POLITECNICO DI TORINO

SEDE DI TORINO

Master degree course in Ingegneria Informatica (Computer Engineering)

Master Degree Thesis

Robustness and reliability of a 1D-ConvNet in trajectory prediction with data augmentation from capacitive sensors

a survey

Supervisors
prof. Luciano Lavagno
prof. Mihai Lazarescu

Candidate
Ivan Airola Sciot
matricola: 271832

Anno accademico 2021-2022

Στους αγαπητούς μου
γονείς, τον αδερφό μου,
τους συγγενείς και
στενούς μου φίλους.

Summary

Nowadays, Machine Learning is an expanding branch of artificial intelligence under computer science. Machine Learning consists in a system, called Neural Network (NN), which can learn something from given data. For this reason, Machine Learning has a wide range of uses, for example from agriculture to spacecrafts.

In this thesis work it will be used to predict a trajectory of a person in a room with the knowledge of data from capacitive sensors, which were used in past as proximity sensors and now they are flourishing on the Internet of Things technology, because they are cheaper than infrared sensors used to detect and track movements. The goal of the thesis is to improve the robustness and reliability of a Neural Network in trajectory prediction, by adding noise to capacitive sensors data in Neural Network pre-process. This noise addition is a form of data augmentation, which leads to have more data, on which the Neural Network can learn.

During data augmentation, it needs to be considered that the Neural Network can learn by heart, if there are too many data. This leads to the phenomena where the artificial mind learns better the data sequence than the ability to "reason" over the data.

The 1Dimensional-Convolutional Neural Network (1D-CNN) is considered for experiments. The 1D-CNNs are simple and compact due to the fact they perform only 1D convolutions (scalar multiplications and addition). 1D CNNs have recently been proposed and immediately achieved the state-of-the-art performance levels in several applications such as personalized biomedical data classification and early diagnosis, structural health monitoring, anomaly detection and identification in power electronics and electrical motor fault detection. Using this setup, I got improved trajectories prediction for testing sets which are some similar and some different to the training and validation set. This result leads to an alternative method to filter the data, which feeds the NN.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Indoor localization

Indoor localization of a person is helpful for services which can improve the life quality and safety, such as tracking the movements of babies and other people with special needs to prevent accidents or to check their wellness, or simpler controlling lights or other smart objects, such as thermostats.

Localization can be implemented in an active or passive way. The active way, which is used in our case, requires an external source of power to operate and can be based on video or pictures, which usually require high computational resources and an adequate lighting.

Indoor localization of a person can be performed with different techniques: infrared sensors, pressure sensors, ultrasound wave sensors, capacitive sensors, cameras and so on [25].

Ultrasound wave sensors measure distance thanks to ultrasonic waves, since the speed of sound is lower than the speed of light, these sensors require a long exposure time, i.e., not moving person.

Infrared sensors use infrared wavelength and heat radiations, moreover they are very sensitive to detect movements.

Pressure sensors use the pressure of air or liquid to detect a variation in ambient.

Capacitive sensors use the variation of capacitance to detect a body or an object, compared to the others, they are tagless, because people don't wear tags; passive, because people don't require interaction with the system; privacy-aware, in fact they don't record video and/or sound; unobtrusive, because they can be placed out of sight. Further details are given in the Section 1.2.

After this comparison, it's worthy to know that our indoor localization experiments are conducted in a 3 m x 3 m room with 4 sensors, each attached to a wall of the room. As shown in Figure 1.1.

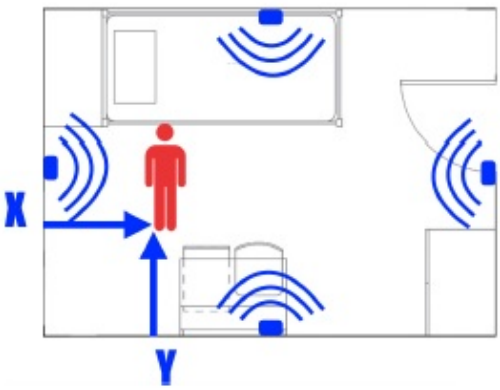

Figure 1.1. Four capacitive sensors centered on the walls of a 3 m × 3 m virtual room in the lab trace the position of a person moving in the space. Source [25].

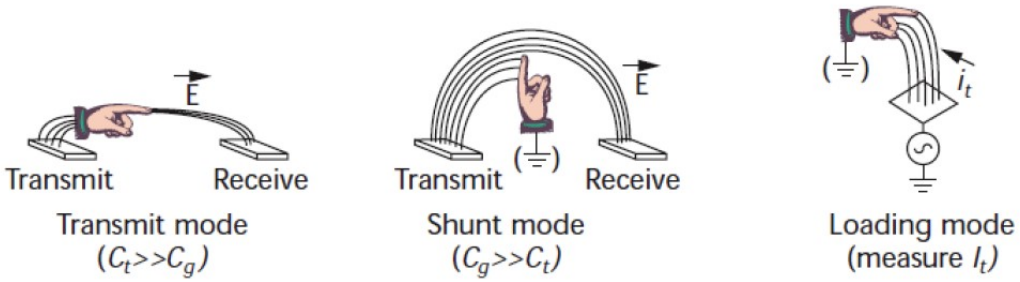

Figure 1.2. Modes of capacitive sensing. Source [22].

1.2 Capacitive sensing

Capacitive sensing has been chosen to determinate a human presence in a room, thanks to their cheap and simple construction.

The capacitive sensors are very popular in the realization of touch screen, and they are used to locate the position of a finger on the screen, by means of the capacitance generated between the finger and the screen [12].

So, a capacitance C exists between two separated electrodes, which is given by the ratio of electrical charge Q and the electric potential V:

$$C = Q/V \tag{1.1}$$

The SI unit of capacitance is the farad (F).

As it is possible to understand, human body is conductive, therefore capacitances exist naturally between people and the environment. The location of a person is obtained by the changes in time of the capacitive coupling between the sensor plate and human body. Furthermore, when the electric potentials on the conductors are time-varying, a current known as the displacement current flows through the capacitor [9].

There are several operating modes to sense the presence of a body with capacitive sensors as explained in [22]: transmit, shunt and loading mode (see Figure 1.2).

The loading mode is implemented for indoor localization, due to its simplicity (the same electrode is used for transmit and receive) and good measurement results. In this mode, the body is like a plate because a capacitor is made usually by two plates. Therefore, the proximity is evaluated by considering that the displacement current increases due to capacitive coupling increment as the body gets closer to the electrode, i.e., sensor plate.

Several studies have been performed on this topic and a discussion about the pros and cons is relevant. The [9] and [30] explain some disadvantages, which are mainly

related to the acquisition and quality of data. The most important problem is the limited tracking range. The general analytical model to determine the capacitance between parallel plates is:

$$C = \epsilon A/d \tag{1.2}$$

Where ϵ is static permittivity of the dielectric between the plates, A is the plate area, d is the distance between the plates.

The sensor plate is continuously charged using a constant voltage V. A higher capacitance C leads to the capacitive sensor to hold a larger charge Q. Another way to increase the capacitance C is by reducing the distance d.

An advantage is that the capacitive sensors can be used also in the presence of non-conductive objects in the line-of-sight, since their influence on dielectric properties is low, therefore they can be mounted inside non-conductive walls. Furthermore, the cheap hardware required (electrodes) and the microcontroller used to inspect the acquired data make capacitive sensors easy to produce and to install.

1.3 Front-ends for capacitive sensors

The purpose of front-end electronics is to:

- Acquire an electrical signal from the sensor
- Tailor the time response of the system to optimize

  a the minimum detectable signal (detect hit/no hit);

  b energy measurement;

  c event rate;

  d time of arrival (timing measurement);

  e insensitivity to sensor pulse shape;

  or some combination of the above.

- Digitalize the signal and store for subsequent analysis [23].

In brief the front-end is the relationship between the electrical signal generated by a sensor, which interacts with physical world, and a digital signal, which is used by computer. Moreover, the front-end should give the stability by reducing the sensitivity of measurements of plate capacitance to environmental electric noise. In our case [17], the operation of the capacitive sensor system consists of:

- indirect measurement of the capacitance of the transducer by measuring the frequency of a relaxation oscillator whose electrical and timing characteristics depend on transducer capacitance;

- use of baseband digital filters to attenuate the noise captured by the sensor, this is valid only in previous work because I don't use digital filters in my front-end;

- use of localization algorithms to infer the position of the person using the data from several loading-mode capacitive sensors within one or several rooms.

We use sensors with different copper clad plate size, as the external capacitor, and 555 integrated circuit in astable configuration, whose oscillation frequency is:

$$1/(0.7(R_1 + 2R_2)C) \tag{1.3}$$

$R_1 = 200k\Omega$ and $R_2 = 560k\Omega$, these choices are a trade-off between the sensor size and its sensitivity.

The frequency is measured by counting the full periods of the oscillator in an interval of 1 s, using an Arduino. Then the measurements are sent to a central node through ZigBee module for a subsequent processing. Our front end is shown in Figure 1.3. The data from the sensors, which follow the characterization of Figure 1.4, can be affected by two types of noise:

- High-frequency noise from appliances and light switches;

- A low-frequency drift, where the Direct Current (DC) component varies very slowly by a significant amount (larger than the useful signal component), due to slow leak of static charges, temperature and humidity changes.

In previous work the research team used low-pass filters to reduce the first noise type, whereas they used high-pass filters for the second noise type. After the filtering phase, the sensor data pass through the localization block, which determines the position of the person within the room.

1.4 Machine learning on circuit board

An innovative technique is to embed the data processing via Machine Learning (ML) directly on circuit board, thanks to Neural Processing Unit (NPU). In our case an introduction of ML on each capacitive sensors, known as Edge Computing, could improve the reliability of the tracking system because a central data processing is a single point of failure and bottleneck. Moreover, we use the ML to implement a filter for data, instead of using a digital median filter as in [17].

It can be stated also that one of the most relevant advantages of ML is its ability to implement a general-purpose function, for example here we use it for tracking inference and filtering the data.

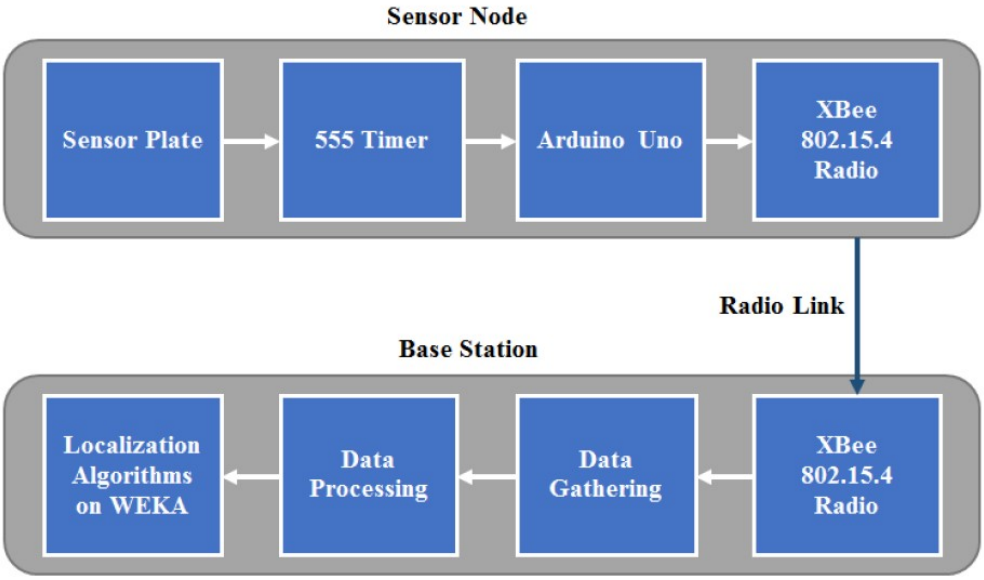

Figure 1.3. Main building blocks of Sensor Node and Base Station. Four Sensor Nodes were connected to a single Base Station. Source [24].

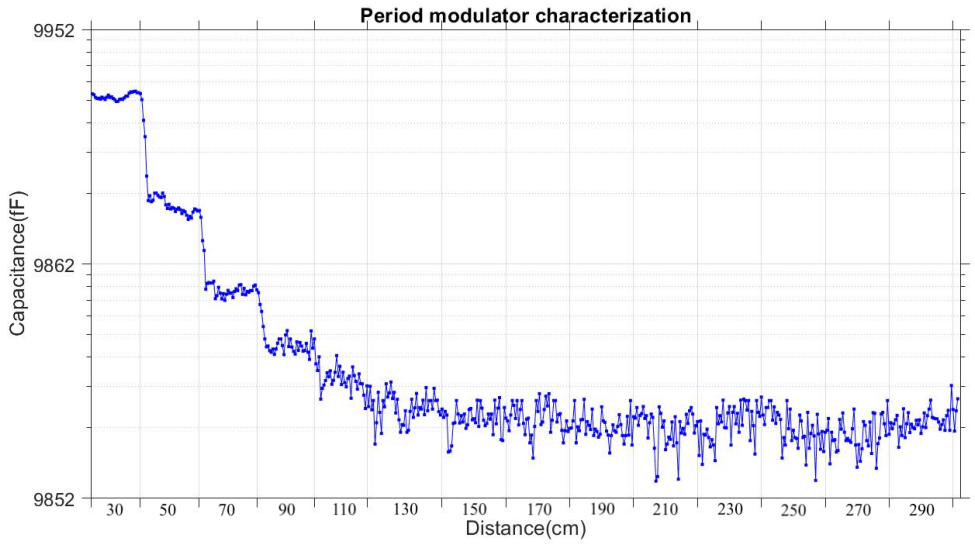

Figure 1.4. Signal characterization for our capacitive sensors.

1.5 Machine learning training technique

In literature, there are different ways for training a Neural Network (NN) [7], for example: Classification, Regression, Clustering and Anomaly detection.

- Classification: draws a conclusion from observed values as categories.
- Regression: is very popular in predictions on numbers and is Supervised Learning. Some algorithms are:
  - Simple Linear Regression Model: It is a statistical method that analyses the relationship between two quantitative variables. The simple in the name means the output is related to a single predictor. This technique is mostly used in financial fields, real estate, etc.
  - Lasso Regression: Least Absolute Selection Shrinkage Operator or LASSO is used when there is a need for a subset of the predictor to minimize the prediction error in a continuous variable.
  - Logistic Regression: It is carried out in cases of fraud detection, clinical trials, etc. wherever the output is binary.
  - Support Vector Regression: In simple regression, the aim is to minimize the error, while in SVR, we adjust the error within a threshold.
  - Multivariate Regression Algorithm: This technique is used in the case of multiple predictor variables. It can be operated with matrix operations.

  - Multiple Regression Algorithm: It works with multiple quantitative variables in both linear and non-linear regression algorithms.

- Clustering: implies classifying data points into specific groups. The data points are grouped for their properties and features.

- Anomaly detection: detects unexpected items or events in a data set. Some areas where this technique is used are fraud detection, fault detection, system health monitoring, etc.

I chose the regression because I didn't need to divide results in groups or classes, or to detect an anomaly.

The regression analysis deals with prediction problems where label space is continuous, e.g., [0,3], and the goal is to find a function $h : X \to Y$, from some function class H, which minimizes error measured using a loss function $L : Y \times Y \to \mathbb{R}$, that is $E[L(Y, h(X))]$.

A loss function L maps decisions to costs: $L(\hat{y}, y)$ defines the penalty for predicting $\hat{y}$ when the true value is y. In regression a squared loss is used:

$$L(\hat{y}, y) = (\hat{y} - y)^2 \tag{1.4}$$

Where $\hat{y}$ is a general predictor:

$$\hat{y} = h(w, b)(x) = w_1 x_1 + .. + w_d x_d + b =< \vec{w}, \vec{x} > +b \tag{1.5}$$

So, the hypothesis class $H = \{h(w, b) : w \in \mathbb{R}^d, b \in \mathbb{R}\}$. In order to find the function h, it is possible to fit a linear function (in $b, w_1, ..w_n$) to a set of points $X = [x_1, .., x_n]$ with their associated labels $Y = [y_1, .., y_n]$. Once the function h is found, this is used to predict the y for new x.

Least squares (LSQ) fitting criterion, shown in Figure 1.5, finds a such function that minimizes sum of square distances between actual y_i in the training set and predicted ones, that is empirical risk minimization. This means to find:

$$\arg\min_{b,w} \sum_{i=1}^{N} (y_i - (b + \langle \mathbf{w}, \mathbf{x}_i \rangle))^2 \tag{1.6}$$

Where N is the number of data tuples and $\mathbf{x}_i \in \mathbb{R}^d$.

The least squares criterion is a convex function of $\mathbf{w}$, then it is sufficient to find $\mathbf{w}$ where the gradient of the function is zero, according to Jensen's inequality:

$$f(\alpha x + (1 - \alpha)y) \leq \alpha f(x) + (1 - \alpha) f(x) \tag{1.7}$$

for all x, y and $\alpha \in [0,1]$. In Figure 1.6 an example of function where Jensen's inequality can be applied. In our case $X = \mathbb{R}^{N \times 4}$ (4 sensor values), so $d = 4$ and the space is a 4-dimensional hyper-plane [26].

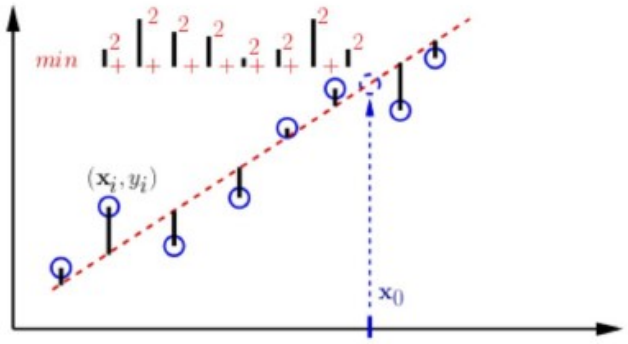

Figure 1.5. Least squares fitting criterion. Source [26].

Figure 1.6. An arch between 2 points of a convex function. Source [26].

Chapter 2

Neural network for sensor data processing

As proven in previous works [24] and [25], neural networks (NN) can be precise enough to infer the trajectory of a person in a room. In fact, Machine Learning (ML) classifiers can effectively mitigate sensor data variability and noise due to environmental conditions. In this chapter are presented some of the main topics about ML, which have been used in the thesis work.

2.1 Machine learning basics

The term "Machine Learning" was introduced by Arthut samuel in 1967 in the article "Some Studies in Machine Learning Using the Game of Checkers" [19]. He described the ML as: "The field of study that gives computers the ability to learn without being explicitly programmed."

Only during the '90s, a more formal definition of the term appears thanks to Tom Mitchell in [13]: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

This quote means that a computer is good at a task without manually building its competence, so the computer changes its parameters (experience) to perform a task without human interaction.

As explained in [26] and [21], a ML problem can be faced as:

- Supervised learning: where inputs x_i and output y_i samples are given to the learning algorithm that has to learn the function f that relates input to output such that $f(x_i) = y_i$. Depending on the type of y_i values it can be distinguished in other two kind of supervised learning problems:
  - Classification problem: y_i are discrete and finite values.

  - Regression problem: y_i is a continuous variable.

  The function f is also named as h and it is called hypothesis [26]. When a new input set is given, the obtained hypothesis h can be used to predict the output values.

- Unsupervised learning: a collection of x_i values is given to the learning algorithm which derives the structure from them without knowing the output y_i. This can be obtained in two ways:

  - Clustering approach: used to group the samples in clusters that share some features.
  - Non-clustering approach: used to find a structure in a chaotic environment. For example the "Cocktail Party Problem", i.e. identifying individual voices and music from a mesh of sounds at a cocktail party [20].

Since supervised learning has been the technique used in the thesis work, a deeper explanation is given.

In supervised learning there are two main steps:

1. Train: a step in which the algorithm that has to learn the hypothesis function that describes the data used, through applying the learning algorithm with different parameters on the training set. Some algorithms requires also a validation phase, which can reduce selection bias.

2. Test: a step in which the generalization capability of the algorithm is tested.

In the training step, a set of input and label samples (x_i, y_i), called training examples, creates a collection called training data set, which is supplied to the algorithm. The performance is evaluated via the loss function $L_{\text{train}}(Y, h(x))$, that measures the difference between the labels and the outputs of the algorithm [26].

The goal is to minimize the loss function in order to have the output values near the label. When the validation is used, also a validation set, smaller than the training one, is added in this procedure to identify the minimum of the loss function by using validation error $L_{\text{val}}(Y, h(x))$. The weights in the loss function are updated after each epoch (i.e., the number times that the learning algorithm will work through the entire training dataset) which may include one or more batches. A batch is the number of samples to work through before updating the internal model parameters and it is useful when there is not enough memory in computer to train the whole training dataset in a once.

At the end, the model with lowest validation error is chosen. In the testing step, a test set is used, made by a lower number of samples respect to the training set. This time labels are hidden to the algorithm, because we are checking the ability of the algorithm to act on unknown data, that is called generalization [26].

Also in this step the loss function $L_{\text{test}}(Y, h(x))$, called simply "error", is evaluated through the comparison between the computed outputs and the known labels. The test error is usually higher than the training one on which the model learns.

Test error can be similar to the training one, only if the model has reached a good generalization.

It is important to consider the complexity of the model: it should match the complexity of the loss function which describes the relationship between input and output data, to have good performance.

Alternatively, generalization is poor and two cases can happen [26]:

- Underfitting the data: when model complexity is lower than the actual function. This means the model doesn't fit the data well enough. So, training error and test error are high.

- Overfitting the data: when the model is too complex than the actual function. This leads to an excellent performance by predictor on the training set, but its performance on the true situation is very poor. So, training error and test error are different.

Among all the machine learning algorithms, neural networks have been chosen to analyse the data collected from capacitive sensors.

2.2 Neural networks

Neural networks can be utilized in various scenarios such as image recognition, speech recognition and machine translation software [29]. This makes them useful for the thesis objective: infer the movements of a person in a room.

Neural network is a machine learning algorithm which relies on brain model [15]: the brain is composed by a basic element, called neuron (Figure 2.1), which gives outputs to other neurons with axons which terminate in synapses, and takes inputs from other neurons with dendrites. So, in the neural network the artificial neuron uses non-linear functions to elaborate inputs from other artificial neurons taking into account the weight of each input, to perform an output to other artificial neurons. In Figure 2.2 an artificial neuron with three inputs is shown.

The circle is the computational region, made by two main parts: the sum between all the weighted inputs $\vec{w}^T\vec{x}$ and the bias b, and the activation function, represented as the sigmoid function of the first part of computation z, that evaluates the neuron output.

The weights w and the bias b are chosen according to perceptron algorithm (Figure 2.3), a linear binary classifier which can be extended to more than two dimensions classification and non-linearly separable problems, for example, via polynomials of order $d :< \vec{x}, \vec{x'} >^d$ [26].

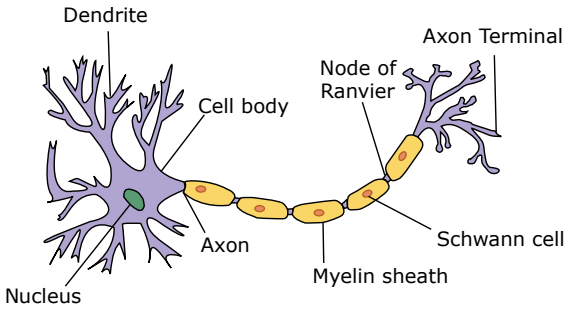

Figure 2.1. A human neuron. Source [27].

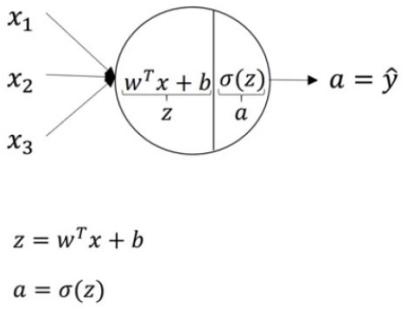

Figure 2.2. An artificial neuron with three inputs. Source [14].

initialize $w = 0$ and $b = 0$
repeat
  if $y_i\,[\langle w, x_i\rangle + b] \leq 0$ **then**
    $w \leftarrow w + y_i x_i$ and $b \leftarrow b + y_i$
  end if
until all classified correctly

Figure 2.3. Perceptron algorithm. Source [26].

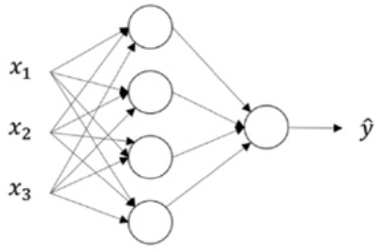

Figure 2.4. Neural Network with one hidden layer. Source [14]

.

In general, the structure of a neural network presents two layers: the input nodes is the first layer, called input layer; the computational node is the second layer, called output layer. In the middle, intermediate layers can be added, called hidden layers, an example is reported in Figure 2.4. This type of structure is called feedforward neural network.

As in human brain, the artificial neurons must be activated and connected each other, therefore they can be turned on by activation functions, which the most common are:

- Sigmoid function: defined as $\sigma = 1/(1 + e^{-z})$ is generally used in the hidden layers.
- ReLU function: Rectified linear unit, defined as $a = \max(0, z)$ is generally used in the output layers.

After creating the network, all weights and biases have to be initialized, usually randomly, then they are tuned during training to optimize network performance.

As introduced in Section 2.1, the performance of the network is evaluated through the loss function, that shows how much labels and outputs are different, in the best case the difference tends to reach the 0.

Optimization algorithms are needed to minimize the loss function. They are used together with the computational technique, known as back-propagation, which is the reverse automatic differentiation. The back-propagation used for evaluation of the gradient ∇I is as fast as evaluation I.

Back-propagation works as follow: compute the gradient of the loss function respect to each weight, computing the gradient one layer at a time, iterating backwards from the last layer to avoid redundant calculations of intermediate terms in the chain rule, which is the forward-propagation technique.

One interesting parameter of the optimization algorithm is the learning rate, by changing its value the optimization algorithm converges in different ways: if its value is too small (the minimum of loss is reached slowly), on the other side if it is too high (the minimum of loss is gone beyond and the algorithm convergence fails) [26].

In the thesis, Mean Square Error (MSE) and Euclidean Distance are used to estimate the error and the optimization algorithm is Adamax.

The Mean Square Error is defined as:

$$MSE = \sum_{i=1}^{N} \frac{(y_i - \hat{y}_i)^2}{N} \tag{2.1}$$

where y is the label, $\hat{y}$ is the output, N is the number of samples. The unity of measurement of MSE is: m^2. While, the Euclidean Distance is defined as:

$$EuclideanDistance = \sqrt{\sum_{i=1}^{N} (prediction_i - slicedTestLabel_i)^2} \tag{2.2}$$

where N is the number of data. The unity of measurement of Euclidean Distance is: m. For having an average distance, the mean is applied to the Euclidean Distance in our case.

Adamax, an optimization algorithm, is a variant of Adam based on the inifinity norm. Moreover, sometimes Adamax is better than Adam, specially in models with embeddings. The initialization is:

```
m = 0  # Initialize initial 1st moment vector
v = 0  # Initialize the exponentially weighted infinity norm
t = 0  # Initialize timestep
```

The update rule for parameter w with gradient g is:

```
t += 1
m = beta1 * m + (1 - beta) * g
v = max(beta2 * v, abs(g))
current_lr = learning_rate / (1 - beta1 ** t)
w = w - current_lr * m / (v + epsilon)
```

Similarly to Adam, the epsilon is added for numerical stability (especially to get rid of division by zero when v at time t is 0).

Opposite to Adam, the sparse implementation of this algorithm only updates variable slices and corresponding to m and v terms at time t, when that part of the variable was used in the forward pass [10].

2.3 Neural networks: types and topologies

There are many types of Neural Network in literature and they differ by the function they perform [26]. Some relevant types are:

- Feedforward Neural Network: is one of the fundamental neural network type. The data flow goes unidirectionally through each layer in sequence until it reaches the output layer. There can be a single layer or multiple hidden layers. This neural network uses the classification activation function in general. The applications include classification, image recognition and so on.

- Convolutional Neural Network (CNN): is a kind of multilayer perceptron. It has a first layer called convolutional layer. There are one or many convolutional layers, and these layers can be fully connected or pooled. Convolutional layers contain filters which reduce the processing parameters. This type of neural network is deployed for semantic parsing, video and image classification, etc.

- Recurrent Neural Network (RNN): the output of a specific layer is fed back to the input. The first layer of RNN is similar to the feedforward neural network. As the data moves across layers, each layer can remember some information

acting as a memory cell. Its advantage is the ability to relearn during the backpropagation, if RNN predicts inaccurately, and to predict again. This process continues until the prediction is enough accurate. RNN is used for text-to-speech conversion. In practical applications Long Short Term Memory (LSTM) is the most used. More details can be found in [8].

- Modular and Sequence Neural Network: a modular neural network contains many networks which work independently to achieve a common goal. In fact, a large complex task can be splitted into subtasks and processed faster in parallel using modular networks. A sequence network has two recurrent neural networks. One is the input encoder, and the other is the output decoder. It is useful when the input data size and output data size are different. Some applications can be natural language processing, translation and chat robots.
- Multilayer Perceptron Neural Network: is a fully connected network made by multiple hidden layers.

2.4 1 Dimension - Convolutional Neural Network

In my thesis work, the 1 Dimension – Convolutional Neural Network (1D-CNN) has been selected for the experiments. So, a deeper explanation is given in this Section.

The 1D-CNN is the most suitable convolutional approach for indoor tracking, my case of study. They are known for their effectiveness on sequences of data analysis, such as semantic parsing and trajectories prediction. This type of neural network overcomes some limitations of the feedforward networks.

The name of this network derives from the mathematical operation used in their layers called convolution (Figure 2.5), which is presented in (2.3), where f and g, input and kernel respectively for us, can be two functions such that $(-\infty, +\infty) \rightarrow \mathbb{R}$ [8].

$$(f \star g)(x) = \int_{-\infty}^{+\infty} f(t)g(x-t)\,dt \tag{2.3}$$

In [16] a good clarification of convolutional neural networks is exposed by underlining the basic concepts and the layers required to build this kind of network.

The convolutional neural network architecture is composed of three types of layers:

- Convolutional layers: they compute the output of neurons by using the scalar product between the input and the kernel, the result is processed through an activation function and the output feature map is created. There are also the same weights on the outgoing edges for each input variable.
- Pooling: performs subsampling of the given input to reduce the number of parameters and computational effort.

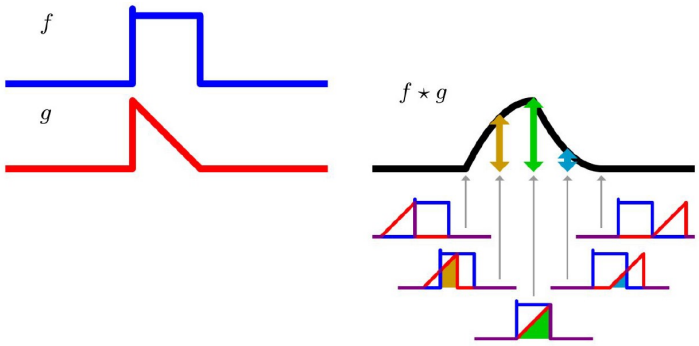

Figure 2.5. A convolution between 2 signals. Source [26].

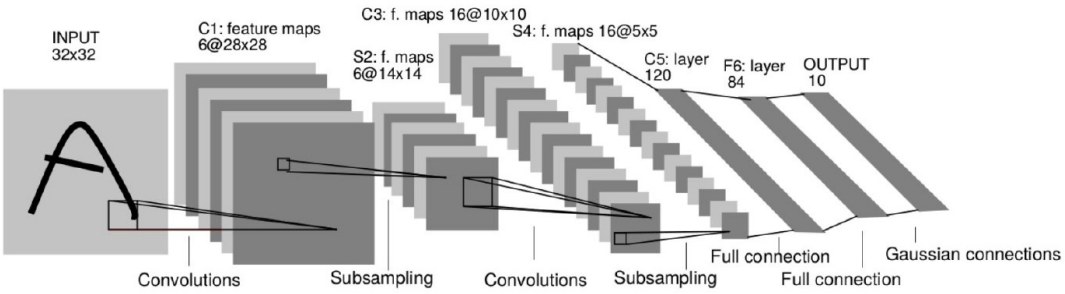

Figure 2.6. A CNN architecture. Source [26].

- Fully connected layers: are used in the end for classification. ReLU is used between these layers to improve performance. Each edge between neurons has a different weight.

A CNN architecture is shown in Figure 2.6. In General, CNNs are trained in supervised learning fashion, as explained in Section 2.1. This time, the parameters related to the gradient of each layers (convolutional and fully-connected layers) are computed. The parameters are update until a certain stopping criterion, which is the Mean Square Error in our case, to prevent overfitting.

The number of hidden CNN layers and kernel size and filter size per CNN layer can be tuned to obtain the desired structure. These hyperparameters, along side with pool size, the number of fully connected layers and the choice of activation functions are the main hyperparameters of this kind of network.

Since our sequential data are 1D arrays, in the CNN the 2D matrices are simply 1D arrays for both kernels and feature maps [11]. As explained in [8], even if the convolution across 1D temporal sequences tries to share parameters in time, this behaviour results superficial, because only a small number of nearby segments of the input is considered in computing the output.

In my thesis work, a window size has been introduced for the data to be passed to CNN, which receives all the tuples that belong to the temporal window, which are labelled with the position coordinates of the window size middle. In Figure 2.7 the temporal window behaviour is shown.

The objective is to analyse these tuples in sequence and infer the position coordinates. Because of the position refers to the middle of a window, the predicted trajectories starts half a window after the beginning of reference trajectory and half a window before its end.

The process is computed by the filters and the kernel that are defined in the CNN: the kernel moves inside the window by one tuple to the right at a time, filters analyse the content of the kernel. When the edge of the window is reached, the window moves: the very first tuple is discarded, and a new tuple is considered.

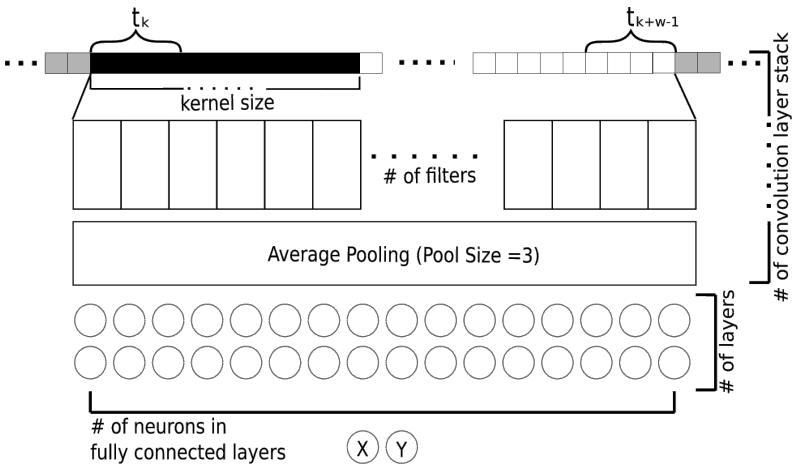

Figure 2.7. Network structure and data access for the convolutional network. Source [25].

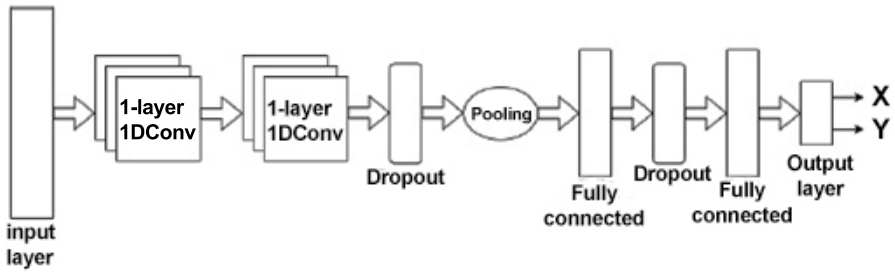

Figure 2.8. 1D-CNN used for the experiments.

The new window obtained is analysed in the same way described before.

The window was set to 5 s. All the trainings are repeated for 1000 epochs. Each neural network configuration of hyperparameters experiment was executed for 10 times, or 20 times, if the configuration was promising, to be sure that results were not casual.

The architecture of 1D-CNN proposed by the research team and used for experiments is shown in Figure 2.8.

It is composed of two convolutional hidden layers, followed by a dropout layer (a regularization technique which can reduce overfitting and consists in turning off a certain percentage of neurons temporarily every epoch) of 0.5, a pooling layer and two fully connected hidden layers of 64 neurons each, separated by a dropout layer, then an output layer.

Chapter 3

Neural Network training

In this chapter some topics about training, already discussed in previous Sections 1.5 and 2.1, are extended and integrated with the data augmentation topic.

3.1 Neural Network training techniques

As it was introduced in Sections 1.5 and 2.1, Neural Network can be trained with regression fashion using a loss function which has to be minimize.

So, in my case the data used in experiments are divided in three parts: training data (60 % of the whole data), validation data (20 % of the whole data) and test data (20 % of the whole data).

If data permits, i.e. the data are enough, a k-fold cross validation can be implemented.

K-fold cross validation consists into split the training data in k groups of training and validation data which vary at every run [26]. Anyway, in my thesis work, I have been kept a static division of the data in training, validation and testing because I don't have a limited quantity of data.

3.2 Data augmentation

Data augmentation techniques are used to increase the amount of data by adding slightly modified copies of already existing data or newly created synthetic data from existing data [1].

Training a neural network with a small data set can cause the network to memorize all training examples, in turn leading to overfitting and poor performance for Neural Network [5].

The data augmentation from slightly modified copies of already existing data can be achieved with geometric transformations (for images: flipping, colour modification, cropping etc.) and/or adding noise (for images and signals). The other

method, which is creating synthetic data from existing data, can be done using a generative adversarial network.

Due to the fact the data are signals, the noise addition has been used in thesis work, and a further explanation is given in the next Section 3.3.

3.3 Data augmentation: noise addition

As discussed in Section 3.2, in order to have an input space smoother and easier to learn, it is good to add noise to inputs during training. In this way, the robustness of the Neural Network is improved, resulting in better generalization and faster learning.

The poor performance for neural network associated to a small training data set is related to a mapping from the specific inputs to their equivalent outputs. A way to solve the mapping problem is to add noise. "Many studies [...] have noted that adding small amounts of input noise (jitter) to the training data often aids generalization and fault tolerance." [18].

We may expect that noise addition can reduce performance. As it is stated in [3]: "Heuristically, we might expect that the noise will 'smear out' each data point and make it difficult for the network to fit individual data points precisely, and hence will reduce over-fitting. In practice, it has been demonstrated that training with noise can indeed lead to improvements in network generalization."

The noise addition during the training of a neural network has a regularization effect and, as result, improves the robustness of the model. Furthermore, it has been shown to have a similar impact on the loss function as the addition of a penalty term, as in the case of Ridge Regression:

$$\frac{1}{2}||\mathbf{y} - \mathbf{X}\mathbf{w}||_2^2 + \frac{\alpha}{2}||\mathbf{w}||_2^2 \tag{3.1}$$

Where $+\frac{\alpha}{2}||\mathbf{w}||_2^2$ is a regularization term, controlled in strength by the hyperparameter α, and the other part is the loss function. Regularize prevents a model to become too complex [26] [2].

In fact, adding noise increases the size of the training dataset. Each time a training set is created, a random noise is added to the input data, making them different every time. In this way, a simple form of data augmentation is obtained [28].

In conclusion, it can be state that the noise is as thought new samples are being drawn from the domain in the vicinity of known samples, smoothing the structure of the input space. This smoothing may mean that the mapping function is easier for the network to learn, resulting in better and faster learning.

Chapter 4

Experimental results

Here I present the results obtain in my experiments about trajectory prediction with unfiltered pre-processed data and with data which have noise addition.

4.1 Goal

The purpose of my thesis work is to study how training data augmentation using different types of noise affects the quality of the inference of a 1-Dimension Convolutional Neural Network (1D-CNN) and its generalization capability using capacitive sensor data collected in different environmental conditions.

In order to achieve the goal two types of noise were selected: white Gaussian noise and pink noise. Because white Gaussian noise is commonly found in environment, while the pink noise emulates the drifts of the capacitive sensor.

In the Section 4.2, my techniques and results are presented. Meanwhile my work flow is shown in Figure 4.1.

4.2 Techniques and results

Having Started from the codes used to build the convolutional neural network and to pre-process data in [25], I optimized them for my study case, for example I implemented a batch training, and created other codes for specific analyses.

Batch training was implemented in Python with Keras API function fit() which, in this case, takes as arguments: 2 batch generator functions, one for training and one for validation, and a custom callback class which shuffles the order of the data sets before each epoch. This shuffle is an useful randomization for the Neural Network (NN) to generalize over test sets. So, every generator passes to the NN 1 data set in each batch for a total of 6 batches per epoch, because I fed the NN with 6 data sets (1 without noise addition, 5 with noise addition obtained by the data

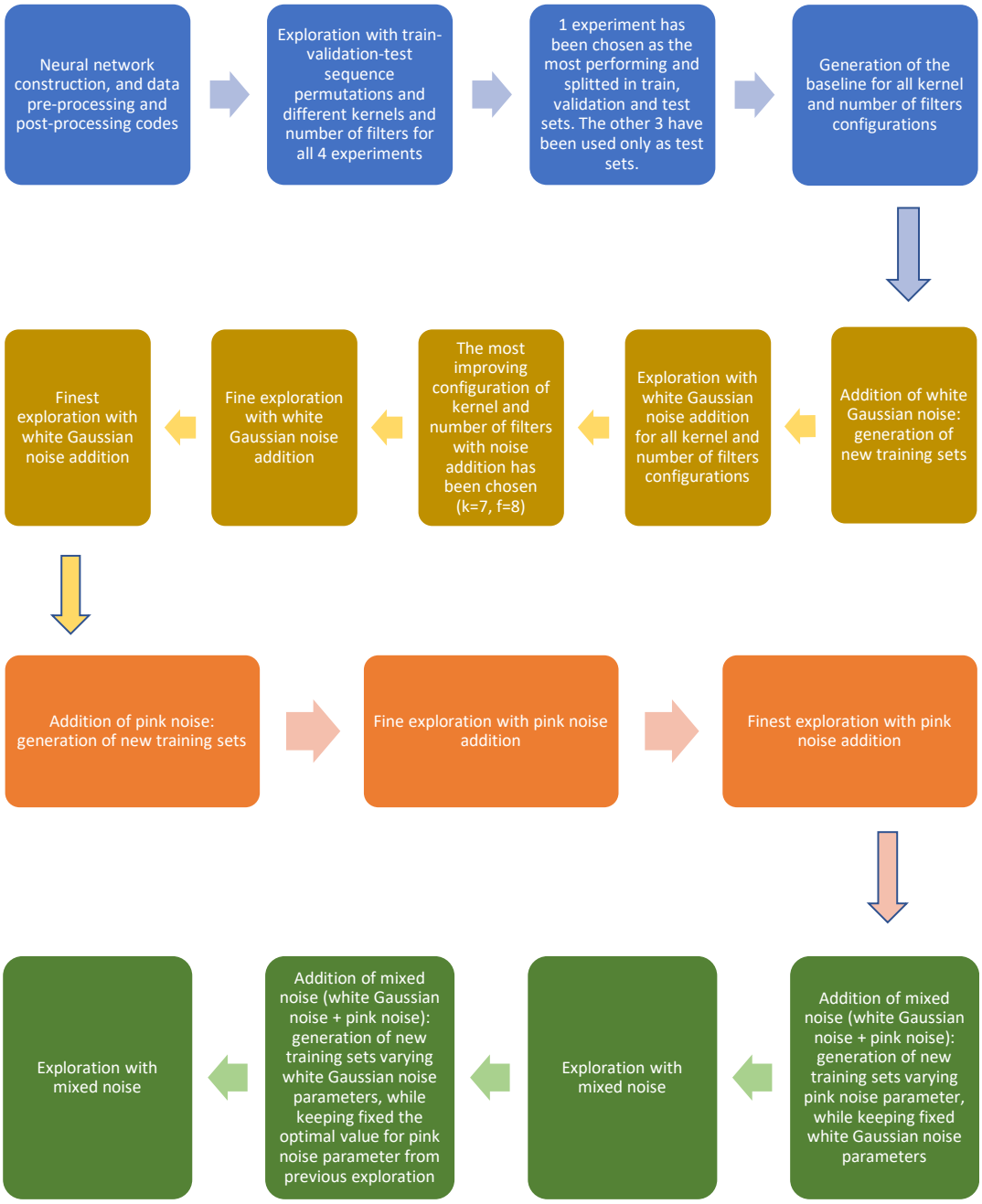

Figure 4.1. Work flow.

augmentation). A later explanation about noise addition will be given, because the first experiments were conducted with 1 training set without noise addition.

The specific analyses codes were made in MATLAB, a tool for numeric computing, and they focused on pre-processing and post-processing data in an automated style, for example datasets, table and plots generation.

In machine learning, data pre-processing is essential due to the fact the neural network needs readable data, which means that data had to be in tuples and in a certain range of values for my work. Then the data pre-processing was made with the aid of MATLAB. And the most relevant parameter used in pre-processing on data, which fed the network, was: sampling frequency; meanwhile the most interesting parameter for the NN was: window input time.

The experiments with capacitive sensors had been conducted in a 3 m x 3 m room. The capacitive sensor data has been resampled at a frequency 5 Hz according to the sampling frequency to level occasional gaps and jitter in the experimental data. The data were then rescaled between 0 and 1, independently for each sensor. These techniques are useful for the Neural Network (NN) to improve its ability to learn [4].

A input window size of 5 s was proved to be the best configuration, having included a small enough number of samples which fed the NN, while also having allowed the NN to generalize the trajectory of the person. In fact, a higher value of window input time makes the NN to memorize the trajectory, because the NN sees trajectory segments too long. Furthermore, I can define the window input time as a batch for the NN because the dataset is splitted in many parts and the NN sees only a part each time, as explaned in Section 2.4, and according to previous experiments a small batch size performs better than large batch, if a low learning rate is used [6].

The next step was to build the 1D convolutional neural network, which was composed by 2 convolutional layers, followed by a 50 % dropout, then a pooling layer with size 3 and a fixed size Multilayer perceptron network (of two layers with 64 neurons each) with a 50 % dropout in the middle, before the output layer.

I used experimental data available from 4 different experiments, of which I chose one, set A, as training and validation sets. While the other sets: B, C and D were used only as test sets. So, I splitted set A into the training set (60 % of the whole experiment), validation set (20 % of the whole experiment), and testing set (20 % of the whole experiment). In order to decide which one of the 4 experiments was the best in terms of sensors measurements, I trained and tested these 4 experiments separately, having considered also all the permutation of the train-validation-test sequences. Having analysed the measurements, test sets A and B had similar measurements, while test sets C and D had similar measurements.

Later, to obtain the results for a baseline, I trained and tested 10 times the neural network for each combination of kernel and filters hyperparameters for a complete Design Space Exploration (DSE).

Table 4.1. Design Space Exploration results for 1D Convolutional Neural Networks - Mean Square Error (MSE) for different convolutional kernel sizes and number of filters.

1Dconv kernel size	**1Dconv filters**	**Test set A MSE** (m^2)	**Test set B MSE** (m^2)	**Test set C MSE** (m^2)	**Test set D MSE** (m^2)
3	8	0.053150	0.420070	0.442975	0.130062
	16	0.054600	0.478832	0.468392	0.146585
	32	0.063547	0.462279	0.398278	0.157259
	64	0.066706	0.480316	0.428212	0.17279
5	8	0.055291	0.415967	0.422625	0.139609
	16	0.060611	0.407964	0.389265	0.150896
	32	0.063744	0.418670	0.335601	0.171747
	64	0.068565	0.462485	0.509757	0.170423
7	**8**	0.062883	0.494872	0.519451	0.137056
	16	0.062822	0.413312	0.418252	0.155779
	32	0.064303	0.432029	0.373408	0.167687
	64	0.065798	0.501124	0.408473	0.148187

I show in Table 4.1 the results of the configurations of each hyperparameter for the baseline. For all results, I picked from the runs the best Mean Square Error (MSE) value for test set A because this method is simple and effective. In fact, I tried also to pick the lowest average of MSE over the test sets, but as it can be seen by the MSE values: test sets B and C give a big contribution to the average because they are more or less 10 times the value of test set A. So an average of MSE over the test sets gives as the best results, those with low MSE on test sets B and C. Moreover, the NN tends to overfit if the lowest average technique is chosen.

Table 4.1 shows that by increasing the size of the kernel and number of filters, in general there is a performance degradation, because the MSE increases over test set A and test set D, which have similar sensors measurements. Meanwhile, test sets B and C don't degrade by increasing the size of kernel and number of filters. In fact, for example, the configuration of kernel=3 and filters=32 shows an improvement respect to kernel=3 and filters=16 on test sets B and C. I show in Figure 4.2 the inference of the X and Y coordinates for the chosen combination of hyperparameters (k=7, f=8) respect to the ground truth. As we might expect from MSE results, the predicted trajectories for test sets A and D follow the ground truth, or at least the direction of the ground truth, but the predictions for test sets B and C don't follow the ground truth and, sometimes, they even go in the opposite direction of the ground truth.

After I got the baseline which was used to compare the neural network performance, I added white Gaussian noise to the capacitive sensors measurements using

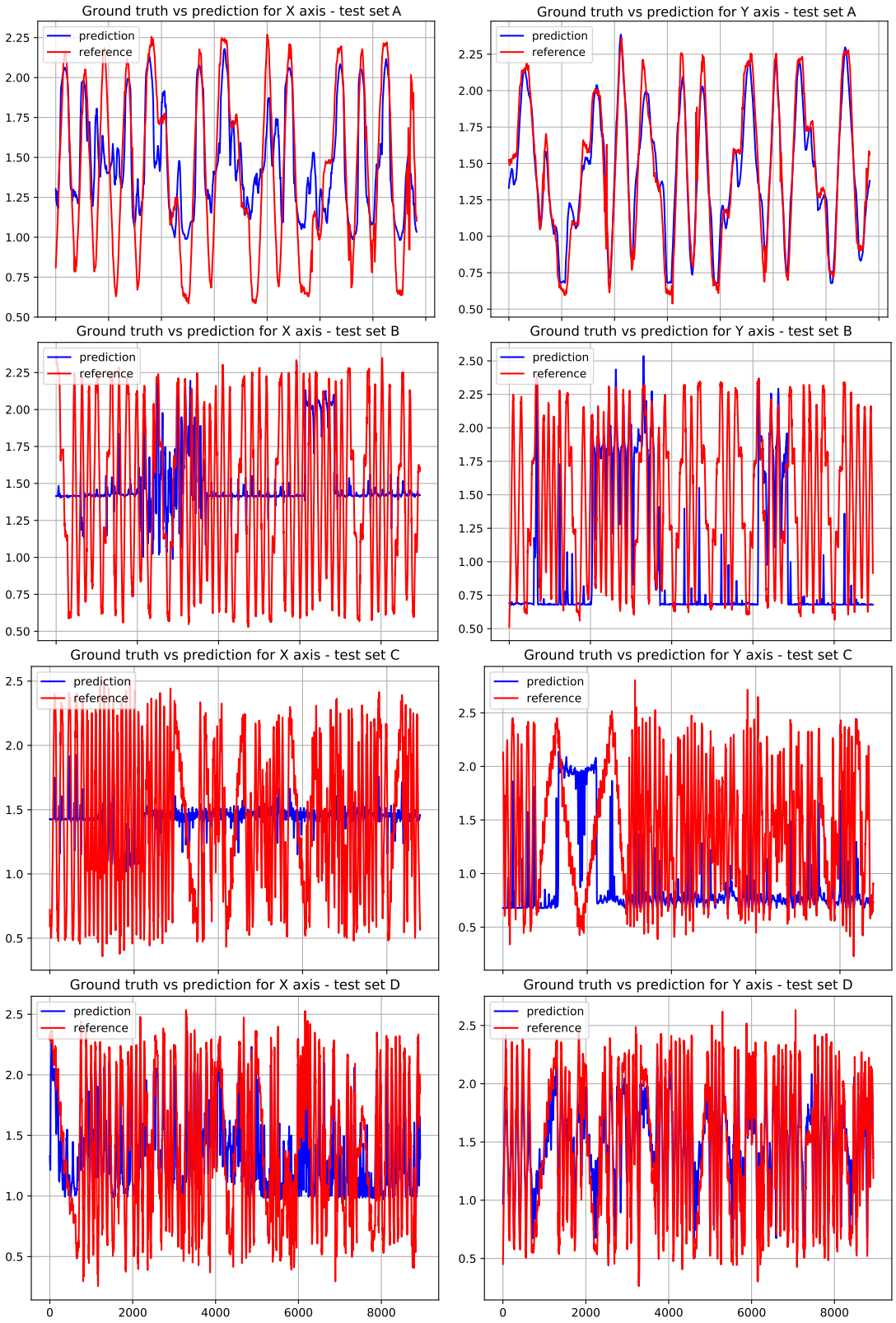

Figure 4.2. Ground truth vs prediction for baseline with configuration kernel=7 and filters=8.

MATLAB. I added the noise after having rescaled the data, then I rescaled again because the noisy data can be outside [0,1] interval [5] [4]. In this way I generated 5 extra training and validation datasets, which were used in batch training with a dataset without added noise used for the baseline. For the first noise exploration, the white Gaussian noise has been added and two parameters, variance and amplitude, can describe this noise. The selected variance values were 0.1 and 0.5, and the selected amplitude values were 0.0001, 0.0005 and 0.001. The variance was chosen taking into the values distribution of a white Gaussian noise in a certain interval, in fact the effect of a low variance (0.001) is to have many values near the edges of the interval, and this is not good for having small modification of the signal. The amplitude was chosen comparable with the signal level of a capacitive sensor at long distances, which can be below 0.01 %.

Then, I chose and fixed the best performing combination of kernel and filters which is kernel=7 and number of filters=8, having considered the improvement respect to the baseline. Then, I refined the exploration of the Gaussian noise parameters with variance values 0.1, 0.25, 0.5, 0.75, and 1, and amplitude values 0.0001, 0.00025, 0.0005, 0.00075, and 0.001.

In Figures 4.3, 4.4, 4.5 and 4.6 I show the MSE results in a 3D surface representation. The addition of noise leads to an improvement respect to the baseline for test sets A, B and C. On the other hand, test set D has only some configurations where there is an improvement compared to the baseline. These configurations are: amplitude 0.0005 and variance 1, amplitude 0.001 and variance 1, amplitude 0.00075 and variance 0.75, amplitude 0.00075 and variance 1, amplitude 0.001 and variance 0.25. Then, I refined the search in the area around the point (variance=0.5, amplitude=0.0001) where test sets A, B, and C have a minimum, but not the test set D. I found an improvement for test set D, so I repeated the refinement around the points where I have found improvements for all the test sets. The results of this last search are in Figure 4.7, alongside with the configurations shown in Table 4.2, in particular the variance=0.55 and amplitude=0.00037 is considered as the most performing configuration for all the test sets.

This confirms also the theory about noise addition 3.2, because a variance=0.55 lets the white Gaussian noise to generate noise samples well distributed inside a range.

Worthy of notice is that in these refinements I trained and tested the neural network for 20 times taking the best result considering the stochastic nature of the training process.

In Figure 4.8 I show the ground truth and prediction for the best white Gaussian noise found. The NN can predict well the ground truth for test sets A and D, but the NN cannot follow the ground truth precisely for test sets B and C, in fact often a changing in position from far to near the capacitive sensors is not smoothly predicted.

Since the addition of white noise was giving a good performance for the neural

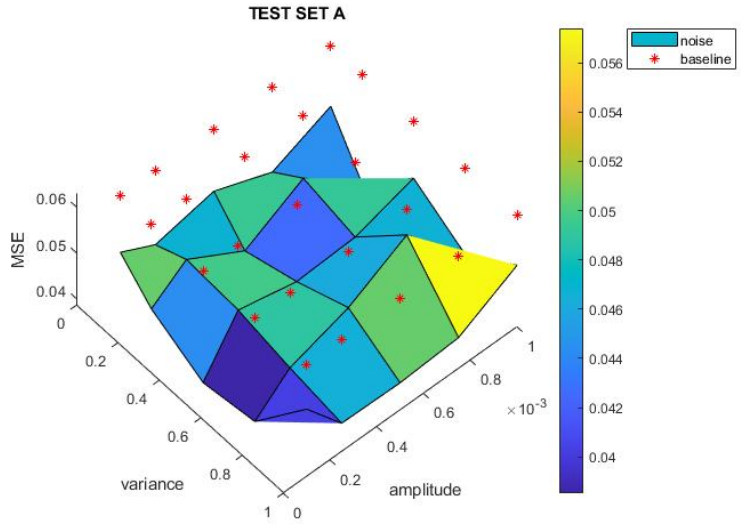

Figure 4.3. Preliminary Design Space Exploration for white Gaussian noise addition in test set A.

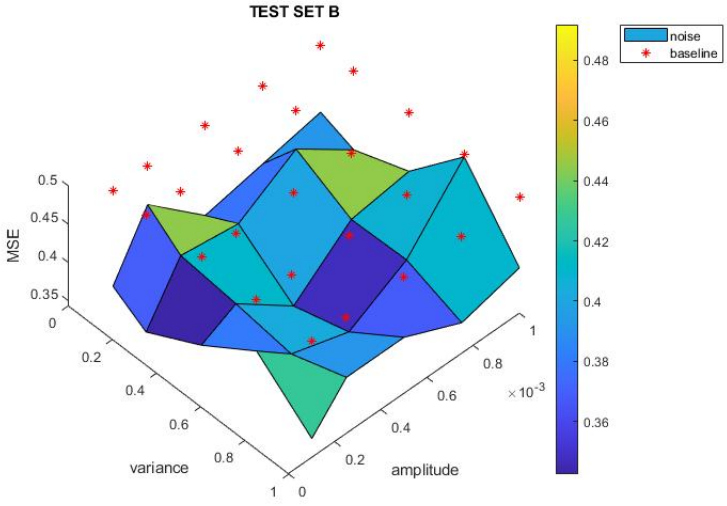

Figure 4.4. Preliminary Design Space Exploration for white Gaussian noise addition in test set B.

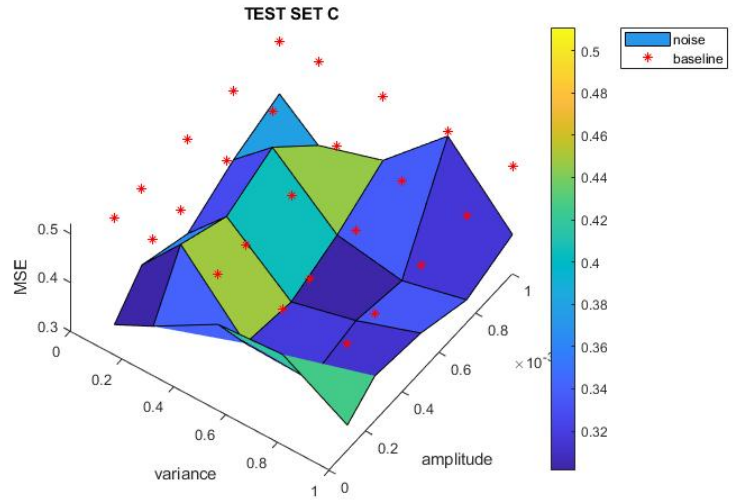

Figure 4.5. Preliminary Design Space Exploration for white Gaussian noise addition in test set C.

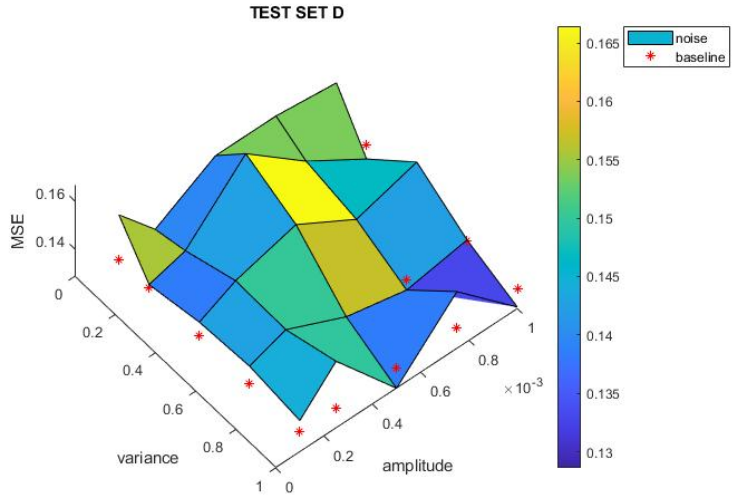

Figure 4.6. Preliminary Design Space Exploration for white Gaussian noise addition in test set D.

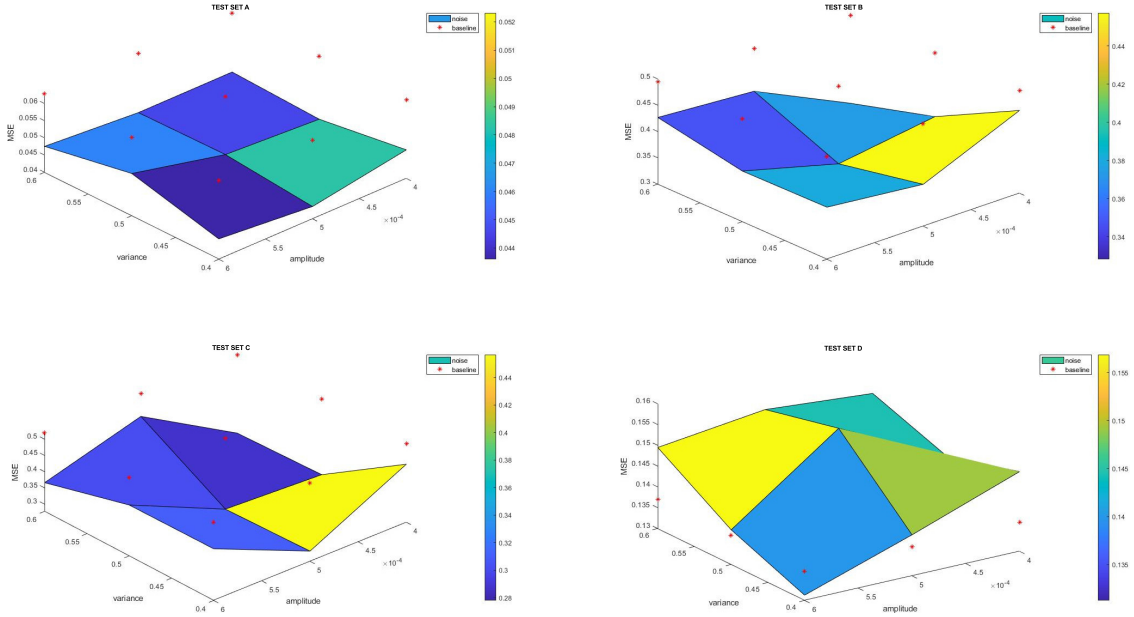

Figure 4.7. Design Space Exploration around variance=0.5 and amplitude=0.0005 for white Gaussian noise addition in test sets A, B, C and D.

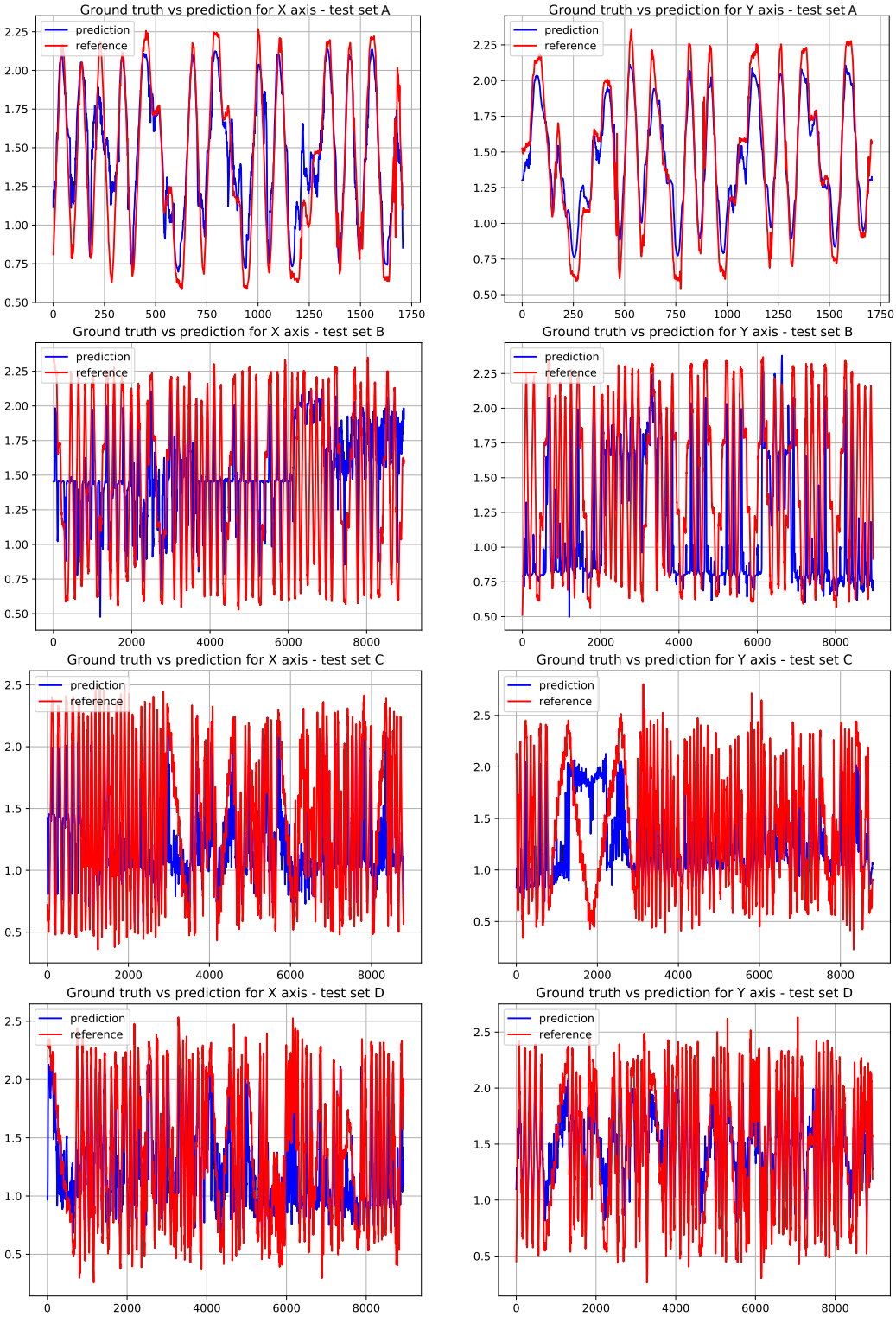

Figure 4.8. Ground truth vs prediction for the best white Gaussian noise found (variance=0.55 and amplitude=0.00037).

Table 4.2. Best configurations for white Gaussian noise addition.

Test set A MSE (m^2)	var 0.525	**var 0.55**	var 0.575
Baseline	0.062883	0.062883	0.062883
ampl 0.00037	0.050704	**0.037540**	0.046611
ampl 0.0004	0.041287	0.047815	0.050409
ampl 0.00043	0.039912	0.045774	0.048816

Test set B MSE (m^2)	var 0.525	**var 0.55**	var 0.575
Baseline	0.494872	0.494872	0.494872
ampl 0.00037	0.449773	**0.342807**	0.353496
ampl 0.0004	0.418419	0.433316	0.491054
ampl 0.00043	0.295110	0.372116	0.400246

Test set C MSE (m^2)	var 0.525	**var 0.55**	var 0.575
Baseline	0.519451	0.519451	0.519451
ampl 0.00037	0.500763	**0.268444**	0.301767
ampl 0.0004	0.386215	0.334298	0.501942
ampl 0.00043	0.275459	0.368593	0.423129

Test set D MSE (m^2)	var 0.525	**var 0.55**	var 0.575
Baseline	0.137056	0.137056	0.137056
ampl 0.00037	0.137427	**0.133188**	0.149449
ampl 0.0004	0.126940	0.131894	0.136977
ampl 0.00043	0.127307	0.155130	0.135770

network, I continued by adding pink noise. Pink noise has higher amplitude for lower frequencies, so it emulates the drift we noticed for these sensors.

I explored the following pink noise amplitudes: 0.0001, 0.00025, 0.0005, 0.00075, 0.001. In Figure 4.9, I show the MSE results. The pink noise addition gives a consistent improvement with an amplitude=0.0001 for all the test sets. So, I proceeded to refine the area around amplitude=0.0001 with the results shown in Figure 4.10. Therefore, I found amplitude=0.000085 which brings also an improvement for all the test sets.

In Figure 4.11 I show the ground truth and prediction for the best pink noise found. The NN can predict well the ground truth for test sets A and D, as also stated for white Gaussian noise, but this time the pink noise addition leads to a better tracking when there is a changing in position from far to near the capacitive sensors.

So, I tried to mix the white Gaussian noise and the pink noise, having explored the space in this way:

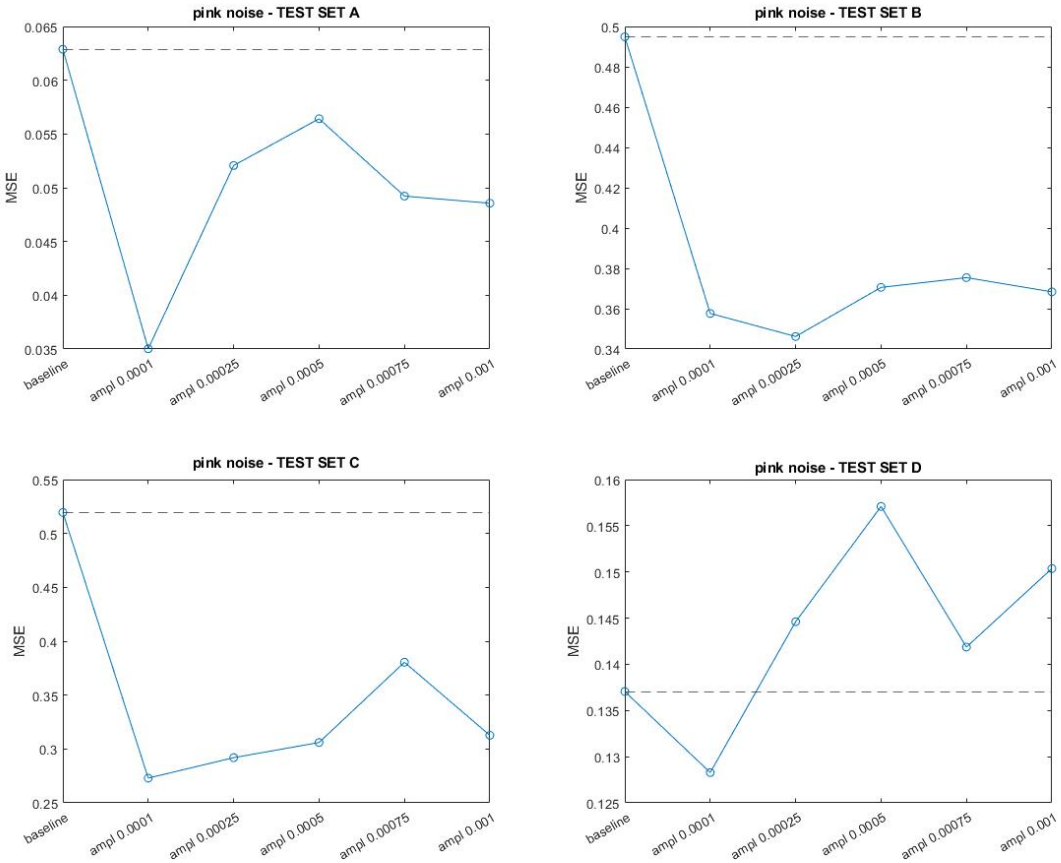

Figure 4.9. Design Space Exploration for pink noise addition in test sets A, B, C and D.

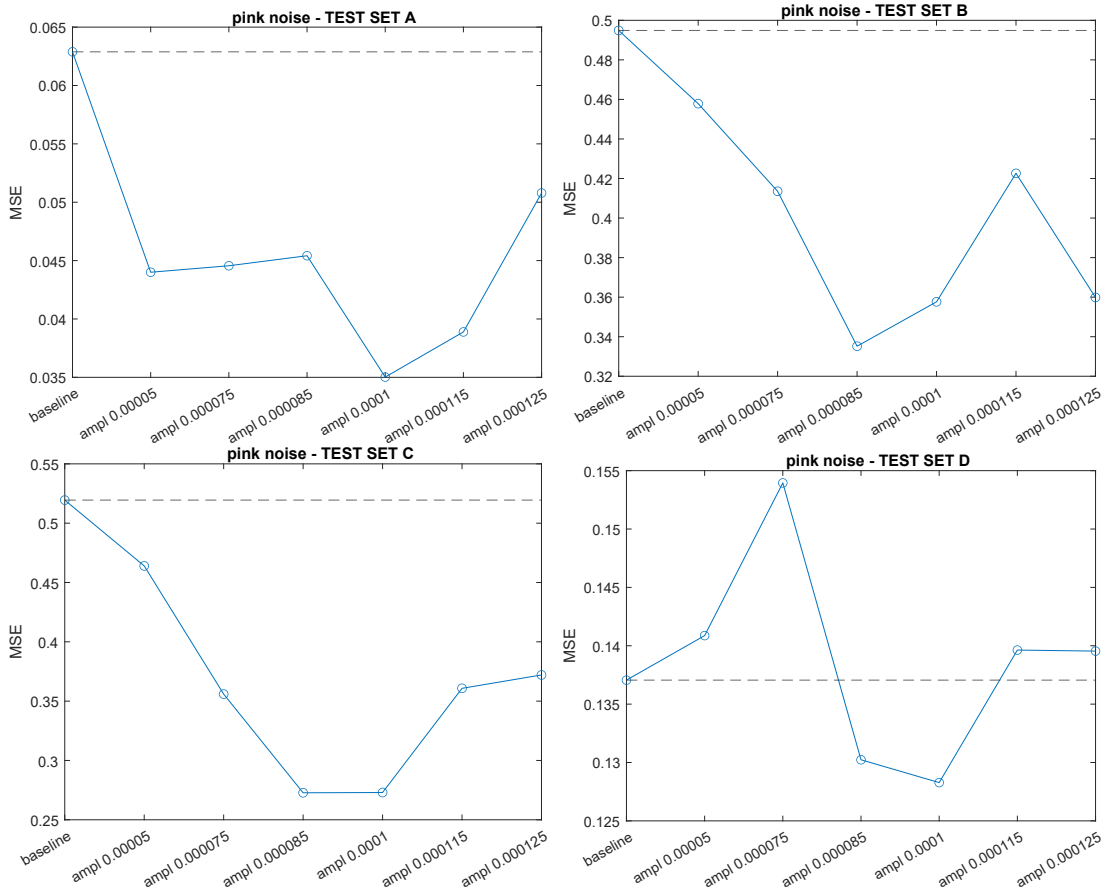

Figure 4.10. Design Space Exploration for pink noise addition around amplitude=0.0001 in test sets A, B, C and D.

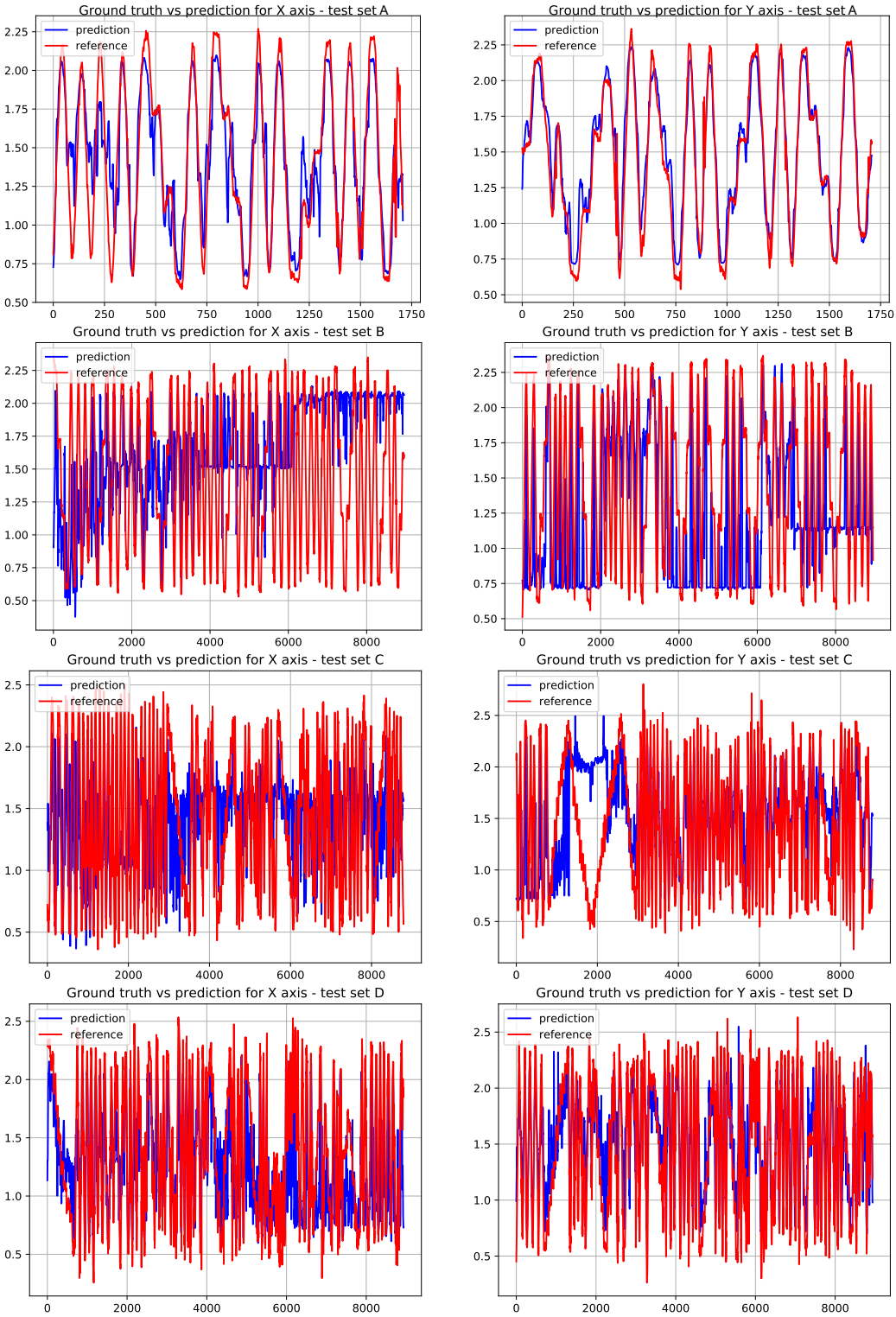

Figure 4.11. Ground truth vs prediction for the best pink noise found (amplitude=0.0001).

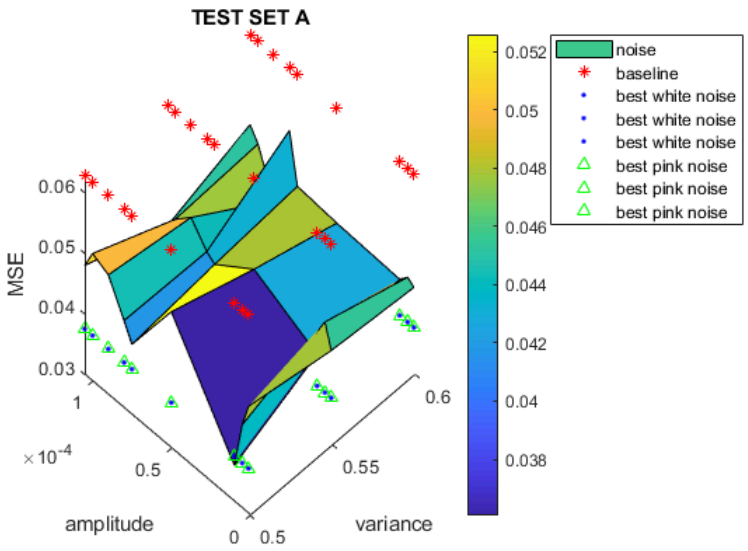

Figure 4.12. Design Space Exploration for mixed white Gaussian and pink noises addition in test set A.

- I kept fixed white Gaussian noise parameters, while I changed the pink noise amplitude;
- Then, I found the best pink noise amplitudes, and I used them while I varied the white Gaussian noise.

Obtaining the results given in Figures 4.12, 4.13, 4.14 and 4.15.

In this case I chose to keep as reference the best MSE values of each single type of noise, independently of the run, to search for an improvement.

The first exploration is with white Gaussian noise (variance=0.5, 0.55, 0.6 and amplitude=0.00037, which correspond to a promising configuration found) and with pink noise (amplitude=0.000001, 0.000005, 0.00001, 0.00005, 0.000075, 0.00008, 0.00009, 0.0001, 0.000105, because pink noise degrades the performances if it's over the value 0.0001). By analysing the plots, it can be seen that the MSE gets minimums with amplitude=0.00001 and 0.00005. Therefore, I continued the exploration fixing the pink noise amplitude with values: 0.00001, 0.00002, 0.00003, 0.00004 and 0.00005, while varying the white Gaussian noise (amplitude=0.00025, 0.0003, 0.00034, 0.00037, 0.0004, 0.00043, 0.00045 and variance=0.5, 0.55, 0.6).

In the case of pink noise with amplitude=0.00001 (shown in Figures 4.16, 4.17, 4.18 and 4.19): a white Gaussian noise with amplitude = 0.00043 and variance=0.55 can be considered the best configuration, because there is an improvement respect to the single noises for test sets A, B and C. Moreover, the surface is concave, which means that there is a minimum in this area. So, the Table 4.3 contains the best

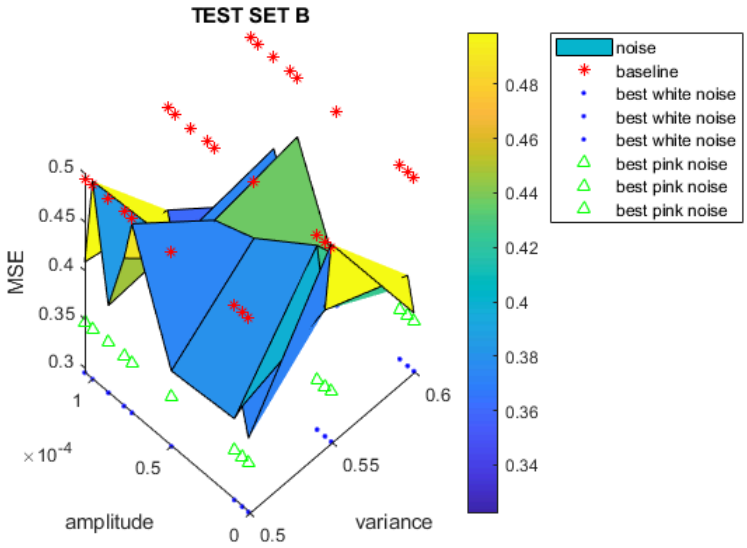

Figure 4.13. Design Space Exploration for mixed white Gaussian and pink noises addition in test set B.

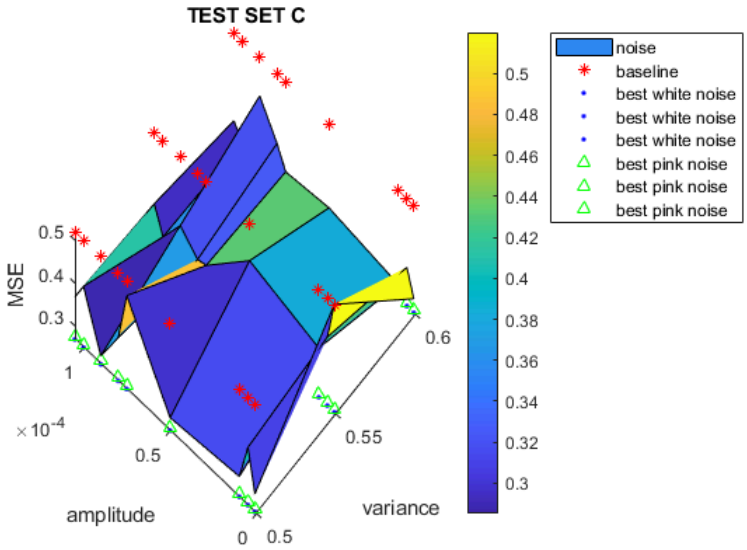

Figure 4.14. Design Space Exploration for mixed white Gaussian and pink noises addition in test set C.

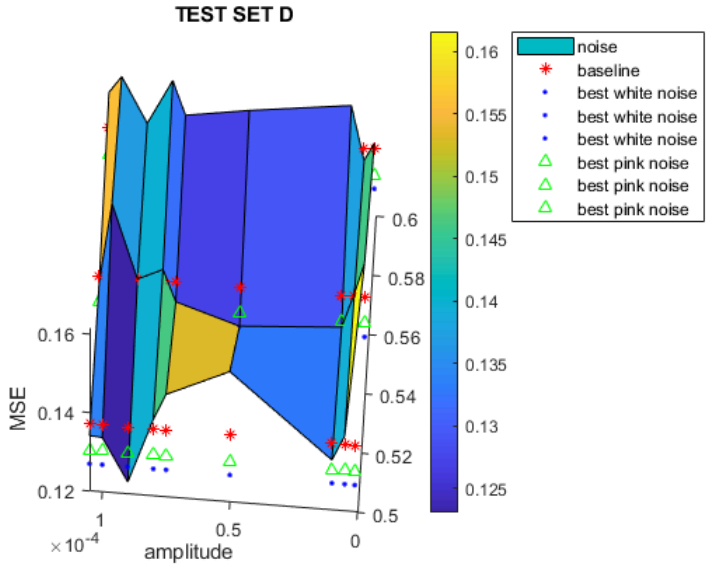

Figure 4.15. Design Space Exploration for mixed white Gaussian and pink noises addition in test set D.

Table 4.3. Best neural network configuration for mixed white Gaussian (variance=0.55 and amplitude=0.00043) plus pink (amplitude=0.00001) noise found with Mean Square Error (MSE).

	Test set A MSE (m^2)	**Test set B MSE** (m^2)	**Test set C MSE** (m^2)	**Test set D MSE** (m^2)
Mixed noise	0.034309	0.326554	0.259722	0.143374

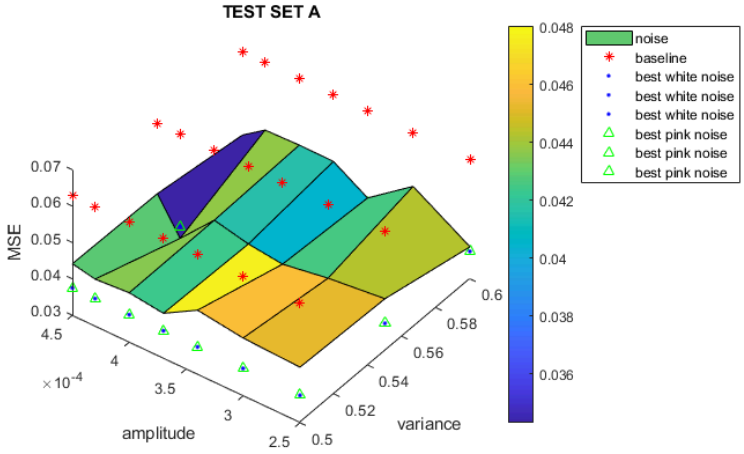

Figure 4.16. Design Space Exploration for mixed white Gaussian and pink noises addition in test set A. Using pink noise with amplitude=0.00001.

mixed noise results.

In the case of pink noise with amplitude=0.00002 (shown in Figures 4.20, 4.21, 4.22 and 4.23): no relevant improvements are found.

In the case of pink noise with amplitude=0.00003 (shown in Figures 4.24, 4.25, 4.26 and 4.27): the white Gaussian noise with variance=0.5 shows interesting results, but these results are not better than the pink noise with amplitude=0.00001.

In the case of pink noise with amplitude=0.00004 (shown in Figures 4.28, 4.29, 4.30 and 4.31): the surfaces are wavy, where it is possible to find improvement for the single noise.

In the case of pink noise with amplitude=0.00005 (shown in Figures 4.32, 4.33, 4.34 and 4.35): a white Gaussian noise with amplitude = 0.00037 is the most promising, but no improvement respect to the single noises with that amplitude.

For a complete analysis, I show in Figure 4.36 the ground truth and prediction for the best mixed noise found.

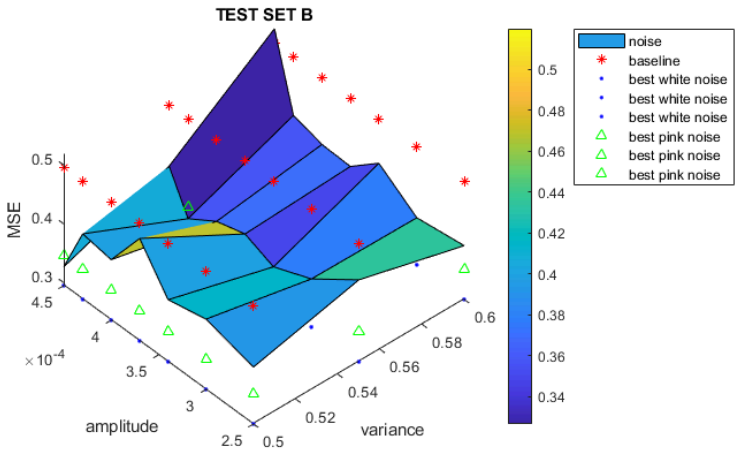

Figure 4.17. Design Space Exploration for mixed white Gaussian and pink noises addition in test set B. Using pink noise with amplitude=0.00001.

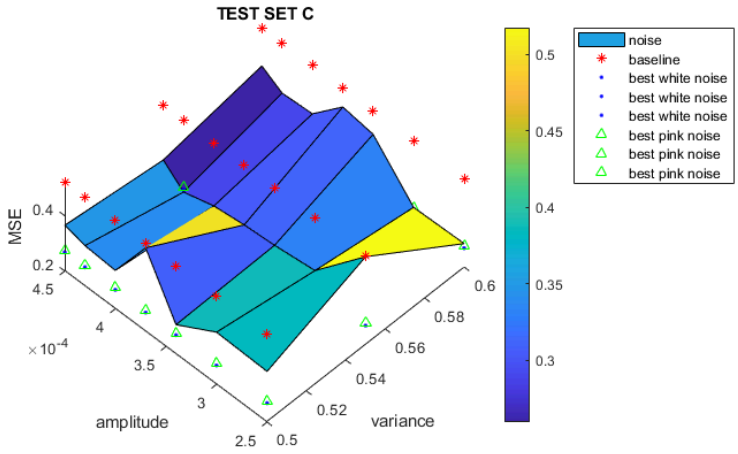

Figure 4.18. Design Space Exploration for mixed white Gaussian and pink noises addition in test set C. Using pink noise with amplitude=0.00001.

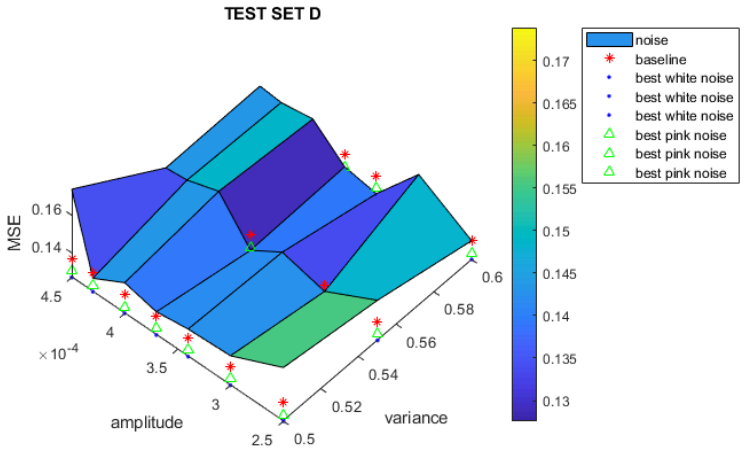

Figure 4.19. Design Space Exploration for mixed white Gaussian and pink noises addition in test set D. Using pink noise with amplitude=0.00001.

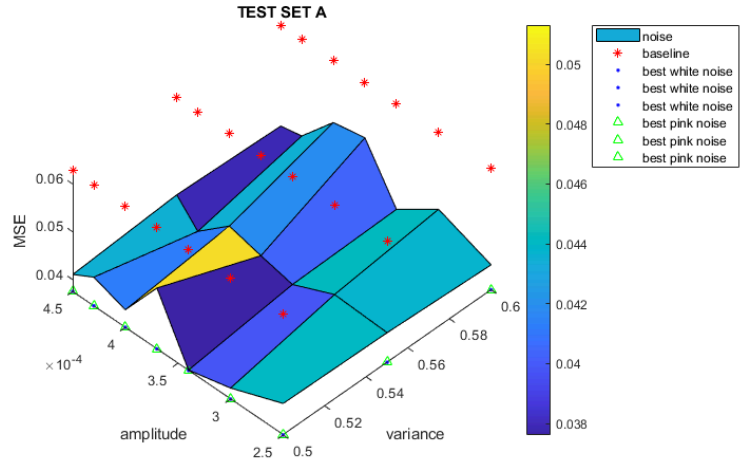

Figure 4.20. Design Space Exploration for mixed white Gaussian and pink noises addition in test set A. Using pink noise with amplitude=0.00002.

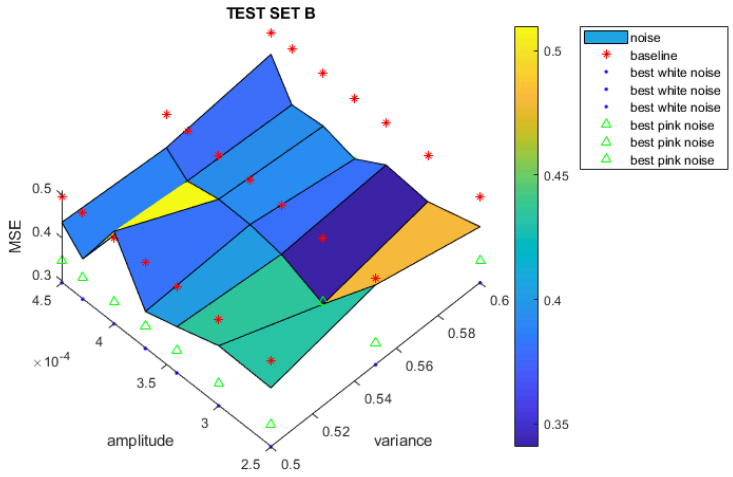

Figure 4.21. Design Space Exploration for mixed white Gaussian and pink noises addition in test set B. Using pink noise with amplitude=0.00002.

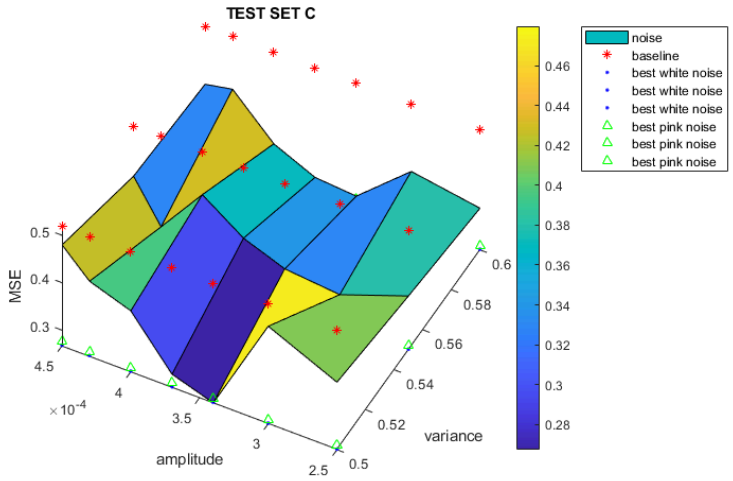

Figure 4.22. Design Space Exploration for mixed white Gaussian and pink noises addition in test set C. Using pink noise with amplitude=0.00002.

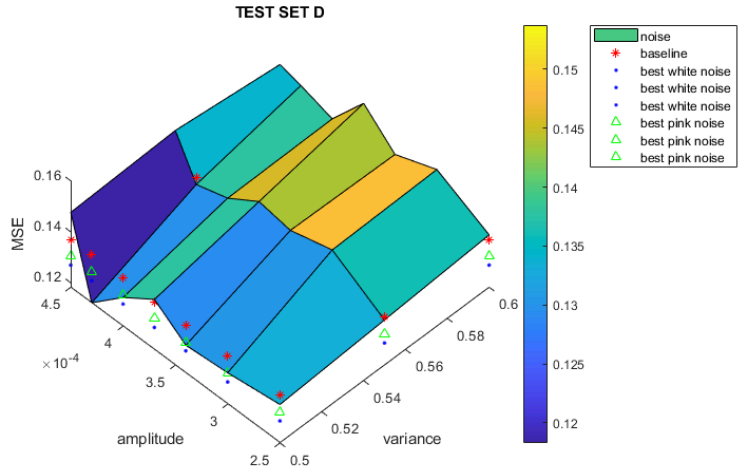

Figure 4.23. Design Space Exploration for mixed white Gaussian and pink noises addition in test set D. Using pink noise with amplitude=0.00002.

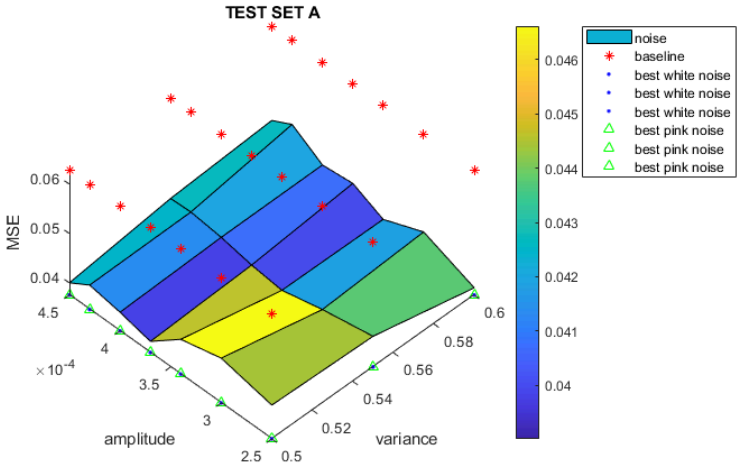

Figure 4.24. Design Space Exploration for mixed white Gaussian and pink noises addition in test set A. Using pink noise with amplitude=0.00003.

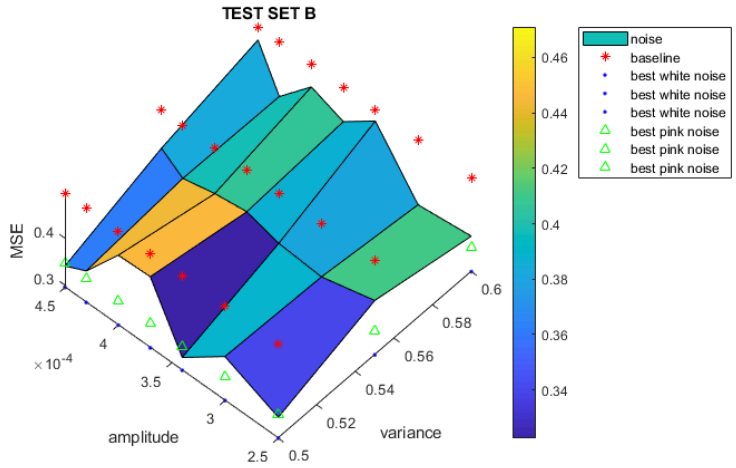

Figure 4.25. Design Space Exploration for mixed white Gaussian and pink noises addition in test set B. Using pink noise with amplitude=0.00003.

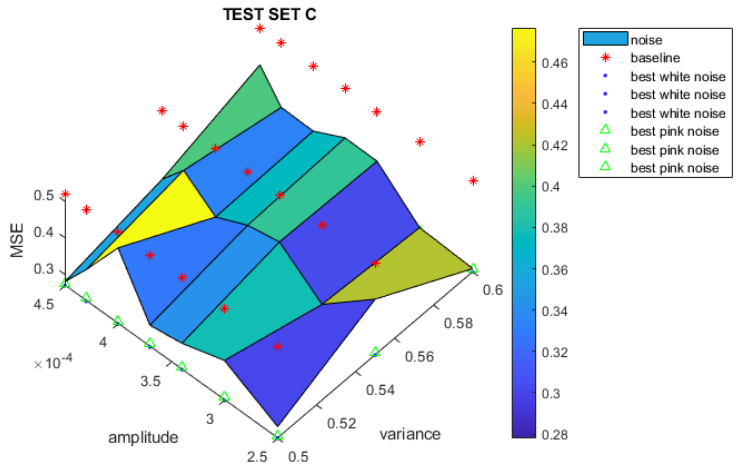

Figure 4.26. Design Space Exploration for mixed white Gaussian and pink noises addition in test set C. Using pink noise with amplitude=0.00003.

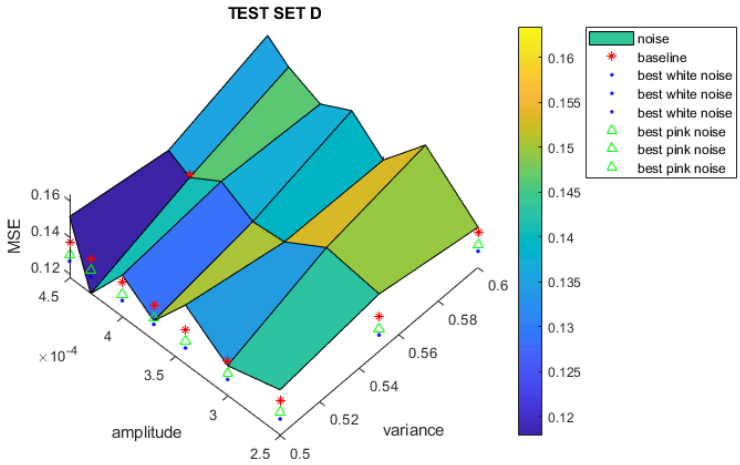

Figure 4.27. Design Space Exploration for mixed white Gaussian and pink noises addition in test set D. Using pink noise with amplitude=0.00003.

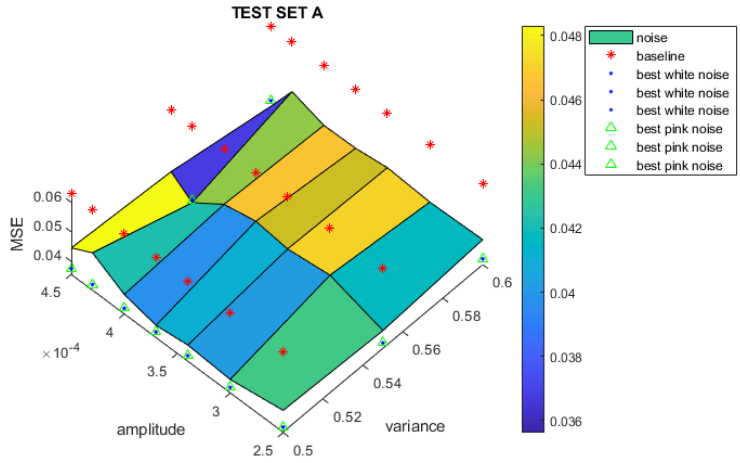

Figure 4.28. Design Space Exploration for mixed white Gaussian and pink noises addition in test set A. Using pink noise with amplitude=0.00004.

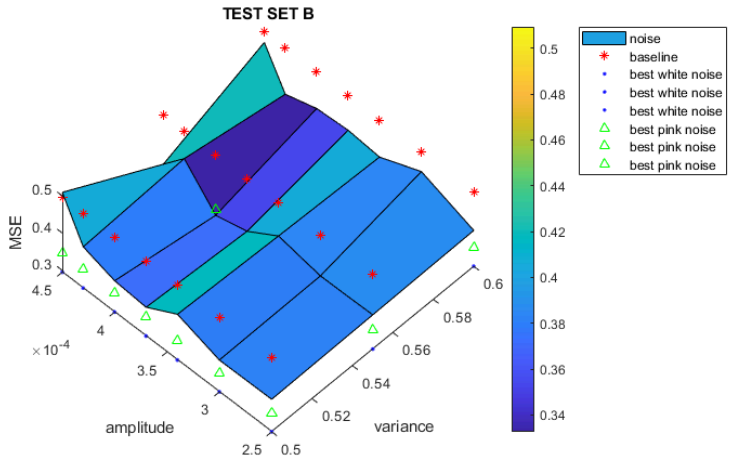

Figure 4.29. Design Space Exploration for mixed white Gaussian and pink noises addition in test set B. Using pink noise with amplitude=0.00004.

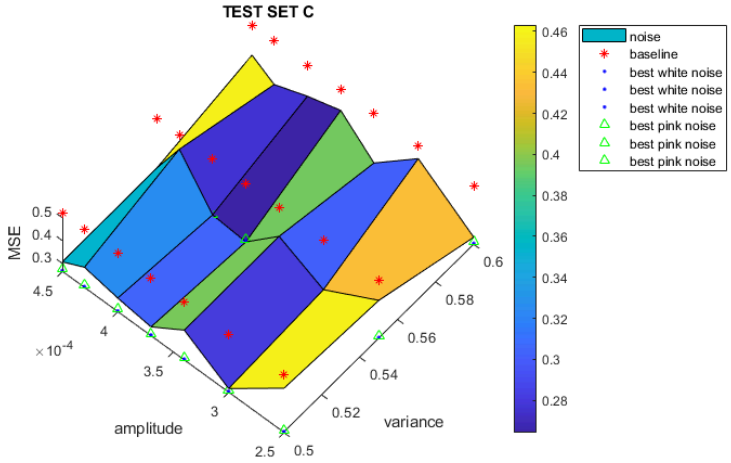

Figure 4.30. Design Space Exploration for mixed white Gaussian and pink noises addition in test set C. Using pink noise with amplitude=0.00004.

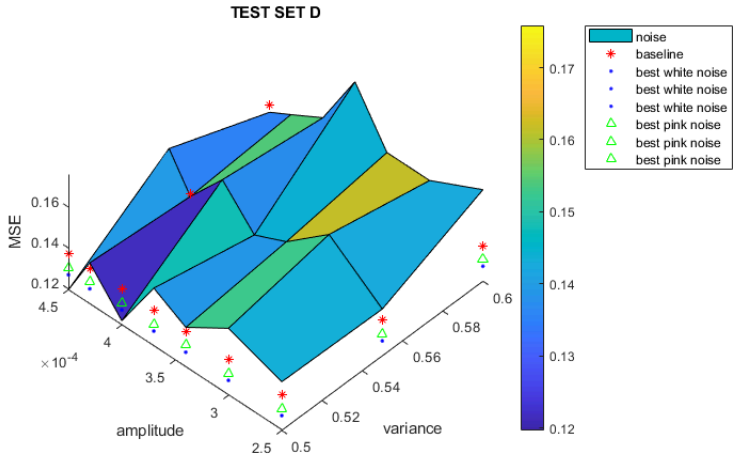

Figure 4.31. Design Space Exploration for mixed white Gaussian and pink noises addition in test set D. Using pink noise with amplitude=0.00004.

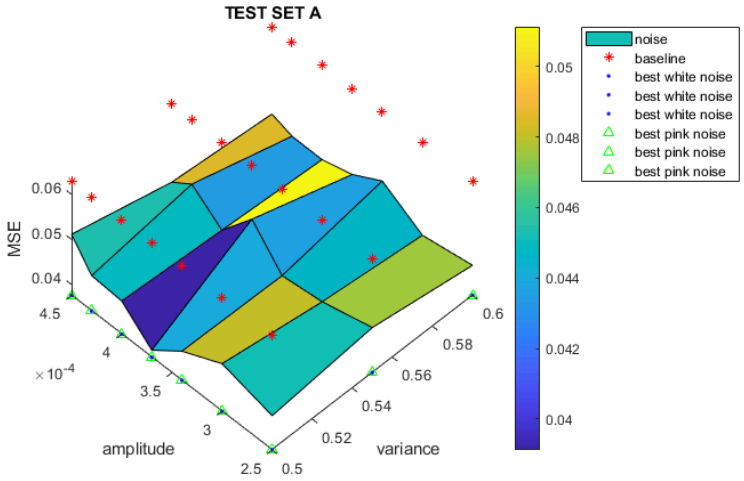

Figure 4.32. Design Space Exploration for mixed white Gaussian and pink noises addition in test set A. Using pink noise with amplitude=0.00005.

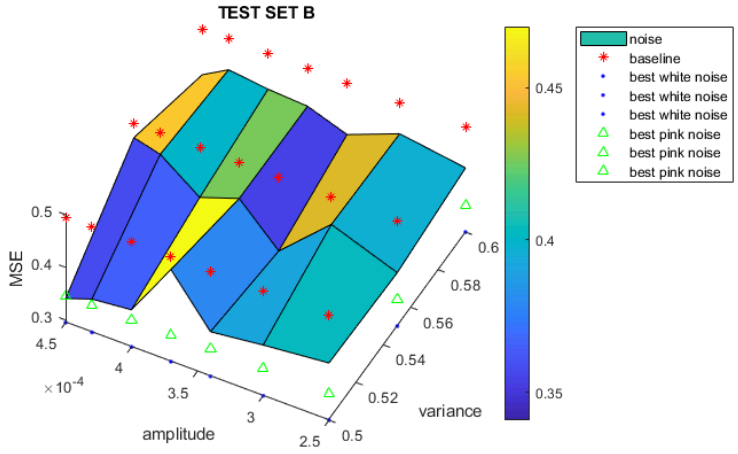

Figure 4.33. Design Space Exploration for mixed white Gaussian and pink noises addition in test set B. Using pink noise with amplitude=0.00005.

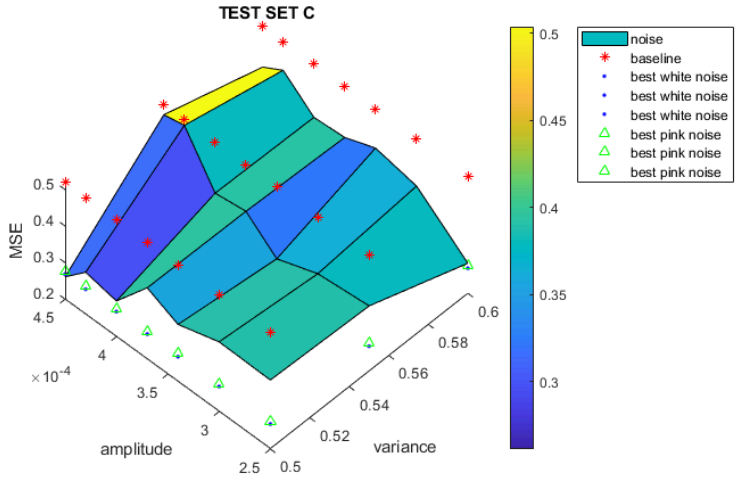

Figure 4.34. Design Space Exploration for mixed white Gaussian and pink noises addition in test set C. Using pink noise with amplitude=0.00005.

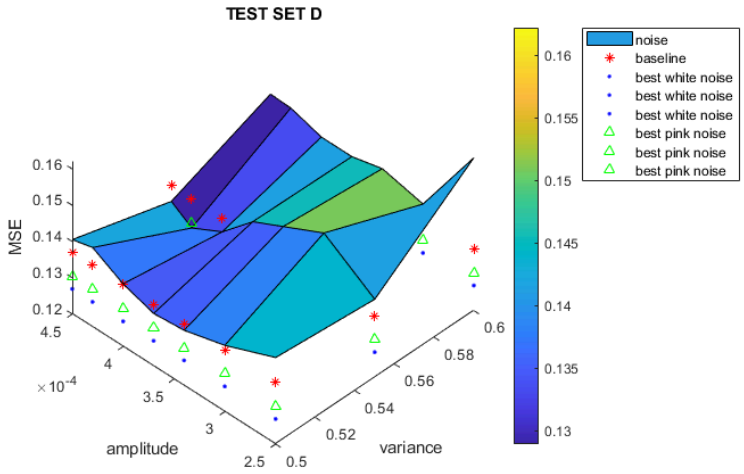

Figure 4.35. Design Space Exploration for mixed white Gaussian and pink noises addition in test set D. Using pink noise with amplitude=0.00005.

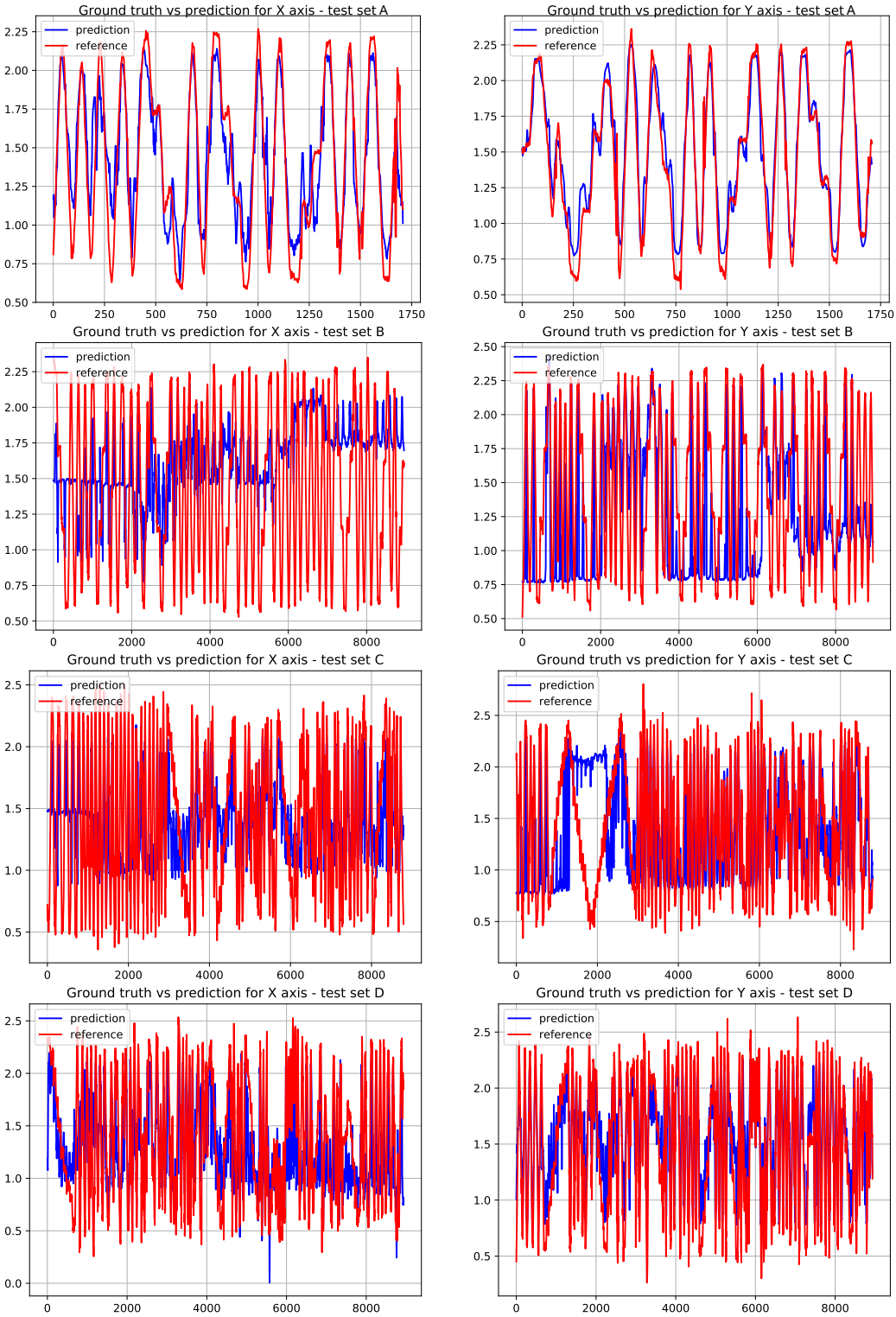

Figure 4.36. Ground truth vs prediction for the best mixed white Gaussian (variance=0.55 and amplitude=0.00043) plus pink (amplitude=0.00001) noise found.

Table 4.4. Best neural network configuration for white Gaussian noise (variance=0.55 and amplitude=0.00037), pink noise (amplitude=0.0001) and mixed white Gaussian (variance=0.55 and amplitude=0.00043) plus pink (amplitude=0.00001) noise found with Mean Square Error (MSE) and Overall Improvement (OI).

	Test set A MSE (m^2)	**Test set B MSE** (m^2)	**Test set C MSE** (m^2)	**Test set D MSE** (m^2)	**OI MSE** (%)
Baseline	0.062883	0.494872	0.519451	0.137056	
White Gaussian noise	0.03754	0.342807	0.268444	0.133188	56.07
Pink noise	0.035022	0.357643	0.272982	**0.128278**	55.40
Mixed noise	**0.034309**	**0.326554**	**0.259722**	0.143374	**57.08**

Table 4.5. Best neural network configuration for white Gaussian noise (variance=0.55 and amplitude=0.00037), pink noise (amplitude=0.0001) and mixed white Gaussian (variance=0.55 and amplitude=0.00043) plus pink (amplitude=0.00001) noise found with Average Euclidean Distance (AED) and Overall Improvement (OI).

	Test set A AED (m)	**Test set B AED** (m)	**Test set C AED** (m)	**Test set D AED** (m)	**OI AED** (%)
Baseline	0.293191	0.900044	0.928212	0.443783	
White Gaussian noise	0.233023	0.740987	0.63975	0.43531	20.12
Pink noise	**0.214586**	0.743198	0.629515	**0.414706**	22.13
Mixed noise	0.222465	**0.714431**	**0.609026**	0.441717	**22.43**

4.3 Conclusions

In Tables 4.4 and 4.5 I resume the best Neural Network (NN) configurations found in my work. For this table I calculated also the Overall Improvement (OI), which is a percentage of improvement respect to the baseline, defined as:

$$\%OI = 100(\sum_{i=1}^{N} newValue_i - \sum_{i=1}^{N} baselineValue_i)/\sum_{i=1}^{N} baselineValue_i \tag{4.1}$$

where N is the number of test sets. And I consider the OI without sign.

These NN configurations compared to the baseline, shown in Tables 4.4 and 4.5, confirm a better learning and a better generalization over different test sets. In fact, these improvements can be easily seen with Overall Improvement.

In particular the white Gaussian noise doesn't give a consistent improvement in

inferring the position of a person in a room, in fact the Average Euclidean Distance error is 23 cm, meanwhile the Average Euclidean Distance error is 29 cm in baseline for test set A. But the way the NN fed by white Gaussian noise is able to generalize over the other test sets where the Average Euclidean Distance errors are 74 cm, 64 cm and 44 cm, respectively for test sets B, C and D, against the baseline 90 cm, 93 cm and 44 cm. Here, I remark that two-two test sets have similar measurements. Anyway an error of 74 cm and 64 cm is a lot considering a room of 3m x 3m.

Meanwhile, the pink noise gives a good improvement in inferring the position of a person. Indeed, I obtain an Average Euclidean Distance errors of 21 cm, 74 cm, 63 cm and 41 cm compared to the Average Euclidean Distance errors of the baseline 29 cm, 90 cm, 93 cm and 44 cm for test sets A, B, C and D respectively. The pink noise addition gives a better generalization compared to the white Gaussian noise, even if the Average Euclidean Distance errors are still high for test B and C to have an accurate person tracking.

In the case of a mixed noise, white Gaussian noise and pink noise added together to the measurements: the Average Euclidean Distance errors are 22 cm, 71 cm, 61 cm and 44 cm compared to the Average Euclidean Distance errors of the baseline 29 cm, 90 cm, 93 cm and 44 cm for test sets A, B, C and D respectively.

It's also interesting that the Overall Improvement of the mixed noise is higher than the pink noise OI and the white Gaussian noise OI, because the mixed noise emulates the environmental disturbances and sensor drifts given by both noises.

In conclusions, I find that is not easy to get an improvement for all the test sets with the same NN configuration. In fact, often the MSE has a minimum for a test set and a maximum for another test set. Furthermore, the high AED for test set B and C could reflect the complexity of posture assumed by the body while a person is walking.

4.4 Future work

In this Section I present some future works which may derive by the results obtained by my work:

- using a deep neural network and Long Short-Term Memory alongside data augmentation can lead to better accuracy than a 1D Convolutional Neural Network.
- preprocessing the sensor data in the sensor frontend may improve the sensing performance, the data post-processing for localization, and the generalization robustness.

Bibliography

[1] Data augmentation. https://en.wikipedia.org/wiki/Data_augmentation.

[2] Chris M Bishop. Training with noise is equivalent to tikhonov regularization. *Neural computation*, 7(1):108–116, 1995.

[3] Christopher M Bishop et al. *Neural networks for pattern recognition*. Oxford university press, 1995.

[4] Jason Brownlee. How to use data scaling improve deep learning model stability and performance. https://machinelearningmastery.com/how-to-improve-neural-network-stability-and-modeling-performance-with-data-scaling/.

[5] Jason Brownlee. Train neural networks with noise to reduce overfitting. https://machinelearningmastery.com/train-neural-networks-with-noise-to-reduce-overfitting/".

[6] Daryl Chang. Effect of batch size on neural net training. https://medium.com/deep-learning-experiments/effect-of-batch-size-on-neural-net-training-c5ae8516e57.

[7] educba.com. Machine learning technique. https://www.educba.com/machine-learning-techniques/.

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[9] Tobias Grosse-Puppendahl, Christian Holz, Gabe Cohn, Raphael Wimmer, Oskar Bechtold, Steve Hodges, Matthew S Reynolds, and Joshua R Smith. Finding common ground: A survey of capacitive sensing in human-computer interaction. In *Proceedings of the 2017 CHI conference on human factors in computing systems*, pages 3293–3315, 2017.

[10] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[11] Serkan Kiranyaz, Onur Avci, Osama Abdeljaber, Turker Ince, Moncef Gabbouj, and Daniel J Inman. 1d convolutional neural networks and applications: A survey. *Mechanical systems and signal processing*, 151:107398, 2021.

[12] Jeffrey Lee, Matthew T Cole, Jackson Chi Sun Lai, and Arokia Nathan. An analysis of electrode patterns in capacitive touch screen panels. *Journal of display technology*, 10(5):362–366, 2014.

[13] Tom M Mitchell et al. Machine learning, 1997.

[14] Andrew Ng. Neural networks and deep learning. https://www.coursera.org/learn/neural-networks-deep-learning.

[15] N. J. Nilsson. Introduction to machine learning, 1998. https://ai.stanford.edu/ nilsson/mlbook.html.

[16] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[17] Alireza Ramezani Akhmareh, Mihai Teodor Lazarescu, Osama Bin Tariq, and Luciano Lavagno. A tagless indoor localization system based on capacitive sensing technology. *Sensors*, 16(9):1448, 2016.

[18] RD Reed and RJ Marks. Ii, neural smithing, 1999.

[19] Arthur L Samuel. Some studies in machine learning using the game of checkers. ii—recent progress. *IBM Journal of research and development*, 11(6):601–617, 1967.

[20] Siddharth Sharma. Unsupervised learning project. the cocktail party effect is the..., 2021. https://medium.com/@ssiddharth408/cocktail-party-problem-using-unsupervised-learning-97fc665a4e94.

[21] Jude W Shavlik, Thomas Dietterich, and Thomas Glen Dietterich. *Readings in machine learning*. Morgan Kaufmann, 1990.

[22] Joshua Smith, Tom White, Christopher Dodge, Joseph Paradiso, Neil Gershenfeld, and David Allport. Electric field sensing for graphical interfaces. *IEEE Computer Graphics and Applications*, 18(3):54–60, 1998.

[23] Helmuth Spieler. Front-end electronics and signal processing. In *AIP Conference Proceedings*, volume 674, pages 76–100. American Institute of Physics, 2003.

[24] Osama Bin Tariq, Mihai Teodor Lazarescu, Javed Iqbal, and Luciano Lavagno. Performance of machine learning classifiers for indoor person localization with capacitive sensors. *Ieee Access*, 5:12913–12926, 2017.

[25] Osama Bin Tariq, Mihai Teodor Lazarescu, and Luciano Lavagno. Neural networks for indoor human activity reconstructions. *IEEE Sensors Journal*, 20(22):13571–13584, 2020.

[26] Tatiana Tommasi et al. Aiml course, 2020.

[27] User:Dhp1080. "Anatomy and Physiology" by the US National Cancer Institute's Surveillance, Epidemiology and End Results (SEER) Program. Wikimedia Commons, March 2019. https://upload.wikimedia.org/wikipedia/commons/b/b5/Neuron.svg.

[28] Rocio Vargas, Amir Mosavi, and Ramon Ruiz. Deep learning: A review. *Advances in Intelligent Systems and Computing*, 5, 06 2017.

[29] Philip D Wasserman and Tom Schwartz. Neural networks. ii. what are they and why is everybody so interested in them now? *IEEE expert*, 3(1):10–15, 1988.

[30] Raphael Wimmer, Matthias Kranz, Sebastian Boring, and Albrecht Schmidt. A capacitive sensing toolkit for pervasive activity detection and recognition.

In *Fifth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom'07)*, pages 171–180. IEEE, 2007.