School of Industrial and Information Engineering

Department of Mechanical Engineering

POLITECNICO MILANO 1863

Master of Science in Mechanical Engineering

Comparison of MLP, RNN and CNN Machine Learning algorithms for damage detection exploiting ultrasonic Lamb waves

Supervisor:

Prof. Claudio Sbarufatti

Co-supervisor:

Dr. Stefano Mariani

Author:

Matteo Urbani

Academic year 2019/2020

“Tien bòta”

Abstract

A large number of different techniques are used for structural health monitoring purposes. In this thesis the physical phenomenon of Lamb waves is exploited, this being elastic waves that propagate in solid materials of small thickness (i.e. plates). Specifically, a 500x500x5 mm steel plate was simulated using a Finite Element software, within which Lamb waves were excited by means of a simulated piezoelectric sensor placed in the center of the plate. A second simulated piezoelectric sensor was positioned at various locations of the plate and was used as receiver. The signal acquired by the second sensor comprises both the direct arrival of the waves transmitted by the actuator and the reflections of the latter from the edges of the plate and from any damage introduced inside the plate itself. Reading and interpretation of the acquired signals was performed via three different sets of Machine Learning (ML) algorithms: multilayer perceptron (MLP), Long Short-Term Memory (LSTM) and WaveNet (an algorithm based on deep convolutional neural networks recently developed for acoustic applications).

The use of a numerical model was essential to generate the high number of examples necessary to train the tested algorithms. The dataset on which the analyses were performed contains signals acquired by the simulated receiver placed in three different locations of the plate, in both undamaged (baseline case) and damaged conditions (obtained by adding a through-thickness defect at 100 different positions), both for temperatures varying from 20°C to 69°C at steps of 1°C (50 temperatures). The instrumentation noise effect have been included into the signals with various levels of intensity, to make them more realistic. The temperature effect was partially compensated using the Baseline Signal Stretch (BSS) technique, the only signal pre-processing tool required for the methodology discussed in this thesis.

The main task of this work was to determine the hyperparameters for each type of ML algorithm that would give the best defect detection performance. WaveNet was proven to be the best algorithm for the task when considering computation speed, ease in hyperparameters tuning and accuracy of defect detection, which reached 100% in a number of cases.

WaveNet was then tested in two other contexts. The first concerned the robustness of the algorithm in evaluating signals acquired at temperatures outside the range used for its training: the measured performance was largely satisfactory, with 100% detection accuracy reported for signals acquired up to 30°C beyond the training range. The second aimed to validate its performance when applied to a real dataset. The experimental set was acquired under a similar configuration to that used in the numerical simulations, although the plate was made of composite material and the excitation frequency was significantly lower. However, in some cases the algorithm was able to completely discern the signals with defect from those without defects, hence suggesting a strong adaptation capability that can be exploited in different applications.

Sommario

Un gran numero di tecniche differenti viene utilizzato nell’ambito del monitoraggio strutturale. In questa tesi viene sfruttato il fenomeno fisico delle onde di Lamb, onde elastiche che si propagano in materiali solidi di piccolo spessore (i.e. piastre). In particolare, una piastra in acciaio da 500x500x5 mm è stata simulata utilizzando un software agli elementi finiti, all'interno del quale le onde di Lamb sono state eccitate per mezzo di un sensore piezoelettrico simulato posto al centro della piastra. Un secondo sensore piezoelettrico simulato è stato collocato in varie posizioni della piastra e utilizzato come ricevitore. Il segnale acquisito dal secondo sensore comprende l'arrivo diretto delle onde trasmesse dall'attuatore e le loro riflessioni date dai bordi della piastra e da qualsiasi danno introdotto all'interno della piastra stessa. La lettura e l'interpretazione dei segnali acquisiti sono state affidate a tre diversi set di algoritmi di Machine Learning (ML): multilayer perceptron (MLP), Long Short-Term Memory (LSTM) e WaveNet (un algoritmo basato su reti neurali convoluzionali recentemente sviluppato per applicazioni acustiche).

L'uso di un modello numerico è stato essenziale per generare l'alto numero di esempi necessari ad addestrare gli algoritmi testati. Il dataset su cui sono state eseguite le analisi contiene segnali acquisiti dal ricevitore simulato posto in tre diverse posizioni sulla piastra, sia in condizioni non danneggiate (caso base) che in condizioni danneggiate (ottenute con l’aggiunta di un difetto passante collocato in 100 posizioni differenti), entrambe per temperature che variano tra i 20°C ed i 69°C con step di 1°C (50 temperature). L'effetto del rumore della strumentazione è stato incluso nei segnali con vari livelli di intensità, per renderli più realistici. L'effetto della temperatura è stato parzialmente compensato utilizzando la tecnica Baseline Signal Stretch (BSS), unico strumento di pre-processing dei segnali richiesto per la metodologia discussa in questa tesi.

Il compito principale di questo lavoro è stato determinare gli iperparametri per ogni tipo di algoritmo di ML che avrebbero dato le migliori prestazioni nel rilevamento dei difetti. È stato dimostrato come WaveNet sia l'algoritmo migliore per tale scopo considerando velocità di calcolo, facilità nella messa a punto degli iperparametri e accuratezza nel rilevamento dei difetti, la quale ha raggiunto il 100% in diversi casi.

WaveNet è stato quindi testato in altri due contesti. Il primo riguardava la robustezza dell'algoritmo nella valutazione di segnali acquisiti a temperature al di fuori dell'intervallo utilizzato per il suo allenamento: le prestazioni misurate sono state ampiamente soddisfacenti, con un'accuratezza nel rilevamento del 100% ottenuta per segnali acquisiti fino a 30°C oltre l'intervallo di allenamento. Il secondo mirava a convalidare le sue prestazioni quando applicato ad un set di dati reale. Il set sperimentale è stato acquisito con una configurazione simile a quella utilizzata nelle simulazioni numeriche, sebbene la piastra fosse realizzata in materiale composito e la frequenza di eccitazione fosse significativamente inferiore. Tuttavia, in alcuni casi l'algoritmo è stato in grado di discernere completamente i segnali con difetto da quelli senza difetti, suggerendo quindi una forte capacità di adattamento che può essere sfruttata in diverse applicazioni.

Table of contents

Introduction 1

Thesis background 3

Lamb waves 5

Machine Learning 6

Novelty of this study 12

Chapters content 13

1. Dataset 15

1.1. Parameters used for the numerical simulations 15

1.1.1 Geometry 15

1.1.2 Excitation signal 19

1.1.3 FE model 21

1.1.4 Signals from numerical computation 22

1.2. Pre-processing of the signals before being fed to the ML algorithms 25

1.2.1 Baseline Signal Stretch (BSS) 25

1.2.2 Adding noise 27

1.2.3 Creation of training, validation and test sets 35

2. Machine Learning architectures 39

2.1. Common hyperparameters 40

2.1.1 Optimizer 40

2.1.2 Learning rate 41

2.1.3 Epochs 42

2.1.4 Batch size 42

2.1.5 Metrics 42

2.1.6 Choice of the parameters to be saved from a given training 43

2.1.7 Number of copies 44

2.2. Multilayer perceptron (MLP) 45

2.2.1 First set of trainings 46

2.2.2 Regularisation 50

2.2.3 Second set of trainings 58

2.2.4 Best MLP hyperparameters performance – comparison with all sensors and SNRs 60

2.2.5 Principal Component Analysis (PCA) 62

2.3. Long Short-Term Memory (LSTM) 73

2.3.1 First set of trainings 75

2.3.2 Best LSTM hyperparameters performance – comparison with all sensors and SNRs 78

2.4. WaveNet 79

2.4.1 First set of trainings 80

2.4.2 Best WaveNet hyperparameters performance – comparison with all sensors and SNRs 85

2.5. Comparison of MLP, LSTM and WaveNet architectures 87

2.6. Generalisation properties of deep neural networks 90

3. Temperature extrapolation capability 93

3.1. Training, validation and test sets 93

3.2. Extrapolation performance 95

4. Experimental dataset 99

4.1. Dataset composition 99

4.2. Training, validation and test sets 104

4.3. Results 106

5. Conclusions 109

References 113

Ringraziamenti 117

Figures

Fig. 1: symmetric (left) and anti-symmetric (right) mode shapes of Lamb waves.......... 5
Fig. 2: operations occurring inside a single neuron 8
Fig. 3: most common types of activation functions 8
Fig. 4: overfitting to the training set can be seen in the cost function of the validation set, which starts to increase after about 100 steps 12
Fig. 5: steel plate dimensions and position of the emitter... 16
Fig. 6: position of defects on the plate... 16
Fig. 7: position of the available receivers on the plate 17
Fig. 8: mirroring scheme of the solutions of the FE simulations. Red circles represent the defects for which the solution of the numerical problem has been computed via numerical simulations; green circles represent the defects for which the solution of the numerical problem has been obtained by mirroring those of the symmetric red "parent" defects. The symmetries exploited are the horizontal and vertical dash-dot axes passing by the centre of the plate. Darker circles highlight an example of set of defects for which the numerical solution is equivalent but symmetric. 18
Fig. 9: locations of the three receivers used in this study 18
Fig. 10: dispersion curves for a steel plate 5mm thick. In the upper plot is represented the phase velocity while in the lower plot is represented the group velocity. Red lines represent S_n modes while blue lines represent A_0 modes. The dashed line indicate the chosen excitation frequency (300 kHz)..... 20
Fig. 11: final shape of the excitation signal 20
Fig. 12: example of one signal coming from the numerical simulations........................ 22
Fig. 13: minimum travel distance of the signal to capture defect information 23
Fig. 14: signals captured by the receiver placed in the three chosen locations 24
Fig. 15: signals taken at three different temperatures.. 26
Fig. 16: signals taken at three different temperatures after BSS.................................. 26
Fig. 17: example of noise after it has been filtered in 270-330 kHz bandwidth............. 28
Fig. 18: two macro sets of lines are represented: the first three of the legend represent the maximum absolute values of the defects' reflections for all the temperatures for three receivers; the last four of the legend represent four times the standard deviation of the four noise levels. Each line is normalised with respect to the maximum absolute value of the pristine signal collected by receiver 1 (about 45 mm from the emitter)......................... 28

Fig. 19: impact of a defect in Lamb waves propagation for noise-less signals; defect nr.79, which produced the largest reflection seen by receiver 2; pristine and defective signals are taken at the same temperature 29
Fig. 20: impact of a defect in Lamb waves propagation for signals with the lowest noise (SNR 1); defect nr.79, which produced the largest reflection seen by receiver 2; pristine and defective signals are taken at the same temperature 29
Fig. 21: impact of a defect in Lamb waves propagation for signals with the highest noise (SNR 4); defect nr.79, which produced the largest reflection seen by receiver 2; pristine and defective signals are taken at the same temperature 30
Fig. 22: impact of a defect in Lamb waves propagation for noise-less signals; defect nr.71, which produced the lowest reflection seen by receiver 2; pristine and defective signals are taken at the same temperature 30
Fig. 23: impact of a defect in Lamb waves propagation for signals with the lowest noise (SNR 1); defect nr.71, which produced the lowest reflection seen by receiver 2; pristine and defective signals are taken at the same temperature 31
Fig. 24: impact of a defect in Lamb waves propagation for signals with the highest noise (SNR 4); defect nr.71, which produced the lowest reflection seen by receiver 2; pristine and defective signals are taken at the same temperature 31
Fig. 25: impact of a defect in Lamb waves propagation for noise-less signals; defect nr.79, which produced the largest reflection seen by receiver 2; pristine and defective signals are taken at temperatures differing of 1°C 32
Fig. 26: impact of a defect in Lamb waves propagation for signals with the lowest noise (SNR 1); defect nr.79, which produced the largest reflection seen by receiver 2; pristine and defective signals are taken at temperatures differing of 1°C 32
Fig. 27: impact of a defect in Lamb waves propagation for signals with the highest noise (SNR 4); defect nr.79, which produced the largest reflection seen by receiver 2; pristine and defective signals are taken at temperatures differing of 1°C 33
Fig. 28: impact of a defect in Lamb waves propagation for noise-less signals; defect nr.71, which produced the lowest reflection seen by receiver 2; pristine and defective signals are taken at temperatures differing of 1°C 33
Fig. 29: impact of a defect in Lamb waves propagation for signals with the lowest noise (SNR 1); defect nr.71, which produced the lowest reflection seen by receiver 2; pristine and defective signals are taken at temperatures differing of 1°C 34
Fig. 30: impact of a defect in Lamb waves propagation for signals with the highest noise (SNR 4); defect nr.71, which produced the lowest reflection

seen by receiver 2; pristine and defective signals are taken at temperatures differing of 1°C 34
Fig. 31: division of defects positions in training set (blue), validation set (green) and test set (red). The order of defects' numbers is the one used in Matlab and Python codes, it does not have any particular meaning 36
Fig. 32: division of defects and pristine temperatures in training set (blue), validation set (green) and test set (red) 36
Fig. 33: visual intuition of the difference between gradient descent and Adam algorithms 41
Fig. 34: mini-batch technique influence on loss function minimisation procedure 42
Fig. 35: architecture of MLP Neural Network 45
Fig. 36: MLP - accuracy plots: 1 hidden layer 47
Fig. 37: MLP - accuracy plots: 2 hidden layers 47
Fig. 38: MLP - accuracy plots: 3 hidden layers 48
Fig. 39: MLP - accuracy plots: 4 hidden layers 48
Fig. 40: MLP - test accuracy comparison 49
Fig. 41: MLP - loss plot: representative case of the trend of training and validation losses along epochs 49
Fig. 42: concept of regularization effect when neural networks tend to overfit over the training set 50
Fig. 43: visual representation of dropout, which randomly sets to zero some neuros 51
Fig. 44: MLP - accuracy plots: 1 hidden layer with dropout regularisation 51
Fig. 45: MLP - accuracy plots: 2 hidden layers with dropout regularisation 52
Fig. 46: MLP - accuracy plots: 3 hidden layers with dropout regularisation 52
Fig. 47: MLP - accuracy plots: 4 hidden layers with dropout regularisation 52
Fig. 48: MLP - accuracy plots: 1 hidden layer with L2 regularisation 53
Fig. 49: MLP - accuracy plots: 2 hidden layers with L2 regularisation 54
Fig. 50: MLP - accuracy plots: 3 hidden layers with L2 regularisation 54
Fig. 51: MLP - accuracy plots: 4 hidden layers with L2 regularisation 54
Fig. 52: MLP - accuracy plots: 1 hidden layer, comparison among different regularisers 55
Fig. 53: MLP - accuracy plots: 2 hidden layers, comparison among different regularisers 55
Fig. 54: MLP - accuracy plots: 3 hidden layers, comparison among different regularisers 56
Fig. 55: MLP - accuracy plots: 4 hidden layers, comparison among different regularisers 56
Fig. 56: MLP - loss plots: learning case without any regulariser 57
Fig. 57: MLP - loss plots: learning case with dropout regulariser ($p = 0.2$) 57
Fig. 58: MLP - loss plots: learning case with L2 regulariser ($\lambda = 1e - 6$) 58

Fig. 59: MLP - accuracy plots: 3 hidden layers, second set of trainings........................ 59
Fig. 60: MLP - accuracy plots: 4 hidden layers, second set of trainings........................ 59
Fig. 61: MLP – test set accuracy plots: 3 and 4 hidden layers, comparison between the cases with neurons distributed in decreasing order (left, first set of training with L2 regulariser) and neurons distributed in increasing-decreasing order (right, second set of training with L2 regulariser) 60
Fig. 62: MLP - accuracy plots: 1 hidden layer, final set of runs 61
Fig. 63: MLP - accuracy plots: 2 hidden layers, final set of runs.................................. 61
Fig. 64: MLP - accuracy plots: 3 hidden layers, final set of runs.................................. 62
Fig. 65: MLP - accuracy plots: 4 hidden layers, final set of runs.................................. 62
Fig. 66: PCA transformation – x represents the original reference system, x' represents the principal reference system.. 63
Fig. 67: PCA - cumulative sum of eigenvalues for different SNR levels, sensor 1 64
Fig. 68: PCA - cumulative sum of eigenvalues for different SNR levels, sensor 2 64
Fig. 69: PCA - cumulative sum of eigenvalues for different SNR levels, sensor 3 65
Fig. 70: difference between an example and its reconstruction with only the first principal component – noise-less case 65
Fig. 71: difference between an example and its reconstruction with only the first principal component – SNR 1 case 66
Fig. 72: difference between an example and its reconstruction with only the first principal component – SNR 4 case 66
Fig. 73: difference between an example and its reconstruction with the first 200 principal components – noise-less case.. 67
Fig. 74: difference between an example and its reconstruction with the first 200 principal components – SNR 1 case.. 67
Fig. 75: difference between an example and its reconstruction with the first 200 principal components – SNR 1 case.. 68
Fig. 76: difference between an original signal and the same after it has been transformed by PCA 68
Fig. 77: MLP - accuracy plots: 1 hidden layer, PCA 69
Fig. 78: MLP - accuracy plots: 2 hidden layers, PCA.. 69
Fig. 79: MLP - accuracy plots: 3 hidden layers, PCA.. 70
Fig. 80: MLP - accuracy plots: 4 hidden layers, PCA.. 70
Fig. 81: visual representation of the operations computed inside an LSTM cell: rectangles represent a linear combination of its inputs, rhombus represent an operator and the two tanh are activation functions mapping directly inputs to outputs 74
Fig. 82: LSTM - accuracy plots: 1 and 2 layers 76
Fig. 83: LSTM - loss plot: representative case of the trend of training and validation losses along epochs.. 77
Fig. 84: LSTM - accuracy plots: final set of runs.. 78

Fig. 85: scheme of WaveNet architecture 80
Fig. 86: WaveNet - accuracy plots: dd=4, ks=2 82
Fig. 87: WaveNet - accuracy plots: dd=4, ks=3 82
Fig. 88: WaveNet - accuracy plots: dd=4, ks=4 82
Fig. 89: WaveNet - accuracy plots: dd=6, ks=2 83
Fig. 90: WaveNet - accuracy plots: dd=6, ks=3 83
Fig. 91: WaveNet - accuracy plots: dd=6, ks=4 83
Fig. 92: WaveNet - test accuracy plots: comparison for all the hyperparameters used 84
Fig. 93: WaveNet - loss plot: representative case of the trend of training and validation losses along epochs 85
Fig. 94: WaveNet - accuracy plots: final set of runs 86
Fig. 95: WaveNet – distribution of the predictions of a model when test set examples are fed as input 86
Fig. 96: architectures comparison - test accuracy plots, SNR 1 87
Fig. 97: architectures comparison - test accuracy plots, SNR 2 88
Fig. 98: architectures comparison - test accuracy plots, SNR 3 88
Fig. 99: architectures comparison - test accuracy plots, SNR 4 88
Fig. 100: division of defects and pristine temperatures for extrapolation case: training set (blue), validation set (green) and 3 test sets (yellow, orange and red respectively) 94
Fig. 101: accuracy plots for temperature extrapolation test – SNR 1 96
Fig. 102: accuracy plots for temperature extrapolation test – SNR 2 96
Fig. 103: accuracy plots for temperature extrapolation test – SNR 3 96
Fig. 104: accuracy plots for temperature extrapolation test – SNR 4 97
Fig. 105: CFRP plate dimensions and positions of emitter, receiver and defects 100
Fig. 106: example of experimental signal 100
Fig. 107: impact of defect nr. 1 in Lamb waves propagation for experimental signals; pristine and defective signals are taken at the same temperature 101
Fig. 108: impact of defect nr. 2 in Lamb waves propagation for experimental signals; pristine and defective signals are taken at the same temperature 102
Fig. 109: impact of defect nr. 3 in Lamb waves propagation for experimental signals; pristine and defective signals are taken at the same temperature 102
Fig. 110: impact of defect nr. 4 in Lamb waves propagation for experimental signals; pristine and defective signals are taken at the same temperature 102
Fig. 111: impact of defect nr. 1 in Lamb waves propagation for experimental signals; pristine and defective signals are taken at temperatures differing of 1°C 103
Fig. 112: impact of defect nr. 2 in Lamb waves propagation for experimental signals; pristine and defective signals are taken at temperatures differing of 1°C 103

Fig. 113: impact of defect nr. 3 in Lamb waves propagation for experimental signals; pristine and defective signals are taken at temperatures differing of 1°C 103
Fig. 114: impact of defect nr. 4 in Lamb waves propagation for experimental signals; pristine and defective signals are taken at temperatures differing of 1°C 104
Fig. 115: division of defects and pristine temperatures in training set (blue), validation set (green) and test set (red) for the experimental dataset 105
Fig. 116: WaveNet - accuracy plots with experimental dataset using defect in position 1 to validate the algorithm 107
Fig. 117: WaveNet - accuracy plots with experimental dataset using defect in position 2 to validate the algorithm 107
Fig. 118: WaveNet - accuracy plots with experimental dataset using defect in position 3 to validate the algorithm 107
Fig. 119: WaveNet - accuracy plots with experimental dataset using defect in position 4 to validate the algorithm 108

Tables

Tab. 1: locations of the three receivers used in this study 19
Tab. 2: phase and group velocities for S_0 and A_0 modes for an excitation source of 300 kHz 20
Tab. 3: material properties for the considered range of temperatures, i.e. from 20°C to 69°C 21
Tab. 4: first arrival times computed from dispersion curves data 24
Tab. 5: number of defect and pristine examples in every set – unbalanced distribution 37
Tab. 6: number of defect and pristine examples in every set – balanced distribution 37
Tab. 7: MLP architecture hyperparameters - first set of runs 46
Tab. 8: number of neurons for every hidden layer in MLP architecture 46
Tab. 9: number of neurons for every hidden layer in MLP architecture: second set of trainings 58
Tab. 10: MLP architecture hyperparameters - final set of runs 61
Tab. 11: MLP architecture hyperparameters - PCA 69
Tab. 12: LSTM architecture hyperparameters - first set of runs 75
Tab. 13: LSTM architecture hyperparameters - final set of runs 78
Tab. 14: WaveNet architecture hyperparameters - first set of runs 80
Tab. 15: WaveNet architecture hyperparameters - final set of runs 85
Tab. 16: architectures comparison – training duration 89
Tab. 17: number of defect and pristine examples in every set for extrapolation case – unbalanced distribution 94
Tab. 18: number of defect and pristine examples in every set for extrapolation case – balanced distribution 95
Tab. 19: position of sensors and defects on the CFRP plate 100
Tab. 20: number of defect and pristine examples in every set of the experimental configuration – unbalanced distribution 105
Tab. 21: number of defect and pristine examples in every set of the experimental configuration – balanced distribution 106

Introduction

Every mechanical system is designed based on theoretical and numerical models which cannot account for all the possible environmental conditions or events that might interfere with the system during its life cycle. Years of experiments and tests on the most common components and loading conditions have been made, which allowed a fine tuning of coefficients that can be used to correct theoretical models' predictions and increase their accuracies. However, relying on the component's lifetime predicted a priori by a model can be an inconvenient approach: at the end of its predicted life the component is substituted no matter its actual condition; it can be worn enough to negatively affect the performance of the entire system (thus its substitution is correct) as well as still working properly (thus its substitution turns out to be a loss of resources); in the worst case the component can break before the end of its predicted life-cycle causing damages to other components, forced stop of the entire system, supplementary maintenance and, most importantly, safety issues.

An approach to assess the condition of mechanical structures relies on data acquisition from the actual working system and their elaboration. This approach is based on the ability to read and extract important information from sensors in order to understand whether there is the necessity to intervene with corrective actions in the structure, i.e. maintenance procedures. Based on the frequency, a classification of different maintenance approaches can be identified:

- planned (preventive) maintenance: control of components' health is taken at fixed-time intervals, decided during components and system design stage;
- predictive maintenance: it relies on both theoretical models and measurements (temperature, pressure, chemical composition, vibrations, noise…) which are analysed each time the maintenance is scheduled. The date of the following

maintenance is decided based on the data acquired, which are used to predict the remaining life of every component;
- condition-based maintenance: maintenance is executed only when one or a set of monitored sensors register unacceptable levels of certain parameters.

When maintenance is required the system must be stopped and completely dismantled in order to assess the actual state of every component. This is a drawback for planned maintenance since it is done regardless of the need of a component to be repaired or substituted. This type of maintenance is suited for small and cheap systems where downtimes are not much relevant and do not lead to economic issues: the smaller and simpler the system, the faster and cheaper the maintenance.
When systems become bigger, more complex and more expensive, it may be worth buying high quality sensors and acquisition systems to have a more accurate control of its status. The initial costs are not negligible but can be justified by a lower maintenance frequency, which means lower downtimes, better usage of resources and economic savings. Predictive maintenance is in fact applied when downtime-related costs are considerable, e.g. Loss of profits during downtime or direct maintenance labour cost.
Condition-based maintenance is the extreme version of predictive maintenance. Initial costs can be even higher due to the need of a larger number of sensors to be installed on the structure; furthermore, sensors and acquisition system bust be run frequently in order to promptly indicate the eventual presence of a malfunctioning. The great advantage of this maintenance technique is that it is able to detect abrupt changes on the health state of the monitored system, that would be impossible for the other two techniques for which the system is monitored only at certain intervals. All these considerations justify condition-based maintenance only in fields where structures are big, complex and subjected to damages related to unpredictable external agents.

Structural Health Monitoring (SHM) is a condition-based monitoring method whose aim is to detect the presence of damages in structures. This is done by acquiring signals from different sensors and extracting useful information about the presence of defects, which affect physical characteristics of the structure. In this thesis Lamb waves are exploited in a SHM framework to assess the presence of a defect in a simulated steel plate. Lamb waves propagate in solid mediums and are reflected either by boundaries of the geometry of the structure or by defects and damages (cracks, holes, corrosion…). These imperfections alter shape and direction of the waves, hopefully allowing to detect presence, location and entity of the damage. Since Lamb waves are extremely complex, it is not straightforward to extrapolate information from them, thus Artificial Intelligence (AI) algorithms can be exploited to develop reliable monitoring systems to be used in real cases.

Thesis background

Several studies have been reported in literature about the interpretation of Lamb wave signals with Machine Learning algorithms for health monitoring purposes. One of the leading fields is the aeronautical one where the complexity of aircrafts requires high levels of accuracy in interpreting the health state of the structure; the increasing performance of health monitoring techniques is spreading also to high speed trains, railways, pipelines, wind turbines and many others.

Most of the works follow a similar trend, which consists in the following steps:

- identification of material and geometry of the structure to be analysed;
- choice of sensors in terms of typology (usually piezoelectric), quantity and position;
- setting of the numerical model used to create the dataset needed to train ML algorithms;
- identification of pre-processing techniques to be used to clean signals or extract particular features;
- build ML algorithms in order to have the required outputs, which usually are detection of a defect and its localisation and damage quantification;
- training of ML algorithms;
- validation of ML algorithms with signals obtained from experimental tests.

Structures are usually made from the well-known steel and aluminium materials to the latest more advanced and promising composite materials, mainly reinforced with carbon fibres. The typical structure analysed consists in a simple square plate, which guarantees both ease of production (necessary to experimentally validate the model) and good generalisation of the obtained results. The quantity of sensors used can be high in order to create a grid or a network; in any case they are mainly used in couples of emitter and receiver and different combinations are tested to understand any possible relation between their position in the structure and relatively to the defects.

The stages that mainly differ between Lamb-waves-based SHM studies involve the pre-processing techniques of the signals and the typology of ML algorithms implemented. Signals coming from numerical simulations are lacking stochastic disturbances, thus are usually contaminated with artificial noise (e.g. white noise) at different amplitudes in order to make them closer to reality. Before being fed to the ML algorithms, the dataset can be pre-processed for the following reasons:

- cut the signal to consider only the most significant part: by knowing material characteristics and geometry of the plate it is possible to find the time needed for the excitation to reach the sensor (i.e. time of flight, tof), allowing to collect the signal with a certain delay and shorten its length;

- reduce the dimensionality of the signal to account for only the most significant features: for example Principal Component Analysis (PCA) is a technique that allows to transform the dataset of signals and reduce its dimensions without losing the most important information of the signal;
- compensate effects given by change in temperatures: it is a well-known issue affecting the shape of Lamb wave signals. Among all the techniques can be found baseline signal reconstruction, baseline signal stretch (BSS) [1] [2], orthogonal matching pursuit;
- extraction of different features: an example can be transforming the dataset of signals from time domain to frequency domain with a Fourier transformation; in some cases also some error indexes can be extracted from the signals, like the Root Mean Square Error (RMSE) between 2 signals, the difference between peaks, the delay of some waves.

For what concerns the ML algorithms it is quite common the usage of multilayer perceptrons (MLPs), which are the most basic types of neural networks and thus the easiest to implement. Also Support Vector Machines (SVMs) are often used for defect detection: they are supervised-learning algorithms that can be used to distinguish whether an input belongs to one class or the other (only binary classification). It is rarer to find the implementation of Recurrent Neural Networks (RNNs) and only during the very last few years Convolutional Neural Networks (CNNs) started to spread their benefits into Lamb waves interpretation in SHM framework.
The accuracies obtained in detection, localisation and damage quantification may largely vary depending on the choices of materials, measurement setup, pre-processing of signals, architecture of ML algorithm and many other parameters. These kind of monitoring systems can correctly detect, localise and quantify even the entire dataset of signals used in specific cases, which is the reason why ML algorithms are becoming increasingly used to extract important features from Lamb waves. However, AI is a fast growing field nowadays but it is still difficult to deal with its complexity: most of the times it is still preferred to rely on deterministic rearrangements (pre-processing of datasets) rather than feeding signals acquired from sensors directly to ML algorithms, which are more difficult to interpret since they act as black boxes whose internal behaviour is not easily understandable. That is the reason why several (sometime sophisticated) types of pre-processing stages are still implemented before feeding inputs to the ML algorithms.

This is only a brief summary of the key points representing the state of the art in health monitoring field exploiting Lamb waves. Several articles can be found in literature regarding these topics; few of these are [3] [4] [5] which furnished helpful ideas to undertake the work made for this thesis.

In the following two chapters a brief introduction to Lamb waves and ML algorithms is reported.

Lamb waves

Lamb waves are elastic waves propagating in thin plates. They are formed by the displacement of particles in the material with energy moving radially along the plate with respect to the excitation point and normal to the plate's surfaces. In the ideal case of an infinite medium these waves are composed by only two modes: *symmetric* (S_0, longitudinal or pressure waves, Fig. 1-left) and *antisymmetric* (A_0, transverse or shear waves, Fig. 1-right). When plates are considered the medium is confined between two surfaces and infinite pairs of modes (symmetric and antisymmetric) arise; these propagate in longitudinal and transverse directions, denoted with S_n and A_n respectively, being n the number of the mode pair [6]. Below a certain frequency only fundamental modes S_0 and A_0 are present; when it comes to higher frequencies other modes sums up and create a wave with increasing complexity. In order to limit the complexity, the excitation signal is commonly generated at a frequency low enough to excite only the fundamental modes. Nonetheless, presence of defects may also distribute wave's energy towards higher frequencies allowing the propagation of other modes [4], raising the complexity of the problem.

Every single mode is characterised by two velocities: group velocity and phase velocity, which are respectively the speed at which the envelope and the phase of the waves propagate. The former can be used to identify the time needed for the wave to travel at a certain distance, so can be used to properly set the acquisition time of the signal. Both velocities turn out to be dispersive, which means that they depend on the frequency of the excitations: it is unfeasible to generate an excitation signal with a bandwidth narrow enough to excite only one frequency, so the generated wave will not be able to maintain the same shape as it propagates. This adds another level of complexity to the signal.

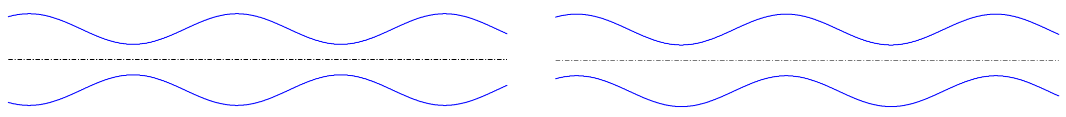


Fig. 1: symmetric (left) and anti-symmetric (right) mode shapes of Lamb waves

In SHM framework these waves can be used to detect the presence of a defect: when the wave encounters a local change in material properties or geometry of the plate (e.g. stiffness reduction due to a void left by a crack) a certain amount of energy is scattered

and released radially from it. This generates another wave propagating in all directions, which sums up to the original wave. The difference in wave propagation can be detected by a sensor and can give useful information about the presence of a defect and possibly also about its location and severity.
The reflection of the waves depend on how much they are sensitive to changes in the plates: neglecting any dissipation of the energy, at the boundaries of the plate the waves are completely reflected back symmetrically with respect to the normal of the boundary; when instead the wave encounters a defect it is not completely reflected: depending on its dimension, a part of the energy of the wave will continue to propagate past the defect, while the other portion will be radially scattered from it. The amount of scattering will greatly depend on the relative dimensions of the wavelength and size of the defect and it becomes very low when the former is much larger than the latter.

Machine Learning

Machine Learning (ML) is the name used to identify algorithms that are capable of accomplishing a large variety of tasks by learning through experience. It is like a software that improves itself learning from its own mistakes, which is why it belongs to AI framework. These types of algorithms are broadening their fields of application during the last decades thanks to the continuous improvement of computational power, which is one of the key factors that allows AI to be actually applied in industries. ML algorithms are already applied for different purposes, like Natural Language Processing (NLP, e.g. Alexa from Amazon or Siri from Apple) or computer vision (e.g. self-driving cars), and are demonstrating their enormous potentialities: they can be adapted for multiple kinds of tasks due to their flexibility, from the easiest to the most difficult problems.

Based on their learning approach, ML algorithms can be grouped in two classes [7]:

- *unsupervised* learning: a set of baseline examples is taken as input by the algorithm, which extrapolates different features from them. When a new data is fed to the algorithm its consistency with baseline signals is checked and any relevant difference can be seen as a deviation from the actual case. In SHM framework this approach can be used when the acquisition of defect signals is unfeasible and only baseline cases can be acquired. As a drawback the only information that can be obtained is a sort of distance from the baseline set of data, typically impeding any possibility to extract further information from the signals acquired, i.e. position or severity of the damage;
- *supervised* learning: a set of examples is taken as input by the algorithm as well as their relative label. The algorithm adapts by trials and errors its parameters

in order to correctly relate examples to labels minimising predictions' error. This technique is useful when there is the need to extrapolate multiple parameters from the inputs, which in SHM framework means to understand not only the presence but also location and severity of a defect. The limitation of this technique stands in the need of examples from all the possible cases that we want the algorithm to classify.

In this thesis only damage detection task is queried. Fundamentally, the aim is to select the best algorithm that will be used for future studies where localisation and damage quantification tasks will be considered. This explains the choice to implement supervised learning techniques in this work.

The scope of ML algorithms in supervised learning is to find the function relating input to output without knowing its shape a priori. This implies the utilisation of several functions with multiple degrees of freedom connected one to the other to form the so called Neural Network (NN): every neuron computes a linear combination of the inputs and passes the output through a non-linear function, which in turn is passed as input to another neuron and so on. Every module computing these operations is called neuron and the strength in having a network of neurons stands on the fact that, depending on their arrangement, they are able to describe from the easiest functions to the most difficult ones. All these characteristics can drastically reduce the number of theoretical models that should be needed to describe a particular phenomenon.

NNs are composed by few basic features that are common to all types of architectures:

$\underline{x}$	inputs vector
$\underline{w}$	weights vector
b	bias
z	linear combination output
g	activation function
y	output

which are arranged as:

$$\begin{gathered} z = \underline{w}^T \underline{x} + b \\ y = g(z) \end{gathered} \tag{1}$$

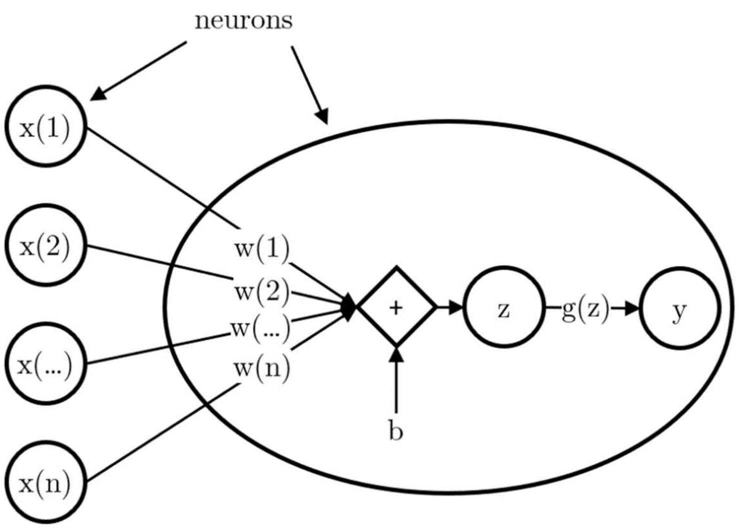


Fig. 2: operations occurring inside a single neuron

In Fig. 2 a neuron is graphically represented with all the operations it accomplishes, i.e. those of eq. (1). In a simple case, neurons are stacked in a layer and each one will compute its own output y from several inputs x. Examples of the most common activation functions used in ML algorithms are reported in Fig. 3 and the relative formulae are in eq. (2).

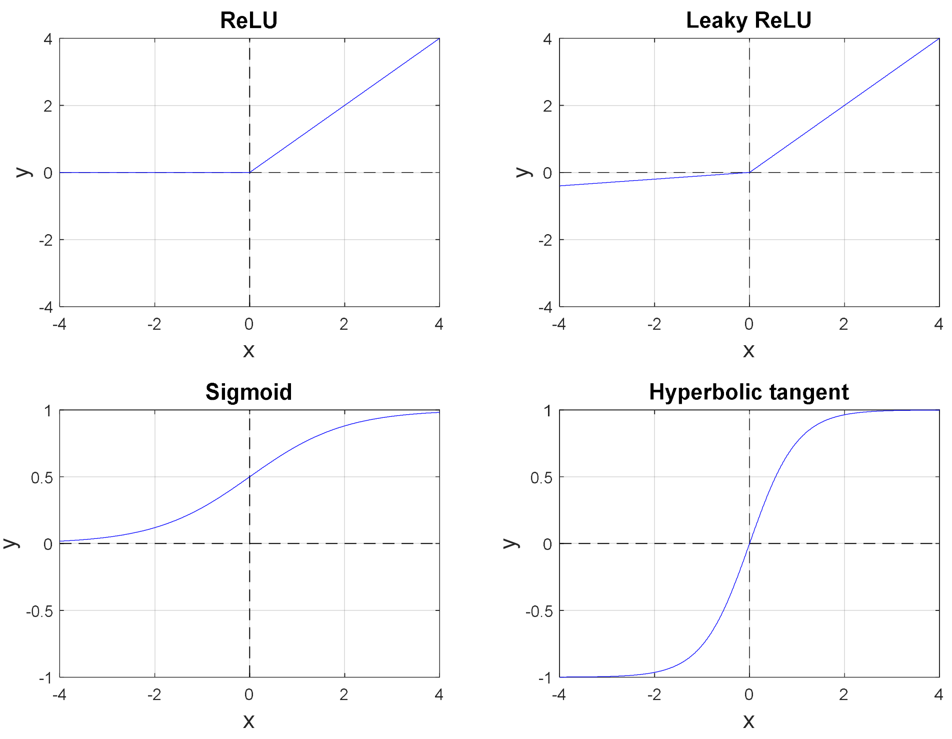
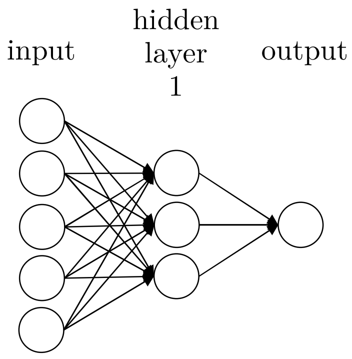


Fig. 3: most common types of activation functions

Relu	Leaky relu (usually λ=0,01)	Sigmoid (σ)	Hyperbolic tangent ($tanh$)	
$y = \mathrm{Max}(0, x)$	$y = \mathrm{Max}(\lambda x, x)$	$y = \dfrac{e^x}{e^x + 1}$	$y = \dfrac{e^x - e^{-x}}{e^x + e^{-x}}$	(2)

The activation function is a fundamental feature for neural networks because it introduces non-linearities in the problem, otherwise the network would be reduced to a simple matrix transforming inputs to outputs applying a simple linear multiplication. Let us consider for example a neural network composed by 2 layers:



The output value is computed as follows:

$$\begin{cases} y_{out} = g_{out}\left(\underline{w}_{out}^T \underline{y}_1 + b_{out}\right) \\ \underline{y}_1 = g_1\left(\underline{w}_1^T \underline{x} + \underline{b}_1\right) \end{cases}$$

$$y_{out} = g_{out}\left[\underline{w}_{out}^T g_1\left(\underline{w}_1^T \underline{x} + \underline{b}_1\right) + b_{out}\right] \tag{3}$$

If both g_1 and g_{out} are set to be linear (i.e. $g_1 = g_{out} = 1$), equation (3) simply becomes:

$$\begin{cases} y_{out} = \underline{w}_{out}^T \underline{y}_1 + b_{out} \\ \underline{y}_1 = \underline{w}_1^T \underline{x} + \underline{b}_1 \end{cases}$$

$$y_{out} = \underline{w}_{out}^T\left(\underline{w}_1^T \underline{x} + \underline{b}_1\right) + b_{out}$$

$$y_{out} = \underline{w}_{out}^T \underline{w}_1^T \underline{x} + \left(\underline{w}_{out}^T \underline{b}_1 + b_{out}\right)$$

$$y_{out} = \underline{w}^T \underline{x} + b \tag{4}$$

Being the purpose of neural networks to adapt to complex highly dimensional problems a simple linear fitting equation like eq.(4) would not be enough.
For the logistic regression problem treated in this thesis the aim is to come out from the last output with only one value between 0 or 1; in particular, values between 0 and 0,5 indicate signals not including a defect reflection while values between 0,5 and 1 indicate signals including a defect reflection. This task is well accomplished by the sigmoid function for which any input x can only assume an output y inside the range (0,1).

The updating process of weights and biases used in supervised learning is called back-propagation: it is basically a gradient method which exploits the derivative of the error function chosen with respect to every single weight and bias. The derivative brings with itself information about direction and intensity of the “slope” towards which the weight or the bias is driven in order to minimise the error function. Back-propagation takes the name from the rule of chain derivatives: the derivative of the error function with respect to weights and biases of the last layer affects the one with respect to weights and biases of the penultimate layer and so on towards the first layers, i.e. backward propagating in the structure of the neural network.
The choice of the error function to minimise depends on the problem that the ML algorithm is asked to solve. This thesis is devoted to a binary classification task: the output of the algorithm consists in the assignment of the input to one of two classes. In this case the use of two outputs is redundant indeed only one value is needed, which is compared to a threshold and assigned to one class or the other. One of the most common error function used for binary classification in logistic regression is reported in eq. (5): it accounts for the distance between the output predicted from the model ($\hat{y}$) and the actual value of the output (y).

$$\mathcal{L}(\hat{y}, y) = -[y \, log \, \hat{y} + (1-y) \, log(1-\hat{y})] \tag{5}$$

During the training phase of the network, the training examples are taken simultaneously so there is the need to define a cost function J, whose task is to account for all the data at the same time increasing the speed of the learning procedure:

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}, y)$$
$$J(w,b) = -\frac{1}{m}\sum_{i=1}^{m} \{y \, log \, \hat{y}(w,b) + (1-y) \, log[1-\hat{y}(w,b)]\} \tag{6}$$

where m is the number of examples considered to train the model.

The time needed to train ML algorithms increases with the number of examples, whose amount should be the highest possible. This leads to the development of several techniques to speed up the updating process of weights and biases. For example, it is a good practise to always normalise the inputs of a NN in order to let the range of data be the least spread possible. This procedure has the following benefits:

- the input data of every neuron will have all values with mean and variance as close as possible to 0 and 1, respectively. Looking at Fig. 3 it is clear that non-linearities of activation functions are concentrated around 0. If values are too big or infinitesimal the weights would need several updates before compensating the different orders of magnitudes of the inputs;

- particularly for sigmoid and hyperbolic tangent, there is a trend for which the higher the input value (x) the lower the slope of the function, which is strictly related to the learning procedure of the algorithm. Indeed, the lower the slope the lower the learning speed of the algorithm during back-propagation phase, i.e. when weights and biases are updated. If the inputs are spread towards high absolute values the derivative of the function would be close to zero, which can lead to a considerable slowdown of the learning procedure;
- being normalised values comparable to unity, there is no need for weights and biases to assume very big or very low values to compensate the input's ones. This will reduce the variations of weights and biases across the learning procedure and will help to speed up the convergence of the algorithm.

Weights and biases are often randomly generated at the first step of the training procedure. The random initialisation is necessary since it is quite unlikely that the cost function has only one local (thus also global) minimum, hence it is common to train multiple algorithms with the same architecture. Starting from different initial positions means to find different local minima of the cost function J, thus allowing the selection of the neural network parameters that minimise the error.

The minimisation of the cost function is made on the basis of a set of examples. The main issue occurs when the neural network is so complex that perfectly connects all inputs to outputs only for the set of examples used to train it. It might occur in fact that its performance can considerably decrease when a new set of examples is fed to the network. This issue is called overfitting and is well-known in Machine Learning environment. To reduce overfitting problem the dataset is usually split in 3 parts:

- *training set*: set of examples used to train the ML algorithm. These examples are the ones that are used to compute the cost function and subsequently all the derivatives needed to update weights and biases;
- *validation set*: set of examples which are not present in the training set. The output of these examples are computed at each updating step but are not considered to actively update weights. The validation set is needed to prevent overfitting (Fig. 4), i.e. it avoid the algorithm to focus too much on reducing the error of the training set;

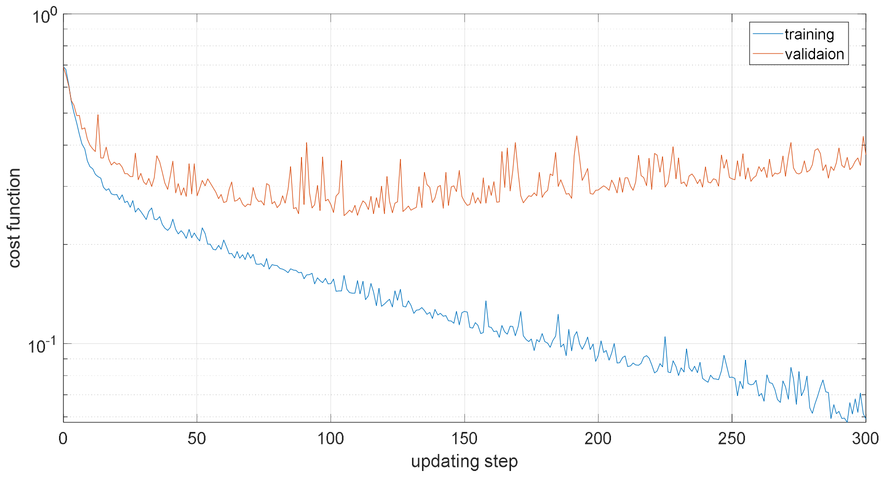


Fig. 4: overfitting to the training set can be seen in the cost function of the validation set, which starts to increase after about 100 steps

- *test set*: set of examples never included in either training or validation sets. This set is used only after the weights and biases update phase and is used to test the effective performance of the trained algorithm.

Novelty of this study

In literature there are a number of studies on different techniques that can be used to exploit Lamb waves. Most of them rely on sophisticated pre-processing of the signals, exploitation of basic ML algorithms and the need of a large number of sensors to have reliable predictions. In this study the main novelty is the effort in tuning deep ML systems to reduce the complexity of the pre-processing stages, while still obtaining a high level of reliability of predictions. The main goals and tasks can be summarised in:

- comparison of different types of architectures able to extract information from Lamb wave signals. To this aim the following architectures are investigated:
  - Multilayer perceptron (MLP), which is the basic architecture that can be used for general purposes. It has been chosen as reference to evaluate the real benefits that more complex architectures can furnish;
  - Long Short-Term Memory (LSTM) [8], one of the most developed, robust and widely implemented architectures among all the Recurrent Neural Networks (RNNs) proposed in literature;
  - WaveNet [9], a Convolutional Neural Network (CNN) specifically developed for raw audio generation and here adapted to suit ultrasonic signals and the binary classification task that this thesis is pursuing;

- the number of sensor used simultaneously is always kept at two, one emitter and one receiver. This relates the goal of extracting useful information on the health status of a structure with the lowest possible number of sensors. This can be made possible by exploiting multiple reflections of the waves when hitting the boundaries of the structure;
- the pre-processing required for a new signal to be fed to the ML algorithm needs to be kept at minimum, i.e. it only consists in applying the BSS technique to compensate for different wave speeds due to different temperatures, which however can be easily implemented;
- the robustness of the ML algorithms was also tested for temperature ranges outside those characterizing the training procedure.

Chapters content

This thesis is organised in the following sections:

Chapter 1: Dataset

> The characteristics of the numerical measurement set are defined in terms of geometry of the plate, position of the sensors, location of the defects and type of excitation signal. The signals obtained from the simulations are corrupted with noise and temperature compensated via BSS. The dataset is divided into training, validation and test sets, as required to train the different ML algorithms.

Chapter 2: Machine Learning architectures

> MLP, LSTM [8] and WaveNet [9] architectures are presented with a brief explanation of their structure. The procedure followed to tune the hyperparameters of each algorithm is reported step by step as well as the results obtained. Finally, the best algorithms belonging to the three types of architectures are compared and the best one is chosen.

Chapter 3: Temperature extrapolation capability

> The best class of algorithms is tested on the detection of defects from signals collected at temperatures out of the range used to train and validate the algorithms.

Chapter 4: Experimental dataset

> The performance given by the best class of algorithms is tested on a dataset acquired from an experiment.

Chapter 5: Conclusions

The aim of the research is recalled and the final results are presented.

1. Dataset

This chapter presents the main steps which were followed to create the dataset used to train, validate and test the different Machine Learning algorithms. The steps are:

- definition of the physical problem and geometrical aspects;
- selection of the shape of the excitation signal;
- structure of the dataset obtained from numerical simulations;
- pre-processing of the numerical data;
- splitting of dataset into training, validation and test sets.

The FE model and simulation results used in this study have been produced by Dr. Stefano Mariani[1] and Dr. Quentin Rendu[2], which I had the opportunity to work with at Imperial College.

1.1. Parameters used for the numerical simulations

1.1.1 Geometry

A simple squared steel plate has been considered, with dimensions of 500 x 500 x 5 mm. The emitter is positioned in the center of the plate (Fig. 5). Each defect is a through-thickness hole with approximate diameter of 4.6 mm, which was inserted in 100 different positions (separated by 50mm in both "x" and "y" directions, Fig. 6). The signals were

[1] Research Associate, Imperial College London, London, SW7 2AZ, UK
[2] Research Associate, Imperial College London, London, SW7 2AZ, UK

recorded from 625 different positions (separated by 20mm in both “x” and “y” directions, Fig. 7), hence simulating 625 possible positions of the sensor used as receiver. Only three of these positions have been used for the study described in this thesis, as later explained.

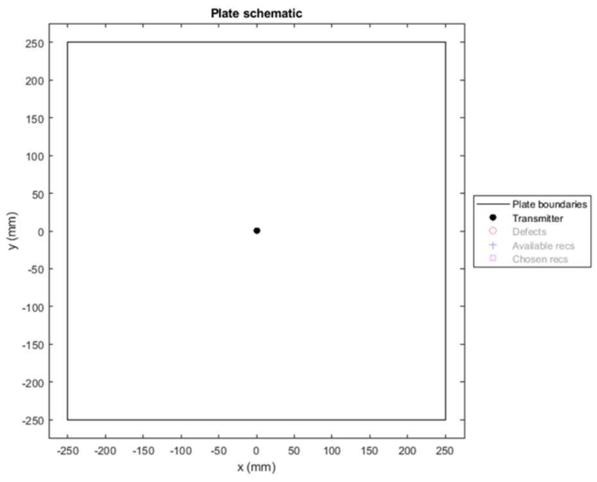


Fig. 5: steel plate dimensions and position of the emitter

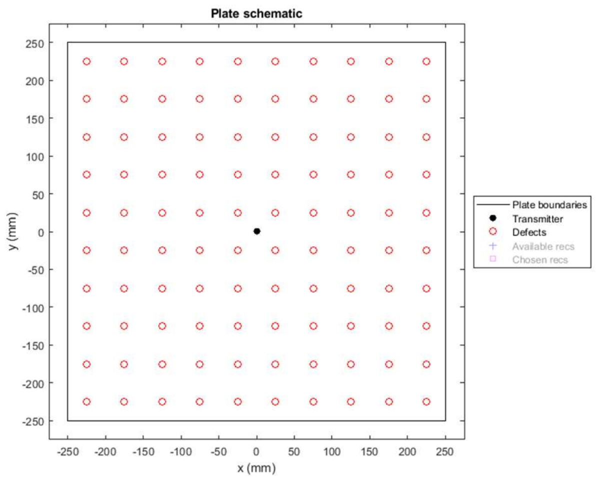


Fig. 6: position of defects on the plate

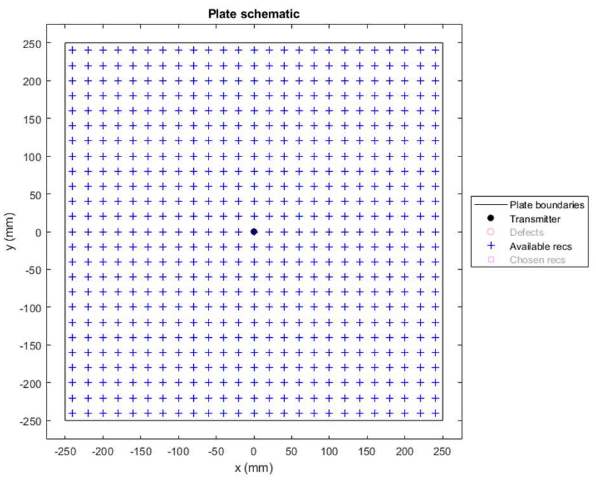


Fig. 7: position of the available receivers on the plate

The aim of the study is to compare different Machine Learning algorithms in detecting defects aiming to a real-case scenario, i.e. keeping the cost of setup, measurement and maintenance at the lowest. For example, this can be achieved by reducing the number of sensors on the plate. At a minimum, one could use only one sensor (acting as both emitter and receiver); however, in this study it was decided to consider the next step in the direction of increasing costs, i.e. to use two sensors, one as emitter and the other as receiver. This allowed to reduce the number of actual FE simulations when inserting the defect at different locations, by exploiting the available symmetries. By carefully selecting the position of the receiver, i.e. by avoiding to place it on the horizontal or vertical axes passing through the centre of the plate as well as on the two diagonals, any FE simulation performed by inserting a defect in one of the four quadrants in which the plate can be partitioned can be mirrored to mimic the presence of the same defect in the other three quadrants. In this particular case, the solution of the FE simulation has been computed separately for each one of the 25 defects located in the top-right quadrant of the plate (red circles in Fig. 8). The solution for the defects in the other three quadrants (green circles in Fig. 8) have been obtained by mirroring those in the first quadrant with respect to the horizontal and the vertical symmetry axes. In Fig. 8 an example of the mirroring procedure is represented by the darker circles.

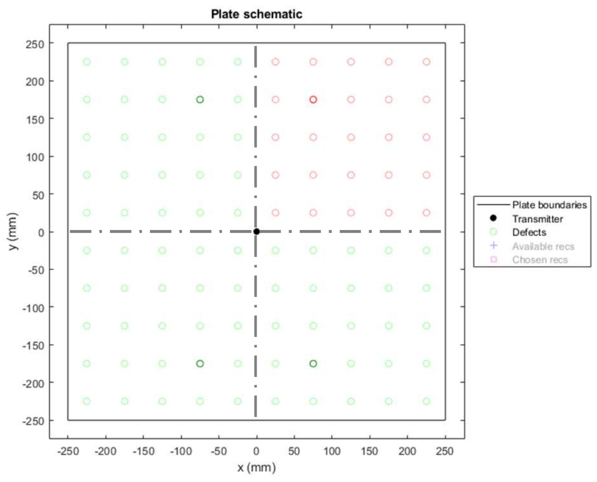


Fig. 8: mirroring scheme of the solutions of the FE simulations. Red circles represent the defects for which the solution of the numerical problem has been computed via numerical simulations; green circles represent the defects for which the solution of the numerical problem has been obtained by mirroring those of the symmetric red "parent" defects. The symmetries exploited are the horizontal and vertical dash-dot axes passing by the centre of the plate. Darker circles highlight an example of set of defects for which the numerical solution is equivalent but symmetric.

Given these considerations three different positions for the receiver have been selected (Fig. 9, coordinates are reported in Tab. 1) and used throughout the whole study. They were used one at a time coupled with the emitter with the aim to create different datasets. The three datasets created from these three receivers were used to test the dependency of the performance of the various ML algorithms on a particular choice for the receiver position with respect to the emitter.

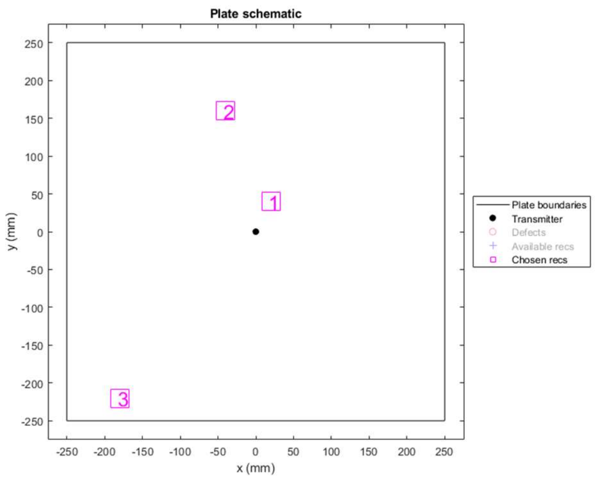


Fig. 9: locations of the three receivers used in this study

Sensor	X coordinate [mm]	Y coordinate [mm]
Emitter	0	0
Sensor 1	20	40
Sensor 2	-20	160
Sensor 3	-180	-220

Tab. 1: locations of the three receivers used in this study

1.1.2 Excitation signal

The analysis of the dispersion curves for the test plate guided the choice of the excitation signal. The dispersion curves consist in graphs having in x axis the frequency of the excitation and on y axis the phase velocity or the group velocity of the propagating waves. The former identifies the velocity of propagation of the different phases of the signal, the latter identifies the velocity of propagation of the envelope of the signal. Dispersion curves depend on the characteristics of the material and the thickness of the plate considered: those shown in Fig. 10 belong to a 5mm thick steel plate, the one used for this study, and have been obtained from Disperse, a software developed in the Non-Destructive Evaluation group at Imperial College London [10].
The excitation signal should have the following characteristics:

- high frequency in order to be affected by small defects and allow their detection;
- frequency low enough to excite only the two fundamental modes (S_0 and A_0);
- narrowest possible bandwidth to limit signals dispersion;
- lowest possible amount of power needed for its generation.

A trade-off must be found between the first two points: looking at Fig. 10 it can be seen that the excitation frequency must be kept at around or below 300 kHz, otherwise S_1 and A_1 modes would be also excited. The third and fourth points require an excitation signal long enough to excite a narrow bandwidth, keeping in mind that the longer the signal the higher the power required. To limit signals' dispersion it is also advantageous to choose the excitation frequency in a region where the slope of the curves is as close as possible to zero. All these considerations led to the choice of an excitation signal consisting in 5-cycles, Hanning windowed sine wave with a frequency of 300 kHz (Fig. 11). Group and phase velocities of S_0 and A_0 modes at the chosen frequency are reported in Tab. 2. The chosen excitation signal was applied as a force directed normally to the surface of the plate, hence exciting almost exclusively the A_0 wave mode.

Dispersion curves

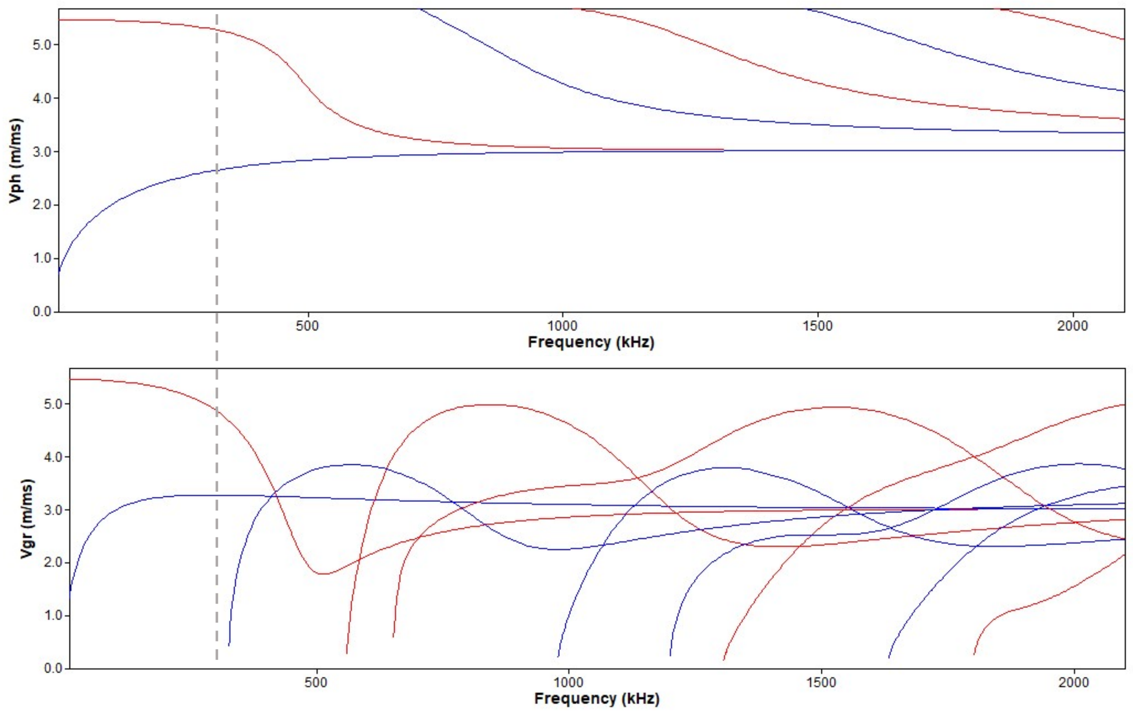


Fig. 10: dispersion curves for a steel plate 5mm thick. In the upper plot is represented the phase velocity while in the lower plot is represented the group velocity. Red lines represent S_n modes while blue lines represent A_0 modes. The dashed line indicate the chosen excitation frequency (300 kHz)

Excitation signal

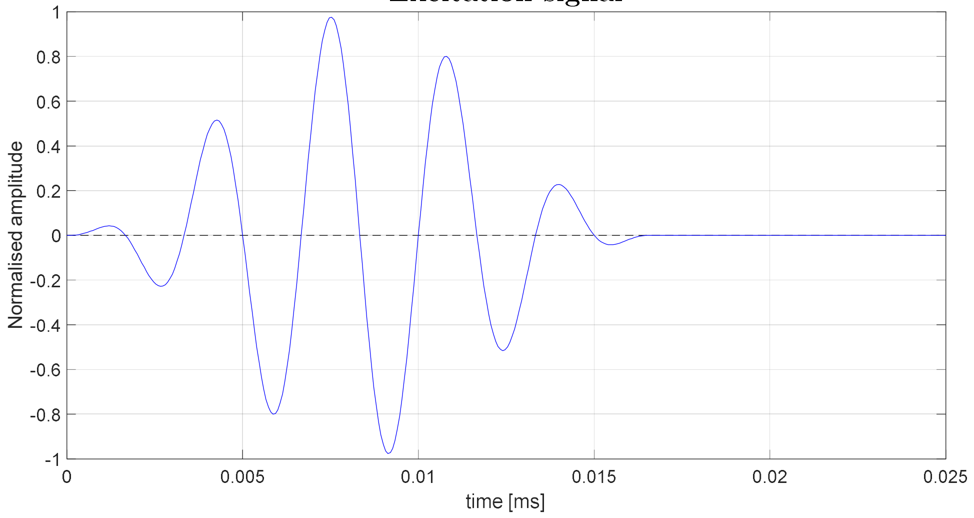


Fig. 11: final shape of the excitation signal

Velocity	S_0 [mm/ms]	A_0 [mm/ms]
V_{ph}	5321	2614
V_{gr}	4917	3282

Tab. 2: phase and group velocities for S_0 and A_0 modes for an excitation source of 300 kHz

1.1.3 FE model

After setting the geometry of the problem and the excitation signal, a software had to be chosen to perform the FE simulations. Pogo software was chosen for the task since it is well-suited to solve wave propagation problems involving a large number of elements and in a fast manner [11].

One of the main difficulties affecting ultrasound-based SHM approaches is given by the effect of the varying temperature in the signal generated, propagating in the structure and then received, since temperature influences both sensors and plate response to excitations. In the FE models the effect of temperature have been addressed in two ways:

- the effects on the sensor behaviour have been modelled by varying the phase of the excitation signal as a function of the simulated measurement temperature [12];
- the temperature effect in the plate has been modelled by changing the material properties accordingly with the temperature.

Temperatures ranging from 20°C to 69°C at steps of 1°C were considered. The steel plate was modelled as an isotropic material, setting density, elastic modulus and Poisson's ratio based on literature values. Tab. 3 reports these values for the upper and lower temperature limits considered in this work. Every simulation was solved by linearly varying the properties of the materials with temperature, in order to mimic the testing in different conditions.

T [°C]	Density [kg/m^3]	E [GPa]	Poisson's ratio [-]
20	7908	219,0	0,2892
69	7895	215,5	0,2905

Tab. 3: material properties for the considered range of temperatures, i.e. from 20°C to 69°C

The plate was discretized using 10 million linear brick elements with 0,5 mm-long edges, and a simulation time-step of 50 ns. Both mesh size and simulation time-step were chosen in order to satisfy the accuracy and stability requirements typical of elastodynamic simulations via FE at the chosen excitation frequency of 300 kHz. Regarding accuracy, it is recommended to use at least 10 elements per shortest propagating wavelength [13]. The shortest wavelength corresponds to that of the lowest phase velocity propagating in the model, which is that of the A_0 mode at 69°C: the phase velocity is equal to 2606 m/s for a wavelength of 8,7 mm. Considering the most unfavourable case of propagation along the diagonal of a cubic element, the accuracy requirement was met by offering more than 10 elements per shortest wavelength. For stability, the Courant-Friedrichs-Lewy (CFL) condition dictates that the fastest propagating wave, this being the bulk longitudinal wave at about 6 km/s, must not travel more than one element in a single time-step [14]; by setting a simulation step of 50 ns the CFL requirement was also met.

The FE model were used to solve the one pristine case (no defects are present in the plate) as well as the 25 defective cases (one for each position of the defect in one of the four quadrants in which the plate was partitioned, that were then mirrored to cover the cases of the whole plate). The solutions for both pristine and defective models (1+25) were obtained by running the FE simulations for all the 50 different temperatures, i.e. simulating 50 cases for each one of the 26 models.

1.1.4 Signals from numerical computation

The results obtained from the solution of the FE analyses were stored in a single matrix with 4 dimensions:

$$dataset = dataset(defect \ , \ timestep \ , \ temperature \ , \ sensor) \qquad (7)$$

where

- *defect* ranges from 1 to 101: the first is the pristine signal (without any defect) while the other 100 are signals with one defect in all the possible positions;
- *timestep* contains all the samples taken in time domain: 2000 samples acquired with a sampling frequency of 4 MHz, i.e. one sample every 250 ns;
- *temperature* ranges from 1 to 50: from 20°C to 69°C at 1°C steps;
- *sensor* ranges from 1 to 3: the positions from which every signal is received.

An example of one such signal is reported in Fig. 12.

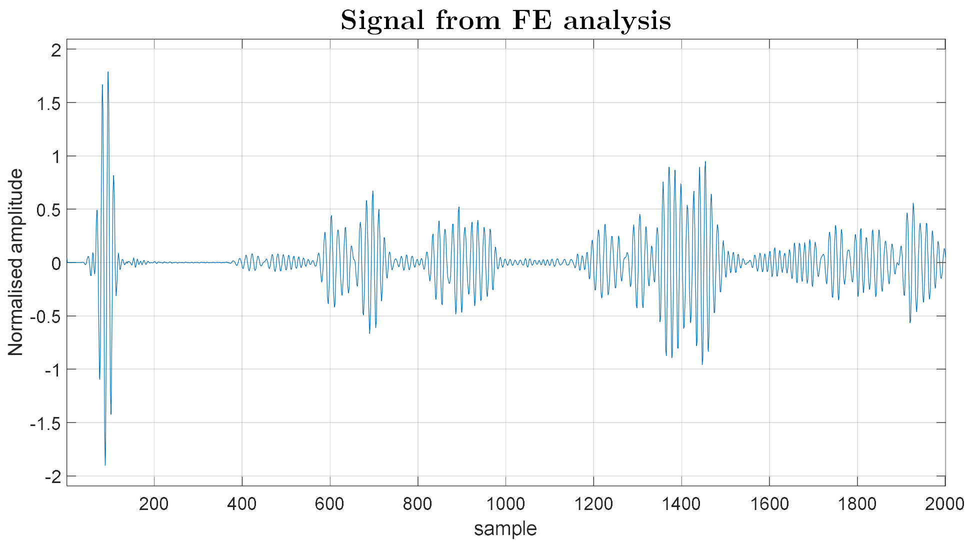


Fig. 12: example of one signal coming from the numerical simulations

The choice of the length of the signals (number of timesteps) is fundamental: the minimum length must be the time that the emitted signal needs to travel to the farthest defect (with respect to the emitter) and to be reflected back to the farthest receiver (with respect to the defect), as depicted in Fig. 13. To comply with this requirement, this distance was set to be 1.5 times the diagonal of the plate, thus:

$$timestep > \frac{1{,}5 \cdot diagonal}{V_{gr}^{A_0}} \cdot f_s = \frac{1{,}5 \cdot \sqrt{2 \cdot 0{,}5^2}}{3282} \cdot 4e^6 = 1293 \tag{8}$$

Where f_s is the sampling frequency of the signal. However, as previously mentioned, it would be advantageous to also include in the acquired signal some reflections from the boundaries of the plate, hence it was decided to acquire 2000 timesteps, i.e. an acquisition time of 0.5ms.

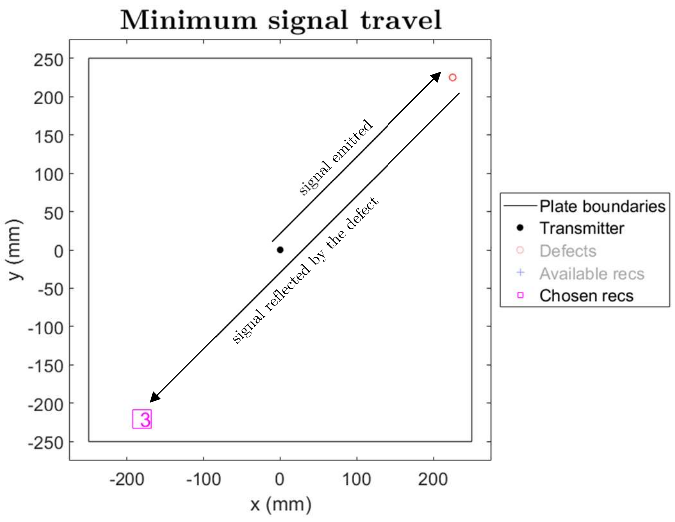


Fig. 13: minimum travel distance of the signal to capture defect information

Fig. 14 shows the signals captured by the receiver placed at the three locations of Tab. 1 for a pristine plate at 20°C. Using the dispersion curves of Fig. 10 and the frequency of 300 kHz of the input signal, it is possible to estimate the first arrival time for both S_0 and A_0:

$$t_{firts\ arrival}^{S_0} = \frac{d_{rec}}{V_{gr}^{S_0}} \quad ; \quad t_{firts\ arrival}^{A_0} = \frac{d_{rec}}{V_{gr}^{A_0}} \tag{9}$$

being d_{rec} the distance of the receiver from the emitter and $V_{gr}^{S_0}$ and $V_{gr}^{A_0}$ the group velocities of S_0 and A_0 respectively. Tab. 4 reports the results obtained by applying the velocities of Tab. 2 in eq. (9).

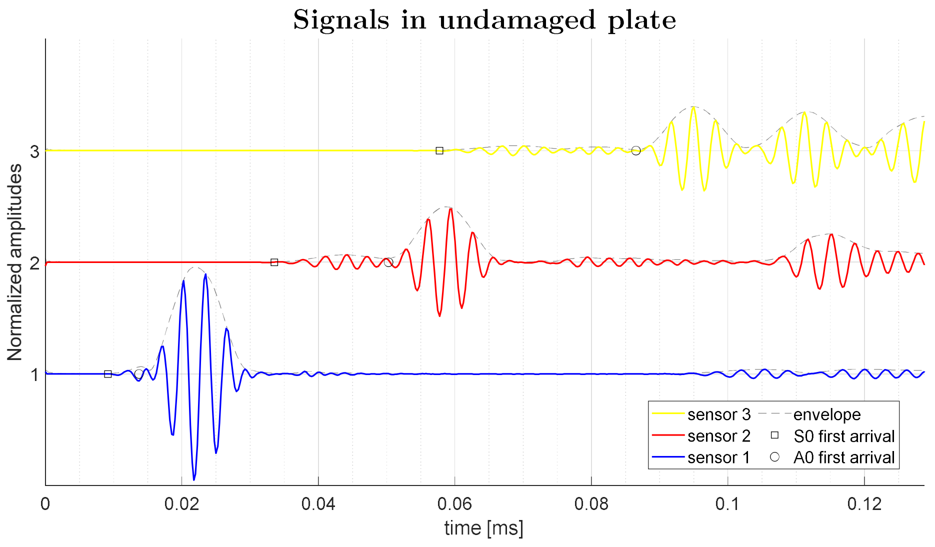


Fig. 14: signals captured by the receiver placed in the three chosen locations

Sensor	D [mm]	First arrival time [ms]	
		S_0	A_0
1	45	0,009	0,014
2	165	0,034	0,050
3	284	0,058	0,087

Tab. 4: first arrival times computed from dispersion curves data

Fig. 14 also shows that:

- the farther the receiver from the emitter, the lower the amplitude of the intercepted signal;
- anti-symmetric mode A_0 has a higher amplitude than the symmetric mode S_0: this is due to the fact that the excitation from the emitter is given in normal direction with respect to the plate, thus exciting almost exclusively A_0 mode;
- the closer the receiver to the boundaries, the faster the arrival of the reflections;
- looking at the envelope of the first wave packet arriving to the sensor it is possible to confirm what expected from the analysis of the dispersion curves of Fig. 10: the excited A_0 mode preserve its shape as it travels through the plate, hence reducing the complexity of the received signal.

1.2. Pre-processing of the signals before being fed to the ML algorithms

The numerical signals were processed prior to be fed to the ML algorithms in order to make them more realistic:

- the temperature effect on signals, which is one of the main issues when dealing with ultrasound-based SHM problems, can be mitigated by applying BSS [1] [2] technique;
- the signals obtained from the numerical simulations are not affected by instrumentation noise, thus an artificial noise is added in order to reproduce datasets with certain SNRs;
- the dataset must be properly divided into training, validation and test sets in order to effectively train all ML models

These 3 steps are now discussed in depth.

1.2.1 Baseline Signal Stretch (BSS)

The effect of temperature in the numerical model was already introduced in chapter 1.1.2. Three numerical signals received by the sensor number 1 from models at different temperatures are shown in Fig. 15:

- envelopes arrives at almost the same time since the distance between emitter and receiver is short. The main visible effect is the phase shift due to the different phase of the signal emitted in each of the three models;
- towards the end of the signal (Fig. 15, zoom B) the main visible effect is the different arrival time of the envelope in each of the three signals, due to the longer distance travelled by the propagating modes.

These distortions hinder the detection of reflections originating from defects. BSS technique is one of the most used temperature compensation approaches [1] [2]. A baseline signal is taken as reference and all the others are compressed or elongated in time to best fit the baseline signal (i.e. to reduce the RMSE of the signal obtained by subtracting the modified signal from the baseline). This method allows to obtain signals with equal time of flight (tof), independently from the temperature at which the signal is acquired.

In this study the baseline signal was set to be the pristine signal at 45°C (in the middle of the range between 20°C and 69°C) for all the receivers; the results are reported in Fig. 16 that shows the improvement given by this transformation: in zoom A the temperature effect is still dominated by the phase change given by the temperature effect in sensors; in zoom B the signals are almost completely overlapped.

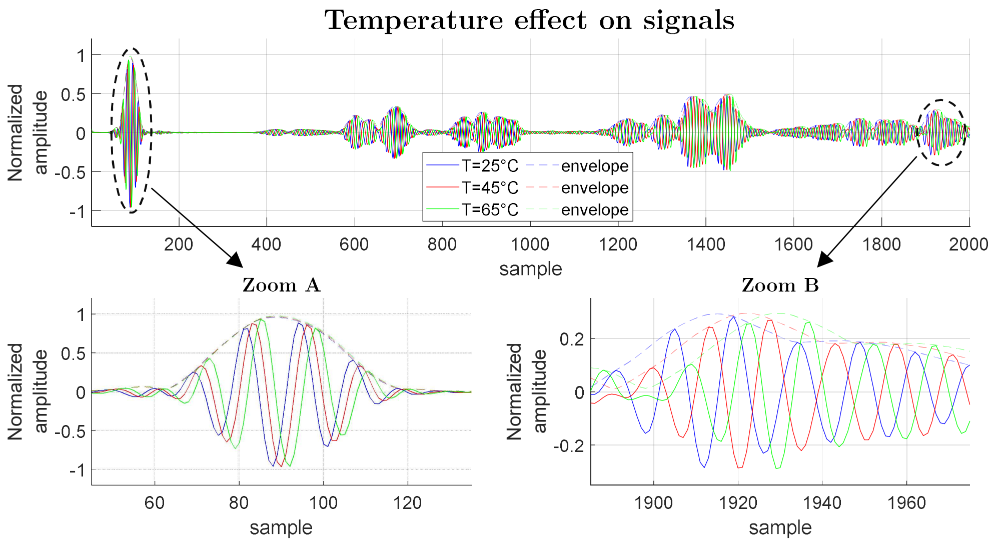


Fig. 15: signals taken at three different temperatures

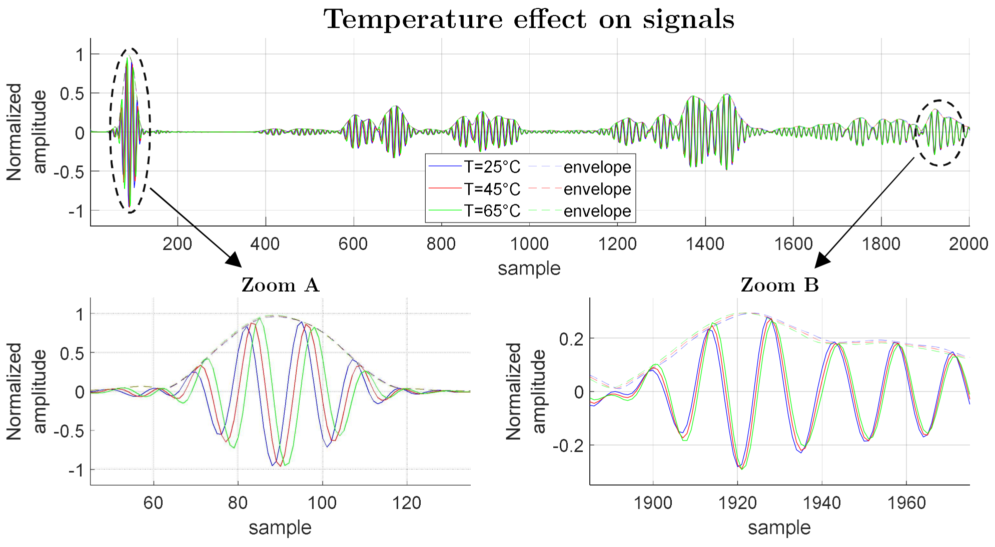


Fig. 16: signals taken at three different temperatures after BSS

As explained in the next section, noise was added to the signals and it would be more realistic to apply BSS to the noise-corrupted signals rather than to the noise-less signals as done in fig. 15. However, since the disturbance due to the introduced noise is significantly smaller than the largest peaks of the signals, only negligible differences would be obtained by applying BSS to the noise-corrupted signals rather than to apply noise on the BSS-compensated noise-less signals. Since at each iteration of the study an entirely different set of noise was added to the signals, in order to save computation time it was decided to first compensate all the noise-less signals with the BSS method and hence to simply apply noise at each iteration to the compensated signals.

1.2.2 Adding noise

When recording signals in a real case there are disturbances affecting the measurements given by the instrumentation noise. Lamb wave signals collected for SHM purposes only have meaningful frequency contents around the excitation frequency of the emitted signal. Bandpass filters are used to acquire signals in the same narrow bandwidth identified by the excitation, therefore random noise outside that frequency range is eliminated but remains almost undisturbed at the frequencies of interest. The ratio between the amplitude of the meaningful signal and the noise is called signal-to-noise ratio (SNR). The higher the SNR the higher the quality of the signal. To reduce the noise and increase the SNR there are usually two possibilities:

- averaging more samples taken subsequently (the lower the time elapsed between samplings, the lower the disturbances introduced by changings in boundary conditions);
- increasing the amplitude of the signal emitted.

Both these approaches require a higher amount of power to be performed so a compromise must be found.

Clearly, it would be ideal to detect whether a defect is present into a structure by using an algorithm able to extract such information from signals with low SNRs. In this study different levels of noise were added to the numerical simulated signals to test the capability of ML algorithms in dividing defects from pristine examples.
The SNR here is defined as the ratio between the envelope peak of the A_0 signal received by sensor 1 (about 45 mm form the emitter) in the pristine model and four times the standard deviation of the artificial noise, which was formed by 1) producing a normally distributed random sequence of numbers and 2) band-pass filtering it between 270 kHz and 330 kHz (Fig. 17).

During the development of the neural network architectures of chapter 2, a large number of different noise multiplying factors have been used to test the ML algorithms performance in different conditions. Finally, four different levels of noise were selected to be used as test bench to compare the different architectures. Fig. 18 shows the SNR levels compared with the relative amplitude of the defects, which were computed as the maximum absolute value from the subtraction of any defective signal from the pristine signal (at any temperature).

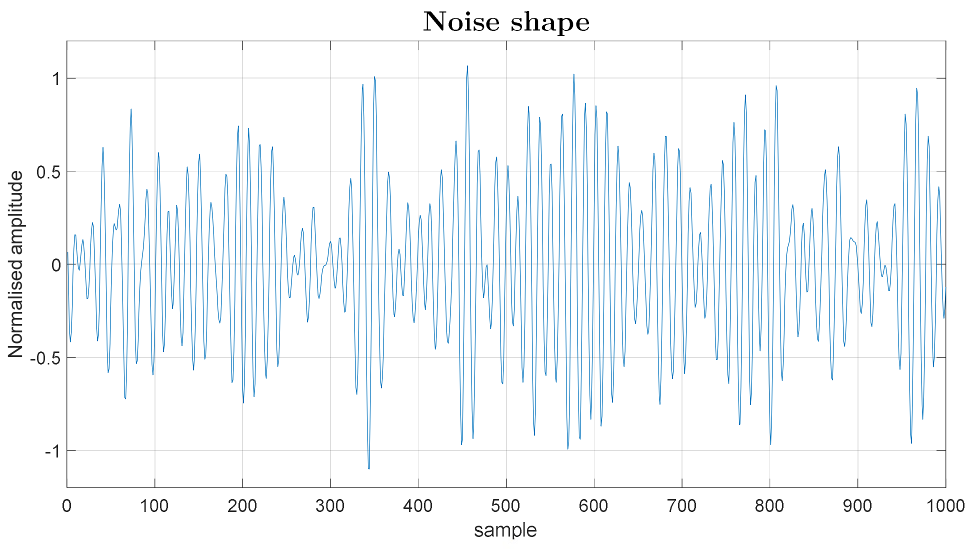


Fig. 17: example of noise after it has been filtered in 270-330 kHz bandwidth

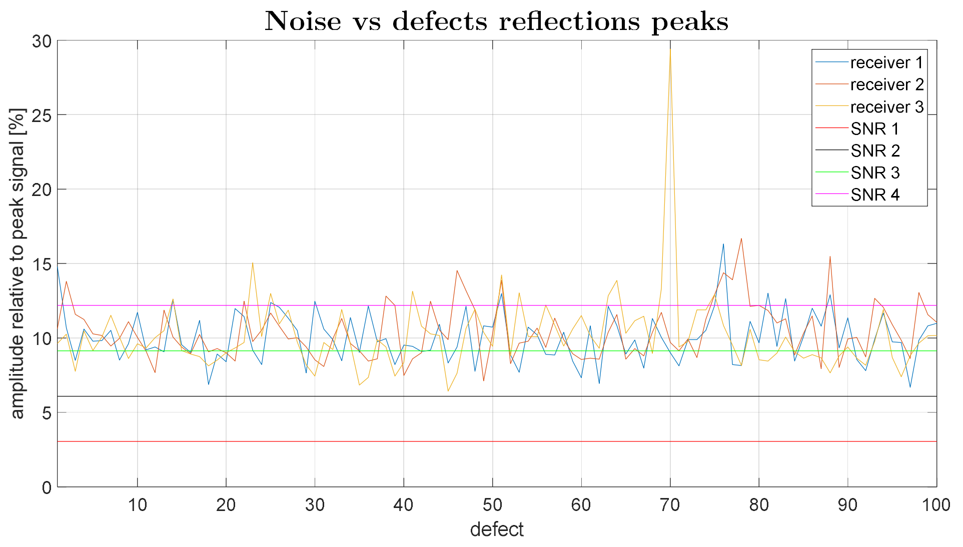


Fig. 18: two macro sets of lines are represented: the first three of the legend represent the maximum absolute values of the defects' reflections for all the temperatures for three receivers; the last four of the legend represent four times the standard deviation of the four noise levels. Each line is normalised with respect to the maximum absolute value of the pristine signal collected by receiver 1 (about 45 mm from the emitter).

The next figures show examples of the effects of the added noise on the signals: pristine and defective signals are represented overlapped on the upper subplot, while their difference is emphasised in the lower subplot. In particular, Fig. 19 shows signals without noise, Fig. 20 shows signals with the lowest noise (SNR 1) and Fig. 21 shows signals with the highest noise (SNR 4). Comparing the differences between pristine and defect signals (lower subplots) for the three considered cases can be clearly appreciated how without noise, as well as for the SNR 1 case, it would be less challenging to detect a defect in the plate; in case of SNR 4 the defect detection would be more difficult since the effect of the defect is confounded within the noise (their amplitudes are comparable).

Fig. 22 to Fig. 24 show similar plots with the effect of another defect, nr.71, which produced smaller reflections with respect to the previous case. The amplitude of the reflections of the defect is comparable with that of the lowest noise considered in this study (SNR 1) and completely confounds with that of the highest noise (SNR 4).

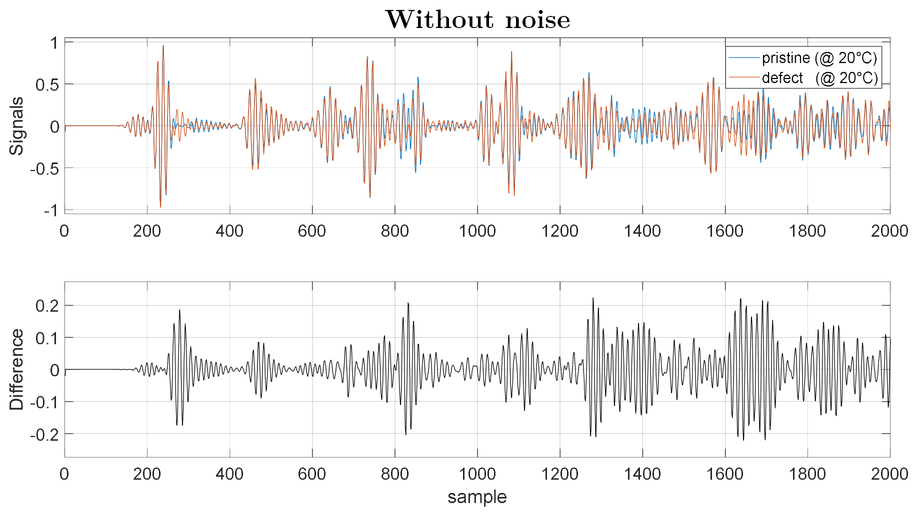


Fig. 19: impact of a defect in Lamb waves propagation for noise-less signals; defect nr.79, which produced the largest reflection seen by receiver 2; pristine and defective signals are taken at the same temperature

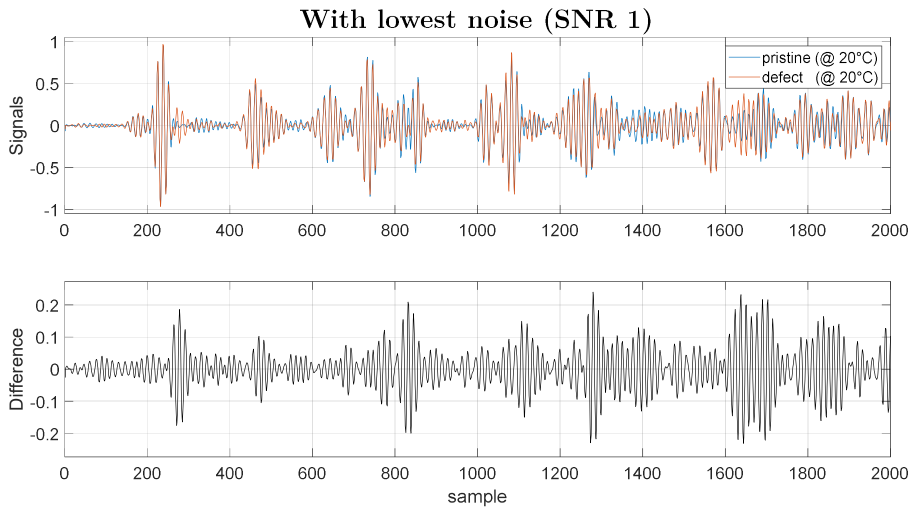


Fig. 20: impact of a defect in Lamb waves propagation for signals with the lowest noise (SNR 1); defect nr.79, which produced the largest reflection seen by receiver 2; pristine and defective signals are taken at the same temperature

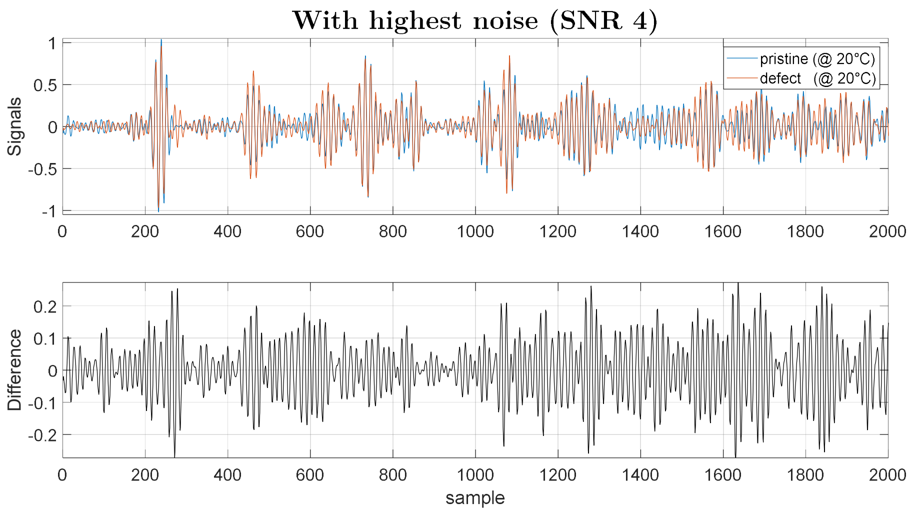


Fig. 21: impact of a defect in Lamb waves propagation for signals with the highest noise (SNR 4); defect nr.79, which produced the largest reflection seen by receiver 2; pristine and defective signals are taken at the same temperature

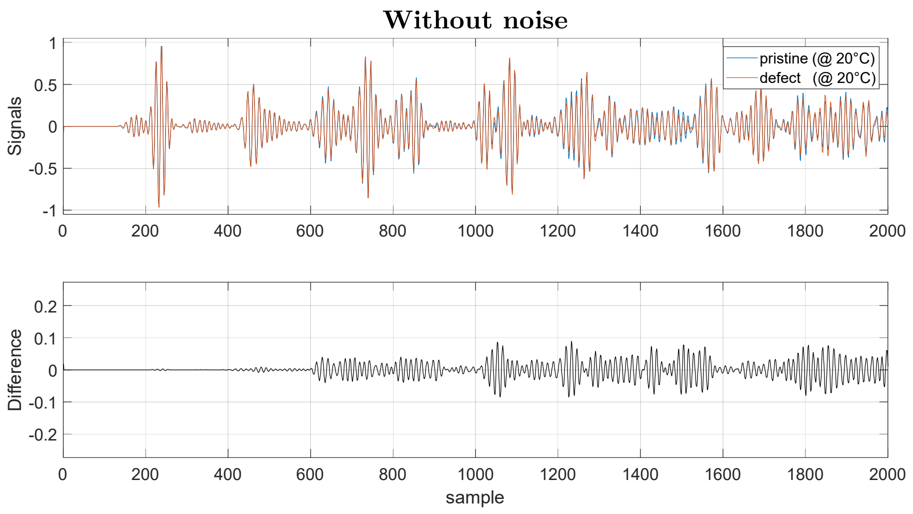


Fig. 22: impact of a defect in Lamb waves propagation for noise-less signals; defect nr.71, which produced the lowest reflection seen by receiver 2; pristine and defective signals are taken at the same temperature

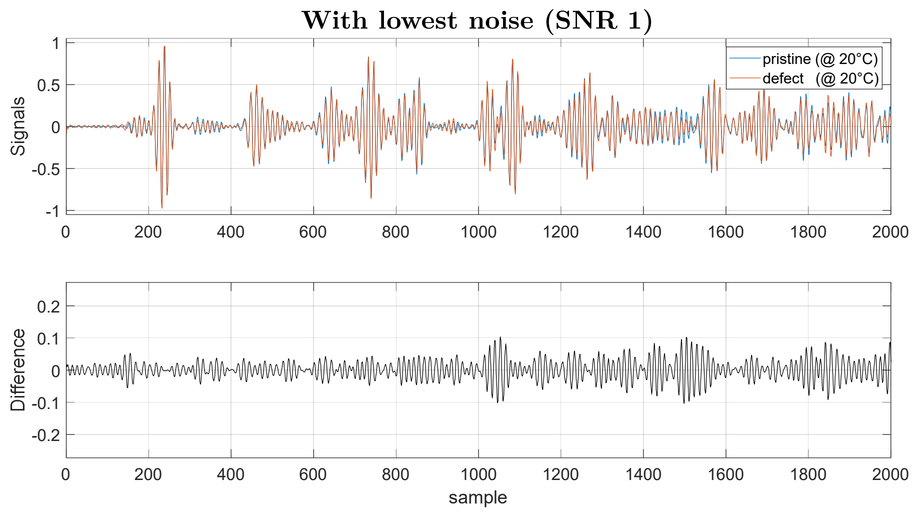


Fig. 23: impact of a defect in Lamb waves propagation for signals with the lowest noise (SNR 1); defect nr.71, which produced the lowest reflection seen by receiver 2; pristine and defective signals are taken at the same temperature

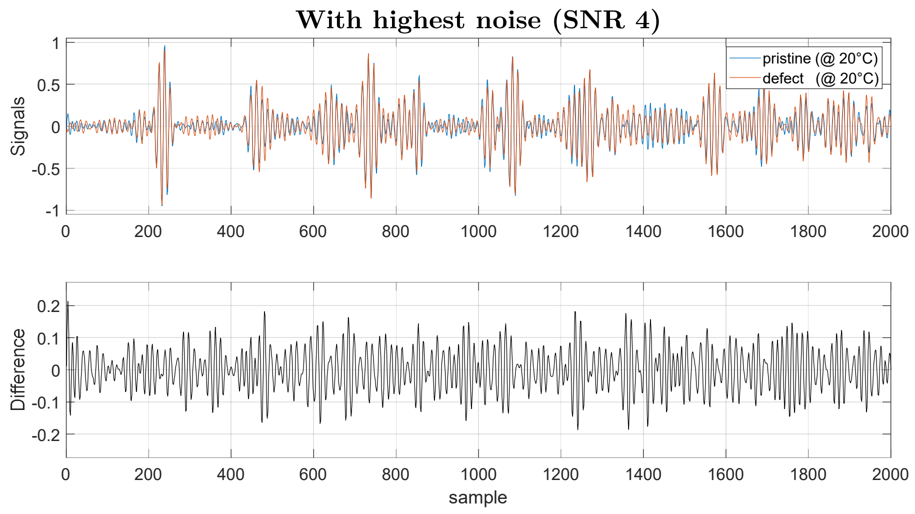


Fig. 24: impact of a defect in Lamb waves propagation for signals with the highest noise (SNR 4); defect nr.71, which produced the lowest reflection seen by receiver 2; pristine and defective signals are taken at the same temperature

It has to be noticed that plots from Fig. 19 to Fig. 24 were obtained by using pristine and defective signals simulated at the same temperature. In a realistic scenario, when a new measurement is acquired at a given temperature, a baseline signal taken at the same exact temperature might not be available. Therefore, the ML algorithm (or any technique that can be used to analyse the signal) cannot simply subtract the pristine signal to the new one to determine whether a defect is present or not. Examples of this condition is represented from Fig. 25 to Fig. 30, which report the same cases of Fig. 19 to Fig. 24 (three noise levels for two defect positions) but with defect and pristine signals taken at different temperatures.

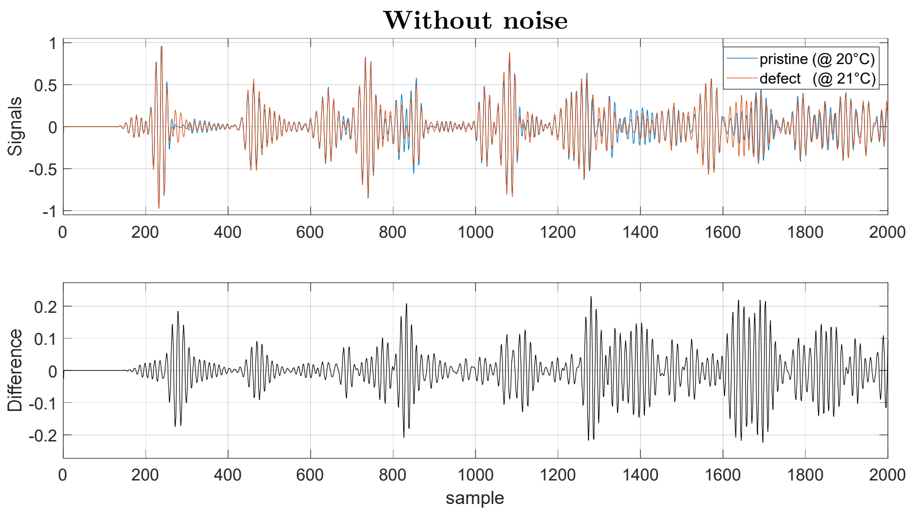


Fig. 25: impact of a defect in Lamb waves propagation for noise-less signals; defect nr.79, which produced the largest reflection seen by receiver 2; pristine and defective signals are taken at temperatures differing of 1°C

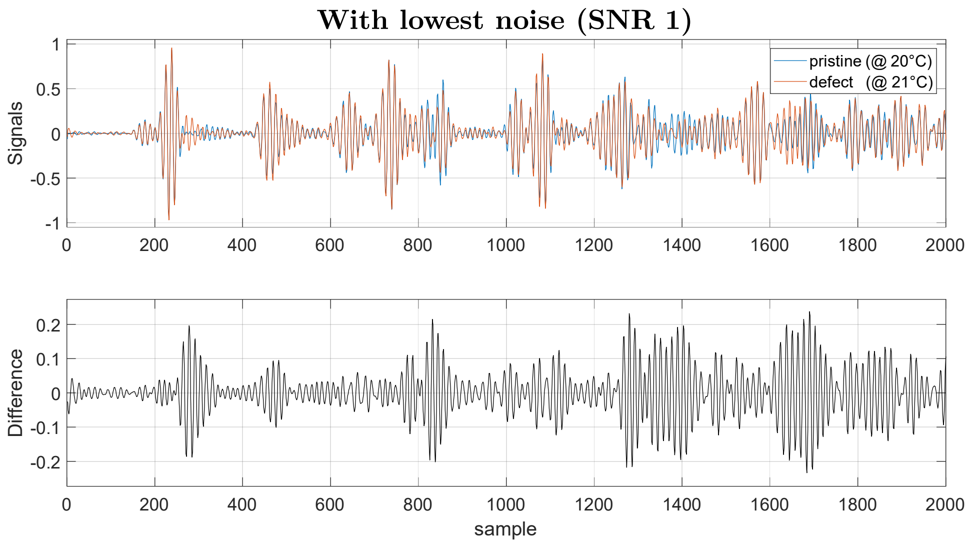


Fig. 26: impact of a defect in Lamb waves propagation for signals with the lowest noise (SNR 1); defect nr.79, which produced the largest reflection seen by receiver 2; pristine and defective signals are taken at temperatures differing of 1°C

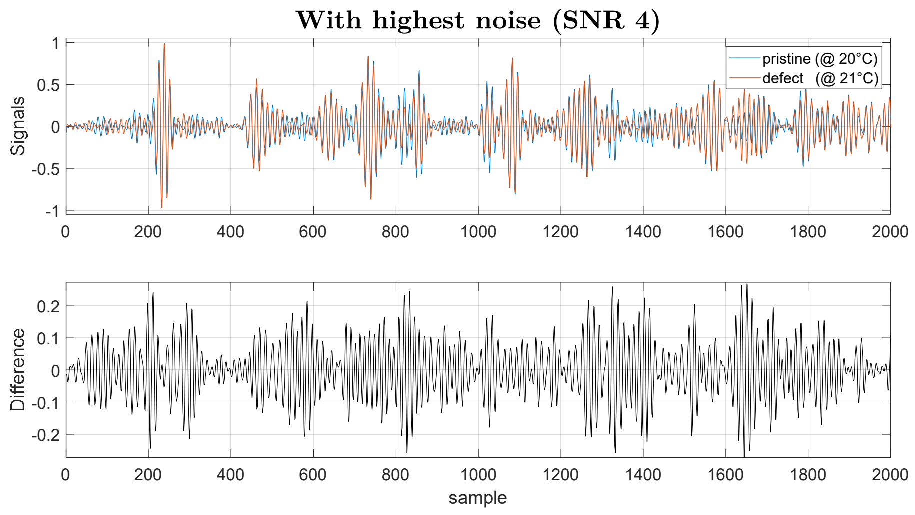


Fig. 27: impact of a defect in Lamb waves propagation for signals with the highest noise (SNR 4); defect nr.79, which produced the largest reflection seen by receiver 2; pristine and defective signals are taken at temperatures differing of 1°C

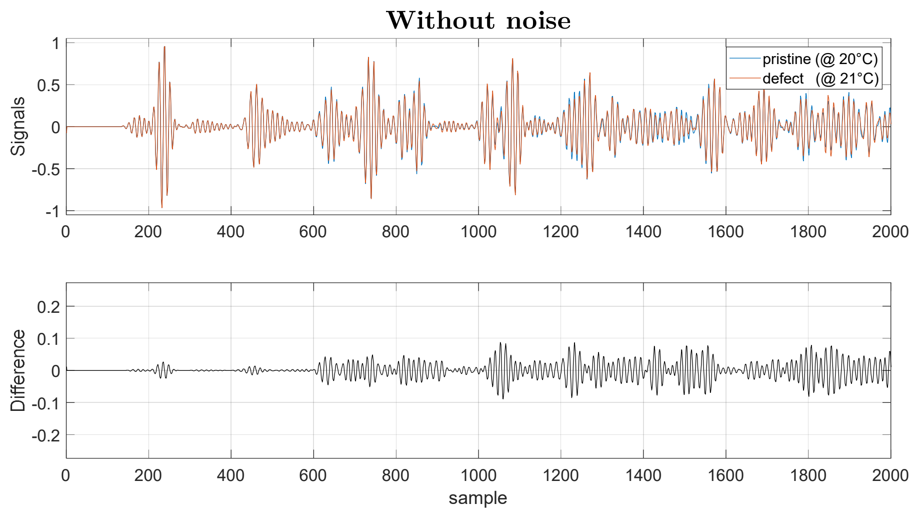


Fig. 28: impact of a defect in Lamb waves propagation for noise-less signals; defect nr.71, which produced the lowest reflection seen by receiver 2; pristine and defective signals are taken at temperatures differing of 1°C

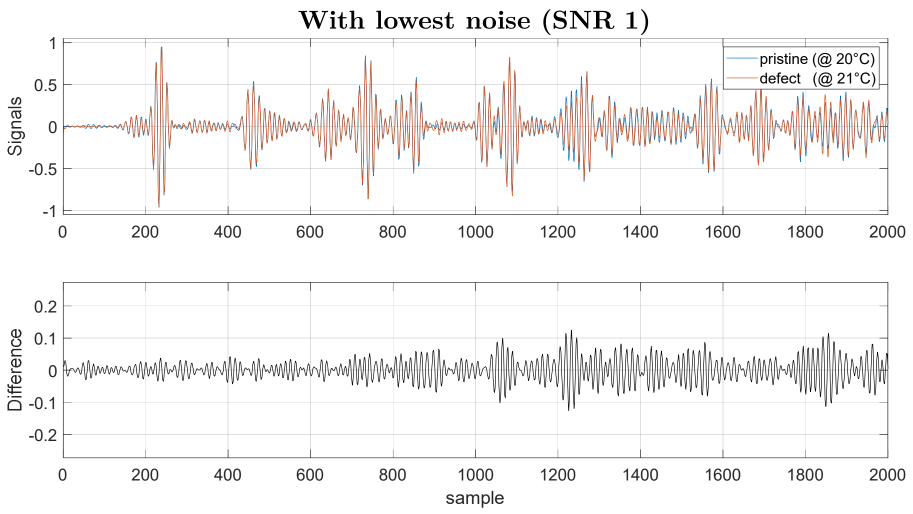


Fig. 29: impact of a defect in Lamb waves propagation for signals with the lowest noise (SNR 1); defect nr.71, which produced the lowest reflection seen by receiver 2; pristine and defective signals are taken at temperatures differing of 1°C

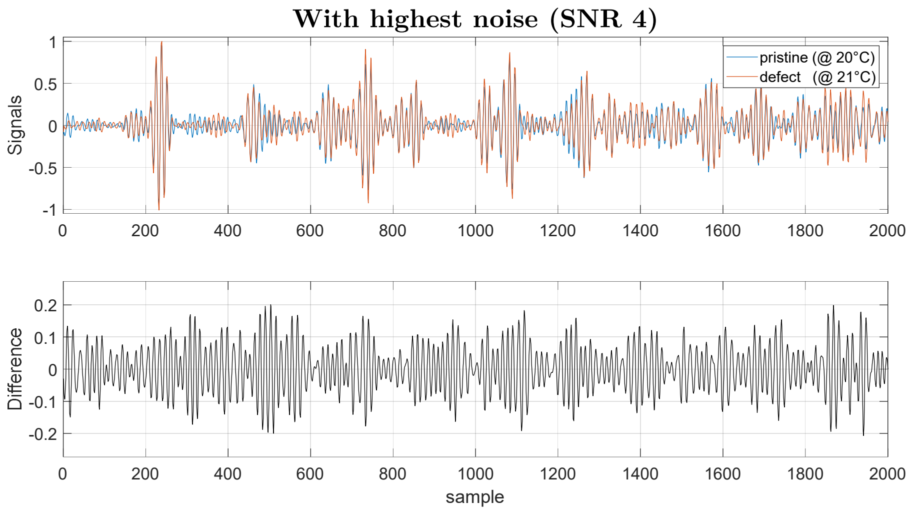


Fig. 30: impact of a defect in Lamb waves propagation for signals with the highest noise (SNR 4); defect nr.71, which produced the lowest reflection seen by receiver 2; pristine and defective signals are taken at temperatures differing of 1°C

The inaccuracy introduced by the difference in temperatures between defect and pristine signals can be seen in correspondence of the peaks of the Lamb wave packages in the initial part of the signals, where BSS imprecisions are amplified (recall Fig. 16 at page 26). Anyway, the influence of this inaccuracy is quite low and confounds with the noise artificially added, proving that BSS technique is quite effective in compensating temperature effects.

Prior to the training of each ML algorithm an entirely different set of noise was added to the signals. This ensured that every signal fed to any ML algorithm was different from all the others used in the previous iterations, thus reducing the influence of any deterministic disturbance on the performance of the algorithms.

1.2.3 Creation of training, validation and test sets

The last step prior to the training of ML architectures is the division of all the signals into training, validation and test sets. The dataset of eq. (7) must be reshaped in a 2D matrix, having “number of examples” rows and “length of signal” columns. For each receiver, every signal can be seen in 2 dimensions:

- defects positions on the plate (Fig. 31)
- temperature sampling (Fig. 32)

where training set signals are blue, validation set signals are green and test set signals are red.
The spatial division of defects in the three sets is chosen in order to have a distribution of 80-10-10% in training, validation and test set respectively; their location is taken randomly but keeping the spatial distribution as homogeneous as possible (Fig. 31). Every temperature at which the signal of a certain defect has been recorded is stored in the set where the defect belong (i.e. for every defect positioned in a set, 50 examples corresponding to 50 temperatures of signals with that defect are allocated in the same set; Fig. 32, defects table).
The pristine signal has the drawback of being only one against the other possible 100 defects. It is preferred not to assign all the temperatures of the pristine signals to the training set, otherwise the ML model could simply base its prediction from a subtraction of each defective signal from the pristine signal at the same temperature. The pristine temperatures are divided in order to let the algorithm train only on half of them (1 every 2) thus reinforcing its generalisation capability during validation and testing of the model. In this way the ML algorithm will not be able to compare pristine and defect signals at the same temperature during training, it has to rely on different features rather than a simple subtraction. Validation and test sets store 1 pristine signal every 4, relatively shifted by 2°C in order to do not overlap (Fig. 32, pristine table).

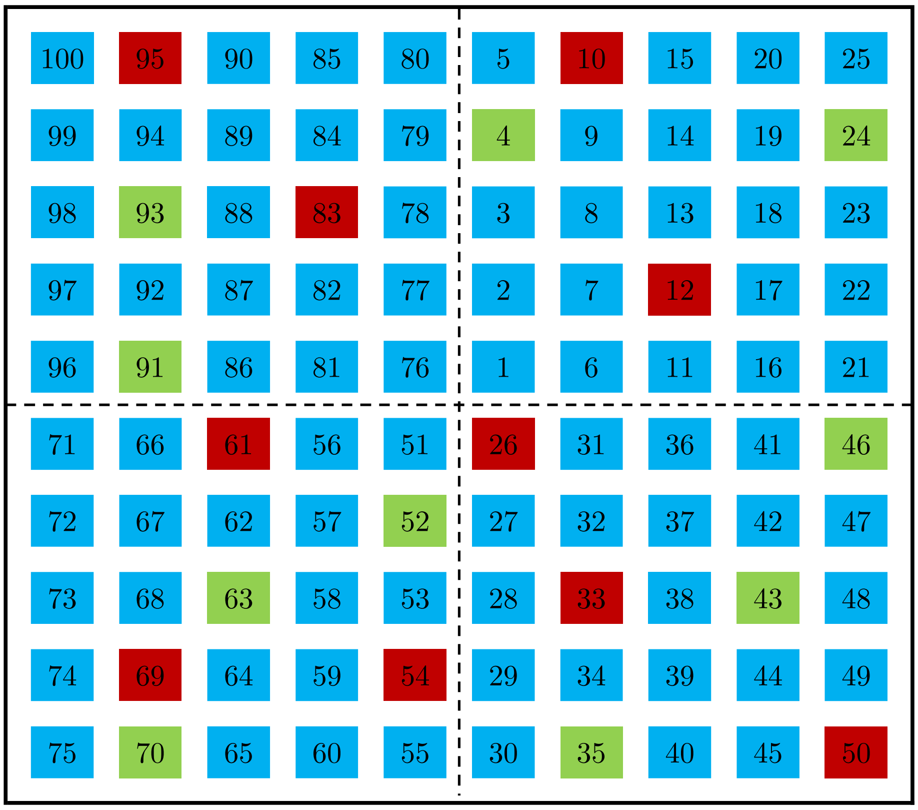


Fig. 31: division of defects positions in training set (blue), validation set (green) and test set (red). The order of defects' numbers is the one used in Matlab and Python codes, it does not have any particular meaning

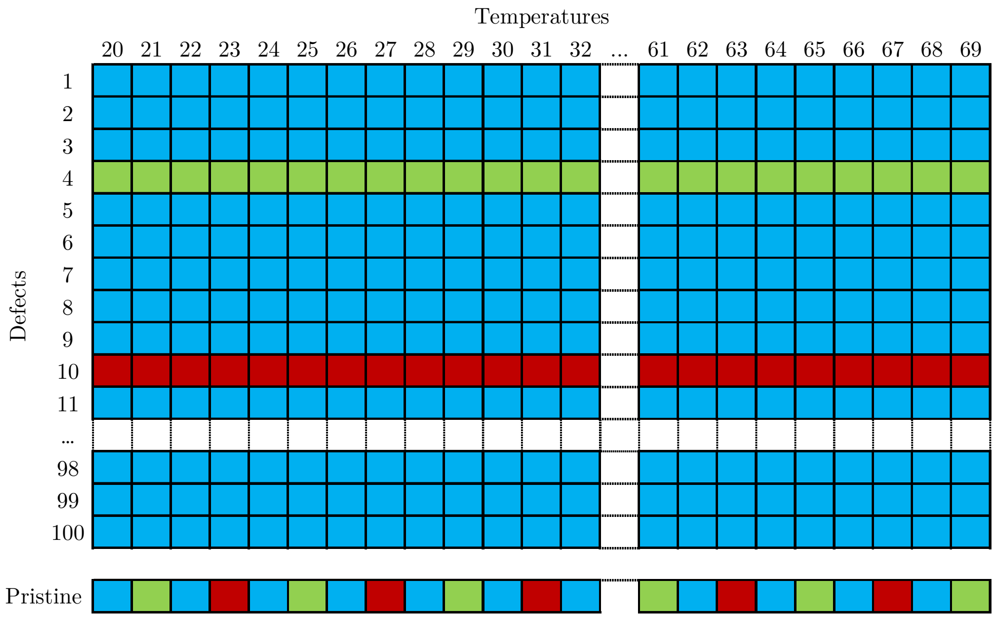


Fig. 32: division of defects and pristine temperatures in training set (blue), validation set (green) and test set (red)

The actual number of examples in the three sets is:

Set	Defect	Pristine	Total	Defect/Pristine ratio
Training	$80 \cdot 50 = 4000$	$1 \cdot 25 = 25$	4025	160
Validation	$10 \cdot 50 = 500$	$1 \cdot 13 = 13$	513	38
Test	$10 \cdot 50 = 500$	$1 \cdot 12 = 12$	512	42

Tab. 5: number of defect and pristine examples in every set – unbalanced distribution

The number of defect examples in every set is much higher than the number of pristine examples in the same sets. This might result in the ML algorithm tending to mark every example as defective, since by simply doing so it would already reach an accuracy close to 100%.

This unbalance in the number of labelled examples in the training set is a well-known problem in the ML community and can be addressed by creating multiple copies of pristine signals in order to achieve a defect/pristine ratio equal to 1, which means 50% of defect cases and 50% of pristine cases. This condition is obtained by copying pristine examples "defect/pristine ratio" times, as stated in the last column of Tab. 5. The final result is:

Set	Defect	Pristine	Total	Defect/Pristine ratio
Training	4000	$25 \cdot 160 = 4000$	8000	1
Validation	500	$13 \cdot 38 = 494$	994	1
Test	500	$12 \cdot 42 = 504$	1004	1

Tab. 6: number of defect and pristine examples in every set – balanced distribution

Pristine signals are copied before the noise is added to the whole dataset in order to have all signals different from each other.

The division of pristine and defect signals in training, validation and test sets is kept constant for all the trained architectures. In this way the performance obtained from the models will not be positively or negatively affected by different relative positions of receivers and defects in training, validation and test sets.

2. Machine Learning architectures

Training an ML model means to update its parameters (weights w and biases b) in order to make the most accurate predictions for a given task, starting from a dataset of examples of inputs and outputs, in this case signals and binary labels respectively. To achieve high performance there are an enormous number of variables that can be tuned, called hyperparameters: some of them are used to set general aspects of the training, others are strictly associated to the structure of the ML.

In this chapter the hyperparameters pertaining to all the architectures are first presented. After that, the Machine Learning algorithm types considered in this work are treated, considering their own hyperparameters and results. The ML architectures are based on:

- Multilayer perceptron (MLP);
- Long Short-Term Memory (LSTM) [8];
- WaveNet [9].

The scope of this chapter is to compare the performance offered by each architecture and finally to select the one giving the overall best results. All architectures have been designed and trained using Keras [15] (version 2.2.4), a renowned package implemented in Python (version 3.7) and used for Machine Learning purposes. Note that Dr. Quentin Rendu[3] had developed preliminary versions of the MLP and WaveNet architectures that were then extensively modified and tuned in the work performed by the author of this thesis, while the LSTM code was entirely designed and tuned by the author of this thesis.

[3] Research Associate, Imperial College London, London, SW7 2AZ, UK

The outputs from the Python codes were a mixture of text files and figures, which were then modified in Matlab where needed.

2.1. Common hyperparameters

Among the hyperparameters pertaining to all the architectures, the most important are:

- Optimizer
- Learning rate
- Epochs
- Batch size
- Metrics
- Choice of the model
- Number of copies

2.1.1 Optimizer

The optimizer algorithm defines how the parameters of the architecture (weights w and biases b) are updated after each iteration. This update step is based on gradient descent method:

$$a_{i+1} = a_i - \alpha \nabla F(x) \tag{10}$$

where:

- a parameter to update (w or b);
- i step index;
- x set of independent variables of function F;
- $F(x)$ function to minimise
- $\nabla F(x)$ gradient of the cost function
- α learning rate

For Machine Learning algorithms eq.(10) can be written as:

$$\begin{gathered} w_{i+1} = w_i - \alpha \frac{\partial J(w,b)}{\partial w} \\ b_{i+1} = b_i - \alpha \frac{\partial J(w,b)}{\partial b} \end{gathered} \tag{11}$$

where the parameters to update are weights w and biases b.

Several types of optimizers have been proposed in the past years for a large variety of different problems but most of them worked well only on the same tasks they have been optimised for. Adam [16] is one of the optimizers that are most used for general purposes because has shown to give very good performance on a large variety of problems, thus it was chosen to train all the architectures tested in this work.

This optimizer embeds two important features: momentum and RMSprop. Both of them aim to speed up the convergence of the algorithm. To understand their benefits we can take a look at the example in Fig. 33 (for the sake of simplicity only two variables are represented, so they can be plotted on a 2D plane; in reality the parameters to optimize can be thousands or millions): we can think of the black ellipses to be iso-values of the cost function and the red dot in the center to be its minimum. The difference between Adam (implementing momentum and RMSprop) and gradient descent methods can be seen as a physical problem where Adam has an inertia term, on the contrary of gradient descent; this implies that Adam is able to smooth its updating direction at every step taking into account the "history" of the previous steps.

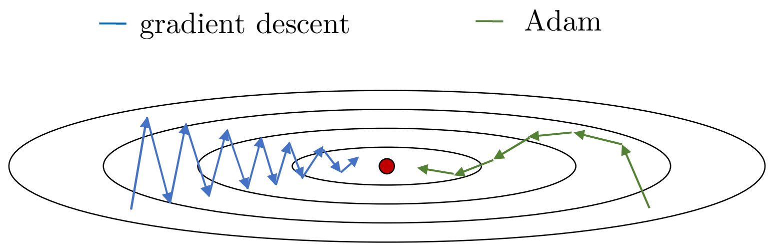


Fig. 33: visual intuition of the difference between gradient descent and Adam algorithms

2.1.2 Learning rate

This hyperparameter has been introduced in the previous paragraph in eq. (10). It is indeed strictly related to the optimizer and cannot be seen separately: it is an index defining the speed of learning of the algorithm. It has to be set:

- high enough to allow reaching the minimum of the function in the shortest possible amount of updating steps;
- low enough to do not incur on diverging problems of the cost function.

After some preliminary tests on the architectures studied in this work the learning rate was set to $\alpha = 1e^{-3}$, which proved to be a satisfactory value for the task of this research. Note that the Adam optimizer is designed to be able to reduce the chosen learning rate as it moves towards the final steps of the learning period. This is done in order to fine-tune the parameters when approaching the minimum of the cost function.

2.1.3 Epochs

One epoch is defined as the step after which the model has been trained on all the examples present in the training set. The number of epochs is set to be high enough to let the algorithm train to decrease the cost function. On the contrary, too many epochs would lead to overfitting, i.e. the algorithm describes very well the relation between inputs and outputs of the training set but is not good in generalising on data that has never seen during training.

2.1.4 Batch size

If the training set contains a very large number of examples, a large number of epochs are needed to update weights and biases of each neuron. A well-known procedure used to speed up the learning process is to update weights and biases several times per epoch: this is done by dividing the training set in smaller groups, each one called mini batch. The final result is that the network updates its parameters only looking at one mini-batch at a time, resulting in no need to wait until the end of the epoch to move towards the minimum of the loss function.

The lower the batch size the faster the learning process but, typically, the lower the accuracy of the update: every batch is formed taking random samples from the training set thus the mini-batch distribution may not be representative of the distribution of the entire training set. This can lead to a non-smooth descent towards the minimum of the loss function (Fig. 34). After some preliminary tests, in this work batch size equal to 64 has been used to train all the neural networks, i.e. the weights and biases are updated after considering only 64 examples per time.

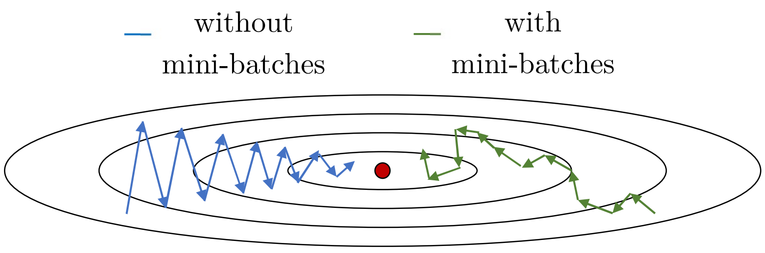


Fig. 34: mini-batch technique influence on loss function minimisation procedure

2.1.5 Metrics

It is fundamental to carefully select the objective functions that can better represent the desired goal of the ML training. When dealing with logistic regression in Machine Learning algorithms there are two main metrics that are taken into account:

- loss: index of the distance between predicted and actual outputs (it is the value of the cost function);
- accuracy: index of the capability of the network to correctly classify each output.

These two metrics are used at different steps of the learning process. The loss is calculated during the training of the model, both for the training and the validation set. Both losses should decrease during the training, hopefully reaching an asymptote after a certain number of epochs. The lower the loss the lower the distance from prediction and actual output values so typically the higher accuracy.
The accuracy is calculated after every epoch but its main use becomes important at the end of the training, when there is the need to objectively verify the performance of the different networks. The result we are interested in is in fact whether the correctness of the prediction, while its proximity to the actual value is of less importance.

2.1.6 Choice of the parameters to be saved from a given training

Typically, there are two possibilities when deciding which set of parameters are to be saved from a given training to:

- save the parameters updated at the last epoch of training;
- set a trigger that will be used to decide at which epoch the model's parameters should be saved.

The former option is easier to implement since it does not require further queries of different trigger types and relative hyperparameters to be set; its drawback stands in the absence of control on the actual performance of the algorithm after every updating step: this simpler model is indeed subjected to overfitting since only the parameters of the last epoch are saved, independently from the history of the whole training process.
The latter can be used to select the best set of parameters based on different events happening during training. For example, one of the most widely-used technique is "early stopping": it monitors the loss of the validation set (called validation loss) and stops the training of the parameters when it has not decreased for a certain number of epochs. The most relevant hyperparameters to set for this technique are "delay" (number of epochs where validation loss does not decrease after which the training is stopped) and "offset" (minimum change of the validation loss to be counted as an improvement, so that the delay count will reset to 0). The usage of this call-back can be effective when the trend of the loss is already known to converge robustly toward low values, which means almost a monotonically descent. "Early stopping" can be misleading when the loss behaviour is not well known: it might happen that the network needs a higher number of epochs than the "delay" imposed to see the validation loss decreasing, assuming that this type of network has bad performance.

A more robust way has been chosen to select the best neural network parameters throughout the whole study. This method monitors the validation loss and saves the parameters only when there is an improvement (i.e. the validation loss has a lower value with respect to the last recorded one), without stopping the training procedure. Therefore the training proceeds till the last step but the set of parameters being saved are those that produced the minimum validation loss. This method can be time consuming (e.g. If the lowest validation loss appears only after 100 epochs out of a total of 1000, 90% of the computation time is wasted since the model did not improve anymore) but it is generally preferred when a new ML algorithm is tested for a given task.

2.1.7 Number of copies

One last important feature that is pertaining to all the architectures is the number of trained copies of the same architecture. This is done to take into account the variability that affects some of the steps of a ML training procedure (such as random weights initialization, shuffling of training set signals, random choice of mini-batches signals), which can produce different performance even when training the same given architecture with the same given hyperparameters. This is ultimately due to due to the high dimensionality of the optimisation problem that ML algorithms usually have to deal with: the cost function in most cases presents several local minima making the localisation of the global minimum a very difficult task. Therefore, any given model of every architecture have been trained for at least five times and only that giving the best performance have been selected.

2.2. Multilayer perceptron (MLP)

This is the most basic and common type of neural network and has thus been used as a reference case. It is mainly constituted by one input layer, one output layer and one or more hidden layers placed in between. Each layer is fully connected, meaning that all the information from a layer are passed to the following one.

The first layer is the input of the network. It is not considered in the count of the number of layers since it is really not computing any operation, it only passes the input values to the following layer, i.e. the first hidden layer. In this work, MLP takes as input a one dimensional vector which is the 2000 time-step-long signal coming from the specific tested sensor.
The operations of eq.(1) are computed in layer 1 (shown in the "Introduction" chapter at page 7 and graphically represented in Fig. 2) for every neuron present in that layer. The outputs of all the neurons of a layer become the inputs for the following layer, which in turn will calculate the outputs for every neuron in it and feed them to next one.
This iterative procedure is done until the final output layer (named L), whose number of neurons is chosen depending on the task that needs to be performed. In this case it was set to one, i.e. a logistic regression.

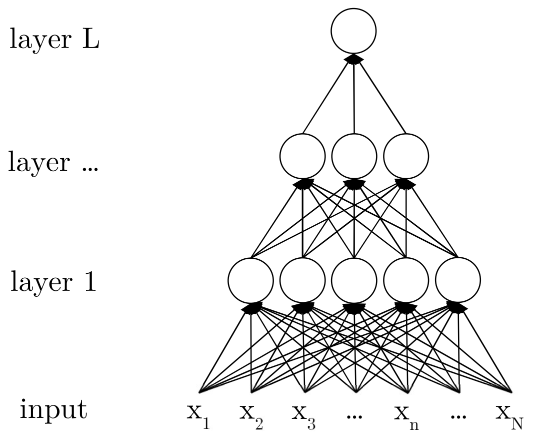


Fig. 35: architecture of MLP Neural Network

The hyperparameters related to the architecture of MLP are:

- number of layers
- number of neurons per each layer

A large combination of these hyperparameters has been tested, and the salient results are given in the next sections.

2.2.1 First set of trainings

As a first trial, the following parameters have been combined for the training of different architectures:

Variable	Values cycled
Number of hidden layers	1, 2, 3, 4
Number of neurons in the first hidden layer	2000, 1500, 1000, 500, 200, 50

Tab. 7: MLP architecture hyperparameters - first set of runs

The number of hidden layers by definition does not account for both input layer and output layer. The number of neurons for the subsequent hidden layers (if any) have been set to be equal to half of the number of neurons in the previous layer (Tab. 8).

Hidden layer (if present)	1	(2)	(3)	(4)
Number of neurons For the 6 different cases	2000	1000	500	250
	1500	750	375	188
	1000	500	250	125
	500	250	125	63
	200	100	50	25
	50	25	13	7

Tab. 8: number of neurons for every hidden layer in MLP architecture

The higher the number of layers and neurons the deeper the network. The number of parameters trained in this set of runs ranges from 100 thousand to more than 3 million. The number of epochs used to train the MLP architectures was set to 3000.

Note that for these preliminary tests only sensor 2 and one SNR 2 have been used, the main scope being to restrict the range of hyperparameters to be searched over. A narrower set of hyperparameters selected from this first trial was then tested with datasets coming from combinations of all the three sensors and all the SNRs selected.

Results

A structured comparison method has been chosen to compare the performance of all the architectures for all the hyperparameters tested. This method is based on accuracy metric. The following figures will present a recurrent scheme: 3 subplots identifying training set on the left, validation set in the middle, test set on the right. Each subplot presents:

- accuracy value (as a percentage) on y axis;
- unique code identifying the set of hyperparameters in x axis. In particular:
  - *nhl* is the number of hidden layers;
  - *nn* is the number of neurons in the first hidden layer;

  - the last three-digits code is an index that can be used to faster compare the accuracies of the same set of hyperparameters (i.e. a combination of *nhl* and *nn*) across the subplots in the same figure (it is not a hyperparameter pertaining to the neural network);
- circles (relating hyperparameters sets to accuracy percentage) of different colours depending on the sensor used, as stated in the legend.

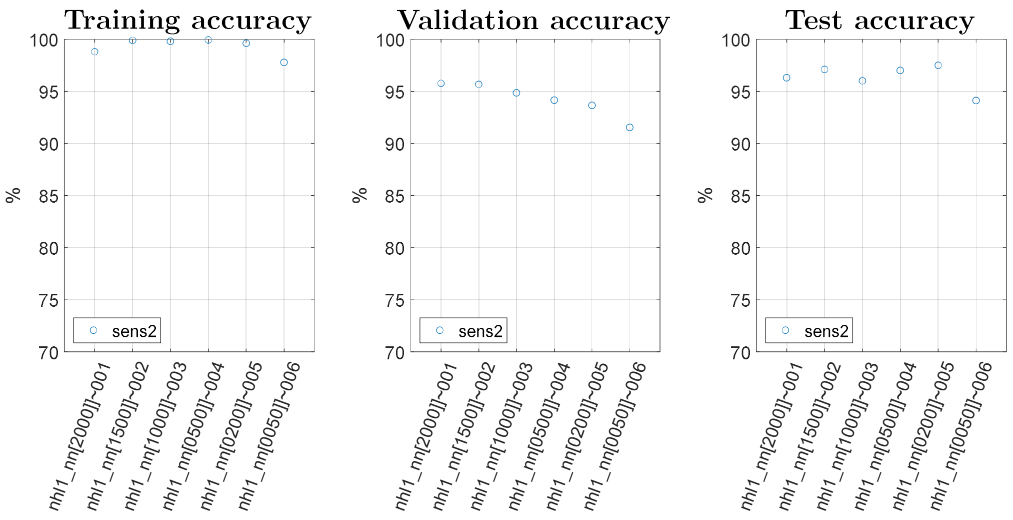


Fig. 36: MLP - accuracy plots: 1 hidden layer

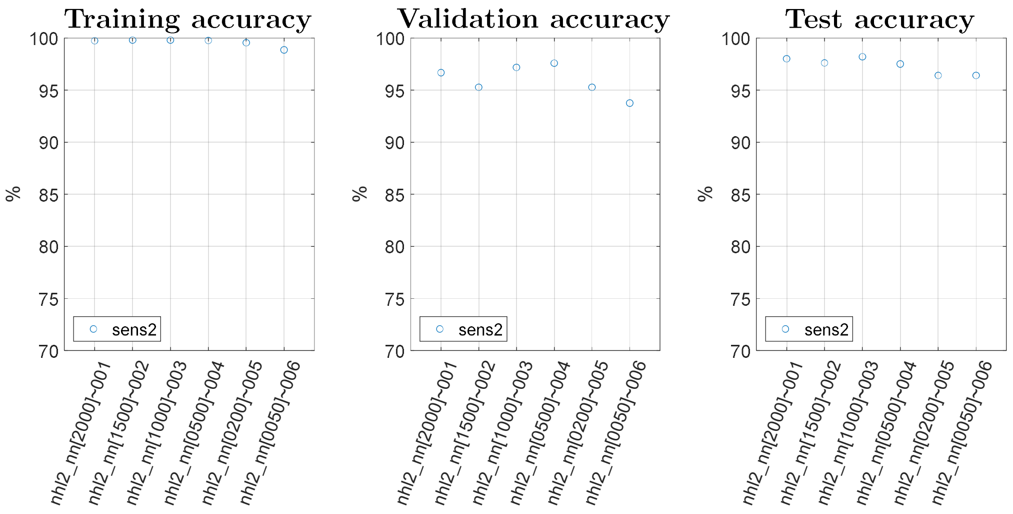


Fig. 37: MLP - accuracy plots: 2 hidden layers

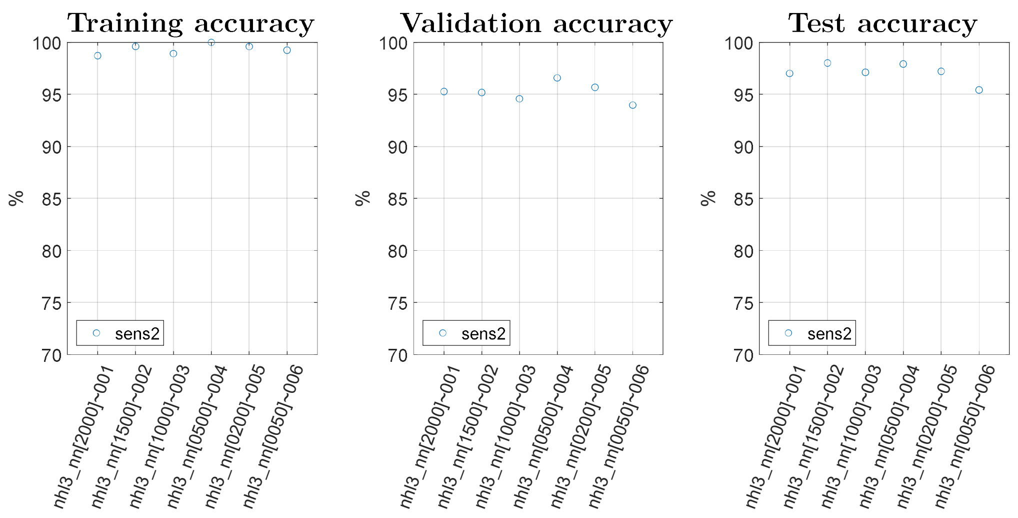


Fig. 38: MLP - accuracy plots: 3 hidden layers

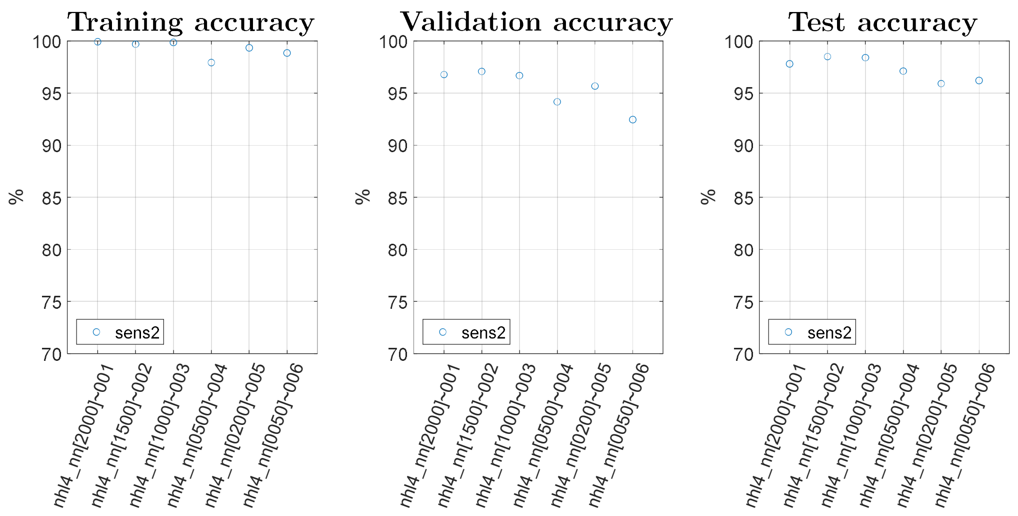


Fig. 39: MLP - accuracy plots: 4 hidden layers

The accuracy of the training set can be useful to understand if the architecture, with a given set of hyperparameters, is able to divide pristine signals from defect signals at least from examples on which the algorithm was trained. Training set accuracies are close to 100%, meaning that the tested MLP architectures are able to extract useful information for defect detection from Lamb wave based signals. The test set accuracy is used to understand the generalisation properties of the ML algorithms, i.e. the ability in correctly classify new examples in pristine or defective class. Test accuracies of Fig. 40 are used to compare the results among different numbers of layers: some improvements are obtained when 2 hidden layers are used rather than 1 but further increase of the number of hidden layers did not produce visible gains. Also, it seems that varying the number of neurons used in each layer did not significantly affect the performance.

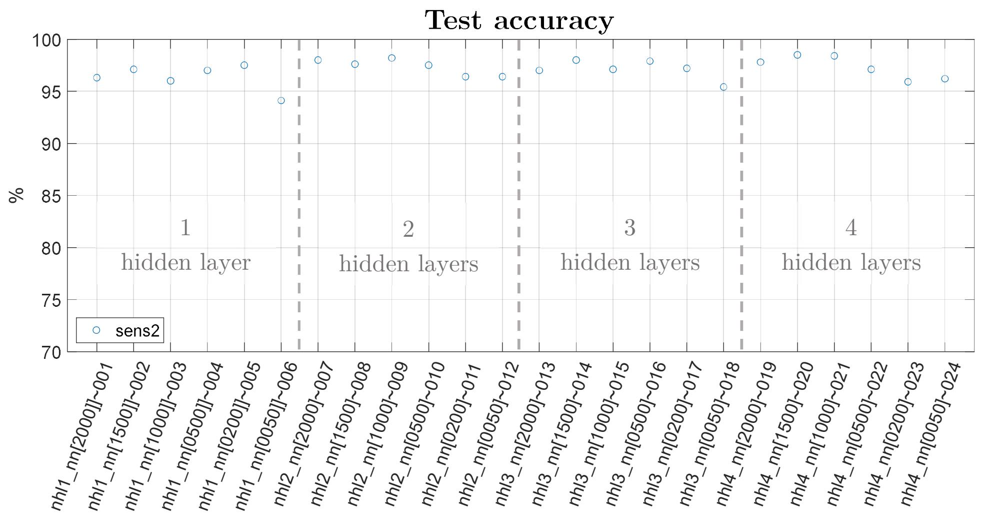


Fig. 40: MLP - test accuracy comparison

These results can be better interpreted by analysing the histories of training and validation loss over the 3000 epochs. Fig. 41 shows one such example, which is representative of a general behaviour seen among these trainings. As seen in figure, the training loss is slowly converging to low values while validation loss remains constant or at most slightly diverges. It can be concluded that this type of algorithm has some difficulties to generalize the results in the training set to the validation (or testing) sets in this specific task of defect detection.

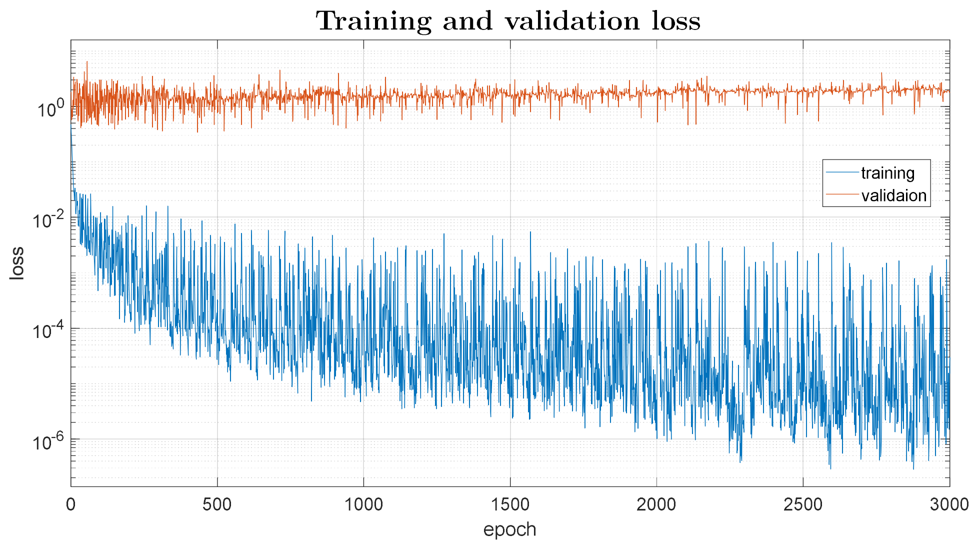


Fig. 41: MLP - loss plot: representative case of the trend of training and validation losses along epochs

To alleviate this effect, two possible solutions are: use a bigger dataset to increase the robustness of the parameters' choice or introduce some regularization methods.
The first is quite common in the ML field (the higher the number of examples the higher the probability that the network well describes the problem) but at a double cost in terms of time needed to obtain these other data and longer time needed to train the

network. The second method can be performed at no extra costs but can still give good improvements. The most common regularisation techniques are investigated in the next section.

2.2.2 Regularisation

Regularisation is a method to increase generalisation capabilities of neural networks. When the accuracies of the training set are higher than those of the validation set it means that the boundaries learnt by the algorithm during learning phase are too sharp. Fig. 42 shows graphically how higher accuracies can be obtained by smoothing the boundaries of the fitting function.

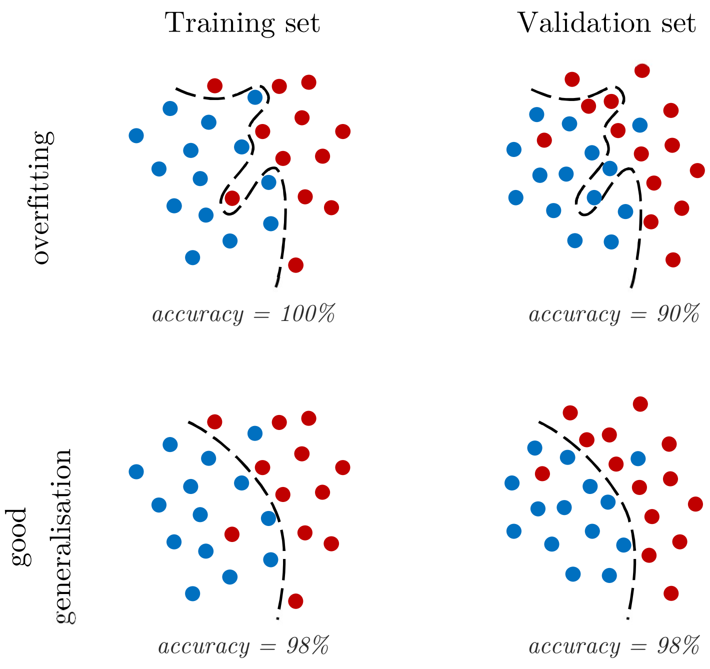


Fig. 42: concept of regularization effect when neural networks tend to overfit over the training set

There are two main regularisation techniques:

- dropout [17] [18];
- L2 [18].

Dropout

A number of neurons in every layer are randomly set to be equal to zero after each update (Fig. 43). For every neuron of the network a random number p is generated such that:

$$0 < p < 1 \tag{12}$$

If p is lower than a predefined threshold $\hat{p}$ the output of the neuron is set to 0. Thus the activation of that neuron becomes:

$$y = \begin{cases} 0 & \text{if } p < \hat{p} \\ g(wx + b) & \text{if } p > \hat{p} \end{cases} \tag{13}$$

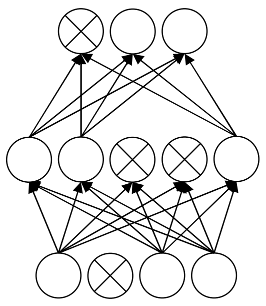


Fig. 43: visual representation of dropout, which randomly sets to zero some neuros

The scope of this technique is to force every neuron to be more independent from those to whom it is connected in the previous layer. This results in reducing the weights of every synapse, forcing the network to create smoother boundaries between pristine signals and defect signals.

The hyperparameters cycled with dropout regulariser are the same as those used in chapter 2.2.1, "First set of trainings", although here dropout regularisation technique was also tested by setting $\hat{p} = 0.2$ and $\hat{p} = 0.5$.

The following results have been obtained:

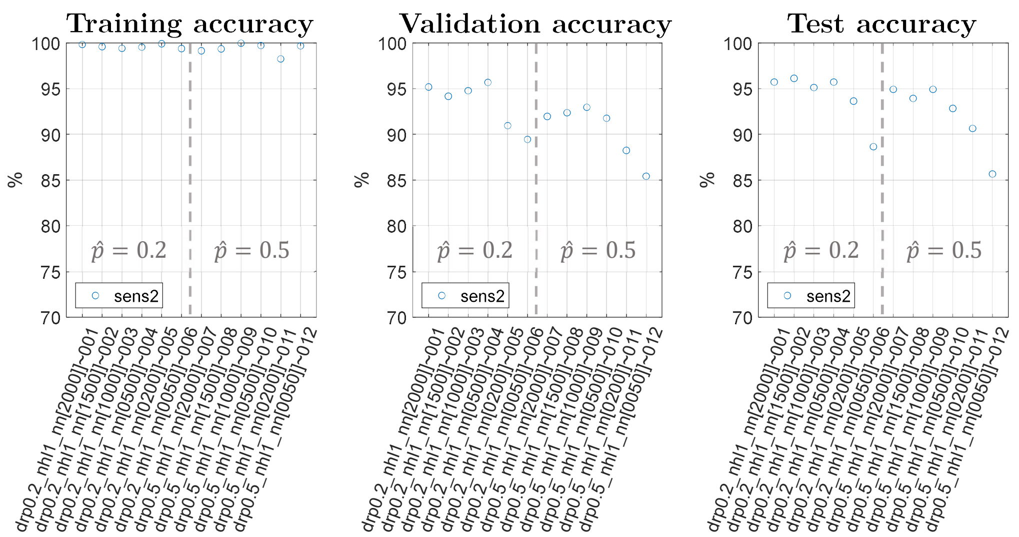


Fig. 44: MLP - accuracy plots: 1 hidden layer with dropout regularisation

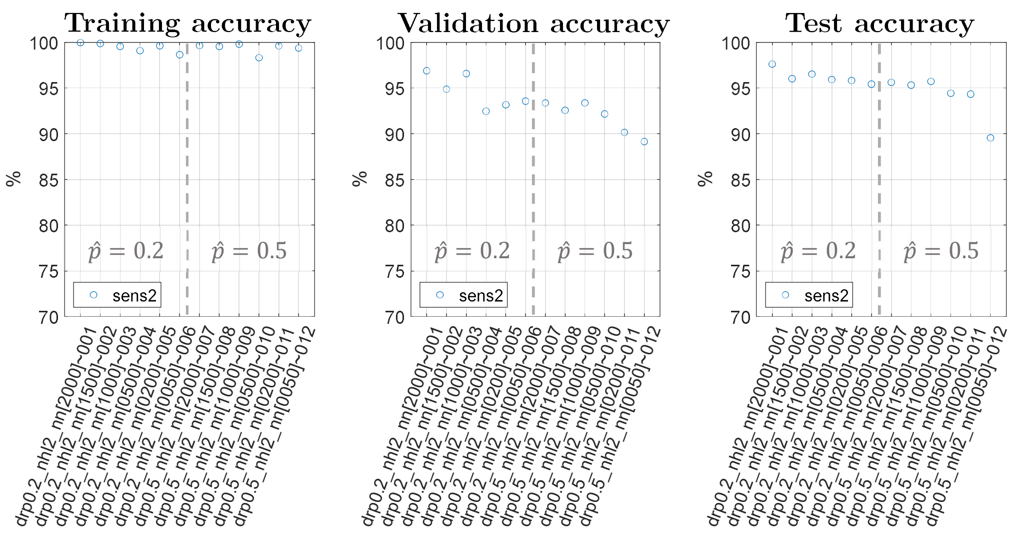


Fig. 45: MLP - accuracy plots: 2 hidden layers with dropout regularisation

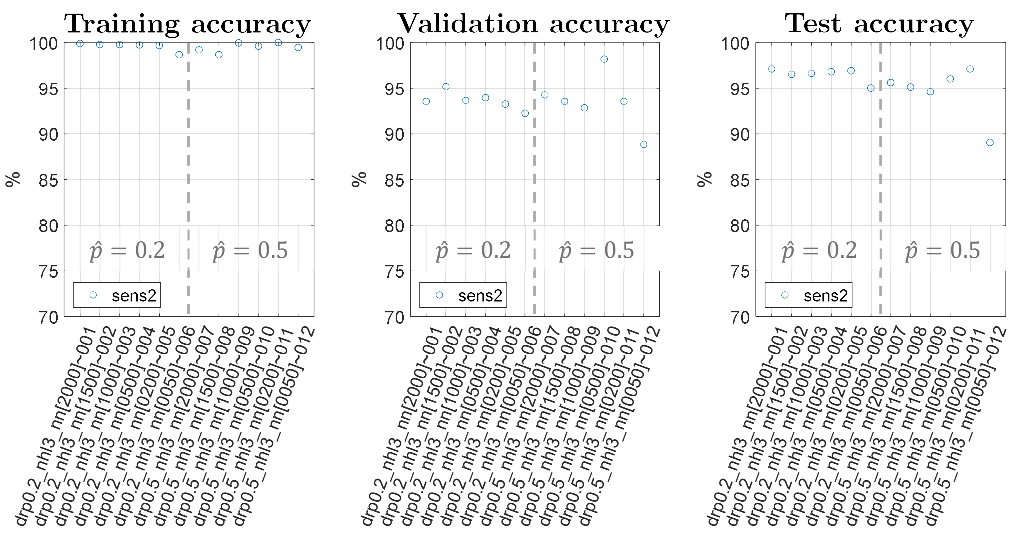


Fig. 46: MLP - accuracy plots: 3 hidden layers with dropout regularisation

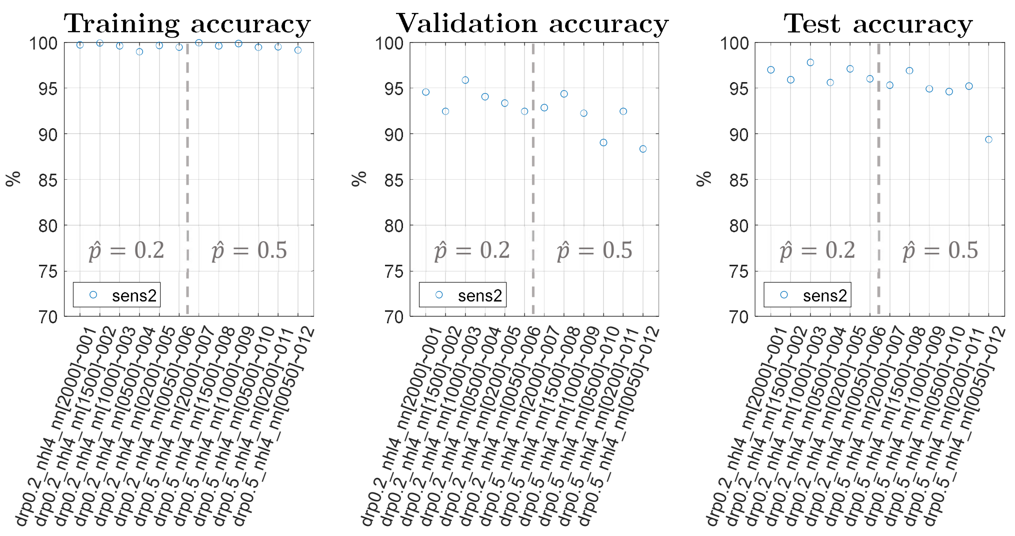


Fig. 47: MLP - accuracy plots: 4 hidden layers with dropout regularisation

The results shown in Fig. 44 to Fig. 47 indicate that even by using dropout there still exists a significant gap between the performance obtained in training and validation/test sets. Also, it is again confirmed that using only one hidden layer yields to worse results than the other cases. Looking at the test sets, drop probability $\hat{p} = 0.2$ generally gives slightly better results so this value was kept for further considerations in the final comparison.

L2

The goal of this type of regulariser is the same as that of dropout, i.e. to keep low the absolute value of the weights of the models. This is obtained by adding the sum of the square of every weight in the network to the cost function, so that eq.(6) (page 10) becomes:

$$J(w,b) = \frac{1}{m}\left[\sum_{i=1}^{m}\mathcal{L}(\hat{y},y) + \frac{\lambda}{2}\sum_{i=1}^{n_{weights}} w_i^2\right] \tag{14}$$

The minimisation of the cost function will lead to the minimisation of both the error and the weights. The hyperparameter to choose for L2 regulariser is λ, which influences the importance of the sum of the weights with respect to the cost function during the training process.

The hyperparameters tested for L2 regulariser are the same as those used in chapter 2.2.1, "First set of trainings", although here the L2 regularisation technique was implemented by setting $\lambda = 1e^{-4}$, $\lambda = 1e^{-5}$ and $\lambda = 1e^{-6}$.

The following results have been obtained using L2 regulariser:

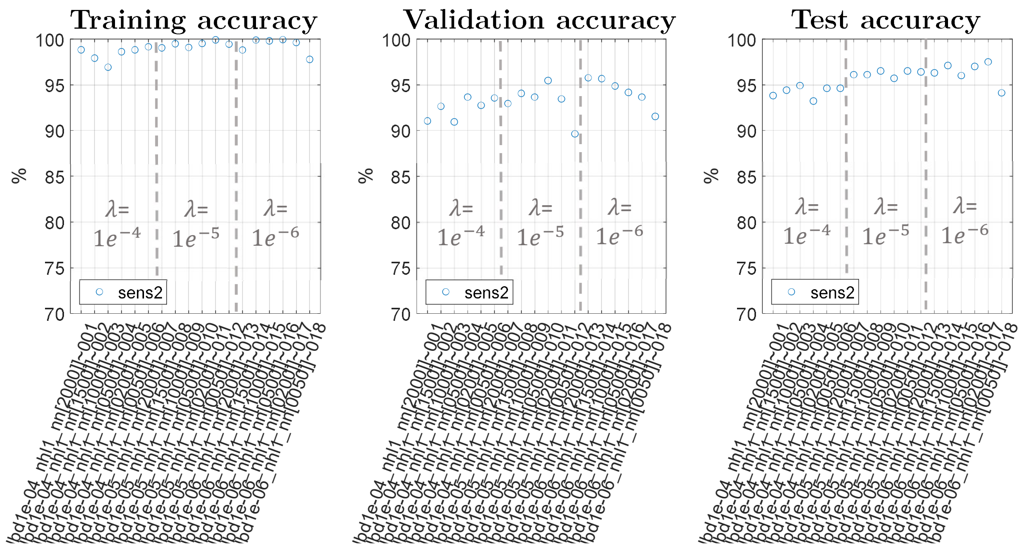


Fig. 48: MLP - accuracy plots: 1 hidden layer with L2 regularisation

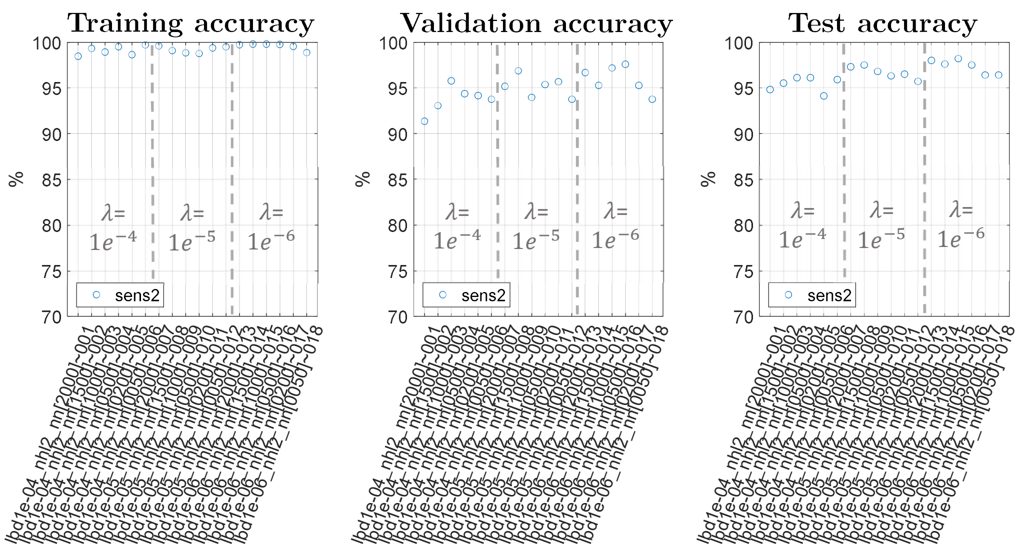


Fig. 49: MLP - accuracy plots: 2 hidden layers with L2 regularisation

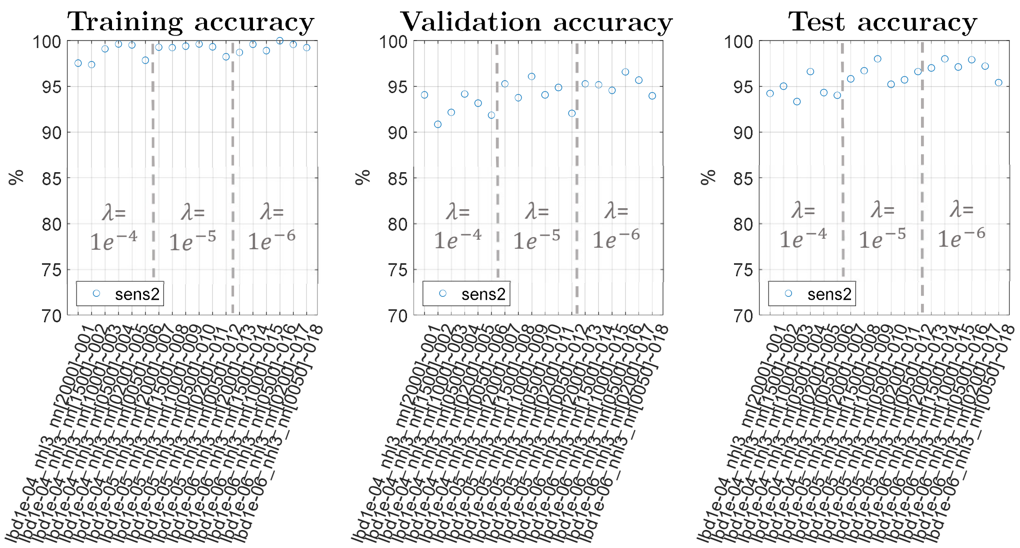


Fig. 50: MLP - accuracy plots: 3 hidden layers with L2 regularisation

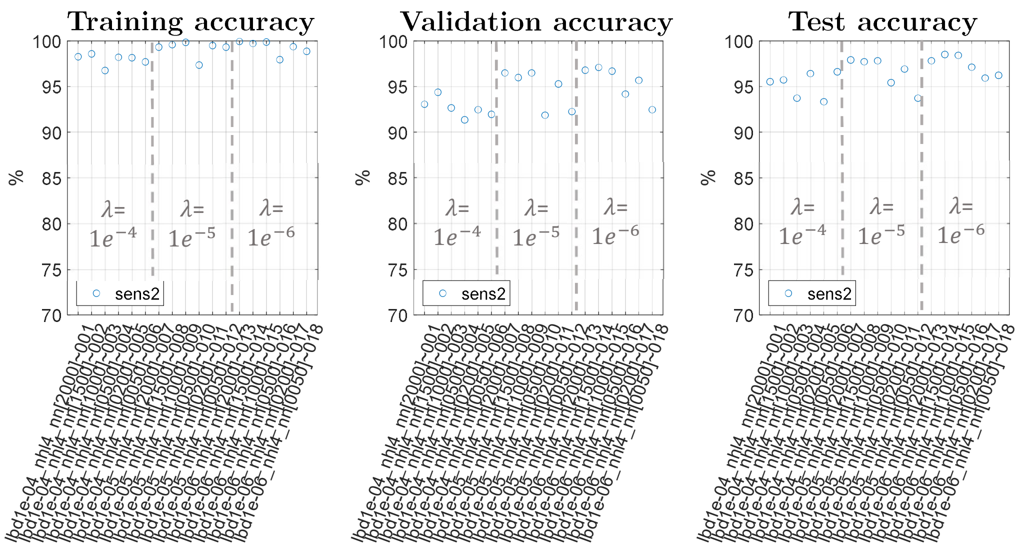


Fig. 51: MLP - accuracy plots: 4 hidden layers with L2 regularisation

The inspection of Fig. 48 to Fig. 51 shows that with $\lambda = 1e^{-4}$ the network was never able to reach 100% accuracy with any set of hyperparameters. It is confirmed also with L2 regulariser that using only one hidden layer yields to worse results than the cases with more hidden layers. Overall, $\lambda = 1e^{-6}$ gives the best results among the tested values, hence it will be used for the comparisons shown in the next section.

Comparison of regularisation techniques

The following plots aid to compare the accuracies of networks with and without regularisers:

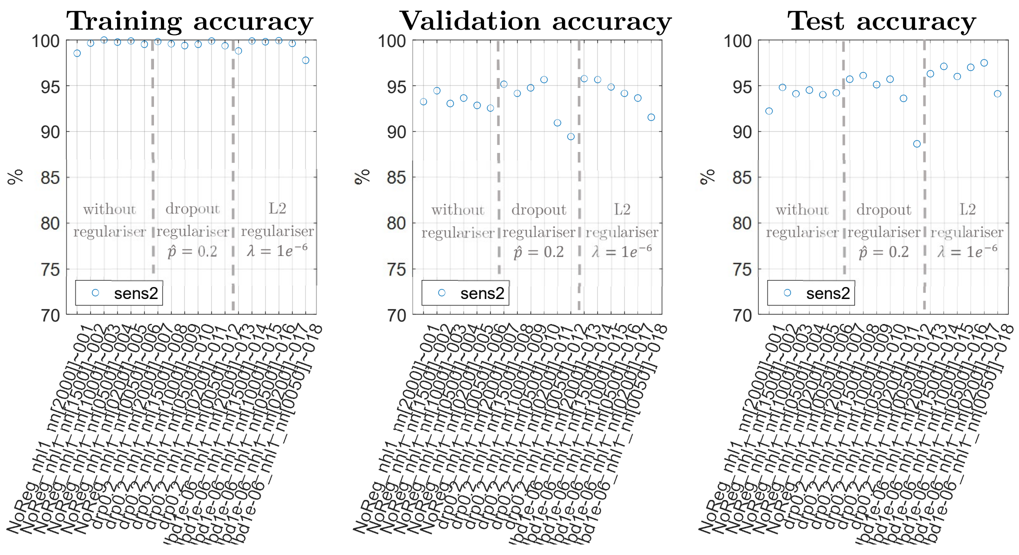


Fig. 52: MLP - accuracy plots: 1 hidden layer, comparison among different regularisers

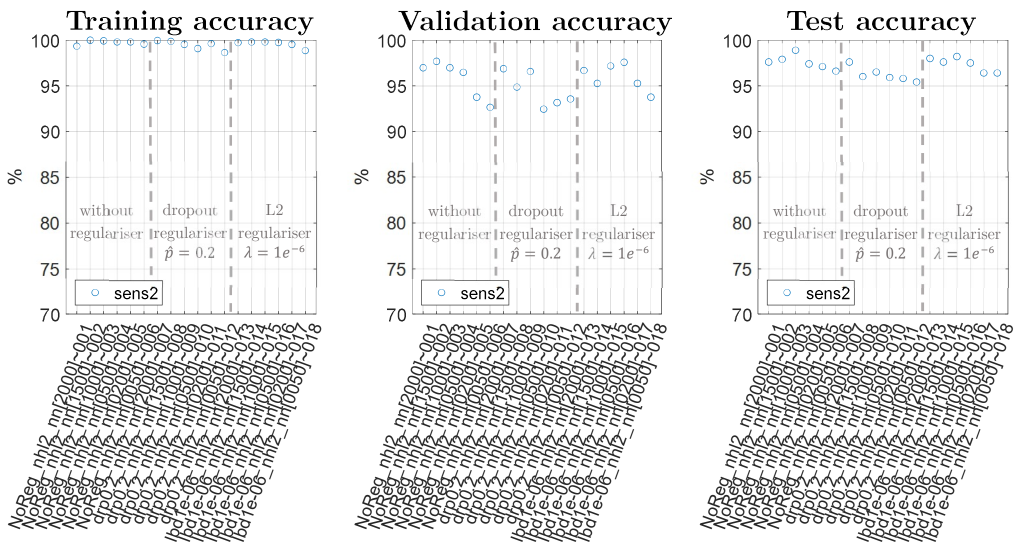


Fig. 53: MLP - accuracy plots: 2 hidden layers, comparison among different regularisers

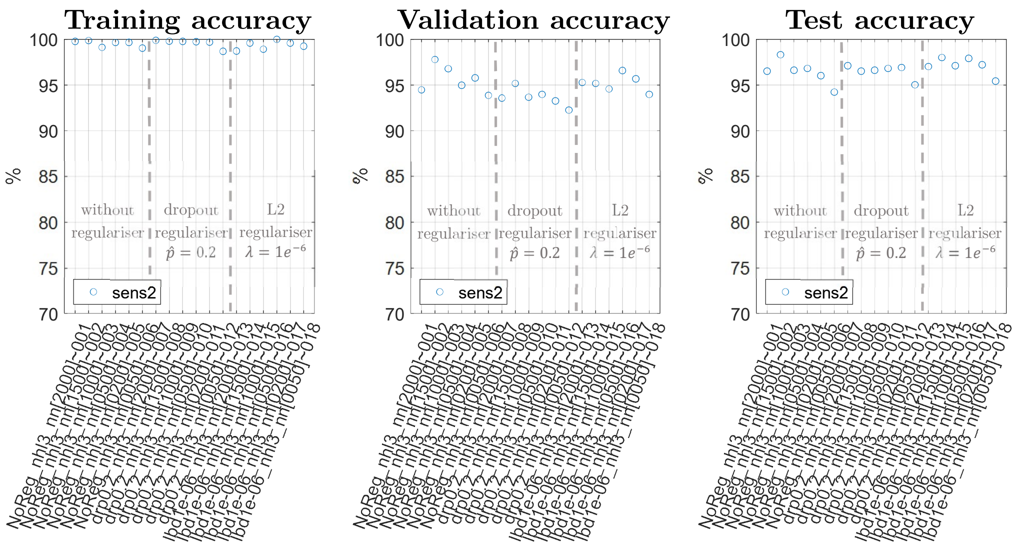


Fig. 54: MLP - accuracy plots: 3 hidden layers, comparison among different regularisers

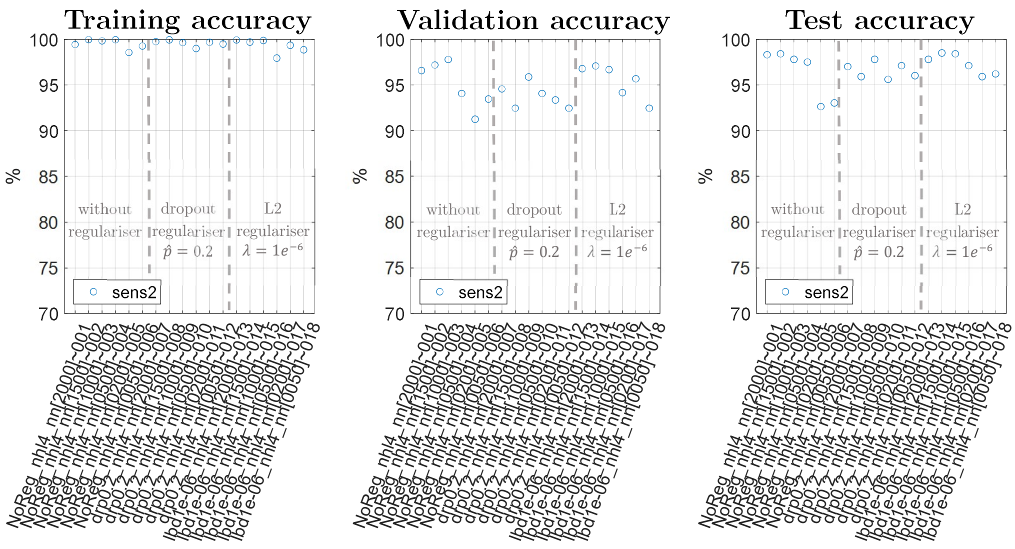


Fig. 55: MLP - accuracy plots: 4 hidden layers, comparison among different regularisers

The plots compares three different learning cases:

- without regulariser;
- with dropout regulariser: $\hat{p} = 0.2$;
- with L2 regulariser: $\lambda = 1e^{-6}$.

It has to be mentioned that dropout is easier to implement than L2: dropout's hyperparameter is a proportionality index pertaining to the relative percentage of neuron activations to drop, while L2's hyperparameter is instead an absolute value which needs a longer iteration procedure to be properly set (in the previous plots only 3 different values of lambda were shown but many have been iterated prior to achieve good results, e.g. $\lambda = [1e^{0}, 1e^{-1}, 1e^{-2}, 1e^{-3}]$).

Training and validation losses during ML trainings have been thoroughly analysed but again only one exemplary case is shown per each regularization technique from Fig. 56 to Fig. 58. Some common features that have been detected are:

- the training loss is continuously decreasing (although typically in a rather noisy pattern) in all three cases and reaches a plateau from around 2000-2500 epochs;
- the validation loss has a slightly divergent trend in the cases without regulariser and with dropout regulariser. With L2 regulariser the validation loss is quite stable and presents some low peaks around 2000 epochs.

The aim of the optimisation of the parameters of the network is to have the lowest possible validation loss. In the best case scenario the validation loss should follow the training loss in the convergence to an asymptote, whose value has to be the lowest possible. In this case none of the regularisers were really able to achieve this condition.

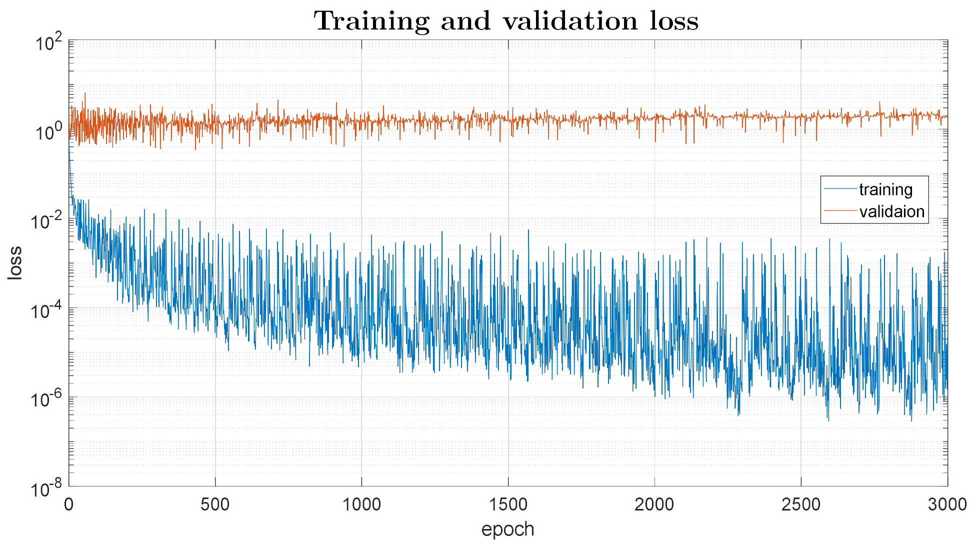


Fig. 56: MLP - loss plots: learning case without any regulariser

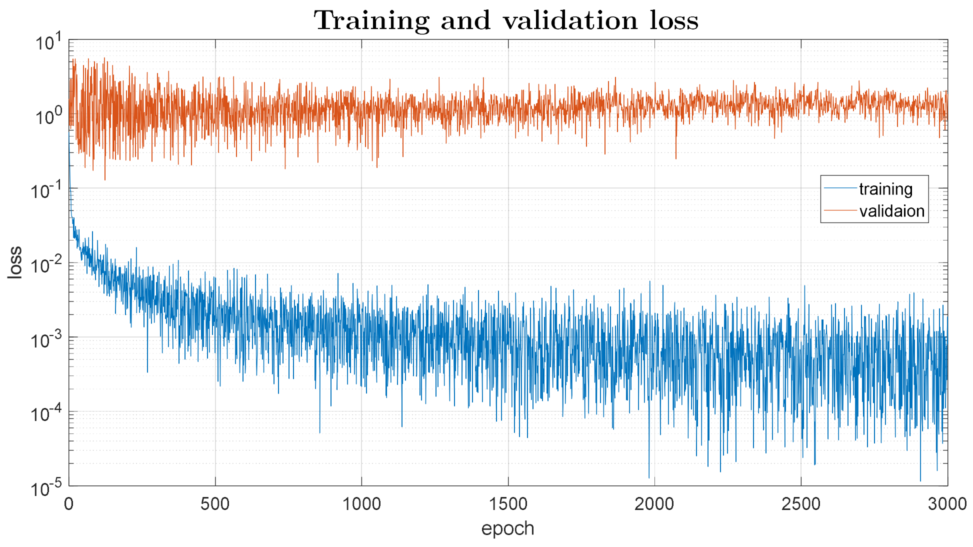


Fig. 57: MLP - loss plots: learning case with dropout regulariser ($\hat{p} = 0.2$)

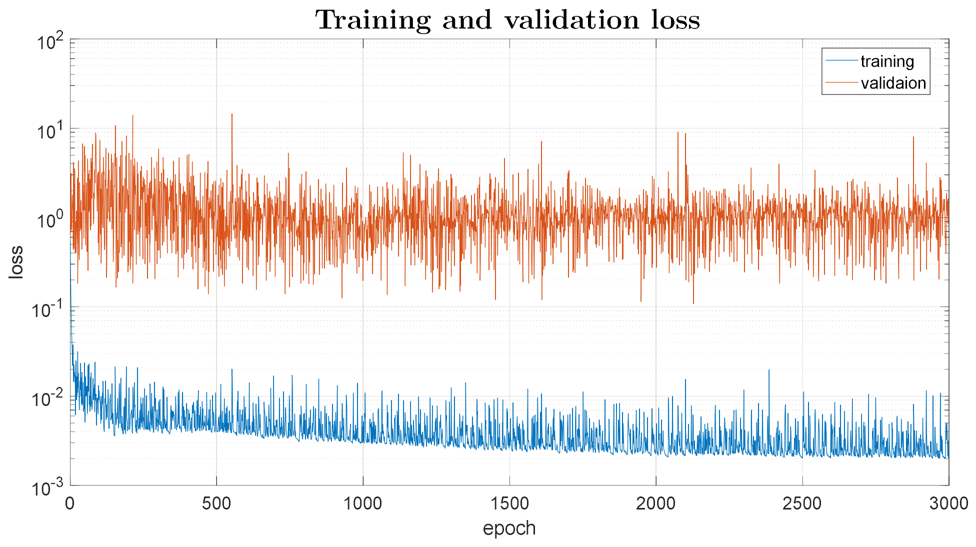


Fig. 58: MLP - loss plots: learning case with L2 regulariser ($\lambda = 1e^{-6}$)

By considering the accuracies of the test sets indicated in Fig. 52 to Fig. 55 and the loss trends shown in Fig. 56 to Fig. 58 it was concluded that L2 regulariser achieves both very high accuracies and a stable/slightly converging trend of the validation loss. Therefore, only L2 regulariser was further tested on the following optimization trials.

2.2.3 Second set of trainings

The MLP architectures described in the previous sections were obtained by assuming a fixed distribution of the number of neurons per hidden layer (i.e. the number neurons in a hidden layer is equal to the half of the number of neurons in the previous hidden layer). In this section a different distribution is considered: the number of neurons in the first hidden layer was set to be the same six values used before (Tab. 8, page 46), the second hidden layer was set to have 1.5 times the number of neurons of the first hidden layer, the third hidden layer for architectures formed by three hidden layers was set to be half the number of neurons of the second hidden layer, while the third and fourth hidden layers for architectures formed by four hidden layers was set to be two thirds and one third the number of neurons in the second hidden layer, respectively. The complete set of tested architectures is summarized in Tab. 9.

	3 hidden layers			4 hidden layers			
	1	2	3	**1**	2	3	4
Number of neurons For the 6 different cases	**2000**	3000	1500	**2000**	3000	2000	1000
	1500	2250	1125	**1500**	2250	1500	750
	1000	1500	750	**1000**	1500	1000	500
	500	750	375	**500**	750	500	250
	200	300	150	**200**	300	200	100
	50	75	38	**50**	75	50	25

Tab. 9: number of neurons for every hidden layer in MLP architecture: second set of trainings

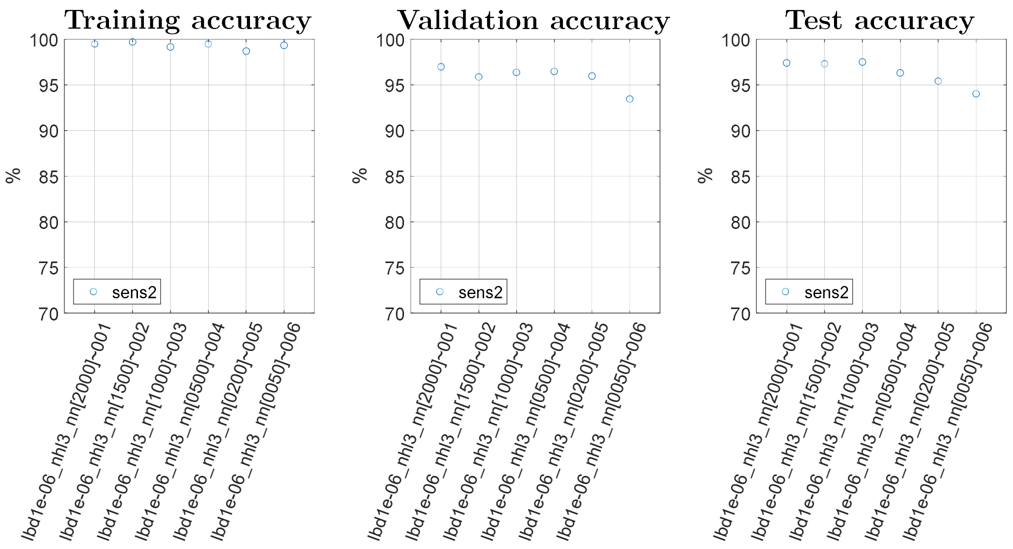


Fig. 59: MLP - accuracy plots: 3 hidden layers, second set of trainings

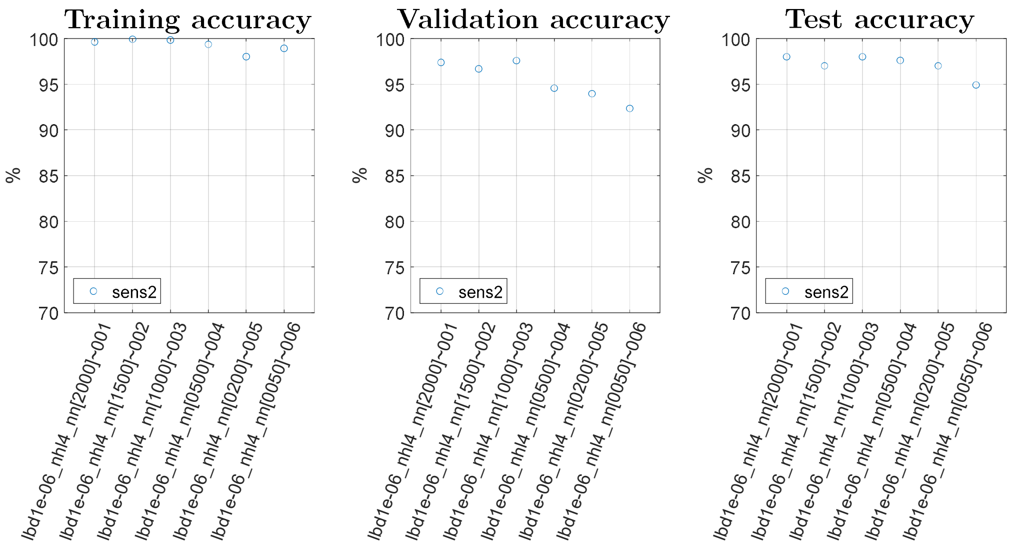


Fig. 60: MLP - accuracy plots: 4 hidden layers, second set of trainings

Fig. 59 and Fig. 60 show the accuracies obtained with this second set of trainings for MLP with three and four hidden layers, respectively. The labels in x-axis recall the hyperparameters used for the algorithms of this section, i.e. the implementation of L2 regulariser ($\lambda=1e^{-6}$) discussed in the previous section, the number of hidden layers (*nhl3* and *nhl4* for three and four hidden layers, respectively) and the number of neurons in the first hidden layer (*nn*), while for the other hidden layers the number of neurons is reported in Tab. 9, as previously explained.
This second set of trainings do not present clear variations in the accuracy of the algorithms with respect to the previous case, where the number of neurons were set to be the half of those in the previous layer. Furthermore, no meaningful differences are present between architectures with three and four hidden layers.

Comparison between first and second sets of trainings

Fig. 61 aids to compare the first (plot on the left) and the second (plot on the right) sets of training, whose only difference consists in the distribution of neurons along the hidden layers, i.e. descending order for the first set of trainings (Tab. 8, page 46) and increasing-descending order for the second set of trainings (Tab. 9, page 58).
No substantial differences can be noticed between the two sets of trainings. In general, the first shows slightly better accuracies, thus it was chosen to be further tested on the following section.

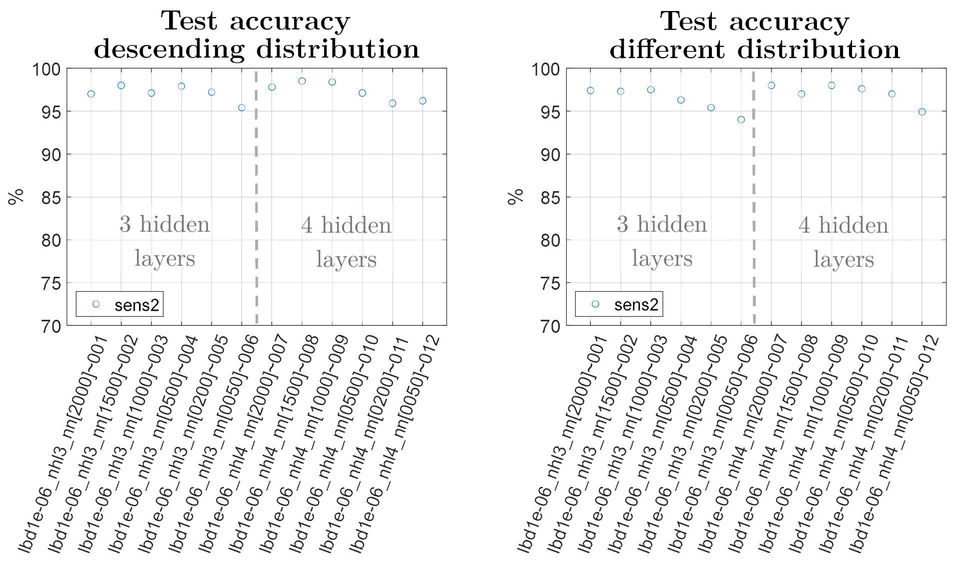


Fig. 61: MLP – test set accuracy plots: 3 and 4 hidden layers, comparison between the cases with neurons distributed in decreasing order (left, first set of training with L2 regulariser) and neurons distributed in increasing-decreasing order (right, second set of training with L2 regulariser)

2.2.4 Best MLP hyperparameters performance – comparison with all sensors and SNRs

A final comparison of the performance given by different MLP architectures was performed by cycling the hyperparameters over the values given in Tab. 10, which were selected based on the preliminary analyses described in the previous sections. As described in previous chapters, the datasets used to train, validate and test the best models come from three different sensors (Fig. 9, page 18) and four different levels of noise were added to the signals.

Fig. 62 to Fig. 65 summarise the results, with each figure plotting the accuracies obtained with a specific number of hidden layers (*nhl* in x-axis labels), from one to four. Each subplot is divided in four portions, each one pertaining to a different SNR level. Every SNR level portion contains six x-axis labels (six identical labels for each one of the four SNR levels, i.e. 24 labels for each subplot) identifying the six different numbers of neurons in the first hidden layer (*nn* in x-axis labels). All the algorithms have been trained with

L2 regulariser with $\lambda = 1e^{-6}$ (*lbd1e-06* in x-axis labels). Tab. 10 summarises all the hyperparameters used for the set of runs in this chapter.

Variable	Values cycled
Number of hidden layers	1, 2, 3, 4
Number of neurons in the first hidden layer	2000, 1500, 1000, 500, 200, 50
Distribution of neurons along layers	descending (as in Tab. 8)
Regulariser	L2 ($\lambda = 1e^{-6}$)
Sensor	1, 2, 3
SNR	1, 2, 3, 4

Tab. 10: MLP architecture hyperparameters - final set of runs

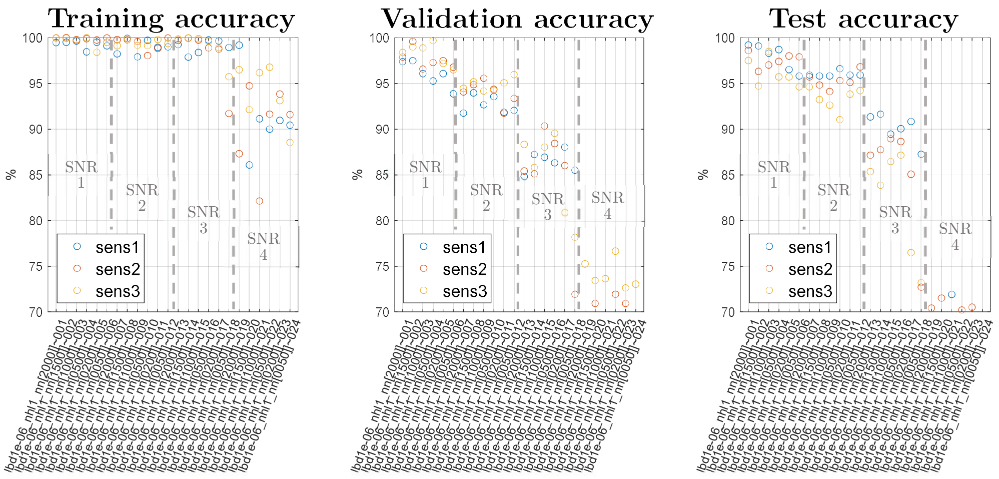


Fig. 62: MLP - accuracy plots: 1 hidden layer, final set of runs

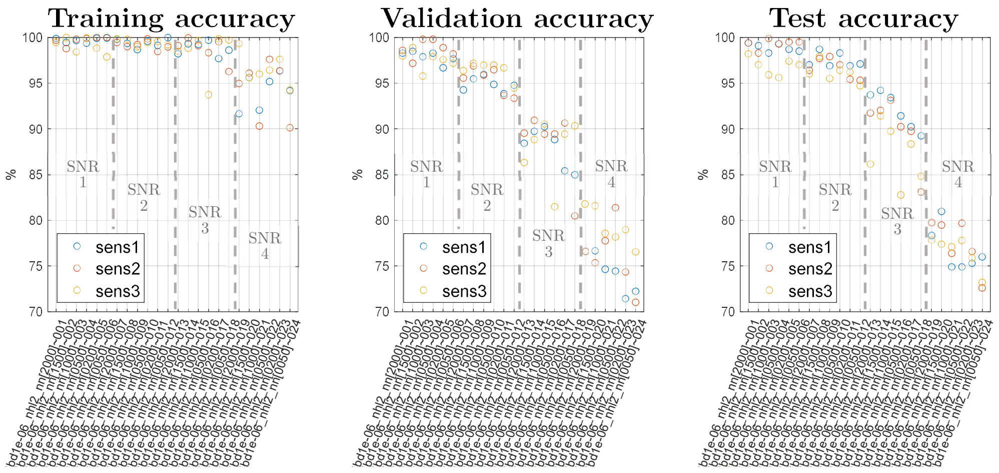


Fig. 63: MLP - accuracy plots: 2 hidden layers, final set of runs

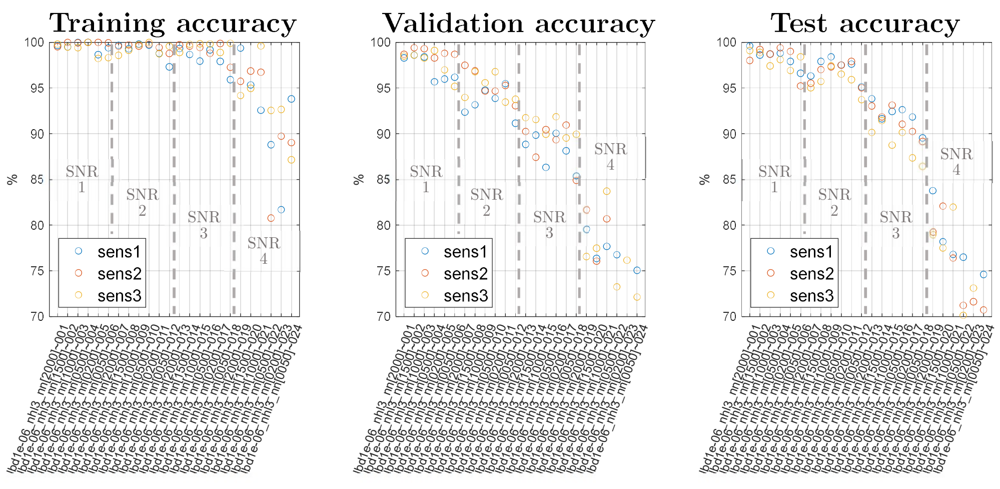


Fig. 64: MLP - accuracy plots: 3 hidden layers, final set of runs

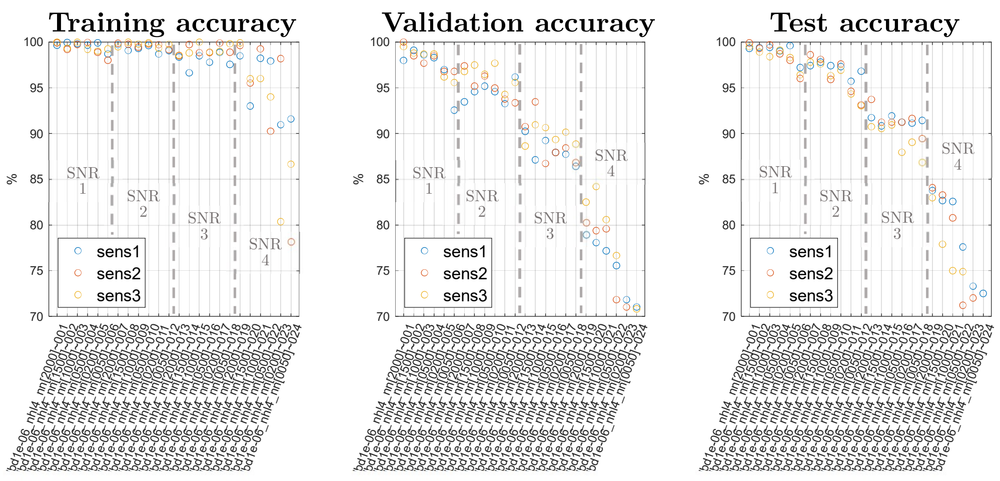


Fig. 65: MLP - accuracy plots: 4 hidden layers, final set of runs

From the plots can be seen that, as expected, the higher the SNR the lower the accuracies. A similar strong trend cannot be seen looking through different hyperparameters, which present a quite noisy and unstable behaviour.

2.2.5 Principal Component Analysis (PCA)

In contrast to the algorithms that will be considered in the next sections, the ordering of the input data does not play any role for MLP architectures (i.e. shuffling the 2000 samples that form an input time-trace would not affect the predictions). Therefore, with MLP it is possible to pre-process the input data in the effort to reduce its dimensionality for example by using Principal Component Analysis (PCA), a normalisation method whose scope is to minimise the variance of a dataset (visual representation in Fig. 66): these data are arranged in a matrix where there are as many rows as the number of examples and as many columns as the number of features characterising each example. Every example is seen as a point in the multidimensional space, where the dimensions are the features. PCA finds a set of new principal axes that are perpendicular to each other and directed such that their distance from all the points is minimised; after that, every point is projected in the new principal component reference system. The transformed dataset has the same dimensions of the original one (examples x features) but its columns (components) are now sorted in order of relevance, i.e. descending eigenvalues. It means that the first component carries more information than the second, which in turn carries more information than the third, and so on. This new arrangement of the dataset allows to consider only the first n components, properly chosen in order to account for a reasonable amount of information needed for the considered problem.

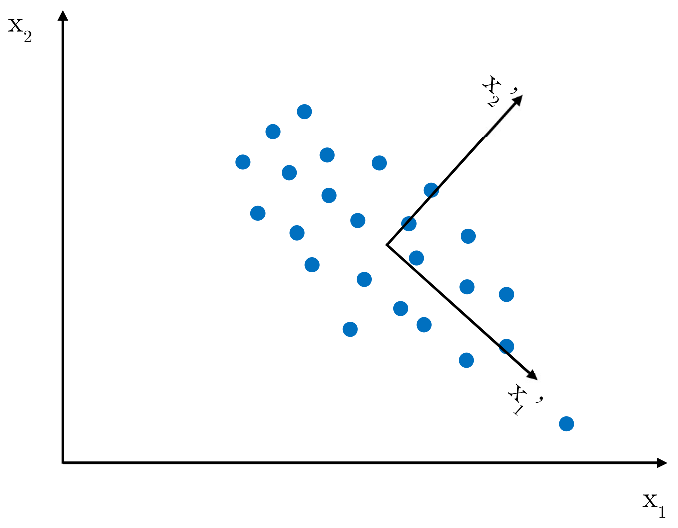


Fig. 66: PCA transformation – x represents the original reference system, x' represents the principal reference system

PCA is performed only on the training set, like it would occur in a practical case. The test set would represent in fact the samples taken from the actual structure and have to be used to assess the structure's actual health status, i.e. they would not be available during the training phase of the ML algorithm. After the principal component reference system was obtained by applying PCA to the training set, all the signals in training, validation and test sets were projected onto the axes of the principal component reference system. The transformed dataset is now expressed in the principal component domain.

The training set has the dimensions reported in Tab. 6, i.e. a matrix with 8000 rows (examples) and 2000 columns (time steps of each signal). From PCA computation the following data are obtained:

- eigenvectors (matrix sized $features$ x $components = 2000$ x 2000): matrix of principal component coefficients, contains the eigenvectors of the principal component space column-wise;
- score (matrix sized $examples$ x $components = 8000$ x 2000): matrix of principal component scores, contains the values of the original dataset referred to the principal components space;

- eigenvalues (vector sized $components = 2000$): vector of principal component variances, index of relative importance among the different principal axes;
- mean (vector sized $components = 2000$): vector of column-wise means of original dataset used to center the original data.

All these data are sorted in descending order of eigenvalues.

From Fig. 67 to Fig. 69, one for each sensor position, is possible to see the cumulative sum of the eigenvalues, pertaining to the percentage of information carried by the first n components. The cumulative sums refer to three different cases: dataset without noise, dataset with the lowest noise (i.e. the highest SNR, SNR 1) and dataset with highest noise (i.e. the lowest SNR, SNR 4). It can be seen for all the sensor positions that the lower the SNR the higher the number of components needed to incorporate a given amount of information about the signals. In the worst case, i.e. SNR 4 (the lowest SNR), the first 200 components carry the 99.7% of the total information of the original signal.

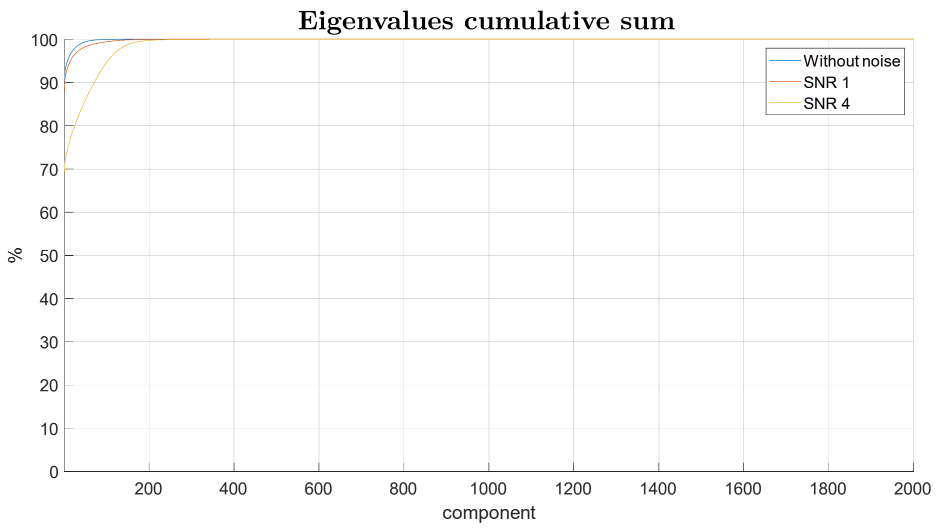


Fig. 67: PCA - cumulative sum of eigenvalues for different SNR levels, sensor 1

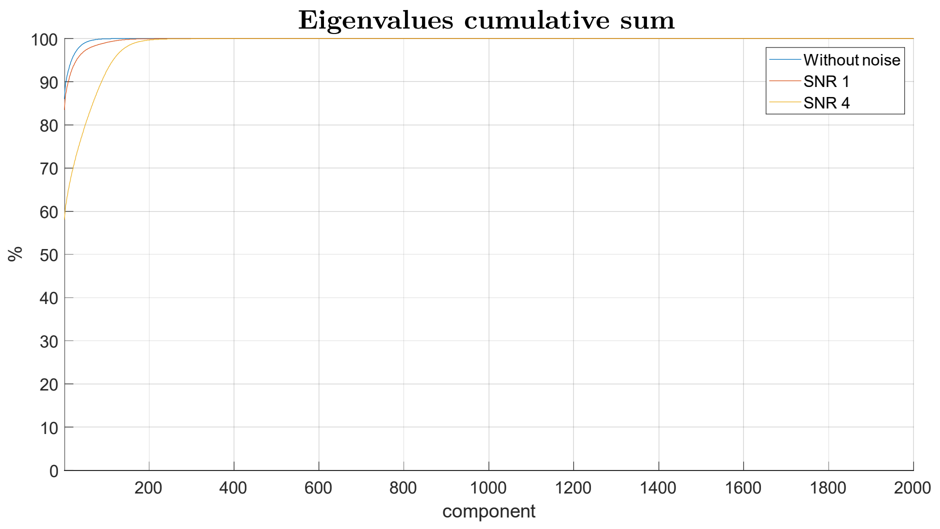


Fig. 68: PCA - cumulative sum of eigenvalues for different SNR levels, sensor 2

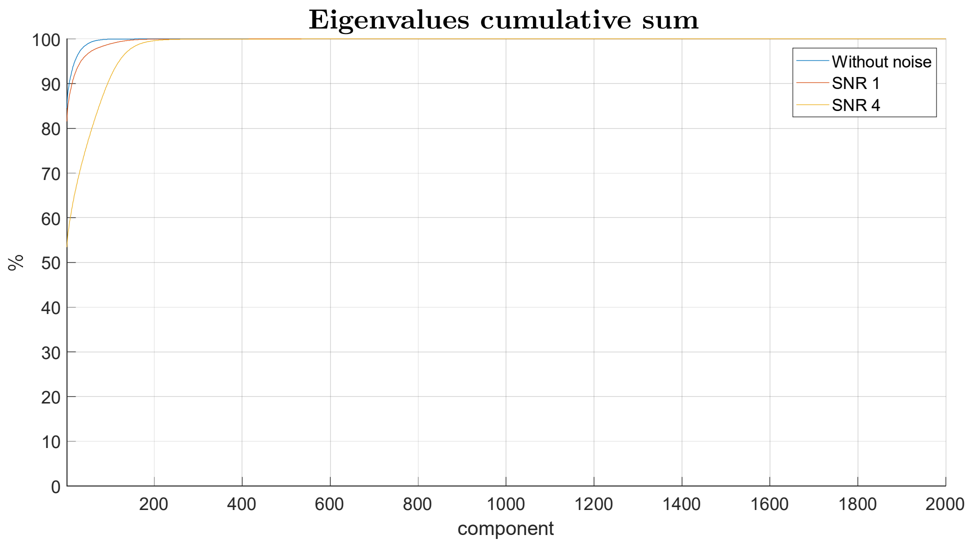


Fig. 69: PCA - cumulative sum of eigenvalues for different SNR levels, sensor 3

Fig. 70 to Fig. 72 show the results of the extreme case where a signal is reconstructed in time domain using only the first principal component. In SNR 4 case (Fig. 72) it can be noticed how this choice might help reducing the noise: from sample 1 up to about 150 the reconstructed signal present a flat trend, similar to that of the noise-less signal (Fig. 70). By considering the entire time-step domain for all three noise-less, SNR 1 and SNR 4 cases, it can be noticed that reconstructing a signal with only the first principal component introduces an error whose amplitude is of the same order of that of a typical damage reflection (recall the two examples of Fig. 19 and Fig. 22, from page 29).

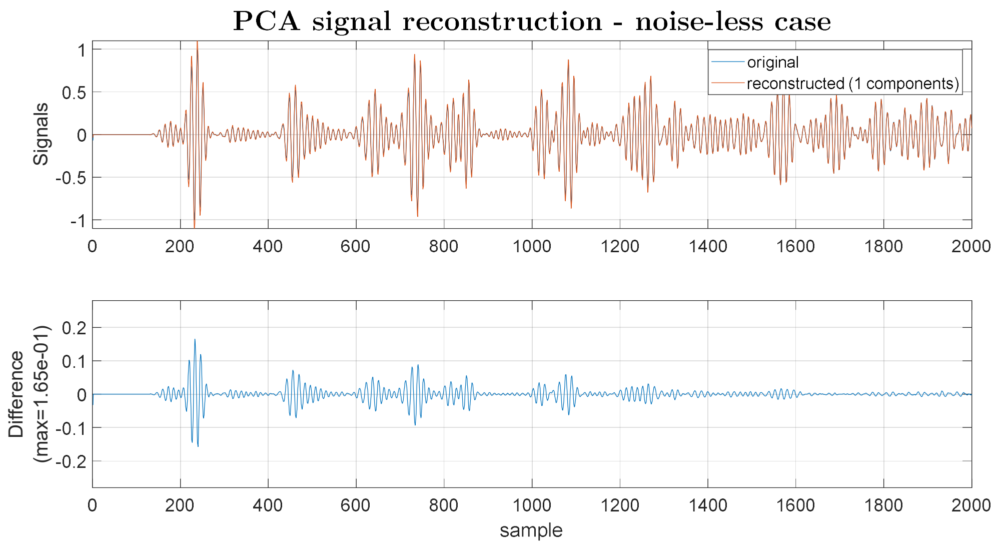


Fig. 70: difference between an example and its reconstruction with only the first principal component – noise-less case

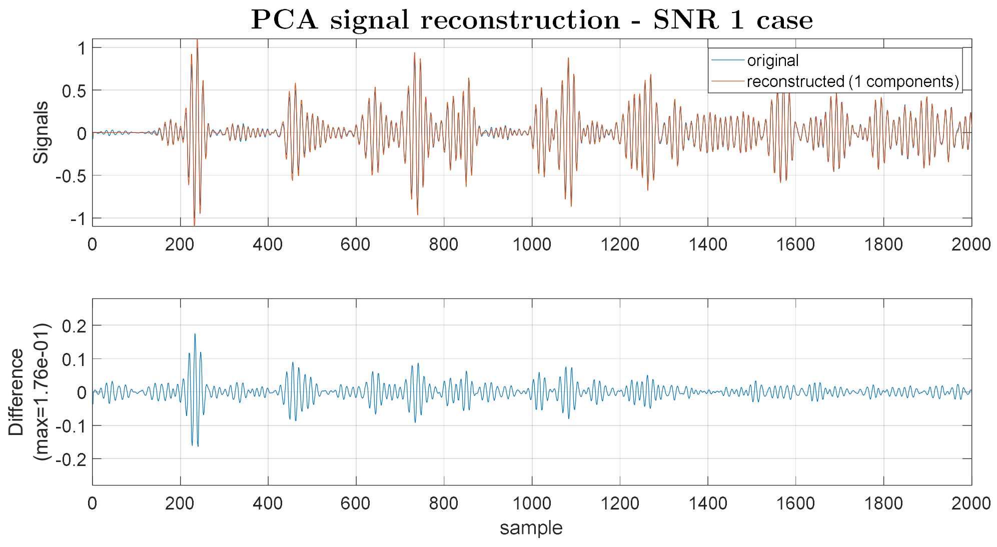


Fig. 71: difference between an example and its reconstruction with only the first principal component – SNR 1 case

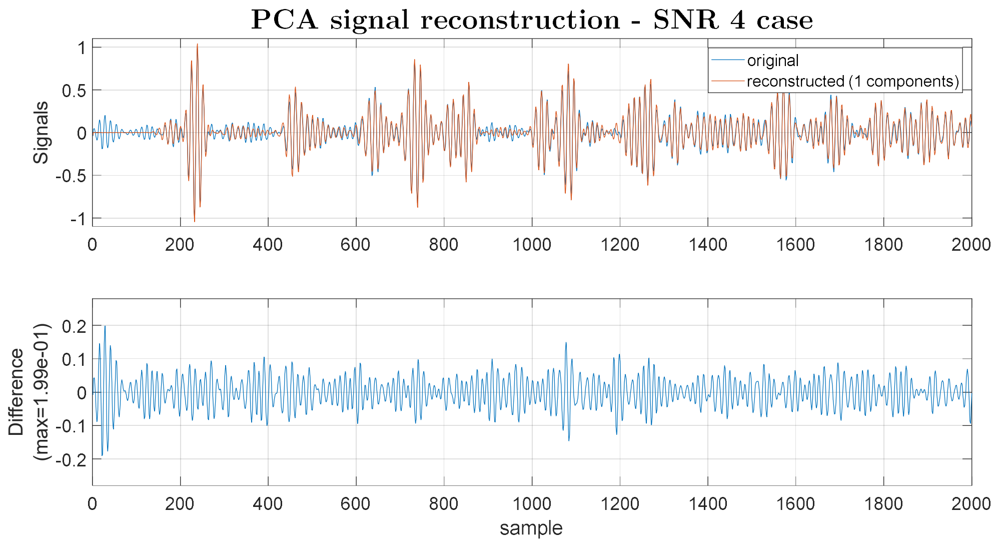


Fig. 72: difference between an example and its reconstruction with only the first principal component – SNR 4 case

The difference shown in the previous figures might be given by both the noise and/or a defect. The number of components to be used to reconstruct the signal could be tuned to reduce only the noise effect on the signals. Nonetheless, the aim of the thesis is to reduce the pre-processing effort and exploit deep neural networks potentiality, thus the number of components was chosen accordingly to their ability in reconstructing each example with at most a negligible error.

Fig. 73 to Fig. 75 show the inaccuracy introduced by reconstructing the signal with the first 200 principal components: the order of magnitude of typical damages was shown to be around 10^{-1} with respect to the normalised signal amplitude (recall the two examples of Fig. 19 and Fig. 22, from page 29), while the reconstructed signal differs from the original one with an order of magnitude of 10^{-2} in the worst case (i.e. SNR 4), hence 10 times lower than the amplitude given by the defect's reflection. This analysis allow to consider as negligible the losses of useful information (i.e. the presence of a defect) introduced by considering only 200 principal components.

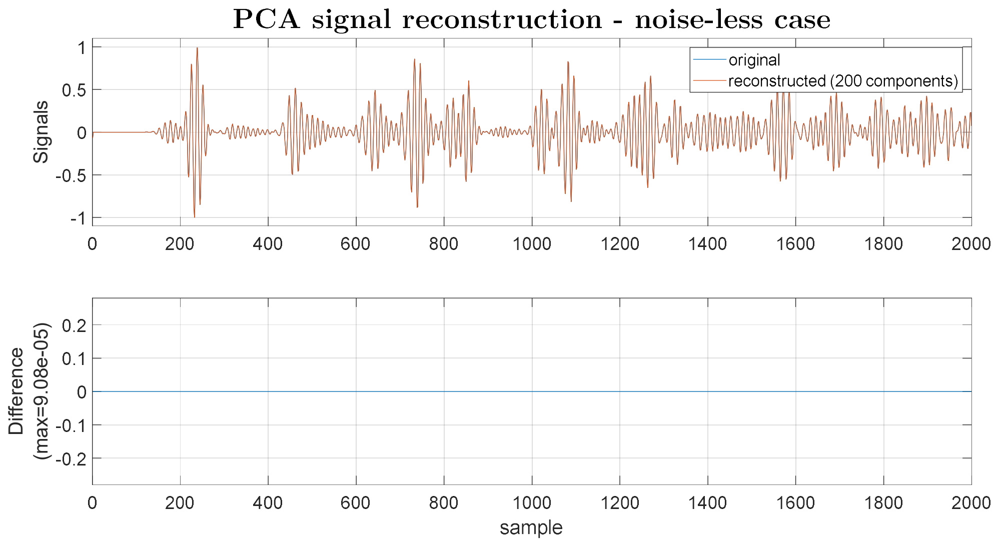


Fig. 73: difference between an example and its reconstruction with the first 200 principal components – noise-less case

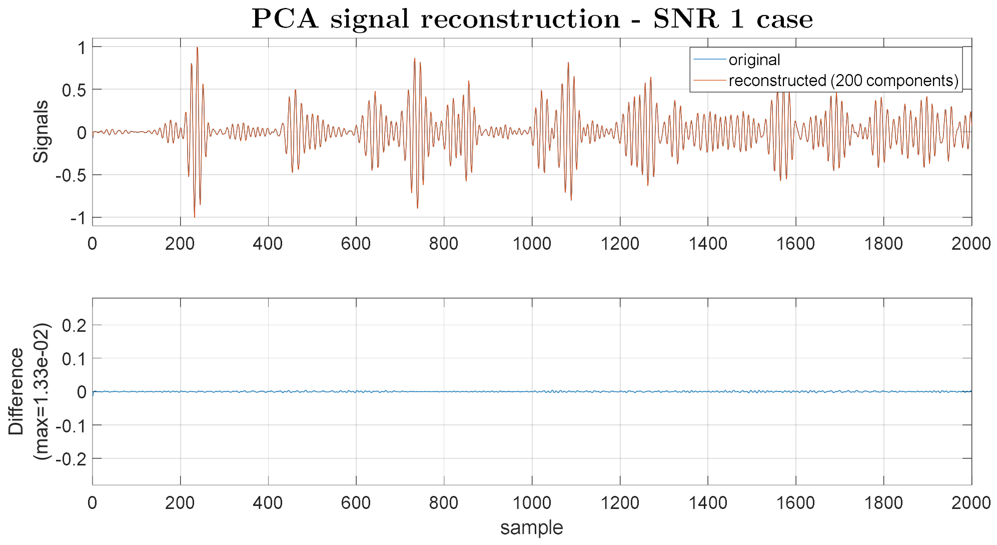


Fig. 74: difference between an example and its reconstruction with the first 200 principal components – SNR 1 case

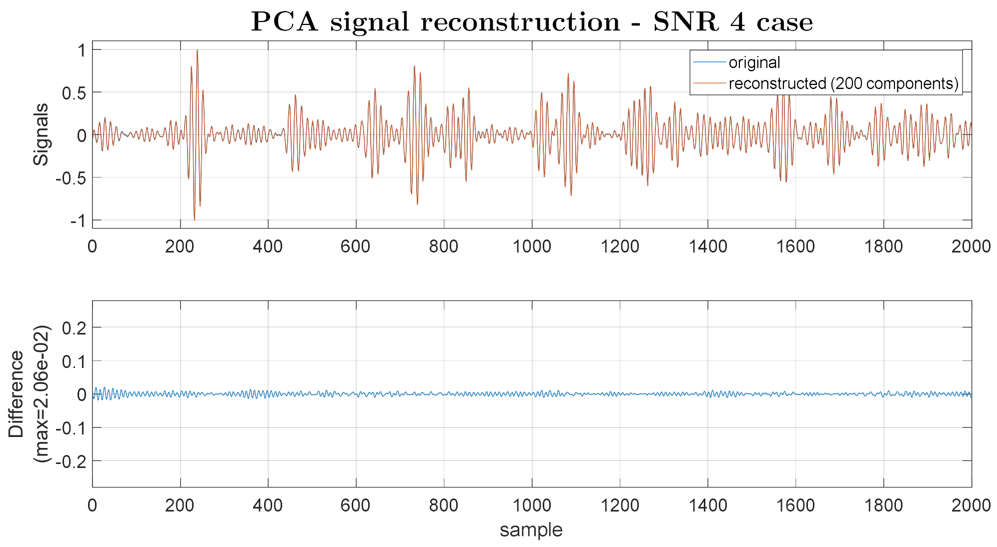


Fig. 75: difference between an example and its reconstruction with the first 200 principal components – SNR 1 case

The first 200 principal components were chosen to project all the signals in the new reference system, hence reducing the dimensionality of each input dataset by one order of magnitude (the length of the signal before PCA was equal to 2000 samples). Fig. 76 shows one of the signals in both time domain (upper plot) and principal component domain (lower plot). It can be seen how by projecting the time-based signal into the new principal coordinate system any temporal (causal) information is lost. As can be expected by looking at Fig. 67 to Fig. 69, the first component already carries most of the information (Fig. 76, bottom plot).

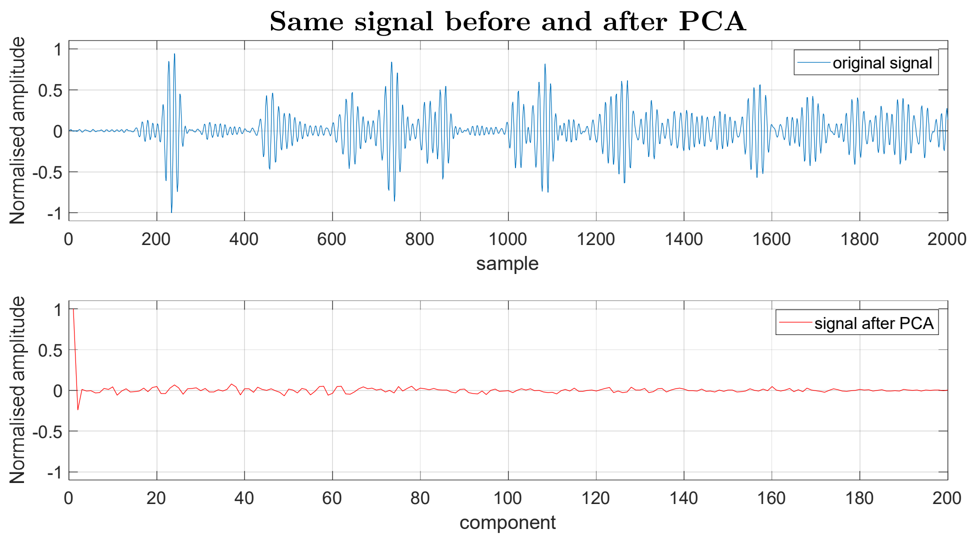


Fig. 76: difference between an original signal and the same after it has been transformed by PCA

MLP architectures trained on PCA datasets have the same hyperparameters as those selected for the previous runs of MLP (Tab. 10), apart from the number of neurons per hidden layer which have been reduced due to the shorter input length of the signals. The actual hyperparameters used are reported in Tab. 11 and the results are shown in Fig. 77 to Fig. 80.

Variable	Values cycled
Number of hidden layers	1, 2, 3, 4
Number of neurons in the first hidden layer	200, 150, 100, 50, 20
Number of neurons in the other hidden layers	Half of the ones in the previous layer
Regulariser	L2 ($\lambda = 1e^{-6}$)
Sensor	1, 2, 3
SNR	1, 2, 3, 4

Tab. 11: MLP architecture hyperparameters - PCA

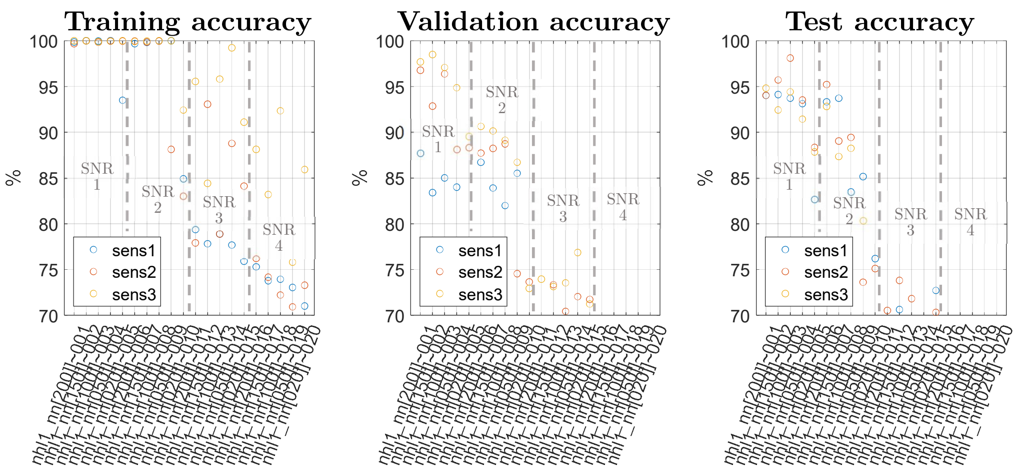


Fig. 77: MLP - accuracy plots: 1 hidden layer, PCA

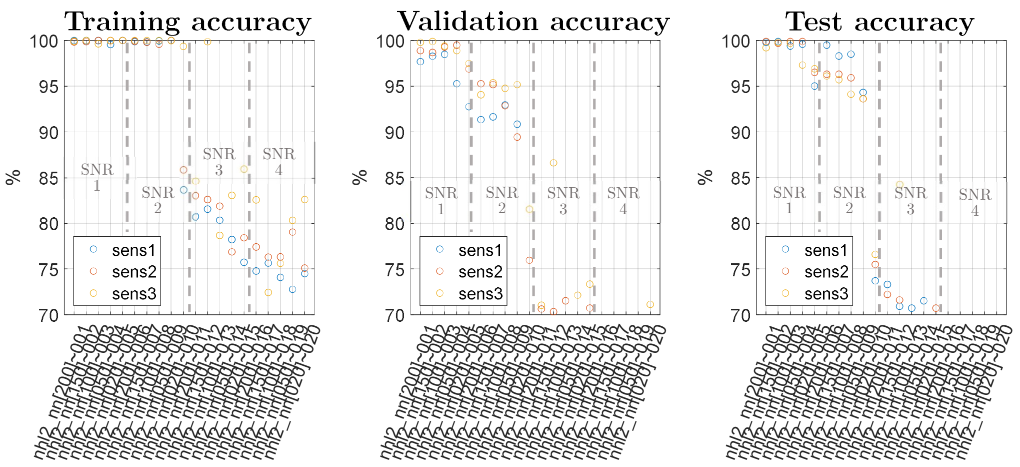


Fig. 78: MLP - accuracy plots: 2 hidden layers, PCA

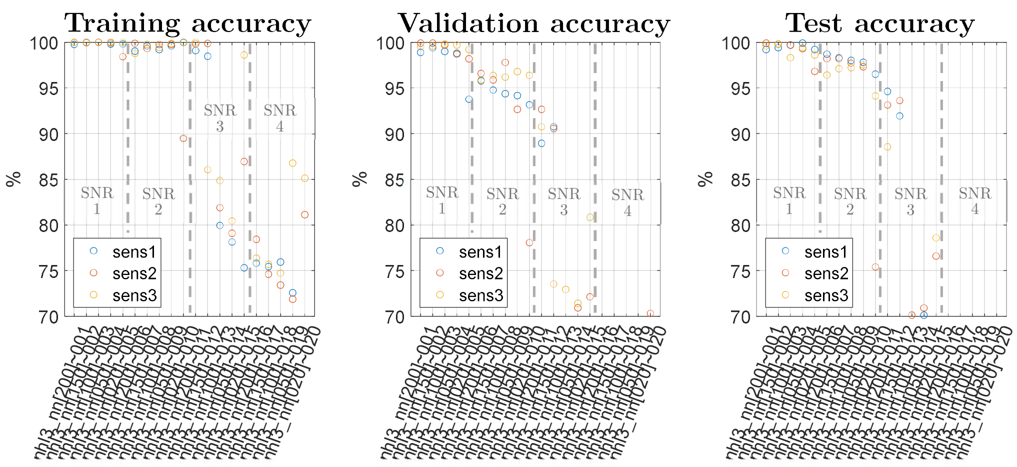


Fig. 79: MLP - accuracy plots: 3 hidden layers, PCA

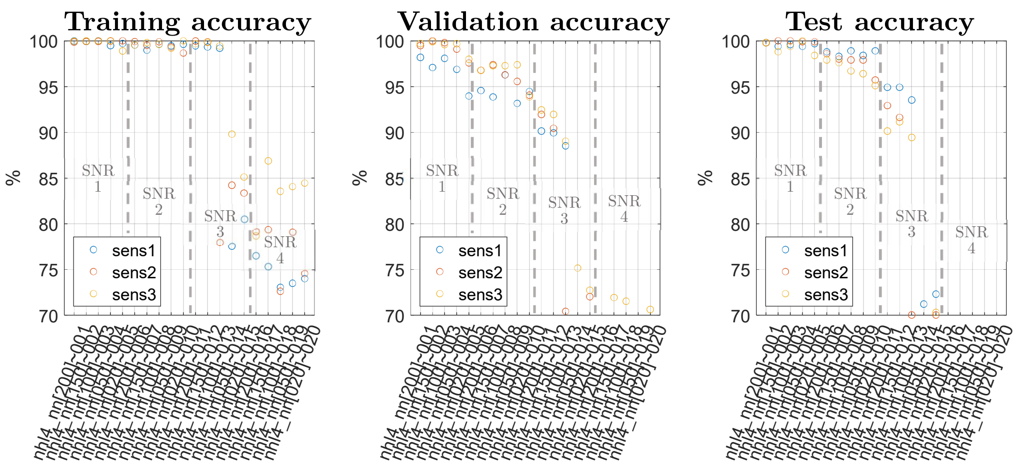


Fig. 80: MLP - accuracy plots: 4 hidden layers, PCA

As before, the accuracies in the test set are used to compare performance of different models. The plots of MLP before (Fig. 62 to Fig. 65) and after (Fig. 77 to Fig. 80) PCA are compared in this paragraph.

When the architecture only contains one hidden layer, MLP with PCA presents clearly worse accuracies with respect to MLP without PCA. For the other numbers of hidden layers there are conflicting trends:

- for the lowest noise (SNR 1), MLP benefits from PCA;
- for SNR 2:
  - for 2 and 3 hidden layers there are low differences in the accuracies with and without PCA;
  - for 4 hidden layers MLP benefits from PCA;
- for SNR 3 and 4 there is an overall reduction of the accuracies when looking at MLP with PCA plots.

As shown in the plots, MLP does not always benefit from PCA. Further analysis can be done to find a possible relation between the optimal number of components and the SNR of the signals: it might be that the number of components considered for SNR 3 and 4 is too low, even if the first 200 carry the 99.7% of information for the highest noise level considered (SNR 4) and an even higher percentage for the lower noise levels.
It has been shown how PCA can lead to higher accuracies for MLP architecture in certain cases but still is not so easy to be optimised. PCA allows MLP to reach 100% accuracies in the test set for only few sets of parameters and in any case only for SNR 1.

2.3. Long Short-Term Memory (LSTM)

Recurrent Neural Networks are ML algorithms whose main characteristic is the exploitation of the order (temporal, spatial…) of the input data. Every cell of an RNN computes their own output from:

- a set of values (features) coming from the input data, based on the relative position of the cell with respect to the sequence-based input;
- a set of values coming from the previous cell, carrying useful information from “back in time” processed inputs.

RNNs are known to be very good at processing sequence-based inputs but at a high cost of time needed for training (every cell computes its output also from the output of the previous cell, thus calculations cannot be parallelised to save time) and a really emphasised problem with exploding/vanishing gradients, especially for long sequences. The phenomenon of exploding/vanishing gradients is well known to affect deep neural networks, thus also and in particular RNNs. During back-propagation step gradients are subsequently multiplied going back through layers; if gradients are even slightly smaller than one they will converge towards zero going back to the first layers, making the parameters updating a very slow process for the first cells; if gradients are even slightly bigger than one they will diverge towards infinity going back to the first layers, making the parameters updating an unstable process for the first cells (exploding gradient issue can be mitigated by setting a threshold to which weights and biases are limited).

A step forward against these problems have been found with Long Short-Term Memory (LSTM) cells [8]. Their characteristic is the addition of another memory term which creates “bridges” along the sequence of cells allowing some information to be stored for a longer time and passed to other cells later along the sequence. This feature allows the algorithm to reduce both exploding and vanishing gradient problems.

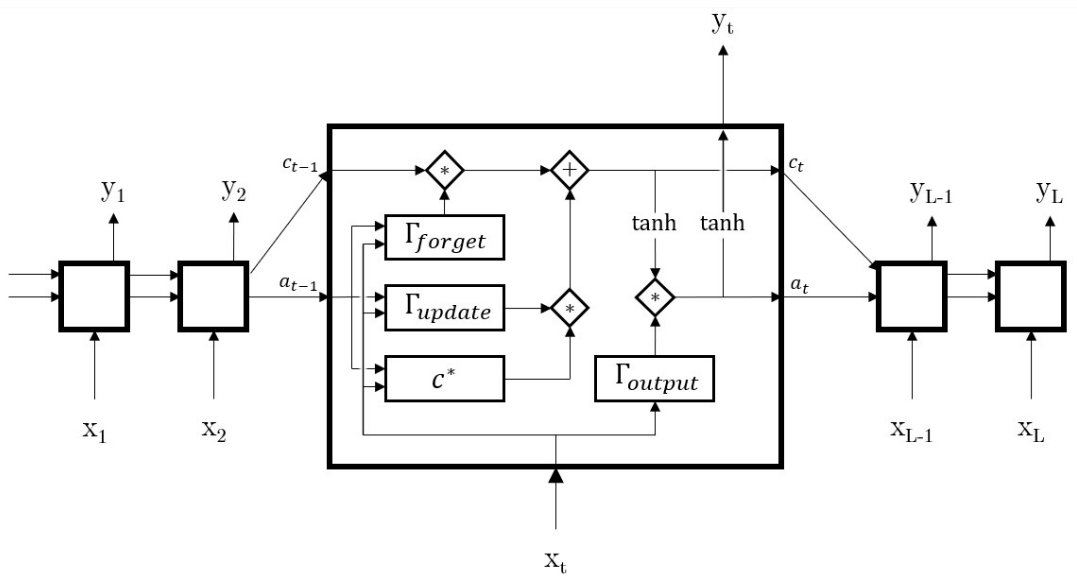


Fig. 81: visual representation of the operations computed inside an LSTM cell: rectangles represent a linear combination of its inputs, rhombus represent an operator and the two tanh are activation functions mapping directly inputs to outputs

The calculations computed by every LSTM cell are reported in Fig. 81 and can be implemented in the following sequence:

The new memory value c_t is computed as follows

$$\Gamma_{forget} = \sigma\left(w_f^T [\, x_t \,, a_{t-1} \,] + b_f\right) \tag{15}$$

$$\Gamma_{update} = \sigma(w_u^T [\, x_t \,, a_{t-1} \,] + b_u) \tag{16}$$

$$c^* = tanh(w_c^T [\, x_t \,, a_{t-1} \,] + b_c) \tag{17}$$

$$\boldsymbol{c_t = c_{t-1} * \Gamma_{forget} + c^* * \Gamma_{update}} \tag{18}$$

The new value a_t is computed as follows

$$\Gamma_{output} = \sigma(w_o^T [\, x_t \,, a_{t-1} \,] + b_o) \tag{19}$$

$$\boldsymbol{a_t = \tanh(c_t) * \Gamma_{output}} \tag{20}$$

The output value y_t of the cell is computed as follows

$$\boldsymbol{y_t = \tanh(a_t)} \tag{21}$$

Where σ is the sigmoid activation function and $tanh$ is the hyperbolic tangent activation function, both reported in eq. (2) (page 9); W_f/b_f are weights/biases of forget gate, W_u/b_u are weights/biases of update gate, W_c/b_c are weights/biases of the new c^* value and W_o/b_o are weights/biases of the output. It is to be noted that weights and biases does not depend on the time-step t considered, they are in fact the same throughout the whole LSTM sequence layer. Due to this consideration it can be stated that the number of parameters to be optimised during training of the LSTM models does not depend on the length of the input sequence.

The number of parameters can vary accordingly to:

- number of LSTM layers;
- number of neurons per LSTM cell, which is the same for every cell belonging to the same layer;

As for MLP, multiple layers can be stack one on the other in order to create a deeper network (usually no more than three layers are used) and each layer can have a different number of neurons.
For LSTM the number of neurons is not connected to the length of the output but to the quantity of features extracted from the input. For example, in this work the input is composed by 2000 time-step long signal and every time-step brings only one feature with it, i.e. the amplitude of the signal; the output of one layer of LSTM can be still a signal with length equal to 2000 but every time-step calculates “number-of-neurons” different features. Seeing it from another point of view it is like extracting “number-of-neurons ” different signals 2000 time-step long, each one carrying different extracted information (the features).

2.3.1 First set of trainings

For this logistic regression problem, architectures with different numbers of layers and different numbers of neurons have been trained. A constrain has been imposed to the last LSTM layer: the number of neurons is set to be equal to one, which means that the output signal will carry out only one feature. This leads to a one-feature output 2000 time-step long which has to be further reduced to one final value, i.e. zero or one. This is done stacking on top of the last LSTM layer a fully-connected layer, reducing the dimensionality from 2000 to 1.

The hyperparameters tried for this set of runs are:

Variable	Values cycled
Number of hidden layers	1, 2
Number of neurons in all but last hidden layer	500, 100, 50, 10, 1
Number of neurons in the last hidden layer	1

Tab. 12: LSTM architecture hyperparameters - first set of runs

The number of parameters trained in this set of runs ranges from 2000 to more than 1 million. The number of epochs is set to 180.

For these runs only sensor 2 and SNR 2 have been used since the main scope at this step is to understand which hyperparameters performs better than the others. After the best are chosen, every model is trained again with only the most promising sets of hyperparameters and with datasets coming from combinations of all the receivers and SNRs.

Results

The architecture with one LSTM layer has been trained with one neuron in that layer due to the constrain previously explained. Its accuracy results are thus being plotted together with the ones of the 2-layers LSTM for the sake of compactness. The next figures are composed in the following way:

- accuracy value (as a percentage) on y axis;
- unique code identifying the set of hyperparameters in x axis. In particular:
  - *nl* is the number of hidden layers;
  - *nn* is the number of neurons in the first hidden layer;
  - the last three-digits code is an index that can be used to faster compare the accuracies of the same set of hyperparameters (i.e. a combination of *nl* and *nn*) across the subplots in the same figure (it is not a hyperparameter pertaining to the neural network);
- different colours of the circles (relating hyperparameters sets to accuracy percentage) depending on the sensor used, as stated in the legend.

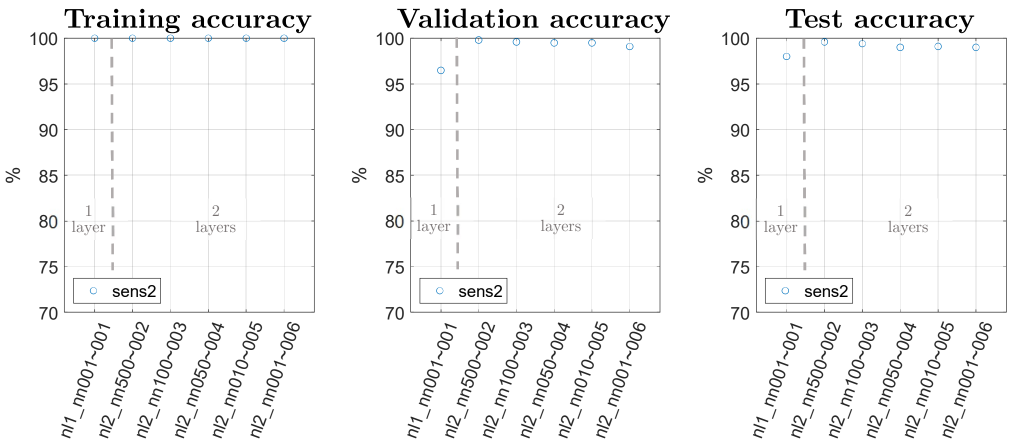


Fig. 82: LSTM - accuracy plots: 1 and 2 layers

All the architectures are able to perfectly define a hypersurface dividing pristine cases from defect ones when looking at the training set. Very good performance show also in validation and training sets where most of the accuracies are close to 100% like in the training one. The architecture with only one LSTM layer has the poorest results when compared to the others looking at validation and test sets. The 2-layers LSTM seems to be low influenced by the number of neuros present in the first hidden layer.

Furthermore, the loss trend in Fig. 83 is way better than the one showed by MLP (from Fig. 56 to Fig. 58): the training loss has a negative gradient even when the maximum number of epochs is reached, the validation loss convergences to an asymptote and both losses have a very low noisy descent (Fig. 83).

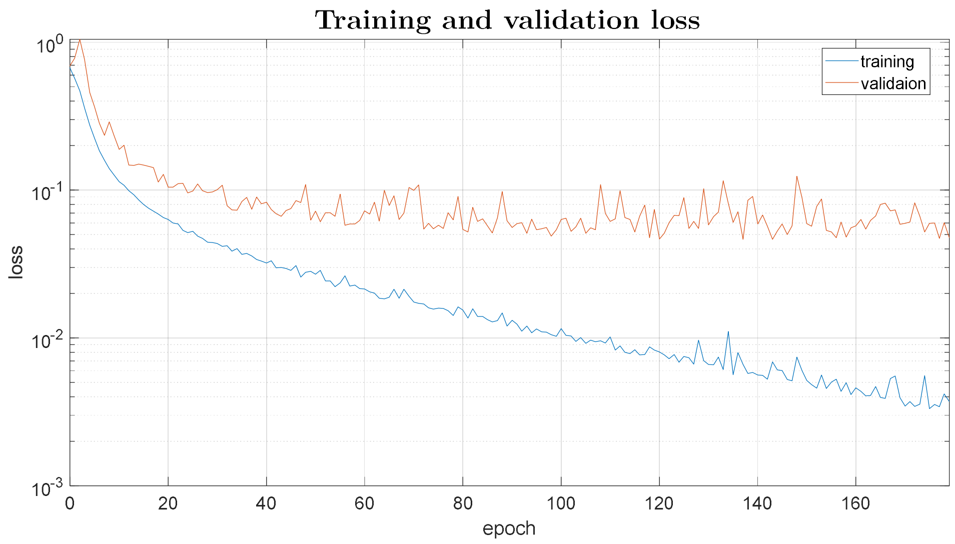


Fig. 83: LSTM - loss plot: representative case of the trend of training and validation losses along epochs

Unfortunately, LSTM architecture has several drawbacks:

- The time needed to train the model with one set of hyperparameters can be as much as 24 hours for the 2-layers LSTM; the training time did not show any dependence on the number of neurons and, even worse, varies quite a lot in a random way (from 18 hours to 24 hours, no matter the number of neurons). Just as a quick comparison, previous MLP architectures training times lasted no more than 2 hours, even for the biggest networks;
- 3-layer LSTMs have been trained as well but the time needed was consequently even more; for the biggest architectures (number of neurons higher than 50) the memory available was not big enough[4] to store some matrices, causing the failure of the training procedure even before it started;
- 2-layers LSTM suffers of exploding gradient from a number of neurons equal to 100. Gradient clipping correction has been implemented but with poor results: the loss function presented *nan* values for certain ranges of epochs during training, leading to non-continuous losses and low accuracies.

Due to these issues, and given the however high accuracies shown, no more hyperparameters have been tested for LSTM architecture.

[4] The training of all the networks have been accomplished using a cluster of 80 RTX6000 GPUs with available memory of 24GB each

2.3.2 Best LSTM hyperparameters performance – comparison with all sensors and SNRs

Also for LSTM architecture a final set of runs with all the sensors (Fig. 9, page 18) and all the SNRs is made.

A summary of the final comparison tests for LSTM is reported in Tab. 13 while the relative plots are in Fig. 84.

Variable	Values cycled
Number of hidden layers	2
Number of neurons in all but last hidden layer	500, 100, 50, 10, 1
Sensor	1, 2, 3
SNR	1, 2, 3, 4

Tab. 13: LSTM architecture hyperparameters - final set of runs

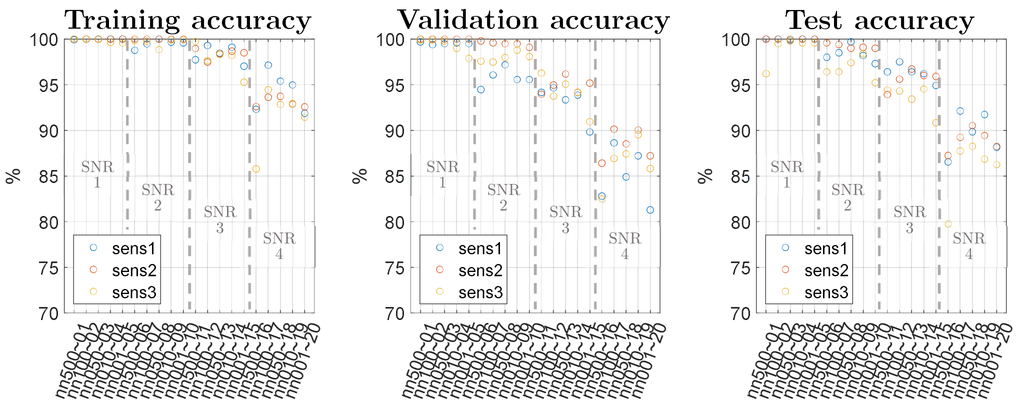


Fig. 84: LSTM - accuracy plots: final set of runs

In each one of the subplots of Fig. 84 are represented the three different sets of accuracies. For each set four columns are represented, each one pertaining to a different SNR level. Each column contains all the values of hyperparameters included in Tab. 13.

As expected the higher the SNR the lower the accuracies. There is not a clear trend allowing one set of hyperparameters stand out from the others.

2.4. WaveNet

Convolutional Neural Networks (CNNs, also known as convnets) are a type of Machine Learning algorithms originally used for image processing. Their main characteristic is given by the presence of several filters which are passed through the whole input. Every filter has a predefined amount of weights and biases which at every training step are shifted to cover the entire input data. This means that all neurons shares a set of weights equal for all of them, on the contrary of MLPs where each neuron has its own weights to optimise.

During last years, convnets broadened their application also to time-series inputs, data types which are usually associated with RNNs. CNNs has shown several improvements in terms of training velocity, robustness of weights optimisation and need of memory with respect to RNNs. In particular, Google has recently developed a CNN algorithm called WaveNet [9] whose purpose is to predict and generate new audio data. It has already shown very good results so now is implemented for the binary classification purpose of this work, in order to see if it can transfer its strengths to logistic regression tasks.

The structure of the architecture used is summarised in Fig. 85 (the input is the 2000 time-step long signal acquired from the receiver but only the first 16 are reported). The main hyperparameters of this architecture are:

- number of filters (*nf*): identifies the number of features that are extracted from every layer of the network; every layer is set to have the same number of filters;
- kernel size (*ks*): indicates the number of neurons in the previous layer that are used to compute the output of a certain neuron. Dealing with one-dimensional CNNs means that the computation of the output of a neuron is made considering filters as long as the kernel size value and as wide as the number of features extracted by the previous layer (i.e. one if the previous layer is the input signal, "number of filters" otherwise);
- dilation depth (*dd*): it is how far back in the sequence the information is taken by every neuron. For WaveNet architecture the dilation depth is fixed and strictly related to the layer considered and the kernel size:

  $$dd = ks^{layer-1} \tag{22}$$

  For example Fig. 85 represents the case with *ks*=2. The output of a neuron in the first layer is computed from two (i.e. *ks*) neurons in the input layer: one from the same time-step and one from *dd*= $2^{1-1} = 2^0 = 1$ time-step before; the output of a neuron in the second layer is computed from two (i.e. *ks*) neurons in the first layer: one from the same time-step and one from *dd*= $2^{2-1} = 2^1 = 2$ time-steps before; the output of a neuron in the third layer is computed from two (i.e. *ks*) neurons in the second layer: one from the same time-step and one from *dd*= $2^{3-1} = 2^2 = 4$ time-steps before, and so on for the other layers;

- pool size (*ps*): identifies the kernel size of the pooling layers present in the final convolutions;
- padding: it is set to "causal", which means that every neuron is computed taking information only from neurons which are "back in time" (i.e. from the left, in the example of Fig. 85).

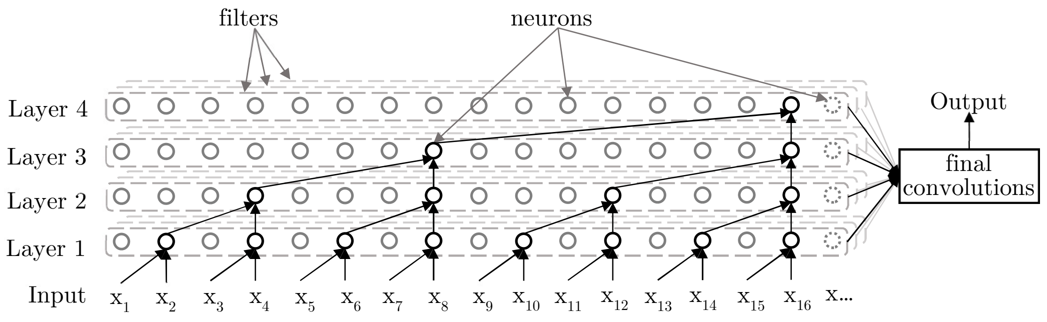

filters
neurons
Layer 4
Layer 3
Layer 2
Layer 1
Input
x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15} x_{16} x...
Output
final convolutions

Fig. 85: scheme of WaveNet architecture

The computation of the final output of the architecture is computed accounting for the output of every layer: they are summed, convolved, average pooled and passed through the sigmoid activation function (eq. (2)) to obtain one value between 0 and 1.

Dilation depth and causal padding are conceptually the most important features of WaveNet: they allow the network to account for information coming only from previous time-steps and store them for a long time along the sequence, similarly to LSTM architectures.

2.4.1 First set of trainings

The hyperparameters used for this set of runs are:

Variable	Values cycled
Kernel size (*ks*)	2, 3, 4
Dilation depth (*dd*)	4, 6
Number of filters (*nf*)	8, 16, 32
Pool size I (*psI*)	2, 4
Pool size II (*psII*)	10, 20

Tab. 14: WaveNet architecture hyperparameters - first set of runs

Pool size I and pool size II are the kernel sizes of two different pooling layers pertaining to the “final convolutions” box of Fig. 85. The number of parameters trained in this set of runs ranges from 2 thousand to 70 thousand. The number of epochs is set to be equal to 900.

For these runs only one sensor and one SNR have been used since the main scope at this step is to understand which hyperparameters performs better than the others. After the best ones are chosen, every model will be trained again with only the most promising sets of hyperparameters and with datasets coming from combinations of all the receivers and SNRs.

Results

All the combinations of hyperparameters have been trained with a dataset coming from sensor 2 and SNR 4. The noise level is doubled with respect to the one used for the first set of trainings of MLP and LSTM architectures. It is recalled from the chapter 1.2.2 “Adding noise” that the levels of the different noises have been tested throughout the entire thesis work in order to understand the limits above which the models started to have low accuracies. Anyway, the three types of architectures will be compared starting from the same datasets in the following chapter. The next figures are composed in the following way:

- accuracy value (as a percentage) on y axis;
- unique code identifying the set of hyperparameters in x axis. In particular:
  - *psI* is pool size I value;
  - *psII* is pool size II value;
  - *nf* is the number of filters;
  - the last three-digits code is an index that can be used to faster compare the accuracies of the same set of hyperparameters (i.e. a combination of *psI*, *psII* and *nf*) across the subplots in the same figure (it is not a hyperparameter pertaining to the neural network);
- different colours of the circles (relating hyperparameters sets to accuracy percentage) depending on the sensor used, as stated in the legend.

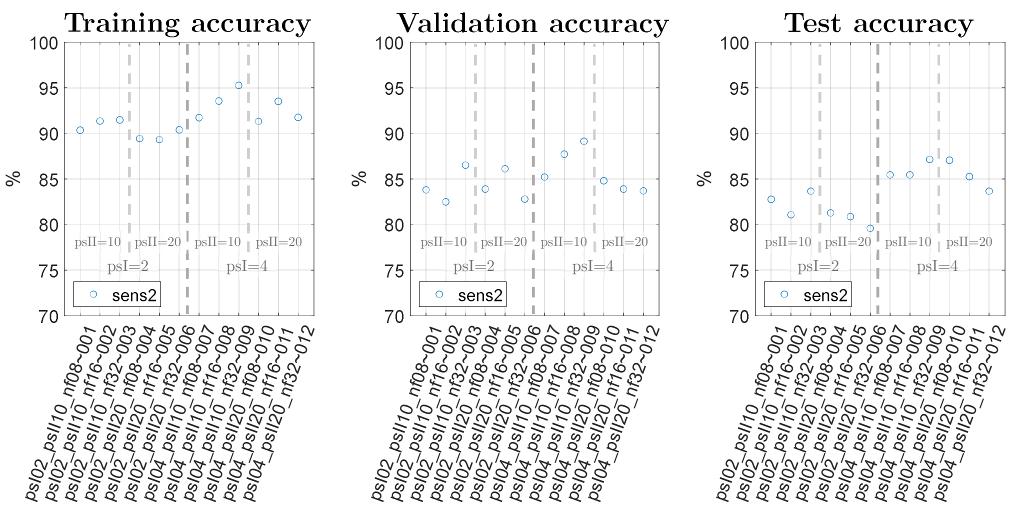


Fig. 86: WaveNet - accuracy plots: dd=4, ks=2

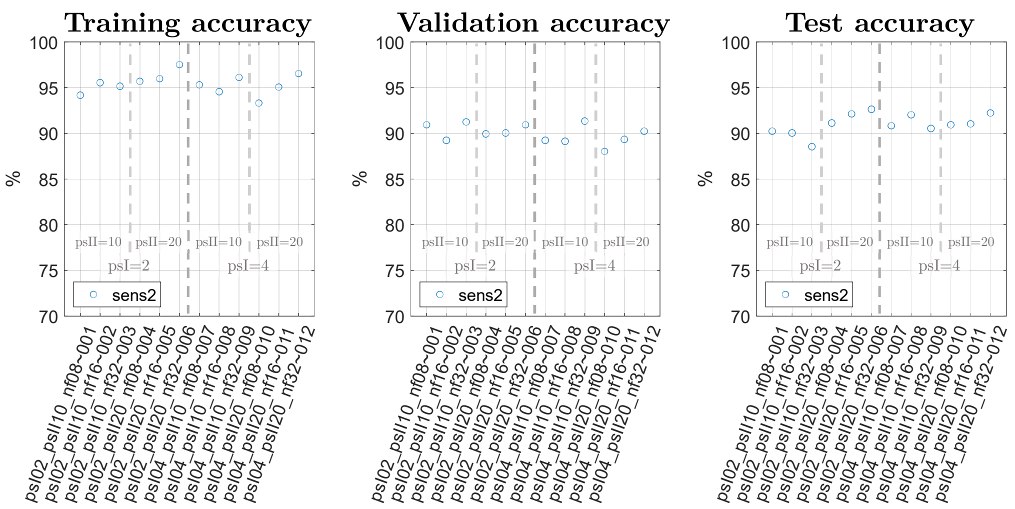


Fig. 87: WaveNet - accuracy plots: dd=4, ks=3

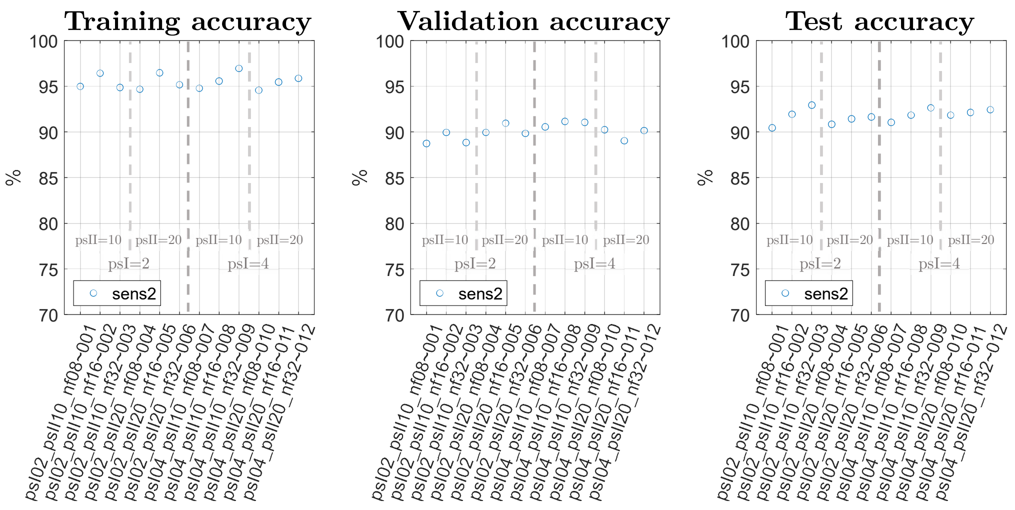


Fig. 88: WaveNet - accuracy plots: dd=4, ks=4

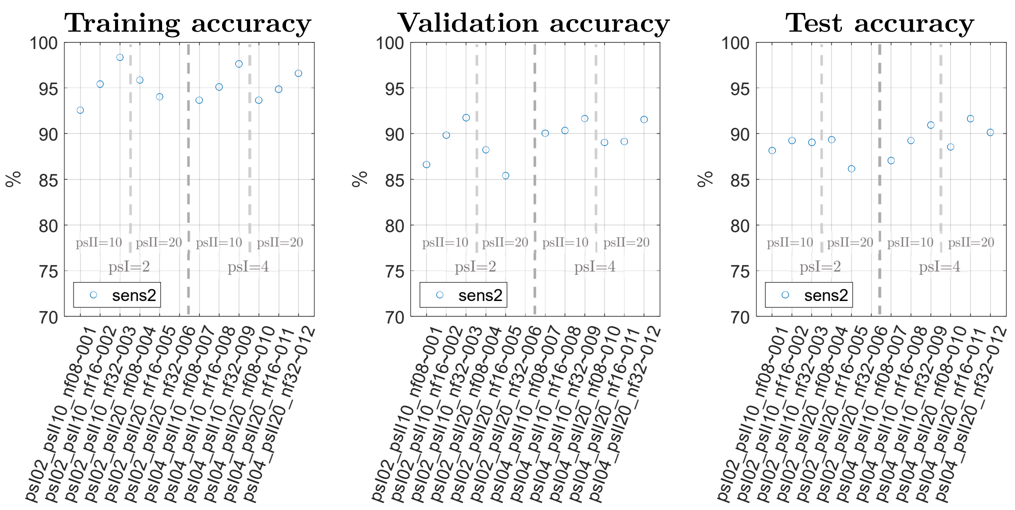


Fig. 89: WaveNet - accuracy plots: dd=6, ks=2

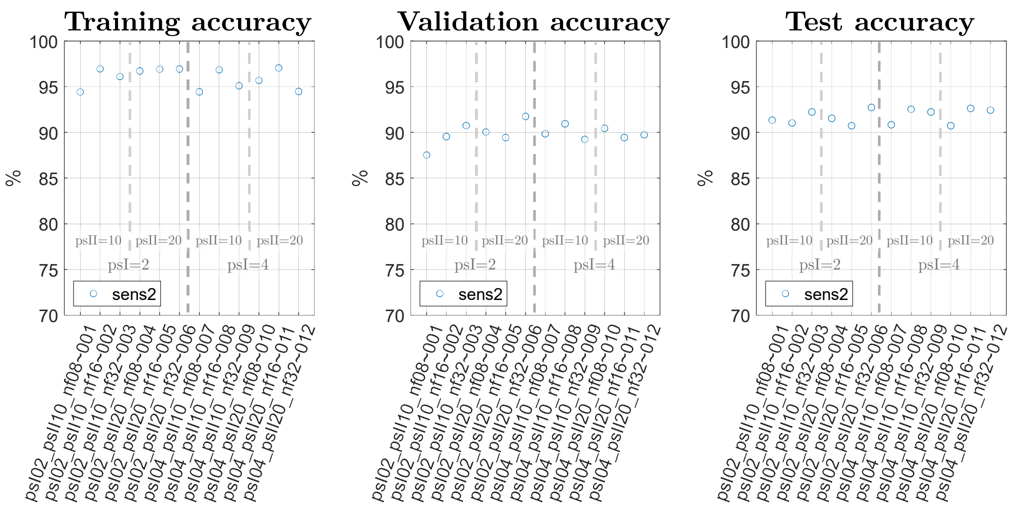


Fig. 90: WaveNet - accuracy plots: dd=6, ks=3

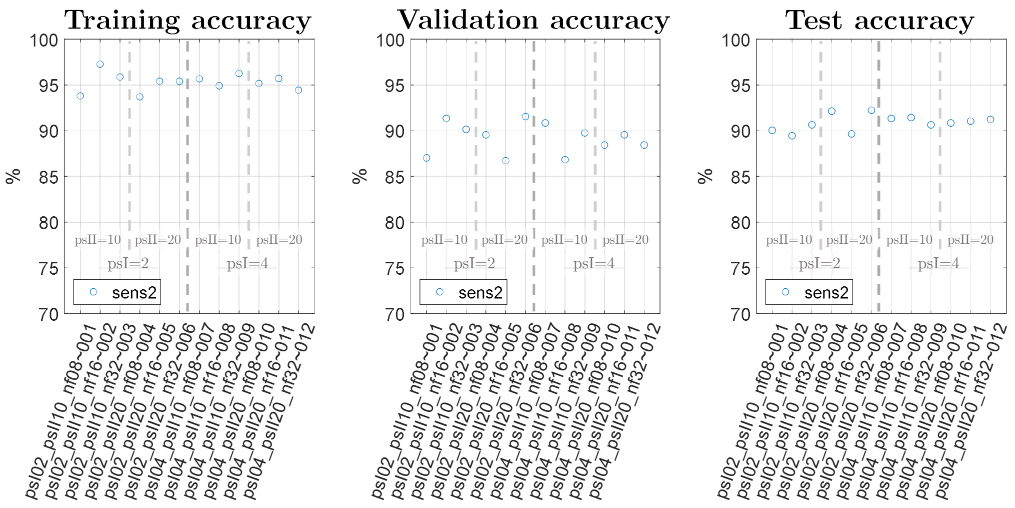


Fig. 91: WaveNet - accuracy plots: dd=6, ks=4

Due to the higher level of noise used for these simulations also the training set is not able to reach 100% accuracy. To better understand the differences among these sets of parameters the test set accuracies are grouped and shown in Fig. 92: *ks*=2 has remarkable lower performance with respect to the other values, especially for dilation depth equal to 4; between *ks*=3 and *ks*=4 there are very low differences looking at the accuracies but their training time differs a lot: *ks*=3 trains almost 3 times faster than *ks*=4. Due to these considerations for the next runs only *ks*=3 will be used; looking at the central column of Fig. 92 (*ks*=3): *psI*=4 leads to slightly better accuracies in general; *psII* does not present a clear trend when *psI*=4. To reduce the number of hyperparameters to try only *psI*=4 and *psII*=10 will be kept for the following runs.

The plot of the losses of Fig. 93 represents an overall monotonic descent for training and validation sets, both converging to an asymptote.

Given that these are quite promising results no further hyperparameters have been tried.

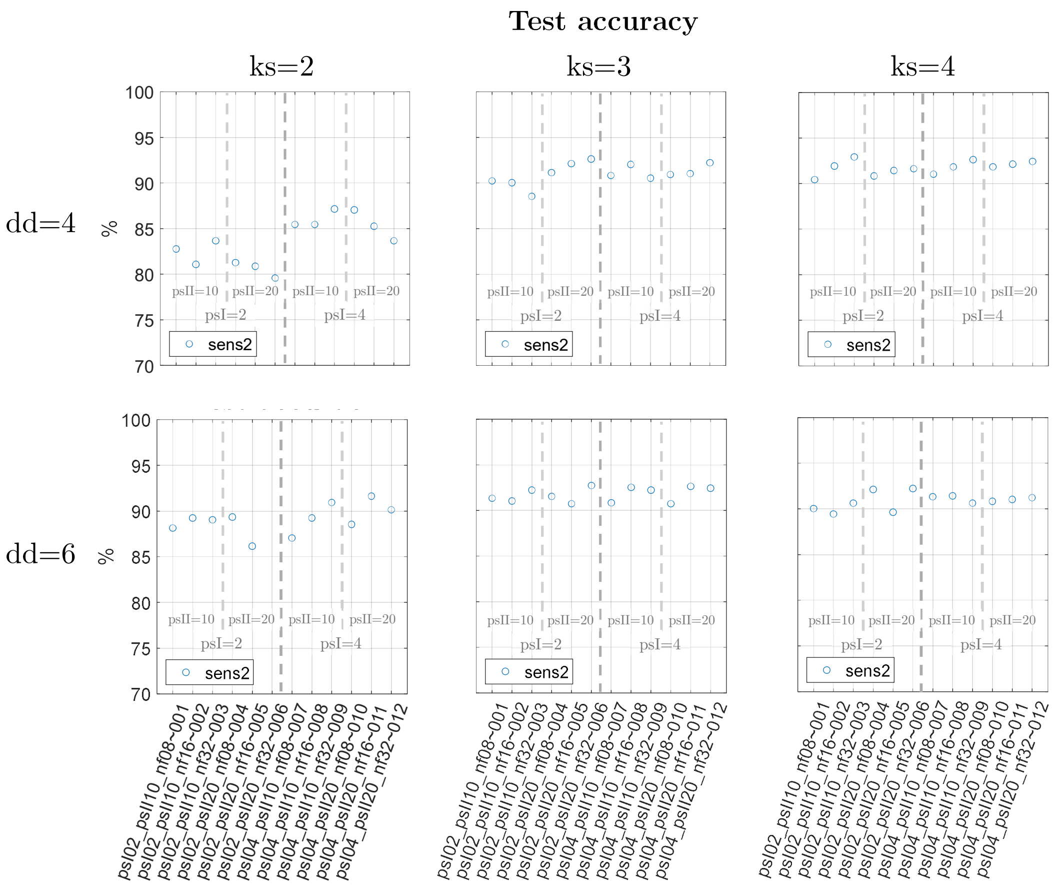


Fig. 92: WaveNet - test accuracy plots: comparison for all the hyperparameters used

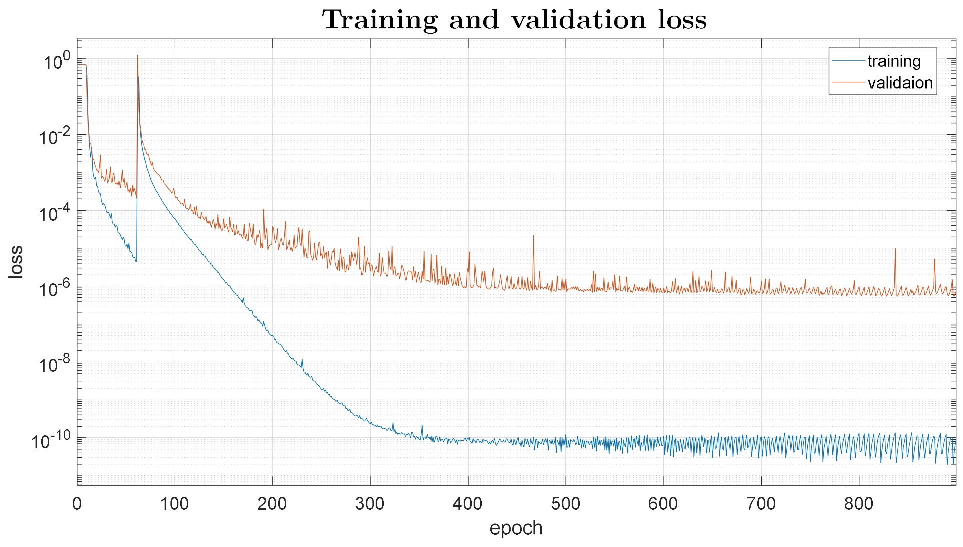


Fig. 93: WaveNet - loss plot: representative case of the trend of training and validation losses along epochs

2.4.2 Best WaveNet hyperparameters performance – comparison with all sensors and SNRs

Also for WaveNet architecture a final set of runs with all the sensors (Fig. 9, page 18) and SNRs is made. A summary of the final comparison tests for WaveNet is reported in Tab. 13 while the relative plots are in Fig. 94.

Variable	Values cycled
Kernel size (*ks*)	3
Dilation depth (*dd*)	4, 6
Number of filters (*nf*)	8, 16, 32
Pool size I (*psI*)	4
Pool size II (*psII*)	10
Sensor	1, 2, 3
SNR	1, 2, 3, 4

Tab. 15: WaveNet architecture hyperparameters - final set of runs

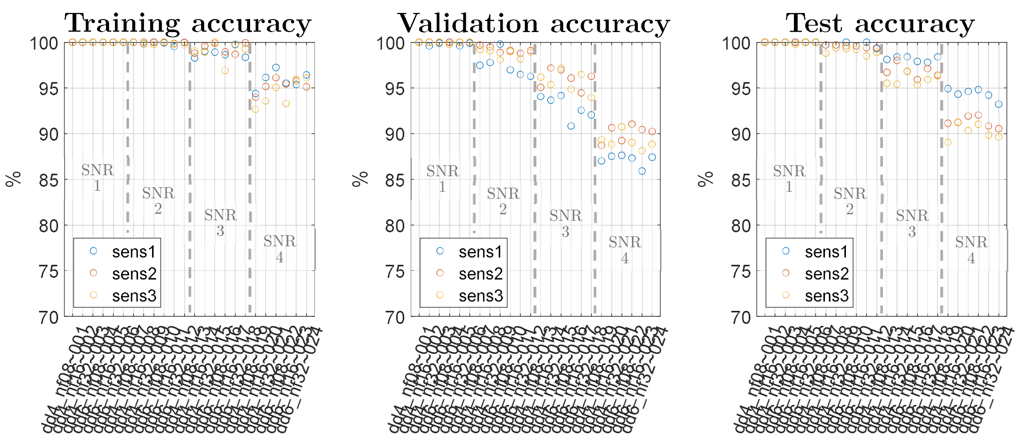


Fig. 94: WaveNet - accuracy plots: final set of runs

In figure above are represented the three different set accuracies. For each subplot four columns are represented, each one pertaining to a different SNR level. Each column contains all combinations of hyperparameters included in Tab. 15.

As expected the higher the SNR the lower the accuracies. There is not a clear trend allowing one set of hyperparameters stand out from the others.
Looking at the lowest noise it is possible to appreciate how training, validation and test sets reach 100% accuracy with almost all the hyperparameters sets. It is interesting for this case to look at the predictions of WaveNet in Fig. 95: the plot identifies the distribution of the predictions of every example coming from the test set. It can be seen how both pristine and defect signals are well separated: the difference between prediction (real number between zero and one) and actual label (either zero for "pristine" or one for "defect") is smaller than 0.05 for all the examples.

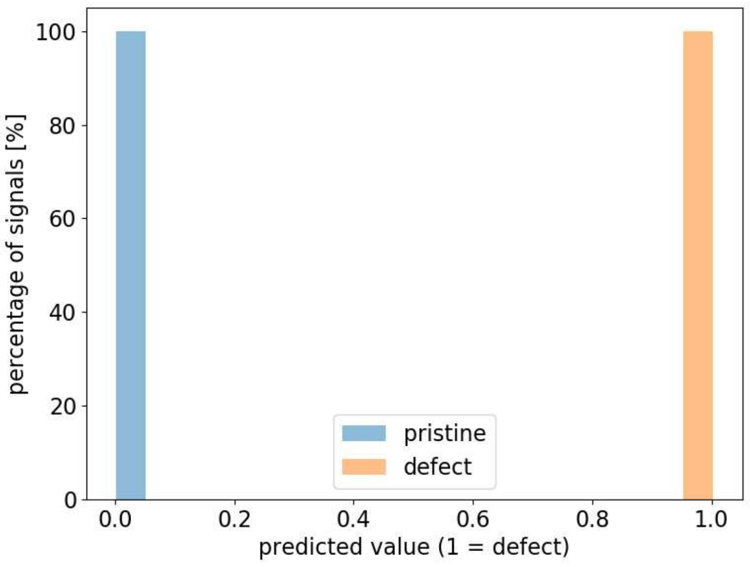


Fig. 95: WaveNet – distribution of the predictions of a model when test set examples are fed as input

2.5. Comparison of MLP, LSTM and WaveNet architectures

To compare the performance of the three different typologies of architectures trained in this study their generalisation capabilities are discussed. This characteristic can be evaluated looking at the test set accuracies for the three architectures.

Already from the highest SNR used (Fig. 96) can be appreciated how MLP architectures are the ones that struggle the most achieving high accuracies in the test set, having indeed low generalisation performance. Doubling the noise level (Fig. 97) also LSTM is not able to perfectly divide all the pristine cases from all the defective ones; WaveNet instead presents still a couple of combinations of sensors and hyperparameters able to completely divide pristine from defective signals. As can be easily expected, the higher the noise the lower the accuracies; in any case WaveNet architectures show in general way better performance than the others.

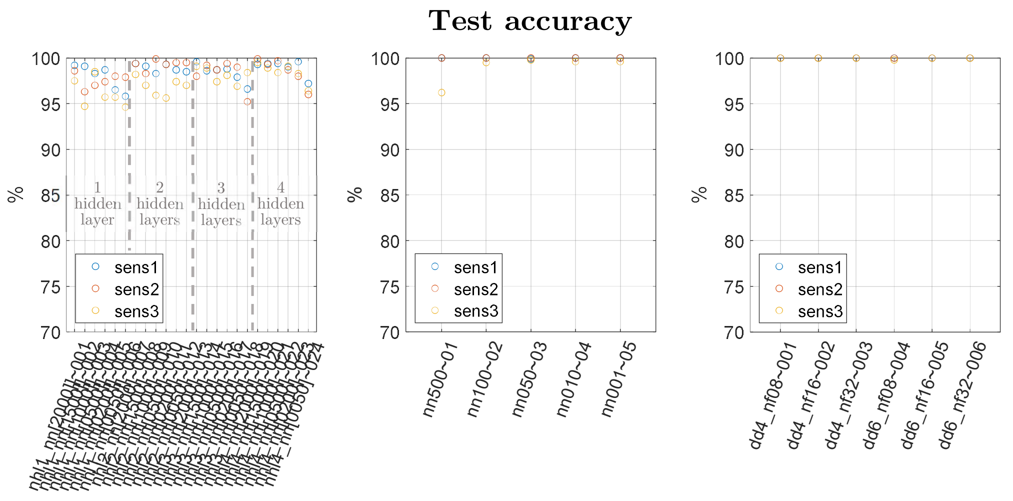


Fig. 96: architectures comparison - test accuracy plots, SNR 1

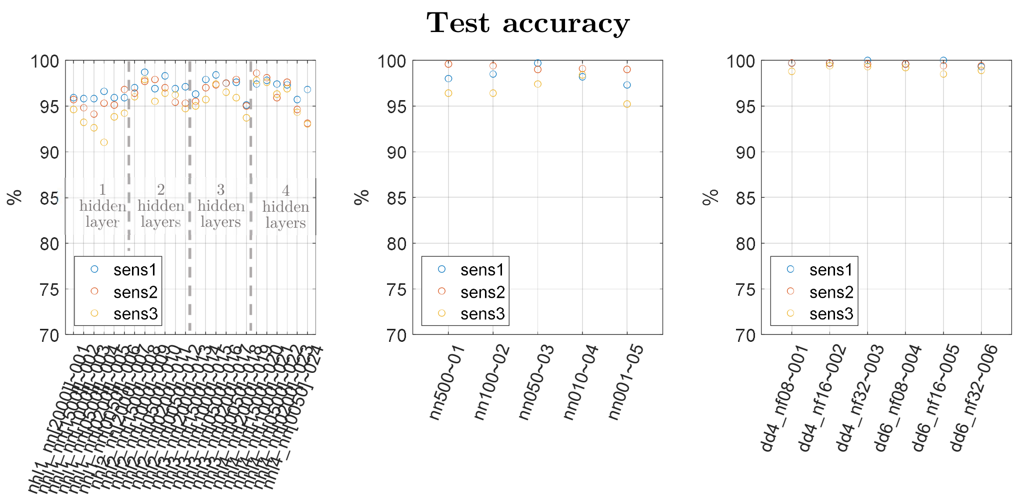


Fig. 97: architectures comparison - test accuracy plots, SNR 2

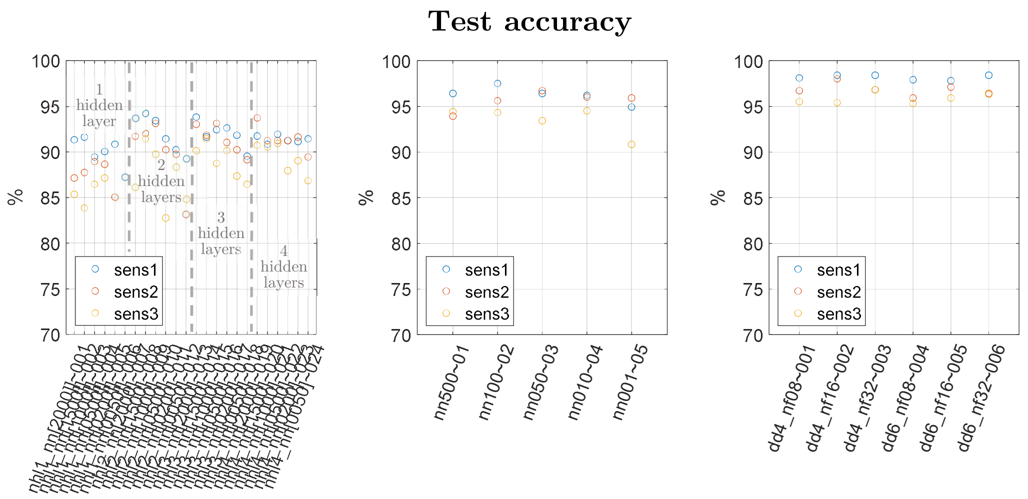


Fig. 98: architectures comparison - test accuracy plots, SNR 3

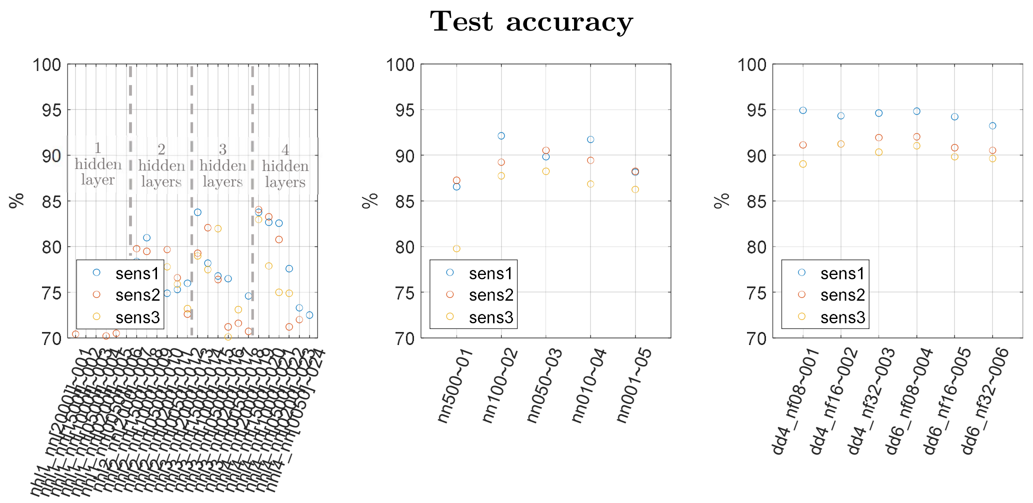


Fig. 99: architectures comparison - test accuracy plots, SNR 4

Recapping the main aspects shown during all these simulations:

- MLP architectures are among the easiest ML algorithms to implement, making it also the fastest among the 3 architectures compared:

Architecture	Training epochs	Highest training time [s]	Mean time per epoch [s/epoch]
MLP	3000	3900	1.3
LSTM	180	82000	456
WaveNet	900	6000	6.7

Tab. 16: architectures comparison – training duration

  Their main limitation encountered in this study by MLP probably comes from its lack of possibility to exploit the chronological order of time-sequence signals: every neuron computes its output using all the data coming from previous layers, not only the ones coming from "back in time" steps;
- LSTM and WaveNet have considerably higher performance than MLP: the first two showed very high accuracy with relatively lower need in discussing many combinations of hyperparameters; the latter have been trained in many different configurations (without regulariser, with dropout and L2 regularisers, changing neurons distribution along layers, pre-processing the datasets with PCA) and still the results did not effectively improve;
- LSTM showed very good results with a ridiculous amount of hyperparameters tuning needed (only 6 different combinations) but the drawbacks are significant: the computational cost is huge since one set of hyperparameters may need up to 24 hours of calculations to complete the training and achieve good results (even if LSTM reduces RNN's vanishing gradient problem); the stability of the convergence is weakened by the exploding gradient, typical issue of RNNs; the memory amount needed to store all the data through the recurrent steps is huge and a large amount of memory is needed to accomplish the computations;
- WaveNet takes advantage from the integration of its nature with powerful GPUs, letting such complex architectures to train rapidly. This allowed the training of a consistent amount of combinations: the hyperparameters tunable to change the complex structure of WaveNet architectures are a lot (five different variables as reported in Tab. 14, against MLP's and LSTM's two variables as reported respectively in Tab. 7 and Tab. 12).

It has been shown that accuracies strictly depend on the SNR of the dataset. Clearly, if the noise is too high with respect to the reflection risen from a defect it would be almost impossible to detect a damage. The graphs from Fig. 96 to Fig. 99 give an idea of the quality that a signal must have in order to trust, with a certain confidence, the prediction of the model used. Based on these considerations it would now be ideally possible to set the power of the emitted signal and/or the number of samples to be averaged in order to obtain a signal with a high enough SNR, based on the results obtained.

2.6. Generalisation properties of deep neural networks

The training process of a ML algorithm can be defined as the procedure needed to tune the parameters of the neural network (i.e. the fitting function) in order to minimise the distance of the output predicted by the algorithm from the actual output value of a set of examples. A very basic analogy can be done with a simple linear function:

$$y = ax + b \tag{23}$$

where x is the independent variable (i.e. the input), y is the dependent variable (i.e. the output) and a and b are two coefficients (i.e. the parameters). Clearly, it is known that two distinct points (i.e. the examples) are needed to univocally set the values of the coefficients of the function, while for the other cases the problem can be defined as:

- *underdetermined* when only one example is given. In this condition there are infinite combinations of parameters which are able to correctly relate the input and the output of the example;
- *overdetermined* when more than two examples are given. In this condition there are no combinations of parameters which are able to correctly relate all the inputs to the outputs of the examples. This condition can be handled by defining an error function and minimising it, by tuning the parameters.

It is quite common in ML field to have that the number of tunable parameters of a neural network is higher than the number of examples available for its training. This condition falls in the underdetermined case and might lead to overfitting: the network learns to correctly predict the outputs of the set of examples used during its training (i.e. the training set) while shows very bad accuracies in predicting the outputs from a new set of examples (i.e. the test set). In this thesis the architectures have been trained with a set of 8 thousand examples while the number of parameters of the optimised neural networks were in the range from 2 thousand to 6 million, i.e. for the majority of the tested cases the fitting problem could be classified as underdetermined and lead to overfitting over the training set. In literature some hints can be found on the relation between the number of parameters and the number of the training examples that are required by the ML algorithm to perfectly fit the training set (e.g. a two-layer neural network needs $2m + l$

parameters to correctly fit a set of m examples, each one with a length equal to l [19]). It is more difficult to find a similar relation when considering generalisation capabilities of ML algorithms, since in this case also the distribution of the dataset into training, validation and test set may affect the accuracies of the predictions.

The main precaution used in ML field to avoid overfitting is regularisation, which can be defined as "any supplementary technique that aims at making the model generalize better, i.e. produce better results on the test set" [20]. Regularisation techniques can be implemented both explicitly or implicitly: an example of the former can be L2 or dropout (used during optimisation of MLP algorithms in chapter 2.2.2) while an example of the latter can be Adam algorithm (used as optimiser for all the neural networks trained; it adapts the learning rate at every training step) or addition of noise to the input data. Many other regularising terms hide in ML algorithms [20] but it is still lacking a theoretical explanation of general validity demonstrating why deep neural networks are not suffering too much of overfitting.

Recalling the results shown in the previous chapters:

- all the types of architecture present high training accuracies up to a certain SNR value, indicating their ability to correctly divide pristine examples from defective examples (considering only the examples used for training);
- MLP architectures (Fig. 62 to Fig. 65, page 61) show that test set accuracies reach similar values to those of the training set only for some cases, i.e. SNR 1 with two or more hidden layers. This goes in favour to the fact that deeper neural networks can generalise better;
- LSTM architectures (Fig. 84, page 78) show great generalisation capabilities for SNR 1 (very low difference between training and test set accuracies) and, for few sets of hyperparameters, also for SNR 2;
- WaveNet architectures (Fig. 94, page 86) show great generalisation capabilities up to SNR 2. With the higher levels of noise there are few cases where the distance between accuracies of training and test sets is negligible (good generalisation);
- in general, the higher the noise the lower the ability of a neural network to generalise to the test set, since the higher the difficulty in extracting information pertaining to damages (recall the noise effect on signals from Fig. 19 to Fig. 30, pages 29-34).

The understanding of deep ML algorithms still lacks of theoretical explanation about their good generalisation performance, which apparently cannot be easily related to the number of parameters of a neural network [19]. From the analysis of training and test sets accuracies of all the ML architectures used in this thesis, it is possible to conclude that deeper and more sophisticated neural networks can be less prone to overfit to the training set, i.e. they show better generalisation capabilities than shallower and simpler neural network architectures.

3. Temperature extrapolation capability

Given the promising performance achieved with BSS technique, ML architectures are now stressed to understand their capability in detecting defects when a signal is acquired at a temperature out of the training range. This task is performed with the best hyperparameters sets of WaveNet architecture, which showed the best accuracies among all the others.

The steps needed to evaluate the extrapolation capability of the models are:

- redefine the dataset division among training, validation and test sets
- train the models on the new dataset and evaluate their accuracies

3.1. Training, validation and test sets

The spatial allocation of defects in training, validation and test sets does not change (the same as in Fig. 31, page 36). Instead, the temperatures division is now set as in Fig. 100:

- training (blue) and validation (green) sets consider the first 20 temperatures of defect signals (i.e. from 20°C to 39°C); regarding pristine signals, validation set accounts 1 temperature every 3, the others are stored in training set (1 over 2 ratio is used to keep higher the number of training set examples with respect to validation set examples);
- test set is divided in 3:
  - yellow examples ranges from 40°C to 49°C;
  - orange examples ranges from 50°C to 59°C;
  - red examples ranges from 60°C to 69°C.

Both defect and pristine signals carry all the temperatures in the respective ranges. This subdivision of the test set is done to better analyse the extrapolation capabilities of the model at different temperature distances from the range used to train the model.

Blank spaces are examples that are not used to train, validate or test the model.

The models in chapter 2 have been trained with a dataset where the baseline signal for BSS was the pristine example in the middle temperature of that range (i.e. 45°C in 20°C to 69°C range). To set the same condition, this new dataset has been temperature-compensated using as baseline signal the pristine example at 40°C for BSS, i.e. the middle temperature of the actual training temperature range.

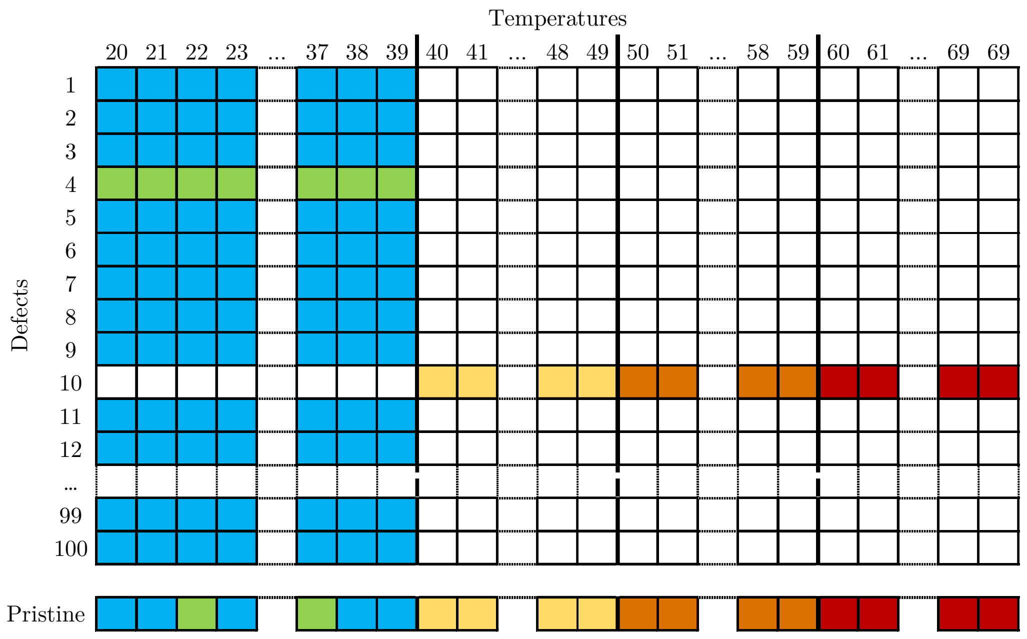


Fig. 100: division of defects and pristine temperatures for extrapolation case: training set (blue), validation set (green) and 3 test sets (yellow, orange and red respectively)

The actual number of examples forming each set is reported in Tab. 17.

Set	Defect	Pristine	Total	Defect/Pristine ratio
Training	$80 \cdot 20 = 1600$	$1 \cdot 14 = 14$	1614	114
Validation	$10 \cdot 20 = 200$	$1 \cdot 6 = 6$	206	33
Test (yellow)	$10 \cdot 10 = 100$	$1 \cdot 10 = 10$	110	10
Test (orange)	$10 \cdot 10 = 100$	$1 \cdot 10 = 10$	110	10
Test (red)	$10 \cdot 10 = 100$	$1 \cdot 10 = 10$	110	10

Tab. 17: number of defect and pristine examples in every set for extrapolation case – unbalanced distribution

As previously explained in chapter 1.2.3 “Creation of training, validation and test sets”, the number of pristine examples should be equal to the number of defects for all the sets. After creating multiple copies of pristine signals the final distribution of examples is the one reported in Tab. 18.

Set	Defect	Pristine	Total	Defect/Pristine ratio
Training	1600	$14 \cdot 114 = 1596$	3196	1
Validation	200	$6 \cdot 33 = 198$	398	1
Test (yellow)	100	$10 \cdot 10 = 100$	200	1
Test (orange)	100	$10 \cdot 10 = 100$	200	1
Test (red)	100	$10 \cdot 10 = 100$	200	1

Tab. 18: number of defect and pristine examples in every set for extrapolation case – balanced distribution

Pristine signals are copied before the noise is added to the dataset and the division of pristine and defect signals in training, validation and test sets is kept constant for all the models.

3.2. Extrapolation performance

WaveNet is the architecture which showed the best performance in defect detection task, thus it is the one chosen to perform the extrapolation test. The hyperparameters used to train this models are the ones reported in Tab. 15 at page 85, i.e. those leading to the best accuracies.

From Fig. 101 to Fig. 104 are represented training, validation and test set accuracies for the 4 levels of SNR. In the x axis are reported the six combinations of hyperparameters used to train the models. As expected, in general the farther the temperature distance from the training range the lower the accuracies. It is interesting the fact that for SNR 1 there are several combinations of hyperparameters and sensors for which the accuracies reach 100% even for temperatures 30°C out of the range used to train the model.

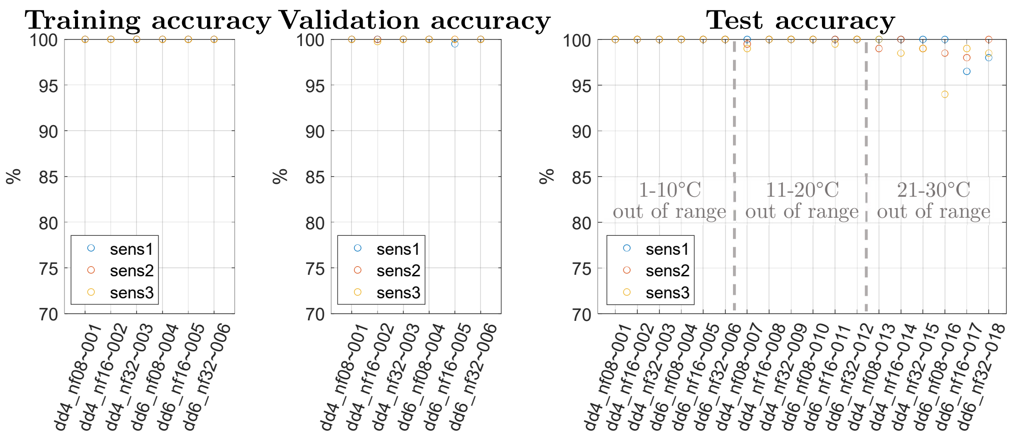


Fig. 101: accuracy plots for temperature extrapolation test – SNR 1

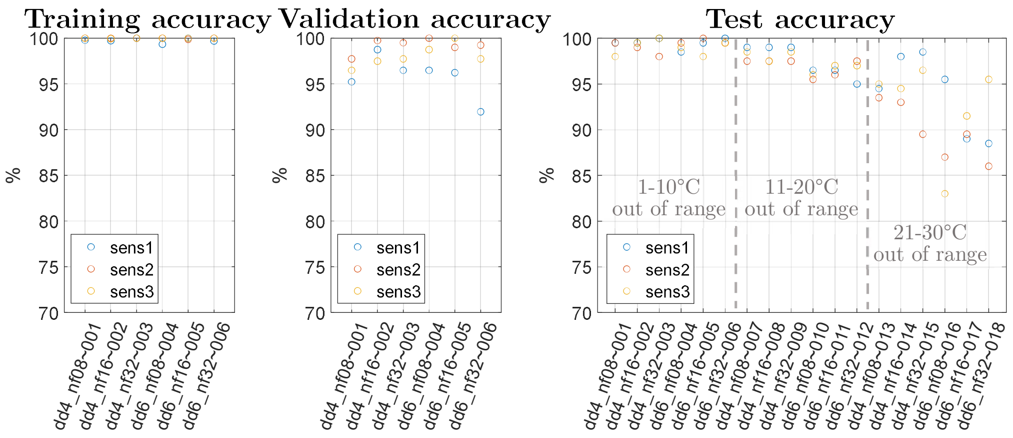


Fig. 102: accuracy plots for temperature extrapolation test – SNR 2

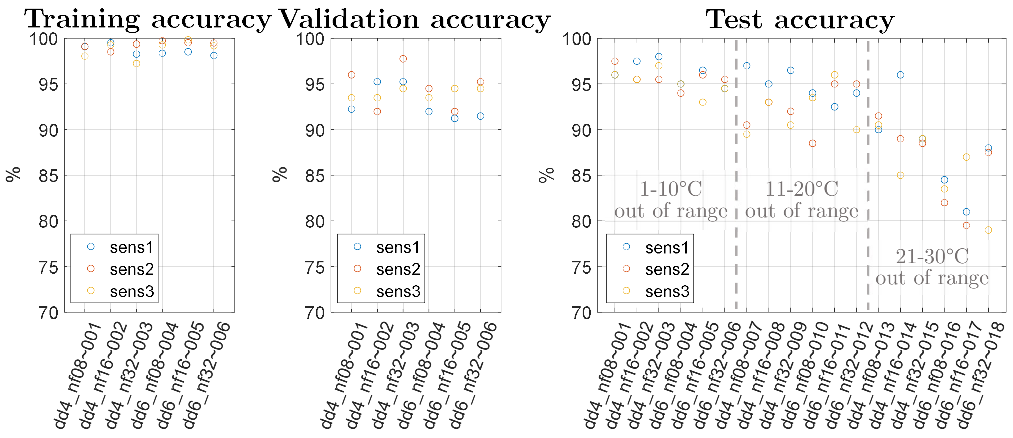


Fig. 103: accuracy plots for temperature extrapolation test – SNR 3

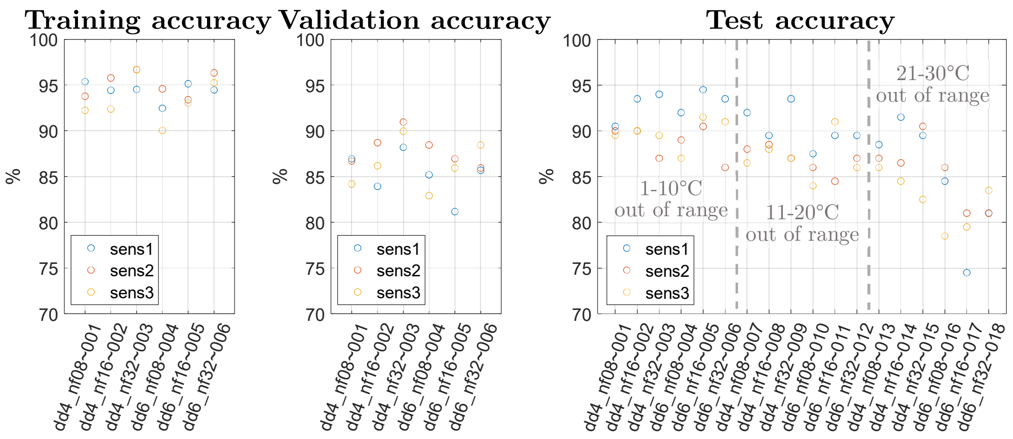


Fig. 104: accuracy plots for temperature extrapolation test – SNR 4

4. Experimental dataset

In this section the ML algorithms are tested on a dataset composed by signals acquired from actual experiments. The aim is to compare the performance of the WaveNet architectures in detecting defects from both numerical simulations and experimental signals, thus demonstrating the benefits in defect detection task from the application of deep neural networks to Lamb waves.
ML algorithms are now tested on an already existing dataset, which was released to the public on "Open Guided Waves" website [21] [22]. Its composition is very similar to that obtained for numerical simulations: signals are acquired from a couple of emitter-receiver piezoelectric sensors in a range of temperatures, all for both pristine and defective cases, the latter obtained by locating defects in different positions in the plate. However some differences apply: the specimen is a square plate made of carbon fiber reinforced plastic (CFRP) and the emitted signal, which is based on the same 5 cycle Hanning windowed sine wave, has in this case a frequency of 40 kHz.

In the next sections is presented:

- a brief introduction summarising the experimental setup presented in the original paper [21] and the signals downloaded from the website;
- the division of the dataset between training, validation and test sets;
- the results obtained from the algorithms trained on these real data.

4.1. Dataset composition

Fig. 105 shows the setup of the experimental dataset with the relative position of every element reported in Tab. 19. In order to reduce costs and save time, the defects were modelled sticking an aluminium disc over the plate: this method allows to affect Lamb waves propagation in a similar way as a real defect would, but with the advantage of avoiding permanent damage in the plate. In Fig. 106 is reported the example of an experimental signal.

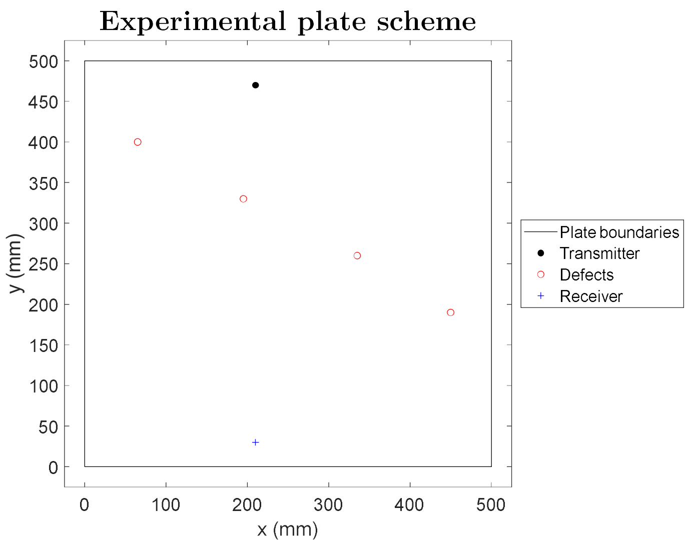


Fig. 105: CFRP plate dimensions and positions of emitter, receiver and defects

Element	X coordinate [mm]	Y coordinate [mm]
Transmitter	210	470
Receiver	210	30
Defect 1	65	400
Defect 2	195	330
Defect 3	335	260
Defect 4	450	190

Tab. 19: position of sensors and defects on the CFRP plate

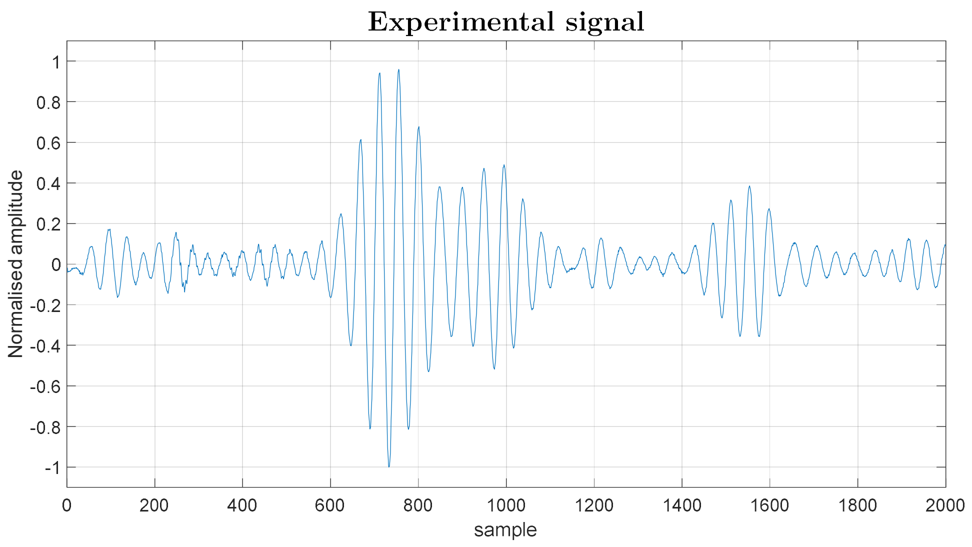


Fig. 106: example of experimental signal

The experimental signal shown in Fig. 106 is similar to those numerical simulated which were used in previous chapters. It is possible to distinguish the first arrival of the excitation package (close to sample 600) as well as the reflections from the boundary at about sample 1400. The signal seems to be shorter with respect to those used for the numerical simulation case: this is due to the combination of acquisition time and lower excitation frequency used to obtain the signals in the real case.

Obviously no noise was added to the experimental signals, while BSS technique was applied setting as baseline signal the pristine signal at 40°C, i.e. the middle of the temperature range from 20°C to 60°C. The next figures show the effect of the defects on the acquired signals: pristine and defective signals are represented overlapped on the upper subplot, while their difference is emphasised in the lower subplot. In particular, Fig. 107 to Fig. 110 show the effect of defect 1 to 4, respectively, when acquisitions are made at the same temperatures, while Fig. 111 to Fig. 114 show again the effect of defect 1 to 4, respectively, but this time when acquisitions pertain to temperatures differing of 1°C.

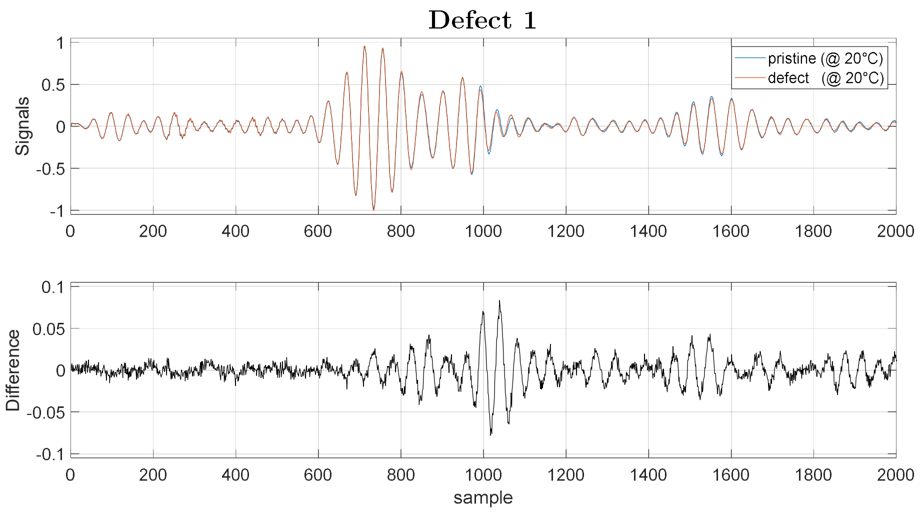


Fig. 107: impact of defect nr. 1 in Lamb waves propagation for experimental signals; pristine and defective signals are taken at the same temperature

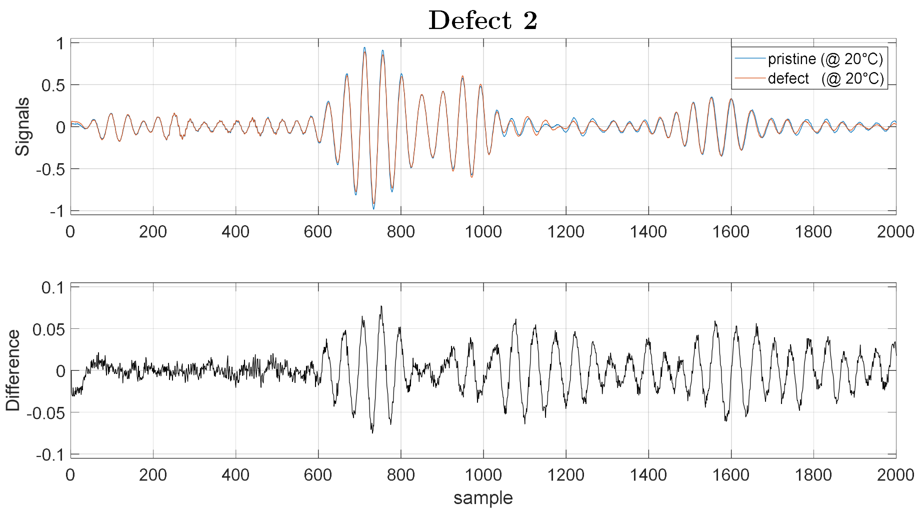


Fig. 108: impact of defect nr. 2 in Lamb waves propagation for experimental signals; pristine and defective signals are taken at the same temperature

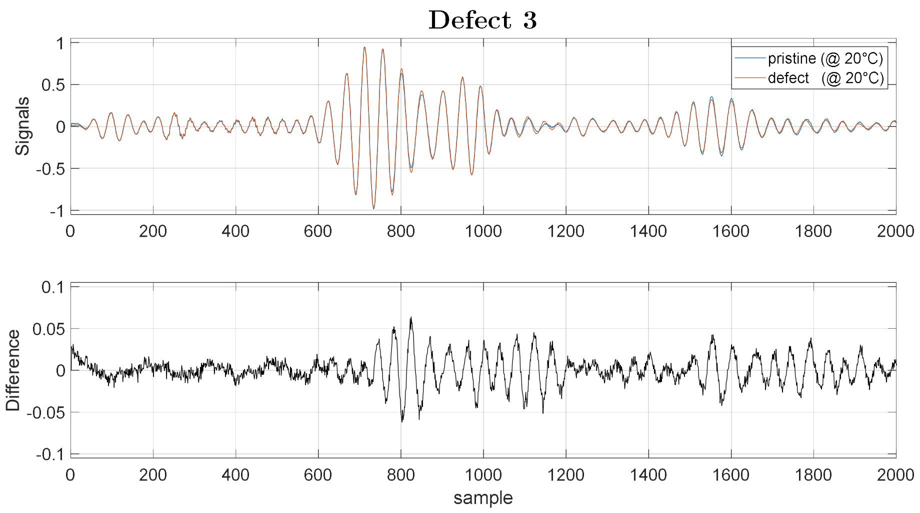


Fig. 109: impact of defect nr. 3 in Lamb waves propagation for experimental signals; pristine and defective signals are taken at the same temperature

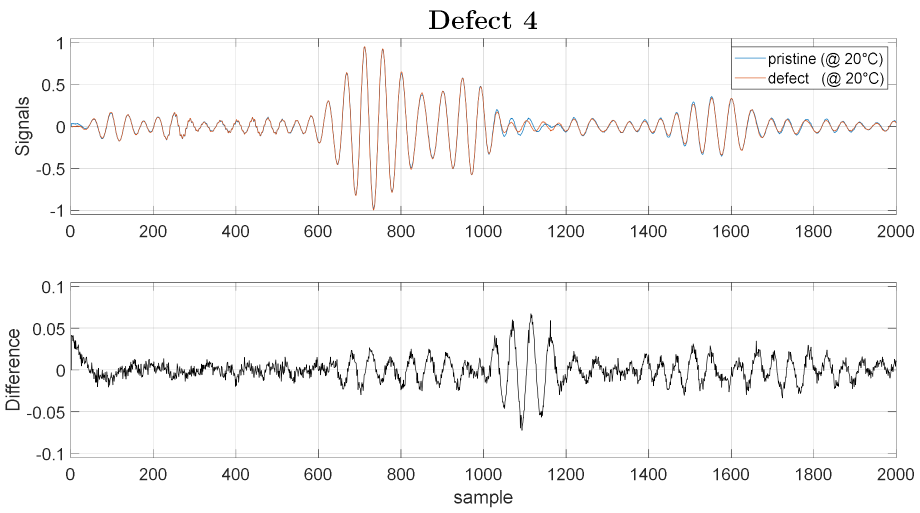


Fig. 110: impact of defect nr. 4 in Lamb waves propagation for experimental signals; pristine and defective signals are taken at the same temperature

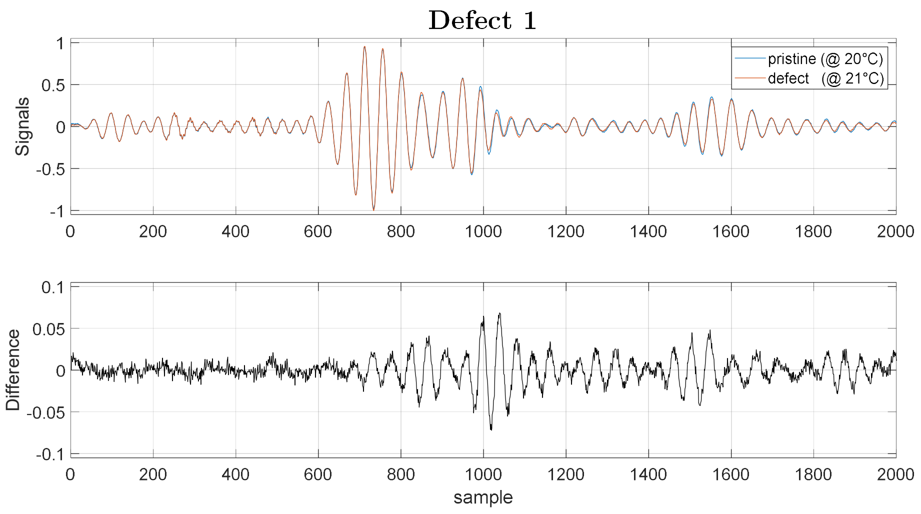


Fig. 111: impact of defect nr. 1 in Lamb waves propagation for experimental signals; pristine and defective signals are taken at temperatures differing of 1°C

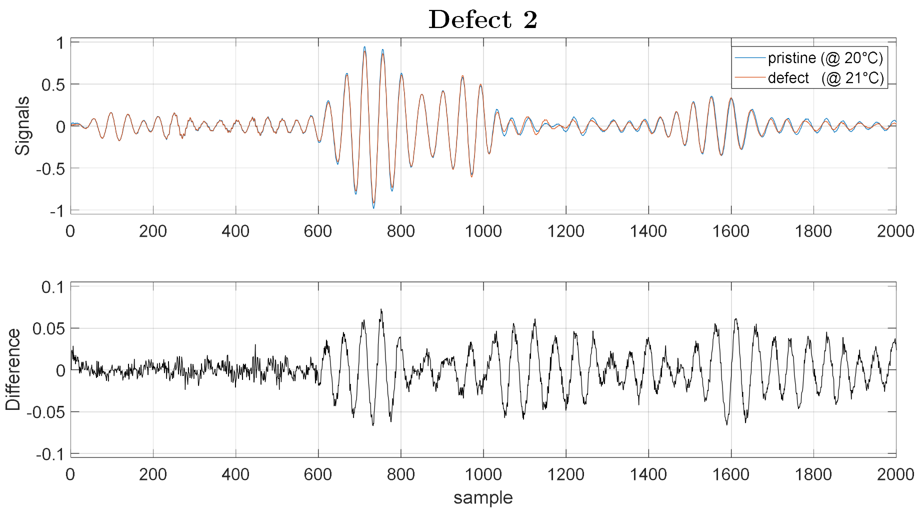


Fig. 112: impact of defect nr. 2 in Lamb waves propagation for experimental signals; pristine and defective signals are taken at temperatures differing of 1°C

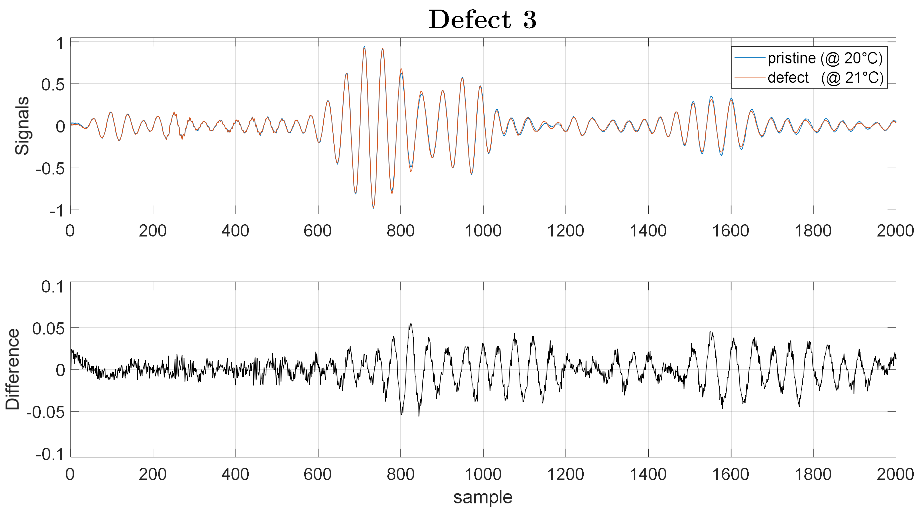


Fig. 113: impact of defect nr. 3 in Lamb waves propagation for experimental signals; pristine and defective signals are taken at temperatures differing of 1°C

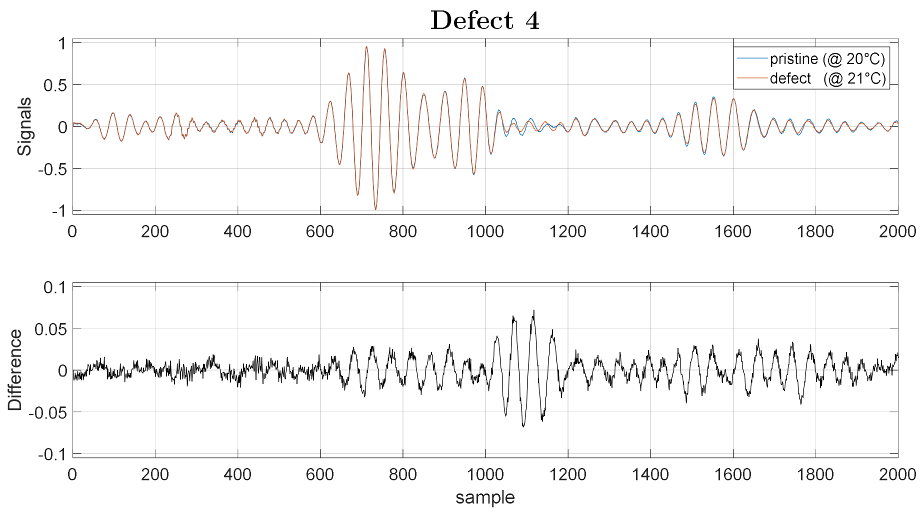


Fig. 114: impact of defect nr. 4 in Lamb waves propagation for experimental signals; pristine and defective signals are taken at temperatures differing of 1°C

The differences between pristine and defect signals (lower subplots) have lower relative amplitudes if compared to those of numerical simulations (Fig. 19 to Fig. 30, page 29 to 34) but the effect of the defect is still clearly visible. Fig. 111 to Fig. 114 confirm that the influence of the different temperatures is well diminished by BSS technique, also when applied to actual datasets.

4.2. Training, validation and test sets

The actual dataset is composed by:

- one pristine signal
- four defect signals
- 2000 timesteps
- temperatures ranging from 20°C to 60°C, at steps of 0.5°C approximately. Each signal has been acquired two times at each temperature step, one during temperature ramping up and the other during temperature ramping down.

This leads having a dataset dimensioned as in eq. (24)

$$dataset = dataset(defect\ ,\ timestep\ ,\ temperature) \tag{24}$$

where:

- defect length is equal to 5 (one pristine and four defective cases);
- timestep length is equal to 2000;
- temperature length is equal to 161.

The dataset comes from a different configuration with respect to the one of the numerical model used to train the ML algorithms in the previous chapters, thus new models need to be trained on this dataset. The low number of defect positions does not allow the 80%-10%-10% division among training, validation and test sets respectively, as used in chapter 1.2.3 “Creation of training, validation and test sets”. The actual choice is to use two defects for the training set, one for the validation set and the remaining one for the test set. Signals acquired at different temperatures but with the same defect were stored in the set corresponding to the defect itself (i.e. every defect carries 161 examples to its own set, first four rows of Fig. 115). For what regards the pristine signals, it was decided to leave a thermal distance of 1°C between training examples and validation and test examples (last row of Fig. 115), accordingly to the division of the pristine signals in training, validation and test sets chosen for the numerical dataset reported in chapter 1.2.3.

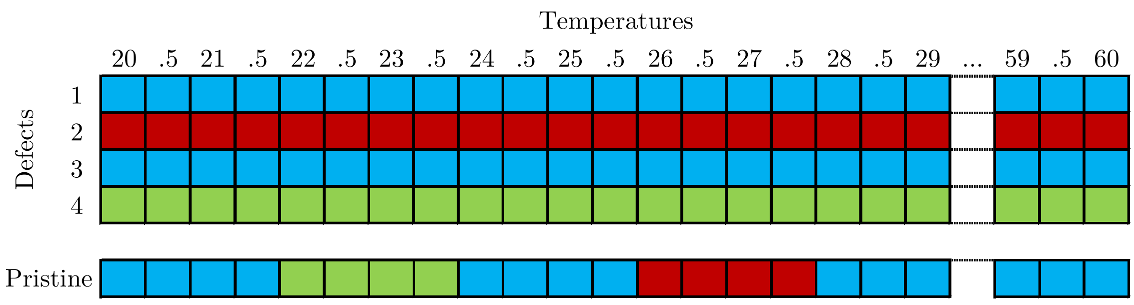


Fig. 115: division of defects and pristine temperatures in training set (blue), validation set (green) and test set (red) for the experimental dataset

The actual number of examples in the three sets is:

Set	Defect	Pristine	Total	Defect/Pristine ratio
Training	$2 \cdot 161 = 322$	$1 \cdot 81 = 81$	403	4
Validation	$1 \cdot 161 = 161$	$1 \cdot 40 = 40$	201	4
Test	$1 \cdot 161 = 161$	$1 \cdot 40 = 40$	201	4

Tab. 20: number of defect and pristine examples in every set of the experimental configuration – unbalanced distribution

Multiple copies of pristine signals are needed to balance their ratio with defect signals, achieving a distribution the closest possible to 50%-50% in every set. Finally, the division of signals in the three sets is:

Set	Defect	Pristine	Total	Defect/Pristine ratio
Training	322	$81 \cdot 4 = 324$	646	1
Validation	161	$40 \cdot 4 = 160$	321	1
Test	161	$40 \cdot 4 = 160$	321	1

Tab. 21: number of defect and pristine examples in every set of the experimental configuration – balanced distribution

4.3. Results

The algorithms tested for the experimental case are the WaveNet architectures with the same set of hyperparameters that showed the best accuracies on numerical datasets, i.e. those of Tab. 15, page 85. The next figures are composed in the following way:

- accuracy value (as a percentage) on y axis;
- unique code identifying the set of hyperparameters in x axis. In particular:
  - *validef* identifies the number of the defect used in the validation set;
  - *dd* is the dilation depth;
  - *nf* is the number of filters;
  - the last three-digits code is an index that can be used to faster compare the accuracies of the same set of hyperparameters (i.e. a combination of *validef*, *dd* and *nf*) across the subplots in the same figure (it is not a hyperparameter pertaining to the neural network);
- different colours of the circles depending on the number of the defect used in the test set, as stated in the legend.

Being the number of defects very low, the relative position of the sensors with respect to the defects may considerably affect the performance of the algorithms, hence every combination of defects in training, validation and test sets is checked.

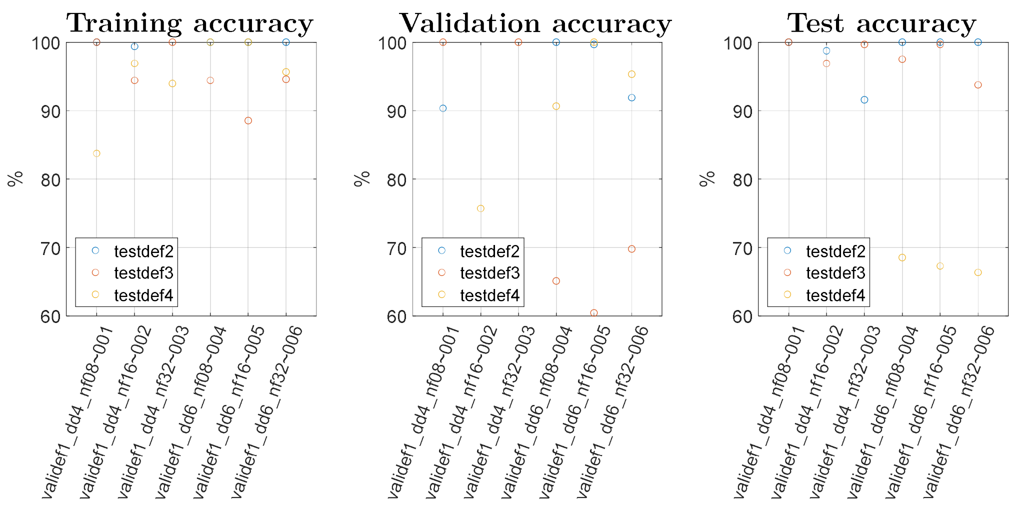


Fig. 116: WaveNet - accuracy plots with experimental dataset using defect in position 1 to validate the algorithm

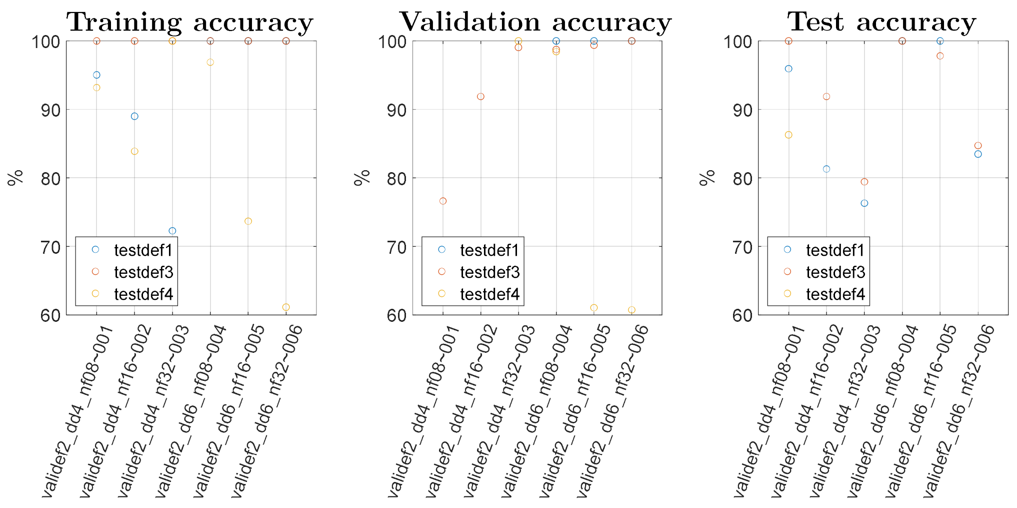


Fig. 117: WaveNet - accuracy plots with experimental dataset using defect in position 2 to validate the algorithm

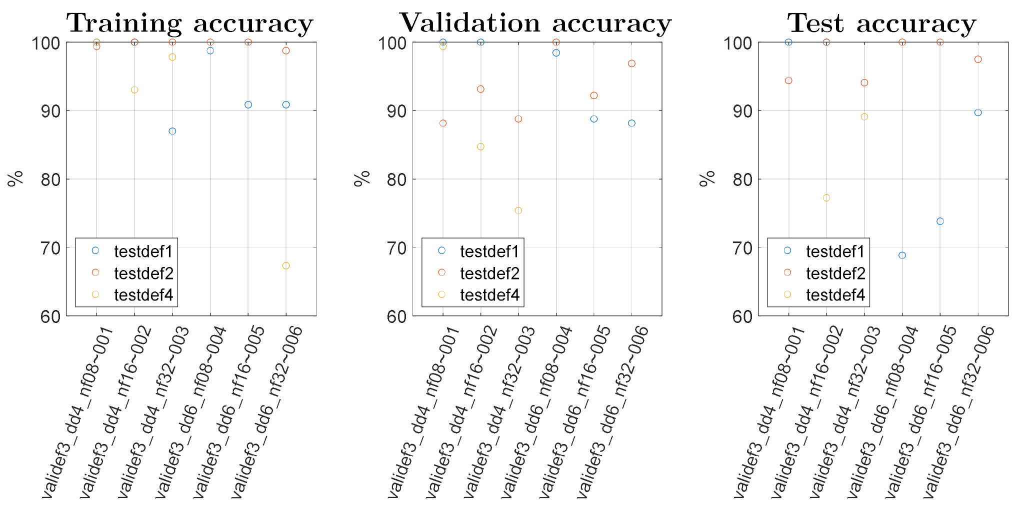


Fig. 118: WaveNet - accuracy plots with experimental dataset using defect in position 3 to validate the algorithm

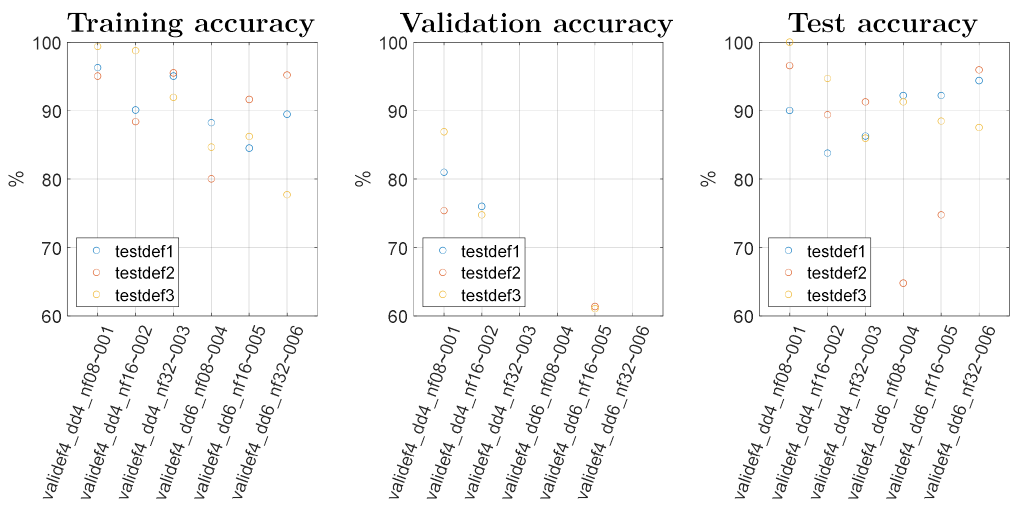


Fig. 119: WaveNet - accuracy plots with experimental dataset using defect in position 4 to validate the algorithm

Fig. 116 to Fig. 119 show that using defect number 4 in validation or test set leads to poor results: it might be due to the difficulty in detecting a defect far from the path connecting the emitter to the receiver. This consideration can justify the need in the exploitation of long signals, like it has been done with the numerical dataset: longer signals can acquire more reflections given by a certain defect, making it easier for the algorithm to detect its presence. The limited number of available experimental data (i.e. four positions of the defect) is not enough to ensure a good generalisation capability of the ML algorithms. There are anyway many combinations for which the accuracies reach 100% values, proving that WaveNet architectures can behave very well for a broad variety of applications. Finally, the accuracies obtained from the experimental dataset show values in line with those obtained from the numerical datasets, thus enforcing the correctness of the results.

5. Conclusions

The work performed in this thesis pertains to the general field of structural health monitoring. The main scope was to develop an algorithm capable to extract information about the integrity of a test structure from ultrasonic signals acquired by piezoelectric sensors installed on the structure itself. Specifically, a number of Machine Learning (ML) algorithms have been tested and compared, which were set to output a binary state, i.e. structure being undamaged or damaged. The main test structure was a Finite Element-simulated steel plate where the adverse effects of instrumentation noise and of varying temperature have been also included, although the final part of the study focused on actual experimental signals acquired from a composite plate.

The long-term goal of the study is to apply the refined ML algorithm to interpret data acquired by a number of sensor networks installed on actual structures, thus one obvious requirement is to reduce the costs associated with the data acquisition system while still obtaining a high enough accuracy of defect detection. The results showed that under the circumstances considered in this study it would be sufficient to use two sensors, one acting as emitter and the other as receiver.

The core of the research has been to test and tune different ML algorithms that were fed with the same datasets as input, those being time-based ultrasonic signals with minimal amount of pre-processing only consisting in the application of the baseline signal stretch method, to partially compensate for the different wave speeds at the different temperatures. The tested ML algorithms were:

- Multilayer perceptron (MLP), among the most common architectures used for general purposes. Its basic structure easily allowed the tuning of an enormous number of algorithms with different hyperparameters and characteristics. On the contrary, its elementary nature might be the cause of MLP's low performance compared to the other two types of architectures.

  Being MLP performance independent from the relative order of input data, Principal Component Analysis (PCA) was performed to account for only the most meaningful information, thus reducing the length of the input signals. The dataset transformation was expected to simplify the detection task for MLP algorithms, thus leading to better performance. The results showed that a general setting of PCA pre-processing was not able to significantly improve the accuracy of MLP algorithms; further tuning of PCA technique would lead to have a more complicated pre-processing stage, which is in conflict with the purpose of this research;

- Long Short-Term Memory (LSTM), a type of Recurrent Neural Network (RNN) that has shown very high performance for a large variety of applications. RNNs are famous for their ability to work with sequence-dependent inputs, thus also signals in time domain as those used in this thesis. Their performance showed to be higher than those of MLP architecture, even with a lower need of hyperparameters tuning. The weakness of LSTM algorithms (but in general of all RNNs) pertains to the enormous amount of computation time and resources needed to be trained, which is highly dependent on the length of the input signal and the number of hidden layers;
- WaveNet, a type of architecture based on Convolutional Neural Networks (CNNs). It was recently developed for audio generation and here adapted to suit the defect detection purpose of the thesis. WaveNet presents a more complex architecture with respect to those of MLP and LSTM algorithms in terms of number of hyperparameters to be tuned. Anyway, WaveNet showed the best performance in terms of computational speed, ease of hyperparameters tuning (more hyperparameters to be tuned but only few of them really affected the results) and, above all, accuracy of predictions, which was able to reach 100% values with several hyperparameter sets and for some levels of noise.

WaveNet architecture outperformed the others thus it was used for the following steps.
The robustness of WaveNet architecture was tested in temperature extrapolation field: the algorithms with the best set of hyperparameters have been tested with signals acquired at temperatures out of the range used to train the algorithms themselves. The accuracies obtained were still able to reach 100% for SNR 1 (i.e. the datasets to whom were added the noise with the lowest intensity among those identified in this thesis) for temperatures at least up to 30°C out of the range. This demonstrated the good linearization properties of WaveNet algorithm joined with BSS technique.
WaveNet was at last tested on a dataset collected during an actual experiment. The experimental setup was similar to the numerical one used throughout the whole study, apart from the material of the plate, made in carbon fiber reinforced plastic (CFRP), and the excitation frequency, which was significantly lower. WaveNet was proven to have great capabilities also in this experimental test, reaching accuracies up to 100% for some sets of hyperparameters.
Future works should focus on the creation of the experimental setup for the steel plate, in order to acquire real data to validate both the numerical model and the actual performance of all the ML architectures. Moreover, an ML-based measurement system should provide information also about position and entity of damages in order to be

effectively used in real structures for SHM purposes. Localisation and damage quantification tasks can be implemented in the already optimised WaveNet algorithms, in light of their good qualities previously discussed, and tested by exploiting the numerical dataset used in this thesis.

References

[1] J. B. Harley and J. M. F. Moura, “Scale Transform Signal Processing for Optimal Ultrasonic Temperature Compensation,” *IEEE transaction on ultrasonics, ferroelectrics, and frequency control,* vol. 59, no. 10, pp. 2226-2236, 2012.

[2] Y. Lu and J. E. Michaels, “A methodology for structural health monitoring with diffuse ultrasonic waves in the presence of temperature variations,” *Ultrasonics,* vol. 43, no. 9, pp. 717-731, 2005.

[3] G. Liu, Y. Xiao, H. Zhang and G. Ren, “Baseline signal reconstruction for temperature compensation in Lamb wave-based damage detection,” *Sensors,* vol. 16, no. 8, 2016.

[4] C. Sbarufatti, G. Manson and K. Worden, “A numerically-enhanced machine learning approach to damage diagnosis using a Lamb wave sensing network,” *Journal of Sound and Vibration,* vol. 333, pp. 4499-4525, 2014.

[5] C. Su, M. Jiang, S. Lv, S. Lu, L. Zhang, F. Zhang and Q. Sui, “Improved Damage Localization and Quantification of CFRP Using Lamb Waves and Convolution Neural Network,” *IEEE Sensors Journal,* vol. 19, no. 14, pp. 5784-5791, 2019.

[6] K. Worden, “Rayleigh and Lamb waves - Basic Principles,” *Strain,* vol. 37, no. 4, pp. 167-172, 2001.

[7] R. Sathya and A. Abraham, “Comparison of supervised and unsupervised learning algorithms for pattern classification,” *International Journal of Advanced Research in Artificial Intelligence,* vol. 2, no. 2, pp. 34-38, 2013.

[8] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Coputation,* vol. 9, no. 8, pp. 1735-1780, 1997.

[9] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior and K. Kavukcuoglu, “WaveNet: a generative model for raw audio,” Google DeepMind, London, 2016.

[10] NDT lab, Department of Mechanical Engineering, Imperial College London, [Online]. Available: https://www.imperial.ac.uk/non-destructive-evaluation/products-and-services/disperse/.

[11] P. Huthwaite, “Accelerated finite element elastodynamic simulations using the GPU,” *Journal of Computational Physics,* vol. 257, no. PA, pp. 687-707, 2014.

[12] S. Mariani, S. Heinlein and P. Cawley, “Compensation for temperature-dependent phase and velocity of guided wave signals in baseline subtraction for structural health monitoring,” *Structural Health Monitoring,* vol. 19, no. 1, pp. 26-47, 2020.

[13] D. Alleyne and P. Cawley, “A two-dimensional Fourier transform method for the measurement of propagating multimode signals,” *The Journal of the Acoustical Society of America,* vol. 89, no. 3, pp. 1159-1168, 1991.

[14] R. Courant, K. Friedrichs and H. Lewy, “On the Partial Difference Equations of Mathematical Physics,” *IBM Journal of Research and Development,* vol. 11, no. 2, pp. 215-234, 2010.

[15] “Keras,” [Online]. Available: https://keras.io/.

[16] D. P. Kingma and J. Lei Ba, “Adam: a method for stochastic optimization,” in *ICLR*, San Diego, 2015.

[17] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research,* vol. 15, pp. 1929-1958, 2014.

[18] E. Phaisangittisagul, “An analysis of the regularisation between L2 and Dropout in single hidden layer Neural Network,” in *7th International Conference on Intelligent Systems, Modelling and Simulation*, Bangkok, 2016.

[19] C. Zhang, S. Bengio, M. Hardt, B. Recht and O. Vinyals, “Understanding deep learning requires rethinking generalization,” 26 February 2017.

[20] J. Kukačka, V. Golkov and D. Cremers, “Regularization for deep learning: a taxonomy,” 2017.

[21] J. Moll, C. Kexel, S. Pötzsch, M. Rennoch and A. S. Herrmann, “Temperature affected guided wave propagation in a composite plate complementing the Open Guided Waves platform,” *Scientific data,* vol. 6, no. 191, 2019.

[22] J. Moll, J. Kathol, C.-P. Fritzen, M. Moix-Bonet, M. Rennoch, M. Koerdt, A. S. Herrmann, M. G. Sause and M. Bach, “Open Guided Waves: online platform for ultrasonic guided wave measurements,” *Structural Health Monitoring,* vol. 18, pp. 1903-1914, 2019.

Ringraziamenti

In questo capitolo finale voglio ringraziare tutte le persone che mi hanno sostenuto durante il periodo di ricerca e stesura della tesi, il vostro aiuto è stato fondamentale per portare a termine con successo un lavoro così importante. Mi riferisco in primis al Professor Claudio Sbarufatti per aver accolto la mia richiesta di intraprendere il lavoro di tesi all'estero, dandomi fiducia e responsabilità fin da subito; le sono grato per avermi dato tutto il supporto e gli stimoli necessari a raggiungere questo traguardo. Grazie ai ragazzi del laboratorio Non-Destructive Evaluation di Imperial College per avermi accolto e fatto sentir parte del loro gruppo di ricerca; grazie soprattutto al Dottor Stefano Mariani per avermi seguito in tutte le scelte intraprese in questi ultimi mesi, per avermi trasmesso la passione per la ricerca e per avermi sempre offerto nuovi spunti di riflessione. Grazie ai professori del Politecnico di Milano per le solide basi che mi hanno fornito con i loro insegnamenti, utili sia dal punto di vista professionale che personale.

La tesi rappresenta "solamente" l'ultimo gradino di una lunghissima rampa di scale, una di quelle che ad ogni passo ti mette davanti ad un nuovo problema che devi imparare a risolvere con i mezzi che hai a disposizione (hai presente le scale della stazione di Bovisa? Ecco, hai capito). Voglio quindi ringraziare le persone che hanno contribuito in tutti questi anni ad ampliare i mezzi a mia disposizione, rendendo questa scalata un pochino più facile.
Parto da chi c'è sempre stato fin dal primo giorno, da chi si è sempre preso cura di me, da chi non mi ha mai fatto mancare nulla: grazie infinite a mamma e papà per avermi fatto crescere sano e forte come un pesce, per aver sostenuto ogni mia scelta, per aver ripreso ogni mio errore, per aver sempre creduto in me; spero che il traguardo raggiunto oggi possa ripagare anche solo in parte tutte le energie che avete speso senza aver mai chiesto nulla in cambio. Grazie a mio fratello Leo per essere stato il mio primo amico e compagno di giochi, per essere tutt'ora un esempio da seguire, per tutte le volte che riesci a tirarmi su di morale con una battuta e per ricordarmi spesso (giustamente) che, in

fondo, ingegneria non è poi tutto! Grazie alla nonna per non avermi mai perso d'occhio un istante, per essere stata una seconda mamma, per tutti i minestroni di verdura che mi hai preparato (uno più buono dell'altro); ti ringrazio perché non vedi l'ora di festeggiare la laurea con me, ed io con te. Grazie infine a tutti gli zii, zie, cugini e cugine che fanno il tifo per me e coi quali passo sempre dei bei momenti; grazie in particolare a Giovanna, Luca e la loro fattoria degli animali, vi prometto che prima o poi un cane lo verremo a prendere.

Ringrazio ora tutte quelle amicizie che l'avventura a Milano mi ha riservato. Grazie a tutti i coinquilini che sono passati dalla casa di via Prestinari, in particolare Ettore, Alessandro e Giovanni, che ogni giorno dopo una lunga giornata di studio riuscivano sempre a colorare anche le giornate più grigie passate in Bovisa. Grazie a Pol, Alluca, Giombo, Sprenghy, Atthomas, Dodò il sommo, Vice il gasatore per aver condiviso insieme gioie e dolori (molto più frequenti) che la vita da studente di ingegneria comporta. Grazie a tutto il team Dynamis PRC, grazie a Danilo per essere stato il primo ad aver creduto in me, grazie a Mango, Shasa, Pablito, Piro, Bruk, Erika, Grem, Jacopo, Tommy, Olivs e tutti gli amici del reparto Telaio e Compositi coi quali ho condiviso quattro intensi anni di sforzi, passione, delusioni, soddisfazioni, sconfitte e vittorie. Xièxiè a Riccardo, Davide e Ettore che hanno reso l'Erasmus in Cina (esperienza che partiva già con altissime aspettative ancor prima di viverla) una di quelle avventure che quando ci ripensi non riesci a trovarci nemmeno un lato negativo.

Tornando alle mie radici venete ringrazio gli amici che mi sembra di conoscere da una vita intera, quelli coi quali mi sento libero di esprimere le mie emozioni ed esperienze, quelli sui quali so di poter contare anche in futuro. Per questo ringrazio Martina, Valeria, Sara, Chiara, Anna, Silvia, Lara, Enrico, Stefano, Danilo, Enrico, Stefano; un grazie particolare lo rivolgo a Davide nel quale ho trovato un amico sincero, che non ha timore nel dirmi la sua opinione anche se in contrasto con la mia, che per cinque lunghi anni non si è fatto problemi ad abusare delle mie cuffiette per ascoltare la musica durante il tragitto casa-scuola (solo per quello mi tenevi il posto, ammettilo…).
La scuola in questione è il famigerato ITIS Rossi di Vicenza, l'istituto superiore che ho frequentato e del quale vado molto fiero. Tra le sue mura ho conosciuto Nicola e Aronne, che ringrazio per essere stati miei compagni di dormite tra i banchi, di discussioni con la Gemma (quando andiamo a trovarla?) e di progetti di robot cingolati che a malapena si reggevano in piedi. Ringrazio ancora il Rossi per avermi dato la possibilità di prendere parte alla mia prima esperienza all'estero: grazie a Elisa, Andrea (Ska), Medi, Sofia, Greta e tutti i ragazzi "del Leonardo" coi quali ho condiviso quel mese speciale a Portsmouth, e la cui amicizia continua a preservarsi nel tempo.
Ringrazio il mio amico d'infanzia Matteo per essermi vicino da più di vent'anni: ne abbiamo fatta di strada dalla scuola materna ad oggi, a tratti ce la siamo dovuta cavare da soli, ma alla fine le nostre strade sono tornate a congiungersi. Ti ringrazio per le giornate spensierate in piscina, per i centri estivi del tennis, per le partite di ping pong

al parchetto. Assieme a te ringrazio Gianpietro, per tutte le stagioni passate insieme a giocare a calcio (per lo più in panchina date le mie scarse doti calcistiche), e a voi aggiungo e ringrazio anche Giulia ed Elena, per tutte le volte che ci siamo abbuffati di sushi, per le serate trascorse a scappare da stanze enigmatiche e per le giornate al mare passate a chiedere informazioni sul mio regalo di compleanno alla bagnina (quasi).

Voglio infine ringraziare il mio amato Derbi che da quasi dieci anni mi fa sentire libero quando lo guido tra le strade di montagna. Grazie a Stuart, che quando mi vede corre subito verso di me per mostrarmi tutto il suo affetto (la verità è che ha fame ma io non riesco a convincermene). Ringrazio il ragionevole numero di capelli che, con ammirevole coraggio, hanno scelto di accompagnarmi fino al giorno della laurea. Ringrazio la ragazza che ancora non ho conosciuto ma che sono sicuro darà una svolta alla mia vita (tranquilla nonna, prima o poi te la presento).

Ringrazio ognuno di voi per aver preso parte alla mia vita, per avermi spinto a dare il massimo, per aver sopportato il mio ennesimo “non posso, devo studiare”. A voi tutti posso finalmente dire: ragazzi, ce l’ho fatta!

Grazie di cuore,

Matteo