UNIVERSITÀ DI PISA

Computer Engineering

Intelligent Systems

Pulse Transit Time
Neural Network Models

Elaborated by
Arsalen BIDANI

Academic year: 2023/2024

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 this project is to apply several intelligent systems adopted for the analysis of the timeseries of the physiological singlas, in order to extract by leveraging the different machine learning algorithms and signal processing techniques, we aim to uncover valuable insights such as the ECG (Electrocardiogram) feature values such as the **Mean** and **STD** or to **classify human activities** based on the data supplied.

In fact according to project specification these systems were carried out on **Matlab 2022a**:

1. Designing and developing **two multi-layer perceptron (MLPs)** artificial neural networks that estimate, respectively the **mean** and **standard deviation** of ecg of a person (XX) during the three activities (YYYY), based on sensor data stored in the timeseries file.

2. Design and develop a **multi-layer perceptron (MLP)** that **classifies** a person's activity among 'sit', 'walk' and 'run'. The MLP takes as input a set of features and returns the corresponding activity.

3. Develop a **fuzzy inference system (FIS)** to classify a person's activity using the k most relevant features ($k \leq 5$) in the set of features used to train the before mentioned classifier.

4. Designinng a **convolutional neural network (CNN)** instead of an MLP, the value to be estimated (mean/standard deviation) is the one that achieved the worst

performance using the MLP. The goal is to find the best CNN architecture along with the **best hyperparameter values** (filter size, number of filters, etc.)

5. Design and develop a **recurrent neural network (RNN)** that predicts one value of a person's **ECG**, based on part or all of the timseries.
   The RNN takes these 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. The goal is to find the **best architecture** of the RNN, along with the **best hyperparameter values** (size of the time window, etc.)

2. Dataset and Features Extractions

2.1.

2.2. Dataset Structure

The **dataset** to be used for this project is **structured** as following:

For each subject we have **3 recordings**, each recording divided into **2 CSV files**, one containing **physiological signals**, and the other containing related **'ECG' target values**:

- sXX_YYYY_timeseries;
- sXX_YYYY_targets;

's' stands for subject, **'XX'** stands for the subject number (1-22), and **'YYYY'** stands for an activity among **'sit', 'walk',** and **'run'**.

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. **Signals** are as follows:

- **pleth_1:** red wavelength PPG from the distal phalanx (first segment) of the left index finger palmar side (sampling rate 500 Hz)

- **pleth_2:** infrared wavelength PPG from the distal phalanx (first segment) of the left index finger palmar side (sampling rate 500 Hz)

- **pleth_3:** green wavelength PPG from the distal phalanx (first segment) of the left index finger palmar side (sampling rate 500 Hz)

- **pleth_4:** red wavelength PPG from the proximal phalanx (base segment) of the left index finger palmar side (sampling rate 500 Hz)

- **pleth_5:** infrared wavelength PPG from the proximal phalanx (base segment) of the left index finger palmar side (sampling rate 500 Hz)

- **pleth_6:** green wavelength PPG from the proximal phalanx (base segment) of the left index finger palmar side (sampling rate 500 Hz)

- **lc_1:** load cell proximal phalanx (first segment) PPG sensor attachment pressure (sampling rate 80Hz)

- **lc_2:** load cell (base segment) PPG sensor attachment pressure (sampling rate 80Hz)

- **temp_1:** distal phalanx (first segment) PPG sensor temperature (°C, sampling rate 10Hz)

- **temp_2:** proximal phalanx (base segment) PPG sensor temperature in (°C, sampling rate 10 Hz)

- **temp_3:** ambient temperature (°C, sampling rate 500 Hz)

Each file sXX_YYYY_targets contains the target time series. The first column contains a timestamp. The other contains the target time series:

- **ecg:** 3-channel ECG sampled at 500 Hz

2.3. Dataset Processing

After analyzing the physiological signals we've had, **12 features** from **time and frequency domain** were extracted as shown in the table below:

TIME DOMAIN	FREQUENCY DOMAIN
MEAN	*MEAN FREQ*
MEDIAN	*MEDIAN FREQ*
VARIANCE	*OCCUPIED BANDWIDTH*
KURTOSIS	—
SKEWNESS	—
INTERQUANTILE_RANGE_GSR	—
ENERGY	—
MINIMUM	—
MAXIMUM	—

On this project we are going to work via the provided dataset on **66 samples** with each constructed by 11 signals explained beforehand.

There's different ways of extraction of the features from the signals we've had namely:

1. Extracting 12 features **directly from each signal**, resulting in 132 feature per sample.
2. Imposing a certain number of **contiguous windows** and extracting the features from each one.
3. Applying a number of **overlapping windows** and extracting from each.

We have opted to use **7 overlapping windows** with an **overlapping factor of 50%**, as it nailed better results when we compared it with the **SequentialFS** tool of Matlab.

924 features were obtained as a result '`create_features_matrix`' function we've created to which we further tuned down to **277 features** by eliminating those which had a **correlation of over 90%**.

Consequently we have normalized the values of input matrix in a **'range'** method based way, in order to rescale all values between **0 and 1,** which help later on training phases of the models.

We have determined that 66 samples won't be enough to properly train the models and an augmentation of the data was need through an **AutoEncoder** of these paramerters**:**

- **Augmentation factor = 40**, meaning that 40 samples for each already existing one.
- **SparsityProportion = 0.1,** a low value for SparsityProportion usually leads to each neuron in the hidden layer "specializing" by only giving a high output for a small number of training examples.
- **Linear Encoding/Decoding functions:** Satlin, Purelin.

As suggested by the given requirements, a down size of the number of the features was executed using **SequentialFS** in order to extract the **top 10 performant features** to predict the Mean and STD, a criterion selection function '`feature_selection`' was developed for this purpose that consists of creating a **fitnet** with a hidden network consisting of **8 neurones.**

In the end have obtained a features matrix composed of **2706 samples** and **10 features**.

3. ECG Estimations using Multi-Layer Perceptron

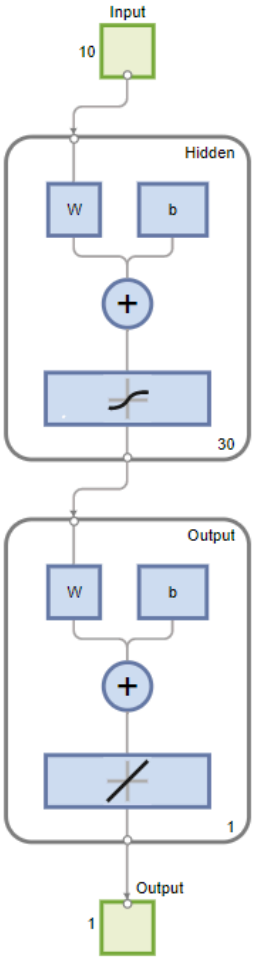
In this part of the project we are going to train two multi-layer perceptron to predict the consecutively the mean and the standard deviation of the ECG, each supported by a different features and target matrix.

3.1. ECG Mean Estimation

In order to craft the estimator model we have used a **fitnet** to train, with **one hidden layer** as the projectation wasn't complex. Consequently to get the best performance we had to tune multiple parameters namely:

- **Number of hidden neurons**
- **Training Algorithms**
- **Number of Epochs**

The structure of the network used was the following:

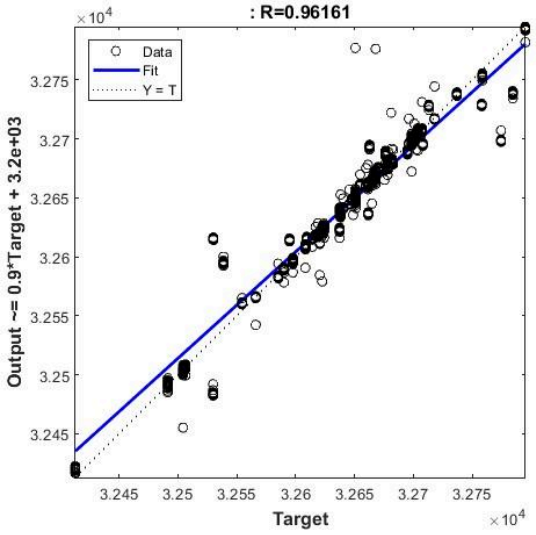


Mean MLP Fitting Neural Network

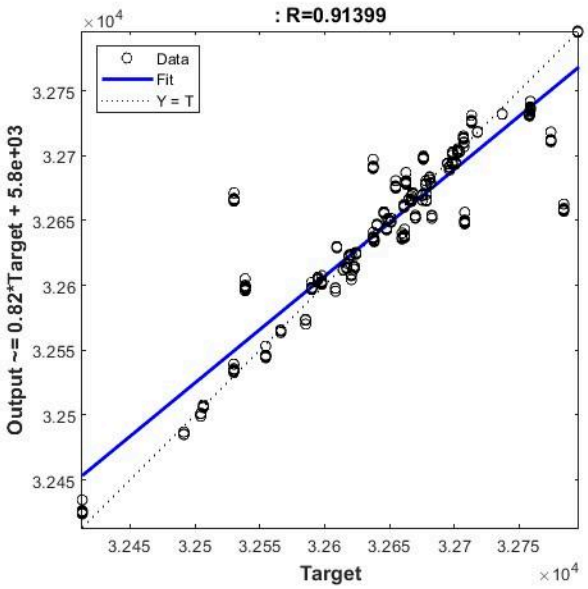
We concluded after analyzing the performance results and changes of the number of neurons that **30 neurons for the hidden layer** provided the most stable results and prevented overfitting of the network.
For the epoch numbern, we've noticed that after 200 epochs the 'R Squared' was nearly 1 and indicated an overfitting of the model.

Training function of the fitting network can be chosen out of the 12 algorithms that Matlab provides, we've tried three main algorithms and we've compared the performances:

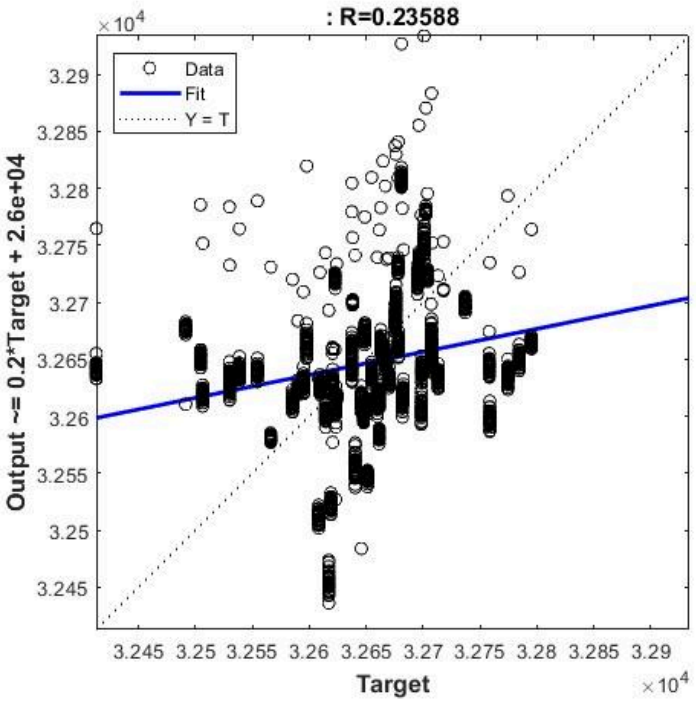
- **Levenberg-Marquardt**:



- **Bayesian Regularization**:



- **BFGS Quasi-Newton**:

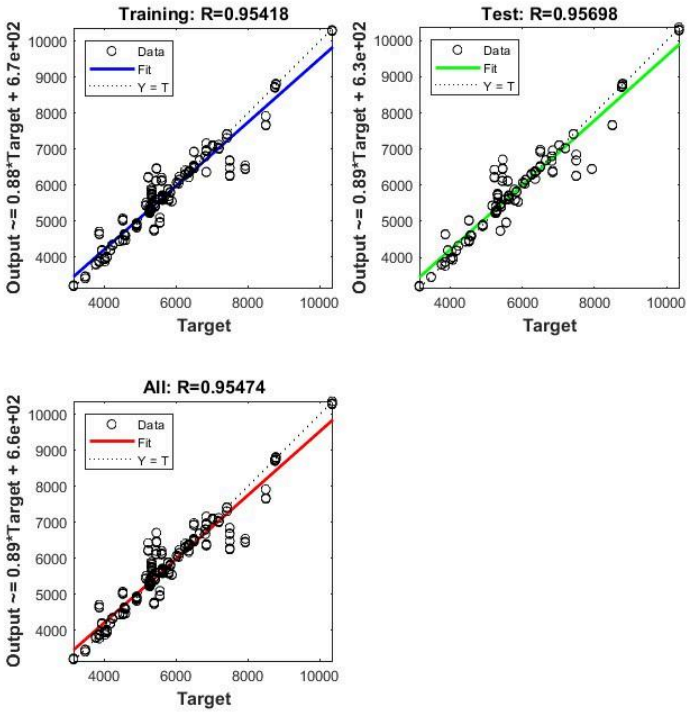


As a result the final network was trained using **Levenberg-Marquardt** algorithm and the obtained results of testing we largely acceptable.

3.2. ECG Standard Deviation Estimation

The same approach was taken for the network of the **STD** estimator, in fact we will list the parameters that were obtained at the end and performance of it:

- A hidden layer consisting of **20 neurons.**
- **Bayesian Regularization** training function.
- 30 epochs were enough to be obtain results without an overfitting of the model.
- An **80/20** training to testing ratio, as the BR algorithm doesn't require a validation set.



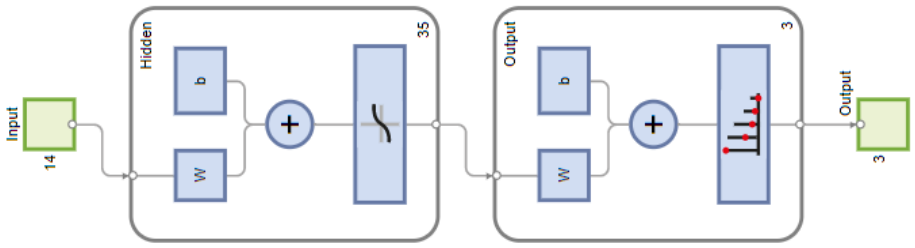
STD MLP Regression Results

4. Multi-Layer Perceptron Activity Classification

This section deals with the **activity classification** of a person through the physiological signals that were recorded, though the design and development of a **multi-layer perceptron classifier**.

The input data to be used consists of the union of the features that were used for mean and standard deviation ECG fittings, in fact we have used **14 features** in total to train the classifier.

For the target data we have used '`ind2vec`' function on activities target vector which takes a row vector of indices, ind, and returns a sparse matrix of vectors, vec, containing a 1 in the row of the index they represent, as indicated by ind.



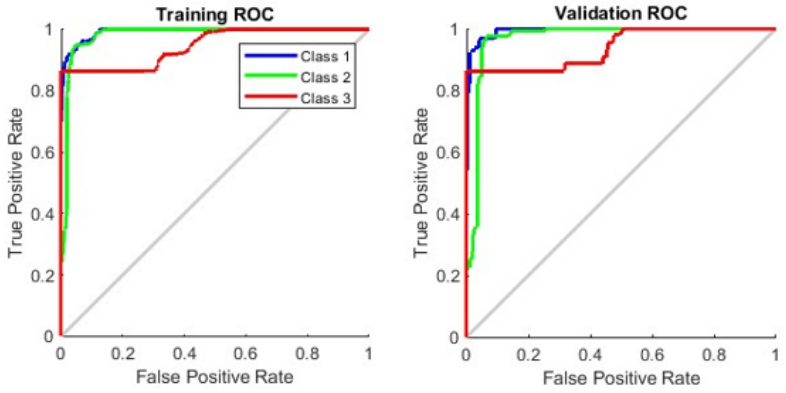
Activity MLP Classifier Strucuture

We have obtained the above architecture of the MLP in fact these were the parameters used:

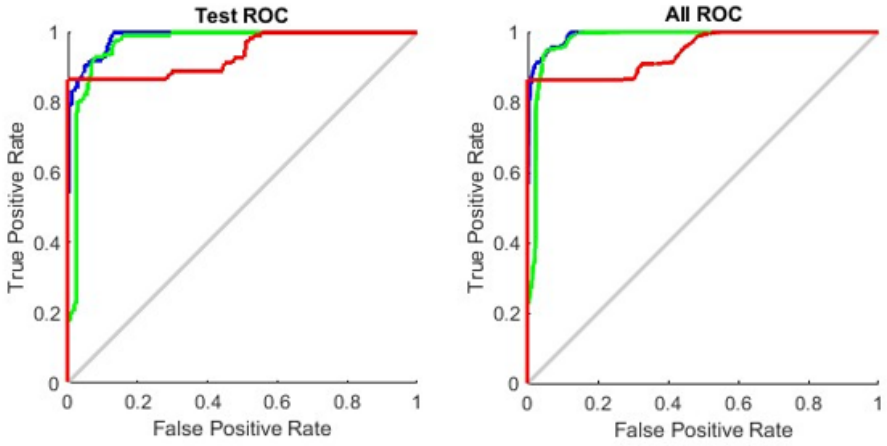
- **Hidden layer neurons number :** 35
- **Number of epochs of training:** 30
- **Training algorithm:** Levenberg-Marquardt ('trainlm')

We've performed several testing phases with the training algorithms available and we were able to obtain a result of **89.4% of correct classifications.**
The dataset was partitioned with a ratio of **70/15/15** consecutively for the training, validation a testing phases, the division function used was **'dividerand'** (randomly divided).

These were the ROC curves obtained in the end:



MLP CLASSIFIER ROC Curves



MLP CLASSIFIER ROC Curves

These were the confusion matrices that showcase a success of **91%** combined correct classifications:

MLP Classifier Confusion Matrices

5. Fuzzy Inference System Activity Classification

We are asked in this part of the project to develop a **fuzzy inference system** to classify a person's activity, based on the data provided by **5 most relevant** features extracted from the set used to train the aforementioned **MLP classifier.**

The 5 features were extracted using SequentialFS with same '`feature_selection`' function but with **18 neuron** in the hidden layer, we obtained the following results:

Step 1, added column 11, criterion value 0.219229
Step 2, added column 9, criterion value 0.00787753
Step 3, added column 4, criterion value 3.03616e-05
Step 4, added column 10, criterion value 1.23764e-05

Step 5, added column 12, criterion value 7.4564e-06

Final columns included: 4 9 10 11 12

We are in a situation where we have an input/output data of the FIS we want to model but do not necessarily have a **predetermined model structure** based on the characteristics of variables of the signals.

Discerning membership functions parameters by looking at data can be difficult in this situation, so we have opted for an **Adaptive Network-based Fuzzy Inference System (ANFIS).**

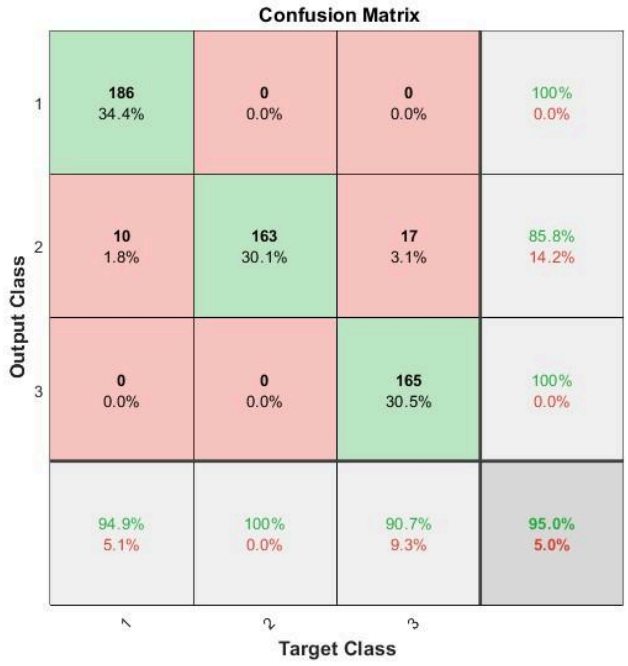
In order to generate and train the FIS we have partitioned the data on **80/20 ratio** between training and validation sets and **randomly shuffled** the rows of the matrices .

We have used '`genfis`' Matlab function in order to generate a **Takagi Sugeno Kang (TSK)** and then tune and train the anfis using '`tunefis`' function.

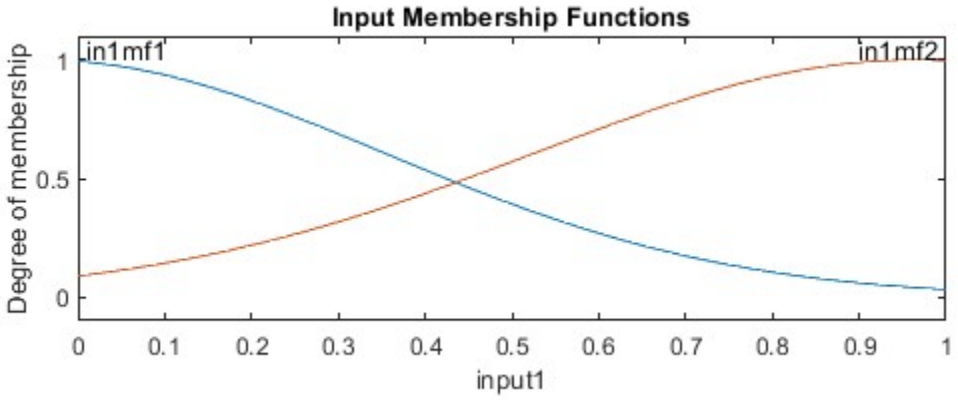
We've had several parameters to tune in order to achieve an acceptable performance of the FIS, mainly we note:

1. **FIS Generation Options:**
    a. **Grid Partition:** Generates input membership functions by uniformly partitioning the input variable ranges
    b. **FCM Clustering:** membership function and rules derived from data clusters found using FCM clustering of input and output data
    c. **Substractive Clustering:** rules are derived from data clusters found using subtractive clustering of input and output data.
2. **Defuzzification Method :**
    a. **"Wtaver":** Weighted average of all rule outputs
    b. **"Wtsum":** Weighted sum of all rule outputs
3. **Input/Output functions**
4. **Number of Traning Epochs**
5.

Our resulting FIS system consisted of **2 membership functions** according to **'Grid Parition'** method, while the combination of the input function type was **Gaussian** and **Linear** for output has resulted for the best performance, and defuzzification method was of the **weighted average of all rule outputs**.
15 epochs were enough to avoid overfit and we were able through to obtain these results notably **95%** was largely efficient:



The final tuned FIS consisted of **10 membership functions** and this figure showcase a plot of the **first input variable** MFs plot:



FIS Input 1 Variable MFs

The final FIS consisted in total of **32 rule base** and this is the 15th rules included in the system :

```
'1. If (input1 is in1mf1) and (input2 is in2mf1) and (input3 is in3mf1) and (input4 is in4mf1) and (input5 is in5mf1) then (output is out1mf1) (1)  '
'2. If (input1 is in1mf1) and (input2 is in2mf1) and (input3 is in3mf1) and (input4 is in4mf1) and (input5 is in5mf2) then (output is out1mf2) (1)  '
'3. If (input1 is in1mf1) and (input2 is in2mf1) and (input3 is in3mf1) and (input4 is in4mf2) and (input5 is in5mf1) then (output is out1mf3) (1)  '
'4. If (input1 is in1mf1) and (input2 is in2mf1) and (input3 is in3mf1) and (input4 is in4mf2) and (input5 is in5mf2) then (output is out1mf4) (1)  '
'5. If (input1 is in1mf1) and (input2 is in2mf1) and (input3 is in3mf2) and (input4 is in4mf1) and (input5 is in5mf1) then (output is out1mf5) (1)  '
'6. If (input1 is in1mf1) and (input2 is in2mf1) and (input3 is in3mf2) and (input4 is in4mf1) and (input5 is in5mf2) then (output is out1mf6) (1)  '
'7. If (input1 is in1mf1) and (input2 is in2mf1) and (input3 is in3mf2) and (input4 is in4mf2) and (input5 is in5mf1) then (output is out1mf7) (1)  '
'8. If (input1 is in1mf1) and (input2 is in2mf1) and (input3 is in3mf2) and (input4 is in4mf2) and (input5 is in5mf2) then (output is out1mf8) (1)  '
'9. If (input1 is in1mf1) and (input2 is in2mf2) and (input3 is in3mf1) and (input4 is in4mf1) and (input5 is in5mf1) then (output is out1mf9) (1)  '
'10. If (input1 is in1mf1) and (input2 is in2mf2) and (input3 is in3mf1) and (input4 is in4mf1) and (input5 is in5mf2) then (output is out1mf10) (1)'
'11. If (input1 is in1mf1) and (input2 is in2mf2) and (input3 is in3mf1) and (input4 is in4mf2) and (input5 is in5mf1) then (output is out1mf11) (1)'
'12. If (input1 is in1mf1) and (input2 is in2mf2) and (input3 is in3mf1) and (input4 is in4mf2) and (input5 is in5mf2) then (output is out1mf12) (1)'
'13. If (input1 is in1mf1) and (input2 is in2mf2) and (input3 is in3mf2) and (input4 is in4mf1) and (input5 is in5mf1) then (output is out1mf13) (1)'
'14. If (input1 is in1mf1) and (input2 is in2mf2) and (input3 is in3mf2) and (input4 is in4mf1) and (input5 is in5mf2) then (output is out1mf14) (1)'
'15. If (input1 is in1mf1) and (input2 is in2mf2) and (input3 is in3mf2) and (input4 is in4mf2) and (input5 is in5mf1) then (output is out1mf15) (1)'
```

These are the results we have obtained using different generation methods:

Generation Method	Correct Classification Percentage
SubtractiveClustering	86.7%
FCMClustering(4 clusters)	69.68%
GridPartition (3 functions)	Model Overfit

6. STD Estimation using Convolutional Neural Network

The aim of this part of the project is improve the performance of the **worst performer** of the **2 MLPs ECG estimators** that we have developed in part ECG Estimations using Multi-Layer Perceptron though the design and development of a **Convolutional Neural Network**.

CNNs are a type of deep neural networks that accepts raw signals as inputs, but in order to train we have to pursue a data augmentation on the dataset that we already have.
A function denoted as
‘`generate_data(dataset_path, window_size,remove_noisy_recording)`’
was put into place as it accepts window size and a boolean for removing noisy recording.

As we have mentioned before one of the techniques of extracting features from signals and it to divide in a **number of windows** and work on them individually, in this case the **number of stamps = '5000'** was chosen and the std is calculated upon it and then attached to targets file.

We have run a phase of removal of outliers from the inputs and further normalizing the input data according to **range based method.**

We have added a **boolean** into the function that permit the **removal of subject 13 recordings** it being noisy during the "Walk" activity.

In our case we are going to develop an CNN to estimate the **"Standard Deviation"** of ECG, it being the worse performer out of the 2 MLPS.
In order to achieve that goal and to develop an reliable CNN architecture, several hyperparameters and layers modification has to be put into place.

6.1. Network Architecture

Our final network structure is composed as shown:

```
% Network structure
layers = [
    sequenceInputLayer(num_channels)

    convolution1dLayer(filter_size(1), num_filters(1), 'Stride', convolutional_stride(1), 'Padding', 'same')
    batchNormalizationLayer
    maxPooling1dLayer(pooling_compression(1), 'Stride', pooling_stride(1), 'Padding', 'same')
    reluLayer

    convolution1dLayer(filter_size(2), num_filters(2), 'Stride', convolutional_stride(2), 'Padding', 'same')
    batchNormalizationLayer
    maxPooling1dLayer(pooling_compression(2), 'Stride', pooling_stride(2), 'Padding', 'same')
    reluLayer

    convolution1dLayer(filter_size(3), num_filters(3), 'Stride', convolutional_stride(3), 'Padding', 'same')
    batchNormalizationLayer
    reluLayer

    convolution1dLayer(filter_size(4), num_filters(4), 'Stride', convolutional_stride(4), 'Padding', 'same')
    batchNormalizationLayer
    maxPooling1dLayer(pooling_compression(3), 'Stride', pooling_stride(3), 'Padding', 'same')
    reluLayer

    globalAveragePooling1dLayer

    fullyConnectedLayer(hidden_layer_size)
    dropoutLayer(0.3)
    leakyReluLayer
    fullyConnectedLayer(output_layer_size)

    regressionLayer
];
```

The architecture is composed by an input layer, **4 cascade fashion** connected **'Base Blocks'** with a last output block.

1. Base Block layers organized sequentially:

- **Input Layer:** This layer accepts input sequences with a number of channels determined by the num_channels parameter.

- **Convolutional 1-D Layers:** These layers perform convolution operations with filters of varying sizes (filter_size) and numbers (num_filters).

- **Batch Normalization**: Each convolution is followed by batch normalization to stabilize and accelerate training. Padding is set to 'same' to ensure that the output size matches the input size.

- **Max Pooling Layers:** After each convolutional layer, max pooling is applied to compress the spatial dimensions of the feature maps, reducing computational complexity and the risk of overfitting. Pooling compression and stride are set according to **pooling_compression** and **pooling_stride parameters**, respectively.

- **Rectified Linear Unit (ReLU)**: activation is applied to introduce non-linearity after each convolutional layer, promoting the model's ability to learn complex patterns in the data.

2. ***Output Block:***
    - **Global Average Pooling Layer:** Following the convolutional layers, a global average pooling layer aggregates feature maps across all spatial locations, resulting in a single value per feature map. This operation reduces the dimensionality of the feature maps and helps in capturing the most salient features.

    - **Fully Connected Layers:** These layers connect all neurons from the previous layer to the next, enabling the network to learn complex relationships between features. A dropout layer is included to prevent overfitting by randomly dropping a fraction of the neurons during training.

    - **Leaky ReLU Activation:** Leaky Rectified Linear Unit (ReLU) activation is applied to introduce non-linearity in the fully connected layers, similar to ReLU but allowing a small, positive gradient when the input is negative.

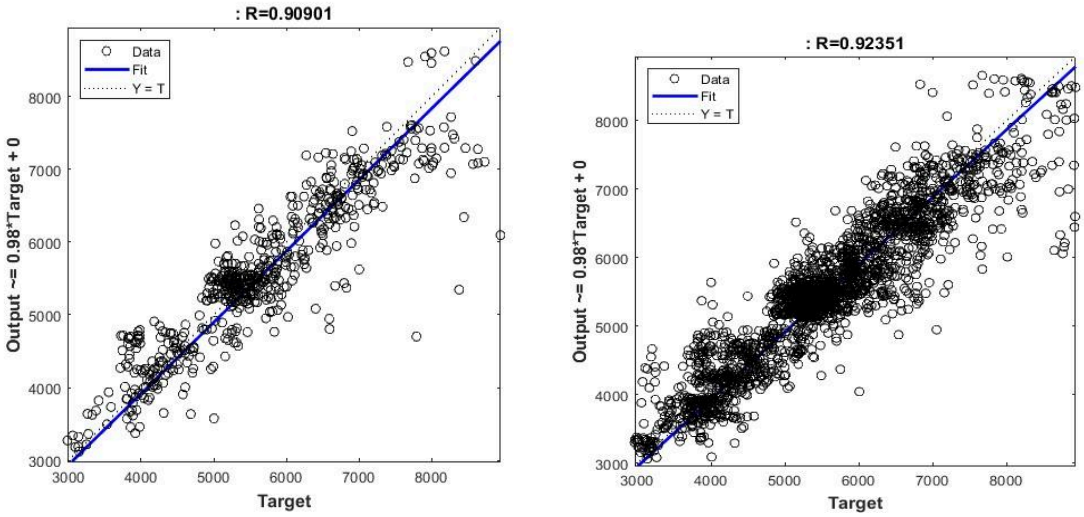
    - **Output Layer:** The final fully connected layer produces the output predictions.

The final network network was trained on **30 epochs** with a mini batch size of **80** and these were the hyperparameters tuned for the CNN final architecture:

Parameter	Value
Number of Channels	11
Number of Filters	[96, 256, 348, 348]
Filter Size	[11, 5, 3, 3]
Convolutional Stride	[4, 4, 4, 4]
Pooling Compression	[2, 2, 2]
Pooling Stride	[2, 2, 2]
Hidden Layer Size	256
Output Layer Size	1
L2 Regularization	0.01
Number of Epochs	30
Mini-Batch Size	80
Initial Learning Rate	0.01
Learning Rate Schedule	piecewise
Learn Rate Drop Period	20
Learn Rate Drop Factor	0.1

6.2. CNN Results

The results we've obtained were on the acceptable range but even though we have tried to tune the network by adding more base blocks made the architecture too complex to train or resulted in worse performance, the figures below showcase the regression results plots of the training and validation sets:



Validation(left) and Training(right) Regressions

7. Forecast ECG Value using Recurrent Neural Networks

The aim of this part is use Recurrent Neural Network in order to forecast a single step of the a person's ECG.

The neural network receives as input one or more signals through values they takes in various time intervals with a specified width of the window they act upon.

7.1. Network Architecture

For the network architecture that we carried on,we chose to develop a **Long-Short Term Memory RNN.**

The provided network structure consists of the following layers:

- **Sequence Input Layer:** Accepts input sequences with a specified number of channels (num_channels =1) because it univariate.

- **Long Short-Term Memory (LSTM) Layer:** Utilizes LSTM units to process sequential data, with a specified number of units (lstm_layer_size). The 'OutputMode' is set to 'last', meaning only the output of the last time step is considered.

- **Fully Connected Layer:** Connects all neurons from the previous layer to the next. It reduces the dimensionality of the output to match the number of input channels.

- **Dropout Layer:** Randomly drops a fraction of the neurons during training to prevent overfitting. The dropout probability is determined by dropout_probability.

- **Fully Connected Output Layer:** Produces the final output predictions. The dimensionality of the output is determined by output_layer_size.

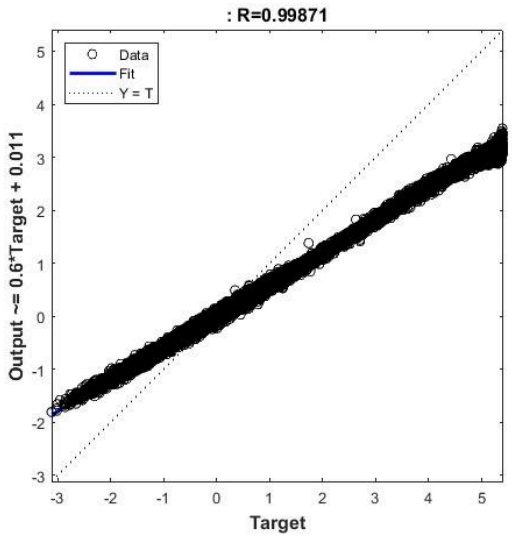
- **Regression Layer:** Computes the loss between the predicted and target values, making this network suitable for regression tasks.

Number of channels is equal to **1** here in this network means that to predict a single value of the ECG, it is enough to supply the previous k= window_size samples to have acceptable values.

This network has reached acceptable results by training it on **10 epochs** with **2500** as **mini-batch size,** the number of **windows size** was equal to **40 stamps.**

7.2. Single Step RNN Results

The RNN architecture showcased a good performance but might have indication of an overfitting which suggests adding validation set into system that we had or to consider lowering the epochs number, below is showcased the testing regression results:



RNN Validation Regression

We also tried to plot curves of the predictions results obtained to determine the accuracy of the prediction and we have obtained the following plot which displays the predicted and actual ECG values: