Artificial Muscle Signal Generation using Generative Adversarial Networks

BACHELORARBEIT
ZUR ERLANGUNG DES AKADEMISCHEN GRADES
BACHELOR OF SCIENCE

HOCHSCHULE RUHR WEST
UNIVERSITY OF APPLIED SCIENCES

INSTITUT **INFORMATIK**

Mahdi El Mesoudy
10007236

INSTITUT INFORMATIK
DER HOCHSCHULE RUHR WEST

Bottrop, den 20. Dezember 2021

Betreuer, Erstgutachter:	*Prof. Dr. Ioannis Iossifidis, Institut Informatik*
Zweitgutachter:	*M.Sc. Marie Schmidt, Insitut Informatik*

Ich versichere, dass ich diese Bachelorarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Mahdi El Mesoudy

Bottrop, den 20. Dezember 2021 Mahdi El Mesoudy

Acknowledgments

I would like to thank my supervisors, Prof. Dr. Ioannis Iossifidis and Marie Schmidt for the chance to take part in this work and also for the support during the time in the robotics laboratory. I would also like to thank everyone who worked in the laboratory during this time. In addition, I would like to thank my parents who gave me the opportunity to come to Germany and complete my studies.

Contents

Part I.

Introduction and Theory

0.1. Abstract

This work aims to generate synthetic electromyographic (EMG) signals using Generative Adversarial Network (GAN). GANs are considered as one of the most exciting and promising approaches in deep learning [6], offering the possibility to generate artificial data based on real data. GAN consists of two main parts, a discriminator that attempts to differentiate between the generated data and the original data, and a generator that tries to fool the discriminator by generating data which looks like real data, the GAN works by staging a two-player minimax game between generator and discriminator networks. To achieve the objective of generating realistic artificial electromyographic signals, two different architectures are considered for the generator and the discriminator networks of the GAN model: Long short-term memory (LSTM), which can avoid the long-term dependency problem and remembers information over a long period of time, and convolutional neural network (CNN), which is a powerful tool at automatic feature extraction. Different combinations of CNN and LSTM including hybrid model are experimented within the GAN using the same training data-set. The results and performances of each combination are compared and reviewed. The generated artificial EMG signals can be used to simulate real muscle activity situations to for example improve muscle signal controlled prostheses using artificial data that may include conditions that does not exist in real data. This method of artificial data generation is not limited to EMG signals, the network can also be used to generate other synthetic biomedical signals such as electroencephalogram (EEG) or electrocardiogram (ECG) that can be practically used for testing algorithms and classifiers [16].

CHAPTER 1

Introduction and Objectives

1.1. Introduction

Artificial data has a wide variety of applications such as software testing, deep learning model development and validation, image processing, and others. For example, in robotics field especially by prostheses development, real-life testing of systems can be expensive and time consuming. Artificial data allows then to test many robotics solutions in thousands of simulations without the need of collecting real-world data by actual events. Thus, EMG signals have gained great interest as they pave the way for a new method of human-machine interface [7]. Artificial EMG signals generation can be useful for data scientists to test existing system performance as well as to train new systems on different scenarios that are not represented in the authentic data without any privacy concerns. There are many methods to generate synthetic data, however, one of the most successful frameworks for generating artificial data is generative adversarial networks. GANs have mainly been developed and applied to the generation of images, and they have been successfully used not only to generate high quality synthetic images but also to produce realistic time series data such as EEG and EMG signals [16].

1.2. Objective and motivation

This work aims to generate artificial EMG signals. Special types of the GAN model are developed for this purpose. The first one known as Deep Convolu-

tional Generative Adversarial Network (DCGAN) [21], which consists mainly of convolutional neural networks. The second one is a combination of convolutional neural network and long short-term memory. The hyperparameters of both GANs are manually tuned and their performances are compared against each other to evaluate which GAN architecture is the best for this type of problem solving. Another objective of this work is to develop a 3 dimensional game to help collecting real EMG signals.

1.3. Structure of the work

The work is divided into two main parts with a total of five chapters. The first part mainly focuses on the background and theory used in this work, it deals with the explanation of the problem and the methods used to solve it. The second part includes two chapters. The first chapter shows the data collection process as well as the signal processing steps and the second chapter handles the proposed GAN networks and shows the results of the work with a discussion about the performances of the networks.

CHAPTER 2

Basics and Theory

2.1. Neural Networks

Neural networks are inspired by the biological brain and can be used for machine learning and artificial intelligence [26]. These networks can be used to solve various problems in a computer-based manner, such as problems that cannot be formulated as an algorithm, due to its ability of learning. The simplest and probably most known neural network is the feed-forward network, which consists of a (possibly large) number of neuron or processing units, organized in three types of layers, input layer, hidden layer and output layer as shown in Figure 2.1. Each neuron in a layer is connected to all the neurons in the previous layer except the input layer, the latter takes information as an input in the form of patterns or signals from the data set and simply passes these values along to the hidden layer without further processing [23]. The hidden layer is located between the input and output layers, and its neurons act as feature detectors, as such, playing a critical role in the operation of the neural network. As the learning process progresses across the network, the hidden neurons begin to gradually discover the salient features that characterize the training data [25]. The output layer transmits information to the outside as a result.

The connection between neurons have weight and are also called connection weight. These connection carries information between neurons and allow the output of one neuron to become the input of the next neuron. The stronger the weight of a connection, the greater the influence of a neuron can have on

another.

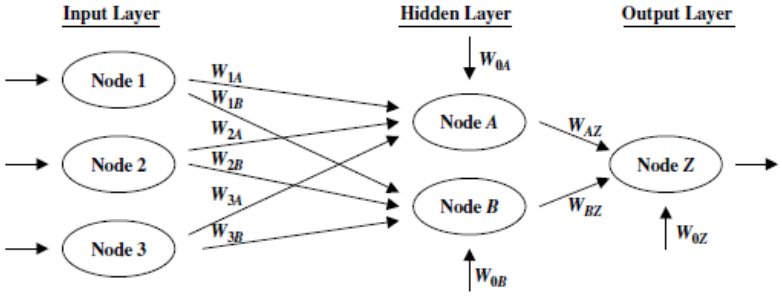

Figure 2.1.: Feed-forward neural network with three input nodes and one output node with two nodes as a hidden layer. [23]

Each neuron in the hidden and output layers carries a value a, that can be calculated by multiplying the value of the previous neuron A and the connection weight W. The result of this operation passes into an activation function g as shown in this formula [20]:

$$a_i = g(W_i * A_i) \tag{2.1}$$

There are two types of activation function, linear and nonlinear activation functions. However, nonlinear activation functions are the most used in neural networks such as logistic function (Sigmoid) and rectified linear unit (ReLU) to add nonlinearity, which is required to handle more complex and nonlinear datasets. The sigmoid function takes any real value as input and bind the output in the range of [0,1]. The larger the input the closer the output value will be to 1 and the smaller the input the closer the output value will be to 0. The sigmoid function is represented by the following formula [20]:

$$f(x)_{\text{Sigmoid}} = \frac{1}{1 + e^{-x}} \tag{2.2}$$

The ReLu function is the most commonly used activation function in convolutional neural networks. It converts all negative values to 0 and applies a linear function on the positive values as shown bellow [20]:

$$f(x)_{\text{ReLu}} = \max(0,x) \tag{2.3}$$

In some cases, the use of ReLu function can lead to a problem known as the Dying ReLU problem, where some neurons takes in the value of zero during training and never get activated again. These neurons are known as the dying neuron and they do not help the network in training [15]. To overcome this

issue, there is Leaky ReLU activation function, which is similar to ReLu on the positive side but does not ignore the negative inputs completely. Instead of that, the negative values get an extremely small linear component of x rather than zero to avoid the dying neuron problem. The Leaky ReLU function is represented by the following formula [20]:

$$f(x)_{LeakyReLU} = \begin{cases} x, & x>0 \\ 0.01x, & x\leq 0 \end{cases} \tag{2.4}$$

2.2. Convolutional Neural Network

Convolutional neural network (CNN) is a special kind of multi-layer neural network and is inspired by the mechanism of an animal's visual cortex [18]. CNN is designed to recognize visual patterns directly from images with minimal processing by automatically and adaptively learning and extracting spatial hierarchies of features [19].

CNN consists of three types of layers: convolution, pooling, and fully connected layers. The first two layers aims to extract features from data, while the last one aims to deliver the final output as a result, such as class scores in the case of an image classification problem. The feature extraction part is applied by the convolution layer, which is the most important component of any CNN architecture. This layer contains a set of convolutional kernels (also called filters), that gets convolved and slide over the input data (N-dimensional metrics) to generate an output feature map as shown in figure 2.2. The mathematical operation shown in this figure called convolution, which is a specialized type of linear operation. The filter size f can be set by the user as well as other parameters such as stride s and padding p. Stride is the number of steps or pixels shifts over the input matrix. In other words, it corresponds to the distance between two successive filter positions, by increasing the stride of the convolution operation, this leads to a lower-dimensional feature map [15]. Padding, typically zero padding, is the process of adding zeros to the border of the input matrix to increase the feature map size or to keep the input matrix and the feature map in the same size. The relationship between the input dimension and the parameters f, s and p to calculate the output feature map size is defined as:

$$h' = \frac{h - f + p}{s} + 1 \tag{2.5}$$

$$w' = \frac{w - f + p}{s} + 1 \tag{2.6}$$

Where h' denotes the height of the output feature map, w' is the width of the output feature map, h stands for the height of the input image, w defines the width of the input image, f is the filter size, p denotes the padding of convolution operation and s denotes the stride of convolution operation.
After each convolution layer, it is strongly recommended to add an activation function such as ReLu function in order to increase the non-linearity transformation in the input data as shown in figure 2.3.

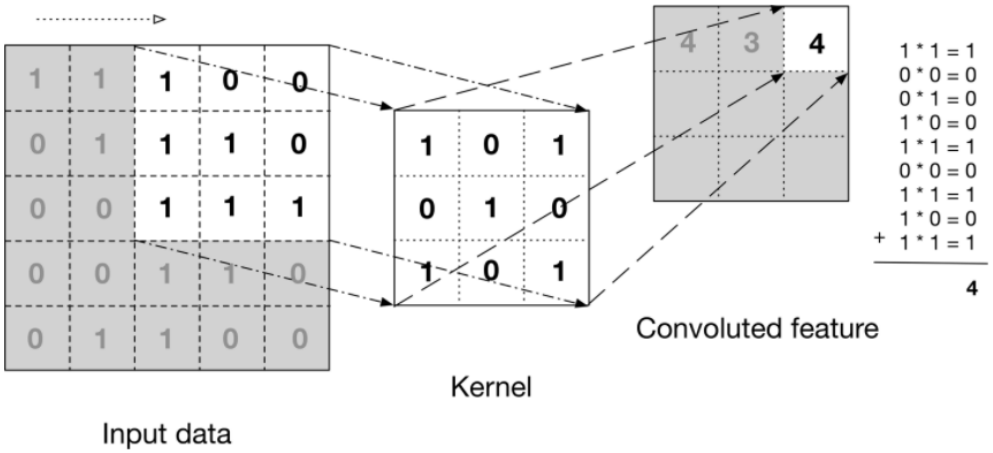

Figure 2.2.: Representation of a convolution operation by applying a 3x3 sliding filter on a 5x5 input that results in a 3x3 feature map. [3]

The other layer is a pooling layer, which follows the convolutional layer. Its main objective is to help CNN to find whether a specific feature is present within the given input, as well as to reduce the dimensions of the feature map by creating a down sampled or pooled feature map, which is a summarized version of the features detected in the input. There are different pooling approaches used in CNNs such as max pooling, min pooling and average pooling. However, max pooling is the most commonly used as pooling operation [15]. In this layer, and similar to convolutional layer, there is a sliding filter (e.i. 2x2) with a stride of 2 applied to the output of the convolution layer (feature maps), calculating the value for each patch of the feature map. In case of max pooling, this means that each 2×2 square of the feature map is down sampled to the maximum value in the square. This leads to a reduction of dimensions in the height and width of feature maps but the depth dimension remains unchanged [19].

The last layer type in CNN is the fully connected layer, which is simply the

feed forward neural network explained in the neural network section. The input of this layer is the output of the final pooling or convolutional layer, which is represented in the form of a set of metrics (feature maps), that are flattened as a vector and then fed into the fully connected layer. This layer is used as the output layer or classifier of the CNN architecture to deliver the final prediction score of the network [15] as shown in figure 2.4.

In this work, the use of another type of CNN layers that is called transposed convolution layer, is needed due to its upsampling functionality. This type of layer is used when some of the data must be reconstructed, for instance signals or images. The transposed convolution layer aims to generate an output feature map that has a spatial dimension greater than the dimension of the input feature map. Such as in GANs, where the generator network gets a random vector as an input and aims to reconstruct real data. Similar to the standard convolutional layer, the transposed convolutional layer uses the padding and stride to generate the output [13]. It is worth noting that there is a difference between the transposed convolutional and deconvolution, as deconvolution is a mathematical operation that reverses the effect of convolution operation. For example, passing an input through a convolutional layer and then passing this output to a deconvolution layer, the result of the deconvolution layer will be the exact same input to the convolutional layer. On the other hand, if a transposed convolutional layer is used instead of the deconvolution layer in this operation, the result will not be the same as the convolutional layer input but only the dimensions will match, as this only reconstructs the spatial dimensions of the input [13].

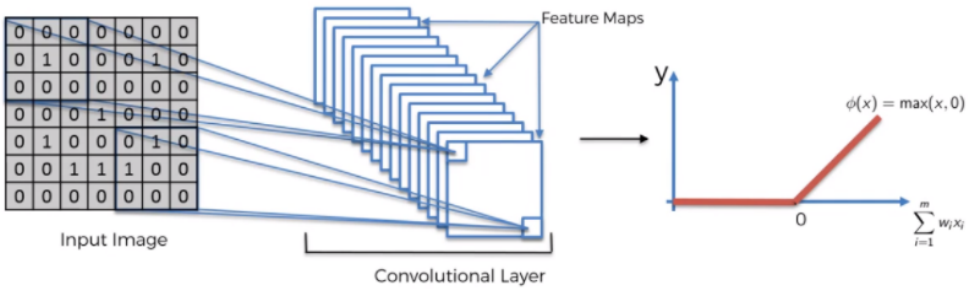

Figure 2.3.: The functionality of the feature extraction from the input by a Convolution layer followed by a ReLU activation function in a CNN network. [2]

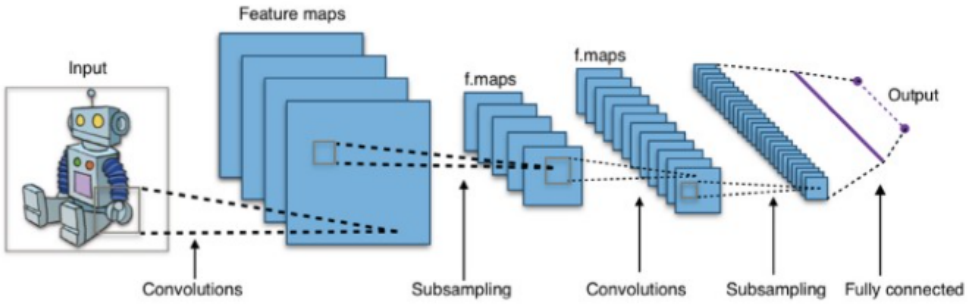

Figure 2.4.: Representation of a convolution neural network that takes an image as input and applies two convolution operations followed by subsampling layer each. And a fully connected layer at the end that results the output of the network. [1]

2.3. Long Short-Term Memory

Long short-term memory (LSTM) networks are a special kind of recurrent neural network (RNN), that were introduced by Hochreiter and Schmidhuber (1997) [28]. LSTM networks are specifically designed to avoid long-term dependencies by remembering information for long periods of time as well as to address the vanishing/exploding gradient problem [8], that RNN networks suffers from when the gradient shrinks or expands as it back propagates through time. LSTM solves these problems thanks to its internal mechanisms, known as gates, that have the ability to regulate information flow over time. As shown in figure 2.5, an LSTM cell consists of several components, the yellow rectangles indicate a layer operation, while red circles/ellipse represent a pointwise operation to combine the cell state and the hidden state with the outputs of several layers.

The cell state is the key concept of LSTM networks, it proceeds in a straight line along the entire chain with only some minimal linear interactions [8], and acts like 'the memory' of the network by adding relative and removing irrelative information with the help of gates, specifically the forget gate by using a sigmoid layer, this allows even information from earlier time steps to make its way to later time steps [10]. The hidden state however, have a different functionality, it acts like 'the working memory' of the network and it is more concerned with the most recent time-step by carrying information from immediately previous events and overwriting it at every step. There are three types of gates that characterizes an LSTM cell, these are: forget gate, input gate and output gate as shown in figure 2.5, each one does its own role:

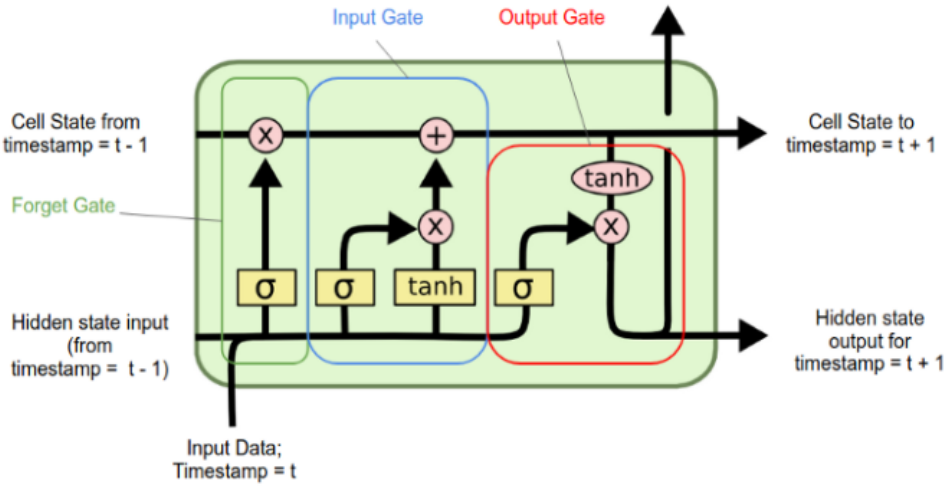

Figure 2.5.: A LSTM cell. The yellow rectangles represent a layer operation, while the red circles/ellipse indicate a pointwise operation to combine outputs of multiple layers with the cell state and hidden state. The symbol x indicates a multiplication operation and the symbol + indicates an addition operation. The green, blue and red rectangles represent the forget gate, input gate and output gate respectively. [9]

- The forget gate decides what information to forget and what information to keep from the previous time step, by passing the current input data and the previous hidden state information through a sigmoid function getting a value between 0 and 1. The closer the value to 1 means to keep and the closer to 0 means to forget. This result gets pointwise multiplied with the previous cell state.
- The input gate decides which parts of the cell state should be updated by passing the current input data and the previous hidden state information firstly through a sigmoid function. While getting a value between 0 and 1 to check the importance of the of this combination and secondly through a tanh function getting a value between -1 and 1 to help regulate the network. Both the outputs of the sigmoid and tanh functions are multiplied and the result is then added to the cell state to be updated, this is the next cell state.
- The output gate decides the next hidden state by firstly passing the calculated next cell state through a tanh function, and secondly passing the

current input data and the previous hidden state information through a sigmoid function, both results gets pointwise multiplied to produce the next hidden state.

2.4. Generative Adversarial Networks

The generative adversarial network (GAN) was proposed by Goodfellow in 2014 [22]. It is a type of deep neural network that combines two main networks: a generative model called generator **G** and a discriminative model called discriminator **D** as shown in figure 2.6. Both generator and discriminator compete against each other in a zero-sum game and are concurrently trained to capture the data distribution.

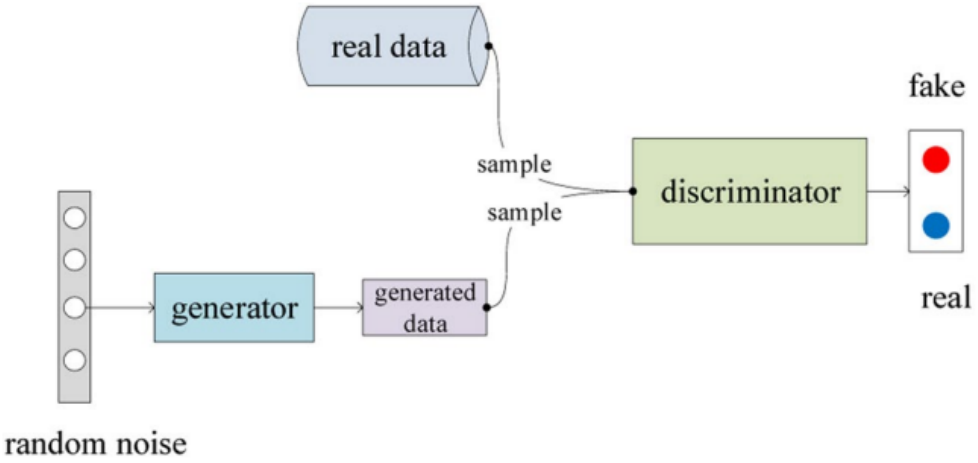

Figure 2.6.: The basic structure of the generative adversarial network model that is made of the two networks namely generator and discriminator. [17]

The generator takes in a random uniform or a Gaussian noise (z) as a latent space and tries to learn to generate synthetic data as an output $G(z)$, in our case EMG signals. The discriminator aims to predict the probability whether its input is from the training set (x) or from the generator $G(z)$ by assigning a low score to fake signals and a high score to real signals $D : y \rightarrow [0, 1]$. The discriminator receives as input either (x) or $G(z)$, where (x) are real signals and $G(z)$ are the generator output or fake signals. As soon as the discriminator learns to better differentiate between real and fake signals, the generator is forced to generate more realistic signals that are more closely to match the

distribution. This adversarial training of the two networks is called the minimax game and is described in the formula bellow [22]. The best performance of artificial data generation is reached when the discriminator can no longer distinguish between real and generated signals and deliver a confident score of 0.5, herewith, the artificial signals seams realistic to the discriminator, so that it is no longer able to classify artificial signals with high confident score.

$$\min_G \max_D V(D,G) = E_{x\sim p_{\text{data}}(x)}[\log D(x)] + E_{z\sim p_z(z)}[\log(1 - D(G(z)))]. \quad (2.7)$$

Taking a closer look at the minimax formula, it consists of two main terms: the first term $E_{x\sim p_{data}(x)}[logD(x)]$ represents the discriminators prediction on the real data. The second term $E_{z\sim p_z(z)}[log(1 - D(G(z)))]$ represents the discriminators prediction on the fake data. The first term is the expectation $E_{x\sim p_{data}(x)}$ of the Log of the discriminator output, when the input is from the real data distribution (x), this gives the average of the discriminators predictions when passing multiple samples of the real data to it. The Log transformation aims to scale the numbers. The discriminators aims that the prediction of the first term to be a large number, because this represents high confidence that the real samples are real. The generator is not involved in this process. The second term is the expectation $E_{z\sim p_z(z)}$ of the Log of the discriminator prediction of the fake samples. $D(G(z))$ is the discriminator output, when passing in fake samples to it. The discriminator wants this value to be as small as possible, then this represents that the discriminator has high confidence that fake samples are fake, since its task is to correctly evaluate the generator's sample with respect to real data. The generator on the other hand wants to fool the discriminator and aims for this value to be as large as possible. The generator uses its loss as the penalty if it fails to fool the discriminator. When the generator outputs excellent samples that are almost similar to the original data the discriminator gets worse at distinguishing real data from generated data [16], In this case, the discriminator thinks that the fake samples are real with high confidence. This is the mechanism of the adversarial game, where both networks play the zero-sum game during the training process until they converge [17]. By adding 1 minus in the second term of the formula, this means that the discriminator no longer needs to maximize the first term and minimize the second one, but aims to maximize the whole function $\max_D V(D,G)$ and the generator wants to do the opposite by minimizing the entire term $\min_G \max_D V(D,G)$.

CHAPTER 3

Methods

3.1. Cube Game

The cube game described in this section was developed in Unity. Unity is a powerful cross-platform, 3D/2D game engine and a user friendly development environment, that gives the user an access to all the tools needed for development. Unity supports variety of console, mobile, desktop and virtual reality platforms including the oculus quest headset. It uses C# as a primary programming language used for the engine to handle code and logic with many classes and APIs that are available for game creation [14].
The cube game is one of the main tasks in this work, as it takes EMG data collection to another level using a delay reaching task. Its purpose is to help collecting data and record EMG signals produced by the upper-arm while the subject plays the game in a 3 dimensional virtual environment using the oculus quest headset and its controller. First, the subject wears the oculus quest headset and holds the right controller with the right hand. Once the game starts, the player sees a yellow ball attached to his/her hand in the 3D environment, which can be freely moved with the quest controller. The player sees also 16 white boxes in a circle form and a green box in the middle of this circle as shown in figure 3.2. The radius of the circle is set to 20 cm for all subjects with the following position angles: 0°, 22.5°, 45°, 67.5°, 90°, 112.5°, 135°, 157.5°, 180°, 202.5°, 225°, 247.5°, 270°, 292.5°, 315°, 337.5° with a constant distance of 22.5° overall as shown in figure 3.1 in a X-Y dimension.

The player can start the game by touching the center box followed by a 10

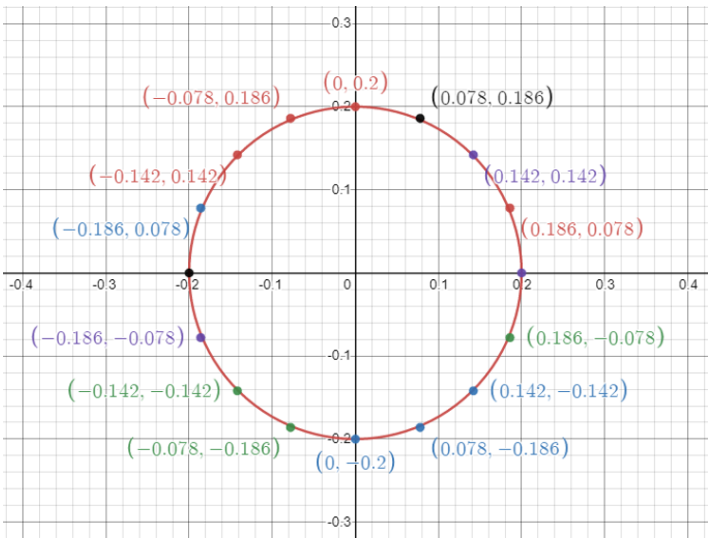

Figure 3.1.: A graphical representation of the calculated PI circle including the cubes positions with a radius of 20 cm.

seconds waiting phase to avoid any positioning errors at the start. After this waiting phase, the game starts and one of the 16 cubes randomly changes its color to green. The player should then move his/her arm to touch the colored cube with the virtual hand. The player has to stay at this position for 2 seconds, in the first second the green cube changes its color to white and after the following second the central cube changes to green, indicating that the player should move the arm to the center to play another round. In case the player is touching a cube and returns the arm to the center before the middle cube is turned green (before 2 seconds), this counts as an error. The yellow ball's color changes from yellow to red. This information including the timing of the error movement is saved. However, the player should keep playing and this is not a reason to stop the game and data recording, after each error and when the player is in the center, the ball's color changes to yellow and the game continues.

The player is able to choose the radius of the position of the cubes in the script and the number of repetition rounds. For example, 5 repetition rounds means that each cube of the 16 cubes will change its color to green 5 times to be touched, which is the case of the data collection in this work. The choice of which cube changes its color is completely random and has no order, meaning that each game has a new colored cubes order. The player can also choose the duration of the arm movement by choosing the waiting time in each box as well as the waiting time in the center.

Figure 3.2.: The cube game in action where the player is making a move to touch the green box with the controller or the virtual hand using the yellow ball. Overall there are 16 boxes arranged in a circular form with 20 cm radius and an additional box in the middle of the circle.

At the end of the game, when the player is at the center, the yellow ball changes color to red, implying that the number of repetition rounds is completed and the player must stop playing as well as the data collection at this point. After the game is over, the following information are saved in the quest headset internal storage as .csv format with the following file names:

- Start and end time of the game as Unix time. (Start-End-Unix-Time.csv)
- Colored boxes order (repetition rounds). (Colored-cubes.csv)
- Yellow ball or player hand position during the game. (Pointer-position.csv)
- Towards movement timestamp in seconds. (Towards-movement-duration.csv)
- Backward movement timestamp in seconds. (Backward-movement-duration.csv)
- Error signals timestamp in seconds. (Error-Signals.csv)

3.2. Oculus Quest

The oculus quest (Oculus VR) is a virtual reality headset that allows the users to dive into the games and media without any external sensors or wires. It is a standalone device with two 6 degrees of freedom (DOF) controllers, and it can run games and software wirelessly under android operating system without the need of computer or game console [11].

3.3. Delsys Trigno

Delsys Trigno is a device that makes EMG signal detection reliable and easy. It transmits signals from Trigno Avanti sensors to a receiving base station using a time-synchronized wireless protocol that minimizes data latency between sensors [4]. It is connected with a software called EMGworks Acquisition, which is a tool for advanced research and has a user-friendly interface for data acquisition and analysis [5]. The muscle activity is recorded at 1926 Hz in range of 11 mV and a bandwidth of 10-850 Hz using the Trigno Wireless EMG System (Delsys Inc., Boston, MA, USA) with one quadro and one due sensors, that consists of four and two electrodes respectively. Figure 3.3 shows the Trigno system base station used for data recording and figure 3.4 shows the EMGworks application during data collection.

Figure 3.3.: Trigno system base station for sensor recharging and communication. [4]

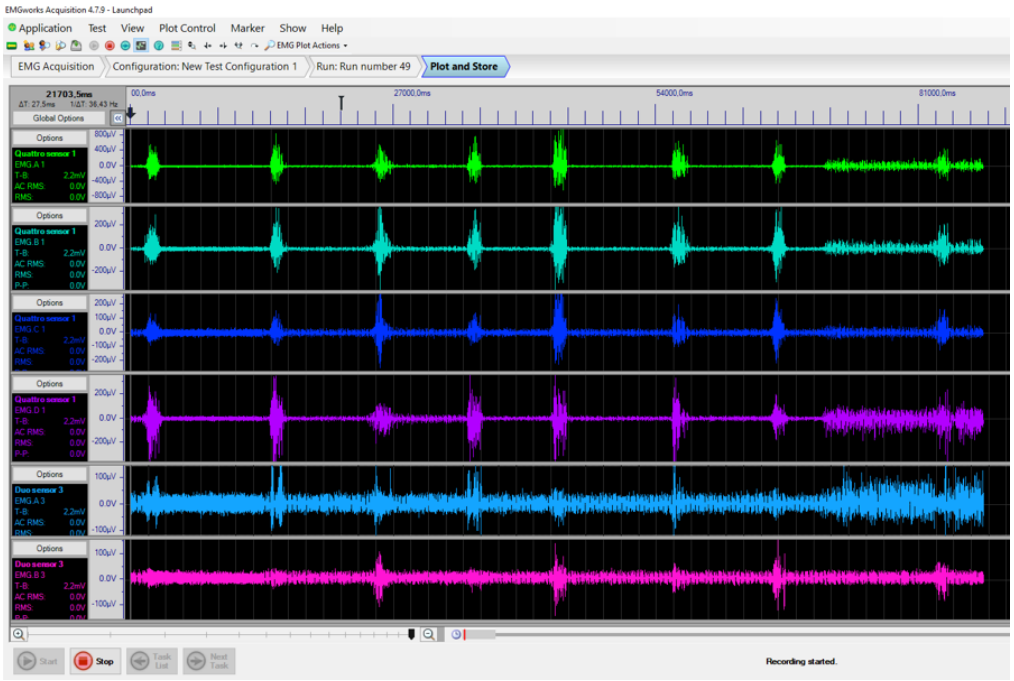

Figure 3.4.: EMGworks Acquisition software in action. The signals corresponds to the following muscles from the top to the bottom: Anterior deltoid, Medial deltoid, Posterior deltoid, Coracobrachialis, Biceps and Triceps.

Part II.

Experiments and Analysis

CHAPTER 4

Implementation and results

4.1. Data collection and preparation

The data-set was collected from six different muscles of the upper arm using the delsys trigno hardware [4]. Therefore a quattro EMG sensor was used to measure signals on the following muscles: anterior deltoid, medial deltoid, posterior deltoid and coracobrachialis, and a duo EMG sensor was used to measure signals on the biceps and triceps. The data recording was accomplished in the same process, and the EMG sensors were first cleaned and connected with the trigno hardware and EMGworks application. After cleaning the arm's skin, each sensor is then placed on the skin at specific muscle of the participant as shown in figure 4.1. In the following, the subject gets an overview and explanation about the functionality of the data collection process and the cube game as described in the previous section. Before the data is recorded, the EMG sensors are tested whether they are correctly placed on the muscle by having a closer look at the EMG signals on the EMGworks application. Afterwards, the baseline signals of all muscles is recorded for 30 seconds while the player's arm is kept in a rest position without any movement. Thereafter, the player wears the quest headset and the game starts as well as the EMG data recording. The subject plays for five repetition rounds, this corresponding to 6 to 8 minutes in total. After each round the subject relaxes his/her arm for few minutes to avoid muscle fatigue and then starts another round. In the best scenario, the subject plays four rounds which corresponds to 25 to 30 minutes of data recording for each subject. Overall, there were ten participants (three

women and seven men) with the overall following criteria: average age: 27, average weight: 80 kg and average height: 1.73 meter. Informations of each subject are represented in table 4.1 including the number of rounds that have been played and the number of repetition rounds chosen for each subject.

Table 4.1.: Characteristics of each subject and information of the played games.

Subject	**Age**	**Height(cm)**	**Weight(kg)**	**Payed rounds**	**Repetitions**
Person 1	28	172	95	4	5
Person 2	26	168	62	2	5,10
Person 3	30	175	78	3	5
Person 4	28	182	72	2	5
Person 5	22	175	70	2	10
Person 6	29	153	54	4	5
Person 7	22	183	70	4	5
Person 8	33	184	102	3	5
Person 9	23	187	130	4	5
Person 10	26	160	68	4	5

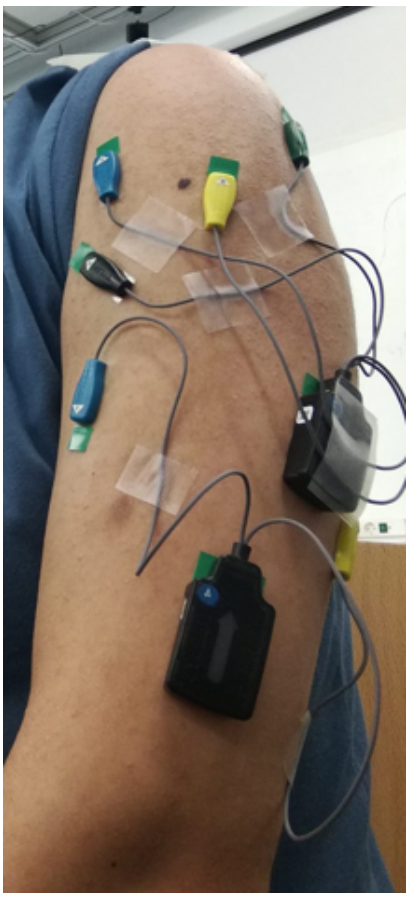

Figure 4.1.: EMG sensors placed on the arm during data collection. The quattro EMG sensors measures activity of the following muscles: anterior deltoid, medial deltoid, posterior deltoid and coracobrachialis. The duo EMG sensors measure activities of the biceps and triceps.

4.1.1. Pre-processing

The raw EMG data collected (see figure 4.2) need to be pre-processed to reduce the complexity and to extract the main features. In addition, EMG signals noise need to be reduced for efficient network training. This includes several signal processing steps that are described as follows: Baseline subtraction, find possible outliers, band-pass filtering, notch 50 Hz filtering, rectification, root mean square and data normalization.

The baseline subtraction or correction is an important pre-processing technique used to separate true EMG signals from interference effects and to re-

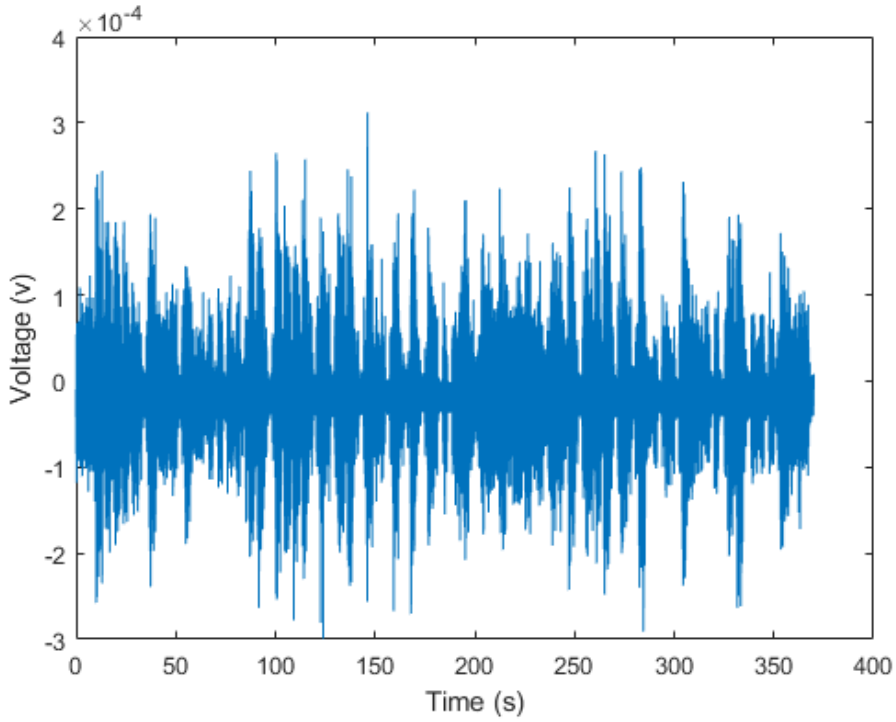

Figure 4.2.: Raw EMG signal of the channel 1 measured by the first electrode of the quattro sensor, which corresponds to the anterior deltoid muscle.

move background effects for each muscle. This is done by calculating the average value of the collected baseline signal and subtracting it from all EMG signals of each particular subject. During the game and data recording, some problems could occur and lead to a noisy signal in the data. For example, if the player makes a backward movement and touches the char with the back arm, this can interferes with the sensors and possible outliers could arise, these should be found in this processing step.

As shown in figure 4.3 the Fast Fourier Transform (FFT) is used to convert the EMG signals from the time domain into the frequency domain in order to visualize the signal in the frequency domain. Considering this paper [27] we can assume to expect the EMG activity in the area between approximately 10 Hz and 500 Hz. Thus, a band-pass filter between 10 Hz and 500 Hz is implemented as well as a notch filter at 50 Hz to remove the undesired frequencies from the EMG signals.

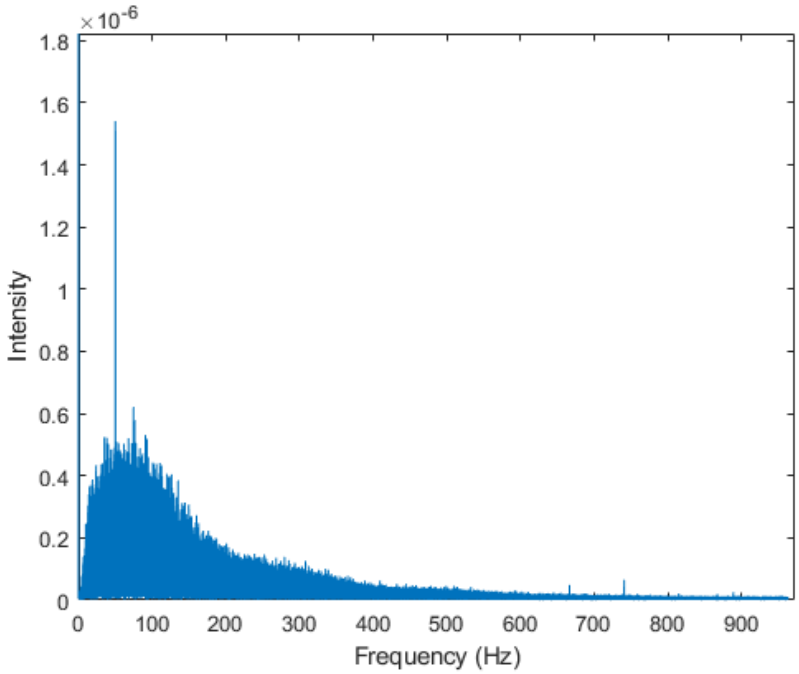

Figure 4.3.: Frequency domain of the channel 1 measured on the anterior deltoid muscle.

So that raw EMG signals have positive and negative values, a rectification technique is used as a positive polarity translation. There are multiple rectification methods, however, the full-wave rectification method is used in this work for all signals to convert negative to positive values of the EMG signals in order to avoid data deletion, which is the case by the half-wave rectification. In the following root-mean-square (RMS) [24] is implemented to provide better insights on the amplitude of the EMG signal that gives a measure of the signal power, while producing a waveform that is easily analyzable. RMS envelope of the EMG signal is calculated using a sliding window of 1000 to obtain the envelope signal according to the following equation:

$$\text{RMS(t)} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} w_i(t)x_i^2(t)} \tag{4.1}$$

where t is the number of samples the analysis window slides, N is the window length, $x_i(t)$ is the i^{th} sample of the signal centered around t as seen through the window $w_i(t)$ [24]. Figure 4.4 shows a raw signal compared with

a filtered and rectified EMG signal and its calculated RMS envelope. Figure 4.5 shows an example of the finale processed EMG signal.

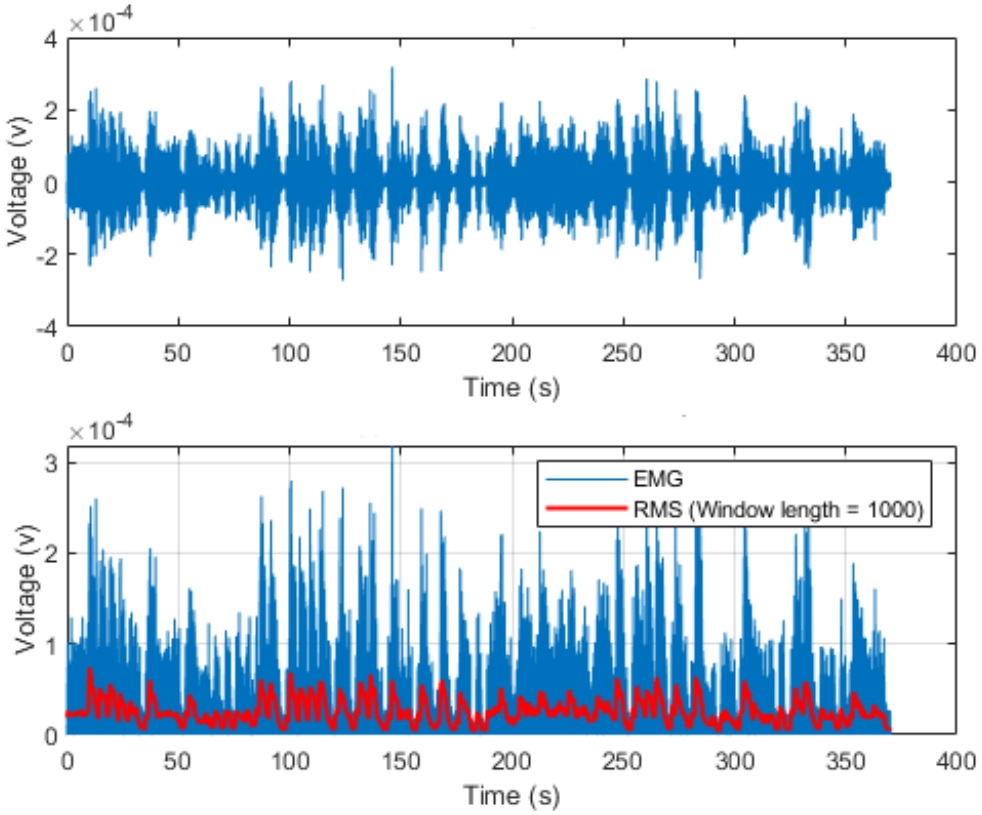

Figure 4.4.: Comparison between the original EMG signal at the top and its filtered variation at the bottom with the calculated RMS signal in red.

Each EMG signal of the data-set is normalized by scaling its values between 0 and 1 in order to overcome the difference in the channel's values between the subjects and to make the training less sensitive to the scale of features also in addition to assure that all data looks and reads the same way across all channels. The following equation was used for the normalization step:

$$x_{\text{normalized}} = \frac{(x - x_{\text{min}})}{x_{\text{max}} - x_{\text{min}}} \tag{4.2}$$

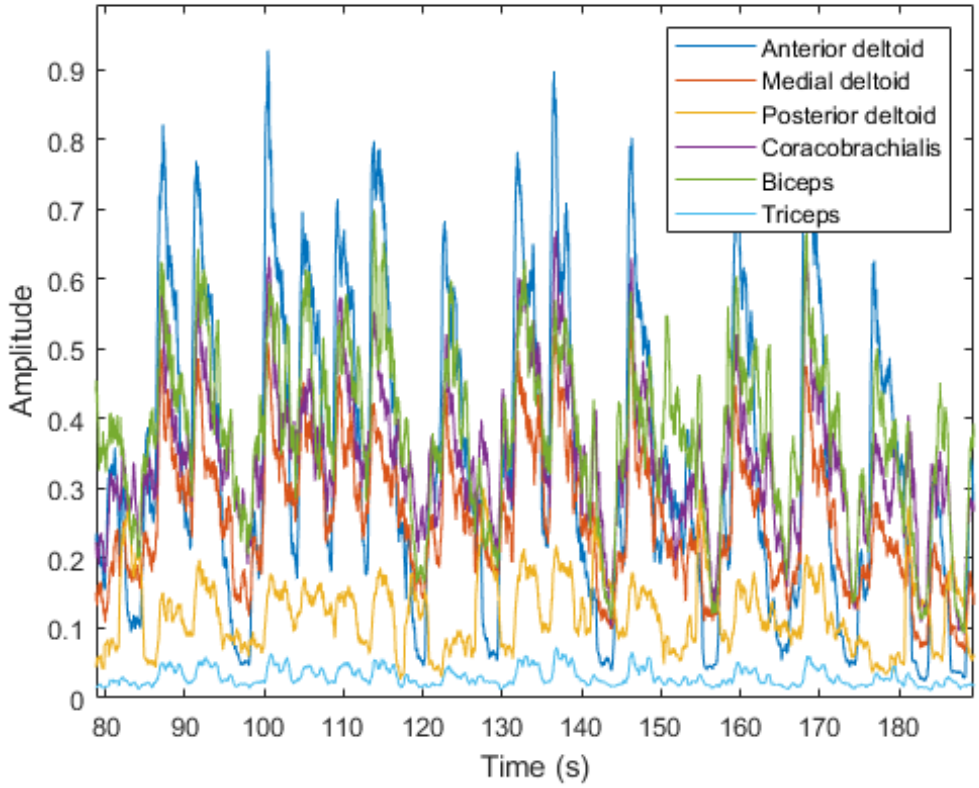

Figure 4.5.: A part of processed EMG signal of the third round of subject seven.

4.2. GAN Models

In this work two different GANs architectures were developed for generating artificial EMG data. The first architecture is a CNN-CNN GAN that is made mainly of convolutional networks for the generator and discriminator. The second architecture is the exact same architecture used in CNN-CNN GAN with the difference that it includes an LSTM layer as input layer of the generator, meaning that both GANs have the same discriminator. The network architecture for the CNN generator and LSTM-CNN generator are shown in table 4.2 and 4.3 respectively. The discriminator architecture for both GANs if represented in table 4.4. The generator in both GANs takes in a random vector as input, which is made of 1000 noise points and generates an EMG sequence with a length that was set to 8640 as output. The choice of 8640 data points is based on the tuning process as this number must be calculated based on the network length and filter size in each layer as well as the stride value by the convolutional layer. On the other hand, the discriminator takes in real data and gives a score as an output to classify whether its input is real or fake. The sigmoid function is used in the discriminator output to scale its confidence score in a range between 0 and 1. In both **D** and **G**, Leaky ReLU activation function is used to allow the pass of a small gradient signal for negative values to avoid the Dying ReLu problem and sparse gradients where the network is not receiving strong enough signals to tune its weights.

4.2.1. Generator architectures

The CNN generator's input is a fully connected layer that takes a random vector (z) followed by a reshape layer to prepare the input of the transposed convolutional layer. Six transposed convolutional layers are used with a Leaky ReLU activation function followed by a batch-normalization layer for each. The output of the six transposed convolutional layers passes through a reshape layer without any changes, and it is then fed into a convolutional layer with a sigmoid activation function to generate the final output. The LSTM-CNN generator has the exact same architecture with one LSTM layer that consists of 1000 LSTM cells as the input layer of the network. This takes the random vector (z) and passes its output to the fully connected layer followed by a reshape layer to prepare the input of the transposed convolutional layer. For both generators the stride=(2,1) and the padding='same' was used in all transposed convolutional layers.

Table 4.2.: CNN Generator architecture.

Layer/Filter size	Act/Norm	Output shape
Latent vector	-/-	1000
Dense 1000	LReLU	58320
Reshape	-/-	135 x 1 x 432
Conv2DTranspose/216	LReLU/0.2	270 x 1 x 216
Conv2DTranspose/108	LReLU/0.4	540 x 1 x 108
Conv2DTranspose/54	LReLU/0.6	1080 x 1 x 54
Conv2DTranspose/27	LReLU/0.2	2160 x 1 x 27
Conv2DTranspose/13	LReLU/0.4	4320 x 1 x 13
Conv2DTranspose/6	LReLU/0.6	8640 x 1 x 6
Reshape	-/-	6 x 8640 x 1
Conv2D/1	Sigmoid/-	6 x 8640 x 1

Table 4.3.: LSTM-CNN Generator architecture.

Layer/Filter size	Act/Norm	Output shape
Latent vector	-/-	1000
LSTM	-/-	1000
Dense 1000	LReLU	58320
Reshape	-	135 x 1 x 432
Conv2DTranspose/216	LReLU/0.2	270 x 1 x 216
Conv2DTranspose/108	LReLU/0.4	540 x 1 x 108
Conv2DTranspose/54	LReLU/0.6	1080 x 1 x 54
Conv2DTranspose/27	LReLU/0.2	2160 x 1 x 27
Conv2DTranspose/13	LReLU/0.4	4320 x 1 x 13
Conv2DTranspose/6	LReLU/0.6	8640 x 1 x 6
Reshape	-/-	6 x 8640 x 1
Conv2D/1	Sigmoid/-	6 x 8640 x 1

4.2.2. Discriminator architecture

The discriminator of both GANs consists of five convolution layers with Leaky ReLU activation function and a dropout layer, these are followed by a flatten layer that collapses the spatial dimensions of the input into a simple vector output. This vector is fed into a fully connected layer with a sigmoid activation function for the classification. The stride=2 and the padding='same' was used in all convolutional layers of the discriminator.

Table 4.4.: Discriminator architecture.

Layer/Filter size	Act/Dropout	Output shape
Input signal	-/-	6 x 8640 x 1
Convolutional 2D/8	LReLU/0.1	3 x 4320 x 8
Convolutional 2D/16	LReLU/0.1	2, 2160, 16
Convolutional 2D/32	LReLU/0.1	1, 1080, 32
Convolutional 2D/64	LReLU/0.1	1, 540, 64
Convolutional 2D/128	LReLU/0.1	1, 270, 128
Flatten	-/-	34560
Dense 1	Sigmoid	1

All networks proposed in this work are developed by TensorFlow and Keras API, which is included with in Tensorflow and is used to simplify the creation of neural networks [12]. TensorFlow is an end-to-end open source platform that helps users to create, develop and train machine learning models. Its applications can be run on a variety of targets such as local machines, clusters in the cloud, android devices, CPUs or GPUs. It uses python as a programming language for building applications and networks then creates a data-flow graph, which describes how data moves through processes and finally executes those applications in high-performance C++ [12].

Generally, the GAN network consists of two main networks, a generator and a discriminator. The discriminator is a straightforward classifier that learns to differentiate between real EMG signals and noise or fake EMG signals. The generator is a network that takes in random vectors and learns to generate fake EMG signals that look like real EMG signals to fool the discriminator. Both networks try to win game and good results are reached when the discriminator loss is at 0.5. This means that it is no longer time able to distinguish real from fake signals. The training in this work is done by taking a batch size number of data from the data-set and the same number is taken to create latent spaces of random noises. These noises are fed into the generator to generate some results. In the first training steps, this result is nothing else but noise since the discriminator is not trained yet. Both signals (real EMGs and generator output) build a training set for the discriminator with correct labels, meaning that real EMG signals are labeled with 1 and the generator output signals (fake signals) are labeled with 0. At this point the learning process of the discriminator is activated and begins to learn to classify real from fake signals. Moreover, a 'train on batch' function from keras is used for training which allows to run a single gradient update on a single batch of data and it returns two values: the discriminator loss and accuracy. In the following, the training of the discriminator is deactivated to longer update its weights but to be used as a classifier. The next

step is to launch the training over the whole GAN with the discriminator being deactivated. In other words, the generator starts to train to generate EMG signals to fool the non-well trained discriminator. The input to the generator is 1000 random points with the size of the batch size, and the output of the generator is then labeled with 1 and is fed into the discriminator. This process is repeated while the discriminator learning process is activated and deactivated each epoch, Over time and early in learning process, the discriminator is able to reject the generator results as they look clearly different from the training data, even if they are labeled as real signals. In terms of loss functions, the generator gets the error of the classification on its output and can not affect the learning of the discriminator directly, as this get activated each time and learns to become a better classifier while minimizing the loss function of the GAN. Thus, the generator is forced to learn to generate better signals to maximize the loss function of the GAN, as it uses the loss as the penalty if it fails to fool the discriminator. When the generator outputs excellent signals that are almost similar to the original data the discriminator gets worse at distinguishing real from generated data and the generator is finally able to generate good artificial signals.

The metrics that are used for evaluating the quality of the real and artificial EMG signals are: Mean Absolute Error (MAE), Root Mean Square Error (RMSE) and Euclidean Distance (ED), as they were used to evaluate GAN networks generating synthetic ECG, EEG and EMG signals in this paper [16]. The objective is to generate artificial signals with the distance or dissimilarity closest to zero in comparison to the EMG signals in the data distribution.

Mean Absolute Error (MAE): calculates the average of the absolute differences between the original (O) and artificial signals (A), it is calculated as:

$$\text{MAE} = (\frac{1}{N}) \sum_{i=1}^{N} |O_i - A_i| \tag{4.3}$$

Root Mean Square Error (RMSE): measures the stability between the original data (O) and generated artificial data (A), and it is calculated as:

$$\text{RMSE} = \sqrt{(\frac{1}{N}) \sum_{i=1}^{N} (O_i - A_i)^2} \tag{4.4}$$

Euclidean Distance (ED): measures the shortest distance between two points in an Euclidean space, which is the length of a line segment between the two

points of the original O and the artificial signal A. It is calculated as follows:

$$\mathrm{ED}(O, A) = \sqrt{\sum_{i=1}^{N}(O_i - A_i)^2} \tag{4.5}$$

4.3. Results

This section presents the results of the CNN and LSTM-CNN generators of the GAN models. It shows a comparison between the original and artificial EMG signals in a graphical form and shows the evaluation metrics in a table and graphical form. Tables 4.5, 4.6, 4.7 show the MAE, RMSE and Euclidean distance comparison between the original and generated signal using our proposed networks. Figures 4.6, 4.7, 4.8 depict a graphical form of the MAE, RMSE and Euclidean distance comparison respectively. ALL graphics of the evaluation have a similar shape, where the channel four has the largest value, however, both generators deliver signals with very close evaluation values to the original signals. For example, the MAE comparison shows an average of 0.067 dissimilarity for the original signals and 0.069 for the CNN generator's signals compared to original signals and 0.072 for the LSTM-CNN generator's signals. Knowing that a MAE comparison between the original signals and noise is close to 1, the results of the evaluation are advantageous. The same notice applies to the RMSE and Euclidean distance comparison as shown in the tables and the graphical representation of the evaluation metrics. Figures 4.9 and 4.10 show an original signal on the left and an artificial signal on the right generated by the LSTM-CNN and CNN generators. They show a comparison of approximately 1400 samples of the signals and they give an overview and a general look of the signals including the generated peaks and resting phases across all channels. Figures 4.11 and 4.12 give a closer look at the nature of the original and generated channels[1] at one or two peaks. From these figures we can notice that the LSTM-CNN signal includes less noise than the CNN signal, therewith, the LSTM-CNN generated channels look similar to the nature of the original channel's width, unlike the signal generated by the CNN generator, which looks slightly different in width due to noise. Figures 4.13 and 4.14 gives a closer look at one single peak of channel four generated by both networks and the original channel. Both GANs are able to generate multi-channel artificial EMG signals in the right order that looks similar to the original signals respecting the highest and lowest channel values as shown in figures 4.9 and 4.10. After each request of both saved generators, they generate a different artificial EMG signal with different values for each channel, while the

[1]The channels names of all channels represented in this section can be seen in appendix.

global structure of the signals remains almost identical. The proposed LSTM-CNN generator seems to capture the data distribution more quickly in terms of epochs than the CNN generator, which takes longer to train and deliver good results in terms of evaluation. Another difference between both generators is the smoothness of the generated signals, as shown in figures 4.11 and 4.12 the results of the LSTM-CNN generator contains less noise overall than the ones of the CNN generator. Both networks respects the resting phases in the signals and the peak duration but LSTM-CNN generator is able to fire also lower channels during a motion, where the CNN generator seems to fire lower channels during resting phases in some cases, which is not the case in the data distribution. The single channel shown in figure 4.13 does not fit exactly to the original signal as in figure 4.14, a perfect match of the channels is also not expected due to the nature of the data distribution, where the peaks do not occur at the same time across all data.

Table 4.5.: MAE comparison between original and artificial signals.

channel	Original	CNN	CNN-LSTM
1	0.045	0.050	0.058
2	0.071	0.095	0.071
3	0.034	0.033	0.039
4	0.185	0.159	0.175
5	0.061	0.059	0.070
6	0.006	0.019	0.022
avg	**0.067**	**0.069**	**0.072**

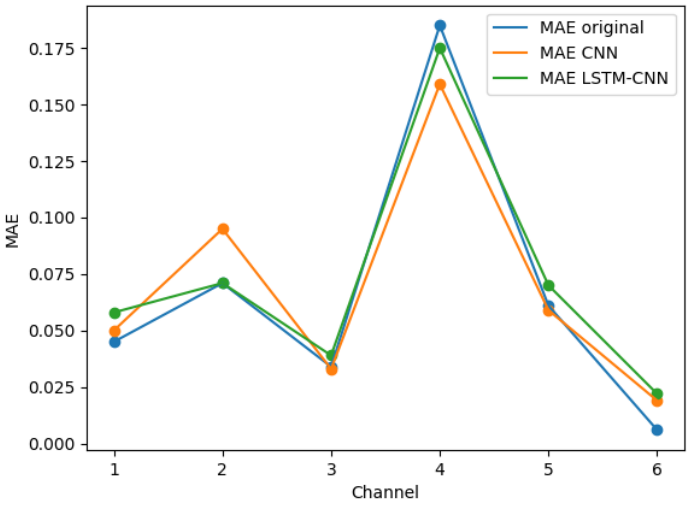

Figure 4.6.: Graphical representation of the MAE comparison.

Table 4.6.: RMSE comparison between original and artificial signals.

channel	**Original**	**CNN**	**CNN-LSTM**
1	0.003	0.004	0.005
2	0.008	0.013	0.008
3	0.001	0.002	0.002
4	0.059	0.044	0.052
5	0.006	0.005	0.007
6	0.081e-3	0.524e-3	0.998e-3
avg	**0.012**	**0.011**	**0.012**

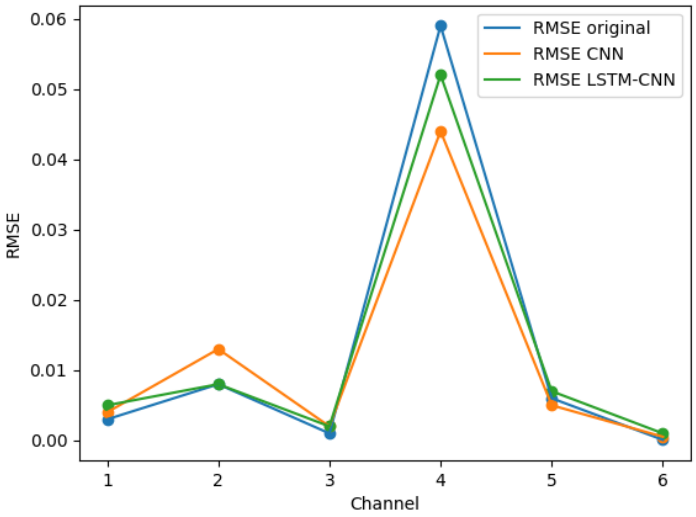

Figure 4.7.: Graphical representation of the RMSE comparison.

Table 4.7.: ED comparison between original and artificial signals.

channel	Original	CNN	CNN-LSTM
1	5.587	6.185	6.917
2	8.598	10.916	8.543
3	3.982	4.233	4.506
4	22.75	19.504	21.358
5	7.201	7.125	8.187
6	0.840	2.128	2.937
avg	**8.159**	**8.348**	**8.741**

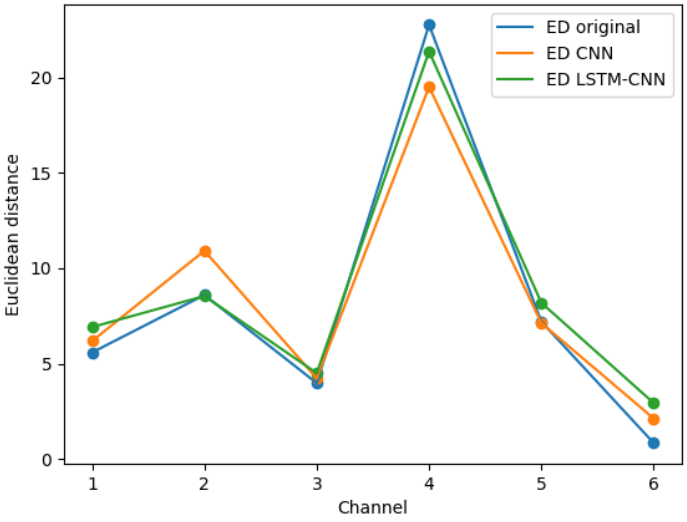

Figure 4.8.: Graphical representation of the Euclidean distance comparison.

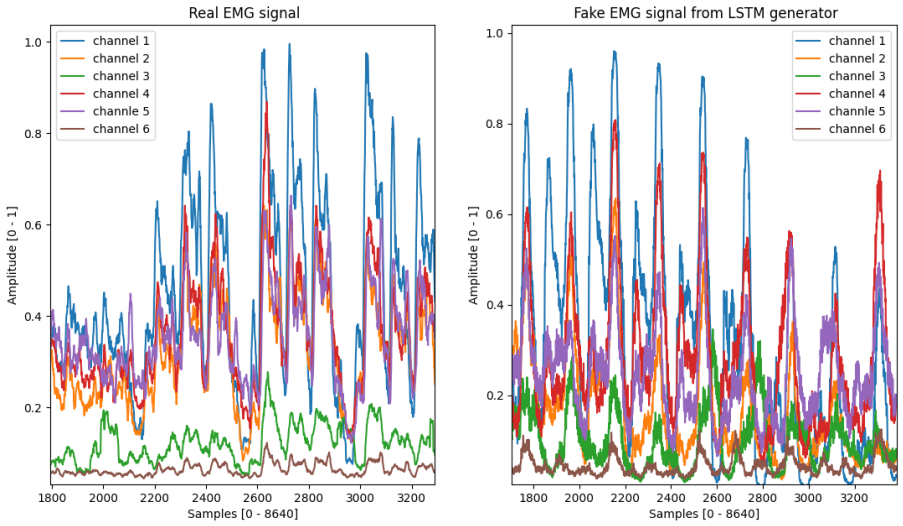

Figure 4.9.: Comparison of 1400 samples in the middle of the signal for all channels between the original signal on the left and the artificial signal generated by the LSTM-CNN generator on the right.The original signal is from the first round of the subject number seven.

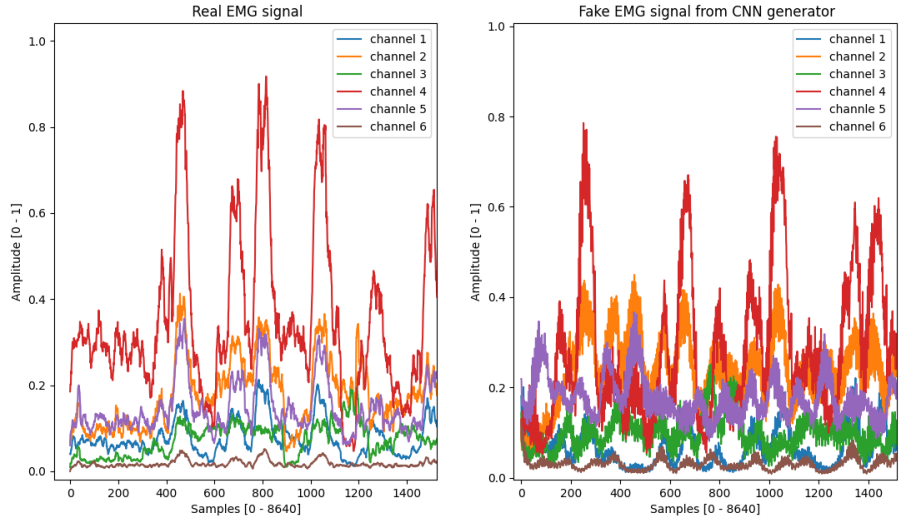

Figure 4.10.: Comparison of 1400 samples in the middle of the signal for all channels between the original signal on the left and the artificial signal generated by the CNN generator on the right. The original signal is from the first round of the subject number one.

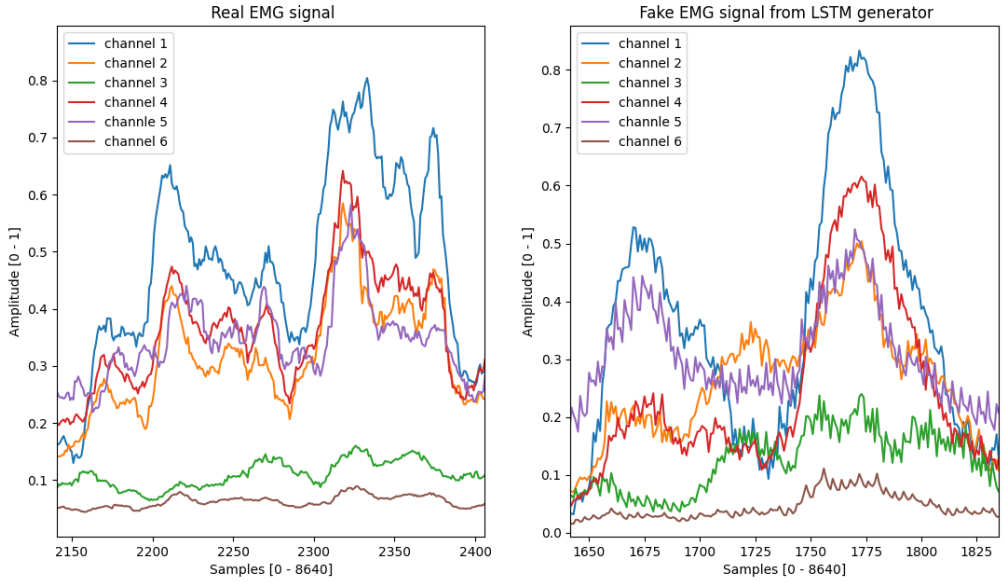

Figure 4.11.: Comparison of one or two peaks for all channels between the original artificial signal generated by the LSTM-CNN generator.

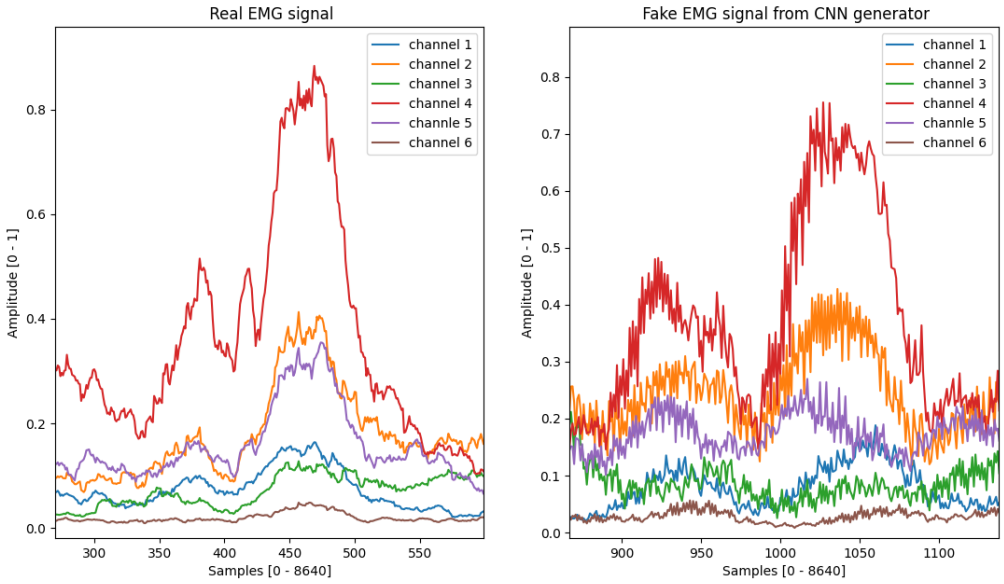

Figure 4.12.: Comparison of one or two peaks for all channels between the original artificial signal generated by the CNN generator.

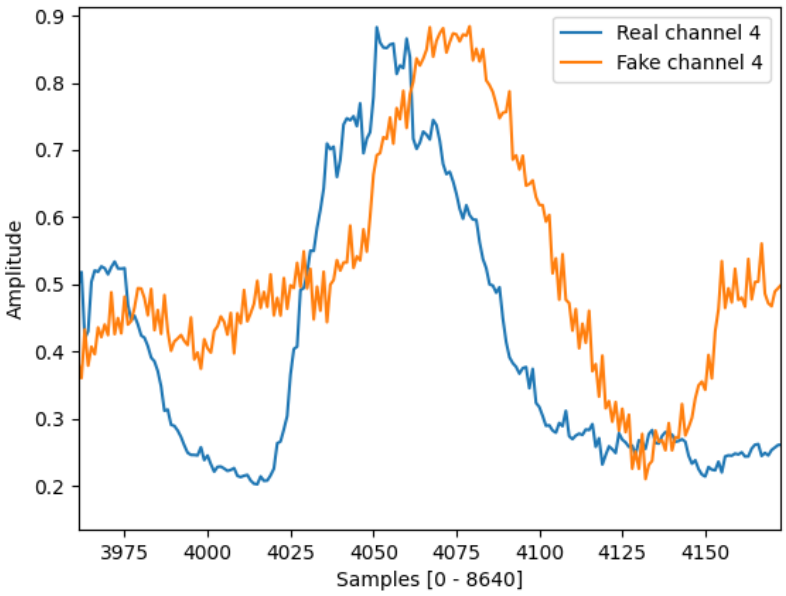

Figure 4.13.: Comparison of one peaks for channel 4 between the original and artificial signal generated by the LSTM-CNN generator.

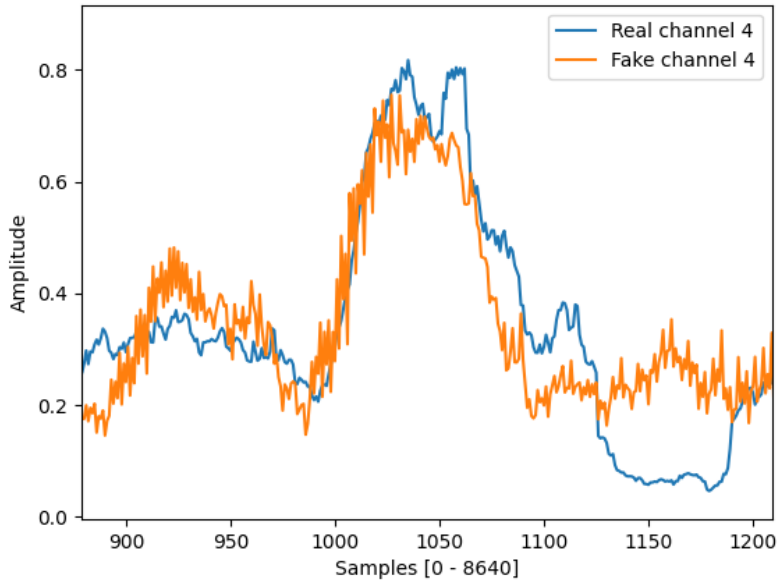

Figure 4.14.: Comparison of one peaks for channel 4 between the original and artificial signal generated by the CNN generator.

CHAPTER 5

Discussion and conclusion

The proposed GAN networks was successfully able to generate artificial EMG signals with low dissimilarity to the original EMG signals and without losing the features of the existing data as shown in table 5.1. The table show the average of the evaluation metrics. When analyzing these values, we can conclude that the difference between the original and generated signals is very low in terms of evaluation. Even that the generated signals by the CNN generator contains more noise, they show lower values at the evaluation in comparison to the LSTM-CNN generator signals. The reason for the noise included in the CNN generator's output signals might be the manual hyperparameters tuning, which can be improved by using an automatic parameters search. On the other hand, we assume that the reason behind the smoothness of the LSTM-CNN generator signals must be its input layer that consists of LSTM cells, which helps the network to memorize the structure of the data distribution and stabilizes its output. As shown in the results section in the evaluation comparison part, the channel four has the highest values across all metrics in the original and artificial signals, as this channel represents the highest values in most data in the data distribution. Our both generator networks and the discriminator are based on a combination of two to tree different neural networks namely feed forward neural network, convolutional neural network and long-short term memory network to take advantage of the benefits of each network specialty. CNN network can be used to extract the time feature of data and recognize patterns, while LSTM can be used for predicting future values and remembering informations over time. During the hyperparameter tuning process, we

have highlighted that the latent space size influences the training of the generator network dramatically, which should gain a lot of interest and should be taken into consideration during GAN parameter tuning. Another observation worth mentioning is that the Conv1D convolutional layer should be implemented to produce single-shaped signals. For multi-shaped signals Conv2D convolutional layer is more stable and robust.

The artificial EMG signals can be used in different ways and in different field such as clinical/biomedical applications, modern human computer interaction and robotics. We assume that the proposed GAN networks can generate an artificial single-shaped biomedical signal such as ECG signals as they were successfully tested to generate a single-shaped EMG signal for one channel. In addition, the proposed GAN networks can help to generate and prepare training data for several interesting future works to meet specific needs, such as training other algorithms to discover muscle fatigue or to predict the direction the arm movement. The artificial EMG signals can also be used for the restoration of corrupted existing signals with the benefit of eliminating privacy concerns and the problem of data accessibility among the researchers.

Table 5.1.: Average of the evaluation metrics.

Metrics	**Original**	**CNN**	**CNN-LSTM**
MAE	0.067	0.069	0.072
RMSE	0.012	0.011	0.012
ED	8.159	8.348	8.741

Bibliography

[1] *CNN network*. https://medium.com/analytics-vidhya/image-classification-a-comparison-of-dnn-cnn-and-transfer-learning-approach-704535beca25. Accessed: 20-12-2021.

[2] *Convolutional layer*. https://www.andreaperlato.com/aipost/cnn-and-softmax/. Accessed: 20-12-2021.

[3] *convolutional layer*. https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/. Accessed: 20-12-2021.

[4] *Delsys*. https://delsys.com/downloads/USERSGUIDE/trigno/wireless-biofeedback-system.pdf. Accessed: 20-12-2021.

[5] *EMGworks*. https://delsys.com/support/software/. Accessed: 20-12-2021.

[6] *GAN*. https://www.analyticsvidhya.com/blog/2021/03/why-are-generative-adversarial-networksgans-so-famous-and-how-will-gans-be-in-the-future/. Accessed: 20-12-2021.

[7] *GAN*. https://www.electronicsforu.com/electronics-projects/human-machine-interface-through-electromyography. Accessed: 20-12-2021.

[8] *LSTM theory*. https://colah.github.io/posts/2015-08-Understanding-LSTMs/. Accessed: 20-12-2021.

[9] *LSTM theory*. https://medium.com/analytics-vidhya/lstms-explained-a-complete-technically-accurate-conceptual-guide-with-keras-2a650327e8f2. Accessed: 20-12-2021.

[10] *LSTM theory*. https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21. Accessed: 20-12-2021.

[11] *Oculus*. https://www.oculus.com/. Accessed: 20-12-2021.

[12] *Tensorflow*. https://www.tensorflow.org/. Accessed: 20-12-2021.

[13] *Transposed convolution*. https://medium.com/@marsxiang/convolutions-transposed-and-deconvolution-6430c358a5b6. Accessed: 20-12-2021.

[14] *Unity*. https://unity.com/. Accessed: 20-12-2021.

[15] Anirudha Ghosh, Abu Sufian, Farhana Sultana, Amlan Chakrabarti, and Debashis De. "Fundamental concepts of convolutional neural network". In: *Recent Trends and Advances in Artificial Intelligence and Internet of Things*. Springer, 2020, pp. 519–567.

[16] Debapriya Hazra and Yung-Cheol Byun. "SynSigGAN: Generative adversarial networks for synthetic biomedical signal generation". In: *Biology* 9.12 (2020), p. 441.

[17] Fei Zhu, Fei Ye, Yuchen Fu, Quan Liu, and Bairong Shen. "Electrocardiogram generation with a bidirectional LSTM-CNN generative adversarial network". In: *Scientific reports* 9.1 (2019), pp. 1–11.

[18] Mohit Sewak, Md Rezaul Karim, and Pradeep Pujari. *Practical convolutional neural networks: implement advanced deep learning models using Python*. Packt Publishing Ltd, 2018.

[19] Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do, and Kaori Togashi. "Convolutional neural networks: an overview and application in radiology". In: *Insights into imaging* 9.4 (2018), pp. 611–629.

[20] Josh Patterson and Adam Gibson. *Deep learning: A practitioner's approach*. " O'Reilly Media, Inc.", 2017.

[21] Alec Radford, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks". In: *arXiv preprint arXiv:1511.06434* (2015).

[22] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative adversarial nets". In: *Advances in neural information processing systems* 27 (2014).

[23] Daniel T Larose and Chantal D Larose. *Discovering knowledge in data: an introduction to data mining*. Vol. 4. John Wiley & Sons, 2014.

[24] Marcelo Caetano and Xavier Rodet. "Improved estimation of the amplitude envelope of time-domain signals using true envelope cepstral smoothing". In: *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2011, pp. 4244–4247.

[25] Simon S Haykin et al. *Neural networks and learning machines/Simon Haykin.* 2009.

[26] David Kriesel. "A brief introduction on neural networks". In: (2007).

[27] Anton Van Boxtel. "Optimal signal bandwidth for the recording of surface EMG activity of facial, jaw, oral, and neck muscles". In: *Psychophysiology* 38.1 (2001), pp. 22–34.

[28] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

Appendix

APPENDIX A

Detailed Descriptions

This work will be given to the supervisors in a USB-stick including the source code of the cube game, the script of the GAN models and the script of the signal processing part. In addition the supervisors will receive a technical report about the installation and setup of the integration of the cube game in the oculus quest headset.

The channels plotted in the result section are associated to the following arm muscles:

Table A.1.: Channel numbers and muscle names.

channel	**muscle**
1	Anterior deltoid
2	Medial deltoid
3	Posterior deltoid
4	Coracobrachialis
5	Biceps
6	Triceps

The software and hardware version used in this work are listed in the table bellow:

Table A.2.: Software and hardware versions.

Software/Hardware	**version**
Tensorflow	2.7.0
Python	3.7.12
Android Studio	2020.3.1
MATLAB	R2021a Update 5
EMGworks Acquisition	4.7.9
GPU	NVIDIA Tesla K80 GPU (Colab)
Delsys	Trigno Inc., Boston, MA, USA
Oculus	Quest