Arnav Jhingran
9th Grade
Bellarmine College Prep
1-18-18

A MACHINE-LEARNING BASED APPROACH TO IDENTIFY THE PATTERNS OF MUSCLE MOVEMENTS CAUSED BY MOTOR TICS IN TOURETTE'S SYNDROME

By ARNAV JHINGRAN

Table of Contents

Abstract:

Tourette's syndrome, causes involuntary muscle movement (known as tics) that occur in sudden, brief episodes and often intermittently. Currently, doctors have no quantitative ways of measuring tics and instead rely on subjective anecdotal data and videos recorded by the patient's family. The goal of my project is to build a software system such that doctors can have quantitative data on the frequency and intensity of tics and automatic classification of the movements. This will better the diagnosis and improve the treatment plans by accurately measuring their effectiveness.

I gather the muscle movement data using an iwatch that can capture data for hand movements. I built the code to pre-process the raw data and have it normalized and transposed to be made ready for analysis. I use a machine learning library called TensorFlow to run a neural network for pattern classification of various hand movement data. I collect training data for the hand movement during various normal activities like walking and climbing and then include one of the motor tics (hand touching nose). I validated the model by using the accuracy metrics which converged to around 97%. I want to also explore this technique, in the future, for other tics like neck and shoulder movements by using a small accelerometer chip that can be worn in the neck. I hope my project will help both doctors and patients including myself in better tracking the data and objectively measure improvements.

Engineering Goal:

The goal of my project is to design a system using hardware and software to continuously record the intensity and frequency of muscle movements and automatically classify those that match the type of movements seen in *Tourette's syndrome*.

The neurological disorder, called Tourette's syndrome, causes involuntary muscle movement (known as tics) that occur in sudden, brief episodes and often intermittently. Tourette's syndrome, named after the French neurologist who identified it, affects almost 1 in 100 children in the world with an onset around 9 years and a peak in early teenage years and a reduction in symptoms in late teen years. Simple tics include eye blinking, shoulder shirking, nose touching, head jerking, facial grimacing, eye twitching and other muscle movements. Tics are often worse with stress, excitement or anxiety and better during calm, focused activities. Certain foods, medicines, physical experiences can trigger or worsen tics. Most tics are described anecdotally with the type of tic and an approximation of the frequency of occurrence. Typically, doctors ask parents to record videos during a tic activity. Given that tics happen randomly and also at night it is hard for the patient to accurately track details

on each episode. Thus both the diagnosis and the effectiveness of treatment options is determined by qualitative data.

My goal is to build a system such that doctors can have quantitative data on the frequency and intensity of tics and automatic classification of the movements. This will better the diagnosis and improve the treatment plans by accurately measuring their effectiveness.

Design Criteria:

The muscle movements called motor tics are classified into simple and complex tics. Simple tics involve the involuntary and repeated muscle movement of the hand, shoulder, neck and eye. These vary in intensity and frequency and can change over time.
For this project the design is focused only on identifying simple motor tics.

- The system design should be able to detect tics that have a duration longer than 2 sec. and a frequency of at least 2 in a 10 sec interval.
- The design should be able to track the most common tics such as hand, neck and shoulder tics.
- The false positive and false negative probability should be under 10%.
- The design should be able to distinguish between normal muscle movements in daily activity and tics.

Design Constraints

Tics can happen any time randomly thus any method used to track them has to be always on without intruding in normal daily activity.

The project constraints are:

- The hardware used for tracking muscle movement should be wearable and always on.
- The cost of the hardware should be under $300.
- The hardware contraptions used for tracking muscle movements should be hidden from view (i.e., work under clothes or be concealable) to not make the patient conscious in public.

Design Description

The project design consists of two parts: i) the hardware device(s) to collect and store muscle movement data, and ii) the software to collect the raw data from the device, data cleaning, data transformation, data analysis and classification using open source machine learning toolkit.

Hardware Design Overview

To track any movement an accelerometer and a gyroscope are required. The accelerometer provides linear acceleration data along the 3 axis. The gyroscope provide angular velocity data along the 3 axis, i.e., roll, pitch and yaw.
For tracking the hand movements I used an iwatch (worn on the wrist) that internally contains an accelerometer and gyroscope. The raw data from the iwatch is available using Apple iOS CMMotionManager package and can be set up to store it in iCloud. The processed device-motion data provided by CoreMotion's sensor fusion algorithms gives the device's attitude, rotation rate, calibrated magnetic fields, the direction of gravity, and the acceleration the user is imparting to the device. I used an app called PowerSense to collect the raw data using the iOS CoreMotion package. The data gathering frequency is set to 50Hz so we can get 50 samples per sec. Each sample consists of:

Timestamp (Unix time)	User Acceleration X, Y, Z axis	Gravity X, Y, Z axis	Device attitude/orientation in radians (roll, pitch, yaw)	Rotation rate X,Y,Z axis

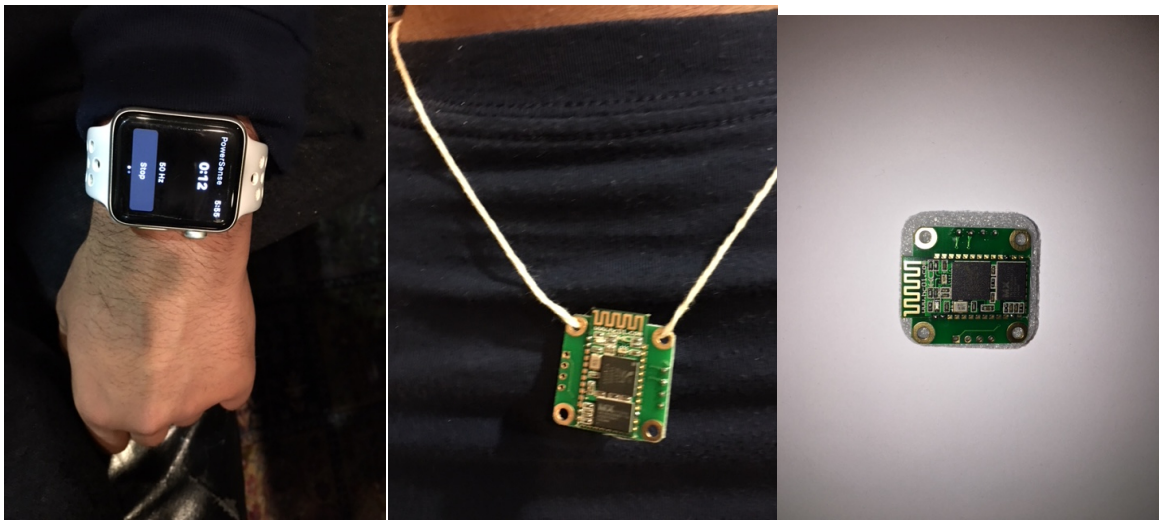

Fig 1: The iwatch with PowerSense to collect the hand tics data and the bluetooth connected gyroscope and accelerometer chip to collect shoulder and neck tic data.

To collect neck and shoulder tics data, I plan to use a bluetooth connected accelerometer and gyroscope 1x1" chip worn with a neck chain under the shirt. For this project I am focusing only on the hand tics and collect the data from the iwatch.

Software Design Overview

After the raw data is collected it is processed in 4 steps: i) Data Collection for training data ii) Data Cleansing and Standardization , iii) Data Transformation and iv) Data Analysis using machine learning.

Training Data Collection
The training data is collected by doing a particular activity for sometime (say 20 to 30 mins) collecting 900 samples where each sample is a 2 sec interval. For normal daily activity I will collect the data for hand movements while walking, running, climbing stairs along with the simulated set of data on the hand movement to touch the nose. The number of samples for the real tic data are small so I repeat the movement voluntarily and compare the data.

Data Cleansing and Standardization
For each data collection training set we drop the first and last 10 seconds to remove any movement tracked in starting and stopping the devices. For each data set we then normalize each data column to eliminate the units and make the data stay within bounds such that the mean is 0 and the standard deviation is 1. For normalization, we compute the mean for each data column and the standard deviation. Then from each value we subtract the mean and divide by the standard deviation, $x_i = (x_i - \mu) / \sigma$
By doing this the different units for the data in the different columns will not affect the results.

Data Transformation (Transposition)
For using the data analysis using the machine learning toolkit called Tensor Flow, I need to transform the data to make it usable by the program. The data collected from the iwatch is at 50Hz which is 100 rows in a 2 sec sample. Each row has 9 columns for the device attitude (orientation), rotation rate and user acceleration, each along the 3 axes (ignoring the gravity values that are there in the raw data). Each column is an attribute that with samples per second that I need to train the data classification machine learning model with. For each column of the original dataset table I create a new table with 100 values of each timestamp transposed as 1 row with 100 columns. The sample in Table 1 shows the subset for 10 samples (rows) of the original raw dataset for a given timestamp and Table 2 shows the transposition for the first column which is device attitude_roll (the columns are also called input attributes). Similarly, Table 3 shows the transposition for the fourth column which is Rotation Rate along x-axis.

Timestamp (unix)	Attitude_roll (radians)	Attitude_pitch (radians)	Attitude_yaw (radians)	Rotation_rate_x (rads/s)	Rotation_rate_y (rads/s)	Rotation_rate_z (rads/s)	User_acc_x (G)	User_acc_y (G)	User_acc_z (G)
1515021317	-0.060	0.476	-1.572	0.077	0.059	-0.011	0.005	0.011	0.005
1515021317	-0.060	0.476	-1.572	0.030	-0.006	-0.039	0.006	-0.006	0.008

1515021317	-0.060	0.482	-1.573	0.199	-0.107	-0.036	0.002	-0.002	0.015
1515021317	-0.060	0.490	-1.574	0.188	-0.028	-0.025	-0.006	0.009	0.008
1515021317	-0.060	0.493	-1.575	0.042	0.045	-0.038	0.004	-0.006	0.014
1515021317	-0.060	0.494	-1.576	0.011	-0.008	0.013	0.009	-0.004	0.015
1515021317	-0.060	0.495	-1.574	0.042	0.036	0.042	0.001	-0.002	-0.001
1515021317	-0.060	0.493	-1.572	-0.131	0.044	0.049	0.005	-0.008	0.016
1515021317	-0.060	0.493	-1.572	-0.182	0.027	0.080	-0.004	0.008	-0.002
1515021317	-0.060	0.490	-1.569	-0.135	0.089	0.069	0.004	0.004	0.026

Table 1: Raw Data for Walking (10 samples in 1 sec)

1515021317	-0.060	-0.060	-0.060	-0.060	-0.060	-0.060	-0.060	-0.060	-0.060	-0.060
1515021318	-0.060	-0.060	-0.060	-0.060	-0.060	-0.060	-0.060	-0.060	-0.060	-0.060
...	...									

Table 2: Transposed data for attitude_roll (radians)

1515021317	0.077	0.030	0.199	0.188	0.042	0.011	0.042	0.131	0.182	0.135
...	...									

Table 3: Transposed data for rotation_rate_x (radians/s)

Data Analysis and Classification with Tensor Flow

TensorFlow (https://www.tensorflow.org/) is a software library in Python, developed by the Google Brain Team within Google's Machine Learning Intelligence research organization, for the purposes of conducting machine learning and deep neural network research. Neural networks can be used for machine-learning based classification and TensorFlow provides some predefined neural networks for classifying exercise movements for fitness which I use as my base to extend upon.
A neural network is a machine learning technology patterned after the human brain. It consists of an input layer, multiple hidden layers and an output layer.

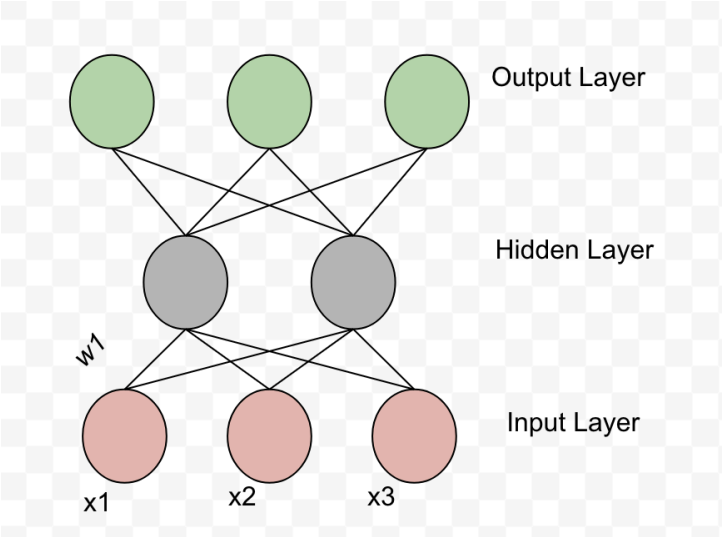

Fig 2a: A simple 1 layer neural network

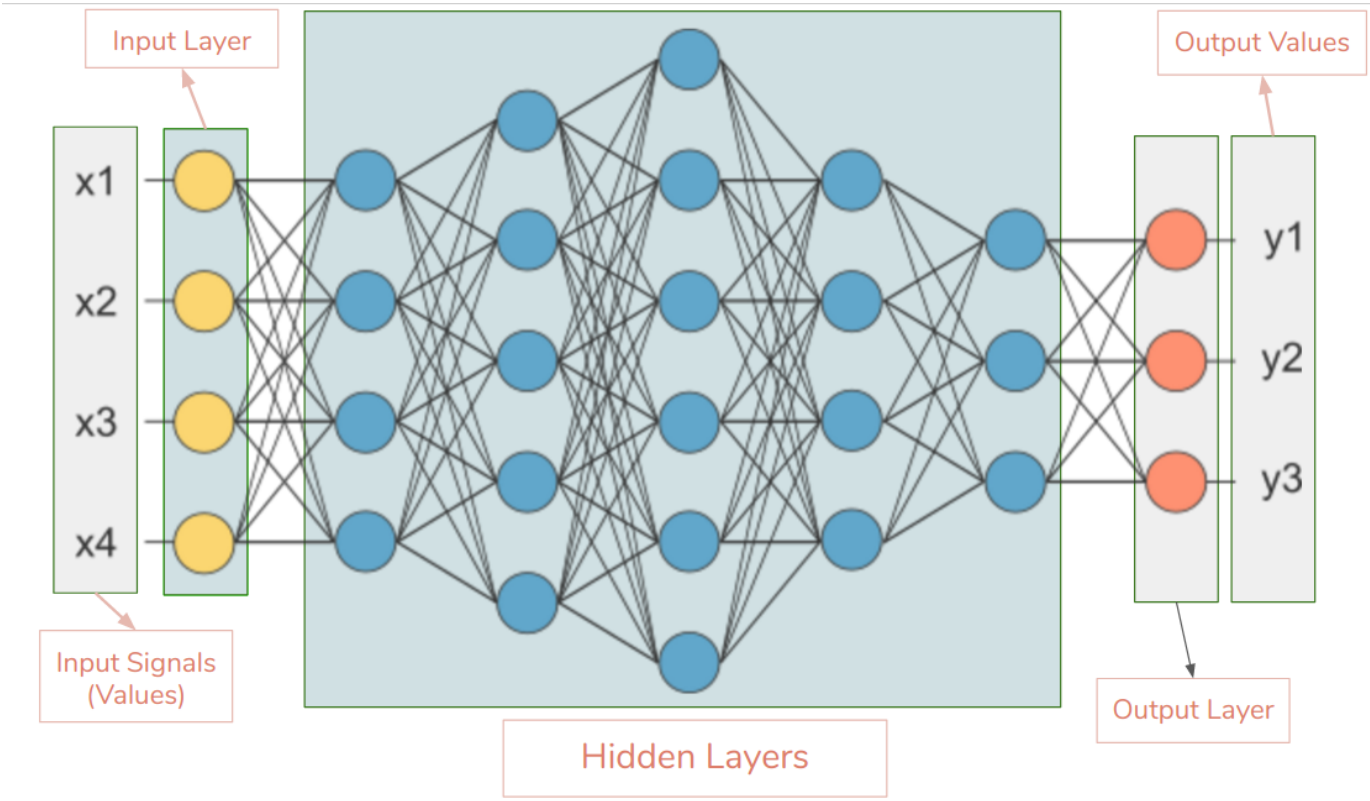

Fig 2b: A multi-layer feed-forward neural network

A neural networks consists of nodes and edges where each edge has a weight. The weights are assigned during the training phases to make sure the output layer nodes fire on the right classification. Each node takes the input values x_1 x_2… along with the edge weights w_1 w_2… and creates an output value y which is a function of the sum of the weighted inputs, i.e., $y=f(\Sigma\, x_i\, w_i)$. The activation function f that is typically used is a sigmoid function or tanh which are similar in shape to a smooth step function. The

simplest neural network in Fig 2 is a feed forward neural network. For the type of classification to detect motion patterns, we need a feedback loop where the output in one training phase is fed as the input into the next training phase creating a type of neural network called a recurrent neural network (RNN). The problem with using simple recurrent neural networks to detect the patterns of motion is that they forget the feedback training from multiple earlier phases and are not able to detect complex patterns. In classification of types of motions simple RNNs either do not converge in the training phase or have poor accuracy. For this we need a type of recurrent neural network that stores some memory (short term and long term) in each node from the previous training phases and has the ability to make a decision when to use the long term memory state and when to not use it by using a function called a gate. Such networks are called long short-term memory neural networks or LSTMs. In an LSTM network along with the input values and the edge weights, each node also has a hidden state *h* which is the working short term memory and a cell state *c* which is the long term memory. The gate function called forget gate is either 1 or 0 which enables the training phase to use the cell state or ignore it.

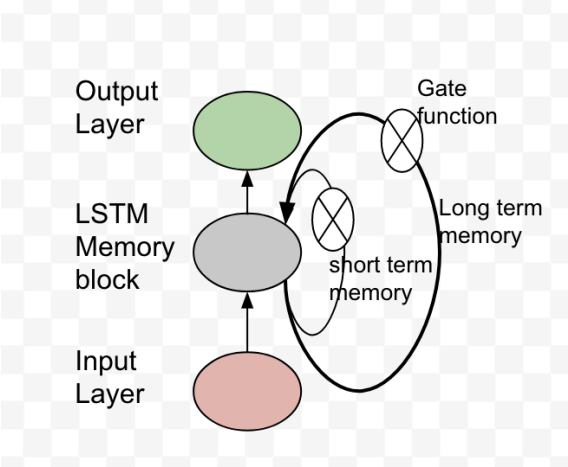

Fig 3: Simple LSTM with single input, block layer and output

The entire flow of the steps I used to classify the hand motion patterns is shown in the following flowcharts:

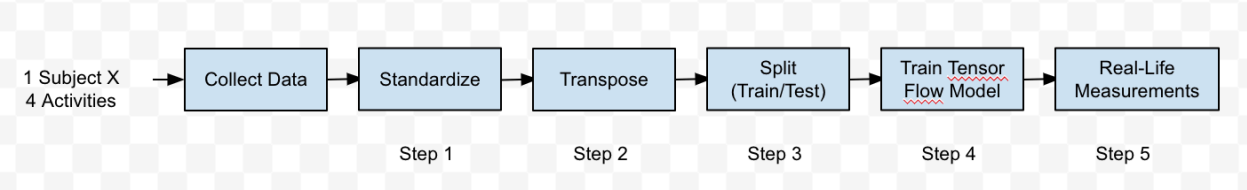

Fig 4: Flow Chart of hand motion pattern classification

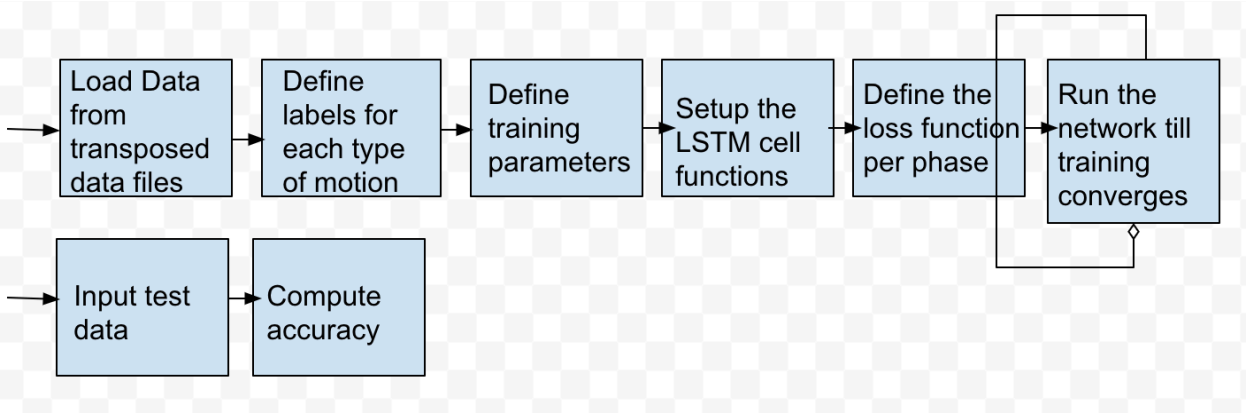

Fig 5: Flow Chart for the "Train TensorFlow Model"

1. Step 1 takes the raw data from PowerSense, labels it by ["subject", "activity"] and **standardizes** the sensor readings so that all data distributions are within bounds (the standardizing for each sensor reduces the mean to 0 and std dev to 1). At the end of Step 1, we get 50 rows per second of activity (given that PowerSense samples at 50Hz), with each row looking like ["subject", "activity", "timestamp", <an array of 9 standardized values>]. The 9 values are ["attitude[roll|pitch|yaw]", "rotation-rate[x|y|z]", "acceleration[x|y|z]"].
2. Step 2 takes 100 of those rows (representing 2 seconds of continuous, same activity) and makes a new **transposed** row, ["subject", "activity", "start-timestamp", <an array of 100 arrays of 9 standardized values>].
3. 70% of these rows are used to train the model, and 30% are used to validate it. So in this step, the **split** is done.
4. Step 4 trains and tests an LSTM (Long Short-Term Memory) using a variation of the TensorFlow model in https://github.com/guillaume-chevalier/LSTM-Human-Activity-Recognition against the data in Step 3. The program was modified with guidance from my dad (who was also my mentor for this project).
5. Step 5 runs that trained model against real-life measurements and classifies the motion that matches that of the hand tics (node touching) and labels the occurrences of tics.

After the classification I create a report of the number of hand tics and their duration each day.

Design Testing

To test the validity of the LSTM neural network model for tic classification we use two measures namely: accuracy and a confusion matrix. The training data is already classified with the type of motion. In that 70% of the samples are used to train the model such that the neural network "converges" and the weights and memory states stabilize

across the phases. The remaining 30% of the samples were used to “test” the LSTM if it classified the data correctly.

Accuracy: The frequency with which a predicted value matches the known value.
Confusion Matrix: This is a specific table layout that allows visualization of the performance of an algorithm. Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class (or vice versa). The name stems from the fact that it makes it easy to see if the system is confusing two classes.

TensorFlow provide a module `tf.metrics.accuracy` that calculates the frequency at which the predictions match the provided labels. The accuracy function creates two local variables, total and count that are used to compute the frequency with which predictions matches labels.

Software Packages used:
I used the following software packages for various stages of data processing.

1. PowerSense App and iOS `CMMotionManager` package to collect the iwatch raw data.
2. `Anaconda 5.0` Python distribution which includes the `Jupyter` notebook and the `Spyder` Python dev environment.
3. In Python included the the libraries: `matplotlib` (to plot the data in the intermediate stages), `numpy` (package for various scientific and statistical functions) and `TensorFlow 1.0.0` (the neural network package).

Model Training and Testing

Step 1: I had 1 subject [“Myself”] and 4 activities [“climbing (treadmill)”, “running”, “walking”, “tics(fingers-to-nose)”]. The tics activity was simulated for training and validated with real data. We had a total of about 15 min of the first three activities and 10 min of the last one (and the tics were simulated by me repeating the movement after watching my video of the movement recorded by my mom -- this was done to give enough samples to the LSTM model). These 15+15+15+10 minutes of activities resulted in approx. 1.5 million sensor readings of the form: [“timestamp”, “sensor”, reading].

Step 2: The Figures below are a 2 second transposed array (100 samples) for [“Me”, “walking”] and I am showing two sensor measurements -- “attitude_roll” and “rotation-rate-x”. I realized that while looking at them gives no intuition to a human being such as me, the neural network learns from these and is able to detect the activity based on these patterns spread across thousands of measurements.

Fig 6: The plots for the device attitude (roll) and rotation rate along x-axis over a 2 sec interval with 100 sample points for the hand motion while walking.

Step 3: During the training phase the LSTM neural networks measure computes a loss function that measures the error (mean square error) in the prediction in that one training step. The average error (loss) across the entire training phase is called the cost function. After training the test data is used to measure the accuracy of the classification. Any accuracy above 90% is considered very good, and our trained LSTM algorithm for hand motion classification got above 97%. In addition, we can also plot the confusion matrix (how many times the algorithm predicted X when in reality it was Y) to get a better visualization. We can see, the confusion matrix in Fig 7.

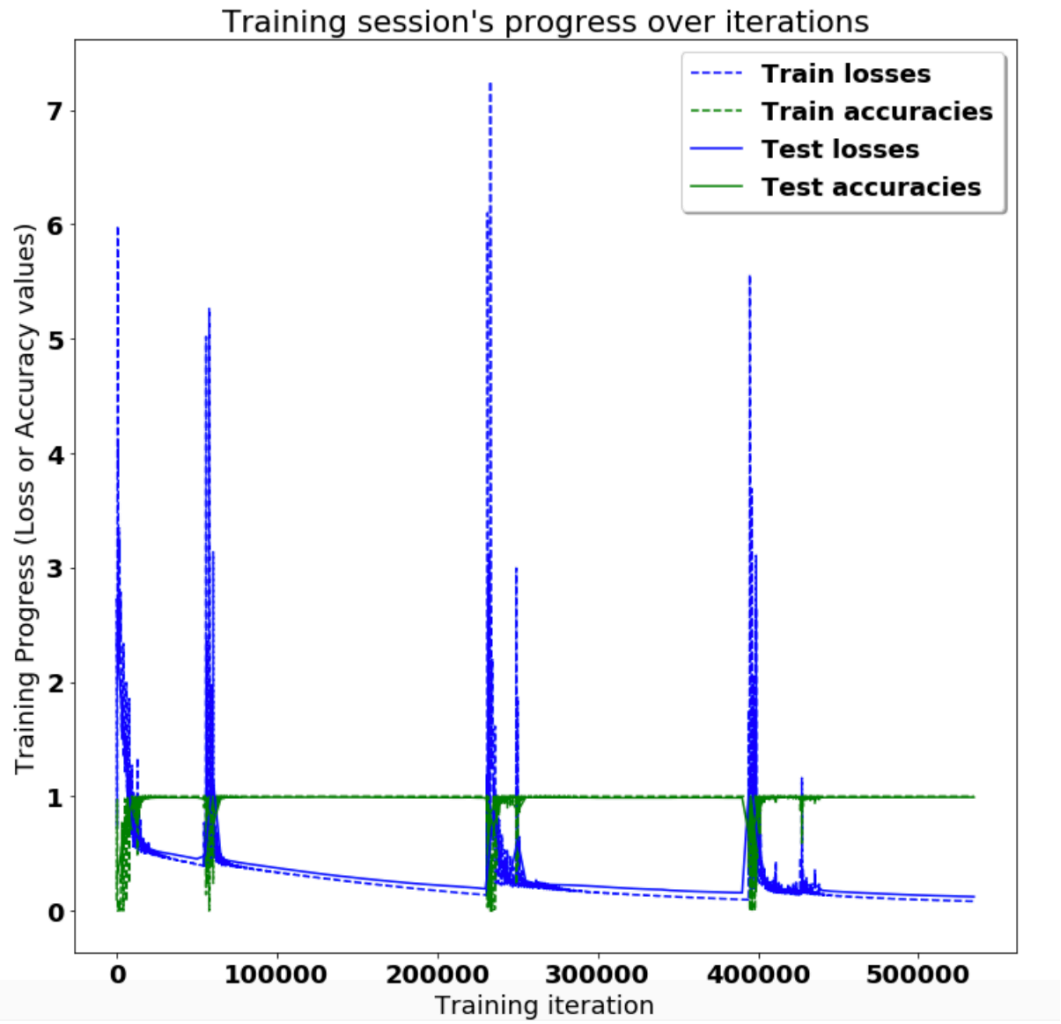

Fig 7: The training session accuracy plots for the LSTM for hand motion classification using Tensor Flow's TensorBoard tool.

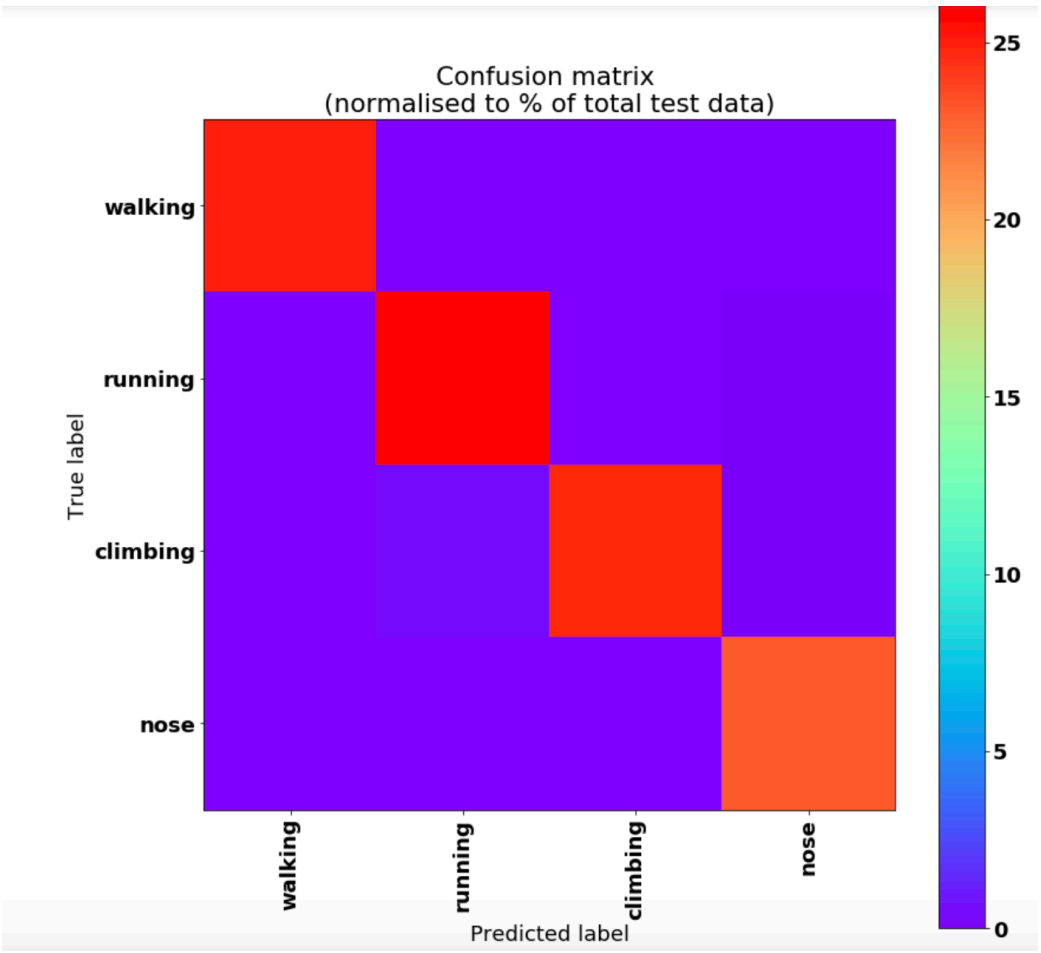

Fig 8: The confusion matrix for the classification of 4 types of hand activities. The x axis is the predicted label by the LSTM and the y-axis is the true label based on the known data classification in the test data.

Acknowledgements: I would like to thank my parents for all the help in doing this project. I would like to thank my mom for her support and helping me with the data collection. I would like to thank my dad for suggesting the idea of using machine learning with neural networks instead of my earlier plan of using a statistical method. He also helped me understand the basics of neural networks and how to use the various Python packages and TensorFlow which I still don't fully understand. I would finally like to thank my teacher Dr. Roy who supported me even when my idea was not well formed and helped me define the problem I was trying to solve.

Bibliography

1. Google, TensorFlow. *https://github.com/tensorflow*
2. G. Chevalier, Human Activity Recognition, *https://github.com/guillaume-chevalier/LSTM-Human-Activity-Recognition*

3. Apple, Apple iOS developer platform, *https://developer.apple.com/documentation/coremotion/cmmotionmanager*
4. Shiyan Xu, PowerSense App, *https://itunes.apple.com/us/app/powersense-motion-sensor-data-logging-tool/*
5. Anaconda Inc, Anaconda 5.0 Distribution, *https://www.anaconda.com/distribution/*
6. S. Guido and A Muller, Introduction to Machine Learning with Python, O'Reilly Media, October 2016.
7. National Institute of Health, Tourettes Syndrome fact sheet, *https://www.ninds.nih.gov/Disorders/Patient-Caregiver-Education/Fact-Sheets/Tourette-Syndrome-Fact-Sheet*