Project Report – Gesture Recognition with a Convolutional Neural Network

Group: Amir Modan, #918234621, amodan1@mail.sfsu.edu

Darshanie Botejue, #917677558, dbotejue1@mail.sfsu.edu

Introduction:

Neural-Machine Interface, or NMI, technology is a system which takes in a biosignal taken from a neural source and then predicts some abstract aspect of the person's state. NMI has progressed medical procedures greatly with helping diagnose people with MRIs as well as helping create more accessible prosthetics for amputees. The development medically has helped a vast amount of people since fMRIs are used to date to diagnose many neural issues such as abnormalities such as strokes and trauma. With Neural-Machine Interface technology predicting gestures and movements, creating prosthetics and tracking development is much easier. Physical rehab patients have also benefited with technology developing to be able to sense muscle function and track progress, and what parts need to be focused on. This is especially helpful with stroke patients, if a stroke patient has lost most if not all mobility in one arm, with the further development of technology to be able to predict the intent of a gesture and thus with the assistance of a prosthetic to fulfil the action and give mobility back to the person. With the current technology, that is mostly possible but there is still room for improvement. NMI technology is ever developing for the future and to help people; the current goal is to refine and develop the technology so the future for rehabilitation and prosthetics is bright for development.

The project on gesture recognition with a convolution neural network will contribute to the ever growing data to develop the best solution for both prosthetics development and medical usage. The goal being to assist physical rehab by being able to predict gestures the user is making based on Electromyography. Ideally the implementation would be alongside a robotic limb that can perform the gestures however the implementation will be focused on data reading and recognition; therefore using neural nets will improve the accuracy of the recognition, rather than machine learning. By focusing on the recognition, we will be developing the usage for progress of physical therapy, not just for stroke patients but for general patients as well. Progress is a strong factor for physical therapy, it is hard to track the progress of physical therapy since at times it is hard to recognize the strength gained from doing long term therapy with small improvements being hard to track without data.

Hardware Design:

System Architecture:

The Neural-Machine Interface we will design will be capable of detecting the user's intent based on muscle activity from within their arm. The Myo Armband will be used as a wearable and wireless alternative to standard lab equipment which will be responsible for measuring this physical activity in the form of Surface Electromyography (EMG) signals. The Myo Armband will then digitize this data and send it to a computer which will act as the central device in our system. A Convolutional Neural Network (CNN) will be responsible for classifying raw EMG signals into one of a set of gestures based on Supervised Learning. While using Neural Networks can be much more costly than standard machine learning algorithms in terms of

computation, they may prove to boost performance when it comes to classifying large and complex data such as the data we intend to receive from the Myo Armband. The computer will also provide a Graphical User Interface so that the final product will be usable. A block design of our system's architecture can be found below in Figure 1.

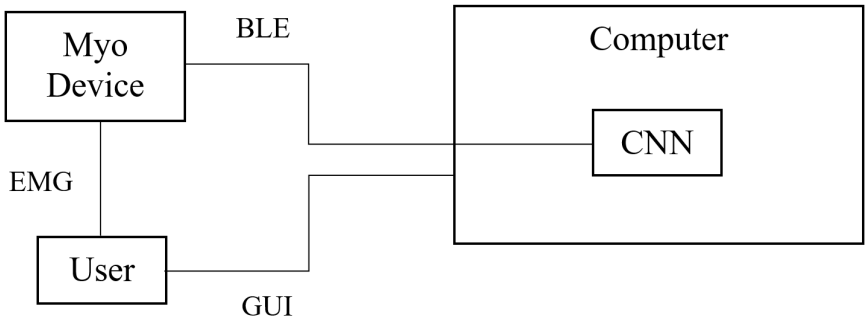

Figure 1 - Neural-Machine Interface System Architecture

Convolutional Neural Networks are particularly extravagant in resource consumption due to all of the matrix multiplications the network has to perform for each neuron in the net. This is especially true when compared to more standard machine learning algorithms such as Linear Discriminant Analysis. Furthermore, systems without a power Graphics Processing Unit (GPU) will exhibit even longer training times since they will have to rely on their Central Processing Unit (CPU). While CPU's can perform complicated instructions (such as branching) with low latency, the simple, yet abundant matrix multiplications present in training a neural network would best be handled by a device capable of performing several simple computations in parallel to achieve high throughput, something that the GPU excels in. In order to accommodate a wide range of host devices, we plan to implement a simple network architecture which would yield relatively low training times even on less capable devices, though this would sacrifice performance.

Critical Aspects:

Hyperparameters pertaining to the Convolutional Neural Network such as the number of layers, number of neurons per layer, number of epochs, etc. will be determined by first testing the Neural Network on a dataset of EMG data. This way, classification performance will be dependent solely on the network itself rather than on any experimental parameters as the data used to train the network on different trials will not change. If we were to optimize based on data from our real-time system, we would not know whether an increase in accuracy would be attributed to the hyperparameters of the network, or simply due to less noise on the current trial. We will optimize the neural net so that maximum accuracy is achieved while maintaining low execution times. An emphasis will be placed on execution time in particular as the user should not have to wait for an unreasonably long time to train the network. This is especially important as the user will have to train a new model each time they wear the Armband or even alter its placement on the arm.

The Myo Armband boasts a seamless setup process due to its elasticity which makes wearing it on one's arm even easier than a watch. This, and the aesthetically pleasing appearance, speaks to its commercial potential over other EMG devices such as the Myoware muscle sensor which requires application of medical electrodes.

The Myo Armband also converts the analog EMG signals into digital values, remaining independent from a microcontroller with ADC capabilities. Because the Myo Armband features a rechargeable battery, the user will not have to worry about providing an external power source. Coupled nicely with this is the Bluetooth Low Energy (BLE) protocol that the armband uses to send data at a reduced power consumption compared to classic Bluetooth.

The Myo Armband will sample 8 channels of data at a frequency of 200Hz. While this is a relatively low sampling rate for an NMI, this should suffice for recognizing simple gestures including Fist, Wave In, Wave Out, and a class for Relaxation, i.e. no gesture. The digitized EMG signals will be sent as a Byte Array consisting of 16 elements, where each channel is sampled twice before sending the array to the client device. Fortunately, the Myo Armband will automatically pair when a client attempts to connect to it, so we won't have to configure server-specific settings such as Baud Rate.

Software Design:

Design Flow:

A summary of our design flow is found below in Figure 2. First, the program will search for BLE devices that are within range of the host device (our Computer). The program then prompts the user to select which device they wish to retrieve EMG data from. If the selection is invalid (input does not match any of the devices), the user will be prompted to re-enter their choice. Then, for each gesture, the user will be prompted by the GUI to hold a gesture for ~1 second (200 samples will be collected). After data for all gestures has been collected, the neural network will be trained. Finally, the program will continuously read data from the Myo device, classify it using the neural network, and display the prediction until the application is closed.

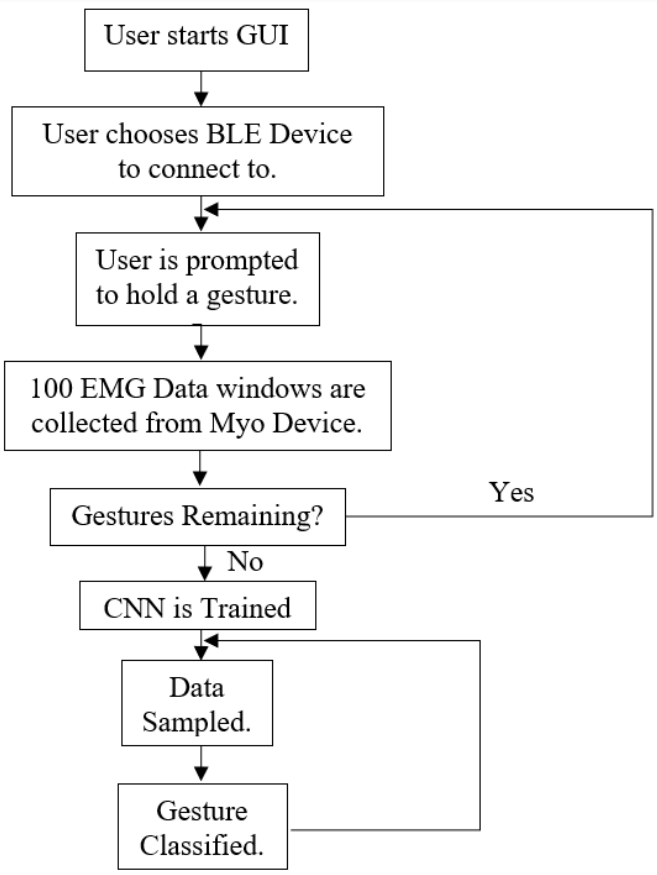

Figure 2 - Neural-Machine Interface Design Flow

Classifier:

We will utilize the widely used "Tensorflow" Library for constructing our Convolutional Neural Network. The input to the Neural Net will be an Array in which each element contains a feature of the data such as the number of zero crossings a window contains and its mean absolute value. Because there are 8 channels of data with 2 features being extracted from each one, the first layer of the Neural Network will consist of 16 neurons. The final layer is directly related to the number of classes we will be differentiating between during recognition. We will have a total of 4 classes, or gestures, that we will be classifying; Therefore, the final layer will consist of 4 Neurons. Additionally, the SoftMax function will be used to assign the weights of this output layer. The SoftMax function will assign weights such that the sum of all weights in the final layer adds to 1, making all classes reflective of their respective probability in the final layer.

A majority of the customization that we can perform will be related to the "Hidden Layers" of the Neural Network. These layers located between the input and output layers will be responsible for determining a gesture from a set of features and will be where the bulk of computation resides. As stated in our Hardware Design section, we aim to simplify the architecture of our Neural Network in order to achieve low training times for our real-time system. Considering this, we will build only one hidden layer.

Of course, this hidden layer will be capable of performing convolutional operations. While utilizing a hidden layer that only performs gradient descent will be significantly simpler, such a

neural network (known as Multi-Layer Perceptron or MLP instead of CNN) will yield extremely poor results such that the system itself would be useless. Tensorflow gives us the option of specifying different dimensions for the convolutional layer. Because our system contains only one independent variable (time), we must use a 1-D Convolutional Layer.

BLE:

Because the BLE protocol will be used, we will need to establish the Generic Attribute Profile (GATT) between the Server (Myo Device) and Client (Computer). We relied on Python's BLEAK (Bluetooth Low Energy platform Agnostic Klient) Library in order to perform all BLE-related operations such as connection, writing to, and reading from the Myo Armband [10]. From this, we will be able to read the characteristics of the Server, i.e. the data that is being transmitted. The Myo Armband has several characteristics from which we can read from including those pertaining to IMU data, battery life, and others. However, we would like to be notified by a characteristic sending raw EMG data only. The Myo Armband does not send this data by default as most users do not require this feature and it is rather expensive to send 8 channels of data at 200Hz. Instead, we will first have to write a command to one of the Myo Armband's characteristics to enable this feature [4].

Once we request EMG data from the Myo Armband, we will continuously receive EMG data until the connection is terminated. 4 separate characteristics will be notifying the computer of the EMG data, with each characteristic sending 8 samples of data for two channels. Several handlers will either store the features of this incoming data in a training set or will make a prediction if the network has already been trained.

GUI:

The GUI was based on a basic GUI in python, which creates a window of a certain size. The GUI has 3 frames which upon clicking the button will switch from the main frame to the selected frame. The GUI will take the input and convert it to a string to process, while the output frame will be where the user will be prompted to not only train the Myo Device but will also display the gesture that is recognized.

In order for the GUI to be responsive throughout the entire application flow, the GUI had to be continuously refreshed when integrated with our system. This had to be done asynchronously with our program as the Front-End should be kept separate from the Back-End for good software design. We found that refreshing the GUI at 20Hz allowed for the GUI to be fairly responsive while not limiting the performance of the application too much. The GUI is shown in Figure 3 where the user is about to select a BLE device.

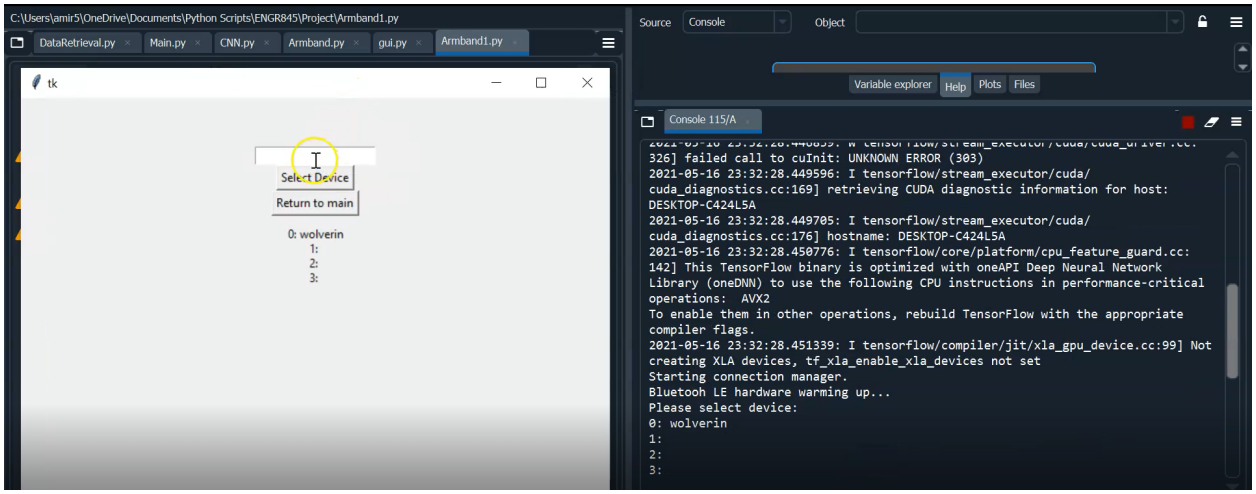

Figure 3 - GUI

Experimentation:

After we train the net, we will provide testing data and will quantitate performance based on Accuracy, Loss, and Speed. Accuracy represents the percentage of tests performed and predicted correctly out of the total number of tests performed. Loss will be based on the Mean Squared Function of the final predictions for each class and will ideally be zero when the target class maintains a weight of 1 where all other classes maintain weights of 0. Speed will be based on execution time of both the training and testing procedures of the neural net and should be kept low. Training and testing data was provided by [5], though zero crossings were impossible to extract as their data had already been rectified. Several hyperparameters in the neural network were fine-tuned in order to optimize performance in all of the aforementioned criteria with a summary of the optimizations shown in Table 1.

Hyperparameter	Optimal Setting
Number of Hidden Layers	1 Layer
Kernel Size	3 Elements
Size of Hidden Layers	8 Neurons
Activation Function	ReLU
Learning Rate	0.01
Number of Epochs	2 (Full Batch) / 15 (Reduced Batch)

Table 1 - CNN Tuned Hyperparameters

The Kernel Size specifies the size of the filter we will use to perform convolution on the input data and will downsample it into features that more closely resemble the classes we are trying to differentiate between. We found that using a filter of odd dimensions produces the best results

for classification accuracy. This is likely due to the fact that odd-sized filters yield symmetry in the output array around the center element. This is contrary to an even-sized filter which is asymmetric, meaning that there will be some distortion when the convolution is being performed. On the other hand, a filter with size 1 will not perform effective classification as no convolution will actually be performed, rendering the network similar to a MLP. Therefore, we have decided to implement a filter that has a length of 3 elements as convolutions on an element will also include its neighbors, allowing for effective downsampling.

We found that the best tradeoff between speed and performance was achieved by setting the number of neurons in the hidden layer to 8. We certainly do not want to have to increase the number of neurons from the previous layer (>16 or <4) as this would be the opposite of the 4 Neuron classification we are trying to achieve. While using anything slightly less than 8 will give greater speed, the loss in accuracy is too great. The opposite goes for layer sizes slightly greater than 8 whose greater training times do not justify the slight increase in accuracy.

In order to create variation in the weights of the Neural Network, several Activation Functions such as Sigmoid and Tanh can be used. We, however, have found the best performance and speed with the Rectified Linear Unit (ReLU) Activation Function. This function will pass any weights with a positive value and will set weights with a negative value to zero (Like a Ramp function). The simplicity of the function allows for Neural Networks utilizing ReLU to achieve faster training times than those using more complex alternatives. As for accuracy, other activation functions struggle with a "Vanishing Gradient" in which the gradient taken at higher weights approaches zero. ReLU does not have this problem as even high weighted neurons will yield a gradient of 1. Therefore, there is no risk of heavily weighted features being discarded during Gradient Descent due to a vanishing gradient.

The Learning Rate determines how quick our Neural Network will adapt to new data. While a greater Learning Rate allows for better generalization of data, the network will also be more vulnerable to noise which our EMG-based system is full of. For this reason, we utilize a rather small Learning Rate of 0.01, so that our Neural Network will be more reliable even in the presence of noise.

The number of Epochs will determine how many times our Neural Network will train a model and adjust its weights according to error. When tuning the number of epochs, one has to be very careful not to train a model for too many epochs as it can lead to "Overfitting" of the data. While an overfitted model will be able to perform remarkably well if tested with its own training data, such a model will not be able to generalize as well. This means that, when provided with testing data that is likely different from the training set due to noise and other factors, the network will perform poorly. Of course, having to repeat training and weight adjustment for multiple epochs will also result in longer execution times. When training off of a large batch size such as an entire dataset, we found only 2 epochs to be optimal for training the Neural Network. However, decreasing the batch size causes the network to have less opportunity to adjust its weights according to error. For this reason, we would have to increase the number of epochs to make up for the smaller batch size. Although this network will be more overfitted and will struggle at making classifications in the presence of noise, such a sacrifice is necessary in order to at least tune the weights sufficiently. When decreasing the batch size of our dataset from 5000 samples/gesture to just 200 samples/gesture (same as our real-time system), we found that increasing the number of epochs to 15 yielded the most consistent results.

Results:

Accuracy and Loss of the Neural Network was evaluated using K-Fold Cross Validation with 8 Folds. We first had to shuffle the entire dataset as, initially, data was sorted by Channel and by Gesture. We then divided the dataset into 8 sections where 1 would be used to test the model while all others will train the model. After training and discarding the model for each fold, we will have 8 Evaluations of Accuracy and Loss. The average of these 8 will be our final Evaluation, shown in Table 2. Performing Evaluation in such a way allows for us to draw more confident conclusions from our results as they will be more reflective of the entire dataset. This is contrary to testing just once off a section of the dataset in which irregularities in testing data can severely skew our results.

Fold	Accuracy	Loss
1	97.50%	0.0900
2	96.86%	0.0868
3	96.30%	0.1246
4	96.20%	0.1217
5	97.08%	0.0755
6	97.22%	0.0937
7	95.58%	0.1638
8	97.22%	0.0759
Total	96.74%	0.1040

Table 2 - CNN Evaluation; Full Batch

Table 2 establishes that our Neural Network can perform quite well when provided with an entire dataset of 40,000 samples. However, collecting 5,000 samples per gestures in a real-time system is simply impractical, particularly due to our Myo Armband sampling at only 200Hz. Therefore, we will have to perform real-time classification with a much smaller batch size. We found collecting 200 samples of data to be reasonable as the user will only have to hold a gesture for 1 second (not including transition periods). Of course, the significant decrease in batch size means that our Neural Network will likely yield lower accuracies as it will not be able to generalize nearly as well. Table 3 shows how the network performs with a reduced batch size of 200 samples/gesture. While testing accuracy is still relatively high, the loss throughout each fold has severely increased. This suggests that the experiments performed by [5] were conducted in a relatively noise-free environment, as our model was still able to make accurate predictions, just with less confidence.

Fold	Accuracy	Loss
1	93.00%	0.2719
2	88.00%	0.4325
3	92.00%	0.3595
4	87.00%	0.5897
5	97.00%	0.0922
6	97.00%	0.0684
7	93.00%	0.5711
8	93.00%	0.2264
Total	92.50%	0.3260

Table 3 - CNN Evaluation; Reduced Batch

One more change has to be made to the dataset in order for it to be completely comparable to our real-time system for hyperparameter optimization. Because our real-time system differentiates between only 4 classes, we should reduce the number of classes in the dataset from 8 to 4. This causes the model to perform extremely well even with a reduced batch size.

Fold	Accuracy	Loss
1	100.00%	0.0005
2	100.00%	0.0016
3	96.00%	0.3733
4	99.00%	0.1466
5	99.00%	0.3321
6	100.00%	0.0077
7	93.00%	0.5711
8	100.00%	0.0077

Total	99.13%	0.1147

Table 4 - CNN Evaluation; Reduced Batch and Reduced Classes

Unfortunately when porting this Neural Network to our real-time system, we had to reduce the number of gestures that were being performed to 4 in order for our system to classify gestures correctly. Using more gestures results in similar gestures being confused at a high rate. For example, the inclusion of "Open Hand" caused our system to frequently yield both false positives and false negatives with the "Open Hand" being mixed up with "Wave In". Because we have already affirmed the validity of our CNN even with a reduced batch size, this problem may have occurred due to an increased presence of noise. This is further supported by the fact that our network was only able to reach a training accuracy of 82.41% after completion. Nevertheless, our system was able to perform quite efficiently with 4 gestures as shown in Figure 4.

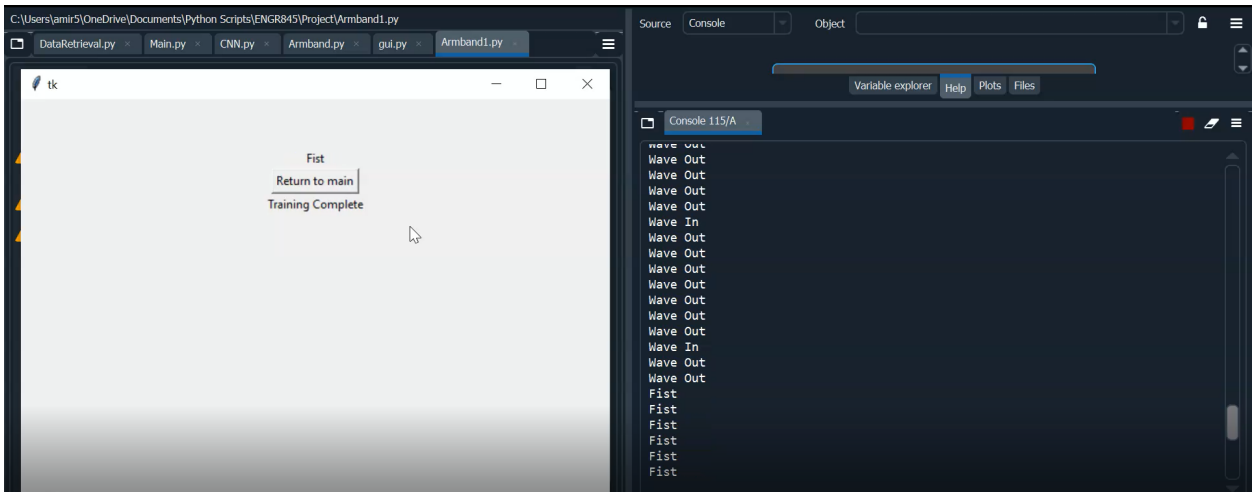

FIgure 4 - User performing Wave Out, then Fist after training CNN

Conclusion:

We set out in this project to improve upon the current technology of gesture recognition for physical therapy, and develop a system to detect the user's intent based on muscle activity using Convolutional Neural Network (CNN). We found that the Convolutional Neural Network can be used for gesture recognition for both dataset and real-time systems. We found in order to obtain optimal performance for our application, we tuned the hyper parameters of the neural network to favor speed over accuracy. Though our classifier was able to perform gesture recognition with great accuracy on pre recorded datasheets, the number of classifications had to be reduced in the real-time to achieve correct classification; this is likely due to the fact that Neural Networks tend to perform at their best when provided a vast amount of data. Despite having only identified 4 gestures including rest, this lack of gestures is mostly due to time deficiency to do the project. If permitted, it would have been possible to recognize more gestures. This is important since we were able to accurately identify the four gestures, we correctly implemented the program and Myo Armband thus adding more gestures with time allotted would be significantly easier.

Discussion:

If we were to do this project another time, we would build upon it to make it more efficient and accurate. While our project was successful there are many ways to improve upon it. The most efficient way to better the performance would be to get a device with a higher processing rate such as a 1kHz sampling rate as discussed by Neurorobot, with the Piezoresistive Array Armband. A higher processing rate would allow for more accuracy and assist with training gestures. With the current Myo Armband, if we were to adapt to the 200Hz sampling rate it would be possible to increase the accuracy by decreasing the sample to be closer to the sampling rate so that it would make the processing required to train and recognize gestures simpler.

Another way to improve the current system would be to incorporate Cloud Computing. Since Cloud Computing has infinite computations available, by having it process the training and gesture recognition it is possible to have the system be as accurate as possible and a faster process. A downside to Cloud Computing would be the cost of communication latency, since there would be a lag between Cloud Computing in comparison to computing on a computer.

Additional ways to improve the system would be to add more sensors to the Myo Armband; this would allow for more gestures to be classified since there would be more information to process which would also help with the accuracy. While these improvements would be able to help our project, our project in itself is great support for the currently developed neural network technology. Therefore accomplishing the goal of demonstrating and improving current technology for gesture recognition for physical therapy.

References:

[1] Front Neurorobot: A Piezoresistive Array Armband With Reduced Number of Sensors for Hand Gesture Recognition. Frontiers in Neurorobotics, 2019, 13:114.
[2] Myo Armband Datasheet
[3] Myoware Muscle Sensor Datasheet
[4] MyoHMI Source Code
[5] Sánchez-Velasco, Leobardo; Arias-Montiel, Manuel; Guzmán-Ramírez, Enrique (2019), "EMG data of the Myo Armband", Mendeley Data, V1, doi: 10.17632/rwbs7645hg.1
[6] Zhang X, Zhou P: High-density myoelectric pattern recognition toward improved stroke rehabilitation. IEEE Transactions on Biomedical Engineering 2012, 59:1649-1657
[7] Young AJ, Smith LH, Rouse EJ, Hargrove LJ: Classification of simultaneous movements using surface EMG pattern recognition. IEEE Transactions on Biomedical Engineering 2013, 60:1250-1258.
[8] Sreenivasa M, Ayusawa K, Nakamura Y: Modeling and identification of a realistic spiking neural network and musculoskeletal model of the human arm, and an application to the stretch reflex. IEEE Transactions on Neural Systems and Rehabilitation Engineering 2016, 24:591-602.
[9] Jiang H: Introduction of Convolutional Neural Networks (CNN). ENGR 492 Lecture Slides
[10] BLEAK Library Source Code; https://github.com/hbldh/bleak
[11] How to Send Data between PC and Arduino using Bluetooth LE; July 11, 2020