UNIVERSITÀ DI PISA

Computer Engineering

Intelligent Systems

ECG Estimation

Team members:

Francesco Martoccia

Salvatore Lombardi

ACADEMIC YEAR 2021/2022

Contents

1 - Introduction

Pulse Transit Time (PTT) is the time that pulse waves require to travel in blood vessels between two sites. It is an important indicator for many medical properties and recent research focused on its potential for blood pressure monitoring.
The aim of the project is to create models that accurately estimate key **ECG** parameters, including the **mean** and **standard deviation**, under different physiological conditions and activities.

In particular, the following specifications have been implemented:

- **Estimating ecg using neural networks:** design and develop two multi-layer perceptron (**MLPs**) artificial neural networks that estimate, respectively the **mean** and **standard deviation** of ecg of a person during three activities, based on sensor data.
  In addition, design and train two radial-based function networks (**RBFs**) that do the same thing as the previously developed MLPs.

- **Determining a person's activity using neural networks:** design and develop a multi-layer perceptron (**MLP**) that classifies a person's **activity** among 'sit', 'walk' and 'run' by taking as input a set of features and by returning the corresponding activity.

- **Fuzzy inference system:** design and develop a fuzzy inference system to classify a person's **activity** using the k most relevant features ($k \leq 5$) in the set of features used to train the classifier developed in the previous section.

- **Improve ecg estimation using convolutional neural networks:** develop a convolutional neural network (**CNN**) to estimate the value (mean/standard deviation) that achieved the **worst performance** using the **MLP** in the first section.
  The goal is to find the best CNN architecture along with the best hyperparameter values (filter size, number of filters, etc.).

- **Predict ecg value(s) using recurrent neural networks:** design and develop a recurrent neural network (**RNN**) that predicts one value of a person's ecg, based on part or all the provided signals.

The RNN takes these signals at time steps $(t-k, ..., t)$ as input along with the corresponding ecg values, and returns the ecg value at time step $t+1$. The goal is to find the best architecture of the RNN, along with the best hyperparameter values (size of the time window, etc.).

- **Multi-step forecasting of ecg values:** design and develop an **RNN** that predicts a set of consecutive future values of a person's ecg.
  The goal is to find the best architecture of the RNN, along with the best hyperparameter values.

2 - Dataset

The data collection involved 22 participants while they performed three different activities ('sit', 'walk', 'run'). The dataset contains 66 recordings. A recording contains 11 time series (signals), each representing a physiological signal for more than 40,000 heartbeats.
The data are in CSV (comma-separated-value) format. Each recording consists of 2 files, **sXX_YYYY_timeseries** and **sXX_YYYY_targets** ('s' means subject, 'XX' is the subject number (1-22) and 'YYYY' is an activity among 'sit', 'walk', and 'run'), whose structures are described as follows:

- Each file **sXX_YYYY_timeseries** is organized in columns. The first column contains a timestamp. Each of the other columns contains a time series corresponding to a physiological signal.
- Each file **sXX_YYYY_targets** contains the target time series. The first column contains a timestamp. The other contains the target time series (ECG signal).

3 - Neural Networks and Fuzzy Systems

In order to extract the features and correctly represent the information that the networks will receive as input, the following steps were taken: **feature extraction**, **data augmentation** and **feature selection**.

Feature Extraction

From the timeseries files, **13 features** (that are shown in table 3.1) were extracted for each of the 11 physiological signals, by using the ***extractFeatures*** function.
The feature extraction was performed with 3 different methods:

- **Without windows:** the chosen features were extracted from all the signals, on the entire set of signal samples that make them up. In this way we obtained 13 features for each signal, for a total of 143.
- **Contiguous windows:** the samples of each signals were partitioned in 4 contiguous windows, and feature extraction was performed on each of them. In this case we obtained 52 features for each signal, for a total of 572.
- **Overlapped windows:** the samples were partitioned in 7 overlapped windows, on which the chosen features were extracted. For each signal we got 91 features, for a total of 1001.

After these operations, **correlated features** were removed and **data normalization** was performed.
The former ensures that the feature set contains independent, non-redundant information which can lead to better model performance. This was done by means of the ***deleteCorrelatedFeatures*** function, that calculates the correlation matrix for the input features and identifies highly correlated features with a correlation coefficient greater than 0.90 (this threshold is configurable). It then keeps only one of the correlated features and discards the rest.
The latter allows bringing the feature values to the same scale, making them directly comparable and preserving the relative relationships between data points within each feature.

Time Domain Features	Frequency Domain Features
1. Mean	13. Power Spectral Density (PSD)
2. Standard Deviation	
3. Maximum Amplitude	
4. Minimum Amplitude	
5. Median	
6. Interquartile Range	
7. Skewness	
8. Kurtosis	
9. Mean Absolute Deviation (MAD)	
10. Root Mean Square (RMS)	
11. Peak-to-Peak Amplitude	
12. Waveform Length	

Table 3.1: Time and Frequency Domain Features

In particular, the adopted approach was **min-max normalization**, which rescales each feature such that the minimum value becomes 0, the maximum value becomes 1, and all other values are scaled proportionally within that range. The function used to do that was ***normalizeFeatures***.

Data Augmentation

After the feature extraction we performed data augmentation, that allows to artificially increase the size of the training dataset by creating new, slightly modified versions of existing data samples.
This technique introduces variability and randomness into the training dataset, making the model more **robust** and less sensitive to **small variations** in the input data. Furthermore, it can help prevent **overfitting** since the model is exposed to a wider range of input variations, allowing it to generalize better to unseen data.
The ***dataAugmentation*** function was used to do this. It takes as input the previously computed matrix, and for each sample in it, 50 more are generated. This is done by adding **random noise** to each of the features in the original matrix. After performing this procedure, the number of samples increased from **66** to **3366**.

Feature Selection

The set of extracted features should be reduced by selecting the most significant features to predict the output. In order to do that, we used the **sequential feature selection** (implemented by the ***sequentialfs*** MATLAB function), with an MLP as a criterion function that assesses the accuracy of each subset of features in estimating each target. In particular, we selected the best **10** features useful in estimating the **mean** and the **std** of the **ECG**.
To do this, we compared the criterion values obtained through the 3 different feature

extraction techniques. The one that yielded the best results was the one **without windows**.

3.1 Estimating ECG using Neural Networks

The content of this section includes:

1. Estimation of **mean** and **standard deviation** of a person's ECG with **MLPs**.
2. Estimation of **mean** and **standard deviation** of a person's ECG with **RBFs**.

3.1.1 Multi-layer Perceptron Networks

We used the Matlab ***fitnet*** function, which allows to create a fitting neural network with the specified **hidden layer size** and **training function**. In order to automate the search for the best training function, the **k fold cross validation** technique was adopted. The value of k was set to 5, so each training function was evaluated with 5 different training and test sets. For each training function, the **MSE** was calculated over the different iterations performed. The results obtained are as follows:

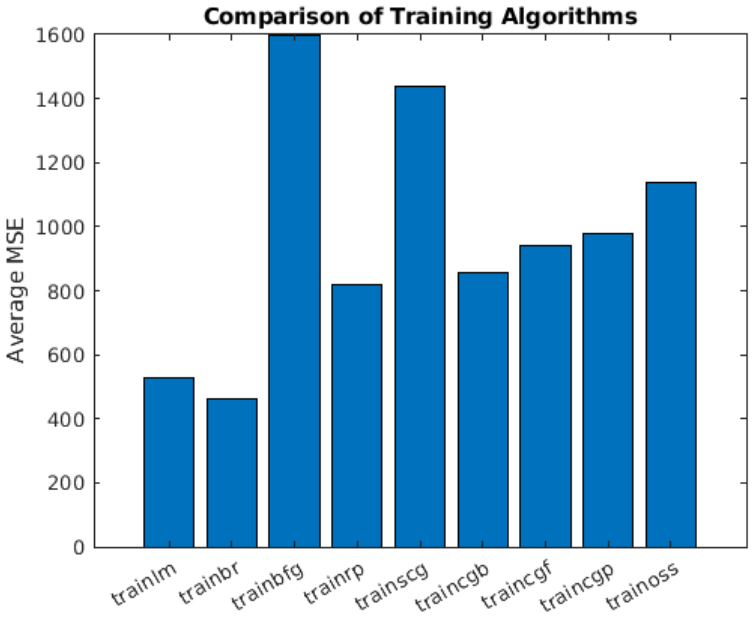

Figure 3.1: MSE values for different training functions (Mean)

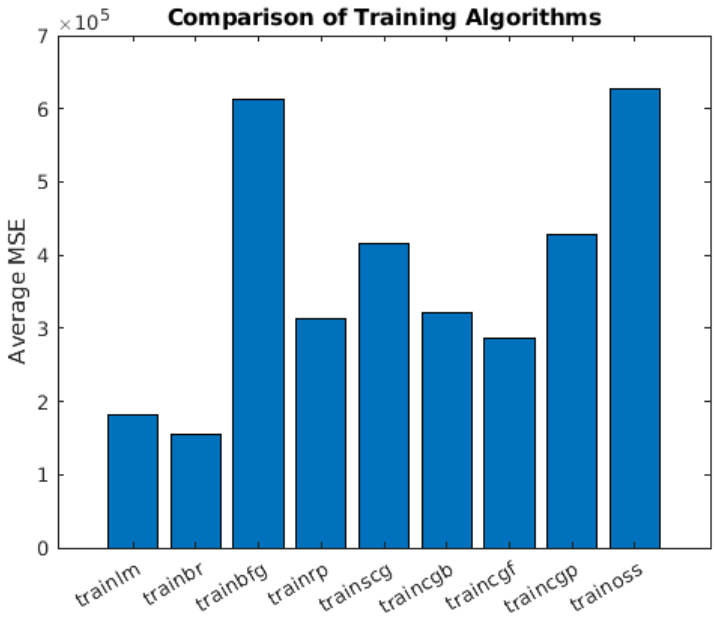

Figure 3.2: MSE values for different training functions (Std)

As can be seen from figures 3.1 and 3.2, the training function that provided the lower MSE was **trainbr**.
After that, several tests were performed to assess the optimal **number of neurons** in the hidden layer and to find the best **proportions** for the **training** and **test** sets (trainbr doesn't need a validation set). The best results were obtained with the following configuration:

- Number of neurons = 60
- Training set ratio = 0.8
- Test set ratio = 0.2

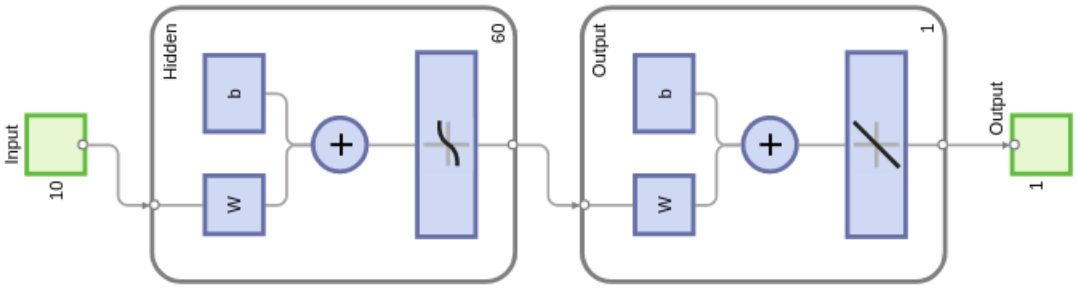

Figure 3.3: Final MLP network structure

The results obtained for the estimation of the **mean** and **standard deviation** of the **ECG** are as follows:

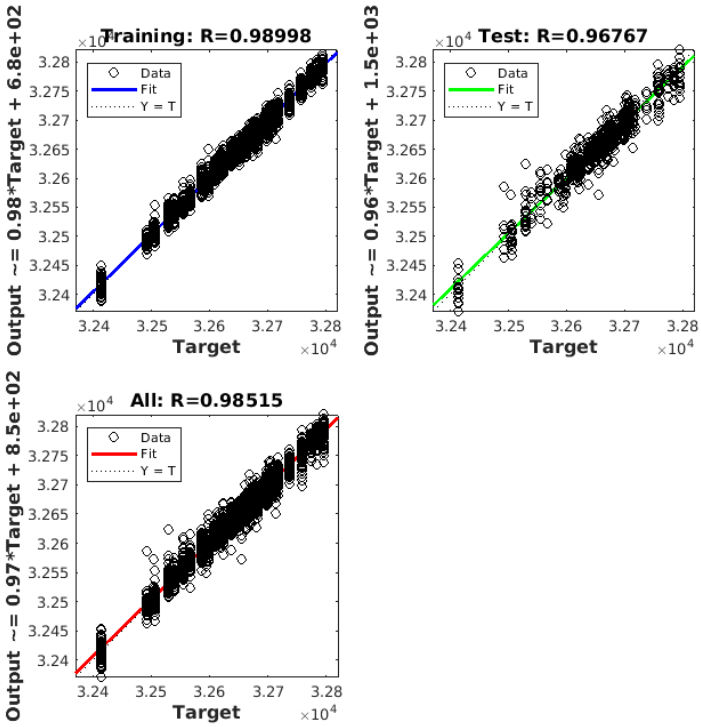

Figure 3.4: Final results MLP (Mean)

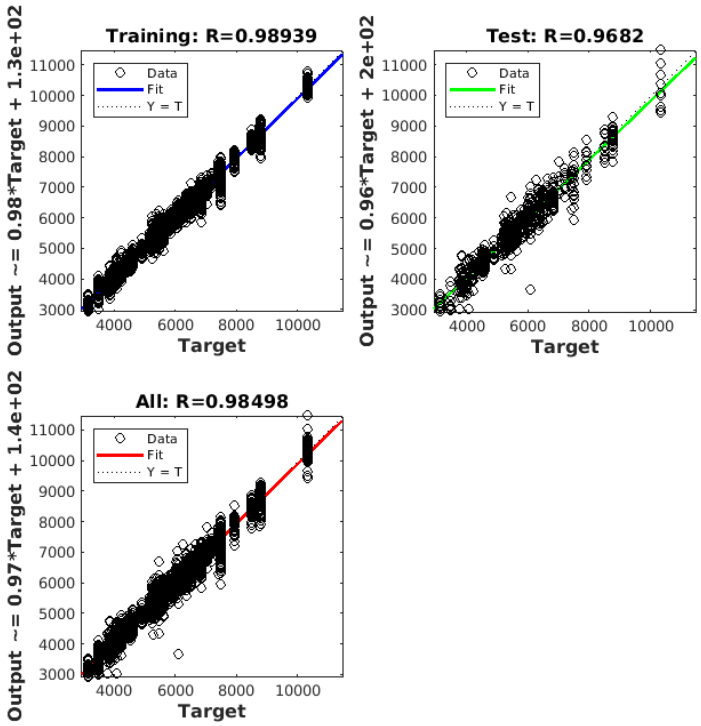

Figure 3.5: Final results MLP (Std)

3.1.2 Radial Basis Function Networks

To realize the **radial basis function network**, the Matlab function ***newrb*** was used, which is the recommended one for networks of this type.
The function ***newrb*** iteratively creates a radial basis network one neuron at a time. Neurons are added to the network until the **SSE** falls beneath an **error goal** or a **maximum number of neurons** has been reached.
The function also takes as input a parameter indicating the **spread** of the radial basis function. For the latter, a value between the minimum and maximum Euclidean distance between pairs of observations was chosen. Through several attempts, we obtained satisfactory results by setting the maximum number of neurons to **300**. The final configurations used have the following parameters:

Mean:

- Error goal = 0.0
- Spread = 0.3
- Maximum number of neurons = 300

Std:

- Error goal = 0.0
- Spread = 0.6
- Maximum number of neurons = 300

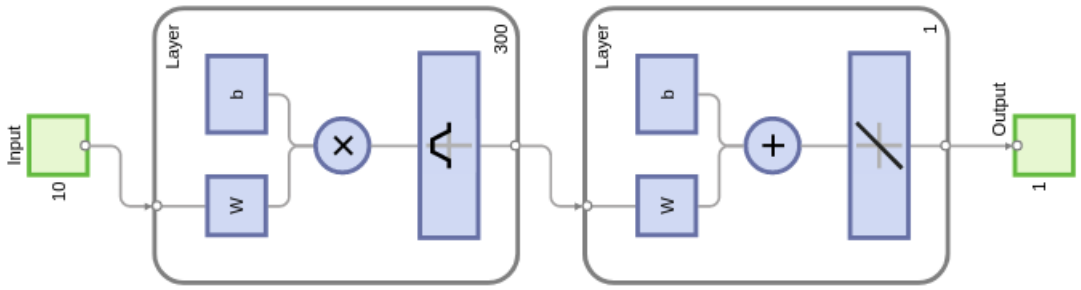

Figure 3.6: Final RBF network structure

The results obtained for the estimation of the **mean** and **standard deviation** of the **ECG** are as follows:

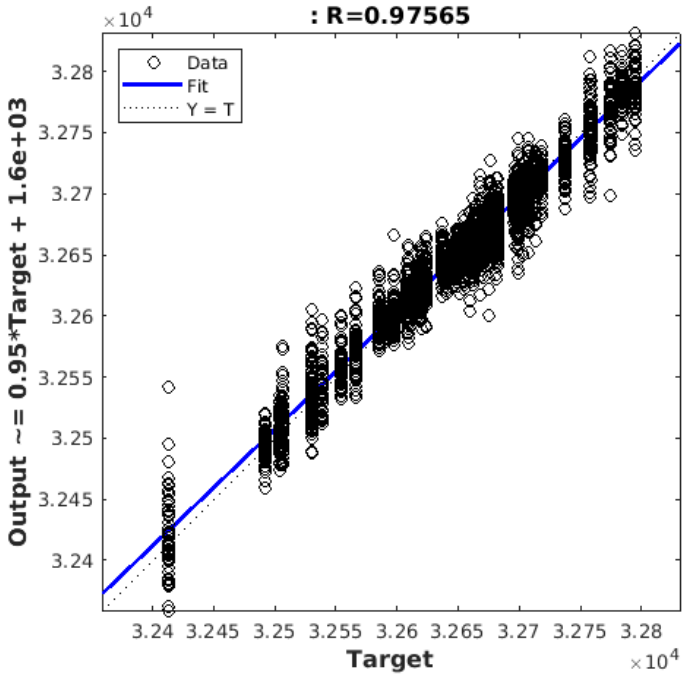

Figure 3.7: Final results RBF (Mean)

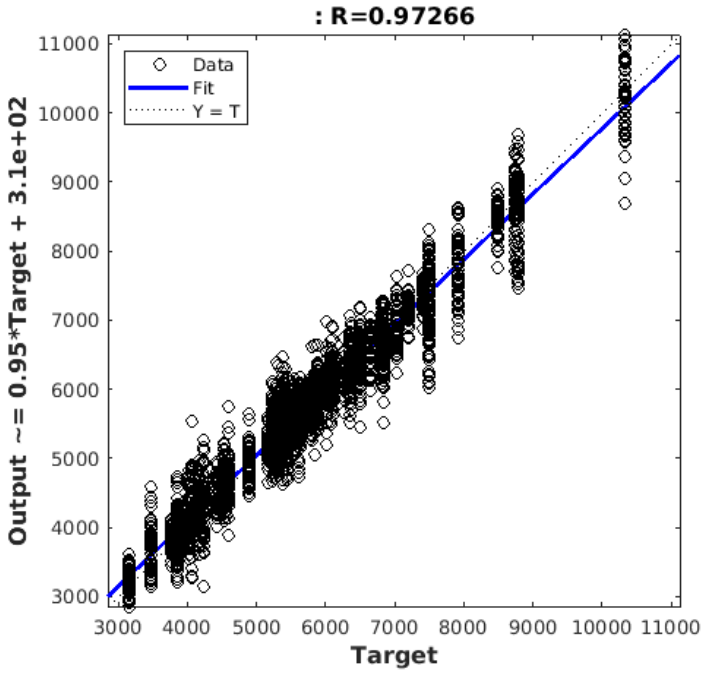

Figure 3.8: Final results RBF (Std)

3.2 Determining Activity using Neural Networks

The aim of this part was to create an **MLP** for classifying a person's **activity** between 'sit', 'walk' and 'run'.
The features used as input consisted of the **union** of those adopted in section 3.1, which resulted in a set of **15 features**.
The ***patternnet*** function of Matlab was used to create a pattern recognition neural network with the specified **hidden layer size**, **training function**, and **performance function**.
The target data for pattern recognition networks should consist of **vectors** of all zero values except for a 1 in element i, where i is the class they are to represent. To meet this requirement, the target data were structured as described in table 3.2.

Activity	Class Label	Target Vector
RUN	1	100
SIT	2	010
WALK	3	001

Table 3.2: Target Activity Vectors

With regard to the network parameters, tests were carried out with different values for the **number of neurons** and with different **training functions**. The number of neurons was varied in a range from 7 to 10, with the best results being obtained with **10 neurons**. The training functions tested were **trainbr**, **trainlm** and **trainrp**, and the results obtained are shown in the table 3.3.

Training Function	Training	Validation	Test	All
trainbr	100%	/	97.8%	99.6%
trainlm	99.8%	96.4%	95.5%	99.0%
trainrp	98.6%	95.0%	95.3%	97.9%

Table 3.3: Training Functions Performance

We also tested different proportions for the **training** and **test** sets, obtaining the best results with the following configuration:

- Number of neurons: 10
- Training function: Trainbr (Bayesian regularization backpropagation)
- Training set ratio = 0.8
- Test set ratio = 0.2

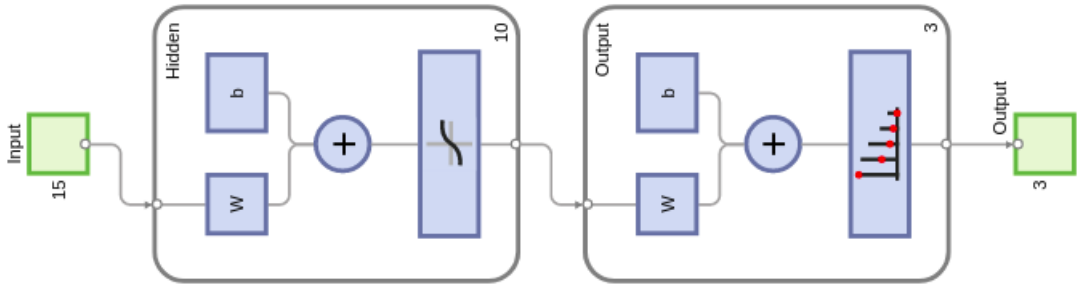

Figure 3.9: Final MLP network structure

The results obtained for the estimation of a person's **activity** are as follows:

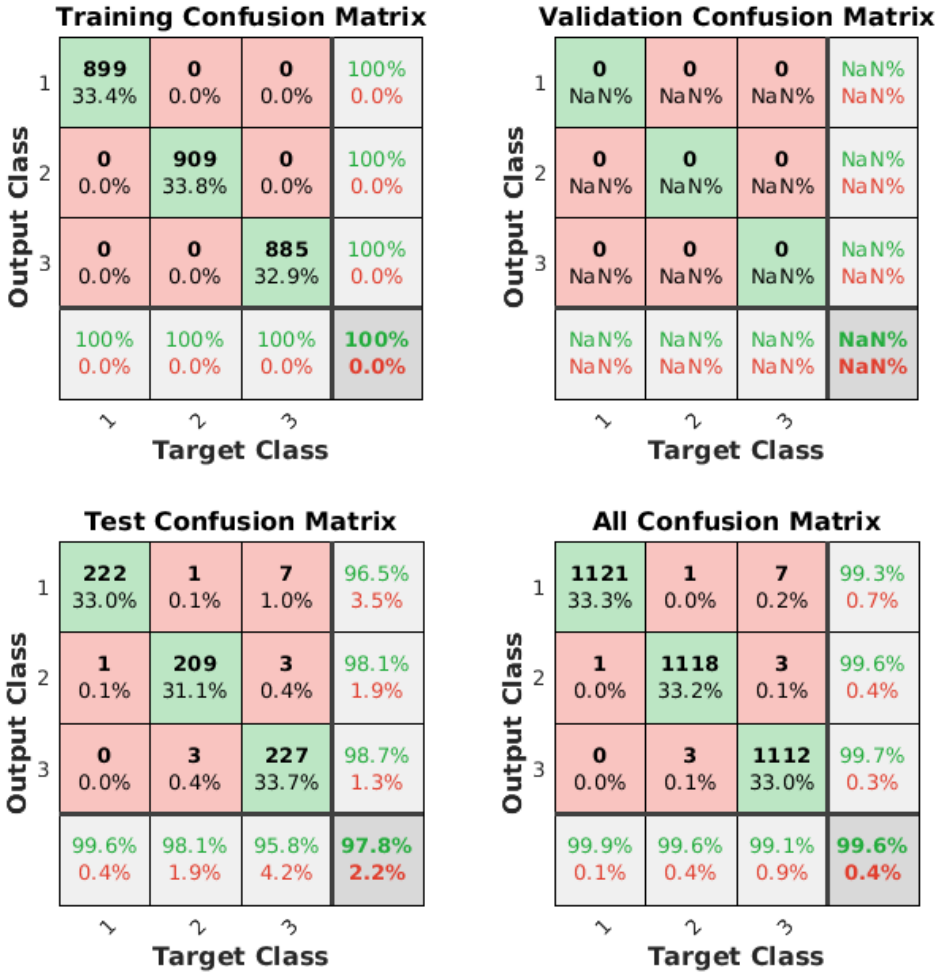

Figure 3.10: Final results MLP (Trainbr)

3.3 Fuzzy inference system

The aim of this part is to design and develop a **fuzzy inference system** to classify a person's **activity**.
From the set of features used in section 3.2, the **3 most relevant** for the classification of a person's **activity** were selected. This was done by means of the ***selectFeaturesFIS*** script, which uses the matlab ***sequentialfs*** function and a **pattern recognition network** as criterion function. The latter makes it possible to assess which are the best features in the estimation of each activity.
To achieve the objective described earlier, a **Mamdani fuzzy inference system** was implemented with the help of the Fuzzy Logic Toolbox.
As a first step in defining the control rules, we created histograms of the 3 features to observe their distribution.

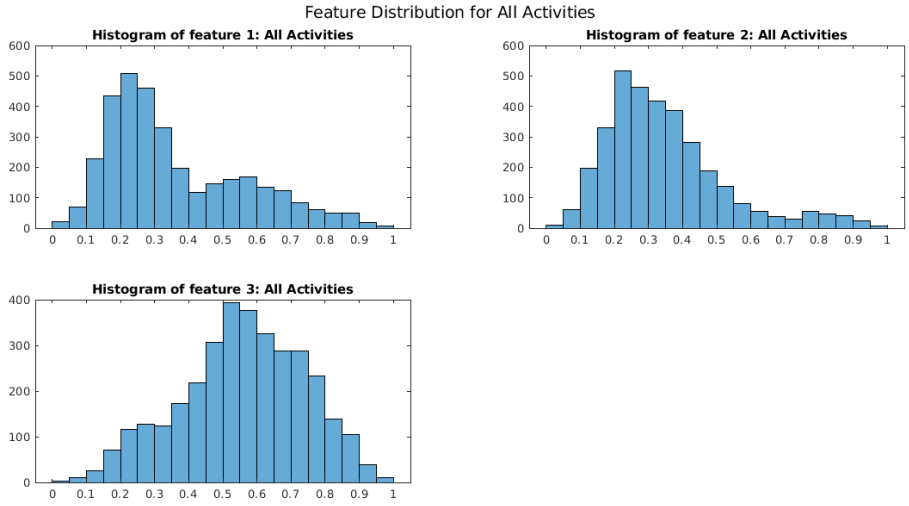

Figure 3.11: Distribution of the 3 features

The feature space was divided into **2 zones** for feature 1 (low and high) and **3 zones** for features 2 and 3 (low, medium and high). The ***trapmf*** (trapezoidal) and ***trimf*** (triangular) functions were used to obtain the membership functions for each of the features.

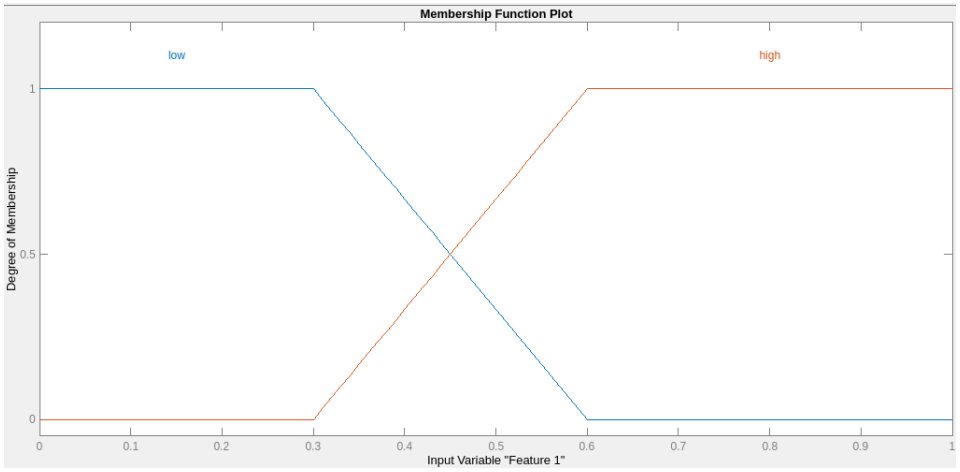

Figure 3.12: Membership function of feature 1

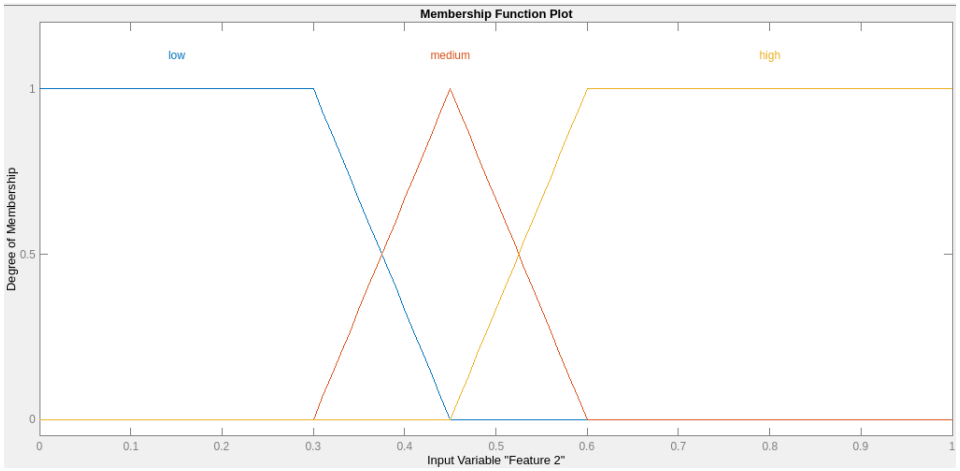

Figure 3.13: Membership function of feature 2

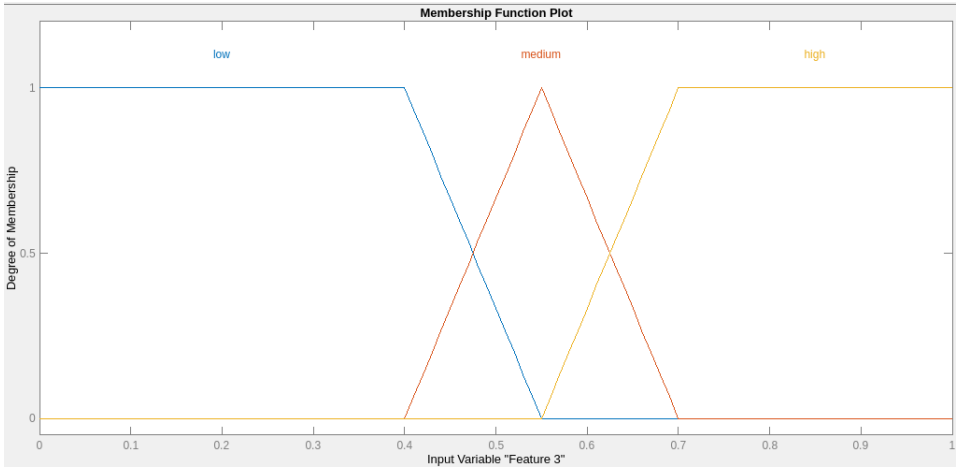

Figure 3.14: Membership function of feature 3

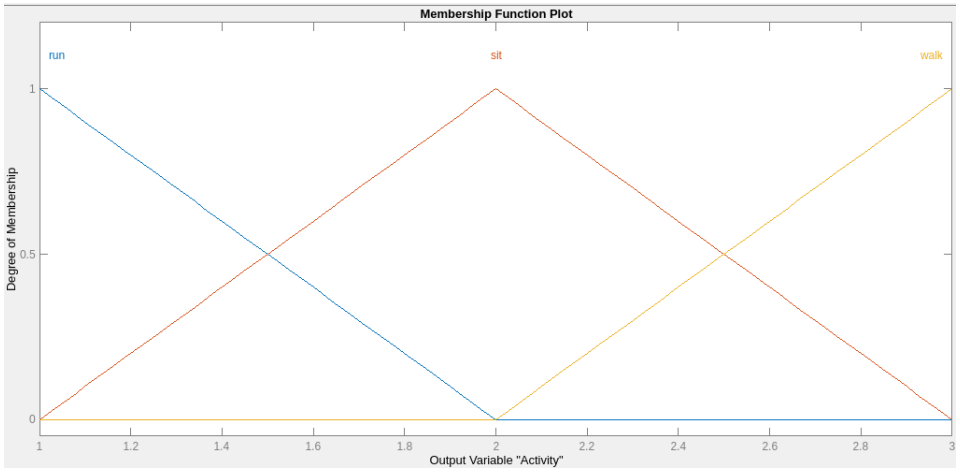

Figure 3.15: Output membership function

After doing so, we analyzed the distributions obtained for each activity to derive the fuzzy rules to be used in the system.

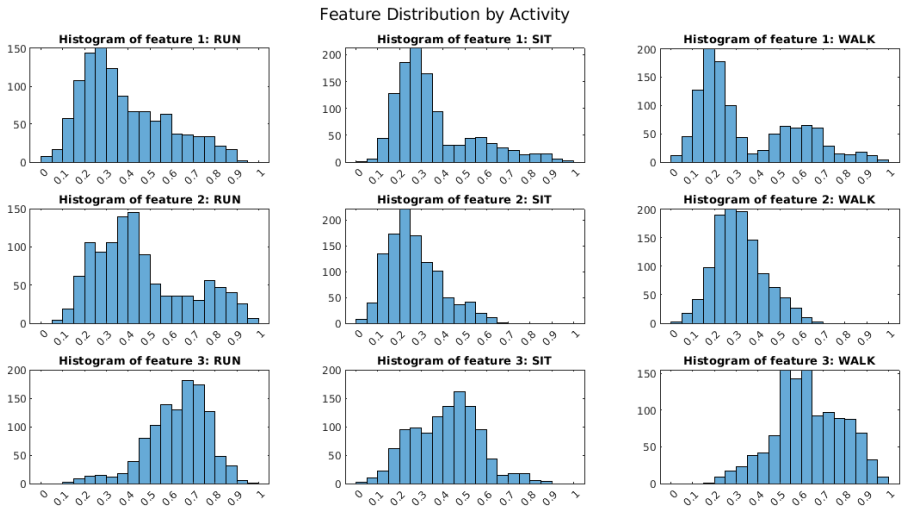

Figure 3.16: Distribution of features for each activity class

To achieve satisfactory accuracy, the **12 rules** shown in the figure 3.17 were defined.

	Rule	Weight	Name
1	If Feature 1 is low and Feature 2 is medium and Feature 3 is high then Activity is run	0.9	rule1
2	If Feature 1 is high and Feature 2 is medium and Feature 3 is high then Activity is run	0.6	rule2
3	If Feature 1 is low and Feature 2 is high and Feature 3 is high then Activity is run	0.6	rule3
4	If Feature 1 is high and Feature 2 is high and Feature 3 is high then Activity is run	0.3	rule4
5	If Feature 1 is low and Feature 2 is low and Feature 3 is medium then Activity is sit	0.9	rule5
6	If Feature 1 is high and Feature 2 is low and Feature 3 is medium then Activity is sit	0.3	rule6
7	If Feature 1 is low and Feature 2 is low and Feature 3 is low then Activity is sit	0.6	rule7
8	If Feature 1 is high and Feature 2 is low and Feature 3 is low then Activity is sit	0.2	rule8
9	If Feature 1 is low and Feature 2 is low and Feature 3 is medium then Activity is walk	0.8	rule9
10	If Feature 1 is high and Feature 2 is low and Feature 3 is medium then Activity is walk	0.5	rule10
11	If Feature 1 is low and Feature 2 is low and Feature 3 is high then Activity is walk	0.6	rule11
12	If Feature 1 is high and Feature 2 is low and Feature 3 is high then Activity is walk	0.4	rule12

Figure 3.17: Fuzzy Rules

In the system configuration, **default values** were used for the *And*, *Or*, *Implication* and *Aggregation* methods. Regarding the *Defuzzification* method, on the other hand, several options were tested, and the one that provided better results was **centroid**.

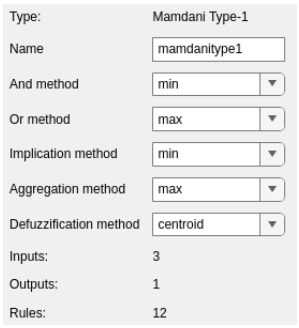

Figure 3.18: Fuzzy System Configuration

In figure 3.19, it is possible to observe, through the Rule Viewer tool, the behavior of the rules defined in the system.

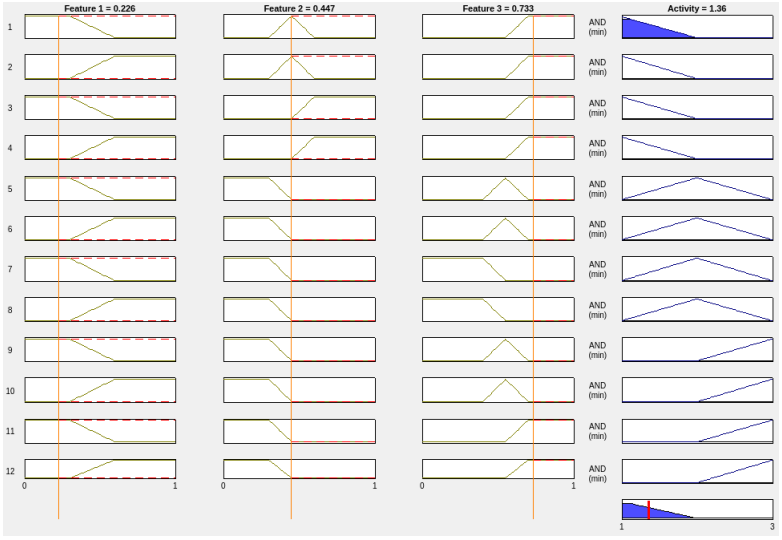

Figure 3.19: Defuzzification output

4 - Convolutional and Recurrent Neural Networks

The content of this chapter includes:

1. Estimation of the **std** of a person's ECG with a **CNN**.
2. Prediction of **one value** of a person's ECG with a **RNN**.
3. Prediction of a **set of consecutive future values** of a person's ECG with a **RNN**.

4.1 Improving ECG Estimation using CNNs

The aim of this part is to estimate the value that achieved the worst performance using the MLP in section 3.1.1, i.e. the **std**.

Data Preparation

In this case, feature extraction was not necessary, so a different data preparation was performed.
From the original dataset, **outliers** and **noisy data** (subject 13 recording during walking activity) were removed.
In addition, **normalization** of the data was also performed using the *scale* method, which was the one that provided the best results.
Finally, the input data were organized into **windows** with size **3000**.
These operations were performed inside the ***dataPreparationCNN*** script.

Convolutional Neural Network

To define the initial structure of the network, a **k-fold cross-validation** was performed with different combinations of hyperparameters. Those considered were: the **number of filters**, **filter size**, and **fully connected layer size**.
Specifically, all possible combinations of the following values related to the hyperparameters considered were tested:

- number of filters = [8, 16, 32]
- filter size = [16, 32, 64]
- fully connected layer size = [50, 100]

For each test performed, the **number of layers** in the network was set equal to **2**, and the **training algorithm** used was *adam*. For each combination of hyperparameters, the **average MSE** was calculated. The script used to perform all the operations described above was ***CNNArchitecture***, which yielded the following results:

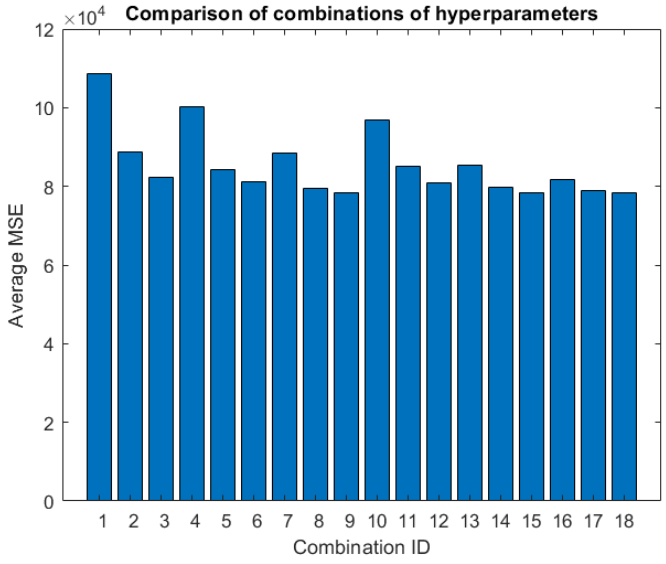

Figure 4.1: Average MSE for different combinations of hyperparameters

The combination of hyperparameters that gave the best results was **9**, which corresponds to the following values:

- number of filters = 32
- filter size = 64
- fully connected layer size = 50

Starting from this architecture, further tests were carried out through the **CNN** script in order to determine the optimal **number of layers** and the best performing **training algorithm**.
Regarding the number of layers in the network, the results obtained were as described in Table 4.1.

N. of Layers	Training	Validation
2	81.21%	82.41%
3	87.24%	88.08%
4	91.14%	89.78%
5	90.04%	88.72%

Table 4.1: Performance of the network with different number of layers

The best results were obtained with **4** convolutional layers in the network.
Finally, given this, the system was also tested with the *rmsprop* training algorithm, which, however, did not result in better performance (89.07% and 88.05% of accuracy for training and validation, respectively).

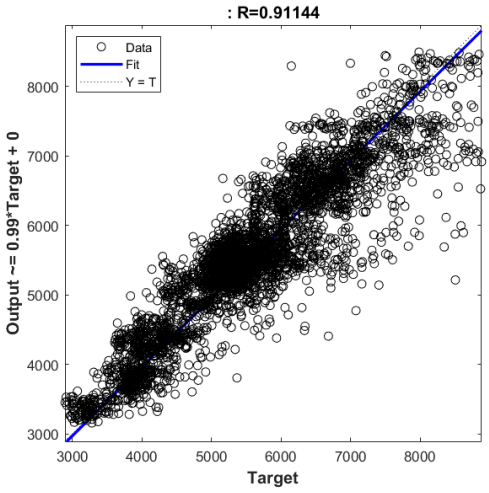

Figure 4.2: Final CNN accuracy on the training set

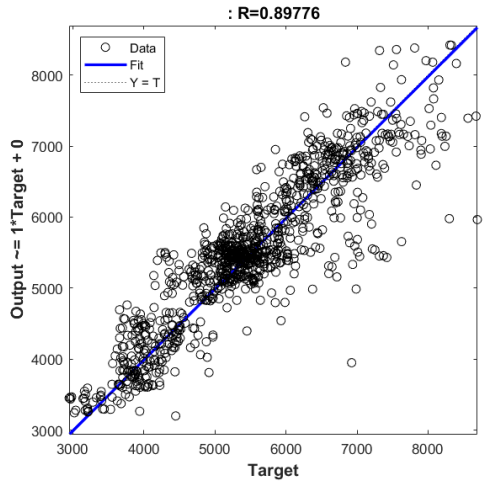

Figure 4.3: Final CNN accuracy on the validation set

4.2 Predicting ECG Value(s) using RNNs

The objective of this part is to implement an **RNN** that takes one or more signals at time steps $(t-k, \ldots, t)$ as input along with the corresponding ECG values, and returns the ECG value at time step $t+1$.

Data Preparation

Since the goal was to predict a person's ECG value based on some or all of the available signals, we decided to use the ECG values in the **sXX_YYYY_targets** files as input to the network. This step was done through the ***dataPreparationRNN*** script.
For the training of the network, from the overall data, measurements for a single target file were entered into the input matrix.
Next, **normalization** of the input data was carried out using the *range* method, which provided the best results.
The data were organized into **overlapping windows** of a specified size. Each window contained a sequence of measurements, and the target value for each window was the measurement immediately following it. The best results were obtained with a window size of **40**.

Recurrent Neural Network

To forecast time series data, we chose a **long short-term memory (LSTM) network**, which is a recurrent neural network (RNN) that processes input data by looping over time steps and updating the RNN state.
The best results were obtained with the *adam* training algorithm and with **80** neurons in the LSTM layer and, in this case, the **RMSE** value was **0.012**. The complete configuration with all training options is defined in the ***RNN*** file.
In Figure 4.4 it is possible to see a portion of the **predicted values** (dotted line) compared with the **target values** (continuous line).

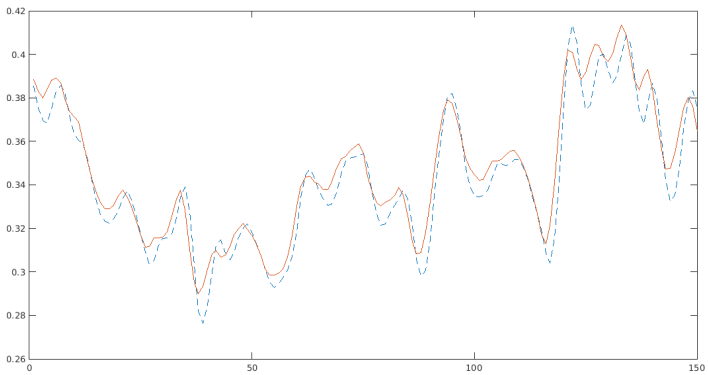

Figure 4.4: RNN results comparison

4.3 Multi-step Forecasting of ECG values using RNNs

Data Preparation

To prepare the data, in this case, we started by considering those generated for section 4.2.
As in the previous case, the data were **normalized** by the *range* method and organized into windows of size **40**. In addition, a data structure was created to maintain the **predictions** associated with each window (5 values were considered).
These operations were done in the ***multiStepRNN*** file, which also had the task of creating the RNN, which was later used to predict the 5 values.
The network's structure is the same as defined in the ***RNN*** file and is saved for use at the next step.

Multi-step Forecasting

The ***multiStepForecasting*** script loads the pre-trained RNN model, the validation dataset, and the true target values.
5 consecutive values are predicted for each window in the validation dataset using Matlab's ***predictAndUpdateState*** function. We, therefore, performed **closed-loop forecasting** using the pre-trained network.
An RMSE value was calculated for each sequence and, subsequently, the **average RMSE** over all windows, which was found to be **0.0264**.

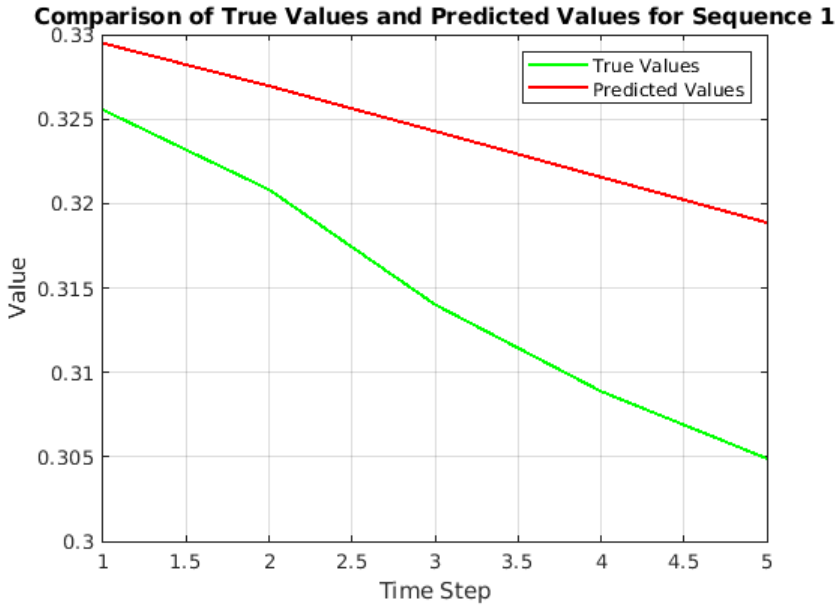

Figure 4.5: Comparison of True values and Predicted values of sequence 1

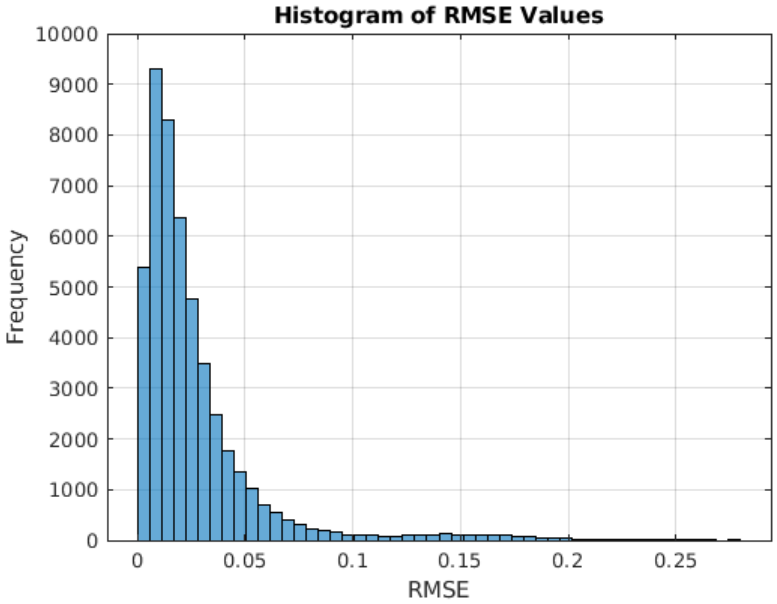

Figure 4.6: Histogram of RMSE values