Serious Games Development:
An Application for Mental Health Assessment

by
Group 19

Authors
Kyle Leduc-McNiven
Genico Melegrito
Belema Jack-Bara
Michael Wurtak

Academic Advisor
Dr. Robert McLeod

Date of Submission
March 12, 2018
Abstract

This Capstone design project addresses the role Serious Mobile Games may play in Mental Health. The games and extensions developed here build upon serious games WarCAT (War Cognitive Assessment Tool) and WarDOG (War Data on Gambling) that were initially proposed to help assess potential social gaming and gambling addictions. WarCAT and WarDOG are simplistic in nature to allow them to be accessible to an aging demographic. New game modes, such as the Race Mode, have been added into the app game to help further collect data for addiction assessment. We have built a system that utilizes the game applications and an array of biometric sensors that are used by an Arduino UNO to collect real-time patient data. Data collected from these sensors are stored into a database and is connected to the app game through serial communication. Collected data is converted into image form, analyzed, and correlated to identify notable features and then used to train a machine learning algorithm. Our machine learning algorithm is able to process new data and provide a medical assessment. As of the report period, two version of the games with several innovative features and data collection capabilities have been prototyped, integrated bio-sensing of concurrent temperature and heart rate monitoring has been demonstrated and initial forays into classification using machine learning have been undertaken.
Contributions

The design of the game and all of its feature was designed and implemented by Kyle Leduc-McNiven. Michael Wurtak and Belema (Jack) Jack-Bara were initially involved with the development of the machine learning algorithm but as a result of losing one team member Jack was assigned to the hardware section of the project. For the hardware section of the project Genico Melegrito designed the heartbeat and temperature sensors and Jack designed the bluetooth module that was used to connect the hardware to the software of the design. Mike continued the design and implementation of the machine learning algorithm. We would also like to give special thanks to Dr. Benedict Albensi and Dr. Tony Szturm for helping us with our research of dementia and serious games respectively. Finally we would also like to give thanks to our supervisor Dr. Robert McLeod, and our instructors Dr. Derek Oliver and Mr. Daniel Card for aiding and guiding us through this project.
## Contents

1 Introduction .................................................. 11

2 Software ................................................... 13  
   2.1 Design Criteria ........................................... 13  
   2.2 Subsection Design ........................................ 14  
      2.2.1 Serious Game Application ......................... 14  
      2.2.2 Server for Data Collection ....................... 15  
      2.2.3 Server for Games ................................... 16  
      2.2.4 Synthetic Data Creation ......................... 17  
      2.2.5 Race Mode ........................................... 18  

3 Hardware .................................................. 20  
   3.1 Design Criteria ........................................... 20  
   3.2 Subsection Design ........................................ 21  
      3.2.1 Microcontroller .................................... 21  
      3.2.2 Sensors .............................................. 21  
      3.2.3 Bluetooth Connection .............................. 22  
      3.2.4 Connections ......................................... 22  
      3.2.5 Housing ............................................... 23  
   3.3 Process ................................................... 24  
      3.3.1 Sensor Selection and Testing .................... 24  
      3.3.2 Hardware Integration .............................. 25  

4 Machine Learning ......................................... 25  
   4.1 Design Criteria ........................................... 25  
   4.2 Subsection Design ........................................ 26  
      4.2.1 Machine Learning Framework ..................... 26  
      4.2.2 System Architecture ................................ 26  
      4.2.3 Optimization Algorithms ......................... 29  
         4.2.3.1 Cost Function .................................. 30  
         4.2.3.2 Backpropagation ............................... 30
## 4.2.3.3 Fine-Tuning Functions

## 4.2.4 Data Analysis

## 5 System Integration, Results, and Evaluation

### 5.1 Software

#### 5.1.1 Greeting Screen

#### 5.1.2 Login and Register Screens

#### 5.1.3 Main Menu

#### 5.1.4 Store Screen

#### 5.1.5 Record Screen

#### 5.1.6 Options Screen

#### 5.1.7 Help Screen

#### 5.1.8 Game Screen

#### 5.1.9 Results Screen

### 5.2 Hardware

### 5.3 Machine Learning

## 6 Conclusion

## 7 Future Work

### 7.1 Software

#### 7.1.1 Software Distractions

#### 7.1.2 Accelerometer

#### 7.1.3 Additional Levels and Bot Strategies

#### 7.1.4 Neilson Heuristics

#### 7.1.5 In-App Bluetooth Connection

#### 7.1.6 WarDOG Application Changes

### 7.2 Hardware

#### 7.2.1 Blood Pressure Sensors

#### 7.2.2 Eye Tracking

#### 7.2.3 Hardware Distractions

### 7.3 Machine Learning
 CONTENTS

7.3.1 GAN ................................................................. 44
7.3.2 Layer-wise Visualization ................................. 45

8 References ................................................................. 46

A Appendix ................................................................. 49
A.1 Software ............................................................... 49
A.2 Hardware ............................................................... 97
A.3 Machine Learning ................................................... 100
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A mid-game screenshot of the WarCAT mobile application</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>Gambling Server Flow Chart</td>
<td>17</td>
</tr>
<tr>
<td>3</td>
<td>Results screen showcasing user scores in Race Mode</td>
<td>19</td>
</tr>
<tr>
<td>4</td>
<td>Layout of connections for the Arduino UNO and biometric sensors</td>
<td>23</td>
</tr>
<tr>
<td>5</td>
<td>Housing layout</td>
<td>24</td>
</tr>
<tr>
<td>6</td>
<td>LeNet-5, an example of a 6-layer CNN architecture</td>
<td>27</td>
</tr>
<tr>
<td>7</td>
<td>Detailed visualization of the convolution operation</td>
<td>28</td>
</tr>
<tr>
<td>8</td>
<td>Visual representation of the convolution and pooling operations</td>
<td>28</td>
</tr>
<tr>
<td>9</td>
<td>Updated filter weights using the deltas (Equation(7)) calculated during back-propagation</td>
<td>32</td>
</tr>
<tr>
<td>10</td>
<td>Visualization of the dropout function applied to a standard neural net model</td>
<td>33</td>
</tr>
<tr>
<td>11</td>
<td>Sample image representation of collected data for 32 games played. A scale bar has been included for color reference.</td>
<td>35</td>
</tr>
<tr>
<td>12</td>
<td>A look at how each layer detects different features of an image of a cat[1]</td>
<td>45</td>
</tr>
<tr>
<td>13</td>
<td>Greeting screen</td>
<td>49</td>
</tr>
<tr>
<td>14</td>
<td>Login Screen</td>
<td>50</td>
</tr>
<tr>
<td>15</td>
<td>Registration Screen</td>
<td>50</td>
</tr>
<tr>
<td>16</td>
<td>Information screen</td>
<td>51</td>
</tr>
<tr>
<td>17</td>
<td>Credits screen</td>
<td>51</td>
</tr>
<tr>
<td>18</td>
<td>Main menu screen for WarCAT</td>
<td>52</td>
</tr>
<tr>
<td>19</td>
<td>Record Screen</td>
<td>52</td>
</tr>
<tr>
<td>20</td>
<td>Leaderboard screen</td>
<td>53</td>
</tr>
<tr>
<td>21</td>
<td>Friends screen</td>
<td>54</td>
</tr>
<tr>
<td>22</td>
<td>Store screen</td>
<td>55</td>
</tr>
<tr>
<td>23</td>
<td>Options screen</td>
<td>55</td>
</tr>
<tr>
<td>24</td>
<td>Screen for changing card options</td>
<td>56</td>
</tr>
<tr>
<td>25</td>
<td>WarDOG game mode selection screen</td>
<td>57</td>
</tr>
<tr>
<td>26</td>
<td>Regular game mode screen</td>
<td>58</td>
</tr>
<tr>
<td>27</td>
<td>Final results screen after points calculation</td>
<td>58</td>
</tr>
<tr>
<td>28</td>
<td>Tournament selection screen</td>
<td>59</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>29</td>
<td>Betting screen</td>
<td>60</td>
</tr>
<tr>
<td>30</td>
<td>Tournament wait screen</td>
<td>61</td>
</tr>
<tr>
<td>31</td>
<td>Tournament play screen</td>
<td>62</td>
</tr>
<tr>
<td>32</td>
<td>Play style selection screen</td>
<td>63</td>
</tr>
<tr>
<td>33</td>
<td>Race game screen</td>
<td>64</td>
</tr>
<tr>
<td>34</td>
<td>Race game break screen</td>
<td>65</td>
</tr>
<tr>
<td>35</td>
<td>Hardware test setup</td>
<td>97</td>
</tr>
<tr>
<td>36</td>
<td>Console output of the heartbeat sensor</td>
<td>98</td>
</tr>
<tr>
<td>37</td>
<td>Console output of the temperature sensor</td>
<td>99</td>
</tr>
<tr>
<td>38</td>
<td>Arduino UNO Schematic</td>
<td>100</td>
</tr>
<tr>
<td>39</td>
<td>High-example of a potential GAN architecture</td>
<td>102</td>
</tr>
<tr>
<td>40</td>
<td>MNIST Tutorial 1: Output Visualization</td>
<td>125</td>
</tr>
<tr>
<td>41</td>
<td>MNIST Tutorial 2 - 3D Space Projected Output Visualization</td>
<td>130</td>
</tr>
<tr>
<td>42</td>
<td>MNIST Tutorial 2 - TensorBoard Output Results</td>
<td>130</td>
</tr>
</tbody>
</table>
# List of Tables

1. System Performance Metrics and Associated Target Specifications . . . . . 14
2. Target metrics for the biometric sensors . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
3. Summary of machine learning design specifications . . . . . . . . . . . . . . . . . . . . . . . . 26
4. Machine Learning Framework Software Comparison . . . . . . . . . . . . . . . . . . . . . . . . . . . . 100
5. System Architecture Comparison . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 101
6. Numerical Methods Comparison . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 101
7. Performance results of different architectures . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 102
Nomenclature

ANN       Artificial Neural Network
BPM       Beats Per Minute
CNN       Convolutional Neural Network
CPU       Central Processing Unit
DC        Direct Current
GAN       Generative Adversarial Network
Hz        Hertz
I/O       Input/Output
LED       Light Emitting Diode
mA        Milliampere
MCI       Mild Cognitive Impairment
ML        Machine Learning
MNIST     Modified Institute of Standards and Technology
MSE       Mean Squared Error
PvP       Player versus Player
ReLU      Rectified Linear Unit
UI        User Interface
USB       Universal Serial Bus
V         Voltage
WarCAT    War Cognitive Assessment Tool
WarDOG    War Data on Gambling
1 Introduction

The games and extensions developed here build upon serious games WarCAT and WarDOG that were initially proposed to help assess potential social gaming and gambling addictions. Early assessment of potential problems and subsequent intervention has the potential to improve the efficacy of both social as well as therapeutic treatment programs. As such, some of the work reported here addresses serious mobile games potential as empirical assessment tools for behavioral processes within the domains of social gaming/gambling. In particular, the development of a serious game that would be sufficiently compelling for a person to play. Along a similar line of reasoning, early on it was conjectured that these types of serious games may even be more effective in other areas associated with mHealth and in particular to assessing a persons cognitive health. Similar to most health related concerns, early recognition of mild cognitive impairment (MCI) and subtle changes to cognitive abilities that precede an MCI diagnosis also has the potential to improve the efficacy of social and therapeutic treatment programs. MCIs are an elusive beast as its symptoms can also be seen in normal aging. Because of this elusiveness rather than tackling MCI head on we have looked toward the later stages of dementia in Frontotemporal and vascular dementia for clues. If we can assess different types of dementia it will become possible over time to see trends in data and from that one could determine if a person has an MCI.

The basis of our project for MCI detection is predicated on the idea of emergent strategy. Emergent strategy is the view that strategy emerges over time as agents intentions collide with and accommodate a changing reality[2]. Emergent strategy can be seen as a set of actions over time that form a pattern which was not originally intended, i.e., a form of learning that emerges from practice. The basis for the addictions detection is the idea that while social gaming and gambling are conceptually different there is research coming out that shows that they have more in common than one might think. Unfortunately social gaming and online gambling are not all that well understood at the current moment, but it is possible to see that certain features in games have gambling esqe qualities. As more and more games come out, research in this area becomes more and more important.

The basic addiction game was repurposed to address its potential as an empirical as-
essment tool for cognitive processes within the domains of attention, recognition, recall, and memory applied to game strategy. Whether oriented to gambling addiction or MCI, both games provide players with immediate feedback but engage different algorithms and heuristics to solve the respective problems at hand. The Game, servers and data collection is what we covered in the software portion of this project using WarCAT and WarDOG as a medium for collecting the data in a way that can hold a user's interest. Our application functions is much like the card game war except that you are dealt a hand of cards like in 5-card poker. In addition like poker we allow for betting and tournament play.

Guided by early suggestions within the Capstone design program it was also suggested that a degree of sensor fusion would help correlate and interpret data collected by the game itself. Towards this end, integration of biosensors was undertaken. Specifically these include concurrent temperature and heart rate monitoring during game play. In addition to augmenting the games with bio-sensing, it is conjectured that collecting player data on large scales will allow for baseline establishment of cognitive abilities across demographic (age) profiles. The hardware for this project is composed of an Arduino UNO microcontroller that powers, programs, and collects the biometric data from the pulse sensor and temperature sensor. The Arduino is powered via USB cable connected to an external laptop/CPU. The housing for the hardware system is a box made of wood material with a glass top cover. Appropriate holes were made on the sides of the housing to access the sensors and USB cable by feeding them through the holes.

From there, the potential exists to employ machine learning (ML) methods to detect subtle changes in an individual's cognitive processes over time. Within this context, an investigation into machine learning technology suitable to classify a person's cognitive functioning during game play was undertaken. The eventual goal would be for ML-aided classification to serve as giant funnels for people at risk, providing a basis upon which to seek more traditional clinician-mediated assessment and subsequent therapeutic or psychosocial care. Ideally, the work will result in reaching a significant number of participants upon which to build a baseline of cognitive fingerprints across demographic profiles from which anomalies can be identified.
2 Software

The software portion of the project is based around the idea of emergent strategy[2]. Emergent strategy is the idea that over time a user will develop a strategy to win the game. The game at its core functions by pitting the player against a machine that will only ever play one strategy. By doing this we will see a strategy start to emerge from the player. We determine a strategy by consistency of play and a winning strategy is determined to be a win loss ratio two standard deviations above random, where random play is a 50/50 win loss ratio. This strategy is how we will form the bases of the assessment. When assessing a user we look at the strategy, see how often they are playing it, and whether it is a winning strategy or not. If we see they are playing a winning strategy infrequently or playing it for long stretches and then not playing it for a time, we can say that they may have a mild cognitive impairment. After we have determined that they may have an MCI we then look at the players game data and see how long it takes them to play cards and then determine if it is actually an early onset form of fronto-temporal or vascular dementia.

2.1 Design Criteria

The main purpose of the software section was to provide a way to collect and store meaningful data that the machine learning program could use for assessment. This is accomplished by using a repurposed social game designed for detecting emergent strategy as the front end that a user interacts with. To change the game in a way such that the type of data we desired was detected and collected new modes and features were added. Metrics were selected that would accomplish what we needed, these metrics can be seen in Table 1.
2.2 Subsection Design

2.2.1 Serious Game Application

The game is designed to assess either gambling addiction or one of two types of dementia, Vascular and Frontotemporal. The games developed are mobile social games that play like a modified version of the card game War. The game is changed up so that each player is dealt 3-5 cards face up from a standard deck of 52 cards see Figure 1 below for an example. Each player as a result only know the value of their own cards. Players choose a card to play from their hands each where the high cards wins. Rounds typically last between 1-10 seconds and a game lasts an average of 20-40 seconds. The interesting thing about this game is that rather than do the obvious and award one point for a round victory, instead we have opted for a weighted system as winning by a narrow margin is more satisfying and can allow a turn around at any point in a game. The Dementia version, or WarCAT, is the simpler of the two versions only having a computer opponent to play against, a store to buy rewards, and a way screen to switch between different owned rewards. The gambling version, or WarDOG, in addition to what WarCAT has, includes extra features and modes of play. The extra features in WarDOG are: the addition of betting, PvP, tournament play.
with variable tournament size (16-64 players), and a way to buy more in-game currency. All of the aforementioned items are designed to get the player addicted to the game so that if they start buying ingame currency we can then start the assessment for a gambling addiction. Both of the games also have a version where they are connected with physical sensors to help in the assessment, for more on that see section 3.

![Figure 1: A mid-game screenshot of the WarCAT mobile application](image)

2.2.2 Server for Data Collection

In order to be able to collect and use data from the application and hardware components, a database was needed. The database stores all the valuable information about the player and every game they play. For every game they play we record how long it took to select a card, which card they selected and in which order, whether they won that game, what strategy the bot was playing, and whether or not the player used our expected optimal strategy. For the version integrated with the hardware sensors we also track an
average of 30 samples of temperature and pulse for each game. To collect the temperature and pulse a second server was created that receives and interprets the sensor data so that we do not cause more strain on the game or game servers. Data gets transferred immediately upon a game finishing and it is sent securely with simple encryption. The database was created using MySQL community server 5.7. MySQL was chosen due to its ease of use and robust command set for extracting and storing data.

2.2.3 Server for Games

The game servers themselves are multithreaded servers designed to deal with a large number of human players at the same time. The servers for the gambling and dementia versions of the game are very different. The dementia version server is a web server that only receives client data and stores it, all other functionality is performed on the user end. The gambling version of the server is much more complicated and flows as can be seen in Figure 2. At the current moment there are still unexplainable bugs in the server but I believe those are caused by dropped packets such that the server or client will never receive a message and thus will never be able to progress in the game.
2.2.4 Synthetic Data Creation

In order to have a benchmark to test against, we have developed a bot that can play the game and simulate forgetfulness and get real results. The bot works on the assumption that the optimal strategy that we have come up with is in fact the optimal strategy. From the results that the bot has shown we can definitively say that the strategy we devised to be the best is in fact one of the best if not the best, we can deduce from the bot win lose ratio where the playerbot wins around 75% of the games it plays which if random wins 50% of the time we can say the player bot wins by more than 2 standard deviations from the norm which can be said to be greater than chance. With that knowledge we can now alter the playerbot so that it can mimic forgetfulness. We mimic forgetfulness by having
the playerbot playing randomly a certain percentage of the time determined by us before it starts. What this will get for us is enough training examples that our machine learning program should be able to distinguish between a human playing a definite strategy or if that person is forgetful and to what degree they are forgetful. For more on how this works with the other data collected see the machine learning section 4.

2.2.5 Race Mode

A mode in the game that we felt would help in assessments is a race mode. This race mode has two purposes: one is to help get users addicted to the gambling game and the other is to help with the sensor data collection. The game is designed to make the player feel rushed to play and beat the game by making them compete against other players. Players win the race mode by winning enough points that they get more than the specified goal number of points, the catch is that you are competing against other people for the fastest time and if they lose a game they will lose progress towards their goal. This helps force a player to create a strategy and get their adrenaline going which if they win will hopefully give them enough of a high that they proceed to continue wanting to play thus getting us the data we need on gambling addictions. The final output of the race mode can be seen in Figure 3 below. The second function was with the sensors, how they work together is that if our conjecture that they will get a rush of adrenaline from the mode we should then see a spike in pulse and body temperature. Those values will then be useful in seeing what modes and functions of the game are the root causes of the gambling addiction.
<table>
<thead>
<tr>
<th>#</th>
<th>USERNAME</th>
<th>TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Z73</td>
<td>28.12131814890185</td>
</tr>
<tr>
<td>2</td>
<td>O3</td>
<td>29.260646061267355</td>
</tr>
<tr>
<td>3</td>
<td>Z77</td>
<td>72.6564811978432</td>
</tr>
<tr>
<td>4</td>
<td>test</td>
<td>257.244153176</td>
</tr>
<tr>
<td>5</td>
<td>U99</td>
<td>724.019996795787</td>
</tr>
<tr>
<td>6</td>
<td>R62</td>
<td>943.0225264404842</td>
</tr>
<tr>
<td>7</td>
<td>Y95</td>
<td>975.0747706811471</td>
</tr>
<tr>
<td>8</td>
<td>H16</td>
<td>1016.336787758382</td>
</tr>
<tr>
<td>9</td>
<td>R95</td>
<td>1069.8181942859794</td>
</tr>
<tr>
<td>10</td>
<td>G30</td>
<td>1179.2438124575683</td>
</tr>
</tbody>
</table>

Figure 3: Results screen showcasing user scores in Race Mode
3 Hardware

For the hardware portion of the project, preliminary research was conducted on the physical symptoms of gaming/gambling addictions, Frontotemporal dementia and Vascular dementia. We wanted to find a way to monitor and measure these physical symptoms with biometric sensors to collect physical data of a player to be processed with machine learning. During our research, we consulted with two professionals in the field of MCI and serious gaming to narrow the scope of this section of the project. It was determined that the most feasible biometric sensors to be used were a pulse sensor to measure a players BPM and a temperature sensor to monitor a players skin temperature during game play. Furthermore, research was carried out to find a suitable microcontroller to power and program the biometric sensors. An Arduino UNO was selected to be used for this task.

3.1 Design Criteria

The main purpose of the hardware section is to integrate the pulse sensor and the digital temperature sensor to an Arduino UNO. This hardware system is used to monitor and measure biometric data of a player during gameplay. Once fully operational, the hardware system would then be integrated with the app game and transfer the collected data from the sensors into a database to be processed with machine learning. Sensor metrics can be seen in Table 2.

<table>
<thead>
<tr>
<th>Performance Metric</th>
<th>Target Spec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Body Temperature Sensing</td>
<td>28 to 42°C ± 0.5°C</td>
</tr>
<tr>
<td>Heart Rate Monitoring</td>
<td>60 to 150BPM ± 2%</td>
</tr>
</tbody>
</table>

Table 2: Target metrics for the biometric sensors
3.2 Subsection Design

3.2.1 Microcontroller

An Arduino UNO microcontroller board is used to program, power and collect data from the pulse sensor and digital temperature sensor. We chose to use an Arduino UNO board because the sensors we are using for this project do not require a powerful processor to run. The Arduino is powered via USB cable connected to a laptop or computer. The USB cable is also used to transfer data from the pulse and temperature sensors into the laptop or computer, to be stored in the database. The Arduino UNO has 14 digital I/O pins and 6 analog input pins[3]. It operates at a voltage of 5V with input voltage at 7 - 12V. The DC current per I/O pin is 40mA while the 3.3V pin runs at 50mA. A total of 4 pins are used on the Arduino board; the 5V pin, ground pin, A0 analog pin, and digital pin 2. More on the layout can be found in A.2.

3.2.2 Sensors

The pulse sensor (SEN-11574) is used to monitor the heartbeats per minute of a player as they play through the games race mode. Abnormalities, or rises in a players bmp are signs of a player under distress or excitement, both of which are physical symptoms of mental gaming addiction[4] that we can monitor and analyze. The pulse sensor is attached to a player using a wrapped Velcro finger strap. Arduino code was modified from a pre-existing code from the SEN-11574 manufacturer[5]. The pulse sensor is programmed to continuously run and take heartbeats per minute readings for as long as the Arduino board is powered on. The pulse sensor has a sampling rate of 500 Hz[6] and a baud rate of 250000.

The digital temperature sensor (DS18B20) is used to monitor the skin temperature of a player. Deviations from normal human skin temperatures (32 - 35 degrees Celsius)[7] while a player is playing through the games race mode are indications of a player under stress or excitement[8] much in the same way that a polygraph works. The temperature sensor is attached to a player using a wrapped Velcro finger strap. The Arduino code was developed using libraries available from the DS18B20 manufacturer. The first library is the OneWire library which is used for 1 wire devices to be able to communicate with
an Arduino board[9]. The second library is the DallasTemperature library which contains functions that are called to execute the tasks of the temperature sensors. These tasks include reading the temperature values in degrees Celsius and display them[10]. A 4.7k Ohm resistor is required to pull up the voltage of the digital pin 2 when the sensor is in use. Without the pull up resistor the signal of the temperature sensor would float and no reading can be taken. The 4.7k Ohm resistor is connected to the 5V pin and digital pin 2. Since the temperature sensor is running at the same time as the pulse sensor, it also has a baud rate of 250000.

3.2.3 Bluetooth Connection

A bluetooth HC-05 module is used to connect the sensors and the Arduino board to a laptop which contains a local server for storing and interpreting the incoming data. The bluetooth module is powered up by 5 volts from the Arduino, connecting RXD pin to digital pin 6 of the Arduino board, TXD pin to the digital pin 5 of the Arduino board, VCC to 5 volts and GND to ground of the Arduino board[3]. An android application, bluetooth terminal is used as a command line for the sensors[4]. The primary use of the application is to visualize data collected from the sensors and to prompt the sensors on when to start collecting or stop collecting data from the sensors. When s is entered as a command into the bluetooth terminal, it communicates via bluetooth connection to the sensors, prompting the sensors to start collecting data from the user. Likewise, when e is entered as a command into the bluetooth terminal, it communicates via bluetooth connection to the sensors to stop collecting data from the sensors.

3.2.4 Connections

The Arduino UNO board and biometric sensors are all interconnected using a single breadboard. Standard 22 gauge coated copper wires are used to wire the Arduino pins to the biometric sensor connections. The pulse sensor is connected to Arduino analog pin A0, and the temperature sensor is connected to Arduino digital pin 2. Data collected from these sensors while a player is playing the app game are transferred via Bluetooth to an android phone. This data can be monitored while a player is playing the game as this data is continuously collected by the sensors and transferred to the mobile device or computer.
3.2.5 Housing

A 8 x 5 x 2.5 inch wooden box is used to house the entire hardware system. The Arduino UNO board and breadboard are secured to the inside of the box housing with the use of double sided adhesive tape. Appropriate holes were made on the sides of the box housing for the USB cable and two sensor wires. The Arduino board emits almost no heat while in use, considering the average duration of gameplay for the race mode, generated heat is not a concern. Therefore, the material of which the housing is made of has more leniency, and in this case, it is made of wood. The housing is painted black in color simply for aesthetic purposes.
3.3 Process

3.3.1 Sensor Selection and Testing

The selection of biometric sensors began during the research phase where it was decided a players heart rate and body temperature needed to be monitored. From there, a pulse sensor, a digital temperature sensor, and an analog temperature sensor were ordered and tested with the Arduino UNO using code provided by their manufacturers. Modifications were made to the manufacturer's code to produce readable output data from the sensors that can be transferred and stored in a database. The testing phase consisted of running the pulse and 2 types of temperature sensors separately to one another to tune each sensor accurately. The digital temperature sensor was found to be more accurate than the analog sensor, therefore it was chosen to be used in the final system. Once fully operational separately, the pulse and digital temperature sensors were unified together with the Arduino UNO to monitor and output readings at the same time. Images of the output data from the sensors can be found in Appendix A.2.
3.3.2 Hardware Integration

The Arduino code was updated to start and stop the sensors from operating, with input entered through serial communication. Java code was developed to read the output data from the sensors in the serial monitor. Once a player begins a game, a signal is sent to the hardware prototype and the sensors would continuously run until the stop signal is sent from the game at the time when a player completes the race mode. Each game is logged in the database with the associated biometric data readings. Communication and data transfer between the game and Arduino UNO is done via USB cable. The integrated hardware works with both desktop and mobile app versions of the game.

4 Machine Learning

Machine Learning technologies have proven themselves invaluable in the design of pattern recognition systems over the past decade. Web searches, content filtering, recommendations on websites, speech and handwriting recognition all utilize an aspect of machine learning known as Artificial Neural Networks (ANNs)[11]. Such technologies utilize learning algorithms that involve providing data to a computer to subsequently determine complex functions and relationships between the input data. The core advantage of machine learning is that there is no need for human domain-specific expertise or hand-designed heuristics[12]. The goal of the machine learning section is to receive an image as an input, perform mathematical optimization known as the learning step, and, based on the extracted patterns, to output a value corresponding to the likelihood of a binary classification. By inputting WarCAT and biometric sensor data to the system, this output value represents the degree of certainty that a given user suffers from a form of Mild Cognitive Impairment.

4.1 Design Criteria

The target specification of the design was selected based on the performance of several types of ANNs when tasked with classifying 60,000 training and 10,000 test images found in the MNIST (Modified Institute of Standards and Technology) database of handwritten digits[13]. The specifications of the machine learning section are summarized in Table 3
Table 3: Summary of machine learning design specifications

<table>
<thead>
<tr>
<th>Performance Metric</th>
<th>Target Spec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training Data Required</td>
<td>10000 samples</td>
</tr>
<tr>
<td>Playtime for MCI Classification</td>
<td>≤ 150 games</td>
</tr>
<tr>
<td>Classification Accuracy</td>
<td>85% vs. example database</td>
</tr>
<tr>
<td></td>
<td>75% vs. MCI data</td>
</tr>
<tr>
<td>Input Image Resolution</td>
<td>56x56 pixels</td>
</tr>
</tbody>
</table>

4.2 Subsection Design

4.2.1 Machine Learning Framework

The backbone of the machine learning section is the TensorFlow[14] open source software library developed by Google. The selection of this framework was determined by comparing four software options[15][16] according to the following criteria: (1) platform compatibility, (2) open-source access, (3) programming language, (4) built-in CNN functions. TensorFlow emerged as the obvious choice since group members were familiar with Python and the C++ programming language. Further, TensorFlow is open source, has a large library for built-in numerical algorithms, and a built-in ability to visualize the network results through the use of TensorBoard[17]. A tabular comparison of potential software options can be found in Table 4 of Appendix A.3.

4.2.2 System Architecture

There are numerous ANN models explored in the literature. The core of these models consists of a series of layers which each transform input data in different ways. For the purpose of this project, the system is based upon a Convolutional Neural Network
(CNN) architecture often used in image- and video-based computer vision. The selection of the system architecture was determined by comparing five different network architectures according to the following criteria: (1) learning paradigm, (2) number of layers, (3) pre-training, (4) amount of data, and (5) convergence time. The CNN architecture was chosen since it excels in image pattern recognition[12] and since group members were familiar with the common MNIST handwritten image classification tutorials[18]. A tabular comparison of architecture options can be found in Table 5 of Appendix A.3.

This type of ANN uses feed-forward and gradient-based algorithms to perform the task of machine learning. The network consists of five layers: two convolutional layers where the learning occurs, two pooling layers to reduce the sample size, and a fully-connected output layer where a binary classification occurs[19]. Figure 6 below is an example of a Convolutional Neural Network, used for handwritten digit recognition[11].

![Figure 6: LeNet-5, an example of a 6-layer CNN architecture](image)

The two convolutional layers are organized in feature maps to act as filters for the input data with associated weights. Following each convolutional layer is a pooling layer which merges similar features together in an effort to reduce the sample size and computation time [12]. Finally, a fully connected layer is applied at the end of the network to act as a linear classifier by calculating weighted sums.
During the feed-forward pass of the input data to subsequent layers, the system performs a mathematical convolution. Figures 7 and 8 above depict how the weight filter $K$ is swept across the input $I$ and thus generates one of the convolutional feature maps, $C$. Mathematically, the operation can be expressed as follows:

$$ (I * K)_{ij} = \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} K_{m,n} \cdot I_{i+m,j+n} + b $$

Following the convolutional layer is a pooling layer. The importance of this layer is to bolster the systems resilience to small input translations or propagated positional errors of the data points in question. By splitting the output of the convolutional layer into patches,
pooling generates a new value based on the previous average value of each patch following this equation:

\[ o = \left\lfloor \frac{i - k}{s} \right\rfloor + 1 \]  \hspace{1cm} (2)

With Figure 8 as an example, the our 2x2 convolution can be reduced to a 2x1 output. This operation promotes faster convergence due to fewer calculations and improved system accuracy due to the translation invariance.

Finally, when both the convolutional and pooling layers are combined in a single expression, the mathematical result is as follows:

\[ x^l_{i,j} = \sum_m \sum_n w^l_{m,n} o^{l-1}_{i+m,j+n} + b^l_{i,j} \]  \hspace{1cm} (3)

where

\[ o^l_{i,j} = f(x^l_{i,j}) \]  \hspace{1cm} (4)

is the Activation function[19].

4.2.3 Optimization Algorithms

There are a few functions which comprise a full learning algorithm. Such functions are the cost or error function, backpropagation function, and activation function. The cost function, which our system tries to minimize after each iteration, characterizes whether or not our system is improving. Though closely related to the cost function, the backpropagation function is where the learning takes place. Backpropagation defines how the system is continually calculating the cost function, with respect to all of the other network parameters, and updating the weights associated with each filter. Though very simple, the purpose of the activation function is to apply a non-linear scaling to the filter weights to reduce convergence time.

The selection of the individual functions was determined by comparing two different numerical methods according to the following criteria: (1) number of layers, (2) computation
time, (3) accuracy, and (4) required amount of data. Mean-Squared Error was selected as the cost function solely due to the simplicity of the function. Gradient descent was selected as the backpropagation function as it is extremely common in practice, provides reasonable accuracy, and as it can work sufficiently with minimal data. Lastly, Rectified Linear Unit (ReLU) was selected as the activation function as it promotes the best convergence rates while also preventing erroneous detection of local minimums during the optimization process. Further information regarding the respective functions is explored in the following sections and tabular comparisons of these potential methods can be found in Table 6 of Appendix A.3.

4.2.3.1 Cost Function
The Mean Squared Error (MSE) is a measure of the magnitude of the difference between the estimator and what is estimated. With respect to our system, the cost function characterizes classification accuracy by comparing the the deviation between predicted and true data labels of the output layer per iteration. For a predictive system to be accurate, this cost function must be minimized. For a given cost $C$, error $E$, a number of predictions $P$, the predicted value $y_p$, and the target value $t_p$, we desire to find an optimal solution $f^*$ such that

$$C(f^*) \leq C(f) \forall f \in F$$

(5)

and that

$$E = \frac{1}{2} \sum_p (t_p - y_p)^2$$

(6)

is minimized[11][19][21].

4.2.3.2 Backpropagation
The backpropagation algorithm is the basis of machine learning. This first-order iterative optimization algorithm is used to find the local minimum of the previously mentioned cost function. By constantly applying the chain rule of derivatives layer-wise back through the network, we can calculate the change in the cost function with respect to the output. Learning will be achieved by iteratively adjusting the filter weights until the error is as low
For each iteration of the network, it is possible to update the weights of the filters using this minimization algorithm. The results essentially train the system with the calculated deltas,

\[ \delta_{i,j}^l = \frac{\partial E}{\partial x_{i,j}^l} \]  \[ (7) \]

The results of the backpropagation can be interpreted as the measurement of how a change in the weight filter \( K \) can affect the cost function. If a specific data point of the feature map is calculated to have a major impact on the output classification accuracy, it is clear that this data point is of great importance and is increased. If there is little to no correlation between a weight change of a given data point and the predictive power of the system, it is clear that this weight should be decreased. Similar to the forward-pass described in section 4.2.2, another convolution operation is used to calculate the new set of weights using the resulting deltas as shown in Figure 9 below.
4.2.3.3 Fine-Tuning Functions

There are two additional functions which have been utilized in the design of this convolutional network. One of which, as mentioned in section 4.2.3, is the Rectified Linear Unit function. It is utilized as a scaling factor to provide non-saturating activations following each set of the convolution and pooling layers. In this application the function help extract more complex features compared to using a linear scaling factor\cite{11}\cite{12}. The ReLU function is also incorporated in the final fully connected layer to further enhance the difference in certainties in the binary classification output. Mathematically the ReLU function is defined as:

\[
\text{ReLU}(x) = \max(0, x)
\]
Another important concept is one known as Dropout. It is a technique useful for preventing the system from overfitting its predictions and helps mitigate the optimization functions from wrongly identifying local minima as global minima\cite{22}. A visualization of the dropout function can be seen in Figure 10 below. Similar to the results seen in literature, the CNN designed for this project had a notable 2% increase in classification accuracy after implementing the dropout function. Tabular results of test trials can be found in Table 7 located in Appendix A.3.

\[
f(x) = \begin{cases} 
0 & \text{for } x < 0 \\
x & \text{for } x \geq 0 
\end{cases}
\]  

(8)

4.2.4 Data Analysis

Two critical aspects of machine learning are data manipulation and data visualization. For example, in the case of pattern recognition classification networks, the commonly-used MNIST database\cite{13} contains 60,000 examples of training images and a test set of 10,000 examples. Prior to providing input data to a neural network, this data must be carefully preprocessed and formatted according to the system structure. A custom Matlab code was
developed to handle the task of collecting, formating, and manipulating the collected soft-
ware and hardware data located in the local server. This code can be seen in Appendix A.3.

A scoring scale is used to convert the order of cards played into a quantitative value based on which bot strategy the player is trying to beat. The values representing the time to play each card have been inverted such that they can be used as a quantitative metric; this is based on the assumption that a player with MCI will have more difficulty deciding which card to play and thus receive a lower score. The biometric data has also been inverted and normalized, such that a rise in heart beat or temperature is indicative of a player’s struggle.

With the order, time, heart rate, and temperature data all formatted, we can proceed with matrix multiplications to generate an \( m \times m \) matrix. This matrix contains the encoded data values representative of a player’s performance over a range of \( m \) games played. For the purpose of this project, \( m = 28 \) as we are generating a 28x28 visual representation to be passed to our machine learning algorithm, as mentioned in the previous section.

Another operation which is used to manipulate the data is an additive scaling factor. Should the player strategy result in win, +0.25 score is awarded; should the player strategy result in a loss, -0.5 score is awarded. The resulting matrix is fed into a normalization function to ensure the final data points are between 0-1 to help the training process of the machine learning system[12]. Passing this matrix into the Matlab function colormap produces an \( m \times m \) pixel image containing the encoded player data for their performance as seen in Figure 11 below.
5 System Integration, Results, and Evaluation

The three sections of the system have all successfully been integrated with one another. The servers can successfully read data from the hardware and write it into the database. The machine learning program can also take that data and run it through its system and produce a result. The data that the machine learning program looks at includes all of the game data and biometric data that a player has. The data is then turned into image form so that the program can read and understand it.

5.1 Software

All design requirements that we specified at the outset of this project have been met and we have added in an additional feature at the behest of our supervisor. The servers can
be seen to collect data at appropriate intervals as well as provide appropriate feedback to us the developers. The database can properly sort and create files for use by our machine learning program. All data transfers are done in a secure and encrypted manner or are set up to do so. The game itself is fun and enjoyable but needs some improvement and proper evaluation in places.

We have decided that it would be important to evaluate our system for future development. We went with the Neilson heuristics[23] for evaluation of the UI that the player will use. If the UI isn’t good and the player choose not to play as a result our design has failed so the heuristic evaluation will provide a good idea on where are system succeeds and where it needs to be improved. A quick run down of the heuristics we are using is; visibility of system status, match between system and reality, user control and freedom, consistency and standards, error prevention, recognition rather than recall, flexibility and efficiency of use, aesthetic and minimalist design, help users, and provide help and documentation. A more detailed explanation of the Neilson heuristics, and images of the screens we evaluated, can be found in Appendix A.1.

5.1.1 Greeting Screen

The greeting screen is static with little to no indication that things are working as they are supposed to however, there is also no sign that anything is wrong. The user can easily tell what each button does and the language is very clear in regards to how a user could interpret it. There is however no clear exit button in the case where they selected this application by mistake. It follows some conventions, however standards for android and iOS games are hard to find. There are virtually no errors that can happen on this screen and those that could are caught and handled. For users that already have an account and have not logged out it will automatically bring them to the main page providing a good accelerator. No documentation or help is available from this screen but it could be for those curious about the game.
5.1.2 Login and Register Screens

The login and register screens are much the same as the greeting screen. They allow users who have pressed the wrong button to go back and select the other option. The screens clearly identify when a user has entered information that is wrong but does not identify what part of the information was incorrect, however for security reasons perhaps this is for the best. Again there is no help or documentation but everything is clear for the user to understand what is happening and as mentioned before, there is an accelerator in the login greeting and register screens for those that already have an account.

5.1.3 Main Menu

For the main menus it is all simple and easy to understand making it easy for the user to choose the option they want. There is a help and documentation button that the users can press, but it is very unclear and small, making it hard for the common user to see or understand that the button is for help. The options button and what it does is also unclear. The only accelerator is that the users last chosen settings are remembered however there is little need for accelerators otherwise. The buttons are unfortunately too small in the portrait mode. It is hard to tell system status other than by selecting options and seeing if it works, and if an error occurs the user has no way to recover other than restarting the application.

5.1.4 Store Screen

The store screens main feature is the scrollable list which is off center and too small making it difficult to read. There is also a lot of wasted space around the list. It is very easy to see if something has been bought but the language is more inline with how programmers view things than how the general public does, as it says owned: false instead of clearly stating not owned. The words on the buttons are not consistent in size and some run over to the next line when others stay on the same line. Users have complete control, however if they accidentally purchase something when trying to scroll through the list there is no refund button to undo that mistake. Users can easily see how much currency they have available but are given no indication on how to get more currency, and there is no help or documentation for this screen that users could rely upon to solve this. The screen has
a small amount of error prevention in that someone cannot purchase the same item twice but nothing else. There are no ways for experts to get things done faster however there would be very little one could do to have accelerators for them in this screen aside from a quick undo.

5.1.5 Record Screen

The record screen is very simple and straightforward for ease of understanding by the user. There is too much open space and the words are too small in portrait mode making it impossible. No help or documentation for users who do not understand the screen to get an understanding. Clear way to get back to the main menu. Screen too simple for any advanced use.

5.1.6 Options Screen

In the options screen it is easy for users to understand what the buttons do and easy for users to go back to the main screen if needed. The screen unfortunately starts with a weird do not own message at the bottom of the screen which could be confusing for a user that may think they have to purchase the ability to change aesthetics in the game. The two cards on the top and bottom of the screen (in portrait) are not labeled and it is unclear what they are and are for. Each option in the list of options has too much information much more than is required, users shouldnt be presented with the option to select items they havent purchased as it could lead to confusion and will slow down the selection process. There are no accelerators here as well and the obvious one would be to save favourites in a separate list for quick access. Once again there is no help or documentation to help walk a user through this page should they not understand it. The card back and face lists are mixed together so if a user wants to change one they have to scan through the whole list of both to do so and may miss the opinion they want. The back button while being coloured differently slightly blends into the background making it difficult to see at a glance. Users will need to remember where everything is rather than being able to use recognition to determine the location of items. There is feedback from the system if things change in the cards and background however if there is an error there is no indication of it happening, however the error checking for this screen catches all known errors.
5.1.7 Help Screen

The help screen is very basic and cannot help if the user has more complicated issues. The explanation of the game is poor at best with mistakes and spelling errors that need to be taken care of. The way the paragraphs are layed out makes it very difficult to read the actual help and the word size in portrait doesn’t make it even worse. In the credits screen a line talking about the license that the assets are under is cut off which could confuse and scare users. Buttons on the help and credits screen are clearly labeled using user language making it very easy to understand what happens when pushed. There is no status that needs to be displayed except whether there is an error or not but that extends back to what was said in all previous versions.

5.1.8 Game Screen

In the game screen there is no help with how to play and there is no indication that you are on a timer to play a card. The scores are small and not easily visible to the user. If you select the wrong card by mistake there is no undo or error prevention to fix that mistake. You have no control over the ordering of your hand and you have to deal with it in the order it is presented. In portrait mode the cards are very small and hard to select for those with large fingers. Each round resets too fast so it is impossible to tell what happened and whether you won or not. There is extensive error prevention, however when an error does occur there is no indication that one has occurred other than the screen being unresponsive. There is in addition no way to fix the error that has happened except for restarting the game which will mean all recent progress has been lost as it was not logged into the database. The users have to remember the rules rather than having a way to access them in the middle of the game to help with memory, however as the game is designed to detect those issues it may be better to leave it as is. There is a lot of unused space in the portrait version. The cards at the top of the screen are unlabeled making it hard for some to distinguish what they are for as they serve no purpose other than aesthetic. If you can distinguish the top cards as the opponents cards that may leave you wondering why there are six and you have five. There is no room for accelerators in the actual game except to have an option that turns off animations.
5.1.9 Results Screen

The results screen is very clear what everything is and means with the exception of the reward points at the bottom of the screen and how you lose or gain them. In the portrait mode there is once again a lot of unused space and as a result the words are very small and hard to read. There is an accelerator for those who want to immediately go back into a game in the play again button. The buttons are very clearly labeled in the user's language. The user has feedback where if they touch the screen while the reward points are changing, the number will jump to the end of its calculation. There is no help or need for help on this screen. There is no way to undo anything or anything to undo. If errors do occur (which because of the error checking they shouldn’t) there is no way to diagnose or determine how to fix it except restarting the application.

5.2 Hardware

The pulse sensor (SEN-11574) and digital temperature (DS18B20) sensors are fully integrated with the Arduino UNO board. This hardware system is able to read the bpm and skin temperature of a player during gameplay and display the data in the serial monitor. Software was developed to read the serial monitor through serial communication and sync the games of the race mode with real-time gameplay of a player. The biometric data collected during gameplay is stored and arranged in a database to be processed by machine learning methods.

5.3 Machine Learning

The machine learning algorithm is fully functional for a 28x28 pixel image input and is able to classify sample data with 98% accuracy. The data analysis portion of machine learning is in a functional but limited state. Further efforts should be made to ensure that the data visualization is able to be properly analyzed by the machine learning algorithm. Since this project is aimed to be a proof-of-concept, as there was a very limited amount of real data to be analyzed, and as the benchmarks used were based off of theoretical values, the results of the binary classification should not be taken as an accurate representation of an individual’s cognitive ability.
6 Conclusion

In conclusion, the project was broken up into three main sections; the software section which includes in-game development and programming, the hardware section that governs the development with the Arduino UNO, Bluetooth module and biometric sensors, and the machine learning section used to process and correlate all data collected and stored in the database from gameplay. These sections of the project would then be integrated with one another to create a single working system to assess a player for potential MCI in Vascular dementia, Frontotemporal dementia, and gaming/gambling addiction. The project was predicated on the idea that with emergent strategy we would be able to see patterns form in a user and assess their cognitive footprint based on said patterns. We can say that we were successful in seeing that pattern as noted by the machine learning visualization of the data we were able to create. Our games are enjoyable enough to keep the attention of a player to gather meaningful data and store it in a secure fashion. We hope our project will encourage potential MCI patients seek medical attention from a professional, if needed.

7 Future Work

7.1 Software

7.1.1 Software Distractions

Our research has shown that having different distractions and pop ups occur, we can better assess the brain’s processing power and directly assessing distractibility. For the brain processing power we can see how long it would take to react to a given distraction and how long it would take to remove that distraction if it can be removed. It would also show how various people react to the different distractions and given the nature of how we collect and assess data we could then better assess for the frontotemporal dementia. One way of doing this would be to create various in game distractions either during, before, or after a game. The distractions that we have come up with form three categories; sound based distractions, colour based distractions, and interactable distractions. Interactable distractions have already been touched on but will consist primarily of minigames that the player will need to complete in order to continue playing the game. This will be touched on
in greater detail in the hardware section of 7.2. The sound based distractions would have the device emit a sound at varying frequencies for varying intervals throughout the game. The colour based distractions would see the background that the game is played on change to varying colours for a varying length of games, what this would do is throw the player off balance by throwing their expectations of the status quo out the window. However the colour method will only work for a short amount of time as the user will inevitably get used to the idea that the colour will change occasionally, but this leads to its advantage as users with higher distractibility will likely be affected by this for a longer duration.

7.1.2 Accelerometer

A facet of our project that we failed to touch upon was using the accelerometer in a phone to measure the level of shakiness in a player’s hand. The main reason this wasn’t touched upon was due to time constraints and the lack of research into how useful that data would be in assessing any of the dementias or gambling addiction. In the future more research into how this data could be used is needed.

7.1.3 Additional Levels and Bot Strategies

The game in its current form has only three strategies the bot can play for simplicity, however more strategies to elongate the play time and thus data collection time in greatly needed. Added more strategies could potentially make the game more interesting for advanced players and allow for a greater variety of data points to draw from assessment. Another facet for bot strategies that needs to be looked into is a bot that can learn player behaviour so that when players play against it, it will naturally evolve and improve with them, making the game more fun for those who do not wish to participate in PvP.

7.1.4 Neilson Heuristics

From the evaluation we can clearly see that help and documentation is sorely lacking in addition to proper aesthetics to fit it designated device better. We can also see that there are still many small bugs and issues that need to be cleaned up and taken care of. Upon finishing the improvements a new assessment will be needed to see if the criterion for a good UI are being met.
7.1.5 In-App Bluetooth Connection

Bluetooth for transfer of sensor data can easily be accomplished. However, it would require that the user be next to or within range of the bluetooth server in order to fully utilise this. Going forward we need to take the Bluetooth functionality and move it onto the user’s device as a background application that wont interfere with the game or the games speed in a reasonable manner.

7.1.6 WarDOG Application Changes

For the gambling app betting is very lackluster and could be improved upon by moving it to in game betting rather than betting before you can even see your hands. If we can move it to in game, the gameplay will be improved by effectively introducing bluffing into the game as well. The downside to this feature is that it would likely slow down the game by a noticeable margin. Slowing down the game while not intrinsically bad means that assessment methods would need to be adjusted to account for it. In addition the new phenomenon of loot boxes and card packs should be integrated into the game. The reasoning behind this is that from current games we can see that these features have the same effect as a slot machine does. Card packs offer that great feeling of satisfaction of getting a desired result or in this case rare cardskin while also having aspects of the sunk cost fallacy.

7.2 Hardware

7.2.1 Blood Pressure Sensors

The blood pressure sensor was one of the more favorable sensors to integrate with the Arduino hardware system because of blood pressure being one of the few physical symptoms associated with MCI. Blood pressure was one of the only physical symptoms Dr. Benedict Albensi suggested we can physically measure using sensors. There were two ways we can integrate a blood pressure sensor with the Arduino UNO. The first being, setting the blood pressure sensor to inflate and deflate slowly to take a reading during game play[24]. The other is keeping the blood pressure sensor inflated during game play and slowly deflate it to take a reading at the end of each game. However, both these methods are time consuming.
The limiting factor of a blood pressure sensor is the time it takes to inflate and deflate to take a reading. Unlike the heartbeat and temperature sensor that can take readings at a fast pace to keep up with the pace of the game.

7.2.2 Eye Tracking

The concept of eye-tracking was to monitor a player’s eye movement during game play. There are many eye movements associated to certain types of body languages. Eyes veering away from the screen would suggest a player is thinking, confused or processing a strategy. Players staring directly at the screen during game play would suggest full engagement, strong concentration and clear-thinking processes. Abrupt eye movement would suggest a player is panicked or confused[25]. The integration of eye-tracking would have been implemented using the front facing camera of the smartphone and programmed through the games code.

7.2.3 Hardware Distractions

To make the game more challenging and to increase the memory processing of a player. The idea of creating interactive hardware that worked in tangent with the game was suggested. This was more of conceptual idea where hardware would have to be designed to attempt to distract a player during game play, forcing a player to have to concentrate more and in turn use more brain power. These distractions were going to be of the form of flashing LED lights, phone vibrations, and interactive game console controllers. The LED lights would be controlled with the use of the Arduino UNO. The phone vibrations would be programmed into the games code. And physical game controls would have been fabricated and powered by the Arduino UNO.

7.3 Machine Learning

7.3.1 GAN

A novel concept known as a Generative Adversarial Network can be utilized as a means to enhance the performance of our CNN. In this architecture, one portion of the network generates images and tries to fool the already developed CNN classifier. At each iteration,
similar to how typical neural networks learn, the generative network learns how to produce increasingly better images. As a result of this operation, the CNN must have more specific feature detection to help decipher the difference between an image of "true" data and an image of "synthetic" data produced by the GAN. An example of a GAN architecture can be seen in Appendix A.3.

7.3.2 Layer-wise Visualization

As machine learning networks often appear as large black boxes[26], data visualization tools such as TensorBoard[17] and Deep Visualization Toolbox[1][26] are commonly used to display visual representations of data at discrete stages of a network. Continue efforts to implement layer-wise visualization would go a long way to help understand how each layer is operating, which features are being extracted, and could give insight as to how the network parameters, such as filter weights, should be initialized.

Figure 12: A look at how each layer detects different features of an image of a cat[1]
8 References


A Appendix

A.1 Software

Figure 13: Greeting screen
Figure 14: Login Screen

Figure 15: Registration Screen
Hello and welcome to the WarCAT! A simple game of war!

From here you may select whether you would like to play against a bot, view your record, or head to the store to purchase new cards to play with and new backgrounds to play on.

The game is simple:
You are dealt a random hand of 5 cards. Using these cards, you will wage war on your opponent by selecting a card to pit against them. The opponent with the highest card wins, but more points will be earned for a closer victory. Winning by the skin of your teeth will net you big points, so don't be afraid to play your lower cards! Sometimes brute force is not the only method of success.

The score is calculated with this equation:
score = 13 - (Winning card value - Losing card value)

Good luck out there!

Figure 16: Information screen

ART ASSETS
Nora Shishi - 8-bit cards
GreyWyvern - Animal Cards
Jefeets - Colorful Poker Card backs
StumpStrust - Space Background
OpenGameArt.org - Grass Background
TextureZ.com - Brick Background
For Most Art Licensed under

Audio
Mike Koenig - Applause
https://creativecommons.org/licenses/by/3.0/

Figure 17: Credits screen
Figure 18: Main menu screen for WarCAT

Figure 19: Record Screen
Figure 20: Leaderboard screen
Figure 21: Friends screen
Figure 22: Store screen

Figure 23: Options screen
Figure 24: Screen for changing card options
Figure 25: WarDOG game mode selection screen
Figure 26: Regular game mode screen

Figure 27: Final results screen after points calculation
Figure 28: Tournament selection screen
Figure 29: Betting screen
Figure 30: Tournament wait screen
Figure 31: Tournament play screen
Figure 32: Play style selection screen
Figure 33: Race game screen
Figure 34: Race game break screen

<table>
<thead>
<tr>
<th>Games Played</th>
<th>Time</th>
<th>Total Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>88.208859408s</td>
<td>79</td>
</tr>
</tbody>
</table>

Do You Want to Continue?

Main Menu  Continue
Capstone Bot Code - AutoPlayForgetful.cs

```csharp
using UnityEngine;
using UnityEngine.UI;
using System;
using System.Text;
using System.Net;
using System.Collections;
using System.Net.Sockets;

public class AutoPlayForgetful : MonoBehaviour {
    public GameObject socketHandler;
    public Text text; // filled with the register error text
    public GameObject canvas;
    public GameObject cardConverter;
    public GameObject CardChanger;
    public GameObject Gameplay;
    public GameObject Login;
    public GameObject register;
    private double forgetfulness = 0.25; // how often it will forget the correct strategy in a percentage
    private string botbasename = "Forgetful25BOT";
    private int botnumber;
    private Boolean playnextflag = false;
    private Boolean shouldLogout = false;
    // Use this for initialization
    void Start()
    {
        botnumber = 7;
        Debug.Log("Start");
        string botname = botbasename + botnumber;
        Debug.Log("BotName");
    }
}
```
canvas.GetComponent<PanelHandling>().registerConnection();
Debug.Log("registerConnection");
socketHandler.GetComponent<Connection>().usernameReg.text = botname;
socketHandler.GetComponent<Connection>().passwordReg.text = botname;
socketHandler.GetComponent<Connection>().playerName.text = botname;
Debug.Log("bot");
socketHandler.GetComponent<Connection>().submitNewUser(text); // Need to wait for a response here or else there will be errors
Debug.Log("submitted new user");

public void GameLoop()
{
    // while (!shouldLogout)
    //{
    //    // while (!canvas.GetComponent<PanelHandling>().isMainMenu())
    //    {
    //        //blocks
    //    }
    //    canvas.GetComponent<PanelHandling>().menuBotsAction();
    StartCoroutine(playgame(StoredData.sd.getDataManager().level));
    // playgame(StoredData.sd.getDataManager().level);
    // if (canvas.GetComponent<PanelHandling>().inLevelUp())
    //{
    //    if (StoredData.sd.getDataManager().level == 4)
    //    {
    //        shouldLogout = true;
    //    //}
    //    else
    //    {
    //        canvas.GetComponent<PanelHandling>().backToMain();
    //    //}
    //}
    //else
    //{
    //    canvasGetComponent<PanelHandling>().backButtonAction();
    //}
    //}
    //}
public void reregister()
{
    string botname = botbasename + botnumber;
    canvas.GetComponent<PanelHandling>().registerConnection();
    socketHandler.GetComponent<Connection>().usernameReg.text = botname;
    socketHandler.GetComponent<Connection>().passwordReg.text = botname;
    socketHandler.GetComponent<Connection>().playerName.text = botname;
    socketHandler.GetComponent<Connection>().submitNewUser(text); // Need to wait for a response here or else there will be errors
}

// plays a card after the fast time of 0.5 sec
public IEnumerator waitToPlay(int number)
{
    switch (number)
    {
    case 1:
        yield return new WaitForSeconds(0.5f);
        Gameplay.GetComponent<GameplayScript2>().playCard(Gameplay.GetComponent<GameplayScript2>().cardButton1);
        break;
    case 2:
        yield return new WaitForSeconds(0.5f);
        Gameplay.GetComponent<GameplayScript2>().playCard(Gameplay.GetComponent<GameplayScript2>().cardButton2);
        break;
    case 3:
        yield return new WaitForSeconds(0.5f);
        Gameplay.GetComponent<GameplayScript2>().playCard(Gameplay.GetComponent<GameplayScript2>().cardButton3);
        break;
    case 4:
        yield return new WaitForSeconds(0.5f);
        Gameplay.GetComponent<GameplayScript2>().playCard(Gameplay.GetComponent<GameplayScript2>().cardButton4);
        break;
    case 5:
        break;
    }
yield return new WaitForSeconds(0.5f);
        Gameplay.GetComponent<GameplayScript2>().playCard(Gameplay.GetComponent<GameplayScript2>().cardButton5);
        break;
    }
    playnextflag = true;
}
}

public IEnumerator playgame(int strategy)
{
    Debug.Log("Start Game");
    yield return new WaitForSeconds(1.0f);
    Debug.Log("Start Game");
    int[] playorder = Gameplay.GetComponent<GameplayScript2>().getMyHand();
    Debug.Log("play order: "+playorder[0]);
    Debug.Log("play order: " + playorder[1]);
    Debug.Log("play order: " + playorder[2]);
    Debug.Log("play order: " + playorder[3]);
    Debug.Log("play order: " + playorder[4]);
    // NEED TO DO ALL THIS TO AVOID POINTER SHENANIGANS
    int[] tempplay = new int[5];
    for (int i = 0; i < playorder.Length; i++)
    {
        tempplay[i] = playorder[i];
    }
    int[] tempplay2 = new int[5];
    for (int i = 0; i < playorder.Length; i++)
    {
        tempplay2[i] = playorder[i];
    }
    Array.Sort(tempplay);
    Debug.Log("sorted cards");
    Debug.Log("play order: " + tempplay[0]);
    Debug.Log("play order: " + tempplay[1]);
Debug.Log(“play order: “ + tempplay[2]);
Debug.Log(“play order: “ + tempplay[3]);
Debug.Log(“play order: “ + tempplay[4]);
Debug.Log(“play order: “ + temporder[0]);
Debug.Log(“play order: “ + temporder[1]);
Debug.Log(“play order: “ + temporder[2]);
Debug.Log(“play order: “ + temporder[3]);
Debug.Log(“play order: “ + temporder[4]);
int[] indices = new int[5];
for (int i = 0; i < 5; i++)
{
    indices[i] = Array.IndexOf(tempplay2, tempplay[i]);
    tempplay2[indices[i]] = 10000;
    tempplay[i] = 10000;
}
Debug.Log(“index: “ + indices[0]);
Debug.Log(“index: “ + indices[1]);
Debug.Log(“index: “ + indices[2]);
Debug.Log(“index: “ + indices[3]);
Debug.Log(“index: “ + indices[4]);
Debug.Log(“The Start of the Strategy descision”);
System.Random rand = new System.Random();
if (rand.NextDouble() < forgetfulness)
{
    strategy = 4;
}
switch (strategy)
{
    case 1:
        tempplay[0] = indices[0];
        tempplay[1] = indices[4];
        tempplay[2] = indices[3];
        tempplay[3] = indices[2];
        tempplay[4] = indices[1];
        break;
    case 2:
tempplay[0] = indices[1];
tempplay[1] = indices[2];
tempplay[2] = indices[3];
tempplay[3] = indices[4];
tempplay[4] = indices[0];
break;

case 3:
tempplay[0] = indices[3];
tempplay[1] = indices[2];
tempplay[2] = indices[1];
tempplay[3] = indices[0];
tempplay[4] = indices[4];
break;

case 4:
tempplay[0] = 0;
tempplay[1] = 1;
tempplay[2] = 2;
tempplay[3] = 3;
tempplay[4] = 4;
break;
}

// Works because of pointer memory shenanigans in other words temp[c] points to the same spot in memory as playorder[c]
for(int c = 0; c<5; c++)
{
    Debug.Log("Play a card "+tempplay[c]);
yield return new WaitForSeconds(0.3f);
switch (tempplay[c]+1)
{
    case 1:
        Gameplay.GetComponent<GameplayScript2>().waitForOpp();
        Gameplay.GetComponent<GameplayScript2>().playCard(
            Gameplay.GetComponent<GameplayScript2>()..cardButton1);
        Gameplay.GetComponent<GameplayScript2>().getResult();
            GetComponent<Toggle_Animation>().TogglePressed(
                    animator);
        break;
    }
break;
case 2:
  Gameplay.GetComponent<GameplayScript2>().waitForOpp();
  Gameplay.GetComponent<GameplayScript2>().playCard(
    Gameplay.GetComponent<GameplayScript2>().cardButton2);
  Gameplay.GetComponent<GameplayScript2>().getResult();
    GetComponent<Toggle_Animation>().TogglePressed(
        animator);
  break;
case 3:
  Gameplay.GetComponent<GameplayScript2>().waitForOpp();
  Gameplay.GetComponent<GameplayScript2>().playCard(
    Gameplay.GetComponent<GameplayScript2>().cardButton3);
  Gameplay.GetComponent<GameplayScript2>().getResult();
    GetComponent<Toggle_Animation>().TogglePressed(
        animator);
  break;
case 4:
  Gameplay.GetComponent<GameplayScript2>().waitForOpp();
  Gameplay.GetComponent<GameplayScript2>().playCard(
    Gameplay.GetComponent<GameplayScript2>().cardButton4);
  Gameplay.GetComponent<GameplayScript2>().getResult();
    GetComponent<Toggle_Animation>().TogglePressed(
        animator);
  break;
case 5:
  Gameplay.GetComponent<GameplayScript2>().waitForOpp();
  Gameplay.GetComponent<GameplayScript2>().playCard(
    Gameplay.GetComponent<GameplayScript2>().cardButton5);
Gameplay.GetComponent<GameplayScript2>().getResult();
Gameplay.GetComponent<GameplayScript2>().cardButton5.
GetComponent<Toggle_Animation>().TogglePressed(
Gameplay.GetComponent<GameplayScript2>().cardButton5.
animator);

break;

playnextflag = false;
}

Debug.Log("Finished Game");
yield return new WaitForSeconds(1.0f);
if (StoredData.sd.levelUpCheck())
{
    if (StoredData.sd.getDataManager().level == 4)
    {
        shouldLogout = true;
    }

    else
    {
        canvas.GetComponent<PanelHandling>().backToMain();
    }

} else
{
    canvas.GetComponent<PanelHandling>().backButtonAction();
}

// Update is called once per frame
void Update()
{

}

Capstone Race Game - RaceGame.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 Server;

import java.sql.Connection;
import java.net.*;
import java.io.*;
import java.sql.*;
import java.time.*;
import java.sql.Statement;
import java.util.Random;

/**
 * @author leducmck
 */
public class RaceGame implements Runnable{
    //References to client connections
    private ConnectionHandler connection1;
    private ConnectionHandler connection2;

    //Reference to QueueHandler object
    private QueueHandler queueHandler;

    //Variable representing the desired scoring type
    private int scoreType;
    private int strategy;
    private Random rand;

    //Tuning variables for the Game class
    private int HAND_SIZE = 5;
    private int GAME_LENGTH = 5;

    //Booleans to make sure disconnections are safe
    public boolean isRunning = false;
    private boolean sentDisconnect = false;
    private boolean connect1Sent = false;

    //Values for the Race
private int gamePoints=0;
private int GOALPOINTS=500;
private int numGamesPlayed=0;
private int wins=0;
private int losses=0;
private double timePerGame=0.0;
private double totalTime=0.0;

//Arrays to store your hand in
private int[] hand1 = new int[HAND_SIZE];
private int[] hand2 = new int[HAND_SIZE];
private int[] hand2highlow=new int[HAND_SIZE];

//Array to determine cards the bot has used
private boolean[] canUse = new boolean[HAND_SIZE];

//SQL access variables
private Connection con = null;
private Statement st = null;
private ResultSet rs = null;
private ResultSet rs2 = null;

//String url = "jdbc:mysql://127.0.0.1:3306/MBROWar";
String url = "jdbc:mysql://localhost:3306/MBROWar";
//String url = "jdbc:mysql://140.193.231.199:3306/MBROWar";

//SQL access information
//!!! --> SENSITIVE
//String user = "jared";
//String password = "5251992";
//!!! --> SENSITIVE

//Alternate device info
//!!! --> SENSITIVE
String user = "root";
String password = "br0r0tPaSs";
//!!! --> SENSITIVE

public RaceGame(QueueHandler queueHandler, ConnectionHandler connection1,
byte[] properties) {

  this.queueHandler = queueHandler;
  this.connection1 = connection1;

  this.scoreType = Character.getNumericValue((properties[0] & 0xFF));
  this.GAME_LENGTH = Character.getNumericValue((properties[1] & 0xFF));

  rand = new Random();
  this.strategy = connection1.getStrategy();
}

public void run() {
  raceLoop();
}

public void raceLoop() {

  String randUserName;
  String[] randUserList = new String[9];
  StandardMessage message;
  try {
    con = DriverManager.getConnection(url, user, password);
    st = con.createStatement();
    // need to add section to ask for gamemode in Database
    // rs = st.executeQuery("SELECT HANDLE, GAMES, WINS, LOSSES, TIME
    // FROM race WHERE STRATEGY=" + strategy + " AND GAMEMODE=" +
    // GAME_LENGTH + " ORDER BY RAND() LIMIT 9");
    rs = st.executeQuery("SELECT HANDLE FROM race WHERE STRATEGY=" +
      strategy + " AND GAMEMODE=" + GAME_LENGTH + " GROUP BY HANDLE ORDER
      BY RAND() LIMIT 9");
    gamePoints = 0;
    numGamesPlayed = 0;
    System.out.println("Started Race Loop");
    if (rs.relative(9)) {
      rs.beforeFirst();
      RaceUserSet[] rus = new RaceUserSet[9];
      for (int c = 0; c < rus.length; c++)
Page 75
{  
  rs.next();
  rs2=st.executeQuery("SELECT HANDLE, GAMES, WINS, LOSSES, TIME FROM race WHERE STRATEGY=\"+strategy+\" AND GAMEMODE=\"+GAME_LENGTH+\" AND HANDLE=\"+rs.getString("HANDLE")+\", ORDER BY RAND() LIMIT 1");
  rs2.next();
  rus[(j%10)].Handle=
  new RaceUserSet(rs2.getString("HANDLE"),rs2.getInt("GAMES"),rs2.getInt("WINS"),rs2.getInt("LOSSES"),rs2.getDouble("TIME"));
  
  int j=0;
  while(gamePoints<GOALPOINTS)
  {
    
    timePerGame=System.nanoTime();
    gamePoints=gamePoints + pvbRaceProcessing(rs.getString(
      rus[(j%10)].Handle));
    timePerGame=System.nanoTime()−timePerGame;
    numGamesPlayed++;  
    totalTime=totalTime+timePerGame;
    j++;
  }
  totalTime=(totalTime/1000000000);
  double [] times=new double [10];
  int rank=0;
  for(int c=0;c<times.length−1;c++)
  {
    times[c]=rus[c].Time;
  }
  times[times.length−1]=totalTime;
  for(int c=0;c<times.length−1;c++)
  {
    if(times[c]==totalTime)
    {
      rank=c;
      break;
    }
  }
}
if (rank==1)
{
    connection1.getRecord().setRewardPoints(connection1.getRecord().getRewardPoints() + 600);
}
else if (rank==2)
{
    connection1.getRecord().setRewardPoints(connection1.getRecord().getRewardPoints() + 360);
}
else if (rank==3)
{
    connection1.getRecord().setRewardPoints(connection1.getRecord().getRewardPoints() + 120);
}

// sending results
String resultString;

try{
    for (int c=0; c<rus.length; c++){
        resultString = rus[c].Handle + " " + rus[c].Games + " " + rus[c].Wins + " " + rus[c].Losses + " " + rus[c].Time + " BOT";
        connection1.getMessageHandler().sendMessage(new StandardMessage(1, MessageType.RANK.ordinal(), resultString));
    }
    resultString = connection1.handle + " " + numGamesPlayed + " wins+" + " losses+" + totalTime + " PLAYER";
    connection1.getMessageHandler().sendMessage(new StandardMessage(1, MessageType.RANK.ordinal(), resultString));
}

catch(InterruptedException e)
{
    System.err.println("Error Sending SQL Data to " +

Page 77
connection1.getHandle("Disconnecting");
connection1.getMessageHandler().disconnectHandling();
}
else
{
    int c = 0;
    int playIndex = 0;
    int botWins = 100;
    int botgames = 0;
    while (gamePoints < GOALPOINTS)
    {
        if (c < randUserList.length)
        {
            randUserName = toAlphabetic((int) Math.floor(Math.
                    random() * 26) + 1) + Integer.toString((int) Math.
                    floor(Math.random() * 100));
            randUserList[c] = randUserName;
        }
        else if (playIndex == randUserList.length)
        {
            playIndex = 0;
        }
        System.out.println("Bot Name: " + randUserList[playIndex]);
        timePerGame = System.nanoTime();
        gamePoints = gamePoints + pvbRaceProcessing(randUserList[
                    playIndex]);
        timePerGame = System.nanoTime() - timePerGame;
        numGamesPlayed++;
        totalTime = totalTime + timePerGame;
        System.out.println("Race Update for " + connection1.
                    getHandle() + " Time: " + (totalTime / 1000000000) + " Points:
                    " + gamePoints);
        c++;
        playIndex++;
        // the below nested if statements are for the intermediary
        // panel and should be uncommented if allowing a stop
        // between ames is allowed
        if (numGamesPlayed % 10 == 0)
try {
    connection1.getMessageHandler().sendMessage(new StandardMessage(2, GameMessageType.SCORE.ordinal(), ""+(totalTime/1000000000) ));
    message=connection1.getMessageHandler().receiveGameMessage();
    if(message.getType()==GameMessageType.ROUND.ordinal()){
        if(message.convertInt()==1) {
            //just skips the else if and continues the function
        }
        else if(message.convertInt()==2) {
            new Thread(new MainMenu(queueHandler, connection1)).start();
            return;
        }
        else {
            System.err.println("ERROR Recieving message Type ROUND in Race Disconnecting");
            connection1.getMessageHandler().disconnectHandling();
        }
    }
    catch(InterruptedException ie){
        System.err.println("Error Sending Time Data to " + connection1.getHandle()+" Disconnecting");
        connection1.getMessageHandler().disconnectHandling();
    }
}
}
totalTime=(totalTime/1000000000);
String resultString;

try{
    for(c=0;c<randUserList.length;c++)
    {
        botgames=(int)((Math.random() * 30) + 9); // a reasonable
        estimate I think for how many games it should
take, will be adjusted if need be
        while(botWins > botgames)
        {
            botWins = botgames - (int)((Math.random() * botgames) + 9);
        }
        resultString = randUserList[c] + " +botGames + " +botWins
        + " +((botgames - botWins) + " +((Math.random() * 14.82) + 0.18) * GAME_LENGTH * botgames) + " BOT";
        connection1.getMessageHandler().sendMessage(new
        StandardMessage(1, MessageType.RANK.ordinal(),
        resultString));
    }
    resultString = connection1.getHandle() + " +numGamesPlayed +" +wins + " +losses + " +totalTime + " PLAYER";
    connection1.getMessageHandler().sendMessage(new
    StandardMessage(1, MessageType.RANK.ordinal(),
    resultString));
}
catch(InterruptedException e)
{
    System.err.println("Error Sending SQL Data to " +
    connection1.getHandle() + "Disconnecting");
    connection1.getMessageHandler().disconnectHandling();
}
}
if(gamePoints < 1000) // checking to make sure they didn’t quit part
    way through or disconnect
{
    st.executeUpdate("INSERT INTO race (HANDLE, STRATEGY,
    GAMEMODE, GAMES, WINS, LOSSES, TIME) VALUES('" +
}
connection1.getHandle()+"', "+strategy+, "+GAME_LENGTH+, "+numGamesPlayed+, "+wins+, "+losses+, "+totalTime+")

265)
266   new Thread(new MainMenu(queueHandler, connection1)).start();
267 }
268 }
catch(SQLException se){
269   System.err.println("An error has occured while connecting to the
270   mySQL database!!");
271   System.err.println("Race Mode");
272 }
273 }
274 }
275 }
276 }
277 }
278 //Bot information variables
279 //String bot_name = "War-O-Matic";
280 int bot_hand_index = 0;
281   int gameLen = GAME_LENGTH;
282
283   for(int c=0;c<canUse.length;c++)
284   {
285     canUse[c]=true;
286   }
287   System.out.println("A new game has been started between client " +
288       connection1.getConnectionID() + " and a bot");
289 }
290 //Reference to the dealer
291   Dealer dealer = new Dealer();
292   //Variables for holding dealt hands
293   //int[] hand1 = new int[HAND_SIZE];
294   //int[] hand2 = new int[HAND_SIZE];
295   //Variables for holding played card values
296   int value1 = -1;
297   int value2 = -1;
//variables for holding player scores
int score1 = 0;
int score2 = 0;

//variables for holding messages
StandardMessage message;
StandardMessage message1;
StandardMessage message2;
StandardMessage message3;

//Send the player the name of the bot
try{
    connection1.getMessageHandler().sendMessage(new StandardMessage(2,
        GameMessageType.PLAYER_HANDLE.ordinal(), bot_name));
} catch(InterruptedException e){
    System.err.println("Could not send "+connection1.getHandle() + "
        the name of the bot; disconnecting");
    connection1.getMessageHandler().disconnectHandling();
    return 100000;
}

// get the game mode, 3,4 or 5 card modes
// try{
    // message = connection1.getMessageHandler().receiveServerMessage();
    // if(message.getType() == MessageType.SELECTION.ordinal()){
    //     gameLen = message.convertInt();
    // } else{
    //     System.err.println("MainMenu");
    //     System.err.println(message.getType());
    //     System.err.println("An error has occurred in message ordering (SELECTION)");
    //     connection1.getMessageHandler().disconnectHandling();
    //     throw new NullPointerException();
    // }
catch(NullPointerException npe)
{
    System.err.println("The message queues have been closed, a
disconnect has occurred");
    Thread.currentThread().interrupt();
}

//Shuffle the deck
dealer.shuffleDeck(1000);

//Deal five cards to player 1
hand1 = dealer.dealFive();

int totalScore1=0;
int totalScore2=0;
int bestScore=0;
boolean goodToPlay =false;

//Deal five cards to player 2 (the bot)
hand2 = dealer.dealFive();
sortBotHand(hand2);

//Checks to make sure that you cant have an automatic win
//while(!goodToPlay)
{
    totalScore1 = 0;
    totalScore2 = 0;
    System.out.println("Checking if hands are fair");
    //Variable used to count how many permutations of cards play
    allow a player to win
    for(int c=0; c<5;c++)//the number of possible permutations is
    25 or 5x5
    {
        for(int v=0;v<5;v++)
        {
            if(hand1[c]>hand2[v])
            {
                totalScore1=totalScore1 + (13-(hand1[c]-hand2[v]));
            }
        }
    }
}
for (int c = 0; c < 5; c++) // the number of possible permutations is 25 or 5x5
{ //
    for (int v = 0; v < 5; v++) {
        if (hand2[c] > hand1[v])
            totalScore2 = totalScore2 + (13 - (hand2[c] - hand1[v]));
    // }
}
if (totalScore1 - totalScore2 < 10 && totalScore2 - totalScore1 < 10) // checks how many cards are greater than opponents and vice versa if unfair reshuffle
{
    goodToPlay = true;
    }
else
{
    dealer = new Dealer();
    dealer.shuffleDeck(1000);
    hand1 = dealer.dealFive();
    hand2 = dealer.dealFive();
}
if (sendHand(connection1, hand1) == false){
    System.err.println("A disconnection has occurred");
    System.err.println("The game will end");
    connection1.getMessageHandler().disconnectHandling();
    return 100000;
}
System.out.println("Player 1’s hand is ");
for (int i = 0; i < 5; i++){
    System.out.print(hand1[i] + " ");
}
System.out.println("Player 2’s hand is ");
for (int i = 0; i < 5; i++){
System.out.println(hand2[i] + " ");

if(sendHand(connection1, hand2) == false){
    System.err.println("A disconnection has occurred");
    System.err.println("The game will end");
    connection1.getMessageHandler().disconnectHandling();
    return 100000;
}

//receive/Dequeue Character Card Information
message2 = connection1.getMessageHandler().receiveGameMessage();
try{
    connection1.getMessageHandler().sendMessage(new StandardMessage
        (2, GameMessageType.CARDFACE.ordinal(), "PNG-cards-1.3/card_")
        );
}
catch(InterruptedException e)
{
    System.err.println("Can’t send to " + connection1.getHandle());
    connection1.getMessageHandler().disconnectHandling();
}
message3 = connection1.getMessageHandler().receiveGameMessage();
try{
    connection1.getMessageHandler().sendMessage(message3);
}
catch(InterruptedException e)
{
    System.out.println("Can’t Send To " + connection1.getHandle());
    connection1.getMessageHandler().disconnectHandling();
}

//Begin the game; lasts for GAME_LENGTH turns
System.out.println("The game has begun");
for(int i = 0; i < gameLen; i++){
    //Obtain played card values
443 System.out.println("Waiting for player card values");
444
445 // For player 1:
446 try{
447 message1 = connection1.getMessageHandler().receiveGameMessage();
448 if(message1.getType() == GameMessageType.CARD.ordinal()){
449     value1 = message1.convertInt();
450 } else{
451     System.err.println("BOT Game Loop");
452     System.err.println("An error has occurred in message ordering (SELECTION)");
453     connection1.getMessageHandler().disconnectHandling();
454     throw new NullPointerException();
455 }
456
457 // Check to see that the card played was contained in the hand
458 if(checkHand(value1, hand1) == false){
459     System.err.println("Hand modification was detected from player 1; booting player");
460     connection1.getMessageHandler().disconnectHandling();
461     return 1000000;
462 }
463 }
464 catch(NullPointerException npe){
465     System.err.println("The message queues have been closed, a disconnect has occurred");
466     System.err.println("The game will end");
467     return 100000;
468 }
469
470 // For the bot:
471 value2 = getValue(hand2, bot_hand_index);
472 if(value2 == 0)
473 {
474     System.err.println("ERROR NO MORE CARDS IN HAND");
475     value2 = hand2[bot_hand_index];
476 }
bot_hand_index++;

// Show player 1 the value of the card played by the bot
try{
    connection1.getMessageHandler().sendMessage(new StandardMessage(2, GameMessageType.CARD.ordinal(), value2));
} catch(InterruptedException e){
    System.err.println("Could not send " + connection1.getHandle() + " the opponent’s card; disconnecting");
    connection1.getMessageHandler().disconnectHandling();
    return 1000000;
}

// Update the score values of the players
score1 += calculateScore(value1, value2);
score2 += calculateScore(value2, value1);

System.out.println("The outcome of the round was " + connection1.getHandle() + ": " + score1 + " + bot_name + ": " + score2);

// Send player 1 their new score as well as the current score of the bot
try{
    connection1.getMessageHandler().sendMessage(new StandardMessage(2, GameMessageType.SCORE.ordinal(), score1));
} catch(InterruptedException e){
    System.err.println("Could not send " + connection1.getHandle() + " their new score; disconnecting");
    connection1.getMessageHandler().disconnectHandling();
    return 1000000;
}

try{
    connection1.getMessageHandler().sendMessage(new StandardMessage(2, GameMessageType.SCORE.ordinal(), score2));
}
catch(InterruptedException e){
    System.err.println("Could not send " + connection1.getHandle() + " their opponent’s new score; disconnecting");
    connection1.getMessageHandler().disconnectHandling();
    return 1000000;
}

// if (connection1.getNumGames() >=10)
//    {  
//        connection1.newStrategy();  
//    }
// else
//    {
//        connection1.incrementNumGames();  
//    }
//
//    wins++;  
//    points=score1;
if (score1 > score2) {
    connection1.getRecord().setWins(connection1.getRecord().getWins() + 1);
    connection1.getRecord().setGames(connection1.getRecord().getGames () + 1);
    //connection1.getRecord().setRewardPoints(connection1.getRecord() .getRewardPoints() + score1);
    wins++;  
    points=score1;
} else if (score2 > score1) {
    connection1.getRecord().setLosses(connection1.getRecord().getLosses () + 1);
    connection1.getRecord().setGames(connection1.getRecord().getGames () + 1);
    //connection1.getRecord().setRewardPoints(connection1.getRecord() .getRewardPoints() + ((score1 - score2)*2));
    losses++;  
    points=score1 - (score1 - score2)*2;
    //points=score1;
} else {
    //connection1.getRecord().setRewardPoints(connection1.getRecord() .getRewardPoints() -connection1.getBetAmount());
connection1.getRecord().setGames(connection1.getRecord().getGames() + 1);

try {
  // connection1.getMessageHandler().sendMessage(new StandardMessage(2, GameMessageType.BET.ordinal(), connection1.getBetAmount()));
  System.err.println("Could not send " + connection1.getHandle() + " their opponent's new score; disconnecting");
  connection1.getMessageHandler().disconnectHandling();
  return 1000000;
} catch (InterruptedException e) {
  System.err.println("Could not send " + connection1.getHandle() + " their opponent's new score; disconnecting");
  connection1.getMessageHandler().disconnectHandling();
  return 1000000;
}

public int getLargest(int[] hand)
{
  int largestSpot = 0;
  int largest = 0;
  for (int c = 0; c < hand.length; c++)
  {
    if (hand[c] > largest && canUse[c] == true)
    {
      largest = hand[c];
      largestSpot = c;
    }
  }
  canUse[largestSpot] = false;
  return largest;
}

public void sortBotHand(int[] bhand)
{
  int largestSpot = 0;
  int largest = 0;
  for (int v = 0; v < bhand.length; v++)
  {
largest = 0;
for (int c = 0; c < bhand.length; c++) {
    if (bhand[c] >= largest && canUse[c] == true) {
        largest = bhand[c];
        largestSpot = c;
    }
}
canUse[largestSpot] = false;
hand2highlow[v] = largestSpot;
}

public int getValue(int[] hand, int bot_hand_index) {
    int value = 0;
    switch (strategy) {
        case 1:
            value = hand[hand2highlow[bot_hand_index]]; // gets the highest card
            break;
        case 2:
            value = hand[hand2highlow[(HAND_SIZE - bot_hand_index - 1)]]; // gets the lowest card
            break;
        case 3:
            value = hand[hand2highlow[(Math.floorMod((bot_hand_index + 3), HAND_SIZE))]]; // goes low->high->mid
            break;
    }
    return value;
}

// Returns true if send successful, false otherwise
/**
 * Send the series of card values to the given client constituting the hand they have been dealt.
 *<p>
 * Each card is sent individually as a game-specific message
 */
public boolean sendHand(ConnectionHandler connection, int[] hand){
    boolean flag = true;
    for(int i = 0; i < HAND_SIZE; i++){
        try{
            connection.getMessageHandler().sendMessage(new StandardMessage(2,
                                         GameMessageType.CARD.ordinal(), hand[i]));
        }
        catch(InterruptedException e){
            System.err.println("Could not send a card to " + connection.getHandle());
            flag = false;
        }
    }
    return flag;
}

/**
 * Given two card values, calculate the score outcome.
 *
 * <p>
 * Score outcome is calculated for the client who has played a card with
 * the value of 'valSelf'. Hence if a score outcome is
 * to be calculated for each player, this method must be called twice,
 * reversing their input parameters in each case.
 *
 * @param valSelf the card value played by the client desiring their
 * score outcome
 * @param valOpp the card value played by the opponent
 * @return the score outcome
 */
public int calculateScore(int valSelf, int valOpp){
    System.out.println("properties: "+scoreType);
}
```java
int score = 0;
int maxDiff = 13;

if(valSelf > valOpp & scoreType == 1){
    score = 1;
}
if(valSelf > valOpp & scoreType == 2){
    score = maxDiff - (valSelf - valOpp);
}
return score;
}
/**
 * Check to see if a given value is contained within the hand.
 *
 * This method is used to prevent client hand modification as a client
 * should not be able to play a card they have not been
 * explicitly dealt.
 *
 * @param value the value of the card played
 * @param hand the player hand
 * @return true if the value is contained within the hand, false
 *         otherwise
 */
public boolean checkHand(int value, int[] hand){
    System.out.println("Checking hand");
    System.out.println(value);
    boolean flag = false;
    for(int i = 0; i < hand.length; i++){
        System.out.println(hand[i]);
        if(value == hand[i]){
            flag = true;
        }
    }
    return flag;
}
/**
 * Author: Kyle Leduc-McNiven
 */
```

Page 92
This method handles the case where a player disconnects during a PVP game.
This method sends an error message to the still active client so that they can replay without needing to restart the application.

@param connection This is the connection that disconnected

```java
public void handleDisconnectError(ConnectionHandler connection) {
    if (!sentDisconnect) // Checks to see if an error message has already been sent
    {
        sentDisconnect = true;
        try
        {
            // println for debugging purposes
            System.out.println("Handling disconnect");
            System.out.println(connection.getHandle());
            System.out.println(connection1.getHandle());
            System.out.println(connection2.getHandle());
            System.out.println(connection.getHandle().equals(connection1.getHandle()));
            System.out.println(connection.getHandle().equals(connection2.getHandle()));
            if ((connection.getHandle().equals(connection1.getHandle())) &&
            connection2 != null) // Gets the Connection that is still Active
            {
                System.out.println("Sending message about Disconnection to " + connection2.getHandle());
                //connection2.getMessageHandler().sendMessage(new StandardMessage(2,GameMessageType.CARD.ordinal(),5));
                connection2.getMessageHandler().clearGameBuffer();
                if (connect1Sent)
                {
                    connection2.getMessageHandler().sendMessage(new StandardMessage(4,ErrorMessageType.DISCONNECT2.ordinal(),"ERROR SEND RESPONSE"));
                }
                else
```
{  
    connection2.getMessageHandler().sendMessage(new StandardMessage(4, ErrorMessageType.DISCONNECT2.ordinal(), "Disconnect Occurred"));
}  

/// Sends an error message to the active client so that they can quit the game
connection2.getMessageHandler().sendMessage(new StandardMessage(2, GameMessageType.BET.ordinal(), connection2.getBetAmount()));

new Thread(new MainMenu(queueHandler, connection2)).start();  

}  

else if ((connection.getHandle().equals(connection2.getHandle())) && connection1 != null)  
{
    System.out.println("Sending message about Disconnection to " + connection1.getHandle());

    // connection1.getMessageHandler().sendMessage(new StandardMessage(2, GameMessageType.CARD.ordinal(), 5));
    connection1.getMessageHandler().clearGameBuffer();
    connection1.getMessageHandler().sendMessage(new StandardMessage(4, ErrorMessageType.DISCONNECT1.ordinal(), "Disconnect Occurred"));
    connection1.getMessageHandler().sendMessage(new StandardMessage(2, GameMessageType.BET.ordinal(), connection1.getBetAmount()));
    new Thread(new MainMenu(queueHandler, connection1)).start();
}

}  

}  

catch(InterruptedException e)  
{  
    System.err.println("Error Occured when trying to send Disconnect Disconnecting "+connection.getHandle());  
    // doesn't send error if both connections are disconnected
    // connection.getMessageHandler().disconnectHandling();  
    infinite Loop if both clients disconnect

}  

}
public String toAlphabetic(int i) {
    if (i < 0) {
        return "-" + toAlphabetic(-i-1);
    }

    int quot = i / 26;
    int rem = i % 26;
    char letter = (char)((int)'A' + rem);
    if (quot == 0) {
        return "" + letter;
    }
    else {
        return toAlphabetic(quot - 1) + letter;
    }
}
A.2 Hardware

Figure 35: Hardware test setup
Figure 36: Console output of the heartbeat sensor
Figure 37: Console output of the temperature sensor
**Capstone Arduino Code** - CapstoneSensorIntegration.ino

### A.3 Machine Learning

#### Table 4: Machine Learning Framework Software Comparison

<table>
<thead>
<tr>
<th>Tool</th>
<th>Name</th>
<th>Owner</th>
<th>Score</th>
<th>Price</th>
<th>Interface</th>
<th>Open Source</th>
<th>CSDR support</th>
<th>Architecture</th>
<th>Documents</th>
<th>Resources</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keras</td>
<td>N/A</td>
<td>Open Source</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>Google</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Scikit-learn</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>PyTorch</td>
<td>Facebook</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Theano</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>mxnet</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>TensorLow</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>TensorFlow.js</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Keras.js</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Caffe</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Caffe.js</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Theano.js</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>mxnet.js</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>TensorFlow.js</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Table 5: System Architecture Comparison

<table>
<thead>
<tr>
<th>Features</th>
<th>Parameters</th>
<th>Weights</th>
<th>Methods</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supervision</td>
<td>Unsupervised</td>
<td>9</td>
<td>Restricted Boltzmann</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>Supervised</td>
<td>4</td>
<td>Stacked auto-encoder</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>Semi-supervised</td>
<td>5</td>
<td>Convolutional NN</td>
<td>44</td>
</tr>
<tr>
<td></td>
<td>Deep</td>
<td>9</td>
<td>Deep Belief</td>
<td>41</td>
</tr>
<tr>
<td># Layers</td>
<td>Shallow</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Deep</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Pre-training</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Latent Feature</td>
<td>Yes</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Detection</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dimensionality</td>
<td>Yes</td>
<td>?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reduction</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Built-in</td>
<td>Yes</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Classification</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Built-in Fine</td>
<td>Yes</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tuning</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6: Numerical Methods Comparison

<table>
<thead>
<tr>
<th>Features</th>
<th>Framework Weights (1 low to 10 high)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cost Function</td>
</tr>
<tr>
<td></td>
<td>Mean-squared error</td>
</tr>
<tr>
<td></td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>7</td>
</tr>
</tbody>
</table>
### Table 7: Performance results of different architectures

<table>
<thead>
<tr>
<th>NN Architecture</th>
<th>Simple NN</th>
<th>CNN v1</th>
<th>Proposed CNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost Function</td>
<td>Softmax</td>
<td>Softmax cross entropy</td>
<td>Max Likelihood Est.</td>
</tr>
<tr>
<td>Optimization Function</td>
<td>Gradient Descent</td>
<td>Adam Optimizer</td>
<td>Gradient Descent</td>
</tr>
<tr>
<td>Activation Function</td>
<td>Softmax</td>
<td>ReLU</td>
<td>ReLU</td>
</tr>
<tr>
<td>Learning Rate</td>
<td>0.005</td>
<td>0.001</td>
<td>TBD</td>
</tr>
<tr>
<td>Training Iterations</td>
<td>60k</td>
<td>200k</td>
<td>60k</td>
</tr>
<tr>
<td>Batch Size</td>
<td>100</td>
<td>128</td>
<td>TBD</td>
</tr>
<tr>
<td>Strides</td>
<td>n/a</td>
<td>1</td>
<td>TBD</td>
</tr>
<tr>
<td>Dropout %</td>
<td>0</td>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td><strong>Testing Accuracy</strong></td>
<td><strong>92.68%</strong></td>
<td><strong>98.43%</strong></td>
<td>TBD</td>
</tr>
</tbody>
</table>

**Figure 39: High-example of a potential GAN architecture**

**Capstone MCI CNN - CapstoneCNN-MNISTv1.py**

```python
from __future__ import print_function
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("data", one_hot=True)
```

Page 101
#define hyperparameters

learning_rate = 0.001
training_iters = 1000
batch_size = 100

a = training_iters / batch_size
n_batches = int(a)
display_step = 10

#Network Parameters

n_input = 784  #28x28
n_classes = 10  #10 different digits to classify
dropout = 1  #keep probability; prevents overfitting etc by Hinton

x = tf.placeholder(tf.float32, [None, n_input])
y = tf.placeholder(tf.float32, [None, n_classes])
keep_prob = tf.placeholder(tf.float32)

#Create convolutional function
def conv2d(x, W, b, strides=1):
    # Conv2D wrapper, with bias and relu activation
    x = tf.nn.conv2d(x, W, strides=[1, strides, strides, 1], padding='SAME')
    x = tf.nn.bias_add(x, b)
    return tf.nn.relu(x)

def maxpool2d(x, k=2):
    # MaxPool2D wrapper
    return tf.nn.max_pool(x, ksize=[1, k, k, 1], strides=[1, k, k, 1], padding='SAME')

# Create model
def conv_net(x, weights, biases, dropout):
    # Reshape input picture
    x = tf.reshape(x, shape=[-1, 28, 28, 1])

    # First Conv layer
    conv1 = conv2d(x, weights['wc1'], biases['bc1'])
    # First Down-sampling with Max Pooling
    conv1 = maxpool2d(conv1, k=2)


```python
# Second Conv layer
conv2 = conv2d(conv1, weights['wc2'], biases['bc2'])

# Second Down-sampling with Max Pooling
conv2 = maxpool2d(conv2, k=2)

# Fully connected layer
# Reshape conv2 output to fit fully connected layer input
fc1 = tf.reshape(conv2, [-1, weights['wd1'].get_shape().as_list()[0]])
fc1 = tf.add(tf.matmul(fc1, weights['wd1']), biases['bd1'])
fc1 = tf.nn.relu(fc1)

# Apply Dropout
fc1 = tf.nn.dropout(fc1, dropout)

# Output, class prediction
output = tf.add(tf.matmul(fc1, weights['out']), biases['out'])
return output

# Store layers weight & bias
weights = {
    # 5x5 conv, 1 input, 32 outputs
    'wc1': tf.Variable(tf.random_normal([5, 5, 1, 32])),
    # 5x5 conv, 32 inputs, 64 outputs
    'wc2': tf.Variable(tf.random_normal([5, 5, 32, 64])),
    # fully connected, 7x7x64 inputs, 1024 outputs
    'wd1': tf.Variable(tf.random_normal([7*7*64, 1024])),
    # 1024 inputs, 10 outputs (class prediction)
    'out': tf.Variable(tf.random_normal([1024, n_classes]))
}

biases = {
    # 32 outputs
    'bc1': tf.Variable(tf.random_normal([32])),
    'bc2': tf.Variable(tf.random_normal([64])),
    'bd1': tf.Variable(tf.random_normal([1024])),
    'out': tf.Variable(tf.random_normal([n_classes]))
}
```

# Construct model
pred = conv_net(x, weights, biases, keep_prob)

# Define loss and optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=pred, labels=y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)

# Evaluate model
correct_pred = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))

# Initializing the variables
init = tf.initialize_all_variables()

# Launch the graph with tf.Session() as sess:
sess.run(init)
step = 1
for i in range(n_batches):
  while i * batch_size < training_iters:
    batch_x, batch_y = mnist.train.next_batch(batch_size)
    sess.run(optimizer, feed_dict={x: batch_x, y: batch_y, keep_prob: dropout})

    # Optimization Finished!
    # Testing Accuracy: 
    sess.run(accuracy, feed_dict={x: mnist.test.images[:256], y: mnist.test.labels[:256], keep_prob: 1.}))

# Train until max iterations
while step * batch_size < training_iters:
  batch_x, batch_y = mnist.train.next_batch(batch_size)
  # Run Backprop
  sess.run(optimizer, feed_dict={x: batch_x, y: batch_y, keep_prob: dropout})
  step += 1
  print("Optimization Finished!")
  print("Testing Accuracy: ", 
  sess.run(accuracy, feed_dict={x: mnist.test.images, y: mnist.test.labels, keep_prob: 1.}))
# sess.run(accuracy, feed_dict={x: mnist.test.images[:256], y: mnist.test.labels[:256], keep_prob: 1.})

## Data Manipulation - alldatasvread.m

```matlab
clear all;
close all;
clc

% %Initialize Variables
% T = zeros(28,5);
% O = zeros(5,28);
% TO = zeros(28,28);

% Read + Filter Data
% Game Data
Col13 = Strat; Col14 = Autowin; Col15 = Autoloss; Col16 = Stratwin; Col17 = Won
rawdata = readtable('alldatatest.csv');
data = rawdata;
data(:,18) = []; % remove used strat (player)
data(:,16) = []; % remove stratwin
% keep in mind that Col16 now becomes "Won?"
data(:,13) = []; % remove strat (bot)

% formatting Matrices
windata = data(:,16);
winstatus = table2array(windata);
timedata = data(:,3:7);
orderdata = data(:,8:12);
orderA = table2array(orderdata);
O = transpose(orderA);
[nRow,nCol] = size(O);

% BPM Data - Keep in mind that this is all Synthetic
Col1 = Handle; Col2 = Row ID; Col3 = 'value'; Col4 = Column ID;
rawdata = readtable('allbpmtest.csv');
bpmdata = rawdata;
BPMdat = table2array(bpmdata);
BPM = zeros(28,28); % how big do I really want this?
```
\[
\begin{align*}
\text{Group } 19: \text{ Final Report} \\
A. \text{ APPENDIX} \\
\end{align*}
\]

35 \[n_{\text{BPMrow}}, n_{\text{BPMcol}} = \text{size}(\text{BPMdat});\]
36 \%filter out 0 entries + reorganize the matrix to match the TIME matrix; 28 rows?
37 \textbf{for } i=1:n_{\text{BPMrow}}
38 \hspace{1em} \textbf{if } \text{BPMdat}(i,3) == 0
39 \hspace{2em} \text{BPMdat}(i,:) = 0;
40 \hspace{2em} \textbf{end}
41 \textbf{for } j=1:n_{\text{BPMcol}}
42 \hspace{2em} \textbf{end}
43 \textbf{end}
44
45 \%@@ Temp Data @@ – Keep in mind that this is all Synthetic
46 \%Col1 = Handle; Col2 = Row ID; Col3 = 'value'; Col4 = Column ID;
47 \text{rawdata} = \text{readtable}('\text{alltemptest.csv}');
48 \text{tempdata} = \text{rawdata};
49 \textbf{for } i=1:n_{\text{BPMrow}}
50 \hspace{1em} \textbf{if } \text{tempdata}(i,3) == 0
51 \hspace{2em} \text{tempdata}(i,:) = 0;
52 \hspace{2em} \textbf{end}
53 \textbf{end}
54
55 \%@@ ORDER @@
56 \% \text{scaledO} = \text{csvOrderLookup}(n_{\text{Col}}, n_{\text{Row}}, O);
57 \textbf{for } j=1:n_{\text{Col}}
58 \hspace{1em} \textbf{for } i=1:n_{\text{Row}}
59 \hspace{2em} \%j = order indexer
60 \hspace{2em} \%where does Order(:,j) = 0? // finding where lowest card is played
61 \hspace{2em} \%for the given game
62 \hspace{2em} \textbf{if } O(i,j) == 0; \%in which row does i=0 occur? based on position, assign values
63 \hspace{2em} \textbf{if } i==1; \%if O=O occurs in 1st pos
64 \hspace{3em} \text{scaledO}(i,j) = 0.9;
65 \hspace{2em} \textbf{elseif } i==2; \%if O=O occurs in 2nd pos
66 \hspace{3em} \text{scaledO}(i,j) = 0.15;
67 \hspace{2em} \textbf{elseif } i==3; \%if O=O occurs in 3rd pos
68 \hspace{3em} \text{scaledO}(i,j) = 0.25;
69 \hspace{2em} \textbf{elseif } i==4; \%if O=1 occurs in 4th pos
70 \hspace{2em} \text{scaledO}(i,j) = 0.4;
71 \textbf{end}

Page 106
scaledO(i,j) = 0.5;

elseif i==5; %if O=1 occurs in 5th pos
    scaledO(i,j) = 0.6;

end

if O(i,j) == 1; %where does Order(:,j) = 1?
    if i==1; %if O=1 occurs in first pos
        scaledO(i,j) = 0.5;
    elseif i==2; %if O=1 occurs in 2nd pos
        scaledO(i,j) = 0.15;
    elseif i==3; %if O=1 occurs in 3rd pos
        scaledO(i,j) = 0.15;
    elseif i==4; %if O=1 occurs in 4th pos
        scaledO(i,j) = 0.5;
    elseif i==5; %if O=1 occurs in 5th pos
        scaledO(i,j) = 0.9;
    end

end

if O(i,j) == 2; %where does Order(:,j) = 2?
    if i==1; %if O=2 occurs in first pos
        scaledO(i,j) = 0.25;
    elseif i==2; %if O=2 occurs in 2nd pos
        scaledO(i,j) = 0.15;
    elseif i==3; %if O=2 occurs in 3rd pos
        scaledO(i,j) = 0.33;
    elseif i==4; %if O=2 occurs in 4th pos
        scaledO(i,j) = 0.9;
    elseif i==5; %if O=2 occurs in 5th pos
        scaledO(i,j) = 0.75;
    end

end

if O(i,j) == 3; %where does Order(:,j) = 3?
    if i==1; %if O=3 occurs in first pos
        scaledO(i,j) = 0.15;
    elseif i==2; %if O=3 occurs in 2nd pos
        scaledO(i,j) = 0.75;
    elseif i==3; %if O=3 occurs in 3rd pos
        scaledO(i,j) = 0.9;
    elseif i==4; %if O=3 occurs in 4th pos
        scaledO(i,j) = 0.33;
    end

end

end

end

end

end

end
111 \[ \text{scaledO}(i,j) = 0.75; \]
112 elseif \( i == 5 \); \%if O=3 occurs in 5th pos
113 \[ \text{scaledO}(i,j) = 0.25; \]
114 end
115 end
116 if \( O(i,j) == 4 \); \%where does Order(:,j) = 4?
117 if \( i == 1 \); \%if O=4 occurs in first pos
118 \[ \text{scaledO}(i,j) = 0.5; \]
119 elseif \( i == 2 \); \%if O=4 occurs in 2nd pos
120 \[ \text{scaledO}(i,j) = 0.9; \]
121 elseif \( i == 3 \); \%if O=4 occurs in 3rd pos
122 \[ \text{scaledO}(i,j) = 0.75; \]
123 elseif \( i == 4 \); \%if O=4 occurs in 4th pos
124 \[ \text{scaledO}(i,j) = 0.25; \]
125 elseif \( i == 5 \); \%if O=4 occurs in 5th pos
126 \[ \text{scaledO}(i,j) = 0.15; \]
127 end
128 end
129 end
130 end
131
132 \%TIME
133 \[ \text{invT} = \min(\text{T}(:))./\text{T}; \]
134
135 \%Matrix Multiplication
136 \[ \text{mmult} = \text{mtimes}(\text{invT},\text{scaledO}); \]
137
138 \%Win/Loss Result Scaling
139 \[ [p,q] = \text{size}(\text{winstatus}); \]
140 \[ [r,s] = \text{size}(\text{mmult}); \]
141 \% \text{mmult2} = \text{X}.*\text{eye}(r);
142 \% \text{MLin} = \text{mmult2}.*\text{W};
143 \[ \text{wlScale} = \text{zeros}(33,33); \]
144 for \( k=1:p \)
145 if \( \text{strcmp}(\text{winstatus}(k,:), 'TRUE') == 1 \)
146 \[ \%W(k,:) = 0.25; \]
147 \[ \text{wlScale}(k,:) = \text{mmult}(k,:) + 0.25; \]
148 else
149 \[ \%W(k,:) = -0.5; \]
\begin{verbatim}
  w1Scale(k,:) = mmult(k,:) - 0.5;
  end
  end
  norm = w1Scale - min(w1Scale);
  X = norm./max(norm);
  MLin = X.*eye(r);

  % Matrix Multiplication
  [r,s] = size(X);
  mmult2 = X.*eye(r);
  MLin = mmult2.*W;
\end{verbatim}

\textbf{Tutorial 1: Custom Visualization}

\begin{verbatim}
import tensorflow as tf
import tensorflowvisu
from tensorflow.examples.tutorials.mnist import input_data as mnist_data
print("Tensorflow version " + tf.__version__)  

tf.set_random_seed(0)

# neural network with 1 layer of 10 softmax neurons
# [input data, flattened pixels]  X
# \( 784 = 28 \times 28 \) 
# fully connected layer (softmax)  W \[784, 10\]
\end{verbatim}
# The model is:

# Y = softmax( X * W + b)
# X: matrix for 100 grayscale images of 28x28 pixels, flattened
# (there are 100 images in a mini-batch)
# W: weight matrix with 784 lines and 10 columns
# b: bias vector with 10 dimensions
# +: add with broadcasting: adds the vector to each line of the
#   matrix (numpy)
# softmax(matrix) applies softmax on each line
# softmax(line) applies an exp to each value then divides by the
#   norm of the resulting line
# Y: output matrix with 100 lines and 10 columns

# Download images and labels into mnist.test (10K images+labels) and mnist.
# train (60K images+labels)

mnist = mnist_data.read_data_sets("data", one_hot=True, reshape=False,
                                   validation_size=0)

# input X: 28x28 grayscale images, the first dimension (None) will index the
# images in the mini-batch

X = tf.placeholder(tf.float32, [None, 28, 28, 1])
# correct answers will go here
Y_ = tf.placeholder(tf.float32, [None, 10])
# weights W[784, 10] 784=28*28
W = tf.Variable(tf.zeros([784, 10]))
# biases b[10]
b = tf.Variable(tf.zeros([10]))

# flatten the images into a single line of pixels
# -1 in the shape definition means "the only possible dimension that will
#   preserve the number of elements"
XX = tf.reshape(X, [-1, 784])

# The model
56 \[ Y = \text{tf.nn.softmax}(\text{tf.matmul}(X, W) + b) \]

57 # loss function: cross-entropy = - sum( Y_i * log(Y_i) )
58 # Y: the computed output vector
59 # Y_: the desired output vector
60
cross_entropy = -\text{tf.reduce_mean}(Y_ * \text{tf.log}(Y)) * 1000.0 # normalized for
61 # batches of 100 images,
62
tensorflowvisu.tf_format_mnist_images(X, Y, Y_) # assembles 10x10 images
63
64 # accuracy of the trained model, between 0 (worst) and 1 (best)
65 correct_prediction = tf.equal(tf.argmax(Y, 1), tf.argmax(Y_, 1))
66 accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
67
68 # training, learning rate = 0.005
69 train_step = tf.train.GradientDescentOptimizer(0.005).minimize(cross_entropy)
70
71 # matplotlib visualisation
72 allweights = tf.reshape(W, [-1])
73 allbiases = tf.reshape(b, [-1])
74 It = tensorflowvisu.tf_format_mnist_images(X, Y, Y_) # assembles 10x10 images
75 # init
76 init = tf.global_variables_initializer()
# You can call this function in a loop to train the model, 100 images at a time

def training_step(i, update_test_data, update_train_data):
    # training on batches of 100 images with 100 labels
    batch_X, batch_Y = mnist.train.next_batch(100)
    # compute training values for visualisation
    if update_train_data:
        a, c, im, w, b = sess.run([accuracy, cross_entropy, I, allweights, allbiases], feed_dict={X: batch_X, Y_: batch_Y})
        datavis.append_training_curves_data(i, a, c)
        datavis.append_data_histograms(i, w, b)
        datavis.update_image1(im)
        print(str(i) + ': accuracy:' + str(a) + ' loss: ' + str(c))

    # compute test values for visualisation
    if update_test_data:
        a, c, im = sess.run([accuracy, cross_entropy, I], feed_dict={X: mnist.test.images, Y_: mnist.test.labels})
        datavis.append_test_curves_data(i, a, c)
        datavis.update_image2(im)
        print(str(i) + ': ********** epoch ' + str(i*100//mnist.train.images.shape[0]+1) + ' ********** test accuracy:' + str(a) + ' test loss: ' + str(c))

    # the backpropagation training step
    sess.run(train_step, feed_dict={X: batch_X, Y_: batch_Y})

datavis.animate(training_step, iterations=2000+1, train_data_update_freq=10, test_data_update_freq=50, more_tests_at_start=True)

# to save the animation as a movie, add save_movie=True as an argument to datavis.animate
# to disable the visualisation use the following line instead of the datavis.animate line
# for i in range(2000+1): training_step(i, i % 50 == 0, i % 10 == 0)
print("max test accuracy: " + str(datavis.get_max_test_accuracy()))

# final max test accuracy = 0.9268 (10K iterations). Accuracy should peak above 0.92 in the first 2000 iterations.
# Y_: tensor of shape [100+, 10] containing correct digit labels (one-hot vectors)

# return value: tensor of shape [280, 280, 3] containing the 100 first unrecognised images (rgb, uint8)

# followed by other, recognised images. 100 images max arranged as a 10x10 array. Unrecognised images
# are displayed on a red background and labeled with the correct (left) and recognised digit (right).

```python
def tf_format_mnist_images(X, Y, Y_, n=100, lines=10):
    correct_prediction = tf.equal(tf.argmax(Y, 1), tf.argmax(Y_, 1))
    correctly_recognised_indices = tf.squeeze(tf.where(correct_prediction), [1])  # indices of correctly recognised images
    incorrectly_recognised_indices = tf.squeeze(tf.where(tf.logical_not(correct_prediction)), [1])  # indices of incorrectly recognised images
    everything_incorrect_first = tf.concat([[incorrectly_recognised_indices, incorrectly_recognised_indices], 0)  # images reordered with indeces of unrecognised images first
    everything_incorrect_first = tf.slice(everything_incorrect_first, [0], [n])  # compute first 100 only - no space to display more anyway
    # compute n=100 digits to display only
    Xs = tf.gather(X, everything_incorrect_first)
    Ys = tf.gather(Y, everything_incorrect_first)
    Ys_ = tf.gather(Y_, everything_incorrect_first)
    correct_prediction_s = tf.gather(correct_prediction, everything_incorrect_first)
    digits_left = tf.image.grayscale_to_rgb(tensorflowvisu_digits.digits_left())
    correct_tags = tf.gather(digits_left, tf.argmax(Ys_, 1))  # correct digits to be printed on the images
    digits_right = tf.image.grayscale_to_rgb(tensorflowvisu_digits.digits_right())
    computed_tags = tf.gather(digits_right, tf.argmax(Ys, 1))  # computed digits to be printed on the images
    #superimposed_digits = correct_tags+computed_tags
    superimposed_digits = tf.where(correct_prediction_s, tf.zeros_like(correct_tags), correct_tags+computed_tags)  # only pring the correct and computed digits on unrecognised images
```

Page 114
```python
correct_bkg = tf.reshape(tf.tile([1.3, 1.3, 1.3], [28*28]), [1, 28, 28, 3])
# white background
incorrect_bkg = tf.reshape(tf.tile([1.3, 1.0, 1.0], [28*28]), [1, 28, 28, 3])
# red background
recognised_bkg = tf.gather(tf.concat([incorrect_bkg, correct_bkg], 0), tf.cast(correct_prediction_s, tf.int32)) # pick either the red or the white background depending on recognised status

I = tf.image.grayscale_to_rgb(Xs)
I = ((1-(I+superimposed_digits))*recognised_bkg)/1.3 # stencil extra data on top of images and reorder them unrecognised first
I = tf.image.convert_image_dtype(I, tf.uint8, saturate=True)
I_slices = [] # 100 images => 10x10 image block
for imslice in range(lines):
    I_slices.append(tf.concat(tf.unstack(tf.slice(I, [imslice*n//lines, 0, 0, 0], [n//lines, 28, 28, 3])), 1))
I = tf.concat(I_slices, 0)
return I

# n = HISTOGRAM_BUCKETS (global)
# Buckets the data into n buckets so that there are an equal number of data points in
# each bucket. Returns n+1 bucket boundaries. Spreads the remainder data.size % n more
# or less evenly among the central buckets.
# data: 1-D ndarray containing float data, MUST BE SORTED in ascending order
# n: integer, the number of desired output buckets
# return value: ndarray, 1-D vector of size n+1 containing the bucket boundaries
# the first value is the min of the data, the last value is the max

def probability_distribution(data):
    n = HISTOGRAM_BUCKETS
data.sort()
bucketsize = data.size // n
bucketrem = data.size % n
buckets = np.zeros([n+1])
buckets[0] = data[0] # min
buckets[-1] = data[-1] # max
```

Page 115
buckn = 0
rem = 0
remn = 0
k = 0
cnt = 0 # only for assert
lastval = data[0]
for i in range(data.size):
    val = data[i]
buckn += 1
cnt += 1
if buckn > bucketsize+rem : ## crossing bucket boundary
cnt -- = 1
k += 1
buckets[k] = (val + lastval) / 2
if (k<n+1):
cnt += 1
buckn = 1 # val goes into the new bucket
if k >= (n - bucketrem) // 2 and remn < bucketrem:
    rem = 1
remn += 1
else:
    rem = 0
lastval = val
assert i+1 == cnt
return buckets

def _empty_collection(collection):
tempcoll = []
for a in (collection):
tempcoll.append(a)
for a in (tempcoll):
collection.remove(a)

def _display_time_histogram(ax, xdata, ydata, color):
    _empty_collection(ax.collections)
    midl = HISTOGRAM_BUCKETS//2
    midh = HISTOGRAM_BUCKETS//2
    for i in range(int(math.ceil(HISTOGRAM_BUCKETS/2.0))):
        ax.fill_between(xdata, ydata[:,midl-i], ydata[:,midh+i+1], facecolor=
if HISTOGRAM_BUCKETS % 2 == 0 and i == 0:
    ax.fill_between(xdata, ydata[:, midl-1], ydata[:, midh], facecolor=color, alpha=1.6/HISTOGRAM_BUCKETS)
midl = midl-1

class MnistDataVis:
    xmax = 0
    y2max = 0
    x1 = []
    y1 = []
    z1 = []
    x2 = []
    y2 = []
    z2 = []
    x3 = []
    w3 = np.zeros([0, HISTOGRAM_BUCKETS+1])
    b3 = np.zeros([0, HISTOGRAM_BUCKETS+1])
    im1 = np.full((28*10, 28*10, 3), 255, dtype='uint8')
    im2 = np.full((28*10, 28*10, 3), 255, dtype='uint8')
_animpause = False
_animation = None
_mpl_figure = None
_mlp_init_func = None
_mpl_update_func = None
_color4 = None
_color5 = None

    def __set_title(self, ax, title, default=''):  
        if title is not None and title != '':
            ax.set_title(title, y=1.02)  # adjustment for plot title bottom margin
        else:
            ax.set_title(default, y=1.02)  # adjustment for plot title bottom margin

    def __get_histogram_cyclecolor(self, colornum):
        clist = rcParams['axes.prop_cycle']
ccount = 1 if (colornum is None) else colornum

colors = clist.by_key()['color']
for i, c in enumerate(colors):
    if (i == count % 3):
        return c

def __init__(self, title1=None, title2=None, title3=None, title4=None, title5=None, title6=None, histogram4colornum=None, histogram5colornum=None, dpi=70):
    self._color4 = self.__get_histogram_cyclecolor(histogram4colornum)
    self._color5 = self.__get_histogram_cyclecolor(histogram5colornum)
    fig = plt.figure(figsize=(19.20, 10.80), dpi=dpi)
    plt.gcf().canvas.set_window_title("MNIST")
    fig.set_facecolor(' #FFFFFF')
    ax1 = fig.add_subplot(231)
    ax2 = fig.add_subplot(232)
    ax3 = fig.add_subplot(233)
    ax4 = fig.add_subplot(234)
    ax5 = fig.add_subplot(235)
    ax6 = fig.add_subplot(236)
    #fig, ax = plt.subplots() # if you need only 1 graph

    self._set_title(ax1, title1, default="Accuracy")
    self._set_title(ax2, title2, default="Cross entropy loss")
    self._set_title(ax3, title3, default="Training digits")
    self._set_title(ax4, title4, default="Weights")
    self._set_title(ax5, title5, default="Biases")
    self._set_title(ax6, title6, default="Test digits")

    #fig.set_figaspect(1.0)

    # TODO: finish exporting the style modifications into a stylesheet
    line1, = ax1.plot(self.x1, self.y1, label="training accuracy")
    line2, = ax1.plot(self.x2, self.y2, label="test accuracy")
    legend = ax1.legend(loc='lower right') # fancybox : slightly rounded corners
    legend.draggable(True)

    line3, = ax2.plot(self.x1, self.z1, label="training loss")
line4, = ax2.plot(self.x2, self.z2, label="test loss")
legend = ax2.legend(loc='upper right') # fancybox : slightly rounded corners
legend.draggable(True)

ax3.grid(False) # toggle grid off
ax3.set_axis_off()
imax1 = ax3.imshow(self.im1, animated=True, cmap='binary', vmin=0.0, vmax=1.0, interpolation='nearest', aspect=1.0)

ax6.grid(False) # toggle grid off
ax6.axes.get_xaxis().set_visible(False)
imax2 = ax6.imshow(self.im2, animated=True, cmap='binary', vmin=0.0, vmax=1.0, interpolation='nearest', aspect=1.0)
ax6.locator_params(axis='y', nbins=7)
# hack...
ax6.set_yticks([0, 280-4*56, 280-3*56, 280-2*56, 280-56, 280])
ax6.set_yticklabels(['100%', '98%', '96%', '94%', '92%', '90%'])

def _init():
  ax1.set_xlim(0, 10) # initial value only, autoscaled after that
  ax2.set_xlim(0, 10) # initial value only, autoscaled after that
  ax4.set_xlim(0, 10) # initial value only, autoscaled after that
  ax5.set_xlim(0, 10) # initial value only, autoscaled after that
  ax1.set_ylim(0, 1) # important: not autoscaled
  ax1.autoscale(axis='y')
  ax2.set_ylim(0, 100) # important: not autoscaled
  return imax1, imax2, line1, line2, line3, line4

def _update():
  # x scale: iterations
  ax1.set_xlim(0, self.xmax+1)
  ax2.set_xlim(0, self.xmax+1)
  ax4.set_xlim(0, self.xmax+1)
  ax5.set_xlim(0, self.xmax+1)

  # four curves: train and test accuracy, train and test loss
  line1.set_data(self.x1, self.y1)
line2.set_data(self.x2, self.y2)
line3.set_data(self.x1, self.z1)
line4.set_data(self.x2, self.z2)

# images
imax1.set_data(self.im1)
imax2.set_data(self.im2)

# histograms
_display_time_histogram(ax4, self.x3, self.w3, self._color4)
_display_time_histogram(ax5, self.x3, self.b3, self._color5)

#return changed artists
return imax1, imax2, line1, line2, line3, line4

def _key_event_handler(event):
    if len(event.key) == 0:
        return
    else:
        keycode = event.key

    # pause/resume with space bar
    if keycode == ' ':  
        self._animpause = not self._animpause
        if not self._animpause:
            _update()
            return

    # [p, m, n] p is the #of the subplot, [n,m] is the subplot layout
toggles = {'1': [1,1,1], # one plot
            '2': [2,1,1], # one plot
            '3': [3,1,1], # one plot
            '4': [4,1,1], # one plot
            '5': [5,1,1], # one plot
            '6': [6,1,1], # one plot
            '7': [12,1,2], # two plots
            '8': [45,1,2], # two plots
            '9': [36,1,2], # two plots
            'escape': [123456,2,3], # six plots
'0': [123456, 2, 3]}  # six plots

# other matplotlib keyboard shortcuts:
# 'o' box zoom
# 'p' mouse pan and zoom
# 'h' or 'home' reset
# 's' save
# 'g' toggle grid (when mouse is over a plot)
# 'k' toggle log/lin x axis
# 'l' toggle log/lin y axis

if not ( keycode in toggles):
    return

for i in range(6):
    fig.axes[i].set_visible(False)

fignum = toggles[keycode][0]
if fignum <= 6:
    fig.axes[fignum - 1].set_visible(True)
    fig.axes[fignum - 1].change_geometry(toggles[keycode][1],
                                         toggles[keycode][2], 1)
    ax6.set_aspect(25.0/40)  # special case for test digits
elif fignum < 100:
    fig.axes[fignum//10 - 1].set_visible(True)
    fig.axes[fignum//10 - 1].change_geometry(toggles[keycode][1],
                                         toggles[keycode][2], 1)
    fig.axes[fignum%10 - 1].set_visible(True)
    fig.axes[fignum%10 - 1].change_geometry(toggles[keycode][1],
                                         toggles[keycode][2], 2)
    ax6.set_aspect(1.0)  # special case for test digits
elif fignum == 123456:
    for i in range(6):
        fig.axes[i].set_visible(True)
        fig.axes[i].change_geometry(toggles[keycode][1], toggles[
                                         keycode][2], i+1)
    ax6.set_aspect(1.0)  # special case for test digits

plt.draw()
fig.canvas.mpl_connect('key_press_event', _key_event_handler)

self._mpl_figure = fig
self._mlp_init_func = _init
self._mpl_update_func = _update

def _update_xmax(self, x):
    if (x > self.xmax):
        self.xmax = x

def _update_y2max(self, y):
    if (y > self.y2max):
        self.y2max = y

def append_training_curves_data(self, x, accuracy, loss):
    self.x1.append(x)
    self.y1.append(accuracy)
    self.z1.append(loss)
    self._update_xmax(x)

def append_test_curves_data(self, x, accuracy, loss):
    self.x2.append(x)
    self.y2.append(accuracy)
    self.z2.append(loss)
    self._update_xmax(x)
    self._update_y2max(accuracy)

def get_max_test_accuracy(self):
    return self.y2max

def append_data_histograms(self, x, datavect1, datavect2, title1=None, title2=None):
    self.x3.append(x)
    datavect1.sort()
    self.w3 = np.concatenate((self.w3, np.expand_dims(probability_distribution(datavect1), 0)))
    datavect2.sort()
    self.b3 = np.concatenate((self.b3, np.expand_dims(probability_distribution(datavect2), 0)))
probability_distribution(datavect2, 0))

    self._update_xmax(x)

def update_image1(self, im):
    self.im1 = im

def update_image2(self, im):
    self.im2 = im

def is_paused(self):
    return self._animpause

def animate(self, compute_step, iterations, train_data_update_freq=20,
            test_data_update_freq=100, one_test_at_start=True,
            more_tests_at_start=False, save_movie=False):

    def animate_step(i):
        if (i == iterations // train_data_update_freq): # last iteration
            compute_step(iterations, True, True)
        else:
            for k in range(train_data_update_freq):
                n = i * train_data_update_freq + k
                request_data_update = (n % train_data_update_freq == 0)
                request_test_data_update = (n % test_data_update_freq ==
                                             0) and (n > 0 or one_test_at_start)
                if more_tests_at_start and n < test_data_update_freq:
                    request_test_data_update = request_data_update
                compute_step(n, request_test_data_update,
                              request_data_update)
                # makes the UI a little more responsive
                plt.pause(0.001)
        if not self.is_paused():
            return self._mpl_update_func()

    self._animation = animation.FuncAnimation(self._mpl_figure,
                                             animate_step, int(iterations //
                                             train_data_update_freq + 1),
                                             init_func=self._mlp_init_func,
                                             interval=16, repeat=False, blit=False)
if save_movie:
    mywriter = animation.FFMpegWriter(fps=24, codec='libx264',
                                  extra_args=['-pix_fmt', 'yuv420p', '-profile:v', 'high', '-tune', 'animation', '-crf', '18'])
    self._animation.save("./tensorflowvisu_video.mp4", writer=mywriter)
else:
    plt.show(block=True)

Figure 40: MNIST Tutorial 1: Output Visualization

Tutorial 2: TensorBoard

# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
import numpy as np
import tensorflow as tf
from tensorflow.python.estimator import run_config as run_config_lib
from tensorflow.python.platform import tf_logging as logging

import math
from mlengine.digits import test_digits
print("Tensorflow version "+tf.__version__)
logging.set_verbosity(logging.INFO)

# This sample shows how to write Tensorflow models using the high-level layers API
# in Tensorflow. Using high-level APIs, you do not have to define placeholders and
# variables yourself. Also, you will not need to write your own training loop by
# using the Estimator interface instead.

# Download images and labels into mnist.test (10K images+labels) and mnist.
# train (60K images+labels)
mnist = mnist_data.read_data_sets("data", one_hot=False, reshape=True,
    validation_size=0)

# In memory training data for this simple case.
# When data is too large to fit in memory, use Tensorflow queues.
def train_data_input_fn():
    return tf.train.shuffle_batch([tf.constant(mnist.train.images), tf.
        constant(mnist.train.labels)],
        batch_size=100, capacity=1100,
        min_after_dequeue=1000, enqueue_many=True)

# Eval data is an in–memory constant here.
def eval_data_input_fn():
    return tf.constant(mnist.test.images), tf.constant(mnist.test.labels)

def predict_input_fn():
    return tf.constant(test_digits)

# Test data for a predictions run

# Model loss (not needed in INFER mode)
def conv_model_loss(Ylogits, Y_, mode):
    return tf.reduce_mean(tf.losses.softmax_cross_entropy(tf.one_hot(Y_, 10),
        Ylogits)) * 100

    if mode == tf.estimator.ModeKeys.TRAIN or mode == tf.estimator.
        ModeKeys.EVAL else None

# Model optimiser (only needed in TRAIN mode)
def conv_model_train_op(loss, mode):
    # Compatibility warning: optimize_loss is still in contrib. This will
    # change in Tensorflow 1.2
    return tf.contrib.layers.optimize_loss(loss, tf.train.get_global_step(),
        learning_rate=0.003, optimizer="Adam",
        # to remove learning rate decay, comment the next line
        learning_rate_decay_fn=lambda lr, step: 0.0001 + tf.train.
            exponential_decay(lr, step, -2000, math.e)
    ) if mode == tf.estimator.ModeKeys.TRAIN else None

# Model evaluation metric (not needed in INFER mode)
def conv_model_eval_metrics(classes, Y_, mode):
    # You can name the fields of your metrics dictionary as you like.
    return {'accuracy': tf.metrics.accuracy(classes, Y_)}

    if mode == tf.estimator.ModeKeys.TRAIN or mode == tf.estimator.
        ModeKeys.EVAL else None

# Model

def conv_model(features, labels, mode):
X = features
Y_ = labels
XX = tf.reshape(X, [-1, 28, 28, 1])

biasInit = tf.constant_initializer(0.1, dtype=tf.float32)

Y1 = tf.layers.conv2d(XX, filters=6, kernel_size=[6, 6], padding="same",
                        activation=tf.nn.relu, bias_initializer=biasInit)
Y2 = tf.layers.conv2d(Y1, filters=12, kernel_size=[5, 5], padding="same",
                        strides=2, activation=tf.nn.relu, bias_initializer=biasInit)
Y3 = tf.layers.conv2d(Y2, filters=24, kernel_size=[4, 4], padding="same",
                        strides=2, activation=tf.nn.relu, bias_initializer=biasInit)

Y4 = tf.reshape(Y3, [-1, 24*7*7])
Y5 = tf.layers.dense(Y4, 200, activation=tf.nn.relu, bias_initializer=biasInit)

Y5d = tf.layers.dropout(Y5, rate=0.25, training=mode==tf.estimator.ModeKeys.TRAIN)
Ylogits = tf.layers.dense(Y5d, 10)
predict = tf.nn.softmax(Ylogits)
classes = tf.cast(tf.argmax(predict, 1), tf.uint8)

loss = conv_model_loss(Ylogits, Y_, mode)
train_op = conv_model_train_op(loss, mode)
eval_metrics = conv_model_eval_metrics(classes, Y_, mode)

return tf.estimator.EstimatorSpec(
    mode=mode,
    predictions={"predictions": predict, "classes": classes},
    # name these fields as you like
    loss=loss,
    train_op=train_op,
    eval_metric_ops=eval_metrics
)

# Configuration to save a checkpoint every 1000 steps.
# Compatibility WARNING:
# This config syntax will change in Tensorflow 1.2 to be more concise.
# Also there is currently a bug in the config the Estimator class prints when
initialised:
# it appears as empty {} but it actually works and overriden values are
taken into account.

```python
class CustomRunConfig(run_config_lib.RunConfig):
    @property
    def save_checkpoints_secs(self):
        return None

    @property
    def save_checkpoints_steps(self):
        return 1000

    @property
    def tf_random_seed(self):
        return 0

estimator = tf.estimator.Estimator(model_fn=conv_model, model_dir="checkpoints",
                                   config=CustomRunConfig())

# Trains for 10000 additional steps saving checkpoints on a regular basis.
# The next
# training will resume from the checkpoint unless you delete the "checkpoints"
# folder.

estimator.train(input_fn=train_data_input_fn, steps=10000)
estimator.evaluate(input_fn=eval_data_input_fn, steps=1)
digits = estimator.predict(input_fn=predict_input_fn)

for i, digit in enumerate(digits):
    print(str(digit['classes']), str(digit['predictions']))
    # Compatibility Warning: the break is necessary for the time being
    # because predict will keep asking for
    # data from predict_input_fn indefinitely. This behaviour will probably
    # change in the future.
    if i >= 4: break
```
Figure 41: MNIST Tutorial 2: 3D Space Projected Output Visualization

Figure 42: MNIST Tutorial 2: TensorBoard Output Results