != null)
    {
      while (client.isDataAvailable())
      {
        writeToFile(client.getData());

        String currentEEGReading = client.getData();

        if (currentEEGReading.contains("eSense"))
        {
```

```
EEGDataCollectorUI.eegDataTxtArea.append(currentEEGReading.substring(currentEEGReading.indexOf("eegPower")) + "\n");

EEGDataCollectorUI.eegDataTxtArea.setCaretPosition(EEGDataCollectorUI.eegDataTxtArea.getText().length());
            }
            else
            {
               EEGDataCollectorUI.eegDataTxtArea.append(client.getData() + "\n");

EEGDataCollectorUI.eegDataTxtArea.setCaretPosition(EEGDataCollectorUI.eegDataTxtArea.getText().length());
            }

         }
      }

   }

   public static void stopSession()
   {
      if (ThinkGearSocketClient.getInstance() != null)
      {
         EEGDataCollectorUI.eegDataTxtArea.append("Stoping Session\n");
         TimerUtil.stopTimer();

         try
         {
            ThinkGearSocketClient.getInstance().close();
         } catch (IOException ex)
         {
            Logger.getLogger(EEGDataCollector.class.getName()).log(Level.SEVERE, null, ex);
         }
      }

   }

   private static void writeToFile(String data)
   {
      FileWriter fileWriter = null;
      try
      {
         String fileName = EEGDataCollectorUI.sampleCaptionTxt.getText();
```

```
            if (fileName.equalsIgnoreCase(""))
            {
               DateFormat dateFormat = new SimpleDateFormat("yyyy_MM_dd_HH_mm_ss");
               Date date = new Date();
               fileName = dateFormat.format(date);

               EEGDataCollectorUI.sampleCaptionTxt.setText(fileName);
            }
            fileName = fileName + ".txt";
            File file = new File( fileName);

            //if file doesnt exists, then create it
            if (!file.exists())
            {
               file.createNewFile();
            }
            //true = append file

            fileWriter = new FileWriter(file.getName(), true);
            BufferedWriter bufferWriter = new BufferedWriter(fileWriter);
            String writeData = data + "\n";
            bufferWriter.write(writeData);
            bufferWriter.close();

         } catch (IOException ex)
         {
            Logger.getLogger(EEGDataCollector.class.getName()).log(Level.SEVERE, null, ex);

         } finally
         {
            try
            {
               fileWriter.close();
            } catch (IOException ex)
            {
               Logger.getLogger(EEGDataCollector.class.getName()).log(Level.SEVERE, null,
ex);
            }
         }
      }
   }
```

C.4 Code for ANN Trainer

AttentionClassificationNeuralNetwork.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package focusgain.eeg.ann;

import focusgain.eeg.ui.ANNTrainerUI;
import java.awt.Color;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import javax.swing.JOptionPane;
import org.encog.engine.network.activation.ActivationSigmoid;
import org.encog.neural.data.NeuralData;
import org.encog.neural.data.NeuralDataSet;
import org.encog.neural.data.basic.BasicNeuralData;
import org.encog.neural.data.basic.BasicNeuralDataSet;
import org.encog.neural.networks.BasicNetwork;
import org.encog.neural.networks.layers.BasicLayer;
import org.encog.neural.networks.layers.Layer;
import org.encog.neural.networks.synapse.Synapse;
import org.encog.neural.networks.synapse.WeightedSynapse;
import org.encog.neural.networks.training.Train;
import org.encog.neural.networks.training.propagation.back.Backpropagation;
import org.encog.neural.networks.training.propagation.resilient.ResilientPropagation;

/**
 *
 * @author charitha.s
 */
public class AttentionClassificationNeuralNetwork
{

    private BasicNetwork network;
```

```
    private int inputNeuronCount = 8;
    private int hiddenNeuronCount = 10;
    private int outputNeuronCount = 1;
    private double learnRate = 0.7;
    private double momentum = 0.8;
    private double error = 0.2;

    private static final int trainingSetSize = 60 * 60;
    private static final String sampleDataPath = "/Sample_EEG_DATA";
    private static final double attentiveBit = 1;
    private static final double notAttentiveBit = 0;
    private static final double ACCEPTED_SIGNAL_LEVEL = 150;

    public double INPUT[][] = new double[trainingSetSize][inputNeuronCount];
    public double IDEAL[][] = new double[trainingSetSize][outputNeuronCount];

    public void createNetwork()
    {
        setNetwork(new BasicNetwork());

        Layer outputLayer = new BasicLayer(new ActivationSigmoid(), true,
outputNeuronCount);
        Layer hiddenLayer = new BasicLayer(new ActivationSigmoid(), true,
hiddenNeuronCount);
        Layer inputLayer = new BasicLayer(new ActivationSigmoid(), false,
inputNeuronCount);

        Synapse synapseHiddenToOutput = new WeightedSynapse(hiddenLayer, outputLayer);
        Synapse synapseInputToHidden = new WeightedSynapse(inputLayer, hiddenLayer);

        hiddenLayer.addSynapse(synapseHiddenToOutput);
        inputLayer.addSynapse(synapseInputToHidden);

        getNetwork().tagLayer("INPUT", inputLayer);
        getNetwork().tagLayer("OUTPUT", outputLayer);

        getNetwork().getStructure().finalizeStructure();
        getNetwork().reset();
    }

    public boolean train()
    {
        boolean trainingStatus = false;
        final String sampleDataPathDir = "Sample_EEG_DATA";
        File sampleEEGDataPath = new File(sampleDataPathDir);
        if (sampleEEGDataPath.listFiles() != null)
```

```
        {
            for (final File fileEntry : sampleEEGDataPath.listFiles())
            {
                ANNTrainerUI.trainingDataSetList.setSelectedValue(fileEntry.getName(), true);
                ANNTrainerUI.trainingDataSetList.setSelectionForeground(Color.GREEN);

                if (fileEntry.isDirectory())
                {
                    continue;

                } else
                {
                    NeuralDataSet trainingSet = getTrainingSet(fileEntry.getAbsolutePath());
                    if (trainingSet != null)
                    {
                        if (!ANNTrainerUI.learningRateText.getText().equals(""))
                        {
                            learnRate =
Double.parseDouble(ANNTrainerUI.learningRateText.getText());
                        }
                        if (!ANNTrainerUI.momentumText.getText().equals(""))
                        {
                            momentum =
Double.parseDouble(ANNTrainerUI.momentumText.getText());
                        }
                        if (!ANNTrainerUI.errorText.getText().equals(""))
                        {
                            error = Double.parseDouble(ANNTrainerUI.errorText.getText());
                        }

                        trainingStatus = trainDataSet(trainingSet);

                    } else
                    {
                        trainingStatus = false;
                    }
                }
            }
        }
        return trainingStatus;
    }

    private boolean trainDataSet(NeuralDataSet trainingSet)
    {
        final Train train = new Backpropagation(getNetwork(), trainingSet, learnRate,
momentum);
```

```
        //final Train train = new ResilientPropagation(getNetwork(), trainingSet);
        int epoch = 0;
        do
        {
            //method is called over and over; each time the network is slightly adjusted for a
better error rate.
            //The following loop will loop and train the neural network until the error rate has
fallen below one percent.
            train.iteration();

            String resultLine = "Epoch #" + epoch + " Error:" + train.getError();
            ANNTrainerUI.resultTextArea.append(resultLine + "\n");

ANNTrainerUI.resultTextArea.setCaretPosition(ANNTrainerUI.resultTextArea.getText().len
gth());
            System.out.println(resultLine);
            epoch++;

            System.out.println("serializing the network");

            try
            {
                FileOutputStream fout = new
FileOutputStream("attention_classification_network.dat");
                ObjectOutputStream oos = new ObjectOutputStream(fout);
                oos.writeObject(getNetwork());
                oos.close();

            } catch (IOException e)
            {
                JOptionPane.showMessageDialog(null, e.getMessage());
            }
            System.gc();

        } while (train.getError() > error);
        System.gc();

        return true;
    }

    private NeuralDataSet getTrainingSet(String sampleDataFilePath)
    {
        NeuralDataSet neuralDataSet = null;
        BufferedReader br = null;
        try
        {
```

```
            br = new BufferedReader(new FileReader(sampleDataFilePath));

            int i = 0;
            String sCurrentLine;
            while ((sCurrentLine = br.readLine()) != null && i <= trainingSetSize)
            {
               if (sCurrentLine.contains("\"eSense\""))
               {
                  double attention = 0, meditation = 0, delta = 0, theta = 0, lowAlpha = 0,
highAlpha = 0, lowBeta = 0, highBeta = 0, lowGamma = 0, highGamma = 0,
poorSignalLevel = 0;
                  sCurrentLine = sCurrentLine.replace("\"", "");
                  sCurrentLine = sCurrentLine.replace("{", "");
                  sCurrentLine = sCurrentLine.replace("}", "");
                  sCurrentLine = sCurrentLine.replace("eSense:", "");
                  sCurrentLine = sCurrentLine.replace("eegPower:", "");

                  String[] eegValues = sCurrentLine.split(",");
                  for (String eegVal : eegValues)
                  {
                     String[] eegReadings = eegVal.split(":");
                     double tempReading = Double.parseDouble(eegReadings[1]);
                     switch (eegReadings[0])
                     {
                        case "attention":
                           attention = tempReading;
                           break;
                        case "meditation":
                           meditation = tempReading;
                           break;
                        case "delta":
                           delta = tempReading;
                           break;
                        case "theta":
                           theta = tempReading;
                           break;
                        case "lowAlpha":
                           lowAlpha = tempReading;
                           break;
                        case "highAlpha":
                           highAlpha = tempReading;
                           break;
                        case "lowBeta":
                           lowBeta = tempReading;
                           break;
                        case "highBeta":
```

```
                highBeta = tempReading;
                break;
            case "lowGamma":
                lowGamma = tempReading;
                break;
            case "highGamma":
                highGamma = tempReading;
                break;
            case "poorSignalLevel":
                poorSignalLevel = tempReading;
                break;
            default:
                break;
        }
    }

    if (poorSignalLevel < ACCEPTED_SIGNAL_LEVEL)
    {
        INPUT[i][0] = delta;
        INPUT[i][1] = theta;
        INPUT[i][2] = lowAlpha;
        INPUT[i][3] = highAlpha;
        INPUT[i][4] = lowBeta;
        INPUT[i][5] = highBeta;
        INPUT[i][6] = lowGamma;
        INPUT[i][7] = highGamma;

        for (int j = 0; j < outputNeuronCount; j++)
        {
            IDEAL[i][j] = attentiveBit;
        }
        ++i;
    }

  } else
  {
  }

}

neuralDataSet = new BasicNeuralDataSet(INPUT, IDEAL);
br.close();

} catch (IOException ex)
{
  JOptionPane.showMessageDialog(null, ex.getMessage());
```

```
            ANNTrainerUI.progressBar.setIndeterminate(false);
            ANNTrainerUI.trainBtn.setEnabled(true);

        }
         return neuralDataSet;
    }

    public boolean resumeTraining()
    {
        boolean trainingStatus = false;

        setNetwork(new BasicNetwork());
        System.out.println("deserializing the network");
        try
        {
            FileInputStream fin = new FileInputStream("attention_classification_network.dat");

            ObjectInputStream ois = new ObjectInputStream(fin);
            setNetwork((BasicNetwork) ois.readObject());
            ois.close();

        } catch (IOException | ClassNotFoundException e)
        {
            JOptionPane.showMessageDialog(null, e.getMessage());
        }

        System.out.println(getNetwork().toString());

        final String sampleDataPathDir = "Sample_EEG_DATA";
        File sampleEEGDataPath = new File(sampleDataPathDir);
        if (sampleEEGDataPath.listFiles() != null)
        {
            for (final File fileEntry : sampleEEGDataPath.listFiles())
            {
                ANNTrainerUI.trainingDataSetList.setSelectedValue(fileEntry.getName(), true);
                ANNTrainerUI.trainingDataSetList.setSelectionForeground(Color.GREEN);

                if (fileEntry.isDirectory())
                {
                    continue;

                } else
                {
                    NeuralDataSet trainingSet = getTrainingSet(fileEntry.getAbsolutePath());
                    if (trainingSet != null)
                    {
```

```
                if (!ANNTrainerUI.learningRateText.getText().equals(""))
                {
                   learnRate =
Double.parseDouble(ANNTrainerUI.learningRateText.getText());
                }
                if (!ANNTrainerUI.momentumText.getText().equals(""))
                {
                   momentum =
Double.parseDouble(ANNTrainerUI.momentumText.getText());
                }
                if (!ANNTrainerUI.errorText.getText().equals(""))
                {
                   error = Double.parseDouble(ANNTrainerUI.errorText.getText());
                }

                trainingStatus = trainDataSet(trainingSet);

             } else
             {
                trainingStatus = false;
             }
          }
       }
    }
    return trainingStatus;
 }


 public NeuralData computeOutput(double[] INPUT)
 {
    setNetwork(new BasicNetwork());
    System.out.println("deserializing the network");
    try
    {
       FileInputStream fin = new FileInputStream("attention_classification_network.dat");

       ObjectInputStream ois = new ObjectInputStream(fin);
       setNetwork((BasicNetwork) ois.readObject());
       ois.close();

    } catch (IOException | ClassNotFoundException e)
    {
       JOptionPane.showMessageDialog(null, e.getMessage());
    }

    NeuralData output = getNetwork().compute(new BasicNeuralData(INPUT));
```

```
        return output;
    }


    /**
     * @return the network
     */
    public BasicNetwork getNetwork()
    {
        return network;
    }

    /**
     * @param network the network to set
     */
    public void setNetwork(BasicNetwork network)
    {
        this.network = network;
    }
}
```


C.5 Code for FocusGain Mobile Application

ConnectFragment.java

```
package mobile.focusgain.focusgainmobile.fragments;

import android.bluetooth.BluetoothAdapter;
import android.content.Context;
import android.content.Intent;
import android.graphics.Color;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.os.Vibrator;
import android.support.annotation.Nullable;
import android.support.v4.app.Fragment;
import android.text.method.ScrollingMovementMethod;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.TextView;
```

```
import android.widget.Toast;

import com.neurosky.thinkgear.TGDevice;
import com.neurosky.thinkgear.TGEegPower;

import java.text.SimpleDateFormat;
import java.util.Date;

import mobile.focusgain.focusgainmobile.R;
import mobile.focusgain.focusgainmobile.thinkgear.ThinkGearConnector;

/**
 * Created by Charitha Senarathne on 5/4/2015.
 */
public class ConnectFragment extends Fragment implements View.OnClickListener {

    BluetoothAdapter bluetoothAdapter;
    TextView tv;
    Button b;
    TGDevice tgDevice;


    public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup container,
@Nullable Bundle savedInstanceState) {

        View view = inflater.inflate(R.layout.connect, container, false);

        tv = (TextView) view.findViewById(R.id.textView);
        tv.setMovementMethod(new ScrollingMovementMethod());
        tv.setText("");
        tv.append("Android version: " + Integer.valueOf(android.os.Build.VERSION.SDK) +
"\n");

        b = (Button) view.findViewById(R.id.button);
        b.setOnClickListener(this);

        // Check if Bluetooth is available on the Android device
        bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
        if (bluetoothAdapter == null) {

            // Alert user that Bluetooth is not available
            Toast.makeText(getActivity().getBaseContext(), "Bluetooth not available",
Toast.LENGTH_LONG).show();
            //finish();
        } else {
```

```
            // create the TGDevice
            tgDevice = new TGDevice(bluetoothAdapter, handler);
            ThinkGearConnector.setTgDevice(tgDevice);

            tv.append("NeuroSky: " + tgDevice.getVersion());
            tv.append("\n");
        }

        return view;
    }

    public void doStuff(View view) {

        if (ThinkGearConnector.getTGDevice() != null) {

            if (ThinkGearConnector.getTGDevice().getState() !=
TGDevice.STATE_CONNECTING && ThinkGearConnector.getTGDevice().getState() !=
TGDevice.STATE_CONNECTED) {

                ThinkGearConnector.getTGDevice().connect(rawEnabled);

            }
        } else {

            Toast.makeText(getActivity().getBaseContext(), "Bluetooth not available",
Toast.LENGTH_LONG).show();
        }
    }


    final boolean rawEnabled = false;
    int focusLostCount = 0;
    final Handler handler = new Handler() {
        @Override
        public void handleMessage(Message msg) {

//            SimpleDateFormat dateFormat = new SimpleDateFormat("dd-MMM-yy hh.mm.ss
aa");
//            Date date = new Date();
//            String formattedDate = dateFormat.format(date.toString());

            switch (msg.what) {
                case TGDevice.MSG_STATE_CHANGE:

                    switch (msg.arg1) {
                        case TGDevice.STATE_IDLE:
```

```
            break;
        case TGDevice.STATE_CONNECTING:
            tv.append("Connecting...\n");
            break;
        case TGDevice.STATE_CONNECTED:
            tv.append("Connected.\n");
            ThinkGearConnector.getTGDevice().start();
            break;
        case TGDevice.STATE_NOT_FOUND:
            tv.append("Could not connect any of the paired BT devices.\n");
            break;
        case TGDevice.STATE_NOT_PAIRED:
            tv.append("No Bluetooth devices paired.\n");
            break;
        case TGDevice.STATE_DISCONNECTED:
            tv.append("Disconnected.\n");
    } /* end switch on msg.arg1 */

    break;

case TGDevice.MSG_POOR_SIGNAL:
    tv.append("PoorSignal: " + msg.arg1 + "\n");
    break;
case TGDevice.MSG_HEART_RATE:
    tv.append("Heart rate: " + msg.arg1 + "\n");
    break;

case TGDevice.MSG_RAW_DATA:
    int rawValue = msg.arg1;
    tv.append("Raw Data: " + rawValue + "\n");
    break;

case TGDevice.MSG_MEDITATION:
    tv.append("Meditation: " + msg.arg1 + "\n");
    break;

case TGDevice.MSG_BLINK:
    tv.append("Blink: " + msg.arg1 + "\n");
    break;
case TGDevice.MSG_RAW_COUNT:
    //tv.append("Raw Count: " + msg.arg1 + "\n");
    break;

case TGDevice.MSG_LOW_BATTERY:
```

```
            Toast.makeText(getActivity().getApplicationContext(), "Low battery!",
Toast.LENGTH_SHORT).show();
            break;

          case TGDevice.MSG_EEG_POWER:
            TGEegPower ep = (TGEegPower) msg.obj;
            tv.append("\n"+
                    "Delta     : " + ep.delta + "\n" +
                    "HighAlpha  : " + ep.highAlpha + "\n" +
                    "HighBeta   : " + ep.highBeta +   "\n" +
                    "LowAlpha   : " + ep.lowAlpha +   "\n" +
                    "LowBeta    : " + ep.lowBeta +    "\n" +
                    "LowGamma   : " + ep.lowGamma +   "\n" +
                    "MidGamma   : " + ep.midGamma +   "\n" +
                    "Theta     : " + ep.theta + "\n");

            Intent dataIntent = new Intent();

dataIntent.setAction("mobie.focusgain.focusgainmobile.EEG_POWER_BANDS");

           dataIntent.putExtra("formattedDate", formattedDate);
           dataIntent.putExtra("hours", date.getHours());
           dataIntent.putExtra("minutes", date.getMinutes());
           dataIntent.putExtra("seconds", date.getSeconds());

            dataIntent.putExtra("Delta", Integer.toString(ep.delta));
            dataIntent.putExtra("HighAlpha", Integer.toString(ep.highAlpha));
            dataIntent.putExtra("HighBeta", Integer.toString(ep.highBeta));
            dataIntent.putExtra("LowAlpha", Integer.toString(ep.lowAlpha));
            dataIntent.putExtra("LowBeta", Integer.toString(ep.lowBeta));
            dataIntent.putExtra("LowGamma", Integer.toString(ep.lowGamma));
            dataIntent.putExtra("MidGamma", Integer.toString(ep.midGamma));
            dataIntent.putExtra("Theta", Integer.toString(ep.theta));
            getActivity().sendBroadcast(dataIntent);

            boolean attentive = new
ANNClassifier().recognizeAttention(getApplicationContext(),ep);
           Vibrator vibrator = (Vibrator)
getSystemService(Context.VIBRATOR_SERVICE);

            if(!attentive )
            {
               ++focusLostCount;
            }
            if( focusLostCount == 20)
            {
```

```
                tv.setBackgroundColor(Color.RED);

                // Vibrate for 500 milliseconds
                vibrator.vibrate(2000);
            }
            if( focusLostCount == 22)
            {
                tv.setBackgroundColor(Color.WHITE);
                vibrator.cancel();
                focusLostCount = 0;
            }

        default:
            break;

      } /* end switch on msg.what */

    } /* end handleMessage() */

  }; /* end Handler */

  @Override
  public void onClick(View v) {

    doStuff(v);
  }
}
```


ANNClassifier.java

```
package mobile.focusgain.focusgainmobile.ann;

import android.app.Activity;
import android.content.Context;

import com.neurosky.thinkgear.TGEegPower;

import org.encog.neural.data.NeuralData;
import org.encog.neural.data.basic.BasicNeuralData;
import org.encog.neural.networks.BasicNetwork;

import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
```

```
/**
 * Created by Charitha Senarathne on 4/28/2015.
 */
public class ANNClassifier {

    private BasicNetwork trainedANN;

    public boolean recognizeAttention( Context context, TGEegPower powerBands) {

        boolean attentiveStatus = false;
        if (getTrainedNetwork() != null) {
            loadTrianedANN(context);
        }

        double[] INPUT = new double[8];
        INPUT[0] = powerBands.delta;
        INPUT[1] = powerBands.theta;
        INPUT[2] = powerBands.lowAlpha;
        INPUT[3] = powerBands.highAlpha;
        INPUT[4] = powerBands.lowBeta;
        INPUT[5] = powerBands.highBeta;
        INPUT[6] = powerBands.lowGamma;
        INPUT[7] = powerBands.midGamma;

        NeuralData output = computeOutput(INPUT);

        double[] allData = output.getData();
        double outputReading = (double) Math.round(allData[0]);

        if (outputReading > 0.5) {
            attentiveStatus = true;
        } else {
            attentiveStatus = false;
        }

        return attentiveStatus;
    }

    public void loadTrianedANN(Context context) {

        setTrainedNetwork(new BasicNetwork());

        try {
```

```
            FileInputStream fin = (FileInputStream)
context.getAssets().open("attention_classification_network.dat");

            ObjectInputStream ois = new ObjectInputStream(fin);
            setTrainedNetwork((BasicNetwork) ois.readObject());
            ois.close();

        } catch (IOException | ClassNotFoundException e) {

        }
    }

    public NeuralData computeOutput(double[] INPUT) {
        NeuralData output = getTrainedNetwork().compute(new BasicNeuralData(INPUT));

        return output;
    }


    public BasicNetwork getTrainedNetwork() {
        return trainedANN;
    }

    public void setTrainedNetwork(BasicNetwork trainedANN) {
        this.trainedANN = trainedANN;
    }
}
```