Fall 12-15-2019

Finding a Viable Neural Network Architecture for Use with Upper Limb Prosthetics

Maxwell Lavin

Kennesaw State University

College of Computing and Software Engineering

DEPARTMENT OF COMPUTER SCIENCE
Master's Thesis

Thesis: Finding A Viable Neural Network Architecture for Use with Upper Limb Prosthetics

A Thesis Presented to The Faculty of the Computer Science Department

By

Maxwell Lavin

In Partial Fulfillment of the Requirements for the Master's Degree in Computer Science

December 2019

In presenting this thesis as a partial fulfillment of the requirements for an advanced degree from Kennesaw State University, I agree that the university library shall make it available for inspection and circulation in accordance with its regulations governing materials of this type. I agree that permission to copy from, or to publish, this thesis may be granted by the professor under whose direction it was written, or, in his absence, by the dean of the appropriate school when such copying or publication is solely for scholarly purposes and does not involve potential financial gain. It is understood that any copying from or publication of, this thesis which involves potential financial gain will not be allowed without written permission.

Maxwell Lavin

_____

Abstract:

This paper attempts to answer the question of if it's possible to produce a simple, quick, and accurate neural network for the use in upper-limb prosthetics. Through the implementation of convolutional and artificial neural networks and feature extraction on electromyographic data different possible architectures are examined with regards to processing time, complexity, and accuracy. It is found that the most accurate architecture is a multi-entry categorical cross entropy convolutional neural network with 100% accuracy. The issue is that it is also the slowest method requiring 9 minutes to run. The next best method found was a single-entry binary cross entropy convolutional neural network, which was able to reach an accuracy of about 95% in as little as 5 minutes. These time values, while being high for this research, are still a good deal faster than those found in some previous studies. These methods show promise in the popularization of machine learning algorithms in commercial prosthetics, which is something that is still uncommon.

Acknowledgements

The author would like to acknowledge Dr.Hung, for his constant help as well as all the members of my thesis committee, Dr.Son and Dr.Xu. I would like to also give special thanks to the members of the Machine Vision lab for their constant help, as well as to Dr.Wahnoun for his guiding advice, my parents for their constant support, and (because I can) my dogs, Keiko and Kotei, who had to put up with my constant procrastination almost as much as I did.

Table of Contents

List of Figures

List of Figures

CHAPTER 1

INTRODUCTION

The field of hand prosthetics has evolved at a steady rate throughout the decades. What was once hand-carved hands and arms strapped over the amputated limbs has transformed into 3D printed, customized semi-functioning systems that can help amputees perform simple tasks. But there is still much that can be improved upon. There is still a very discernable difference between the performance of a human hand and what can be done with a prosthetic. This gap is something that we are constantly striving to close, and it is my goal to help further this cause.

Currently, research is being done into the melding of computer science into the field of biomechanics in order to create prosthetics that can "think" so as to allow the prosthetics to act more like the natural limb. This melding currently takes the form of utilizing machine learning algorithms to help predict and classify the signals being received from different electrode sensors, with the most common type of sensor being the non-invasive surface electromyography (sEMG) which records the signal produced by muscle activity. These sensors are common in this research as they are non-invasive (something a direct brain electroencephalogram (EEG) probe would be) and localized towards a single function (by which we mean that there is less crosstalk in EMG than in EEG). The machine learning algorithms utilized, on the other hand, vary; depending on the research being done and on who is doing the research. There are commonalities, most research utilized either SVM, or some form of neural network. However, even within these two methods a plethora of differences and variance is possible. With that in mind it is important to note that even with advances in computer science and robotics, more conventional and simplistic prosthetics are still the most popular in the field. I believe that this is due, in part, to the complexity of the program required for these more advanced methods to

work, as well as the time it takes for these methods to be trained and utilized. From this, I arrived at the question of is it possible to produce a simple, accurate, and quick machine learning algorithm that could be utilized in upper limb prosthetics?

To solve this question, it was decided that a combination of feature extraction and neural networks (a machine learning algorithm that has been shown to produce the highest accuracies) would be examined and implemented on EMG data. For feature extraction, it was decided that at first mean absolute value (MAV) and root mean square (RMS) would be examined. If differences could be found, then both would be examined to find which method produced the best results. From these combinations, the most optimal method would be concluded.

To decide what is considered most optimal, we first must explain what is meant by optimal. In terms of speed, we should be looking for a method that is able to both process the received data as well as train and implement the neural network in the shortest possible time. As important as speed is, however, a greater importance must be placed on how accurate the tested method can be. This is to prevent a method that takes no time to run but produces an accuracy no better than random guessing from being considered better than a slightly slower, but more accurate method. The final metric, simplicity, is much harder to define. As the coding for this research was done in parts, I believe that a good way to define this metric is to say that the more modules (I.E programmed sections) required for the method to work, the more complex it would be considered. It should be mentioned that, with the case of the speed metric, there are possible external influences in what affects the run time, including the power of the computer and the possibility of other programs being run. To limit the impact that this could produce, only one computer was utilized to run the methods, limiting the change that the power of the computer

could have on the run time of the methods, as well as only one program was run at any one time. This should give us a generalized idea of how much time it takes for the processes to run.

The layout of this thesis will follow conventional thesis guidelines and be split into eight chapters and the reference section. This introduction is the first of the chapters, and mainly serves to explain the basics of what the thesis will cover. The second chapter is the literature review, which entails a more in-depth discussion of the background information and necessary details to help better understand the importance of the research being done in this thesis. Chapter three entails s quick and simple explanation of the concepts at the core of this research, including EMG, CNN and ANN. The fourth chapter is the methodology chapter, and involves the full discussion of what was done for the research, any and all important information that needs to be known to understand how to repeat the work done in this thesis, as well as an overview of how each of the tested machine learning algorithms processes information. The results of the thesis are discussed in detail in the fourth chapter, and the implication of these results is examined in the fifth chapter. The fifth chapter also contains a discussion of possible sources of error that could have occurred during the research as well as an explanation of methods that can be used to overcome these errors if they were not done so in this research. The sixth chapter serves as a conclusion to the research and reiterates any important information that should be drawn from this thesis. Future research possibilities are discussed in the final chapter of this thesis. Through these chapters we will answer the question of: is it possible to produce a simple, user-friendly, quick, and accurate neural network for use in upper limb prosthetics?

CHAPTER 2

LITERATURE REVIEW

The scope of the literature review was focused on three essential aspects of the research, those aspects being the feature extraction of EMG data, classification methods utilized with EMG, and what the future holds for upper limb prosthetics. Heavy emphasis was placed on the classification aspect of the research as this was the major focus of the work being done. In order to best focus on the classification aspect it was decided to split it into two general categories; those being non-neural-network-based and neural-network-based classification methods. As the main focus of this research revolved around neural networks, the goal of studying non-neural-network methods was both to provide a better, generalized background into the work being done with EMG data as well as explain why neural networks were chosen as the central concept in this paper. The research done on future possibilities, while speculatory and somewhat tangential, provides insight into the motivation behind why the subject of EMG and upper-limb prosthetics was chosen, something I feel is important to understand before moving on with the actual research. The research done on feature extraction was more rudimentary and was essentially done to show what methods are utilized in conventional research, as well as why the methods utilized in this research were chosen.

Feature extraction for EMG data generally falls into one of three domains, time, frequency, or time-frequency, depending on what variable the data is extracted from. For the time domain features are obtained through methods that involve the amplitude of the EMG signal with respect to time, while the frequency domain extracts the features from the power spectrum density of the EMG signal [1]. Time-frequency, meanwhile, is a feature extraction method that utilizes both aspects to conclude its analysis of the EMG data. In this research the methods

utilized, MAV and RMS, fall into the time-domain as they both extract and process the EMG signal data with regards to how the data changes over time. This was done as it is conventionally known that time domain feature extraction is excellent for detecting general muscle activity [2]. Since the classification in this experiment is determined by the EMG read muscle activity located in the forearm and bicep area, choosing time-domain feature extraction methods seemed like the logical choice. The equation for both of these methods is well known, but still provided with MAV being provided in equation 1 and RMS being provided in equation 2.

$$MAV = \frac{1}{N}\sum_{i=1}^{N} |x_i| \quad (1)$$

$$RMS = \sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2} \quad (2)$$

In the two equations, N represents the total size of the sample being examined, with x_i representing the EMG signal value at point *i* within the sample being examined. As can be seen, the two extraction methods are extremely similar, which would explain the similar results reached during testing between the two methods. Frequency-domain feature extraction is less frequently utilized in the classification of EMG data, instead being used to examine and classify either muscle fatigue or neural abnormalities [2]. This is likely due to the nature of extracting based on the amplitude of the signal, which would be impacted by such changes to a much greater degree than a time-domain method would. While time-domain and feature-domain are considered the more conventional methods, the time-frequency domain method of wavelet transformation has found a solid perch in use in EMG signal processing [1].

Wavelet transform is a feature extraction method that is further split into two possible types, those being either continuous wavelet transform (CWT) or discrete wavelet transform (DWT). While both forms are usable with EMG data, DWT is the method most commonly

associated with EMG feature extraction [3][4]. This is due to the nonstationary nature of EMG signals, which is to say that EMG signal frequency varies chaotically with time, an aspect DWT is well versed at handling [3]. This chaotic change is something inherent in the nature of any signal that is produced by a living subject and is something that must be handled either in the feature extraction or in the actual data recording. This will be further detailed in chapter 3.2 when the concepts behind EMG are discussed for clarity. At the core of DWT is an algorithm called the wavelet basis function or the mother wavelet. This function is utilized to continually decompose the data into arrays of feature coefficients [4]. These coefficients are considered either approximal or detailed, depending the frequency of the decomposed signal, with approximal coefficients having a low-frequency and the detailed coefficients having the higher-frequency [3]. While this is a very popular method of feature extraction, it was not utilized in this research for two important reasons. The first, and most important, of the reasons is that the number of variations of wavelet transformation that could be produced is staggering, with each possible wavelet basis function producing a vastly different result and each method allowing countless levels of decomposition. This seemed counter intuitive to the end goal of a simple, quick, and accurate EMG classification method for real world utilization. The second reason is that DWT is best utilized on raw EMG data [3][4]. This is to say that the method truly shines when there is still noise in the EMG data. Unfortunately, or fortunately depending on how you look at it, the data provided by NINApro has already been filtered, negating one of the benefits DWT would offer as a feature extraction method [5]. While it is true the DWT was not utilized for this research, the possibility for use in future work is still possible.

Outside of neural network algorithms, the most popular algorithm utilized in EMG classification is support vector machine (SVM). This is, in large, due to the ability of SVM to

produce classifications from a comparatively small number of training data points [6]. This makes it cases where either the subjects are unable to complete many of the actions required for training (which could be the case if they are in some way impaired or in which a small number of subjects is available). The flexibility of SVM, shown in its ability to handle binary classification, multi-categorical classification, linear data, and non-linear data, allows it to handle the multi-channeled nature of EMG data with little complication [7]. When you consider the relative simplicity of programming and running an SVM, in addition to its flexibility and robustness, it is easy to see what makes it a well-regarded EMG classification method.

That does not mean that it does not have its drawbacks, however, a few of which have discounted its use in this research. The greatest of SVM's issue lies in how it *requires* an external feature extraction method. While this is true for many other methods, it is not true for neural networks, where are able to utilize the un-extracted EMG data and produce their own conclusions without the need for a secondary feature extraction method. It has been found that the total training time for SVM is on the higher end of the spectrum [8]. It has also been shown that SVM produces accuracies lower than artificial neural networks [8]. As these are the two main aspects being utilized to determine what is optimal, SVM was decided against in this research.

As the core element of the research performed in this paper, more time was spent reviewing the work done with neural networks in the field of EMG classification than was spent on researching feature extraction or non-neural network classification methods. From this review of the field, I found that CNN was by far the most popular method for EMG classification research. However, I have also found that most of the upper limb prosthetics in the world today do not utilize machine learning based control methods [9]. This is most likely due to the chaotic

conditions that are present in real world settings, conditions that are removed for research work. Because of these conditions, it can be somewhat difficult to perfectly repeat actions that occur in labs in real world settings. Especially in the case of sensitive instruments, such as EMG electrodes, which will be described in more detail in chapter 3.

CNN is well known in its use in classifying image data, in part due to the two-dimensional nature of the convolutional and pooling layers that make up the heart of CNNs. And while the data utilized in this research is not image data, there have been instances where EMG data has been transformed into spectrographic images and classified with CNNs [10]. From these images, the data from multiple EMG electrodes was able to be examined and classified using CNN, achieving an accuracy in the 90's [10]. While this does show that CNNs can be used with EMG data it does bring the issue of requiring the data to be converted into a spectrographic image, a processing that would increase the complexity of the program and make it ineligible for use in a commercial prosthetic.

Another CNN method commonly used with EMG data is to treat the data as a one-dimensional array, which can then be processed in a simple one-dimensional CNN. This is done by, essentially, turning each EMG electrode into its own separate dataset and plotting the frequency bands for each of them over time [9]. This, however, does require more computations, as the frequency band must be calculated for as long as the EMG electrode is recording. It has, however, been shown produce an accuracy consistently greater than similar SVMs, ranging anywhere from 5-10% better [11]. This method has also been proven to work better with recurrent convolutional neural networks (RCNN), a form of neural networking not touched upon in this research [9]. If this research were to be continued, RCNNs could be tested as well, however it was decided that it was best to start with just CNNs and ANNs.

In regard to ANNs, much like the conversion done for the RCNN, the EMG data must be one-dimensional. This does limit what can be done with ANNs, however it has been found that by combining the feature extraction of DWT, it is possible to get an accuracy of over 70% [12]. While not perfect, this is impressive for a simplified method combining two complex algorithms (DWT and ANN). The major issue with ANN is that, as the dimensionality of the input increases, so too does the complexity of the structure [13]. This means that by increasing the number of electrodes present, there is a noticeable increase in the required complexity of the structure. This makes ANN a less than optimal method for use in commercial prosthetics, where an increase in complexity means an increase in possible errors.

This returns us to the issue of complexity vs accuracy with CNNs. While utilizing raw EMG data is possible in CNNs, it has been shown to produce lower accuracies, and at times requires a great deal of time. In one experiment, it was found that the accuracy ranged from 38% - 66% depending on the data used and required a total training time of one hour and 42 minutes [14]. This reinforces a key component to using CNNs with EMG data, that the end result of the algorithm is heavily depended on a number of factors, including how the data was recorded and the architecture used [15]. This further complicates the use of CNNs in commercial settings as, depending on the architecture used, the data from one subject might not produce the same results with another.

One possible solution is to simplify what is actually being recorded. By this I mean, instead of attempting to read finger, wrist, and hand motion through EMG, focus on just one of those aspects and program an architecture that can classify it. It has been shown that by just focusing on finger motions, and by utilizing hyperparameter optimization, a one-dimensional CNN can produce close to 100% accuracy [16]. The only issue with this method, other than the

fact that it only read finger motion, was that it still required over 30 minutes to run the full program. It can be expected that the processing power of a prosthetic limb will be vastly inferior to that of a conventional computer, just due to the inherent nature of having no external power supply and the need to make the prosthetic wearable and lightweight. This means that any processing time that is produced on a computer can be expected to be much smaller than that produced by prosthetics.

The greatest solution to solving processing time issue is through the use of transfer learning, a technique by which the architecture of the method is trained on a separate system from where it is to be used. This would solve the issue of time and processing power, allowing the training and implementation of the CNN to be done on a more computationally advanced system, and leaving the simple testing calculations to be done on the prosthetic itself. Indeed, some research has found that the overall accuracy of the system would not be drastically impacted, so long as the number of systems utilized for training the CNN was not too great [17]. However, this brings us back to a possible problem with the architecture. If the data being used comes from a separate subject, depending on how the CNN has been designed, the training data might not be applicable to a separate subject. This is something that could be interesting to research in the future, whither it is possible to produce an upper limb prosthetic that is continually training and correcting itself as it classifies data.

This leads us to the motivation behind the research done, the possibility of improved upper limb prosthetics. The greatest change occurring to the field of upper limb prosthetics is the shift away from EMG data to EEG data. This helps bypass the greatest issue with reading EMG signals from amputations, that being the missing muscle and the need to generalize the data from intact subjects for amputated subjects. It has been recently found that by combining near infrared

spectroscopy and EEG a more accurate recording of imagined grip strength could be produced [18]. As this was just done with surface EEG (sEEG), there has been no study on how much of an improvement implanted EEG could produce. This is where the greatest issue in this field of research comes in. With sEEG, you have lower frequencies being recorded, making the impact of external noise that much greater. With implanted EEG, special labs and considerations must be taken into account, making it not very feasible for not-doctoral students to work with, unless they are working with a group already utilizing the implanted EEG. Since this condition was not met, EEG work was not utilized in this research.

The other great change coming to the field of prosthetics is the inclusion of sensorimotor reactions. That is to say producing prosthetics that are able to have some form of sensory feedback. So far this can be split into two fields, non-invasive and direct stimulation. Non-invasive methods normally involve the electrical stimulation of nerves near the amputation site [19]. It has been found that the proper combination of quick succession electrical stimulation was able to stimulate the sensation of touch in different sections of the hand [20]. Extending this to prosthetics would essentially mean that “phantom” sensations could be created by having the prosthetics activate the correct series of electrical current into the specific nerve clusters when the proper section of the prosthetic is stimulated. Of course, the issue here is that the nerves need to be both present and undamaged in order for this method to work. That is not the case for invasive stimulation, which shows the possibility of allowing the programming of artificial sensations in rats [21]. The ability for prosthetics to feel is the logical next step in the field, but before this can come about there is a need to perfect the prediction capabilities of the prosthetics.

CHAPTER 3

CONCEPTS

3.1) *ARTIFICIAL NEURAL NETWORKS*

Before we can get into the principles behind CNN, it is prudent to first discuss generic ANN as many of the fundamental principles are if not exactly the same then extremely similar. ANNs were also used in this research as a comparison basis by which to show the improvements that CNNs have on the same datasets. These improvements will be detailed in the results chapter of the paper, with the explanations behind why they occur being located in the discussion chapter.

A basic ANN is made up of three layers; input, hidden, and output, each containing a predetermined number of neurons connected to one another through weighted connections. The neurons of the input layer are connected to the hidden layer, which is then connected to either another hidden layer or to the output layer. The output of each neuron is determined by an activation function, with the output of the output layering representing the predicted classification data for the problem the ANN is attempting to solve. The most common activation functions utilized today are rectified linear unit (ReLU), which is used within the hidden layers and works by checking if the received input is above a certain threshold, and SoftMax, which is used for the output layer as it produces probability distributions that are used to determine the probability of the received input being in any of the designated classification. Each layer may

also contain a dropout function, which is a method by which a set percentage of random neurons is shut down and produces no output. This is done in order to help prevent overfitting.

3.2) CONVOLUTIONAL NEURAL NETWORKS

The main algorithms utilized for this research were CNNs the architecture and implementation of which will be discussed in the methodology chapter of this paper. In many ways CNNs are similar to ANNs, they utilize the same activation functions, are made of layers, utilize dropout functions, and are even capable of containing layers that are identical to those found in ANN (which are called dense layers). The major difference is that the key component for CNN is a convolutional layer, which is a three-dimensional filter of a pre-set size that passes over the input data and convolves it to match the filter size (hence the name convolutional). CNNs may also contain pooling layers, where are essentially small-dimensional filters that are utilized to reduce the number of neurons present in the data by either finding the maximum or average value of neighbors and outputting that value. The number of neighbors this checks is dependent on the size of the pooling filter as well as the size of the input data, and the pooling filter moves across the input data at a set pace, determined by a variable called stride. Because CNN's are able to handle two-and-higher dimensional data, they are commonly used in image classification.

3.3) *ELECTROMYOGRAPHY*

EMG is, essentially, a process by which the activation of muscles within the human body are quantified and recorded for use in a multitude of applications. There are two forms of EMG that are commonly utilized, surface EMG, which is both the most common form and the form being studied in this research, and invasive EMG, which involves the implantation of the EMG electrode directly on the muscle (or muscle group) being studied. These electrodes are capable of picking up the minute electrical impulses that are produced when a muscle is activated or contracted, with the implanted electrodes being the most accurate and precise of the two.

The greatest issue and hassle of working with EMG data is the two-fold. The first issue, which is impossible to remove, is the fact that work is being done on human subjects. This means that there are an infinitesimal number of variables that could go into why EMG signals are behaving differently from one subject to the next. This also means that the data recorded from one subject will be completely unique, as the unique biochemistry of each subject affects the way in which the EMG signal is recorded. Differences in such simple things sweat composition, fat percentage, skin dryness can cause drastic changes in the received signal. This also contributes the second greatest problem when dealing with EMG, which is noise. As the EMG electrodes are required to be sensitive so as to best pick up the muscle signal, they are also extremely sensitive to other sources of electrical signals as well as changes in environment. This includes other external electrical devices (such as cellphones and computers), movement of the sensor (either from the subject moving or just from improper placement), the slight EM radiation given off by all people, and the activation of other muscles near the recording zone (which is called cross talk). All these issues need to be addressed either before the data is recorded (by turning off

electronic devices or double securing the electrodes to the subject for example) or must be filtered out of the data in pre-processing.

CHAPTER 4

METHODOLOGY

4.1) *"RAW" DATA*

The raw data utilized for this research was obtained from the Non-Invasive Adaptive Hand Prosthesics database (NINApro), which is set of public access databases each containing EMG data representing different experimental conditions [5]. The database used for this research was labeled DB2 and represents 40 healthy, intact (I.E no amputations) subjects that were tasked with three different experiments. The first of these experiments involved the recording of 17 different hand and wrist movements/positions, each repeated six times and each followed by a period of rest (which represents position 0). The second experiment involved the subjects enacting 23 grasping tasks, each involving the use of a real-world object for the subject to grab as well as three requiring the subjects to enact a movement action (twisting off a bottle cap, turning a screwdriver, and chopping with a knife). The final experiment was a force measurement test in which the subjects were fitted with a finger force sensor and were required to apply force to each finger and to a combination of different fingers. Of these, the research done utilized the data from the first experiment.

The seventeen hand and wrist movements were, essentially, split into two basic groups, with the first group representing eight different possible hand and finger positions that were held for a set length of time of five seconds and the second representing nine different wrist movements that were performed for five seconds. Each of these movements was assigned a

numerical value for easier classification. As mentioned previously, the hand and wrist at rest was classified as the 0 value for classification. The eight hand positions were; thumbs up, peace sign, index, middle, and thumb extended, all fingers except for thumb extended, all fingers extended, closed fist, and two forms of index finger extension. These were classified as values 1 through 8, respectively. The examined wrist motions were; a counter-clockwise rotation of the wrist, a clockwise rotation of the wrist, a clockwise twisting of the wrist, a counter-clockwise twisting of the wrist, moving the hand anteriorly from the wrist (which is to say moving the hand so that it is perpendicular to the inner-forearm), moving the hand posteriorly from the wrist (the opposite of

Figure 1: Hand and Wrist Positions

the previous movement), closing the fingers into a fist, a lateral movement of the hand from the wrist (which is to say moving the hand towards the thumb-side of the wrist), and a medial movement of the hand from the wrist (the opposite of the previous movement). These were classified as values 9 through 17, respectively. All of these movements can be seen in figure 1, which contains photos provided by the NINApro database [5].

In order to collect the data from the experiments, 12 different electrodes were used and placed at three major muscle junctions on the subjects. Two electrodes are placed on the triceps and biceps, close to the elbow joint, another two electrodes are placed on the flexor and extensor digitorum muscle (which are located in the forearm, close to the elbow joint area on the anterior and posterior of the forearm, and the final eight electrodes are spaced equally around the forearm

between the elbow joint area and the two digitorum electrodes. The electrodes were manufactured by Delsys, which were used to collect and sample the signal at a rate of 2 kHz. To ensure minimize the noise produced by movement, a combination of adhesive bands and elastic bands were used to try and keep the electrodes from shifting out of place during the movements. A cyberglove and accelerometer (which was located on the wrist of the subjects) were also included in the data acquisition, with the glove recording the kinematic data that occurred during the experiments and the accelerometer recording the change in acceleration. However, the data from these two sources was ignored during this research, as the sole focus of the research was on EMG data.

Before the data was given for public use, it was first processed to ensure that the data was both correctly classified and labeled, and to help remove some of the noise found in the signal. The processes by which both of these are done was proposed by Kuzborskij et al. in the paper "*On the challenge of classifying 52 hand movements from surface electromyography*" [22]. In order to ensure that the data was properly labeled, a combination of matching the EMG data and the classification data through the use of high-resolution timestamps as well as using a likelihood ratio algorithm to relabel the data where error could have occurred [5]. For the filtering, a Hampel filter was used to remove the noise, more specifically EM noise that occurred thanks to the close proximity of the electrodes to other electrical appliances. The Hampel filter works by taking a hyper-parameter window of data and a tuning parameter which the filter uses to determine if the value at the center of the window is an outlier in comparison to the median value [23]. If the central value is found to be an outlier, it is replaced with the median value and the window moves to the next value.

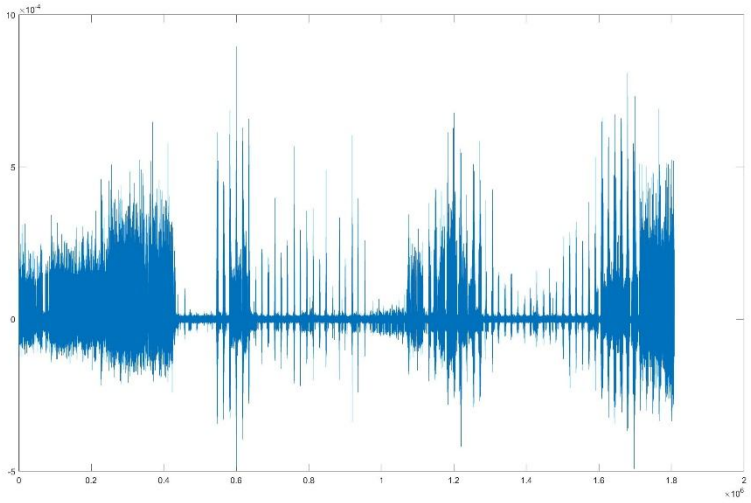

Figure 2: Visual Representation of EMG Data

Once the data was filtered and properly labeled, it was converted into a MATLAB file and placed in the online database. From the MATLAB file, we are able to produce a visualization of the EMG electrode readings, which is provided in figure 2. In this figure, the X-axis represents time (in seconds), while the Y-axis represents the recorded muscle activity picked up the EMG electrode. In the figure we can points of great activity, especially around the $0.6*10^6$ second point, representing major actions, and periods of general inaction, like what can be seen at the $0.4*10^6$ second point.

In order for the NINApro data to be best used in the research, it had to be pre-processed and reformatted for use in Python. This was done by first extracting the datafile with MATLAB and removing the portions that contained the EMG data as well as the refitted classification data. The data was reformatted into something usable in Python through the use of SciPy, which contains the “loadmat” method of reading MATLAB files. This method was called for both sets of data, creating a python-usable data structure. From this, the EMG data took the form of a Mx12 matrix, with M representing the number of samples in the data structure and the 12 representing the number of electrodes, and the classification data took the form of a Mx1 matrix

holding the classification value of the data at time M. An example of what these data structures were then labeled with the correct column titles (this being the electrode number for the EMG data, and just the word “Classification” for the classification data) and concatenated together to create a singular dataframe. The first five entries of the dataframe for subject 1 can be seen in figure 3, which is provided below to help visualize the dataframes. From this, it can also be seen how miniscule the order of the electrode data is, being in the magnitude range of 10^{-5} to 10^{-8}.

	EMG 1	EMG 2	EMG 3	EMG 4	EMG 5	EMG 6	EMG 7	EMG 8	EMG 9	EMG 10	EMG 11	EMG 12	Classification
1	-2.85E-06	3.36E-06	1.60E-06	-1.39E-06	-2.08E-06	2.75E-06	-2.35E-06	1.54E-07	-2.48E-07	1.47E-06	-5.61E-06	4.02E-05	0
2	-4.70E-06	3.36E-06	1.43E-06	-3.07E-06	-3.06E-06	2.25E-06	-1.66E-07	2.50E-06	-3.56E-07	2.48E-06	-3.26E-06	4.96E-05	0
3	-2.68E-06	4.03E-06	9.26E-07	-5.42E-06	-3.70E-06	1.24E-06	2.35E-06	5.52E-06	-2.45E-07	2.48E-06	-7.05E-08	4.71E-05	0
4	2.18E-06	2.85E-06	7.58E-07	-6.60E-06	-3.94E-06	1.58E-06	2.02E-06	5.19E-06	1.09E-06	2.82E-06	1.44E-06	3.18E-05	0
5	4.20E-06	6.69E-07	1.76E-06	-5.42E-06	-5.65E-06	7.36E-07	8.41E-07	8.24E-07	1.15E-06	2.82E-06	9.70E-08	1.40E-05	0

Figure 3: First Five Entries for Subject 1

This was done for the data for four subjects, which represents over 5.4 million datapoint for each of the 12 electrodes. In order to be used for the multi-entry CNN, the length of the dataframe had to be limited to a multiple of the input size, which will be described in more detail in section 4.5.

4.2) *SINGLE-ENTRY NEURAL NETWORKS*

The first series of neural networks produced and tested in this research have been dubbed single-entry neural networks as their architecture was designed to accept a single data entry (representing a singular point in time) as the input value. This means that the single-entry CNNs have an input value array of (1, 12, 1), which represents an array of data that is one-time entry

-2.85E-06	3.36E-06	1.60E-06	-1.39E-06	-2.08E-06	2.75E-06	-2.35E-06	1.54E-07	-2.48E-07	1.47E-06	-5.61E-06	4.02E-05

Figure 4: Example of CNN input

high, with a width of 12 EMG signals, and a depth of one series of data. An example of an accepted input data is shown in figure 4, which is the first entry of subjects 1's electrode data.

The general architecture for the three CNN's is essentially the same, with the only changes being in either what loss function, which is the function that is used to improve the neural network, was utilized or in how the inputted data was pre-processed.

The loss function first utilized was a binary cross entropy algorithm, as it was found that there was a history of using it in CNN with EMG signals [24]. This loss function worked by comparing the output value of the neural network with each possible classification that it could be and utilizing the ground truth to see if the predicted value falls into the examined classification. It is called binary as the loss function determines if the value falls into each of the individual classifications (in which case the statement is true and represents a 1 value) or doesn't (in which case the statement is false and represents a 0 value). The second loss function utilized was a categorical cross entropy algorithm, as it was my opinion that it fit the requirements of the CNN to a better degree than a binary cross entropy loss function. This loss functions examines the probability that is produced by the output layer with the overall probability produced from the ground truth to see if the output value falls into the correct classification. This is called categorical as it examines each possible classification value at the same instance, as opposed to the binary cross entropy function which checks each possible classification value one at a time. The final change made was with the input data, which was either the standard data preprocessed with the method described in section 4.1 or data that had been further processed with a feature extraction method. For the purpose of this research RMS feature extraction was utilized. It should be noted that both RMS and MAV were initially examined, however the difference between the two methods was found to be negligible.

The convolutional layers in the single-entry CNNs all utilize a ReLU activation function which was chosen as it was the method the author had the most experience in as well as being the most conventional method used today. The first convolutional layer contains 40 filters (chosen at random) of size 1x6, which was chosen as it was half the input shape. The first convolutional layer is followed by a max pooling layer of 1x3, which was chosen because it was half the size of the previous filter. The next layer is a convolutional layer with 40 filter of size 1x2, a size that was chosen arbitrarily. This is followed by another convolutional layer with 40 filters of size 1x1, which was chosen as it is the lowest possible value for a filter. Each convolutional layer had a dropout rate of 30% to ensure that there was minimal risk of overfitting the data. Following the final convolutional layer, the CNN has a global max pooling layer which helps convert the data from a convolutional usable two-dimensional data structure to a one-dimensional data structure usable by the dense layers. This produces the first dense layer with 40 neurons, which leads to a second dense layer of 20 neurons, chosen as it is half the value of the previous layers. These two dense layers also have a dropout rate of 20%. The final layer was a dense layer with 18 neurons which represents the expected output of the CNN. For this final layer, the activation function was switched from the ReLU algorithm used in the other layers to a SoftMax algorithm that is conventionally used for the output layer of classification CNNs. This general architecture of the CNN can be seen in figure 5 below. The graphical representations were produced utilizing a public access graphic interface found at URL http://alexlenail.me/NN-SVG/index.html [25].

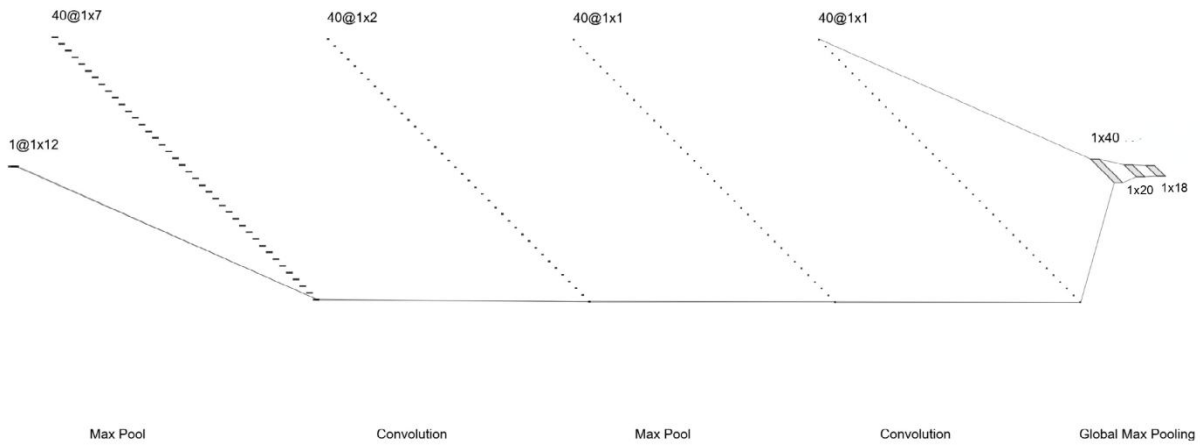

Figure 5: Single-Entry CNN Architecture

To provide a comparison to the CNN, simple ANNs were designed for the same experimental conditions as the three CNNs. These ANNs are made from six layers, including the input and output layer, each using a ReLU activation function with the exception of the output layer which uses a SoftMax function much like the CNNs. The input layer is made of 12 neurons, representing the data from the 12 electrodes. The first hidden layer is made of 300 neurons, a value chosen at random, and has a dropout rate of 20%. The proceeding hidden layer contains 150 neurons and is followed by a hidden layer of 75 neurons both values chosen as they are half the value of the previous layer. The final hidden layer is made from 25 neuron, chosen as it was a third the value of the previous layer. Much like the first hidden layer, these three also had a dropout rate of 20%. The final layer is the output layer, containing 18 neurons representing the 18 possible classifications of the EMG data. The ANN architecture described above can be seen on the next page in figure 6.

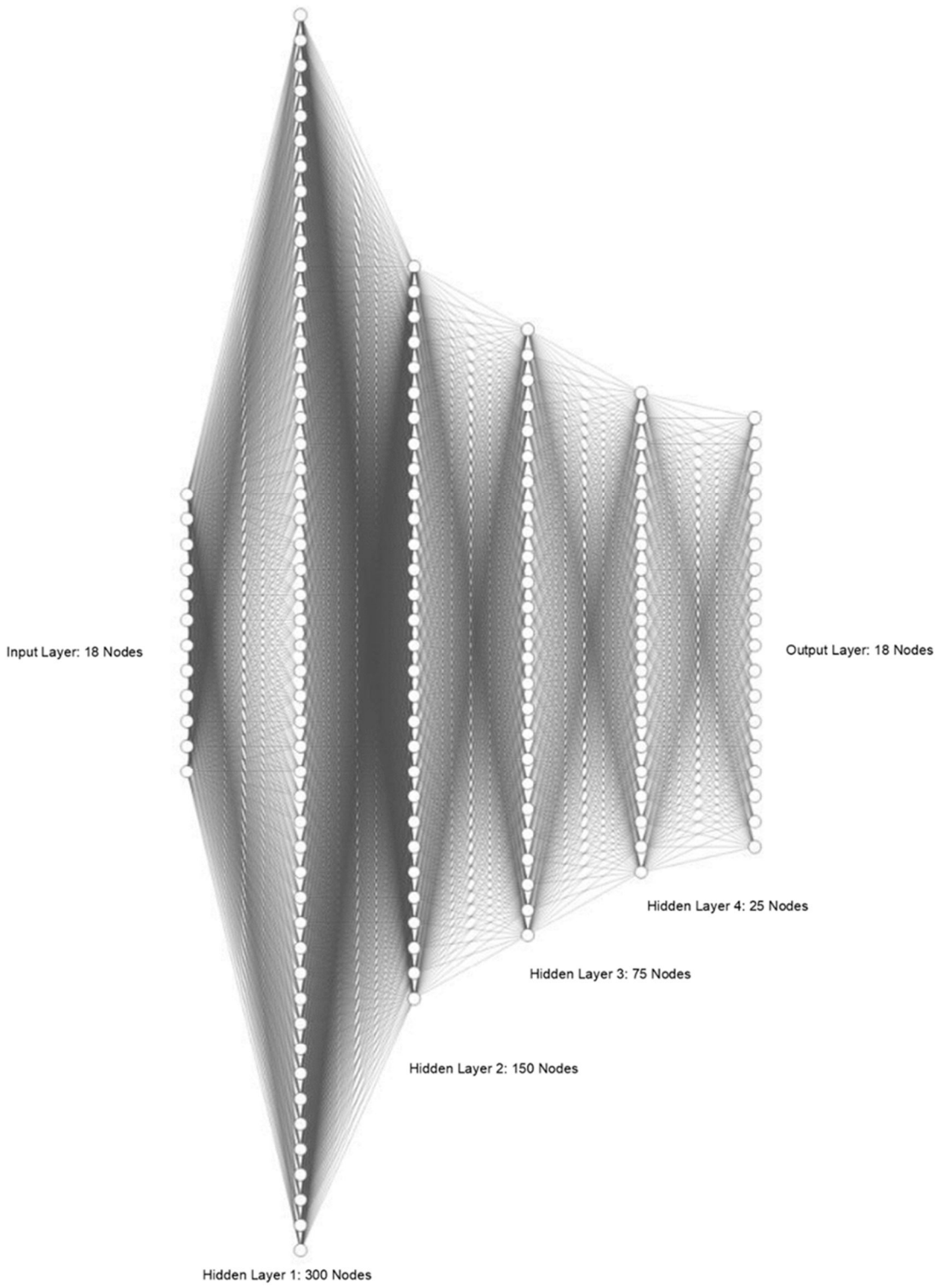

Figure 6: Single-Entry ANN architecture

Before the EMG and classification data could be fully utilized for the single-entry neural networks, they first had to be reshaped, resized, and in some cases reformatted. This process was the simplest for the single-entry cross entropy ANNs as the EMG data for both the testing and validation/training was just the EMG data converted to an array. Before this data could be used in the CNN's it first had to be reshaped to size (x, 1, 12, 1). In this, x represents the number of entries that are contained in the EMG data array, the first one represents the fact that there is only single entry being examined at any one time, the 12 represents the data from the 12 EMG electrodes, and the final one represents the fact that this data has no depth (I.E there is no other EMG data series being examine after the first one is completed). The classification data required a little more work to be usable. First it had to be converted into an array using a for-loop, which while simple does increase the total processing time. The classification array then had to be one hot encoded, a process by which the data is converted from a numerical value to a binary array that represents the original numerical value. Essentially this means that, if there were five different possible classification results a one hot encoded "3" would become "[0 0 1 0 0]". Once that was done, the data could be properly utilized in the CNNs and the ANNs. This process was the same for both the binary and the categorical cross entropy neural networks.

Reformatting the data for the feature extracted neural networks was a more involved process due, in part, to the need to program the feature extraction method. This method was programmed to accept the "raw" EMG dataframe that contained the EMG electrode data and the classification data, as well as an integer "*i*" that represented which electrode the data was extracted from. A while-loop was utilized to run through a data window containing 700 points of the EMG electrode data. If the last value of this window was found to be outside the bounds of

how many points of electrode data there were, the window was shrunk to end at the last point of data in the dataset. Within each data window, the values were run through an RMS algorithm (as described in chapter 2, and the classification values for the data in the window were averaged and rounded to the next integer value. Once this process was done, the window shifted down by 600 data points, allowing for a 100-point overlap in each window. This overlap was done in order to improve the resolution of the feature extracted data. This method was run on each individual electrode, which was then converted into an array of arrays, with each array containing the feature extracted data from one of the electrodes. The array of arrays was then converted into a single dataframe, which was then reshaped and resized in the same way the data for the single-entry cross entropy neural networks was.

The neural networks all utilized the Adam algorithm for their optimizer metric, other optimizers were tested and were found to be less than optimal when compared to the results that the Adam algorithm produced. The number of epochs, that is to say the number of times the neural networks ran through the data, was set to five for all single-entry neural networks. This value, too, was examined and found to be the most optimal value (values lower than five saw a reduction in accuracy, and values over five had an increase in processing time). The batch size, on the other hand, had to be reduced for the feature extracted neural networks, due to the reduced number of values being examined. For these, the batch size was set to 21, while the batch size for the much larger single-entry cross entropy neural networks was set to 210.

It should also be mentioned that the neural networks were trained on the data from one subject, validated on the data from a completely different subject, and re-tested on the data from yet another subject. This was done to fully ensure that the data would work on any subject and that overfitted wouldn’t occur. For each of these neural networks, the time it took to complete

one epoch was recorded for use in a speed comparison, which can be found in the results chapter. The accuracy and loss value for both the training data and the validation/testing data were recorded for each epoch and used to produce a learning curve and an accuracy curve for each neural network. These will be used as the basis for the conclusion of this research, the graphs can be found in chapter 5 and the discussion can be found in chapter 6.

4.3) *MULTI-ENTRY CNN'S*

For the most part, the processes behind the implementation of the multi-entry CNNs was much like what was done for the single-entry CNNs. The major difference being that the multi-entry CNNs accepted a window of data, as opposed to a singular point of data. For the two cross entropy CNNs, this window was set to 400 data points which represents a time frame of about 0.2 seconds. This value was chosen as the average time for a signal to travel from the central nervous system to the arm is generally recognized as 0.3 seconds. This created a slight issue, in that the length of the datasets for each subject was not even divisible by 400. This forced me to shorten the number of points in each EMG dataset, and while the number of points cut was relatively small this did introduce a possible point of error that will be discussed in more detail in chapter 7. Other than this shortening of the data, the processing of the data was much the same as what was done for the single-entry CNNs, the electrode data was converted to an array and reshaped to size (x, 400, 12, 1) with x representing the number of 400-point windows that the data creates, the 400 representing the window, the 12 representing the number of EMG electrodes, and the one representing the fact the data set has no depth. The classification data was

again one hot encoded, however this time the classification data first had to be reshaped to match the size of the EMG electrode data. This was done by implementing a helper method that found the rounded average of the classification value for each 400-point window, outputting a list of the values. From this list, an array was produced and one hot encoded.

The architecture for the two cross entropy CNNs were identical, with the exception of the loss function. Each starts with a convolutional layer accepting an input of size (40, 12, 1) with 40 filters of size (200,6), chosen as it is half the size of the input. This followed by a max pooling layer with a pool size of (2, 2), the value of which was chosen at random. This is followed by another convolutional layer with 40 filers of size (75, 3), chosen because it is a little under half the size of the previous filer. This was followed by another convolutional layer with 40 filters of size (25,1), chosen as it was 1/3rd the size of the last filter. This leads to a global max pooling layer, which converts the architecture from a convolutional layout to a dense layout. The first dense layer is made from 40 neurons, each representing the data from one filter. This leads to a dense layer with 20 neurons, chosen as it was half the size of the previous layer. Finally, we have the output dense layer with 18 neurons, each representing a different possible classification outcome. Each of the layer except the output layer uses a ReLU activation function, with the output layer using a SoftMax algorithm much like in the single entry CNNs. Each convolutional layer has a dropout rate of 30%, with the dense layers having a dropout rate of 20%. Finally, the architecture utilized an Adam algorithm for its optimizer function, has the same epoch and batch size as the single-entry feature extracted CNNs (epoch size of 5 and a batch size of 21) for much

the same reasons (optimal mix of speed and accuracy). The architecture for these CNNs can be found in figure 7.

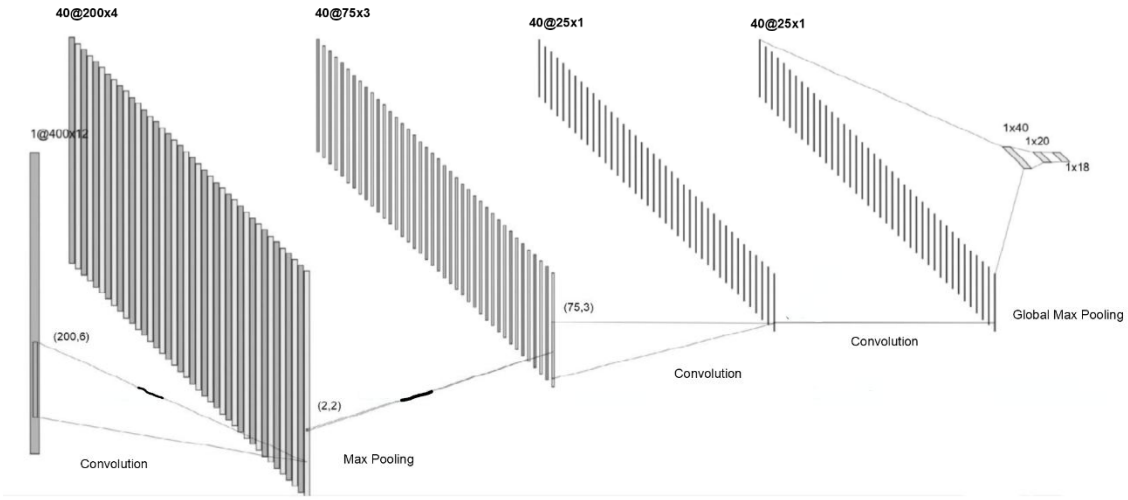

Figure 7: Architecture of Multi-Entry CNN

The feature extracted multi-entry CNN required the most work in regards to data preparation as it was required to first run the data through the RMS method before converting it to a size that allowed for a significantly sized window to be created. The feature extraction method was much the same as with the single-entry feature extracted CNN with the exception of the window size, which was changed to 400 for the multi-entry version. Once this was run for the 12 electrodes, the data was shortened so that both the training and the testing/validation data set only had 6000 points. This value was chosen as it allowed the data to create 300 windows of 20 data points, which seemed like a decent value to use. From this, the electrode data first had to be transposed, as somehow the column and row values had become swapped somewhere in the code. These transposed values were then reshaped into size (300, 20, 12, 1) representing the fact

that there were 300 windows of data size 20, each having the data from 12 electrodes, with only one dataset being used.

Unlike in the case of the single-entry CNNs, the architecture of the multi-entry feature extracted CNNs differs from its non-feature extracted counterpart. For these CNNs, the first convolutional layer was made from 40 filters of size (10, 6), representing half the size of the input shape (which was (20, 12)). This is followed by a max pooling layer with pool filter size of (2, 2). The next layer is a convolutional layer with 40 filters of size (5, 3), chosen as it was half the value of the previous convolutional layer. After this, a global max pooling layer is used to convert the convolutional layers into dense layers, starting with a dense layer of 40 neurons representing the 40 previous filters. This leads to a dense layer of 20 neurons, which leads to the output layer of 18 neurons. The dropout values, optimizer function, and activation functions for this CNN are the same as for the other multi-entry CNNs (30% dropout for convolutional layer, 20% for dense, Adam optimizer function, ReLU activation function for all layers except the output layer which uses SoftMax). For these CNNs, the epoch size was set to 15 as lower values still saw drastic changes in accuracy and loss (which will be shown in the graphs in the results chapter) and the batch size was set to 2, as larger values did not allow the CNN to train properly thanks to the small number of windows present in this format. The architecture described for the multi-entry feature extracted CNNs can be found in figure 8.

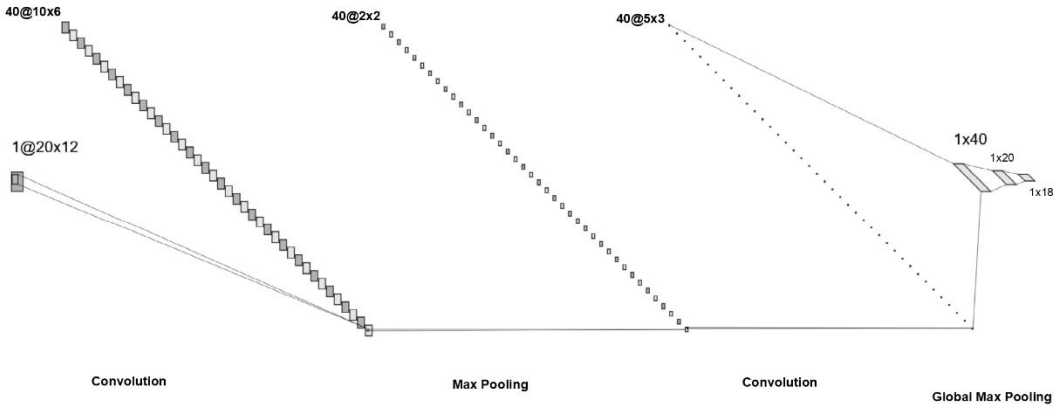

Figure 8: Multi-entry Feature Extracted CNN Architecture

CHAPTER 5

RESULTS

The results of the assorted neural networks are best represented through the use of learning curves, which are produced by recording the loss value for each iteration (which in the case of this experiment is the end of each epoch), accuracy curves, which are produced by finding the accuracy for each iteration, and two charts representing the time calculations done on the neural networks.

5.1) TIME CHARTS

The first time chart, labeled Neural Network Time Chart and found in figure 9, contains the information pertaining solely to the CNN run time, including average time per epoch, the number of epochs, and the calculated run time (which is simply the average time per epoch times the number of epochs). The second time chart, labeled the Total Run Time Chart and found in figure 10, contains the calculated total time from the previous chart, as well as the average time taken for each method to pre-process the data for use in the neural networks. It should be noted that some of the pre-processing times are the same due to the fact that the data processing and running of the neural networks was either done at the same time in a single code (which is the case for the single-entry CNN/ANN pairs), or utilized the same code with no change to the pre-processing method (which is the case for the multi-entry CNNs). The average preprocessing time

was calculated by timing how long it took for the program to go from start to the time the neural networks began running for three different runs, with the final value being the average of these runs. The final entry in the Total Run Chart represents the calculated total run time for each neural network which was found by adding the average pre-processing time to the calculated run time.

	SEBCNN	SEBANN	SECCNN	SECANN	SECFECNN	SECFEANN	SEBFECNN	SEBFEANN	MEBCNN	MECCNN	MEBFECNN	MECFECNN
Average Time Per Epoch	69.4	74.6	76.6	75.2	1	1	1	1	107.2	111.8	1	1
Number of Epochs	5	5	5	5	5	5	5	5	5	5	15	15
Calculated Run Time	347	373	383	376	5	5	5	5	536	559	15	15

Figure 9:Total Run Time Chart (in Seconds)

	SEBCNN	SEBANN	SECCNN	SECANN	SECFECNN	SECFEANN	SEBFECNN	SEBFEANN	MEBCNN	MECCNN	MEBFECNN	MECFECNN
Calculated CNN run time	347	373	383	376	5	5	5	5	536	559	15	15
Average Processing time	10.18667	10.18667	10.09667	10.09667	148.38	148.38	147.5567	147.5567	16.98667	16.98667	291.16667	291.1667
Total Time	357.1867	383.1867	393.0967	386.0967	153.38	153.38	152.5567	152.5567	552.9867	575.9867	306.16667	306.1667

Figure 10:Neural Network Time Chart (in Seconds)

From the Neural Network Time Chart, we see that the methods that involve feature extraction have the lowest average time per epoch, all at one second. The highest time per epoch, and total time, is found in the multi-entry CNNs, with an epoch run time of 107.2 seconds for the binary cross entropy CNN and 111.8 seconds for the categorical cross entropy CNN, and a total run time of 536 seconds for the binary CNN and 559 seconds for the categorical CNN (or 8 minutes and 56 seconds, and 19 seconds respectively). The single-entry neural networks had the median time ranging from 69.4 seconds for the single-entry binary cross entropy CNN to 76.6 seconds for the single-entry categorical cross entropy CNN. This produces a run time in the high 300's, with the fastest of the four methods being the single-entry binary cross entropy CNN with a run time of 347 seconds, or about 5 minutes and 47 seconds.

From the Total Run Time Chart, we can see that the two multi-entry feature extracted CNNs have the greatest processing time, spending a total of 291.167 seconds or 4 minutes and 51 seconds pre-processing the data so it can be used in the CNN. This trend can also be seen in the single-entry feature extracted neural networks, which have a data processing time in the mid-140's. The six non-feature extracted neural networks all have processing times in the 10's, with the time being slightly longer for the multi-entry CNNs at 16.99 seconds. However, even with the shorter processing times the six non-feature extracted neural networks take much longer than the feature extracted neural networks. The single-entry feature extracted neural networks finish their total run in about 150 seconds, or 2 minutes and 30 seconds, while the multi-entry feature extracted methods take 306.1667 seconds, or 5 minutes and 6 seconds. The multi-entry non-feature extracted methods take the longest time with the multi-entry categorical cross entropy CNN taking 575.98 seconds, or 9 minutes and 36 seconds. The total times of the single-entry non-feature extracted neural networks have the second highest values, being found in the mid-to-to-upper 300's.

5.2) *LEARNING AND ACCURACY CURVES*

As mentioned before, all neural networks run in this experiment produced two graphs, a learning curve produced from the loss function, and an accuracy curve produced from the neural networks self-calculated accuracy. Each of these graphs is provided and will have a detailed explanation of what they mean and the impacts they have being found in chapter 6.

5.2.1) SINGLE-ENTRY BINARY CROSS ENTROPY CNN

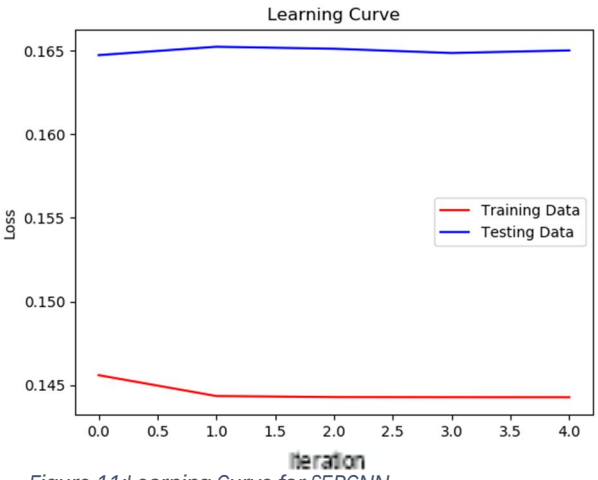

Figure 11:Learning Curve for SEBCNN

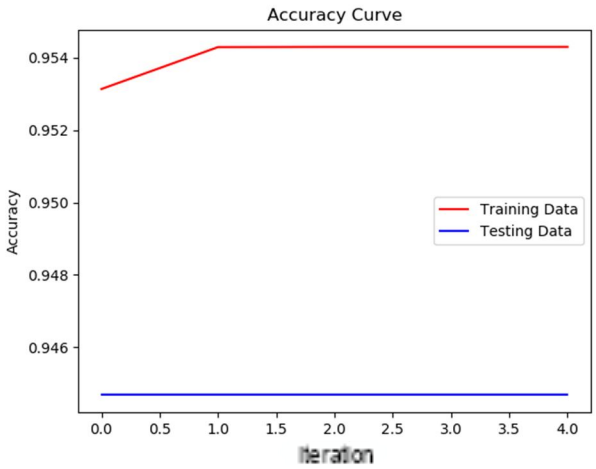

Figure 12: Accuracy Curve for SEBCNN

From the learning curve we see a general trend of both the training data and the testing/validation data staying close to their initial values, with the training data hovering around 0.145 and the testing/validation data centering around 0.165. It should be noted that iteration 0.0 represents the first epoch, with iteration 4.0 representing the fifth and final epoch. Simularily, we can see that the accuracy for the testing/validation data is constant throughout the iterations, being about 94.5%. The training data does see an increase in value jumping from 95.3% at the end of the first epoch to 95.4% at the end of the second epoch.

5.2.2) SINGLE-ENTRY BINARY CROSS ENTROPY ANN

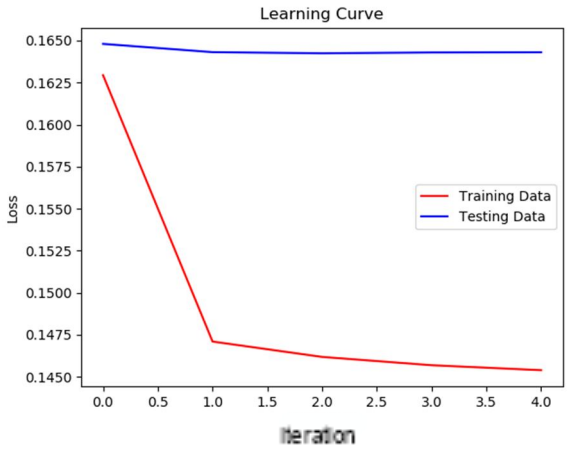

Figure 13:Learning Curve for SEBANN

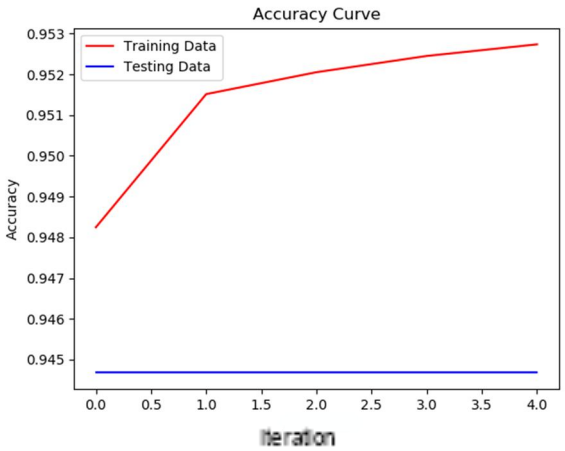

Figure 14:Accuracy Curve for SEBANN

From the learning curve we can see that as there is first a great decrease in the loss value of the training data from epoch one to epoch two, and a continual trend of decrease as the neural network runs, with the raining data approaching 0.145. There is also a minor decrease in the loss of the testing/validation data, however it is miniscule as the testing/validation data stays around 0.165. As for the accuracy, we see a continual improvement for the training data, as it slowly approaches a 95.3% accuracy. The testing/validation data however doesn't move from 94.5% accuracy.

5.2.3) SINGLE-ENTRY CATEGORICAL CROSS ENTROPY CNN

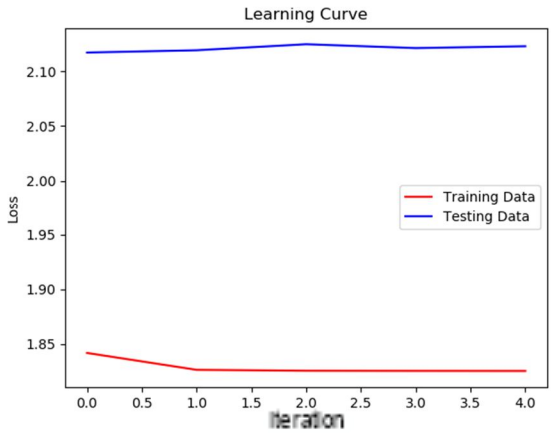

Figure 15:Learning Curve for SECCNN

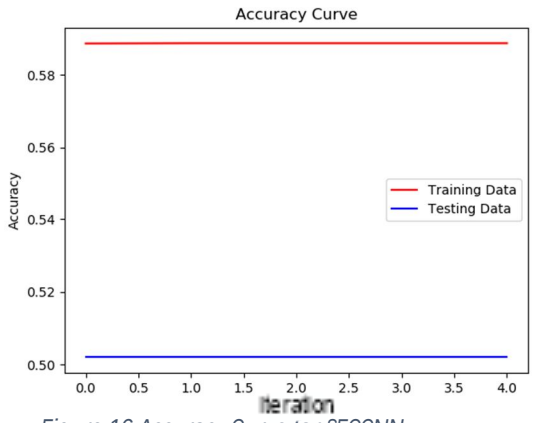

Figure 16:Accuracy Curve for SECCNN

From the learning curve graph, we see a drastic change compared to the previous CNN method. The training data loss has increased to about 1.85, representing an almost 10-times increase when compared to the single-entry binary cross entropy CNN. The testing/validation data sees an even worse increase, finding itself centered around 2.13. We see this drastic worsening reflected in the new accuracies, with the training data have an accuracy of about 59% and the testing/validation data having an accuracy of 50%.

5.2.4) SINGLE-ENTRY CATEGORICAL CROSS ENTROPY ANN

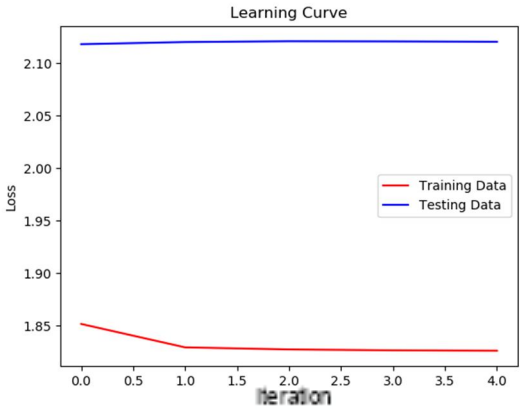

Figure 17: Learning Curve for SECANN

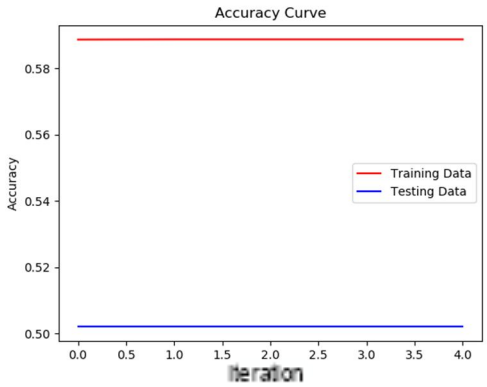

Figure 18:Accuracy Curve for SECAAN

We see that the learning and accuracy curves for the single-entry categorical cross entropy ANN is exactly like the single-entry categorical CNN, with the loss value of the testing/validation data centering around 2.1, the loss value for the training data decreasing to about 1.85, and both accuracies being in the 50% range. To make sure that this was not a mistake in the outputting of the graphs, I re-ran the neural networks and received the same results.

5.2.5) SINGLE-ENTRY FEATURE EXTRACTED BINARY CNN

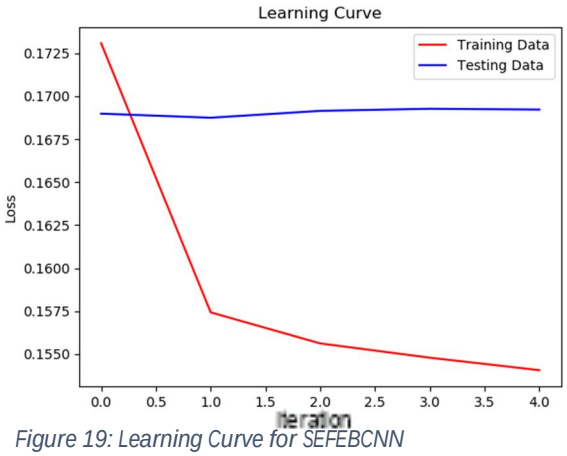

Figure 19: Learning Curve for SEFEBCNN

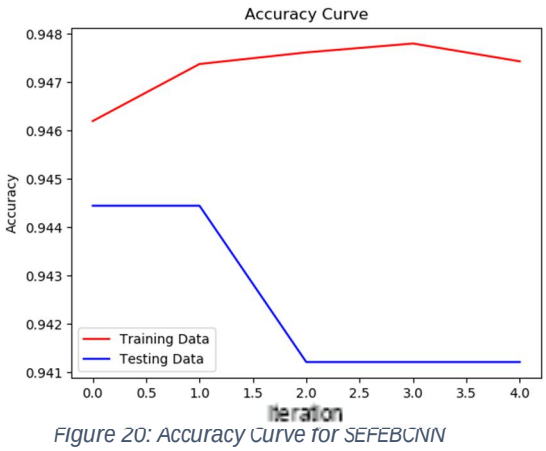

Figure 20: Accuracy Curve for SEFEBCNN

When combining the RMS feature extraction with the binary cross entropy, we get interesting results for the learning and accuracy curve. The trend for the train data in the learning

curve is much like it was for the non-feature extracted binary cross entropy ANN, with the only real differences being that the testing/validation data is centered around 0.1685 and the training data starting with a value higher than the testing/validation data, and ending at 0.155. The accuracy curve is extremely different as well, with the testing/validation data decreasing in accuracy after the second epoch and ending at 94.1% accuracy. The training accuracy makes an odd parabolic shape, starting at 94.6% accuracy, reaching a high point of 94.8% at the fourth epoch, and decreasing again to 94.7% at the final epoch.

5.2.6) SINGLE-ENTRY FEATURE EXTRACTED BINARY ANN

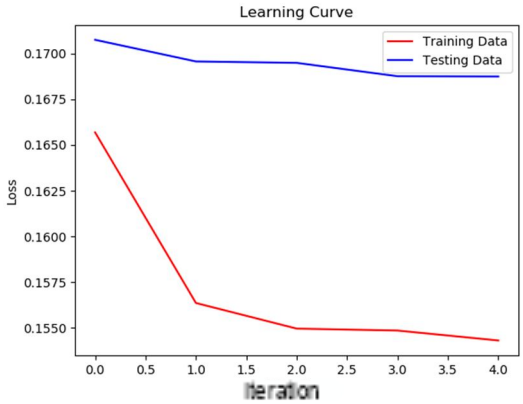

Figure 21:Learning Curve for SEFEBANN

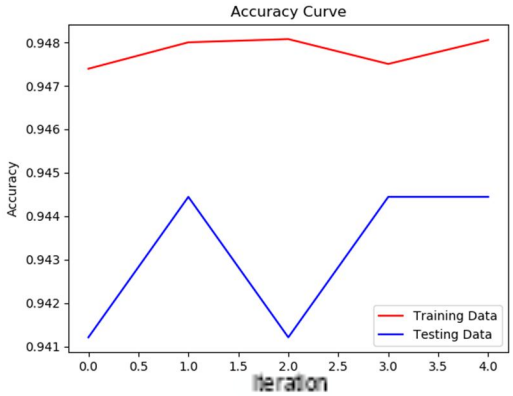

Figure 22:Accuracy Curve for SEFEBANN

The learning curve for the single-entry feature extracted binary ANN shows that both data sets produce a decrease in loss, with the training data starting at 0.165 and ending at 0.155 and showing a slight drop around 0.17. The accuracy curve, on the other hand, shows an oddity in the testing/validation data, with a spike in accuracy at the second and fourth epoch. The testing/validation data has a low point accuracy of 94.1% and a high point accuracy of 94.4%, with the training data accuracy fluctuating around 94.75%.

5.2.7) SINGLE-ENTRY FEATURE EXTRACTED CATEGORICAL CNN

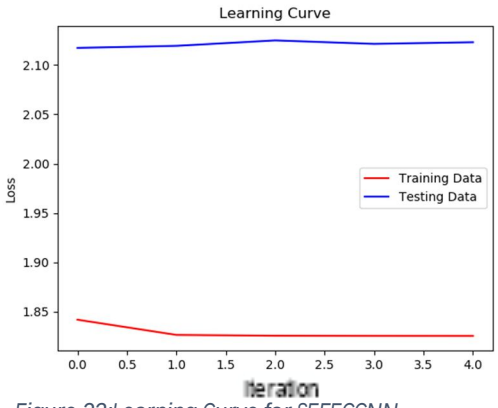

Figure 23:Learning Curve for SEFECCNN

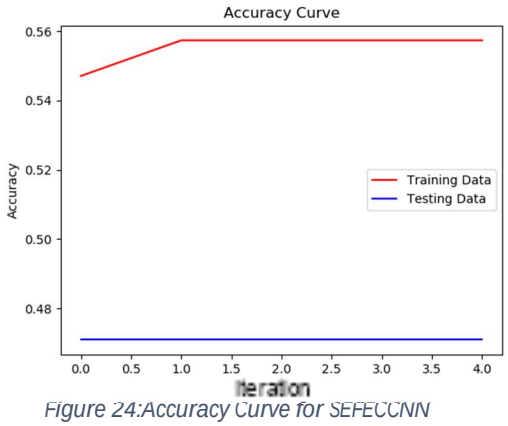

Figure 24:Accuracy Curve for SEFECCNN

The learning and accuracy curves for the single-entry feature extracted categorical CNN reflect the pattern seen in the two previous categorical cross entropy neural networks, with the training loss being around 1.85, the testing/validation loss being around 2.1, the training accuracy being around 56%, and the testing/validation accuracy being around 48.5%.

5.2.8) SINGLE-ENTRY FEATURE EXTRACTED CATEGORICAL ANN

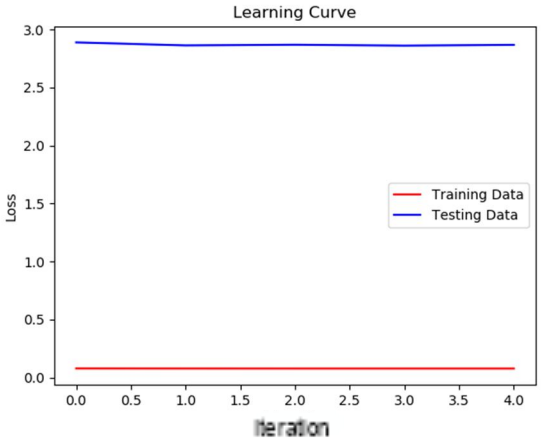

Figure 25:Learning Curve for SEFECANN

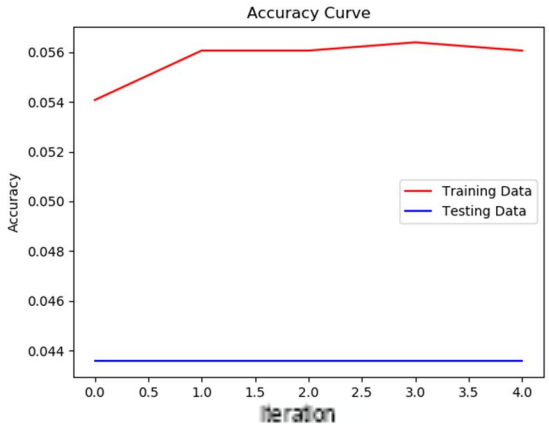

Figure 26:Accuracy Curve for SEFECANN

With the single-entry feature extracted categorical we see the absolute worse outcome. For the learning curve, the testing/validation data has a loss value of almost 3, the highest achieved throughout this experiment. There is also the greatest difference in loss value from

training data (which was a loss value of 0.1) to the testing/validation data. For the accuracy we see that the testing/validation data has an accuracy of 4% and the training data has an accuracy of 5%, the worst value achieved in this research.

5.2.9) MULTI-ENTRY BINARY CROSS ENTROPY CNN

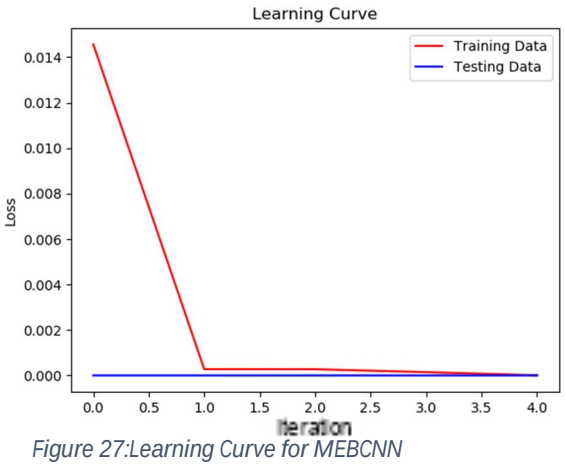

Figure 27:Learning Curve for MEBCNN

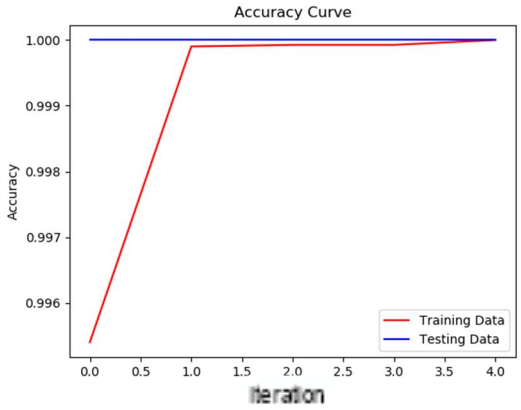

Figure 28:Accuracy Curve for MEBCNN

With the switch to multi-entry CNN's we see a drastic improvement in the loss and accuracy of both sets of data. For the training data, we see a starting loss of 0.014 that eventually decreases to 0, and a testing/validation loss that stays at 0 throughout the testing. The accuracy curve is a direct inverse to the learning curve, with the training accuracy starting at about 99.5% and increasing to 100%, and the testing/validation data staying at 100% the entire time.

5.2.10) MULTI-ENTRY CATEGORICAL CROSS ENTROPY CNN

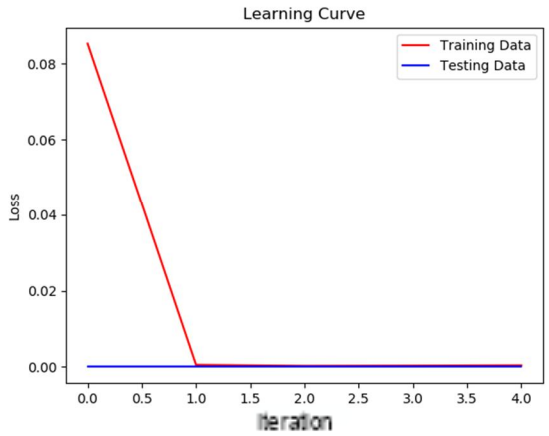

Figure 29:Learning Curve for MECCNN

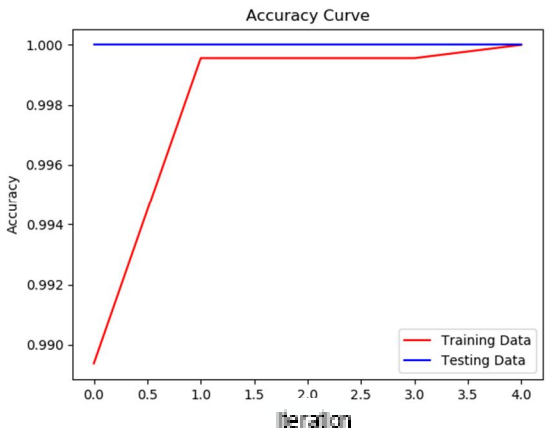

Figure 30:Accuracy Curve for MECCNN

The learning and accuracy curves for the multi-entry categorical cross entropy CNN, unlike in the single-entry case, are extremely similar to their binary cross entropy counterparts. The only difference is that the training loss starts at 0.08, and the training accuracy stays at about 99.9% for three epochs before reaching 100%.

5.2.11) MULTI-ENTRY FEATURE EXTRACTED BINARY CNN

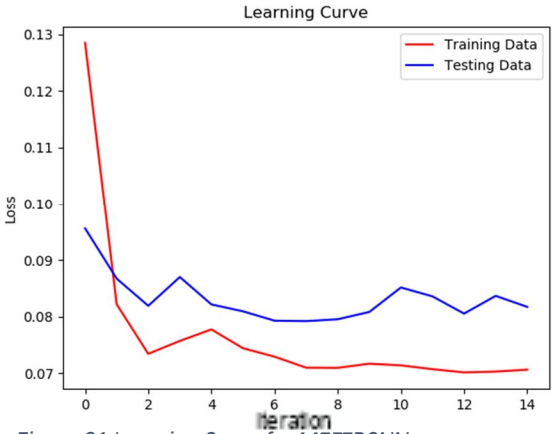

Figure 31:Learning Curve for MEFEBCNN

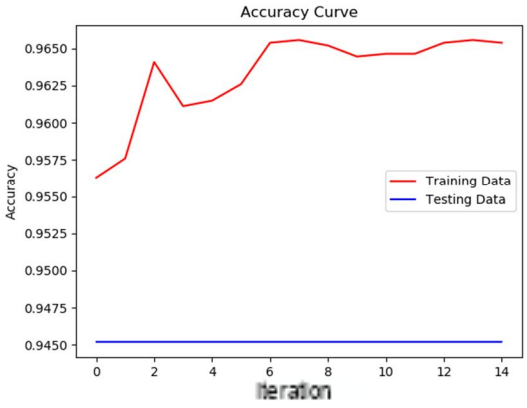

Figure 32:Accuracy Curve for MEFEBCNN

For the multi-entry feature extracted binary CNN, we see very interesting fluctuations in both the learning and the accuracy curves. Both the training and the testing/validation loss

generally decreases, with some infrequent spikes. We see that the testing/validation data is centered at around 0.085 for loss, with the training loss starting at 0.13 and dropping to around 0.07. The Accuracy curve, meanwhile, shows no change in the testing/validation data being 94.5% the entire time, and having the training data generally increase for 95.56% to 96.5%.

5.2.12) MULTI-ENTRY FEATURE EXTRACTED CATEGORICAL CNN

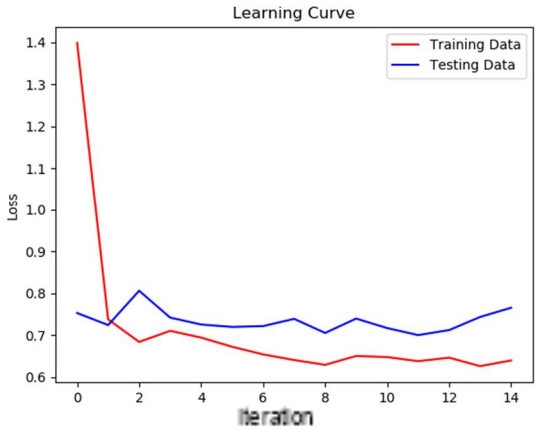

Figure 34:Learning Curve for MEFECCNN

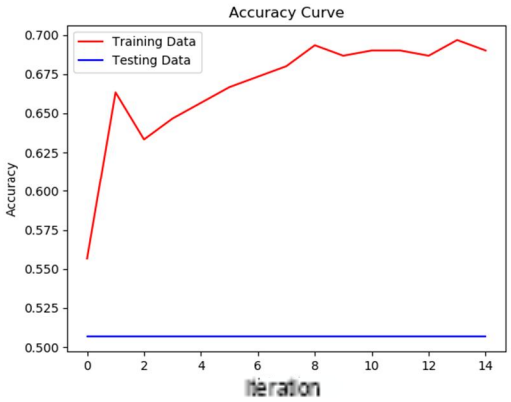

Figure 33:Accuracy Curve for MEFECCNN

The final test condition in the research saw a return to low accuracies, with the testing/validation accuracy being 51% throughout the testing, and the training accuracy starting at 55% and reaching a maximum of 67.5%. The loss, meanwhile, stayed relatively low, with the training loss starting at 1.4 and ending at 0.65. The testing/validation loss was centered at 0.75 and saw a little fluctuation.

CHAPTER 6

DISCUSSION

Starting with an analysis of the different loss charts, what we want to see is a lower loss value and a general trend of the testing/validation data decreasing. For every single-entry neural network (with the exception of the two binary ANNs), we see either a constant or an increasing training loss value. For each of the single-entry categorical cross entropy neural networks, we also see a relatively high loss value. These facts lead to the conclusion that the single-entry categorical cross entropy neural networks would make extremely poor models for use in a commercial upper-limb prosthetic. This is also reflected in the fact that overall accuracy for the single-entry cross entropy models is extremely low, being in the 50% range for both non-feature extracted models and the feature extracted CNN and reaching as low as 5% in the feature extracted ANN. Looking at the data, we can see a general trend of the ANN working better under the binary cross entropy than the CNN under the same conditions.

If we begin taking into account the accuracy graphs, we see that the binary cross entropy models produce close to the same accuracy no matter the condition, falling in the 94-96% range. As mentioned previously, the single-entry categorical cross entropy feature extracted ANN performs the absolute worse in regard to accuracy, which completely removes it from consideration in the possibility of being a usable prosthetic neural network. The multi-entry non-feature extracted CNN performs the best in terms of accuracy, reaching an overall total of 100% for both the training and the testing/validation data. The multi-entry feature extracted categorical

cross entropy CNN performs slightly worse than its binary counterpart, however, not enough to completely discount it from possible use. It could be possible to improve the accuracy of the multi-entry feature extracted categorical cross entropy CNN through the alteration of any number of variables. This would need to be examined in further detail in the future.

With all that in mind, we can add in the information from the time charts. We see that the feature extracted methods have both the lowest epoch and network run time as well as the longest processing time. Comparatively, the difference in processing time between the single-entry and the multi-entry non-feature extracted methods is not enough to discount the multi-entry models. The model run time, however, does show that the multi-entry non-feature extracted models require about 1.5x the time to complete than the single-entry models. This culminates in a long total run time, maxing out at a total time of 9 minutes and 36 seconds. While this is better than some of the models discussed in chapter 2, it is still the worse run time found in this research. This leads to the difficult decision of deciding if the difference in accuracy is worth the difference in time. We can also see that the multi-entry feature extracted CNNs have a total run time extremely similar to the single-entry non-feature extracted neural networks. Showing that the time lost in the double processing of the data is balanced by the rapid run time.

Complexity wise, the single-entry non-feature extracted models are by far the simplest, requiring only the EMG data to produce results. The single-entry feature extracted models have a comparative complexity to the multi-entry non-feature extracted CNNs as both required the data to be reformatted only once. Looking at the data processing time, however, shows the possibility that the feature extraction process is more complex than the data reshaping process. This, of course, means that the multi-entry feature extracted CNN would be the most complex of the methods tested, as it requires both methods to be carried out.

Taking all these factors into account, it is safe to say that any of the single-entry binary neural networks would work for a quick, simple, and mostly accurate method of classification for prosthetics. However, it can also be said that the overall best method would be the multi-entry non-feature extracted categorical cross entropy CNN as the increase in time is worth the overall increase in accuracy. It is also possible that the multi-entry feature extracted categorical cross entropy CNN could potentially be the best method, if future work shows that the accuracy can be increased to the same level as its non-feature extracted counterpart.

While possible points of error were minimized in this research, there are still some points that could cause errors in the results. The chief among them is the ever-present human error, which is always extremely likely to occur during research and experimentation. It is also possible that the code created is working in a way that is unlike that described in this report, which would introduce an understanding error. The data could also contain an error as it was provided by a third party (NINApro), however there is nothing that can be done to remedy this error for the current research. The best way to counter this in the future would be to record and produce a dataset specifically for this research, something that was avoided in this instance as time was an important factor in the completion of this research. Most other sources of error were countered, each part of the code was examined to ensure that it was understood, and the human factor was limited as much as possible.

CHAPTER 7

CONCLUSION

It is the conclusion of this report that, in regards to the creation of a simple, quick, and accurate EMG classification method for use in upper limb prosthetics the best architecture is either a single-entry binary cross entropy neural network, with no feature extraction method, or a multi-entry categorical cross entropy CNN. These methods produce an accuracy in the high 90%'s, with the multi-entry CNN producing an overall accuracy of 100%. Both methods have a low complexity, requiring very little data processing time. The single-entry neural networks are better where speed is the deciding factor, with a total run time around the six-minute range. If accuracy is the primary concerning factor, then the multi-entry CNN is best, as it produces the highest accuracy at the slowest run time, taking over 9 minutes.

There were some limitations to the work done in this research that were unavoidable. Most prominent among them was that the dataset utilized in this research was provided by other researchers. While this did save time by allowing the avoidance of the data collection stage (the most time consuming stage in any research) and the signal processing stage (which is a vital step in all EMG research), it also limits what can be done in regards to data collection and signal processing. While the time save was worth the loss of flexibility in data collection and signal processing in this instance, it would have been interesting to examine how these would have impacted the final results. The other limitation was that invasive EMG was impossible to utilize, as I lack both the proper training and the proper facilities to perform the surgery needed to implant invasive EMG probes. While this does mean the research done had no ethical issues involved, it is also true that the signal recorded by invasive EMG would have solved some issues presented by surface EMG, like movement noise (due to the electrode being attached to the

target muscle surgically) and cross talk (for the same reason as the movement noise). Finally, there was not enough time in my research to attempt to utilize surface EEG as well, as that would have required data acquisition, as well as training to properly place the EEG electrodes in the proper places. This is not much of a limitation, as the conventional method by which upper limb prosthetics are controlled is EMG, however it would have provided an interesting contrast and expand the possibility of this research into the field of patients with paralysis (where EMG would be unusable) as opposed to patients with simple surgical amputations.

CHAPTER 8

FUTURE WORK

Continuing on to the future, it would be interesting to look into the possibility of improving the accuracy of the multi-entry feature extracted cross entropy CNN, as I feel it shows the greatest potential for future use. It would also be useful to determine how much of a change the usage of noisy, raw EMG data has on the methods, and if there is a way to complete the entire process, from signal acquisition to signal processing to classification, in a shorter time. With the use of noisy data, it would also be interesting to see what impact DWT would have on the speed and complexity of the final product.

If the opportunity arises, I would also be extremely interested in shifting towards sensorimotor functionality of upper-limb prosthetics, a much more complicated and intense aspect of advanced prosthetics. With training, I would also be interested in working on the hardware of the prosthetics in addition to the software. With this I would be able to better meld both aspects together and have a better understanding of the end results. Additionally, work with invasive electrodes and EEG would provide the opportunity to expand the possible patient base that could utilize the produced prosthetics. All of the aforementioned future work would require research in additional field, ranging from robotic in the case of the prosthetic hardware to neural physiology in the case of the EEG research. While this does limit what can be done in the immediate future, with time it should be possible to expand the research into each of these fields.

REFERENCES

[1] Parajuli, N.; Sreenivasan, N.; Bifulco, P.; Cesarelli, M.; Savino, S.; Niola, V.; Esposito, D.; Hamilton, T.J.; Naik, G.R.; Gunawardana, U.; Gargiulo, G.D. Real-Time EMG Based Pattern Recognition Control for Hand Prostheses: A Review on Existing Methods, Challenges and Future Implementation. Sensors (2019), 19, 4596.

[2] Karan Veer & Tanu Sharma A novel feature extraction for robust EMG pattern recognition, Journal of Medical Engineering & Technology, 40:4, 149-154, (2016) doi: 10.3109/03091902.2016.1153739

[3] F. Duan, L. Dai, W. Chang, Z. Chen, C. Zhu and W. Li, "sEMG-Based Identification of Hand Motion Commands Using Wavelet Neural Network Combined With Discrete Wavelet Transform," in IEEE Transactions on Industrial Electronics, vol. 63, no. 3, pp. 1923-1934, March 2016. doi: 10.1109/TIE.2015.2497212

[4] Phinyomark, Angkoon & Limsakul, Chusak & Phukpattaranont, P. Application of Wavelet Analysis in EMG Feature Extraction for Pattern Classification. Measurement Science Review. 11. 45-52. (2011). DOI:10.2478/v10048-011-0009-y.

[5] Atzori, M., Gijsberts, A., Castellini, C. et al. Electromyography data for non-invasive naturally-controlled robotic hand prostheses. Sci Data 1, 140053 (2014) doi:10.1038/sdata.2014.53

[6] Cai S, Chen Y, Huang S, Wu Y, Zheng H, Li X and Xie L SVM-Based Classification of sEMG Signals for Upper-Limb Self-Rehabilitation Training. Front. Neurorobot. 13:31. (2019). doi: 10.3389/fnbot.2019.00031

[7] Lucas, Marie-Françoise & Gaufriau, Adrien & Pascual, Sylvain & Doncarli, Christian & Farina, Dario. Multichannel Surface EMG Classification Using Support Vector machines and Signal-based Wavelet Optimisation. Biomedical Signal Processing and Control. 3. (2008). Doi: 169-174. 10.1016/j.bspc.2007.09.002.

[8] A. Subasi, A. Alaskandarani, A. A. Abubakir and S. M. Qaisar, "sEMG Signal Classification Using DWT and Bagging for Basic Hand Movements," 2018 21st Saudi Computer Society National Computer Conference (NCC), Riyadh, 2018, pp. 1-6. doi: 10.1109/NCG.2018.8593010

[9] Xia, Peng & Hu, Jie & Peng, Yinghong. (2017). EMG-Based Estimation of Limb Movement Using Deep Learning With Recurrent Convolutional Neural Networks. Artificial Organs. 42. 10.1111/aor.13004.

[10] Wentao Wei, Yongkang Wong, Yu Du, Yu Hu, Mohan Kankanhalli, Weidong Geng,A multi-stream convolutional neural network for sEMG-based gesture recognition in muscle-computer interface, Pattern Recognition Letters, Volume 119, 2019, Pages 131-138, ISSN 0167-8655, https://doi.org/10.1016/j.patrec.2017.12.005.

[11] Zhai, X., Jelfs, B., Chan, R., & Tin, C. (2017). Self-Recalibrating Surface EMG Pattern Recognition for Neuroprosthesis Control Based on Convolutional Neural Network. *Frontiers in neuroscience*, *11*, 379. doi:10.3389/fnins.2017.00379

[12] M. Arozi et al., "Electromyography (EMG) signal recognition using combined discrete wavelet transform based on Artificial Neural Network (ANN)," 2016 2nd International Conference of Industrial, Mechanical, Electrical, and Chemical Engineering (ICIMECE), Yogyakarta, 2016, pp. 95-99. doi: 10.1109/ICIMECE.2016.7910421

[13] Chowdhury, R. H., Reaz, M. B., Ali, M. A., Bakar, A. A., Chellappan, K., & Chang, T. G. (2013). Surface electromyography signal processing and classification techniques. *Sensors (Basel, Switzerland)*, *13*(9), 12431–12466. doi:10.3390/s130912431

[14] Atzori, M., Cognolato, M., & Müller, H. (2016). Deep Learning with Convolutional Neural Networks Applied to Electromyography Data: A Resource for the Classification of Movements for Prosthetic Hands. *Frontiers in neurorobotics*, *10*, 9. doi:10.3389/fnbot.2016.00009

[15] Phinyomark, A.; Scheme, E. EMG Pattern Recognition in the Era of Big Data and Deep Learning. *Big Data Cogn. Comput.* 2018, *2*, 21.

[16] R. Alam, S. R. Rhivu and M. A. Haque, "Improved Gesture Recognition Using Deep Neural Networks on sEMG," *2018 International Conference on Engineering, Applied Sciences, and Technology (ICEAST)*, Phuket, 2018, pp. 1-4. doi: 10.1109/ICEAST.2018.8434493

[17] U. Côté-Allard *et al.*, "Deep Learning for Electromyographic Hand Gesture Signal Classification Using Transfer Learning," in *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 27, no. 4, pp. 760-771, April 2019. doi: 10.1109/TNSRE.2019.2896269

[18] Y. Fu, X. Xiong, C. Jiang, B. Xu, Y. Li and H. Li, "Imagined Hand Clenching Force and Speed Modulate Brain Activity and Are Classified by NIRS Combined With EEG," in *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 25, no. 9, pp. 1641-1652, Sept. 2017. doi: 10.1109/TNSRE.2016.2627809

[19] L. Vargas, H. H. Huang, Y. Zhu and X. Hu, "Merged Haptic Sensation in the Hand during Concurrent Non-Invasive Proximal Nerve Stimulation," *2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, Honolulu, HI, 2018, pp. 2186-2189. doi: 10.1109/EMBC.2018.8512707

[20] L. Vargas, G. Whitehouse, H. Huang, Y. Zhu and X. Hu, "Evoked Haptic Sensation in the Hand With Concurrent Non-Invasive Nerve Stimulation," in *IEEE Transactions on Biomedical Engineering*, vol. 66, no. 10, pp. 2761-2767, Oct. 2019. doi: 10.1109/TBME.2019.2895575

[21] D. A. Bjånes and C. T. Moritz, "A Robust Encoding Scheme for Delivering Artificial Sensory Information via Direct Brain Stimulation," in *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 27, no. 10, pp. 1994-2004, Oct. 2019. doi: 10.1109/TNSRE.2019.2936739

[22] Kuzborskij, I., Gijsberts, A. & Caputo, B. On the challenge of classifying 52 hand movements from surface electromyography. in Proc. EMBC—34th Annu. Conf. IEEE Eng. Med. Biol. Soc. 2012, 4931–4937 (2012).

[23] Pearson, R.K., Neuvo, Y., Astola, J. et al. Generalized Hampel Filters. EURASIP J. Adv. Signal Process. 2016, 87 (2016) doi:10.1186/s13634-016-0383-6

[24] Morbidoni, C.; Cucchiarelli, A.; Fioretti, S.; Di Nardo, F. A Deep Learning Approach to EMG-Based Classification of Gait Phases during Level Ground Walking. *Electronics* 2019, *8*, 894.

[25] LeNail, (2019). NN-SVG: Publication-Ready Neural Network Architecture Schematics. Journal of Open Source Software, 4(33), 747, https://doi.org/10.21105/joss.00747