

**A Novel Vertical Ground Reaction Force Based Detection System for
Parkinson's Disease**

Sidharth Srivastava

1. Abstract

Parkinson's Disease (PD) is a debilitating neurological disease that primarily affects older patients. PD is characterized by dementia, tremors, and gait issues. PD is the fastest growing neurological disease in the world, with over 60,000 new cases each year in the United States. In addition, PD has increased in prevalence across the developing world, where access to trained medical professionals and equipment is limited. Current PD diagnosis techniques involve a wide battery of tests conducted in medical facilities. While early diagnosis in the developed world is common, the developing world lacks precise methods for PD diagnosis. In this study, a novel system for PD detection is proposed using the collection and classification of vertical ground reaction force (VGRF) data collected during gait. VGRF signals can be used to characterize gait parameters in patients. Two Arduino microcontroller circuits were created in this study to obtain force data from each foot at four different locations. A desktop application was created to collect and display the data. A deep neural network using the LSTM framework was programmed and trained on an existing dataset to classify these signals between healthy and affected patients. The model yielded a training and test accuracy of 91% and 88% respectively. This study demonstrates the promise of using gait parameters as a diagnostic tool for PD. It provides a basis for future systems that can be used in the developing world to quickly find this devastating and rapidly growing disease.

2. Engineering Design Brief

Designer	Sidharth Srivastava
Client	N/A
Problem	Accurate, convenient methods for Parkinson's disease diagnosis and monitoring are currently nonexistent. Most diagnoses are delivered by highly trained medical professionals, who may not be available in developing countries. Thus, quick, reliable methods of diagnosis and monitoring must be developed.
Design Statement	Design, build, and test a wearable sock biosensor and desktop app to measure vertical ground reaction force during gait and detect Parkinson's disease at different onsets and severity using machine learning.
Constraints	Must be unobtrusive and easy to wear Must be reusable
Deliverables	Desktop app Prototype of sock wearable Fritzing sketches of circuits Engineering journal Summary presentation Final paper

Materials

Material	Provider	Quantity
Tekscan FSR	Tekscan	8
Arduino Uno	Researcher	1
Arduino Nano	Researcher	1
MicroSD Card Reader	Adafruit	1
Push button	Researcher	1
Jumper Wires	Researcher	25
nRF21 Wireless Transceiver	Amazon.com	2
Sock	Researcher	2
Various ohm resistors	Researcher	10
Tape	Researcher	1
Breadboard	Researcher	2
9V Battery	Researcher	2

Equipment

Material	Provider	Quantity
Soldering Iron	Researcher	1
Laptop w/Arduino IDE, Python	Researcher	1
3D Printer	Researcher	1

Methodology

1. First Electronic Design Iteration
 - a. Wire 4 sensors to each Arduino using voltage divider circuits and breadboard
 - b. Write calibration program for FSRs
 - c. Conduct quadratic regression on Force vs FSR output
 - d. Connect RF Modules and test pairing/transmission ability
2. Machine Learning Algorithm Development
 - a. Download dataset from <https://physionet.org/content/gaitpdb/1.0.0/>
 - b. Read input files using Python and create Numpy arrays for each file with labels
 - c. Normalize data
 - d. Split data into 60/40 train-test split using Scikit-learn
 - e. Write LSTM neural network architecture
 - f. Train model on prepared data
 - g. Modify hyperparameters as necessary to increase accuracy
 - h. Collect train/test accuracy data, training/test loss
3. Second Electronic Design Iteration
 - a. Add push button to breadboard
 - b. Connect MicroSD Card Reader to breadboard and Arduinos
 - c. Clean up wiring
4. Final Assembly
 - a. Glue 4 sensors on bottom of socks using the locations given by the sock diagram
 - b. 3D print Arduino backing
 - c. Tape each Arduino and breadboard to the backing and strap around foot with velcro
 - d. Connect sensors to attached Arduino/breadboard
 - e. Push button to start data collection and collect data for 40 seconds
5. App development + Final Testing
 - a. Write all Tkinter components for app
 - i. Button for opening file
 - ii. Graph for displaying data
 - b. Insert SD Card in computer
 - c. Load data collected during gait
 - d. Display

3. Wearable Design

Two sock wearables were constructed in this study to collect force data. Each sock had 4 force-sensitive resistors (FSRs) taped to its bottom. The FSRs were connected to an Arduino through a voltage divider circuit and powered through the Arduino's 3.3V power line. Because FSR output was not equal to the force applied, calibration was performed on the sensors. FSRs were placed under 13 different force loads and the sensor output graphed against the load. A quadratic regression was performed to generate an equation of best fit and the equation was used by the Arduino to calculate force values from raw force sensor data. Each Arduino was also connected to a nRF24l01 wireless transceiver module. The wireless modules were also powered by the Arduino 3.3V power line and data transfer was conducted through the Serial Peripheral Interface (SPI) protocol. Each Arduino and its circuitry was mounted on a 3D printed backing and attached around the foot using Velcro straps.

The wearable interface is very easy to use. Users should simply put on the sock, connect the batteries to the Arduino to power them on, and press the button to initiate data collection. They then walk normally for 40 seconds. The data is written to the SD card, which can be taken out and read by the desktop application.

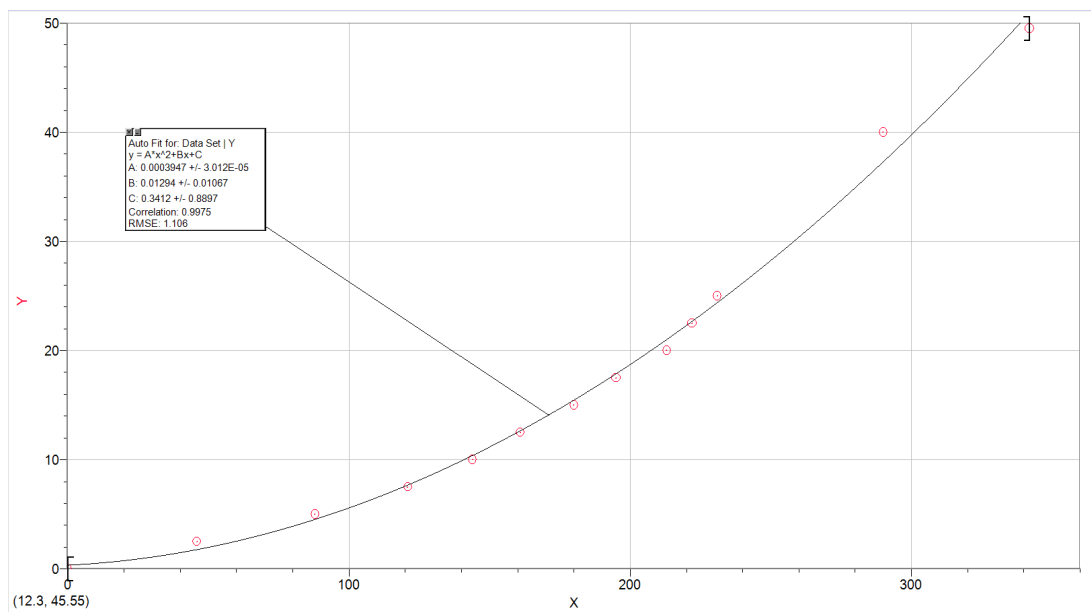


Fig 3.1 Graph of FSR Output vs Force Data with Quadratic Regression Curve

Table 3.1 FSR Output vs. Force Load Applied

FSR Output	Force Load Applied (lb)
0	0.0
46	2.5
88	5.0
121	7.5
144	10.0
161	12.5
180	15.0
195	17.5
213	20.0
222	22.5
231	25
290	40
342	49.5

R-value for Regression: 0.9975

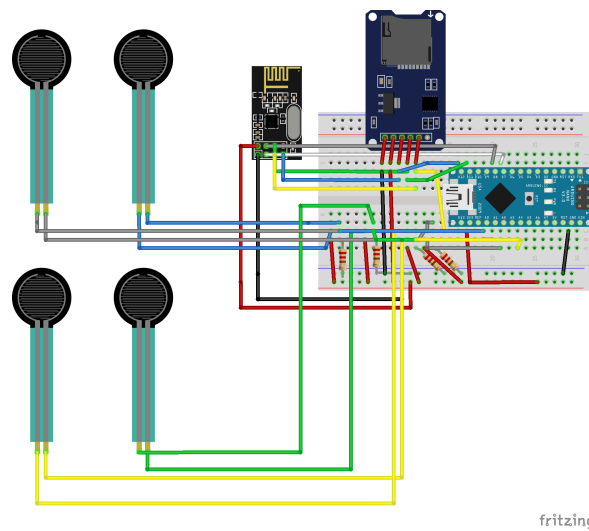


Fig 3.2 Breadboard Schematic of Arduino Nano Circuit



Fig 3.3 Diagram of Force Sensor Positions on Sock

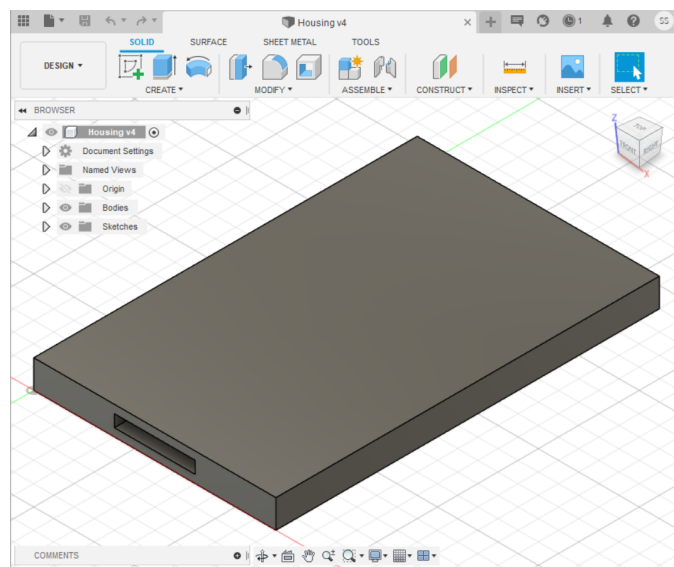


Fig 3.4 CAD of Arduino Backing



Fig 3.5 FSRs on Socks

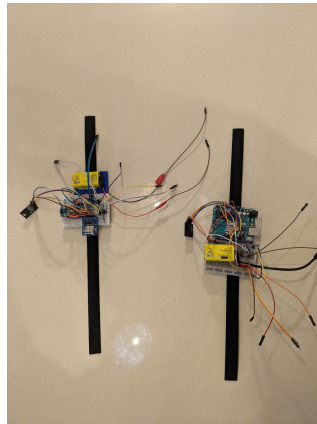
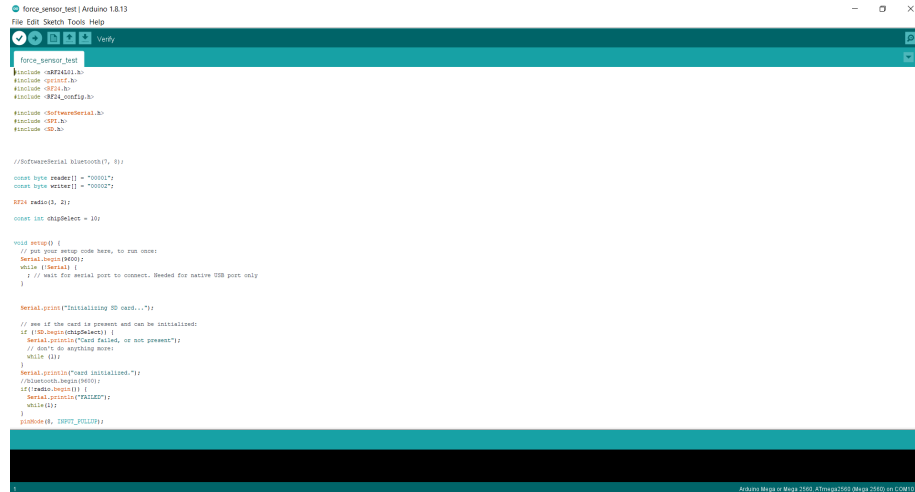


Fig 3.6 Final Assembled Electronic Circuits



Fig 3.7 Final Assembled Sock Wearables



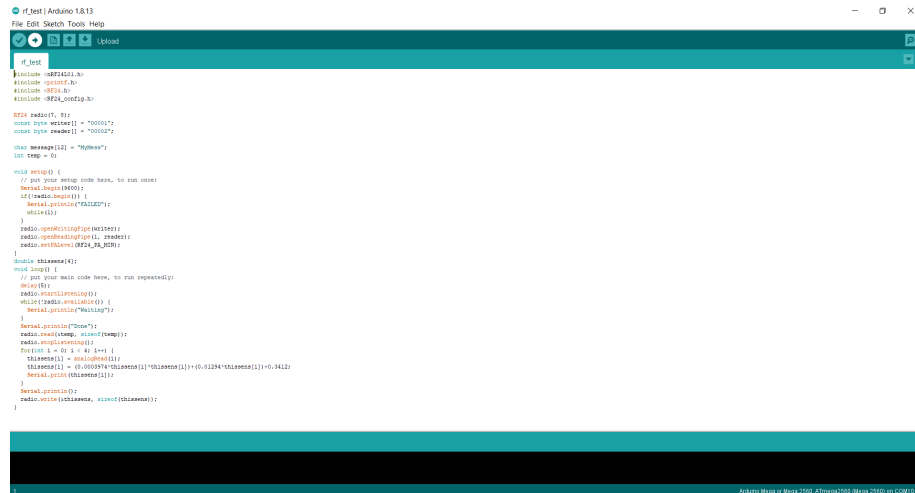
```
force_sensor_test
#include <SPI.h>
#include <SD.h>
#include <SoftwareSerial.h>
#include <SPI.h>
#include <SD.h>

//SoftwareSerial kbluetooth(7, 8);
const byte rxpin[] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F};
const int rxpinlen = 16;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }

  Serial.println("Initializing SD card...");
  // use if the card is present and can be initialized:
  if (!SD.begin(4)) {
    Serial.println("Card failed, or not present");
    // don't do anything more:
    while (1);
  }
  Serial.println("Card initialized.");
  //Initialize kbluetooth:
  if (kbluetooth.begin(9600)) {
    Serial.println("kbluetooth");
  }
  pinMode(4, OUTPUT);
}
```

Fig 3.8 Screenshot of Arduino sketch for sock receiving data/writing to SD card



```
r_test
#include <SPI.h>
#include <SD.h>
#include <SoftwareSerial.h>
#include <SPI.h>
#include <SD.h>

//SoftwareSerial kbluetooth(7, 8);
const byte rxpin[] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F};
const int rxpinlen = 16;

char message[16] = "kbluetooth";
int temp = 0;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }

  Serial.println("Initializing SD card...");
  // use if the card is present and can be initialized:
  if (!SD.begin(4)) {
    Serial.println("Card failed, or not present");
    // don't do anything more:
    while (1);
  }
  Serial.println("Card initialized.");
  //Initialize kbluetooth:
  if (kbluetooth.begin(9600)) {
    Serial.println("kbluetooth");
  }
  pinMode(4, OUTPUT);
}
```

Fig 3.8 Screenshot of Arduino sketch for sock sending data

4. Desktop App Design

The desktop app was written in Python and Tkinter. Tkinter is a built-in GUI framework for Python. The desktop app prompts the user to select a file from which to read force sensor data. It then allows the user to choose which force sensor's data to see, then displays the graph of VGRF over time in the application. The graphs are rendered using Seaborn and Matplotlib.

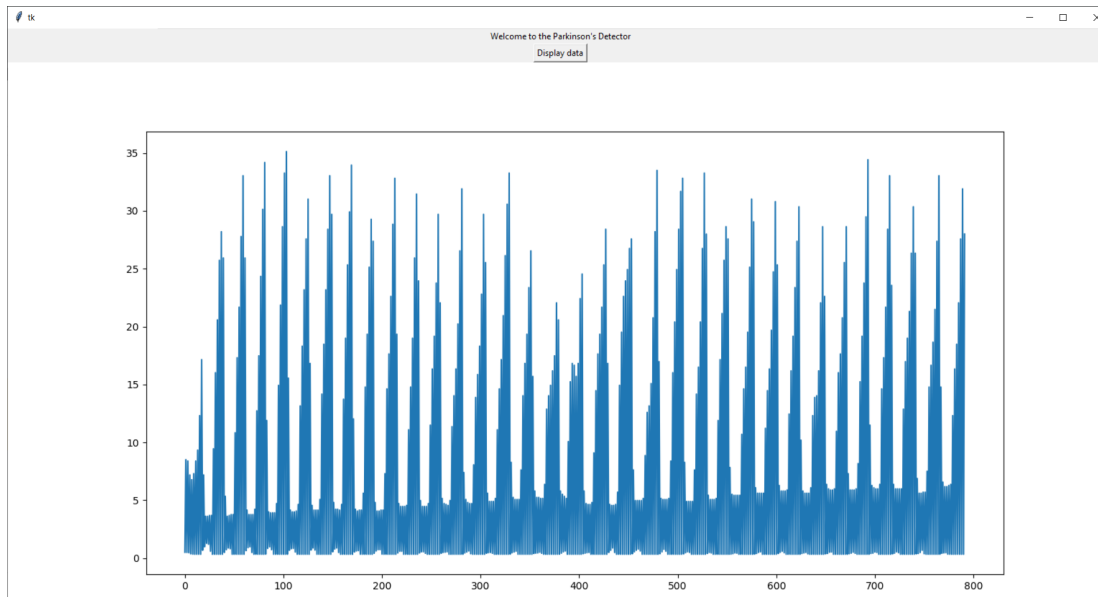


Fig 4.1 Screenshot of Application Showing Force Sensor Data

5. Machine Learning Model

The machine learning model implemented in this study is a type of deep neural network called a Long short-term memory network, or LSTM. LSTMs are modified versions of a common neural network architecture called a Recurrent Neural Network (RNN). RNNs, like most deep neural networks, consist of artificial neurons inspired by biological neurons that take in and output numerical data. Each neuron in an RNN can be described by two equations:

$$h_t = f(x_t, h_{t-1})$$

$$y_t = W_h h_t$$

h_t is the unweighted output of the neuron at time t . The function f is the activation function of the neuron which takes in the previous unweighted output of the neuron, h_{t-1} , and the input data at time t , x_t . The function is nonlinear and has a categorical output (-1 or 1). The final output of the RNN neuron, y_t , equals the unweighted output times a weight W_h . The weight is what is calculated during training. In RNNs, neurons are arranged in layers. Each neuron in a layer will pass its output to a neuron in a subsequent layer. This is called feedforward or forward propagation. When the model trains, it updates its weights based on its current output and the desired output.

RNNs are very useful neural networks because they take into account their previous states. This property allows them to classify sequential and related inputs, like time-series data. However, as the model trains, the influence of the previous inputs is often shrunk and becomes insignificant. Thus, RNNs are not able to learn “long-term” dependencies in the data. The LSTM architecture solves this problem by introducing input gates, forget gates, and output gates into the RNN neuron. Input gates use another nonlinear categorical function to determine whether the neuron should remember the current input and previous output x_t and h_{t-1} . The forget gate uses a similar function to determine whether the cell should consider the current input and previous output in its output. Finally, the output gate uses the same function to determine whether to send the output data to the next neuron. This “forgetting” ability allows LSTMs to consider noncontiguous values in the input, allowing it to perform better on more complex and nuanced signals.

An LSTM was chosen in this study as the model architecture because of its ability to learn on time-series signals with long term dependencies. The various parts of the gait cycle (heel strike, toe-off, etc.) have widely disparate force values but show a temporal dependency.

The LSTM model in this study was programmed in Python using Keras and Scikit-learn. The dataset used for training was the PhysioNet Gait in Parkinson’s Disease dataset that contained VGRF values during gait for healthy and PD patients. The VGRF signals were normalized to have a mean of 0 and a standard deviation of 1 to eliminate differences in force magnitude that

could adversely affect training. Each normalized VGRF signal was labeled as healthy or diseased and the data was split using a 60/40 train-test split. The model was trained for 60 epochs.

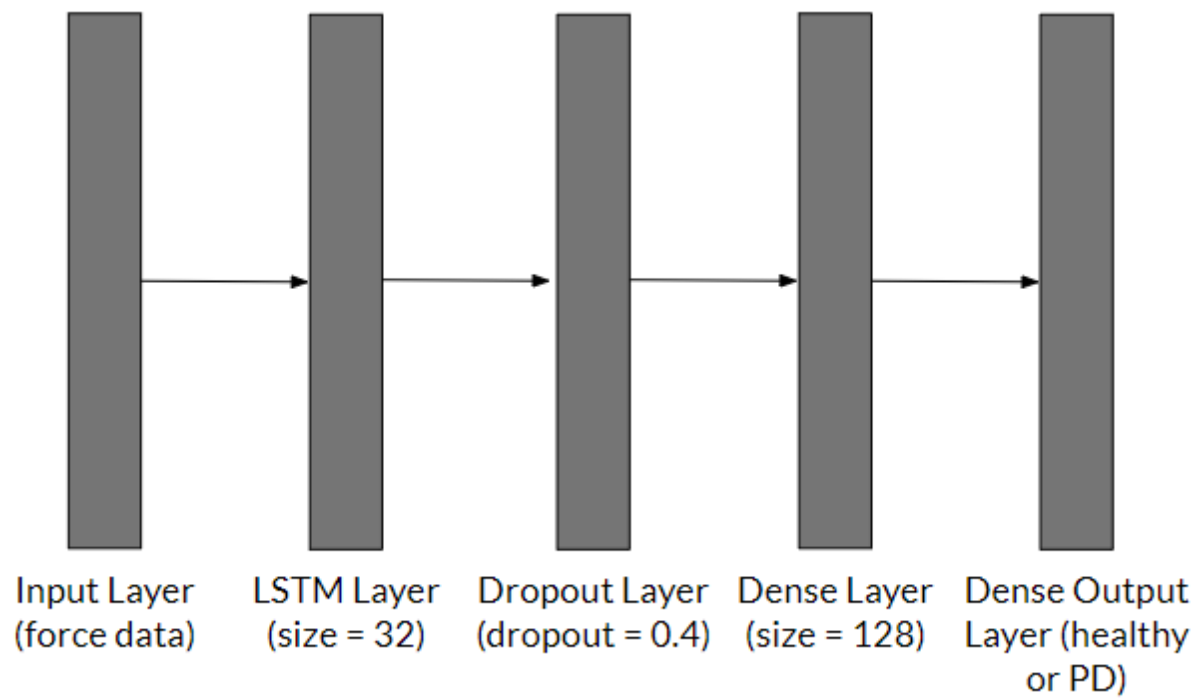


Fig 5.1 Diagram of ML Model Architecture

Table 5.1: Evaluation of Machine Learning Algorithm

Metric	Value
Training Accuracy	91.12%
Validation Accuracy	88.04%
Training Loss	0.2436
Validation Loss	0.3844

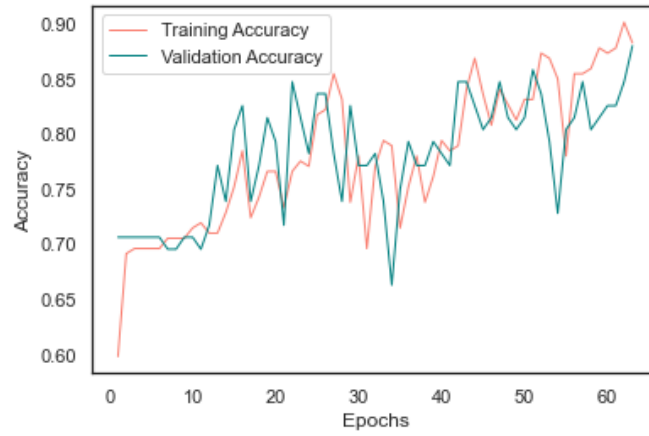


Fig 5.2 Graph of Training and Validation Accuracy over Epochs

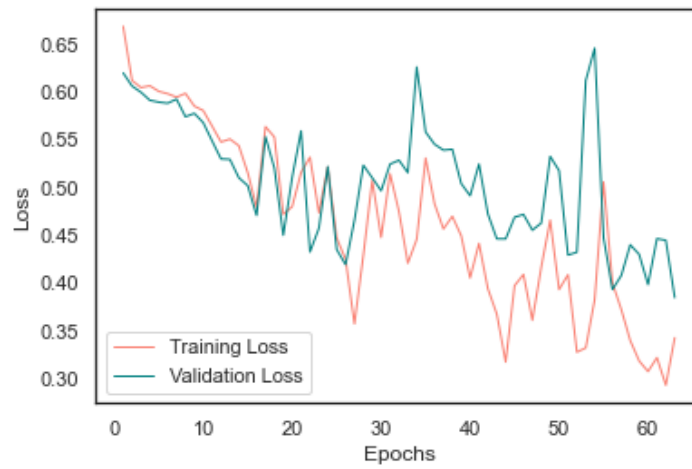


Fig 5.3 Graph of Training and Validation Loss over Epochs

6. Conclusions and Future Studies

Overall, this study was able to meet most of its engineering goals:

- A wearable sock assembly was created to collect VGRF data
- A desktop application was created to load and display the data
- A machine learning model was trained with high accuracy to differentiate between healthy patients and PD patients.
- However, the machine learning model was not adapted to differentiate patients at different Hoehn-Yahr scales of the disease.

This study provides a baseline for many future projects and products. The electronic design can be improved to make the system easier to wear and more compact. The circuits can be transferred to customized PCBs that reduce form factor and wire clutter. The sensors could be sewn into the fabric of the sock, making it much easier to wear and more ergonomic. In addition, many improvements can be made to the ML model. Further hyperparameter tuning can improve classification on the healthy/PD patient dataset. Different ML approaches suited for time-series classification, such as 1D ConvNets and Echo State Networks, can be implemented to improve accuracy and specificity. Finally, the model could be adapted to differentiate the Hoehn-Yahr stages of PD, as well as newer, more personalized rating scales such as the Unified Parkinson's Disease Rating Scale (UPDRS). Overall, the use of VGRF data and machine learning is a very promising diagnostic tool for PD.

References

- [1] Alam, Md Nafiul et al. "Vertical ground reaction force marker for Parkinson's disease." PloS one vol. 12,5 e0175951. 11 May. 2017, doi:10.1371/journal.pone.0175951
- [2] Brownlee, Jason. "Sequence Classification with LSTM Recurrent Neural Networks in Python with Keras." Machine Learning Mastery, 2 Sept. 2020, machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networks-python-keras/.
- [3] Dorsey, E. Ray, et al. "Global, Regional, and National Burden of Parkinson's Disease, 1990–2016: a Systematic Analysis for the Global Burden of Disease Study 2016." The Lancet Neurology, vol. 17, no. 11, 1 Nov. 2018, pp. 939–953., doi:https://doi.org/10.1016/S1474-4422(18)30295-3.
- [4] El Maachi, Imanne, et al. "Deep 1d-Convnet for Accurate Parkinson Disease Detection and Severity Prediction from Gait." Expert Systems with Applications, vol. 143, 2020, p. 113075., doi:10.1016/j.eswa.2019.113075.
- [5] Feffer, Stuart. "The Complete Guide to Machine Learning for Sensors and Signal Data." Reality AI, 4 Mar. 2020, reality.ai/machine-learning-for-sensors-and-signal-data/.
- [6] Goldberger, A., Amaral, L., Glass, L., Hausdorff, J., Ivanov, P. C., Mark, R., ... & Stanley, H. E. (2000). PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals. Circulation [Online]. 101 (23), pp. E215–e220.
- [7] Mittal, Aditi. "Understanding RNN and LSTM." Medium, Towards Data Science, 12 Oct. 2019, towardsdatascience.com/understanding-rnn-and-lstm-f7cdf6dfc14e.
- [8] "Parkinson's Disease: Challenges, Progress, and Promise", NINDS. September 30, 2015. NIH Publication No. 15-5595
- [9] "Rating Scales." Rating Scales | European Parkinson's Disease Association, Mar. 2017, www.epda.eu.com/about-parkinsons/symptoms/rating-scales/.

Code

<https://github.com/sidsrivastava001/ParkinsonsDetector>