POLITECNICO DI TORINO

Master's Degree in Data Science and Engineering

Master's Degree Thesis

Patient simulation. Generation of a machine learning "inverse" digital twin.

Supervisors

Prof. Paolo GARZA

Prof. Erik FRANSÉN

Annaclaudia MONTANINO

Candidate

Paolo CALDERARO

December 2022

Summary

In the medtech industry models of the cardiiovascular systems and simulations are valuable tools for the development of new products ad therapies. The simulator Aplysia has been developed over several decade and is able to replicate a wide range of phenomena involved in the physiology and pathophysiology of breathing and circulation. Aplysia is also able to simulate the hemodynamics phenomena starting from a set of patient model parameters enhancing the idea of a "digital twin", i.e. a patient-specific representative simulation. Having a good starting estimate of the patient model parameters is a crucial aspect to start the simulation. A first estimate can be given by looking at patient monitoring data but medical expertise is required. The goal of this thesis is to address the parameter estimation task by developing machine learning and deep learning model to give an estimate of the patient model parameter starting from a set of time-varying data that we will refers as state variables. Those state variables are descriptive of a specific patient and for our project we will generate them through Aplysia starting from the simulation presets already available in the framework. Those presets simulates different physiologies, from healthy cases to different cardiovascular diseases. The thesis propose a comparison between a machine learning pipeline and more complex deep learning architecture to simultaneously predicting all the model parameters. This task is referred as Multi Target Regression (MTR) so the performances will be assessed in terms of MTR performance metrics. The results shows that a gradient boosting regressor with a regressorstacking approach achieve overall good performances, still it shows some lack of performances on some target model parameters. The deep learning architectures did not produced any valuable results because of the amount of our data: to deploy deep architectures such as ResNet or more complex Convolutional Neural Network (CNN) we need more simulations then the one that were done for this thesis work.

Table of Contents

List of Tables

List of Figures

Acronyms

AI
artificial intelligence

IQL
Independent Q-Learning

LAN
Local Area Network

Wi-Fi
Wireless Fidelity

MTR
Multi Target Regression

MTS
Multi-variate Time Series

EHR
Electronic Health Record

ECG
Electrocardiogram

DFT
Discrete Fourier Transform

DTW
Dynamic Time Warping

MLP

Multi-Layer Perceptron

SVM

Support Vector Machine

k-NN

K-Nearest Neighbours

PCA

Principal Components Analysis

LCA

Linear Components Analysis

CNN

Convolutional Neural Network

FCN

Fully Convolutional Network

MC-DCNN

Multi Channel - Deep CNN

EEG

Electroencephalogram

NLP

Natural Language Processing

MTF

Markov Transition Field

GAF

Gramian Angular Field

RNN

Recurrent Neural Network

LSTM

Long Short-Term Memory

TFT

Temporal Fusion Transformer

RC

Regressor Chain

MTRS

Multi-Target Regressor Stacking

ODE

Ordinary Differential Equations

WLAN

Wireless Local Area Network

UN

United Nations

SDG

Sustainable Development Goal

Chapter 1

Introduction

1.1 Context

1.1.1 Bio-medtech modelling and simulation

Modelling and simulation tools are indispensable during R&D in the bio-medtech field. A simulation model can be adapted to the properties of an individual patient in order to enhance personalized monitoring, diagnosis and prognosis. Those type of systems might in the future be used to taylor treatments, a need that has become apparent especially during the recent COVID-19 pandemic, in which the importance of intensive care heart or lungs support equipment has been highlighted even more. Furthermore, a fully functional simulator can be used by clinicians and students for training and gain knowledge of the pathophysiology of several diseases. For those reasons, several studies aim at reproducing cardiovascular systems and its dynamics [1]. In this field, the simulator Aplysia [2] has been developed over several decades. It can replicate a wide range of phenomena and processes involved in the physiology and pathophysiology of breathing and circulation. Aplysia is able to reproduce and simulate the hemodynamics phenomena starting from a set of patient model parameters and by exploiting a set of non-linear ODE that well describe the dynamics of our vascular and breathing system. These model parameters are the input to the simulation and there are key parameters in the equations that have a large impact on the simulation output. Since Aplysia is based on a set of ODE we will refer to the Aplysia output as state variables. Most of the state variables choosen for the analysis are time-varying signals such as the blood pressure, that changes periodically during a full heart cycle.

1.1.2 Machine learning for a digital twin

The vision of this project is to have a model capable of predicting the correct set of model parameters starting from time-varying simulation output. This would enhance the full capability of the Aplysia simulator and make the idea of the 'digital twin' closer to reality. By 'digital twin' we mean having a running simulation of a patient that emulates as close as possible that patient's physiology. The Aplysia simulator at the current stage can be only manually tuned by adjusting its model parameters to represent different patient conditions. The idea in this master thesis project is to start from Aplysia's time-varying simulator output, we can train a model that learn to return the correct model parameters to set into Aplysia in order to recreate the dynamics of a patient. Thus, by using a Machine Learning model the output state variables could then be used to automatically tune the patient model and make it patient-specific: such model can be trained on the time-varying simulator output and return a set of model parameters, as showed in picture 1.1. Once the trained model is deployed, we can use it to start patient-specific simulation and Aplysia could become part of the decision support steps, by testing drugs or medical procedures on the patient model in order to check the behaviour of the 'digital twin' before testing on the actual patient. A scheme of the proposed use case is shown in figure 1.2.

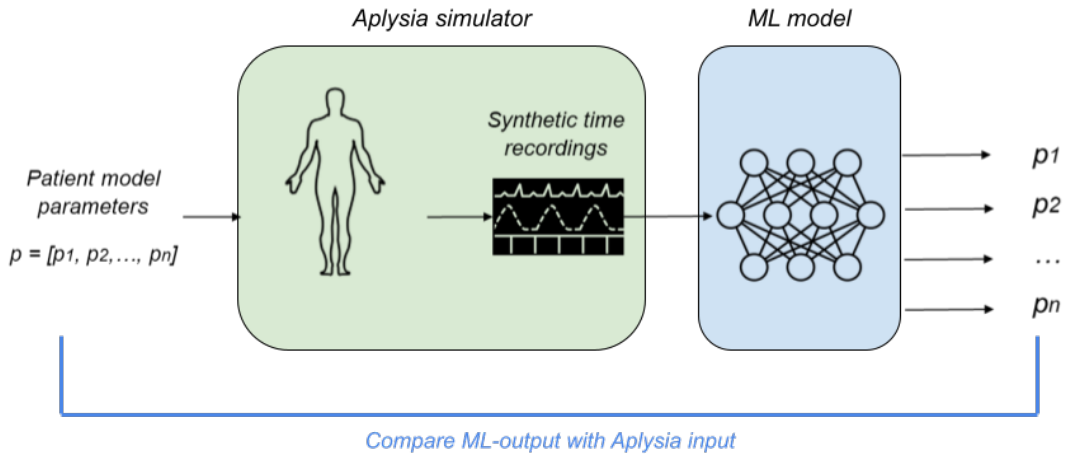


Figure 1.1: Scheme on how we plan to train our Machine Learning model. Starting from a given set of model parameters we are going to gather the time-varying simulation output. Those signals will then be fed into the model that will learn to give the model parameters as output.

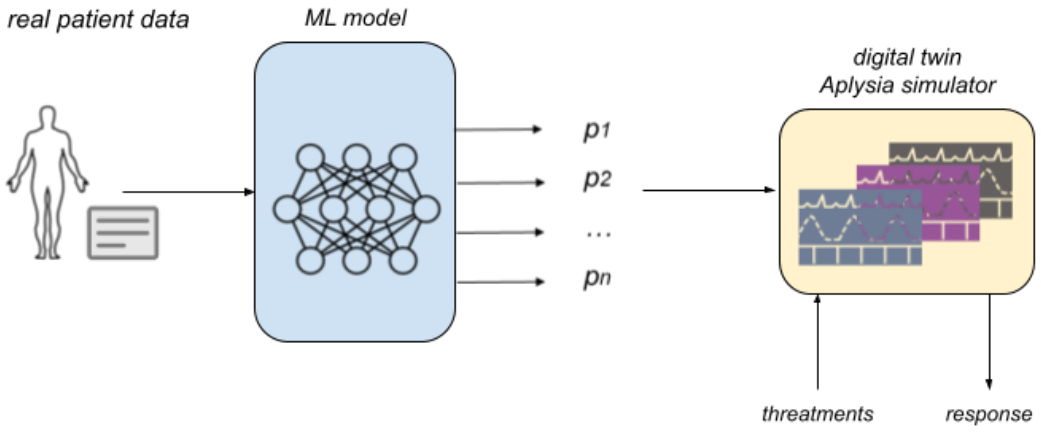


Figure 1.2: proposed deployment of the model. Using the existing data from a patient we can extract the set of model parameters necessary to drive Aplysia to make a patient-specific simulation (digital twin, in the figure)

1.2 Problem

1.2.1 Problem specification

The aim of this project is to develop a machine learning framework that is capable of analyzing the time series data that Aplysia generates in an efficient way. The final task is then to predict a set of real values that Aplysia uses as input in order to recreate the input time series data. The literature refers this problem as MTR, as opposed to standard regression problem in which we predict a single value. An high-level design of the project is showed in figure 1.1. The time-varying patient data that will be used to train the model will be generated by Aplysia itself, so this project includes also a dataset generation phase.

1.2.2 Challenges

Different challenges needs to be faced to perform this project:

- **Dataset generation**: A general knowledge and background of the physiology of the cardio-vascular system needs to be acquired to understand how different model parameters can influence each other in order to properly generate the dataset;
- **Heterogeneous data**: we are dealing with extremely heterogeneous data, since we need to handle both MTS data (such as systemic and pulmonary

arterial pressure or oxygen saturation) and static data (such as heart rate or blood volume);

- **Correlation among state variables**: several state variables may heavily interact with other and this high correlation could interfer with the learning process of our framework by making our models learning incorrect features. On the other hand, thanks to correlation among all the available state variables in the simulator we can believe that is possible to extract enough information to estimate the model parameters even if we limit the analysis on few time series;
- **Lack of literature**: there is a lack of literature regarding MTR task applied on MTS. We need to benefit from the literature on time series classification and forecasting in order to acquire a proper knowledge of the possible framework that can fit this task;
- **Computational resources**: deep architecture may require some time to train and test. This could become a bottleneck that need to be faced by taking into account solutions that are efficient and can be run on the hardware we have.

1.2.3 Research questions

The main questions we will address with this project will be:

- Which approach would provide the the best features from Aplysia output time series in terms of serving as input to a regressor?
- Which machine learning or deep learning framework would fit this task and provide the best estimates for the Aplysia model parameter values, as measured in terms of MTR scores [3]?
- How much robust is the choosen algorithm in terms of correct extimation of the model parameters? Robustness includes sensitivity to variations in the input from noise, small changes in expansion or compression of the time scale, variations in mean etc.?

1.3 Purpose

The main purpose of this project will be fulfilled if a working machine learning framework to analyze patient data will be developed. The work done could be useful for future studies in the field of medical simulation and hemodynamics related patient assistant. The reading of the final work can also be helpful for people

interested in handling multivariate time-series for generic regression problems: since most of the literature is focused on time series forecasting or classification, the analysis of this regression problem could further promote the use of time series data for different type of problems.
The project is developed in collaboration with Getinge AB, a Swedish healthcare company that offers products and solutions for intensive care, cardiovascular procedures, operating rooms and life science. The company will benefit from this work since it will be able to further exploit the simulator as a part of a decision support tool.

1.4 Structure of the thesis

Chapter 2 (Background) introduces the reader to notions of hemodynamics and the physiology of cardiovascular system in order to provide some medical context for the thesis, as well as introducing the paramters of our analysis. There will be described also machine learning and deep learning approaches and challenges from the literature analyzed. Chapter 3 (Method) starts with describing more in details the methods involved in the experiments, followed by a description on how we generated the dataset followed by some data analysis, details of the developed models, and the evaluation metrics. Chapter 4 (Results) presents the results of the experiments while Chapter 5 (Discussion) contains a discussion of the results, suggestions of future work, and a discourse on sustainability. Chapter 6 (Conclusions) summarizes the main findings of this thesis.

Chapter 2

Background

2.1 Hemodynamics

In this section, a brief overview of hemodynamics, physiology of cardiovascular system and nature of monitoring signal is given. By hemodynamics we mean a branch of physiology that studies the forces and mechanism involved in circulation. The blood flows between different components through different types of blood vessels: arteries, veins, capillaries. The aim of this chapter is to provide to the reader with some medical context for the thesis, as well as introduce the relevant parameters on which our analysis will focus.

2.1.1 Cardiovascular system

The cardiovascular system is responsible for delivering blood to the whole body [4]. It is composed by the heart, arteries, veins and capillaries. Veins, arteries and capillares compose the human vasculature.
The human vasculature is divided in two distinct circuits: the pulmonary and the systemic circulation.

- **pulmonary circulation**: transport of blood between the heart and the lungs. It transports deoxygenated blood from the right heart to the lungs to absorb oxygen through the pulmonary arteries. The oxygenated blood then returns back to the left side of the heart through the pulmonary veins.
- **systemic circulation**: transport of blood between the heart and the rest of the body. It sends oxygenated blood out to cells and returns deoxygenated blood to the heart.

Two important quantities in hemodynamics are pressure and flow. Pressure is the force applied per unit area. Usually in hemodynamics we think of that in terms

of a pressure difference since is the gradient itself that causes the flow of blood. Flow is usually measured in liters/minute. By monitoring those two quantities in different point we can derive properties of the arterial system and the hearth.
The heart consists of four different chambers: two atria and two ventricles with valves between them. Aspects such as heart contractility, the mechanical description of valve opening as well as mechanical properties of the vasculature determine blood flow and pressure throughout the system. In particular, the equation for heart contractility describes the heart cycle (or cardiac cycle) and determines the heart rate. In an healthy patient a full cardiac cycle takes around 0.8 seconds. This information is usually carried through the heart rate: we would rather hear "heart rate of 75 BPM (Beat Per Minute)" instead of "0.8 seconds cardiac cycle".

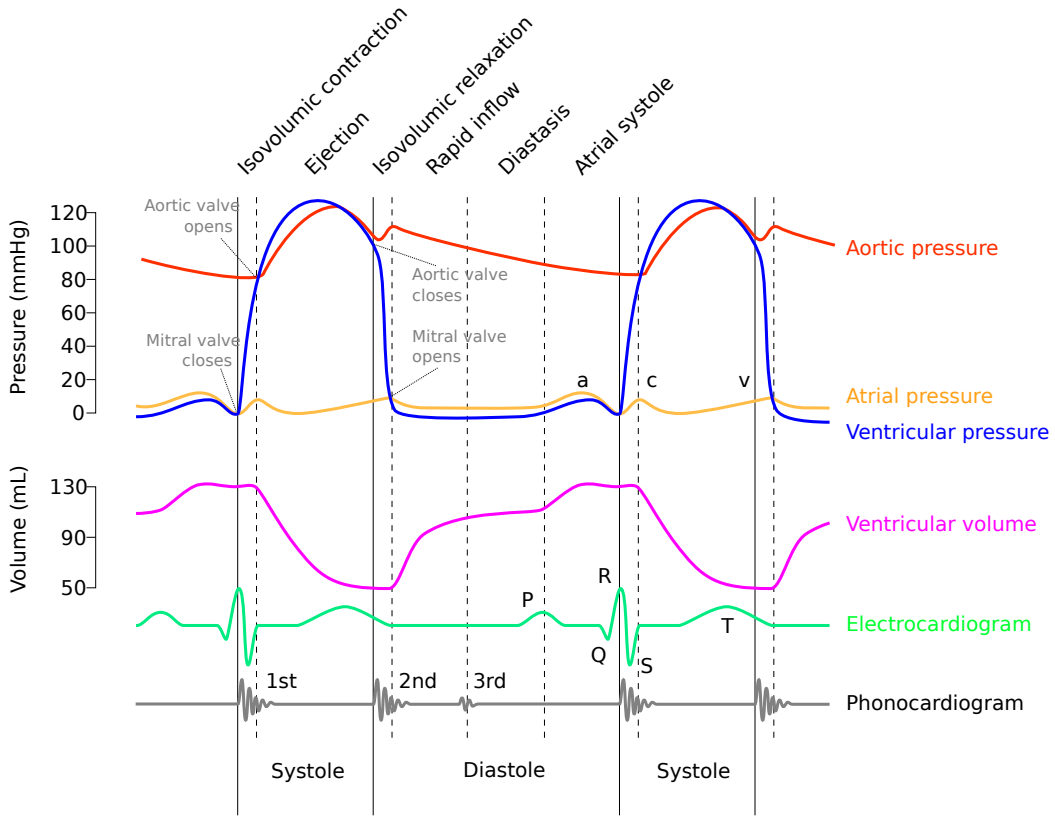


Figure 2.1: Wiggers diagram of a full hearth cycle.

The cardiac cycle consists of two main phases: systole and diastole. Diastole is the phase in which the heart muscle relaxes and the chambers fill with blood

coming from the pulmonary and the systemic veins. Once the ventricles are filled, the ventricles start to contract and systole begins. As a result, when the pressure in the ventricles exceeds the pressure in the arteries, blood is ejected from the right and left ventricle into the pulmonary and aortic arteries respectively. In figure 2.1 we can appreciate how pressure and volume vary over an entire heart cycle. The plot, called Wiggers diagram, describes how those two forces vary over the cycle in different ways for each part of the heart.

2.1.2 Healthcare-aided simulator: Aplysia

Due to the complexity of the cardiovascular system, simulators are becoming more and more popular since they help to get an overall understanding of different pathophysiological conditions. Simulators can also be integrated as part of bedside decision tool [1]. When this happens, the model is connected to a patient's data in order to reproduce the corresponding hemodynamic condition. Then, one or more clinical interventions can be simulated on the model to predict patient's response and elaborate possible improvements on the treatments. Aplysia CardioVascular Lab is a flexible simulator that can be used as a user-friendly teaching tool, but it also allows to control every parameter in detail, making it possible to study the interaction between many combination of vascular or cardiac disease or physiological mechanism in different set of patients. The input model parameters that can be modified in each simulation are more then 200. We recall that the final goal of this project is to predict a set of model parameters starting from the time-varying monitoring data: a one-to-one relation between the available monitoring data and all 200 parameters is challenging. Hence, it is necessary to identify a subset of parameters that are realistic and relevant to extract: we aim at simulating as close as possible the patient physiology. Therefore, a set with the most important model parameters to identify was choosen for this thesis. A brief description of those is available in table 2.1.

2.2 Time series analysis

In this section we are going to define time-series data in a more analytical way by introducing definitions that will ease the understanding of the following work. As we are dealing with medical monitoring data we focus on biomedical time series data literature. Several dataset are available to the research community for this field: most of them focus on ECG data or EHR, such as ECG200 or MIMIC [5], which are often used as benchmark for the new algorithms proposed. Even if those data are often used, the amount of information that we can extract from those is not relevant for our scope because it is either un-related to Aplysia monitoring output (such as information in EHR) or too coarse for our project (such as the info

Name (target variables)	Unit	Typical values / range	Description
Heart rate	bpm	70 / (40-140)	Heart beat per minutes.
Blood volume	mL	5500 / (3500-7500)	Total amount of blood flowing in the body.
Left ventricular contractility	mmHg/mL	2.8 / (0.5 - 5.0)	peak value of the time-varying elastance function.
Right ventricular contractility	mmHg/mL	0.6 / (0.2 - 5.0)	See left ventricular contractility.
Left ventricular stiffness	mmHg/mL	0.04 / (0.01 - 0.10)	Basal level of elastance in the time-varying elastance function.
Right ventricular stiffness	mmHg/mL	0.03 / (0.01 - 0.10)	See left ventricular stiffness.
Systemic Vascular Resistance	mmHg $\cdot$ s/mL	1.0 / (0.1 - 5.0)	Amount of force exerted on circulating blood by the vasculature of the body.
Pulmonary Vascular Resistance	mmHg $\cdot$ s/mL	0.1 / (0.05 - 2.00)	Resistance against blood flow from the pulmonary artery to the left atrium.
Systemic Arterial Stiffness	mmHg/mL	0.4 / (0.1 - 5.0)	Measure of aging of the arteries. Determined by both geometry and tissue properties.
Pulmonary Arterial Stiffness	mmHg/mL	0.7 / (0.2 - 2.0)	See Systemic Arterial Stiffness.
Hemoglobin	g/L	140 / (40 - 200)	Oxygen carrying capacity of the blood.
Total oxygen consumption	mL/min	250 / (100 - 500)	Indicator of metabolic activity.
Pulmonary shunt	%	5 / (0 - 100)	Percentage of blood flow in the pulmonary circulation not being oxygenated in the lung before entering the left side of the heart

Table 2.1: subset of model parameters choosen for the analysis. Those will be the output to predict in our task. In the third column from left there are typical values from healthy patients together with humanly possible ranges for each parameter.

in MIMIC). Those are still interesting since they come from real patients and not from simulations, as in our case.
As the name suggest, time series is a set of ordered data. This means that every problem related to data that is registered taking into account some notion of ordering can be casted as a time series problem. This aspect makes time series data valuable and interesting in multiple fields. According to [6] during the last years there have been a large number of new time series-related algorithm proposed

in the literature. Most of the work is focused on time series classification and time series forecasting. As we already mentioned in the introduction, since we have a lack of literature regarding MTR applied on MTS we will mention all possible approaches tailored for classification and forecasting that can also be useful for our MTR task.

2.2.1 Univariate and multivariate time series

We define a univariate time series as:

$$X = [x_1, x_2, ..., x_T]$$

where $x_1, x_2, ..., x_T$ is an ordered set of real values. The length of X is equal to T, which is the number of element that compose the time series.
We define an N-dimensional MTS as:

$$X = [X^1, X^2, ..., X^N]$$

where $X^1, X^2, ..., X^N$ is a set of N univariate time series of length T. With Aplysia we can generate both univariate and MTS time series depending on how many parameters we want to monitor at the same time. For our task we will generate and exploit MTS.

2.2.2 Time series forecasting and classification

With time series forecasting we mean the process of making prediction on the next values of the time series based on the available historical data. Forecasting has a wide range of applications in different industries [7]: weather forecasting, economic forecasting, finance forecasting and healthcare forecasting. In many particular circumstances we talk about multi-horizon forecasting which stands for forecasting for multiple steps in the future. Multi-horizon forecasting has gained more and more attention in recent literature [8] [9] since it provides support for decision making over mid to long periods of time.
In time series classification, we aim at predicting a class label associated to time-series by extracting meaningful features from the time domain data. There are mainly two main approaches to time series classification [10]:

- **feature-based**: we first extract time series features such as statistical moments, maximum and minimum, both from the time domain and frequency domain through DFT;

- **distance-based**: we implement a a definition of suitable distance between time series, such as DTW [11] or euclidean distance and we combine it with a k-NN algorithm [6].

We are not going deeper in the analysis of distance-based approaches because they can not be useful for a regression task. We can infer useful insights from the feature-based approaches. The most important step in those approaches is the features extraction and selection step: the feature representation of the input time series is the main discriminative aspect for the final performances. Given that, the idea of the following study is that we can benefit from previous works on feature representation since we can always deploy a MTR model on top of those features and switch the task from a classification to a MTR task.
In [12] the authors explore three different ways to obtain a structured representation of time-series, exploiting different type of metrics ranging from statistical metrics and complexity measures to temporal patterns. Experiments were performed on five medical time series dataset (mainly ECG data) and good classification results were obtained by combining all three types of metrics and deploying a naive MLP on top of the features representation. Also in [13] a MTS classification task is performed on ECG data by deploying a simple SVM classifier on the data after extracting intra-temporal patterns within each component and inter-temporal patterns among different components of the MTS. Those first approaches shows that the feature representation step is crucial when dealing with time series and great results can be achieved even by naive models. Despite the results, those methods still has some drawbacks. In those approaches, time series are required to be equally spaced in time, resulting in not being able of handling MTS of different length. Also, important correlation information among different channel could be lost during the feature extraction procedure and, furthermore, redundant information could be extracted. A solution to this last problem could be to apply PCA or LCA, but still this leads to additional overhead to the computational time. Also features extraction and signal processing techniques may require expertise knowledge on the data. Despite their drawbacks those approach still represent a good baseline to perform our task.

2.2.3 Deep learning for time series

Aside from hand-crafted features techniques, with the advent of Deep Learning new methods and models to automatically infer features from time series were proposed. Deep Learning has a lot to contribute to the bio-medtech field. A promising aspect of deep learning approaches is that we can customize the architecture in order to full-fill our needs. For this project, it is particularly useful since we can make deep architecture output a vector of real values by properly acting on the layers.

As proposed in [14] we can separate deep learning approaches for time series in two branches:

- **generative approaches**: models use an unsupervised training step that precedes the classification phase. Often reffered as model-based;
- **discriminative approaches**: models directly learns the mapping between the time series and the desired output, without any pre-training for features extraction.

By "generative" we mean that we generate a set of features on top of the raw time series data, as opposed to "discriminative" which means that our model classify time series without the needs of any elaborate preprocessing step. Adopting a generative approach lead to multiple advantages. First of all, it reduces the need for expertise of the data since we leave to the architecture the features learning step. Experiments also showed that these models are able to capture all the salient trends and characteristics of a time series. Although, the general consensus in the literature is that generative models are usually less precise than discriminative. Implementing and training generative models is also harder since we need a first learning stage to learn the embeddings and a second in which we train the classifier. The choice of the type of classifier is not trivial also considering that the final accuracy mainly depends on the classifier choosen (which often is not even a neural network [15]).
Discriminative approaches are then preferred, also because there is a wide range of architecture that covers time series problem and they allow enable the network to learn the most discriminant features by not incorporating any domain knowledge. For our goal, both discriminative and generative approaches could be useful.

2.3 MTR: Multi-Target Regression

The aim of this section is to give an overview of the existing ways of tackling MTR problem from the literature. The goal is to identify the common approaches and main challenges that may arise when dealing with this task. MTR is also known in the literature as multi-response, multi-variate or multi-output regression. As already stated in previous sections, this task refers to models that are able to predict simultaneously multiple continuous variables at the same time. When the numerical values to predict are binary we talk about multi-label classification task.

2.3.1 Approaches

According to [16], we can categorize existing approaches for MTR as:

- **problem transformation methods**: also known as local methods, these methods transform the multi-output problem to several single-target problem, then one model for each sub-problem is developed;
- **algorithm adaptation methods**: also known as global methods, these methods predict all the target using a single model.

With respect to problem transformation methods, algorithm adaptations simultaneously predict all the target using a single model. Those approaches lead to better performance especially when targets are correlated. By properly adapting the iterative weight learning procedure SVM [17] were generalized and introduced to MTR. It is important to notice that in algorithm adaptation methods the algorithm itself get modified while in problem transformation we rely on already existing model: multi-output SVM can be obtained also through a problem transformation approach. Regression trees can also be used for multi-target problems [18]. Those algorithms in their multi-variate version has two major benefits: a single multi-target tree is much smaller then the total size of one single-target tree for each target; the performance can be improved by using ensemble learning techniques such as random forest or bagging of multi-target regression trees. We can categorize as algorithm adaptation also neural network approaches since the network architecture learn how to predict the entire set of target at once. In this set of algorithms, further exploitation of the learning procedure can be used to boost the performance even more [19].

2.4 Related work

In the following section we present works in the literature that deals with related problem. As already stated there is no similar task to this in the literature but we can still infer useful information from literature on medical-related tasks.

2.4.1 Inverse modelling through optimization

The aim of this thesis is to retrieve unobservable information, the model parameters, given the observable state variables generated by Aplysia in form of time-series signals. In modelling this problem is referred as inverse modelling and can be done also through optimization algorithms that aim at minimizing the error between the target data and the model output. This approach has been used in biochemical models [20] since mathematical modelling of biological processes is effective in different applications and while some model parameter can be obtained from the literature other can be inferred from measured observations. One of the main problem about tackling this task with optimization algorithms is the risk of

formulating non-linear problem that could be non-convex. This mean that most of the optimization algorithm risk of finding sub-optimal solution and not the optimal. This problem even affects neural networks approaches [21] since the training process can be seen as an optimization problem solved with a gradient-based algorithms. To overcome this problem, different approaches can be found in the optimization literature. We can classify the different algorithms in local, global and hybrid algorithm. Local methods usually requires less to converge to a solution but this often lead to find sub-optimal. Global algorithm aim at finding the global solution having the drawback of being computationally heavy. In [22] a genetic algorithm is deployed to identify parameters related to a cardiac model. The focus of the study is to analyze cardiac wall stress rather then particular hearth disease but the analysis is interesting because deals with model parameters that are very close to ours. The results proved the feasibility of using a global optimization strategy to optimize the model parameters in analysis. Hybrid approaches combine the two previous approaches by combining an exploratory phase with an intensification phase. Different hybrid algorithms have been proven effective for parameter estimation in system biology [23] [24].

2.4.2 RNN-based architectures

The RNN are artificial neural networks in which the connections are not feed-forward only: connections between RNN units form directed cycles, providing an implicit internal memory. Because of their nature and their capability of storing values in memory those network can well be adapted to problems dealing with signals evolving through time [25]. In the field of medical time series analysis LSTM units are often used. Those LSTM cells are able to solve some problem and limitations that RNN has with gradient-based algorithm during the training phase [26]. Application of LSTM models aside with classical features for ECG classification showed that RNN capture temporal dependencies in sequential data more efficiently compared to other types of networks [27]. LSTM cells can be also used to build autoencoder structure. In those approach [28] one LSTM layer runs as an encoder and another runs as a decoder: the encoder layer takes the input sequence and encodes it into a learned representation vector. Then, the decoder layer takes that vector as input and tries to reconstruct the input sequence and forecast. This approach also propose an unsupervised pre-training phase in order to avoid random initialization. Is worth mentioning that this model handles well MTS. LSTM auto-encoder can also be used as generative architecture prior to learning the classifier [15]. The LSTM auto-encoder is used to generate syntetic additional channel to the time series data in order to improve the learning phase. This particular data augmentation technique helps when we have limited channel data, which may happens in several scenario in healthcare (e.g. limited sensors, fault

during sensing, invasiveness etc.). For instance, an ECG is natively recorded on twelve channel, but if some errors or fault occurs only few of them may be available for the analysis. This approach aims at reconstructing the original twelve channel signals by generating the missing ones. Despite the great performance in capturing long term dependencies in time signal, deep LSTM-based architecture requires a lot of computational resources to train. To limit that several hybrid architecture are proposed in the literature. PP-Net [29] combine LSTM units and CNN to predict cardiovascular disease starting from PPG (photoplethysmogram) and ECG data. This architecture deals with a MTR task since it needs to predict diastolic blood pressure, systolic blood pressure and heart rate, which are the three main indicator of cardiovascular disease in a patient. The model stacks CNN as feature extractor with two LSTMs layer followed by one last dense layer. Results shows that the joint framework leverages the advantages of both network providing an efficient and light-weight model. Another promising hybrid architecture recently proposed is the TFT [9]. No application of this architecture on medical data can be found in the literature yet but this model stands as top-performer in multi-horizon forecasting by also accounting model explainability. Here LSTM units are combined with more complex variable selection network and gating mechanisms. This architecture aims at tackling the heterogeneity of MTS input while keeping explainability of the model. Even if the TFT is born to perform multi-horizon forecasting it can be adapted to different type of task and input data due to its flexibility: we can feed to the architecture static data, time-varying signal or a-priori known future inputs.

2.4.3 Time series to 2D representations

Given the recent successes of deep learning in computer vision, several studies proposed of reformulating features of time series as visual clues. Time series imaging consist in transforming the problem completely into the computer-vision domain by turning the time signal into images. Interesting application of this method can be found on time series forecasting problems [30] and classification [31]: time series data are first transformed into images using recurrency plots and then a feature extraction methods is developed upon the processed data. Results vary depending on which type of plots we decide to use to encode time series data into images. Several studies focus mainly on the encoding map and solutions range from MTF to more sofisticated encoding such as GAF.
Some approach that relies on medical imaging also address privacy concerns [32]. The development of cloud computing has increasingly allowed hospitals to offload expensive computation tasks and deploy algorithms on cloud servers, reducing the overall costs of local servers, but this aspect introduces several privacy concerns since we need to preserve privacy. In the pipeline proposed convolutional layers are combined with LSTM and Homomorphic Encryption,

a type of encryption algorithm tailored for privacy preservation. The resulting HE-CLSTM (Homomorphic Encryption-Convolutional LSTM) architecture is then able to extract features from the input images and perform classification in a fully encrypted way.
Having images data also enable the use of transfer learning [33]. Transfer learning has been widely used in fields such image classification and NLP and it consists in using publicly available models already trained on large dataset: we can re-use the pre-trained model by fixing the weights of all the layers except for the last fully connected layer and fine-tune the model using the task data. The main advantage we obtain by transferring the already trained parameters is that we reduce the training time since we just fine-tune the last layers of the architecture to learn the new task. We also gain in performance since pre-trained architecture already knows how to extract and recognize high-level features from images (such as edges, curves, lines etc.) .

Chapter 3

Methods

3.1 Approaches for MTR

As explained in 2.3.1 we have different approaches to tackle MTR problems. The baseline approach of developing one model for each target (referred as Multi Output regressor) does not take into account dependencies among the multiple features and the target variables: the target are predicted independently and this may affect the overall quality of the predictions since the relationship among the target are ignored. To limit this drawback two approaches were proposed [34]: MTRS and RC. In MTRS we have a two-stage training phase: we first learn n single-target model (one for each target output) and then a second set of n model is learned. In the second stage we augment the input using predictions coming from all the first stage models: the idea behind is that a second-stage trained model is able to correct its prediction by considering information from all the target variable coming from first-stage prediction. So if we want to make predictions for a new instance, we first need to generate the estimates from the first-stage models and then we apply the second stage models on the augmented output to produce the final estimate.
RC is inspired from the multi-label chain classifiers. The idea is to select a random permutation of the set of the target variables and build a regression model for each one following the order of the series. Each model will use as input also the output of the previous regressors in the chain. One remarkable issue of using RC is that results are sensitive to regressor chain ordering: to limit that dependencies a set of regressor chain can be used and the results are then averaged. This approach is called ensemble of regressor chain.

3.2 CNN

CNN are common for image processing tasks. Those architecture are characterized by convolutional layer (or filters) that performs convolution by shifting and sliding over the input. This reduces the number of parameter to be learned since we train to compute the weights of the filters. Every time the input goes through a convolutional layer we obtain a new representation of it. Usually convolutional layers are paired with pooling layer which downsamples the input through the different convolutions. We talk of FCN when the networks does not contain any pooling layer, which results in keeping the length of the input unchanged throughout the convolutions.
Even if those architectures are widely famous for computer vision task they have recently been applied also on time series. In [35] a first framework to analyze MTS through CNN is proposed. This network is often referred as MC-DCNN. The architecture has a generative approach: in a first stage the multivariate time series is broken into univariate ones and feature learning through convolution is performed on each univariate series individually. At the end of all the convolution layers the output of all channels is flattened and then a standard MLP layer performs the classification. Even if the architecture is composed of just two convolutional layers followed by pooling layers, experiments conducted on ECG data showed better results than the state of the art 1-NN + DTW. In [36] a similar approach is proposed but an additional aggregating CNN layer is placed before the final MLP layer. This additional layer is capable of extracting channel-wise information from the processed MTS.

3.3 Residual Network

ResNet [37] extends the neural networks to very deep structures thanks to the introduction of shortcut connections. These allow the most common gradient-based algorithm to converge faster, hence deeper architecture could be developed by still keeping reasonable training time. ResNet architecture represented a turning point for the deep learning community, especially in the computer vision field. It achieves the state-of-the-art performance in object detection and other vision related tasks and interesting applications on time series data can be found in the literature [38]. The literature work on this architecture make possible to easily pre-train a model on a source dataset, then transfer and fine-tune it on our target dataset. Since ResNet architecture are usually deep networks we need to be aware of not overfitting the data: the ResNet overfits the training data much easier if the datasets is comparatively small [39]. In figure 3.1 is represented a scheme of a ResNet applied on MTS.

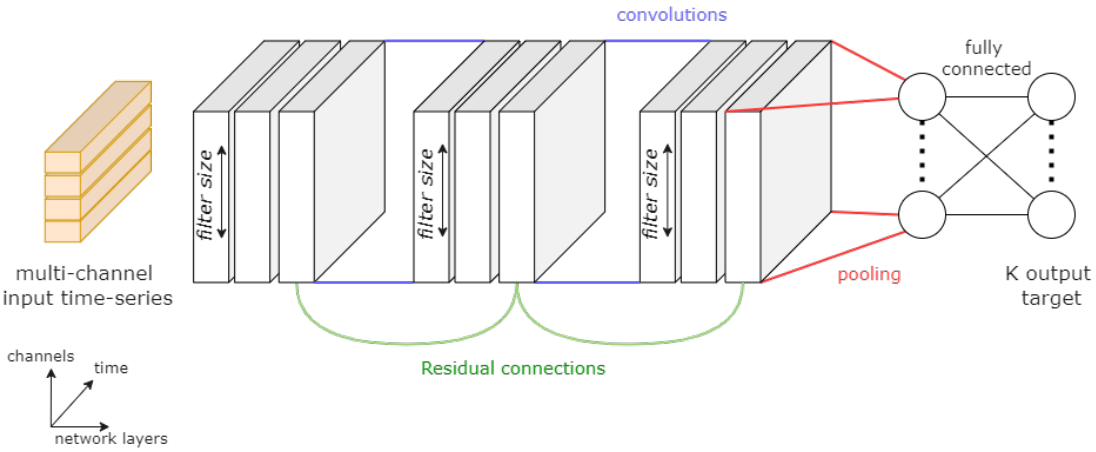


Figure 3.1: Scheme of ResNet architecture. Input is a multivariate time-series and output is a set of target values.

3.4 Evaluation metrics

To assess the behaviour and performance of a MTR model several evaluation metrics are available. Let be T target variable, n_{test} number of unseen data, y and $\hat{y}$ respectively the ground truth and the prediction for the input x, $\bar{y}$ the vector of averages of the actual output. The mostly used evaluation metrics in the literature are:

- **MSE: Mean Squared Error**. This metrics is often used also in single-target regression problem. In the multi-target version we subtract for each target attribute the ground truth and its prediction and we then square it. The final metrics will be the sum over all target attributes scaled by the number of test sample, as in the following:

$$MSE = \sum_{i=1}^{T} \frac{1}{n_{test}} \sum_{j=1}^{n_{test}} (y_{i,j} - \hat{y_{i,j}})^2$$

- **aRMSE: Average Root Mean Squared Error**. In this metric we extract the root mean squared error for each target and we average it through the number of training sample and then with respect to the number of target variables :

$$aRMSE = \frac{1}{T} \sum_{i=1}^{T} \sqrt{\frac{\sum_{j=1}^{n_{test}} (y_{i,j} - \hat{y_{i,j}})^2}{n_{test}}}$$

- **aRRMSE: Average Relative Root Mean Squared Error**. Also known as Normalized Root Mean Square Error (NRMSE), this measure. It is composed

of a Root Mean Squared error normalized by the squared error of a simple predictor (i.e. a model that predicts every data point with the average values of the target). The result is then averaged across each target variable:

$$aRRMSE = \frac{1}{T}\sum_{i=1}^{T}\sqrt{\frac{\sum_{j=1}^{n_{test}}(y_{i,j}-\hat{y_{i,j}})^2}{\sum_{j=1}^{n_{test}}(y_{i,j}-\bar{y_i})^2}}$$

Aside from average metrics is also useful to check the performance for each one of the target. In this way we can get further insight on how our model is performing overall.

3.4.1 Problems and challenges

One first challenge that needs to be handled is data normalization. The choice mainly depends on the evaluation metrics: when the metrics is averaged through the different targets (as in MSE or aRMSE) a scaling factor is necessary in order to have comparable contribution to the error from all the targets. The re-scaling factor could be either determined by previous knowledge or several trials: a good starting point would be to normalize each target with its standard deviation and see the effects on the overall performance.
Relative metrics (such as aRRMSE) automatically re-scale the contribution of each target with its average, hence no additional scaling factor is needed.
Computational complexity represents another challenge: since we have more then 10 target attributes we need to train the same number of models in problem transformation approaches.
Another problem to deal with is related to non-identifiability of the target parameters [23]. Non-identifiability means that a unique solution to the parameter estimation problem does not exist: there may be several sets of parameter values that fit the data equally well. A possible solution refers to data-driven normalization. A data-driven normalization approach consist in finding a sample that will be used as normalization factor. The choice of such sample influences the identifiability so it is necessary to spot a sample which well represent the entire sample population, which is often not trivial.

3.5 Correlation analysis for time series

A correlation matrix is a table that displays correlation coefficients for different variables. It is a common and powerful tool to summarize a dataset and to visualize patterns in given data. This tool may be particularly useful in our task since we can discover linear relationship between variables which may be interesting to

understand the results. Different type of correlation can be computed when dealing with time series [40]. For this task we decided to compute the Pearson correlation coefficient which is among the simplest and most used indicator of correlation. This index describes the linear relationship between two time series as a number between -1 (negatively correlated) to 0 (not correlated) to 1 (perfectly correlated). We can picture this index as a measure of global synchrony between two time series: if both time series has an overall increasing trend the correlation coefficient will be a value near 1. We need to remark that this is a global coefficient so it does not describes local synchrony of the signals. A way to compute local synchrony would be to measure the Pearson correlation in a small portion of the signals and then repeat the process along a rolling window until the entire signals are analyzed. An example can be seen in figure 3.2.

3.6 Data generation

The experiments will be performed on syntetic data generated by Aplysia. The following section gives a detailed explanation of the different stages involved to generate the final dataset. Compared to other machine learning task we were not provided with a dataset. The way in which we generate the data is then crucial: we need a strict methodology in order to get valuable results that could ideally be used to further deploy the model on real and non-syntetic data.

3.6.1 Input of the regressor models

We already discussed which will be the output model parameters (a brief recap is available in table 2.1).
It has been important to identify a relevant subset of state variables to sample from Aplysia. Those state variables will be the time-varying data that we will feed to our models, hence they should have the following characteristics:

- **clinically accessible**: ideally the final models will be applied on real patient data so we need to work with parameters that are easy to gather from a patient and that do not necessarily require invasive procedures.
- **clinically relevant**: the data that we gather needs to be clinically significant, which means that we need to choose a relevant subset of parameters from which it can be possible to predict the target.

The output model parameters should be ”composited” of more then one state variables from the input subset. Ideally the higher the number of state variables the easier it would be to predict the model parameters. This would mean that we

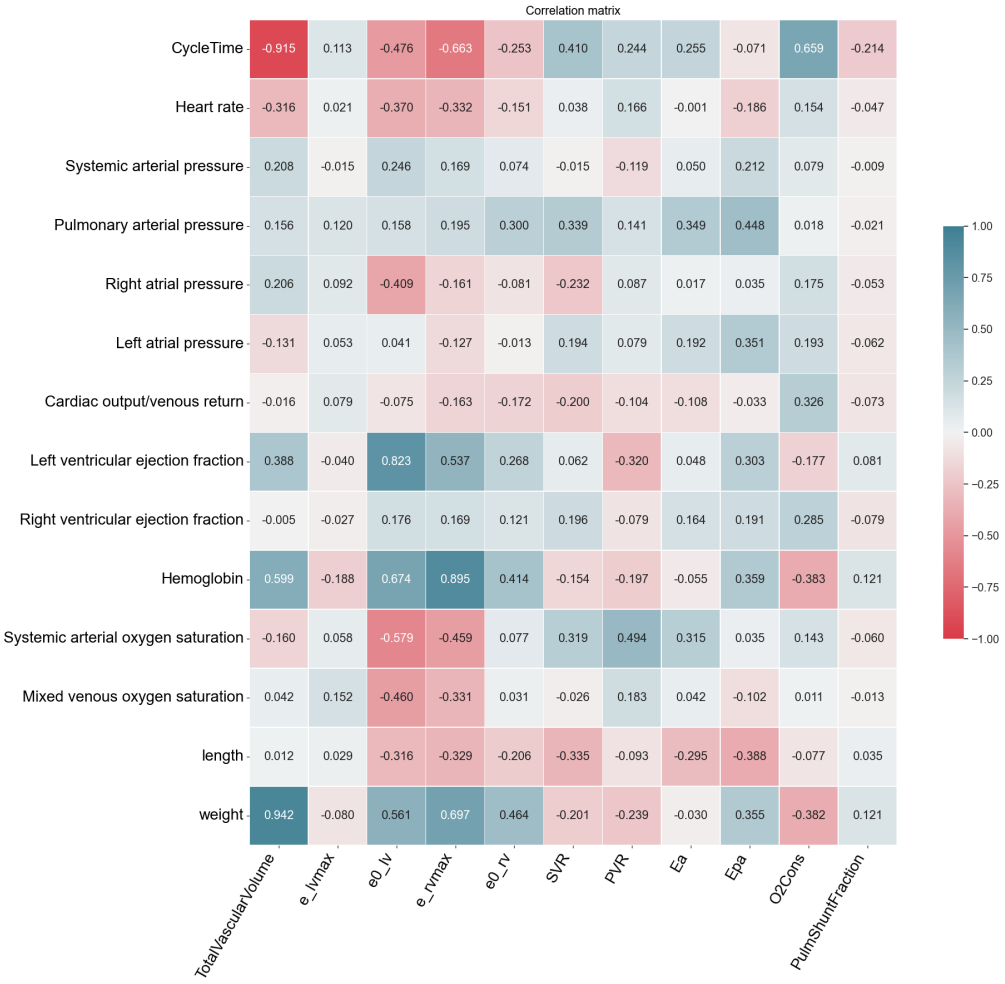


Figure 3.2: Pearson correlation between state variables (x-axis) and model parameters (y-axis).

should take into analysis all the state variables vailable from Aplysia. However not all those variables are available clinically, therefore we need a good subset that satisfies the aforementioned criteria in order to predict correctly the output. The final choice of the state variables that will be fed in the machine learning models is available in table 3.1.

Name (target variables)	Unit	Typical values	Description
Heart rate	bpm	70	Heart beat per minutes.
Systemic arterial pressure (mean)	mmHg	90	Mean pressure of the circulating blood in the vessels during an entire cardiac cycle.
Pulmonary arterial pressure (mean)	mmHg	15	Mean pressure of the circulating blood in the pulmonary artery during an entire cardiac cycle.
Right atrial pressure (mean)	mmHg	5	Mean blood pressure in the right atrium of the heart during an entire heart cycle.
Left atrial pressure (mean)	mmHg	8	Mean blood pressure in the left atrium of the heart during an entire heart cycle.
Cardiac output	L/min	5	Amount of blood that the heart pumps for each minute.
Left ventricular ejection fraction	%	65	Percentage of blood leaving the left heart chamber after a contraction.
Right ventricular ejection fraction	%	65	Percentage of blood leaving the right heart chamber after a contraction.
Hemoglobin	g/L	140	Oxygen carrying capacity of the blood.
Arterial oxygen saturation	%	95	Measure of blood oxygenation in the arterial compartment of the circulating system. Strictly related to the hemoglobin level.
Mixed venous oxygen saturation	%	95	Fraction of oxygen coming back to the right side of the heart. Strictly related to the hemoglobin level.

Table 3.1: subset of state variables gathered from Aplysia. Those will be the actual data that we will use to predict the target model parameters. The "typical values" columns shows some common values coming from healthy adult physiology.

3.6.2 Models of cardiovascular physiology and pathophysiology

Several considerations needed to be made in order to create our dataset. Aplysia can in fact generate data starting from different presets of cardiovascular model

(from pediatric to adult) and pathological state. Depending on that the behaviours of clinical parameters can significantly change. As an example, we can think of the difference between a child and an adult in terms of heart rate. Children's heart rate is around 140bpm for newborns against the 70bpm for a standard adult. Also the amount of blood significantly changes. Other parameters such as the oxygen saturation do not change at all. These different behaviours can possibly mislead the model during the prediction if we train our models on such different physiologies. Because of that the first choice was to focus only on adult physiology and avoid pediatric cases.
Once a preset is chosen Aplysia automatically sets all the model parameters in order to simulate that given physiology. Since those presets comes from medical expertise they were deemed to generate more patient data. A summary of the presets used for this analysis together with a brief description can be found in Table 3.2.

Physiology name	Description	# number simulations
Adult, 20y	Adult normal case, 20 years	40
Adult, 40y	Adult normal case, 40 years	40
Adult, 60y	Adult normal case, 60 years	40
Adult, 80y	Adult normal case, 80 years	40
Biventricular failure	Disease coming from both side of the heart: both ventricles fails to contract correctly. May happen when the muscle is weak or not elastic enough.	20
Severe left ventricular systolic failure	Left ventricle loses its ability to contract in the proper way.	20
Stiff ventricle	The patient is characterized by a stiff left ventricle, which usually leads to an increase in the end diastolic pressure.	20
Relaxation abnormality	The heart is not able to relax fast enough, leading to abnormal behaviours during cardiac cycle.	20
Right hearth failure	Disfunction in the right heart structure, predominantly in the right ventricle.	20

Table 3.2: Presets used for the patient generation. Each of these entries has different characteristics.

3.6.3 Cardiovascular model generation

As stated in the previous section we need to generate more model parameters in order to increase the number of simulated cardiovascular systems in our dataset. In order to do that, the following pipeline has been developed:

1. We choose a preset and we extract the cardiovascular model parameters.

2. We define two quantities:

   - parameters range: a lower and upper bound to threshold the values of each parameters during the generation of new ones;
   - parameters variations: an interval that indicates a lower and upper bound for how much a parameter can vary in percentage.

3. We take each model parameter and we randomly update it with variations bounded by the lower and upper bounds described in the previous point.

The final result will be a set of target parameters that will be similar but not identical to the ones in one of the predefined parameter sets. In this way we explore the input feature space and this helps the model to have a better representation of the input parameters and how they vary with respect to the output. The pipeline described above is summarized through an example in figure 3.3.

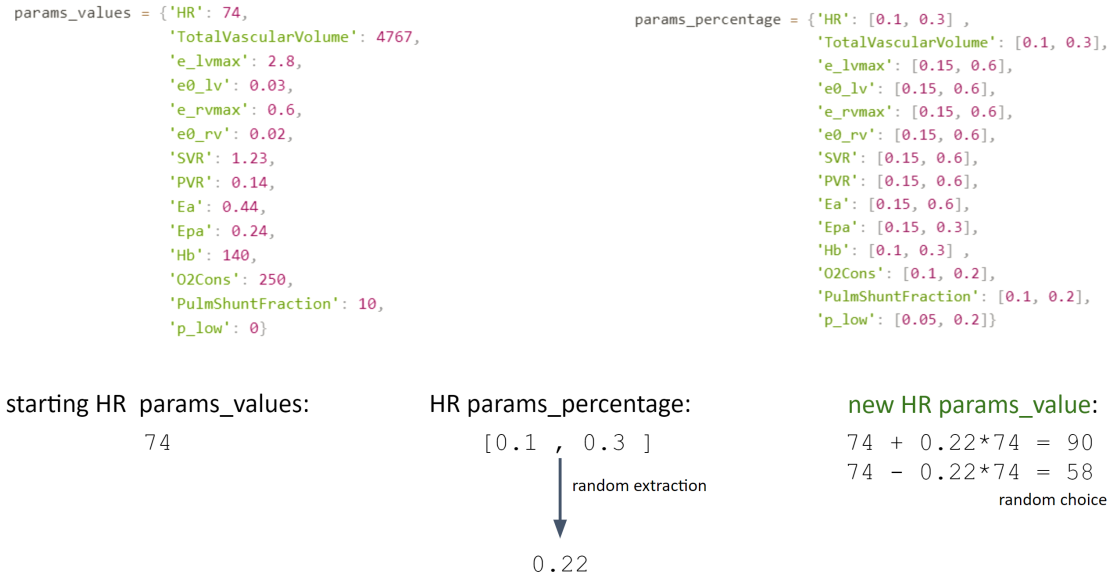


Figure 3.3: Example of the pipeline. On top we have *params_values* that contains the target parameter for a given physiology and *params_percentage* which contains the intervals. From left to right: we extract a parameter from *params_values* (Heart Rate in the example) we then access the relative interval contained in *params_percentage* and we randomly extract a percentage in that interval. We then increase (or decrease) the starting value of the extracted amount. By repeating this procedure for all the parameters we obtain a new model parameter set.

Once obtained those set of target parameters we fed them into Aplysia and we record the choosen subset of state variables discussed in table 3.1.
The most challenging part in this stage was to properly identify the range in which each model parameter could vary. If we vary too much one we can reach an unrealistic set of parameters. Nevertheless Aplysia is really sensitive to big changes of certain parameter (e.g. the vascular volume) and big variations during the data gathering leads the simulator to stop working. This happens because Aplysia has consistency-check that triggers on large changes in state variables and if they are too large the simulation is halted. Finetuning the interval of variation was then needed in order to properly generate the dataset in the smoothest possible way.

3.6.4 Dataset(s)

The dataset consisted in a total of 240 patients with state variables sampled at 200 Hz frequency for 5 seconds (1000 samples for each patient). In order to check if the patient were generated in the proper way a first clue was given by the t-SNE visualization in 3.4. t-SNE [41] is a technique to visualize high-dimensional data in a two or three-dimensional map. This technique is very important for data that are described by an high number of variables and can raise relevant observations. This technique is capable of capturing much of the local structure of the high-dimensional data very well, while also revealing global structure such as the presence of clusters at several scales. In 3.4 we have a t-SNE of the model parameters (table 2.1) from all the 240 simulations. We can notice how most of the clusters are formed by simulations with the same starting preset: this full-fill the requirement of having simulations that resembles the original presets without being duplicates of them. However we can notice that some cluster are heterogeneous and composed by patients with different physiologies. This means that some diseases share similar values of the state variables even if their starting presets is different. This is possible since some starting presets describes similar disease and it can happen that they are rather similar and thereby get clustered together.

Aside from this first dataset, another one was created including the patients' body weight and length. Including those two informations is a good way to test the model robustness to changes in the input: we need to make sure that our models does not make prediction on the model parameters based on an estimation of the body dimension since many of the model parameters we want to estimate are correlated to the patient body dimension. A clear example is the blood volume, which is directly proportional to the patient surface. This type of correlation may be good to improve the prediction performance but at the same time we risk that our model focus on predicting those characteristics rather then the target parameters. To make sure that our model is not predicting the parameters by guessing the patient body dimension we will make a comparison training the top

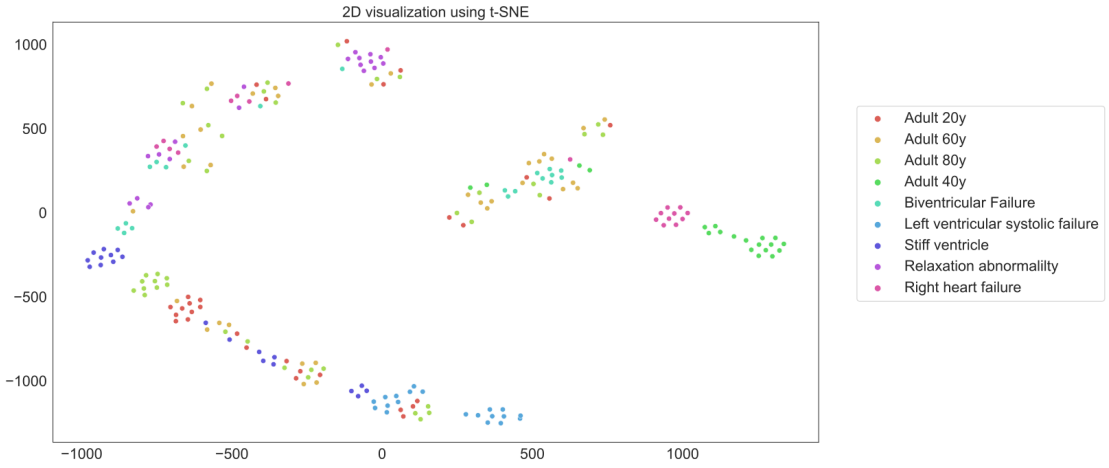


Figure 3.4: t-SNE visualization of the generated dataset.

performer model on a dataset in which those information are already available. If the model is actually able to learn different insights from the data rather then just body weight and length then we should not notice a huge drop of performance among the two datasets.

Before starting with the experiments the dataset was checked and wrong data entries were removed and generated again. This data cleaning operation was necessary since we need to have a properly generated dataset in order to have valuable results.

3.7 Machine Learning experiments

All the machine learning experiments were performed on a laptop with the following characteristics: Intel(R) Core(TM) i7-10710U CPU @ 1.10GHz processor, 16GB Ram, NVIDIA GeForce 1650 graphic card. The main framework used was Scikit learn (or sk-learn) [42] which is among the most used framework to perform machine learning tasks. Most of the computation relied on the CPU since sk-learn estimators and utilities do not support GPUs, but they do support multiple CPU cores computation.

3.7.1 Features extraction

For the machine learning experiments we will adopt a feature-based approach (as explained in 2.3.1). In this way most of the common models for machine learning task can be used for the task and nevertheless we can rely on previous work on

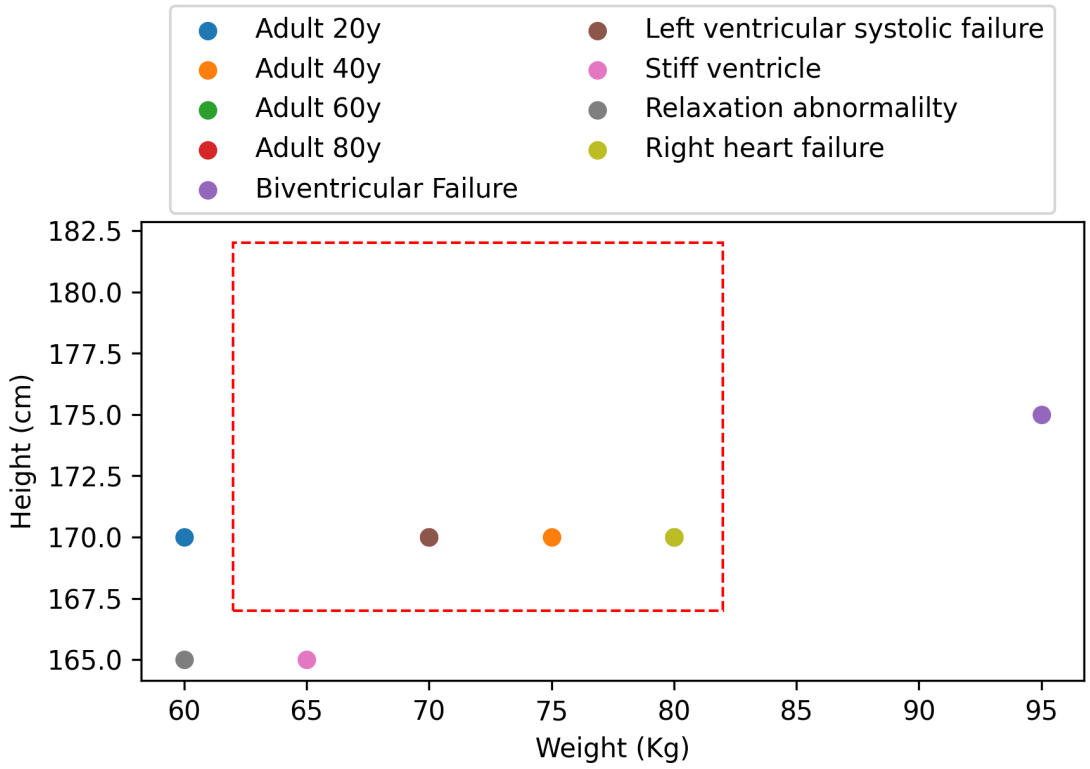


Figure 3.5: Scatter plot of the length and weight combination generated for the second dataset. The red rectangle marks the area between the 25 and 75 percentiles for both attributes considering the average population.

features representation for time series. The first step of the machine learning pipeline is then the features extraction. This step was done through TsFresh library [43] available in Python. The library is able to extract a comprehensive number of time-series features. The procedure applied for features extraction was the following:

- Extract a wide set of features. At the beginning of a project we still do not know which one could be the most significative for the task, so we need an exploratory analysis. We then characterizes time series with comprehensive feature mappings describing meta-information.
- Perform a features relevance analysis. Each feature extracted is individually and independently evaluated with respect to its significance for predicting the target under investigation. The result of these tests is a vector of p-values.
- Filter features based on the relevance. The vector of p-values is evaluated on

the basis of the Benjamini-Yekutieli procedure [44] in order to decide which features to keep.

Most of the time training a model on a wide number of features is computationally expensive, hence the need for filtering. Each of those steps has been applied independently on the different channels of each patient's time-series. The features were also fine-tuned based on the final performances. Surprisingly we obtained the best results in terms of performance by filtering just few features, keeping the training time low. The only features that were filtered were the one that generated null entries or non-numeric results. This aspect is most likely due to the low amount of patients generated: even if we increase the number of dimensions (features) of the dataset we still have only 240 patients available for the experiments, which is a low number if compared to other experiments analyzed in the literature.

3.7.2 Validation

In order to validate the results an holdout technique has been used: this means that we randomly split the dataset into train and test data. Since the split is performed on a random basis each model has been trained and tested 5 times with a different split every time. By doing that we make sure that the final performance of the model does not depends on how the training and test data were shuffled. We can take a look at the different splits coming from the 5 seeds tested in figure 3.6.
As performance metrics the following were used (further details on the metrics are available in section 3.4):

- MSE averaged through all the target parameters;
- MSE for each target;
- RMSE for each target;
- aRRMSE.

Together with that we kept track of the runtime of each model in order to compare also the efficiency in terms of computational time.

3.7.3 Regression models

Here it follow a list of the regression models tried with a brief overview of the algorithm behind. We are testing different models since there is no previous studies related to similar task (MTR applied over MTS) hence we need to explore different alternatives in order to find out which one may work best.

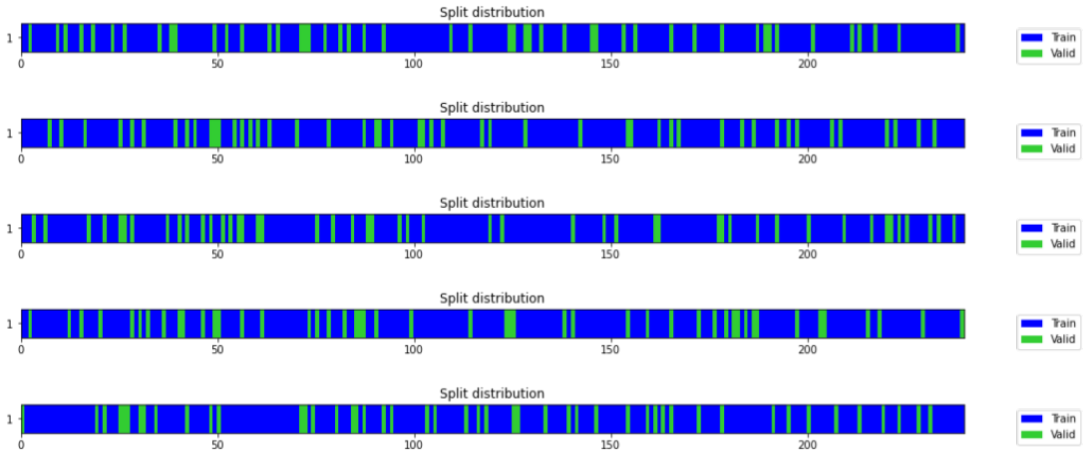


Figure 3.6: 5 different train and validation splits. We can notice how for each seed we have different data falling into the validation subset (the green segments).

- **Ridge Regressor**: one of the naive regressors used. This model solve the regression problem by minimizing the least squares function, regularized by an euclidean norm. From the optimization point of view, we want to minimize (with respect to w) the following:

  $$||y - Xw||^2 + alpha * ||w||^2$$

  with y as predicted target, X the training data and $alpha$ fixed. This is one of the most basic regression model that can be used for multi-output problems;

- **Decision Tree Regressor**: Non-parametric and unsupervised method. The algorithm learn simple decision rules on the data features and makes decision using them. It was hoosen because it requires no data preparation or normalization, cost of the tree is logarithmic to the number of data samples and it is considered a white-box model;

- **Random Forest Regressor**: Ensemble of decision trees. When we use ensemble models we combine predictions of several base estimators in order to improve the robustness over a single one. Chosen because often individual decision tree tend to overfit the data although it does not support natively multi-output;

- **Gradient Boosting Regressor**: or Gradient Boosted Decision Trees (GBDT) is an effective off-the-shelf model that can be used for our task. It does not natively support multi-output;

- **KNN Regressor**: not properly a model since it is a instance-based learning algorithm, so it simply stores instances of the training data without training any internal model. The label assigned to a sample is computed on the mean of the labels of its neighbors;

- **mSVR**: Support Vector Machines (SVM) can be both used to perform classification and regression (the m stands for multi-output). This model was chosen because of its versatility (different kernel functions can be used to deal with different data spaces, the so called "kernel trick") and because it is effective in high dimensional space, even when the number of dimension is greater than the number of sample.

All these models were available in the sklearn library except for the mSVR which was implemented by customizing an available model[1] in order to make our data format readable.
As explained in section 2.3.1 not all the algorithm natively support multi-output problems. Regressors stacking and regressors chaining are a workaround to this limitation:

- **Regressors stacking**: simple strategy that consists in fitting one regressor per target. It is possible to inspect each regressor and gain knowledge about the target but this strategy does not take advantage of correlation among targets;

- **Regressors chaining**: each model makes a prediction in the order specified by the chain using also the predictions of the earlier model in the chain. The order of the chain is important and in general we don't know the optimal ordering, hence we need to fit many randomly ordered chains.

in table 3.3 we can notice which one were the models tested with those approach.

3.7.4 Experiments settings

Each experiment is characterized by a seed for the reproducibility of the results and also to ensure that we split the dataset in a different way. Each model has been tested 5 times each time with a different see. In this section we will refer as experiment an entire training and validation of a single model using a unique seed. Furthermore, a baseline regressor has been trained during each experiment. When it comes to this type of experiments is good to compare the trained estimator against a simple rule of thumb. We refer to that as a baseline regressor, i.e. a

[1] https://github.com/Analytics-for-Forecasting/msvr

Regressor	Native	Stacking	Chaining
Ridge		✓	✓
Decision Tree	✓	✓	✓
Random Forest	✓	✓	✓
Gradient boosting		✓	✓
KNN	✓		
mSVR	✓		

Table 3.3: Summary of how the different models were managed to perform the multi-output task.

regressor that always give as a result the mean of the training target. If the model actually learned from the data we should notice a huge difference in performance between the dummy estimator and the model under experiment. Results and outcomes of these experiments are discussed in chapter 4.
Most of the work was due to hyperparameters tuning for each model. It was not feasible to perform a grid search because the regressors stacking and chaining approach needs to train one model for each target and grid searching hyperparameters for each of the 11 models was too time consuming. The final approach consisted then in applying the same set of hyperparameters to all the sub-model during the search, which was done in a random search way. For the algorithms which natively support multi-output (see table 3.3 to see which we are referring) it was possible to perform a grid search hence the hyperparameters optimization was faster.

Regressor	Hyperparameters	Description
Ridge	alpha	Controls the regularization strength.
Decision Tree	criterion	Function that measure the quality of a split. It can be thinked as of a loss function.
	max_depth	Maximum depth of a tree.
Random Forest	n_estimators	Numbers of tree to ensemble in the model.
	criterion	Same as Decision Tree.
Gradient boosting	loss	Loss function to be optimized.
	max_depth	Maximum depth of nodes in the tree.
	n_estimators	Number of tree that contributes to the prediction.
KNN	n_neighbors	Number of neighbors to use for the algorithm.
	weights	Weight function to use.
mSVR	kernel	Specifies the kernel type to use.
	gamma	Kernel coefficient.
	epsilon	Specifies the epsilon within no penalty is associated in the training loss.
	C	Regularization parameter.

Table 3.4: List of the fine-tuned hyperparamters for each model.

3.8 Deep Learning experiments

All the deep learning experiments were performed using the free computational resources available from Google Colab. For deep learning most of the computation are GPU-centric. Services as Google Colab offer GPUs for a limited amount of time for each user, hence computational resources were the main challenge during those experiments. Most of the deepest architecture requires a lot of time to deploy and train and is really easy to run out of GPU allocation even before starting the training phase. The deep learning experiments were then resized on the amount of computational resources available. Three main framework were exploited for those experiments:

- **TensorFlow** [45]: open source, python based platform for machine learning and deep learning developed by Google. It allows to easily build and modify state-of-the-art architectures. Thanks to that there is no need to re-implement architectures from scratch but is necessary to modify the proper layers. This flexibility was the main reason behind the choose to perform experiments within this framework;

- **PyTorch** [46]: open source machine learning and deep learning framework. Its popularity is recently increasing since it simplifies the creation and deployment of complex models. As TensorFlow does, this framework offers the flexibility of customizing deep architectures, such as the TFT architecture which is already available in this package: that's one of the main reason of choosing to perform experiments also within this framework;

- **tsai** [47]: open source deep learning package that focuses on providing state-of-the-art architectures (such as ResNet, LSTM and 1-dimensional CNN) to train and customize in a user-friendly way The package is built upon PyTorch and fastai. It was used in order to have a quick feedback on how a pre-implemented architecture could work on the dataset. Most of the architecture are adaptable to multi-channel time series input and they support multi-output.

A first exploratory analysis was needed in order to evaluate which architecture were computationally feasible to develop and train.
As already pointed out computational resources were the main challenge. Because of that it was not possible to perform a single complete experiment with the TFT architecture because it required a lot of resources both to deploy and then train: it was not possible to finish a full training cycle without running out of GPU resources. Furthermore, the number of parameter sets generated was not enough to feed such a deep architecture, as the results from the experiment will show. So the TFT architecture was discarded. The following architectures were the one used

in the deep learning experiments: ResNet Plus (from tsai package), 1D ResNet implemented from scratch, 1D custom CNN implemented from scratch.

3.8.1 Training methods

Even if the architecture tested comes from different framework, the training process was almost the same for all the 3 architectures tested. Most of the deep learning architecture final performance depends on the training stage. The training of a network is made through the Stochastic Gradient Descent algorithm (SGD). The algorithm consists in a weight update procedure by scanning through the training data: we take the training data, we go through the network, we compute how much our result is far from the ground truth (the so called loss) and we then update the weight depending on how much the prediction was wrong.
There are several variations of this algorithm and we can customize the training stage by fine-tuning some hyperparameters. Here it follows a brief explanation of the most important parameters involved during the tuning stage. It is useful to get a knowledge of those since they will be cited through the description of the experiments:

- **Validation size**: percentage of data that will not be used for training but to validate the model throughout the training process;
- **Batch size**: the training data is partitioned in mini batches of a fixed size. This procedure helps the learning algorithm (SGD, Stochastic Gradient Descent) to converge to a solution. The optimal size depends on many factors and most of the time is task-specific;
- **Epochs**: number of times all the training data have passed through the neural network during training. As already pointed out, the learning process is basically a weight updates and those updates are done in small steps, hence the need to scan through the training data multiple times;
- **Learning Rate**: hyperparameters strictly related to SGD. It states the magnitude of the weight update through each epochs. Commonly this parameters change through the epochs since is best to make smaller adjustment when we are approaching to a solution: by decaying it through the epochs we allow to make big update in the beginning of the training when the network does not know anything about the data and small change towards the end when the network has been through several updates;
- **Dropout**: regularization methods that randomly drop nodes in the network during the training stage. Studies have shown that this strategy helps to avoid overfitting the data [48].

In order to quantify how far we are from the ground truth during the training phase we need to define a loss function. The loss function used during training were the MSE and RMSE (further details are given in 3.4) averaged through all the target. Since multi-target loss function were not available out-of-the-shelf it was necessary to implement customized loss functions to fit our task.

3.8.2 Data normalization

Even if our time series are all of the same length their values are in different ranges (we can get a feeling of that from table 3.1). This does not represents a problem for most of the machine learning algorithms, but is not ideal for a deep neural network. One of the most common normalization used for time series classification [39] is the z-score normalization. Also referred as Z-normalization [49] or "normalization to zero mean and unit of energy" it consist in forcing the sample of each time series to have 0 as mean and 1 as standard deviation. This is obtained by rescaling each sample x_i of a through the mean μ and standard deviation σ in the following way:

$$x_i = \frac{x_i - \mu}{\sigma}$$

For the following experiments all the time series gathered where first z-normalized before feeding them to the architectures developed.

3.8.3 ResNet Plus configuration

This model is a PyTorch-based implementation of a ResNet adapted for multi-input and multi-output task (that's what the "Plus" in the name stands for). This first experiment was performed in order to check if an off-the-shelf implementation of a ResNet could give good results.
This experiment has been quick and straight-forward since tsai offers ready to use functions to perform all the steps needed for a deep learning-oriented task. The starting learning rate has been found through a learning rate finder function that starts the training with very low learning rate and change it at each batch of data. By looking at the loss plot during this process we can get a clue on the starting learning rate. A good choice is a value that is nearby the loss drop (see figure 3.7) After the learning rate we tuned the batch size. A usual choice is to choose a batch size as large as the memory consent (a batch is kept in memory while training). As for all the parameters there is not a general rule but the choice is task dependent. Since we have a small dataset the batch-size search has focused on relatively small values (ranging from 8 to 32). Since the model was already implemented no other architecture-related parameters were tuned.

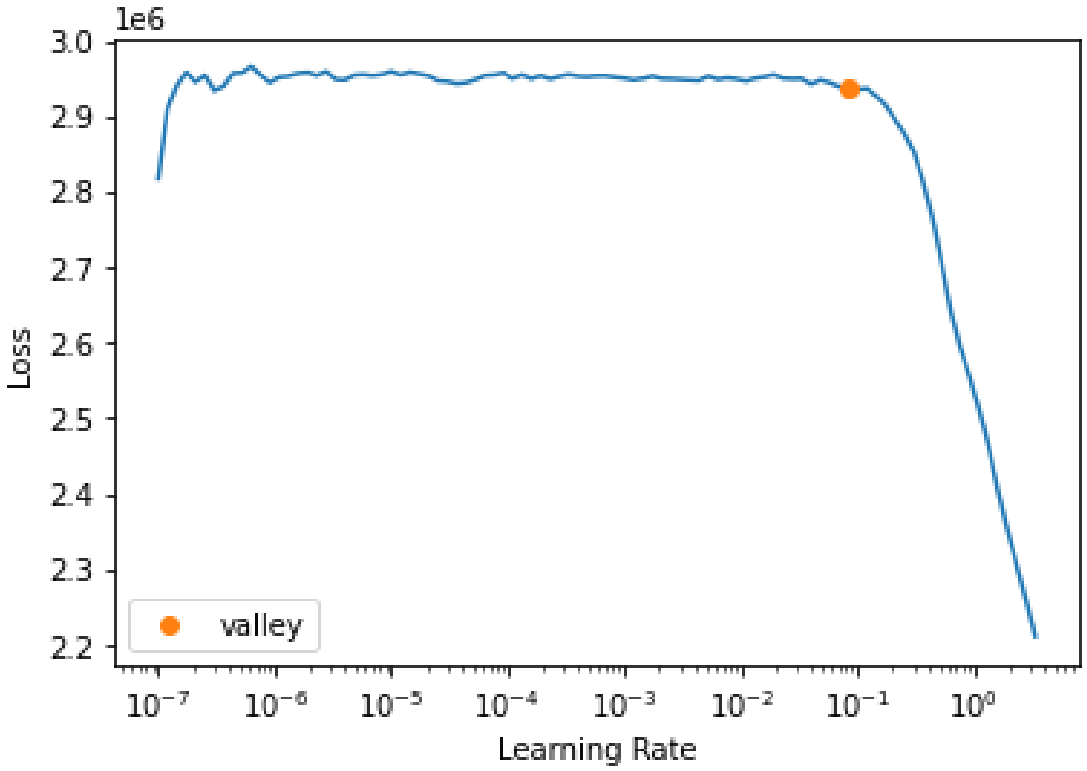


Figure 3.7: Learning rate identification plot. The orange dot states the last spot in which the slope is low: a good starting learning rate can be found around that area (in this specific example, a good choice would be between 10^{-2} and 10^{-1}).

3.8.4 1D ResNet configuration

This architecture had to be implemented from scratch since most of the ResNet architecture are tailored for computer vision tasks, so they relies on 2D convolutional layer which does not work for our task since we need convolution only on the time dimension. Furthermore, we need an architecture that support multi-output. Because of those two aspects it was not possible to re-use any model already implemented. A scheme of the final architecture can be found in figure 3.1.
To obtain the multi-output feature it was necessary to modify the last layers of the architecture: a last dense layer was added with the same number of output neurons as of the target values. No softmax function is needed since we are dealing with a regression task so ideally the final output of the network needs to be the real values of the target parameters (and not likelihood of belonging to a certain class). After adding the 1D convolution and implementing the skip connection, the

1D-ResNet architecture was ready to be trained.
Since deeper architecture often requires more epochs to reach convergence a learning rate scheduler usually helps throughout the learning process: usually we start with an high learning at the beginning when we do not know anything about the data, and so is better to explore more the solution space. When the architecture has learned something we lower it in order to make small steps towards the optimal. The learning rate decay was set such that if the validation loss values get no improvement for a certain amount of epochs the learning rate get reduced by a factor of 2. After searching for the correct batch size in low range of values (for the same reason explained in the previous experiment) also some trials with and without dropout were made.

3.8.5 1D CNN configuration

Those experiments involved developing from scratch a shallow CNN with 1 dimensional convolution layer in order to have an effective and light weight architecture as a baseline for the deep learning approach. This type of approach was also adopted in [6].
In the figure 3.8 we can get a clue about the architecture which is composed of just 1D convolution followed by batch normalization layers and a final average pooling layer. No skip connection are included. Since this architecture is small and easy to manage it was possible to perform an automatic architecture parameters optimization using KerasTuner, a framework tailored for parameters search. In this way we could tune the architecture parameters like kernel size, filters, the usage of batch normalization by a random search.

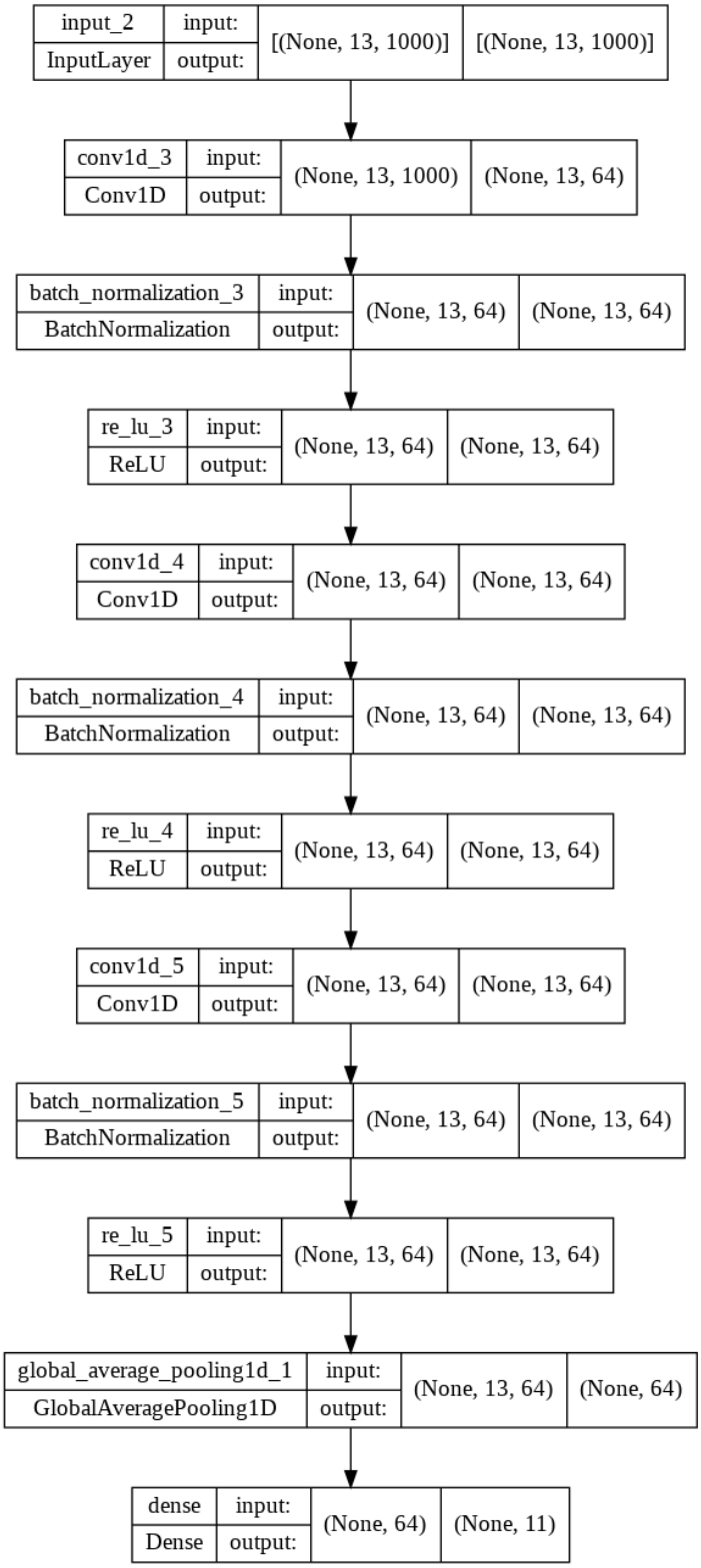


Figure 3.8: Scheme of the 1D-CNN used in the experiments. The number in the parenthesis states the input and output dimension of each layer. We can notice how after each convolution the depth of the time series changes because of the convolutions.

Chapter 4

Results

In this section we will comment and evaluate the results obtained through the experiments described in the previous section. In order to be clear, in the following table 4.1 there is a recap of all the acronyms that will be found through the results discussion.

Parameters	Abbreviation
Hearth Rate	HR
Blood volume	$TotalVascularVolume$
Left ventricular contractility	e_lvmax
Right ventricular contractility	e_rvmax
Left ventricular stiffness	$e0\_lv$
Right ventricular stiffness	$e0\_rv$
Systemic Vascular Resistance	SVR
Pulmonary Vascular Resistance	PVR
Systemic Arterial Stiffness	Ea
Pulmonary Arterial Stiffness	Epa
Hemoglobin	Hb
Total oxygen consumption	$O2Cons$
Pulmonary shunt	$PulmShuntFraction$

Table 4.1: Abbreviation for the target model parameters.

4.1 Machine learning results

In table 4.2 we can notice the final performances of all the machine learning models tested. The results refers on the performances on the test data. All the models were fed with the raw time series, except for the mSVR which required z-normalization

(more details in 3.8.2) since all the SVM-based models requires data to be in a standard range.
Among all the models tried the top performer results to be a Gradient Boosting regressor with regressor stacking technique. It performs best both in terms of MSE (averaged through all targets) and aRRMSE. The worst performer turned out to be the KNN Regressor which we recall support natively multi-target problem.
The results we got from these experiments were both satisfactory and unexpected. We were expecting that the top performer would have been one of the native multi-output models for their capability of taking into account the correlation among all the targets during the prediction itself. Apparently this aspect was not enough to gain in performance for this task and it might have misled the model during the prediction. This shows us how correlation among target and input is not helpful in any task but is strictly task-related. The native multi-output regressors were still the one with the fastest computational time: if we look at the mSVR performance they are quiet decent if we consider that it required just 0.12 sec to train and deploy on the test data which is quiet impressive if compared with the 110.2 sec needed to train the top performer Gradient Boosting regressor. This aspect was quiet expected since with regressor stacking we are training several models (one for each target) instead of a single one.

Regressor	Approach	MSE averaged (mean ± std)	aRRMSE (mean ± std)
Ridge	Multi-output	3621.3 ± 633.0	0.55 ± 0.02
Decision Tree	Chain	4741.4 ± 1351.9	0.72 ± 0.05
Random Forest	Multi-output	2497.0 ± 1109.0	0.52 ± 0.03
Gradient boosting	Multi-output	**1749.9 ± 564.2**	**0.46 ± 0.04**
KNN	Native	24884.4 ± 3650.7	0.72 ± 0.02
mSVR	Native	4507.8 ± 609.0	0.54 ± 0.03
Baseline	Native	31923.2 ± 8221.5	1.01 ± 0.01

Table 4.2: Performances of the machine learning models tested. The MSE and aRRMSE score are computed on the test set and averaged through 5 different runs. If a model has been tested with different approaches only the one with the best scores is here reported. The best scores are in bold characters.

4.1.1 Different performance among model parameters

In the following section we are going to analyze more in the details the results obtained with the top performer model: a stacked Gradient Boosting regressor.

This further analysis has been conducted only on the top performer since all the different models tried can be seen as an exploratory study of the task: since it is the first time a study of this type has been conducted it was worth trying a wide range of models to spot which one could work best. Once we found it, we can proceed with further analysis.
From figure 4.5 we can infer the performance obtained on each target. As a rule of thumb to read those visualization we can consider that the more the distance from the red line the more the model is wrong. The data point are colored depending on the patient group they belong.
We can notice how the most difficult attribute to predict were the PVR and SVR since all the data points are scattered in the graph and we can also notice that most of the predicted values lies around the average true value. This last aspect is more evident in the SVR graph, in which we can notice how the trend resembles a horizontal line rather then the ideal bisector line. Those two parameters are actually difficult to measure in a precise way. They can be estimated from some formulas since they are "composites" by other body parameters. For instance, we can formulate the SVR as:

$$SVR = \frac{MeanSystemicArterialPressure - RightAtrialPressure}{CardiacOutput}$$

Even if we have all of those parameters as input for our model, our results are still not satisfactory. This could be because even this formula gives just an approximation of the real SVR value, which may depends from different physiology aspects such as the musculature of the patient, length and shape of the vessels and so on. Those parameters were not taken as input state variables, so we might be missing some information to correctly predict those model parameters.
We do have good performances on parameters such as total vascular volume and $e0_{lv}$ (Left ventricular stiffness). We can notice from the scatter plot that those parameters were correctly estimated for most patient groups. This result was impressive since is really difficult to estimate correctly the total vascular volume of a patient: as for the SVR we can do this by formulas but we would obtain just an approximation. Also most of the formulas for this parameter depends on the body dimension, which was not included in the dataset for this analysis: that makes this result even more valuable.
If we focus on the patient groups we can not notice any particular behaviour coming from the difference in the physiology, we can though appreciate that to different physiology correspond different ranges of values for each parameter: this is perfectly coherent with the data generation procedure that we adopted. For some physiology some values are in the same range, such as in the $PulmShuntFraction$ graph in which we can notice a mixed cluster in the left part or in the middle section of the $O2Cons$ graph. We can tell that this was a consistent problem for the prediction:

for the parameters in which the patient values were spread much more evenly in the space we obtained better results (see again the $TotalVascularVolume$ as a clear example). We recall that from different physiology groups derives differences in some state variables that were excluded from this analysis: this aspect can be a problem since the model might not distinguish two different physiology, assigning to them similar prediction even though they are really different.
The visualizations in figure 4.5 plot the predicted and ground truth distribution of the test data. We can notice that for most of the target parameters the model is able to predict correctly the true distribution of the data even when it is a bi-variate normal distribution (which happens for most of the target). The same comments from previous section applies also on those visualization: we can notice how the predicted SVR and PVR distribution are far from the actual one. A similar behaviour happens for the $O2Cons$. If we compare the two distribution, the predicted expected value is similar to the true one but the predictions seems biased towards that value since we can notice an high peak in the distribution.

4.1.2 Problems deriving from high dimensionality

As a comparison, here it follows the same visualizations but for the worst performer. This analysis is useful to understand that other models were actually able to learn useful information from the data gathered. For these results the scatter plot is not color-mapped with the patient physiology since the model performs bad independently from that. We can notice from the scatter plot in 4.7 that the predicted values lies around the true mean for all the parameters. This can be noticed also from the high peak in the predicted distribution plots: the model hardly recognize a bi-variate distribution correctly. Sometimes the predicted distribution is completely different from the true one (see systemic arterial stiffness). Even if this algorithm is a native multi-output one, we can explain this huge fail in tackling this task probably because of the number of features of the dataset: the more dimension we have the more we complicate the distance calculating process along each dimension. We refer as this problem as curse of dimensionality, which more in general refers to the set of problems that may arise when we analyze high dimensional spaces.

4.2 Deep learning results

Even if the basic machine learning models gave great results we got worst when we applied deep learning architecture on our data.

4.2.1 Risk of overfitting with deep architectures

The first results we are going to show are related to the ResNet architecture. As explained in the previous section we tested two type of ResNet: ResNetPlus from tsai as an exploratory analysis and a custom ResNet implemented from scratch within the TensorFlow network. From the first experiment with ResNetPlus we could already infer that the ResNet architecture is too deep for the amount of data we have since it quickly overfit the data (as we can see from the learning figures in figure 4.1). This behaviour does not change even with further fine-tuning and regularization. We could not get any valuable results from the tsai architecture.
The hypotesis elaborated after the first ResNetPlus experiments were confirmed after training the custom ResNet. After several fine-tuning steps we were still not able to obtain any useful results. We can notice from the figure 4.2 that the validation loss curve quickly goes under the training loss: this behaviour is not regular and suggest that there are some problems during the learning process. One problem could be related to the data shuffling: it may happens that "easy" patient are getting into the validation set. To avoid that we performed several tests with different seed but the results were about the same. Also those curves seems to suggest that there was data leakage during the experiments: that means that some data are duplicated and they are present in both train and validation subsets. This could actually be true since we had no control on how the patient data were generated since the procedure was random, then it could be that some patients had really similar parameters. A double check on the dataset confirmed that there were no duplicates in the data. Adding further regularization (such as increasing dropout) helped a bit but we still obtained the same behaviours.

4.2.2 Fine-tuning architecture depth

Those experiments were the most interesting among the deep learning ones. Since deep structure could not give any valuable results we decided to perform different experiments by modifying the CNN architecture. Each experiment consisted in training a CNN with different number of convolutional layers: in this way we could check if the dimension and deepness of the architecture is a relevant factor that affects the overall performance on this task.
The experiments consisted in developing three different 1D CNN for time series with different numbers of convolutional layers (3, 5 and 8 respectively). As we can see from the learning curves in figure 4.3 the results confirmed what was our hypothesis after the ResNet experiments: a deep architecture quickly overfit the data. From the learning curve we can infer that the 3-layers architecture is underfitting the data since there is a huge gap in performance between the training and validation curve: the model is not able to extract enough knowledge to perform good also on never-seen data. The gap get thinner when it comes to the 5-layers architecture:

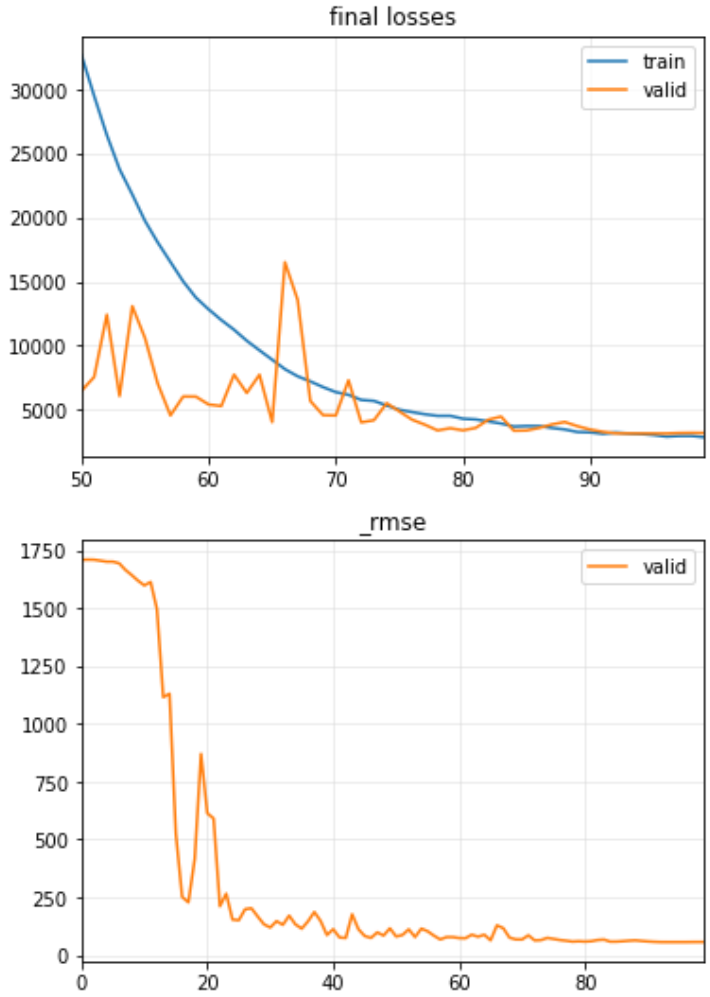


Figure 4.1: Learning curves from tsai experiments. The curve on the bottom shows the loss (MSE) both on the training (blue curve) and validation (orange curve) set during the epochs while the curve on the bottom displays the RMSE through the epochs.

the learning curves shows that the model is able to extract the right amount on information since is not either overfitting or underfitting. With that architecture we reach a plateau in the performance after 350 epochs. The 8-layers architecture shows the same ResNet learning curve, so the same considerations applies to that. The only architecture that was worth testing was the 5-layers 1D CNN. We then kept 10% of the data as testing data and trained the architecture on the remaining part. The average MSE was collected as metrics: the results inferred were not as

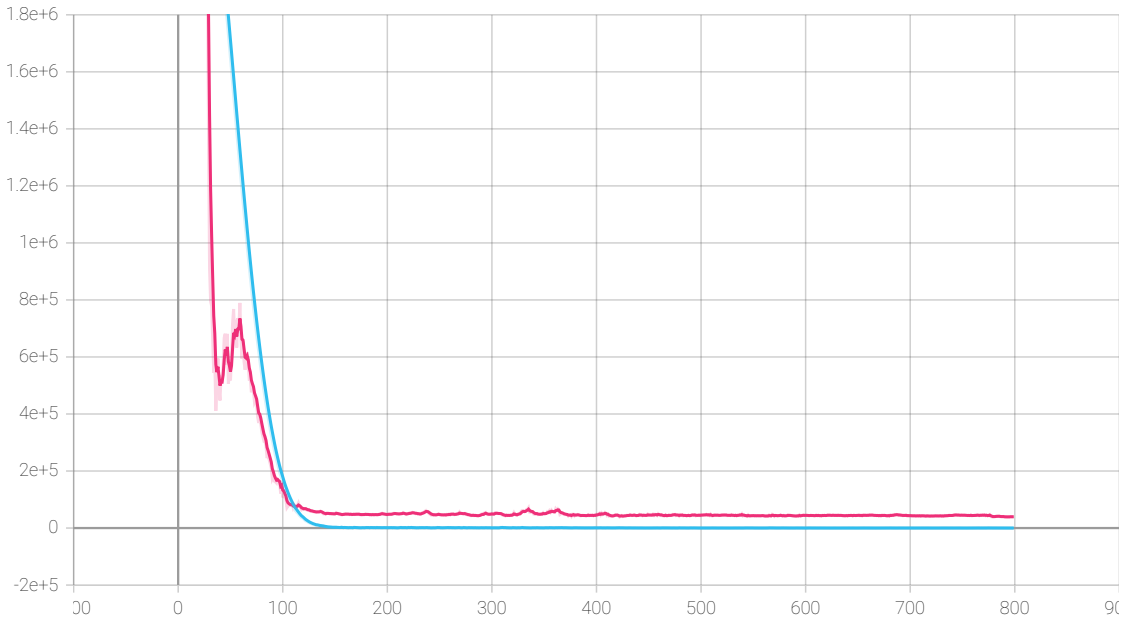
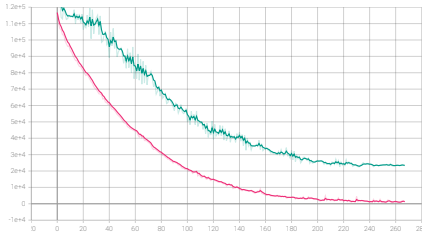


Figure 4.2: Loss (MSE) from 1D ResNet throughout the epochs. Blue: validation loss, purple: Train loss.

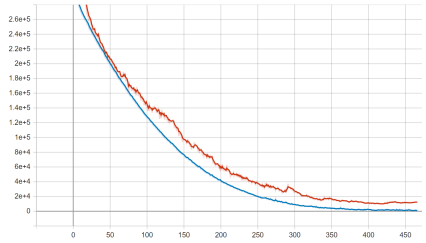
good as than the best machine learning approach since we measured an average MSE of 2354.1 and and aRRMSE of 0.52.

4.3 Variation in input: adding body dimensions

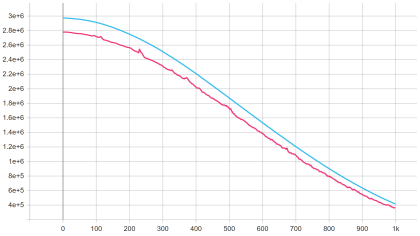
After performing the experiment on the first dataset we decided to test the Gradient Boosting regressor on the dataset containing the body size and length. We can see a comparision from table 4.3. The experiment consisted in training a new Gradient Boosting regressor instance with the second dataset and then test it. Again a 80/20 holdout technique has been used. The hyperparameters search has been performed starting from the best hyperparameters combination found for the previous experiments and by randomly varying the values.
We can notice that there is not an huge difference among the two models. By adding the body weight and length we still gain in terms of MSE and we have a slight improvement in the aRRMSE since we have lower standard deviation. We can say that the model is actually able to learn different insights from all the time-series channels and it is not making its prediction by inferring the patient body dimensions from the data.



(a) Loss (MSE) from 3 layers CNN throughout the epochs. Purple: training loss, green: validation loss.



(b) Loss (MSE) from 5 layers CNN learning throughout the epochs. blue: validation loss, red: training loss.



(c) Loss (MSE) from 8 layers CNN throughout the epochs. Blue: training loss, purple: train loss.

Figure 4.3: Learning curves for the different CNN tried.

Parameter	w/ body size	w/o body size
$TotalVascularVolume$	**17156.7**	21348.1
e_lvmax	**0.051**	0.095
$e0_lv$	3.45e-05	**3.13e-05**
e_rvmax	0.011	**0.008**
$e0_rv$	8.05e-05	**3.79e-05**
SVR	0.024	**0.017**
PVR	**0.00025**	0.00041
Ea	0.009	**0.008**
Epa	**0.009**	0.013
$O2Cons$	**806.39**	1088.99
$PulmShuntFraction$	**3.23**	12.29
Overall aRRMSE:	0.49	**0.46**
Overall RMSE:	**1420.0**	1749.9

Table 4.3: Comparision of performance among the gradient boosting regressor trained on the dataset contained for each target model parameter. The value is the MSE error so it has the same unit of error of the target. In bold are indicated the lower error for each target parameter.

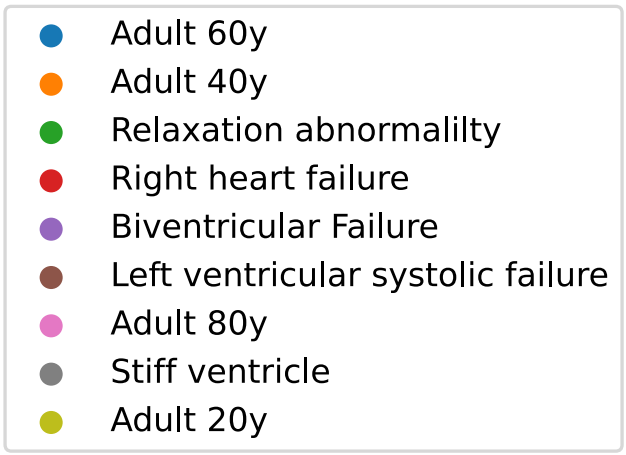
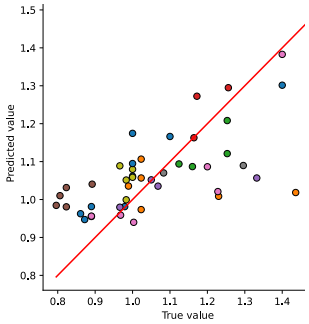
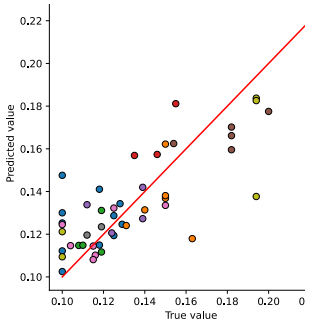


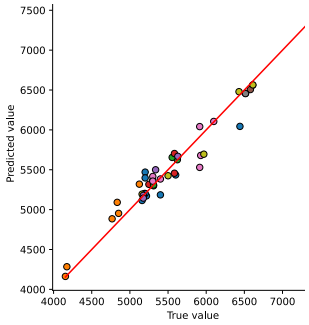
Figure 4.4: Legend for the scatter plot visualization.



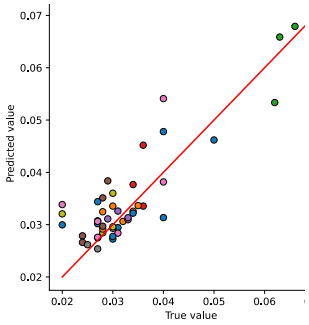
(a) Systemic Vascular Resistance.



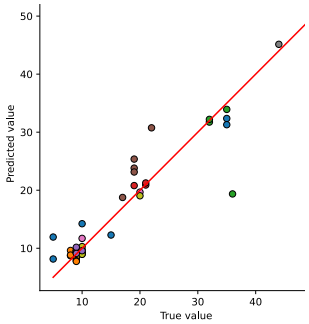
(b) Pulmonary Vascular Resistance.



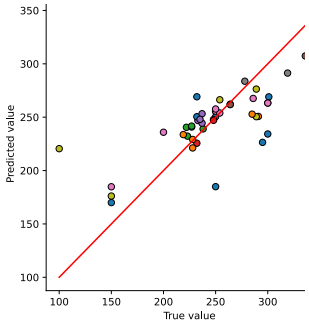
(c) Total vascular volume.



(d) Left ventricular stiffness.

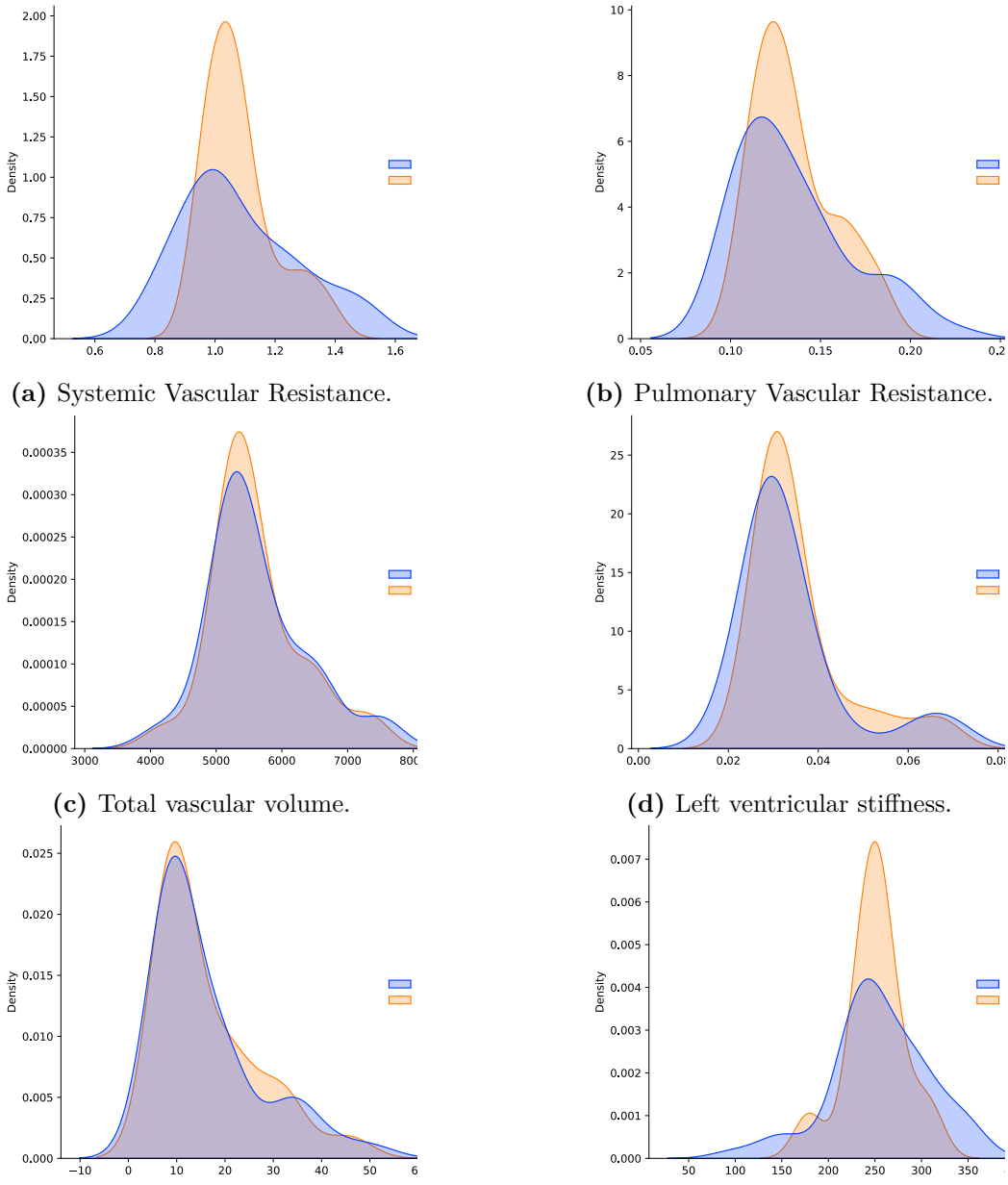


(e) Pulmonary shunt.



(f) Oxygen consumption.

Figure 4.5: Scatter plots of the test data, color coded by starting preset. Gradient boosting regressor. Legend can be found in 4.4. The same plots for the remaining model parameters can be found in A



(a) Systemic Vascular Resistance.

(b) Pulmonary Vascular Resistance.

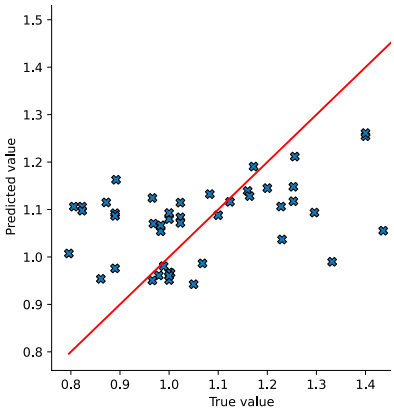
(c) Total vascular volume.

(d) Left ventricular stiffness.

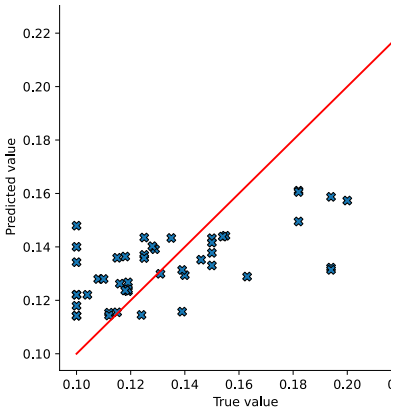
(e) Pulmonary shunt.

(f) Oxygen consumption.

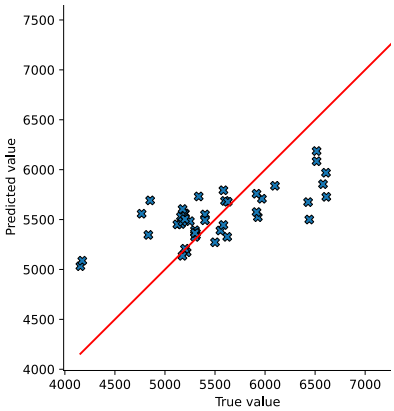
Figure 4.6: Distribution plots of the test data displaying the true and predicted distribution. The same plots for the remaining model parameters can be found in A.



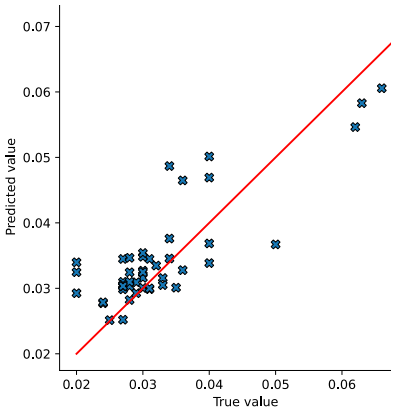
(a) Systemic Vascular Resistance.



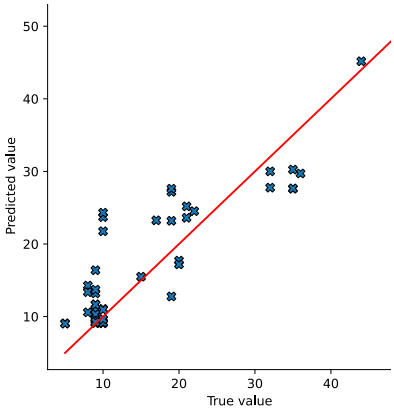
(b) Pulmonary Vascular Resistance.



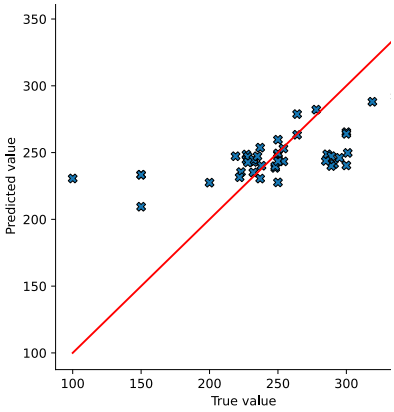
(c) Total vascular volume.



(d) Left ventricular stiffness.

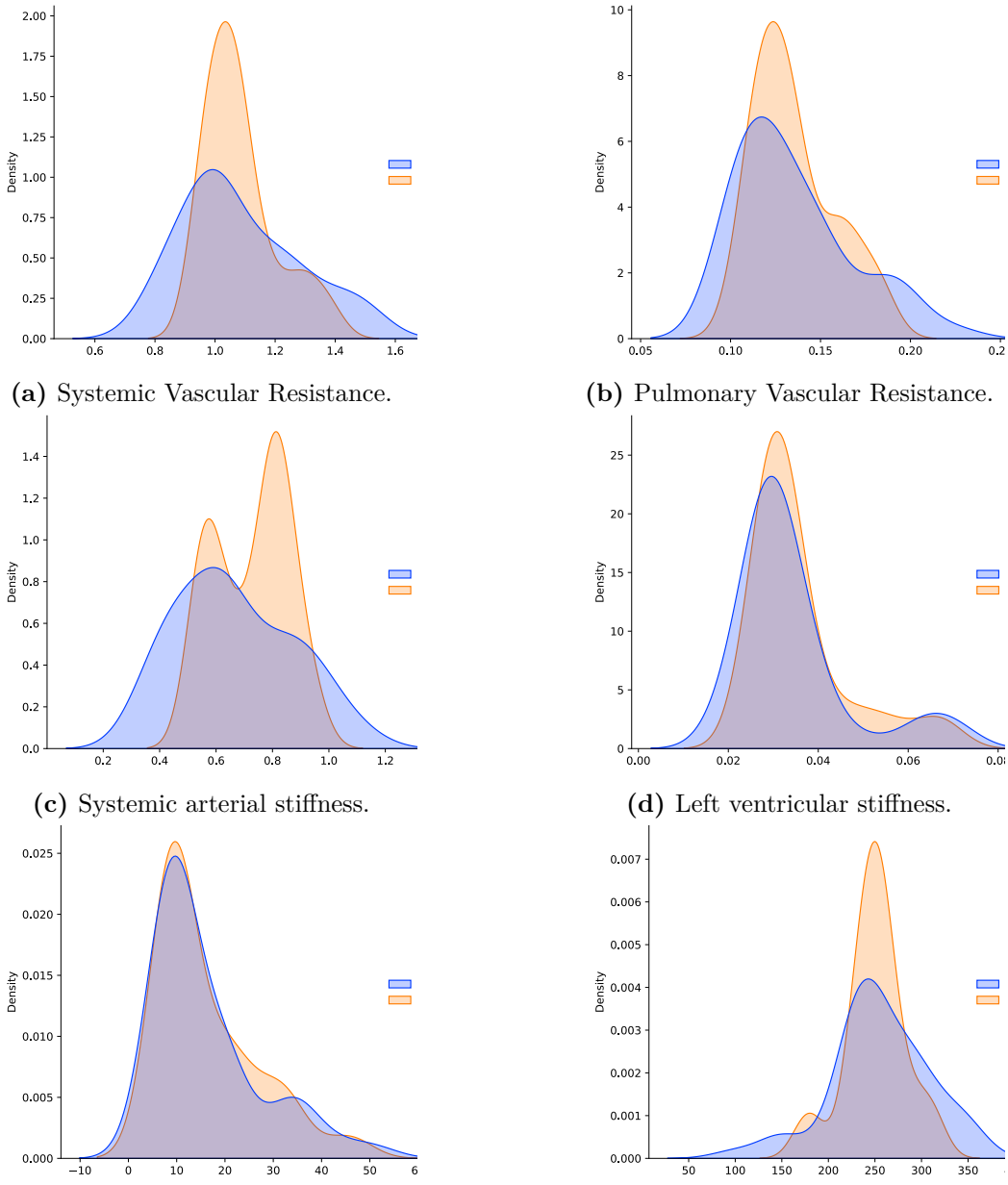


(e) Pulmonary shunt.



(f) Oxygen consumption.

Figure 4.7: Scatter plots of the test data, KNN Regressor. The same plots for the remaining model parameters can be found in A.



(a) Systemic Vascular Resistance.

(b) Pulmonary Vascular Resistance.

(c) Systemic arterial stiffness.

(d) Left ventricular stiffness.

(e) Pulmonary shunt.

(f) Oxygen consumption.

Figure 4.8: Distribution plots of the test data displaying the true (blue) and predicted (orange) distribution for the KNN regressor. The same plots for the remaining model parameters can be found in A.

Chapter 5

Discussion and limitations

5.1 Experimental findings

5.1.1 Parameter estimation

Before starting this thesis project it was known that this task is a difficult problem to solve in an accurate way. The main concerns were related to the complexity of the cardiovascular system, which we know is difficult to reproduce and analyze accurately. The multi-input and multi-output aspect was the most challenging aspect since there were no similar previous work. Because of that we had an open scenario in front of us since we had no clue of which results to expect. After performing all the planned experiments we can affirm that the results itself were indeed satisfying and valuable.
In terms of performance there is a slight difference between the Gradient Boosting regressor and the 5 layers 1D-CNN which were the top performer respectively from the machine learning and deep learning experiments. Even though the performances are similar is worth mentioning that the deep learning model requires more time to train and deploy and is also more difficult to re-deploy if new data are acquired. When it comes to decide whether a model is worth deploying or not we should also take into account those factors.
Also the native multi-output SVR model (referred as mSVR in the experiments section) brought impressive results not in terms of performance but for its computational time. Predicting all target at once does not improve the overall performance but it is still a huge improvement with respect to developing one model for each target. A lot of interesting possibilities may be explored by focusing on that model.
We were expecting the deep learning results since we were working with a small dataset. Deep architectures such as ResNet requires enough data to be able to learn properly. Also since the data was already simulated no data augmentation techniques have been used [50] but this could be a field that can be explored for

further development and improvement.

5.1.2 Cardiovascular parameter analysis

The analysis has been conducted on several different patient phisiology. We can refer to those also as patient state: if we think of a patient lifetime in ?? its condition can switch from one state to a more critical one. Also different disease might show up during the patient monitoring. If we look at figure 3.4 we can notice a C-shape on the left in which patient state slowly fade from a disease to another: this aspect show us how different disease are really similar and differs only by small change. Ideally, all patient states should be reached by "fading" from one preset to another. This indicates that, even if the amount of data is small, the patient states were generated in the proper way.
From the results obtained we can infer that our best model has low performance with some particular patient state. An example is the relaxation abnormality. This state consists in having delayed and not homogeneous relaxation of left ventricles and right ventricles. Those disturbances affects then the pressures measures over time by shifting them and this has clearly mislead the model in the prediction. Another problematic state is the adult 60 years physiology. Even if it is not a disease state the model had problem in performing well (see the oxygen consumption in 4.5). This behaviour is not surprising: we have different set of healthy state in our dataset with different body physiologies and most likely the differences between an healthy 20 years patient and 60 years patient are not caught by the time-varying patient data gathered for the analysis. Hence the model does not have enough information to discriminate among those cases. Including the age as input data would have been useful for that but it could have lead to other ethical problems discussed in 6.2.

5.2 Correlation analysis

We have already introduced correlation [40] for time series in section 3.5. Correlation is usually analyzed to describe how features in a dataset are related to each other before feeding the data into a model. We left this analysis at the end of the work since this is a multi-output task, hence we could compute the correlation among all input and all output and this can highlight relationship that can better explain the performance obtained on certain target.
For this analysis we will refers to figures 3.2 that refers to our first dataset and 5.1 that refers to the dataset containing information about body weight and length. Even if they refers to different data the pattern in the matrices are the same, the only differences lies in the values of the correlation factor. The correlation matrices were sorted using the Cuthill–McKee algorithm. This algorithm is capable

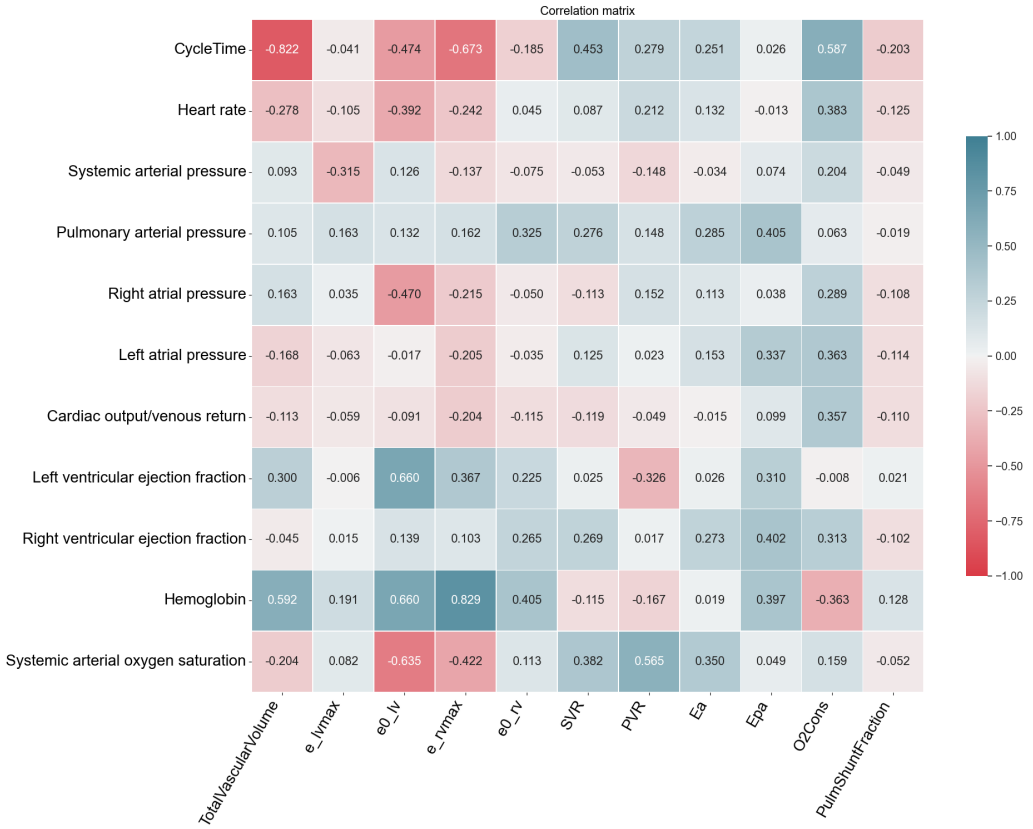


Figure 5.1: Pearson correlation between state variables (x-axis) and model parameters (y-axis).

of sorting rows and columns such that is easier to spot cluster of correlation in the matrix so it make easier to read the correlation matrix even when we have multiple input and output as in our case.
We can notice some high correlation factor that are straight-forward to understand such as the high correlation among weight and vascular volume: we expect that those two factor were linearly related since they both grow together. Also the high positive correlation between the left ventricular ejection fraction and the left ventricular stiffness describes correctly the interaction among those two parameters. Other correct patterns are the negative correlation among the length and left and right ventricular stiffness and contractility. All those correlation pattern

actually reflects on the final results: if we focus on the SVR and PVR column we will not notice high values of correlation with the input values if compared with the ventricular stiffness that has high correlation values wit a lot of input parameters. SVR and PVR were the most difficult model parameters to predict: the low correlation among those factors and the input monitoring parameters can be one of the reason of those low performance. We do not notice any huge drop in performance on parameters that has high correlation patterns with input data. Correlation is not the only factor that influences the performances.

5.3 Limitations

After performing all the experiments we can identify in the following list all the limitations that has been encountered and that can be explored to improve even more the quality of this work in the future:

- **Use of synthetic data**: even if those data are generated through Aplysia, which is built upon medical knowledge, we are still not generating all the possible events and conditions that real patient might shows during their monitoring. It would be much more valuable to train the models found using real patient data, but no such dataset are available as this project is going;
- **Small amount of data**: another drawback of having to generate our data was that we could not generate a huge amount of patients since we could risk of generating too similar phisiologies;
- **Simulator instability**: we also had to take into account Aplysia software technical limits. The simulator fails to adapt to bigh changes in some parameters values, such as the blood volume. Also, different consistency and error check contributes to slow down the process of generating data. The lack of data is an aspect that also made impossible to further explore deep learning architectures such as the TFT;
- **Lack of medical expertise**: The lack of background knowledge in the medical field has made it difficult to contextualize our results in terms of how they relate to known and possibly correlations between different all the different physiological parameters taken into analysis.

Chapter 6

Ethics, future work and conclusions

6.1 Future work

This work can be a good starting point for further research in the field of cardiovascular parameter estimation but there are still some improvements that can be made. The first thing would be to increase the amount of generated data. For this job the amount of generated data was limited by the number of available presets in Aplysia. Generating completely random set of parameters would be for shure useless since we could end up with non realistic combination of model parameters but it can be interesting to apply the data-generation routine to generate new model parameter set starting from real patient data. In this way we can generate syntetic model parameters that resembles the starting real physiology. Having more data would enhance the use of the architectures that did not gave good results in this study. Other improvements can be made on the interpretability of the model. Since this work could be used as a component of a support decision tool it would help to avoid models whose inner workings are difficult to interpret. Modern and complex architecture are more suited to this type of analysis, such as the transformer-based architectures. Enhancing the use of those architectures by having a proper amount of data would be a huge step towards the interpretation of the outcomes since we could take a look at where the model is looking by plotting the attention maps [30]. Another step forward would be to turn this study into a real-time model that keep learns from the monitoring data. Currently this study focus on fixed patient state but it could happen that during the patient life-time in a ?? different diseases shows up and the patient shifts from one particular state to another. In order to keep the digital model coherent with the patient state we would need a model that is able to correct its prediction of the model parameters over time depending on

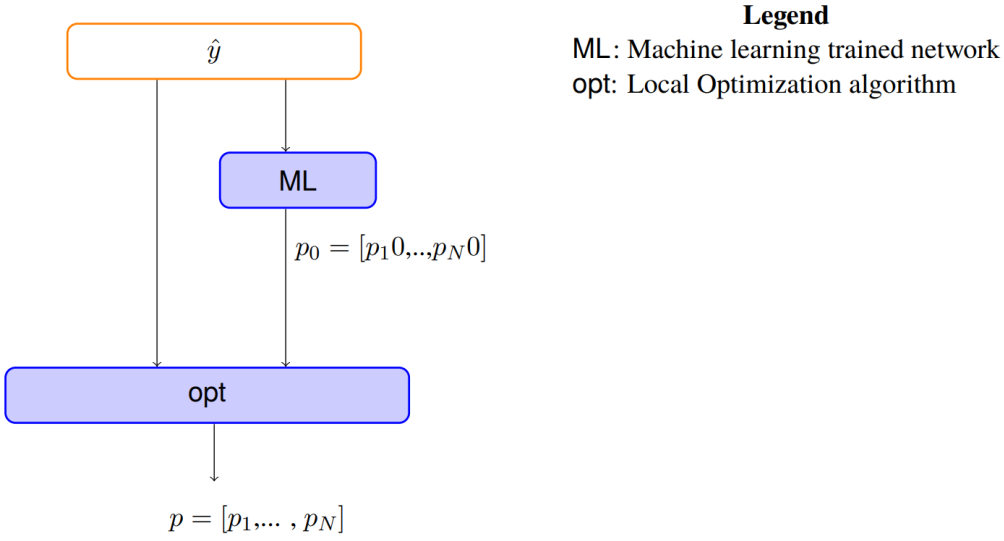


Figure 6.1: Scheme of the local optimization loop with the deployment of the machine learning model.

the variations of the input time signals. Some alternatives can be found in the literature to perform such things: incremental learning is a branch of deep learning that focus on architectures that are able to learn from new data without forgetting the previous knowledge acquired [51]. These framework could be useful to turn this work into a real-time patient monitoring system in which the patient model keeps get updated depending on the input data.
Another useful development of this project would be to generate and publish a dataset using Aplysia in order to enhance research on this topic. As inferred from the literature studies there are few articles that focus on predicting different parameters at the same time and this could be because of the absence of dataset that consent to develop and perform such experiments. Giving a dataset to the research community would be for sure helpful to further explore this field.
A proof on how the developed methods is worth to be explored more is its application for parameter optimization. The machine learning model trained and developed for this model has been used by Giulia Tuccio, a KTH student performing a Master's Thesis at Getinge AB in the field of parameter estimation through optimization. By using a trained machine learning model we can enhance the use of local optimizer, which ensure faster convergence during the optimization process. In particular, as we can notice from figure 6.1 the machine learning model gives an initial guess of the parameters to further optimize with a local optimizer: without an initial guess we need to use global strategies, which requires longer time to

converge.

6.2 Ethical considerations

Ethics is an issue to consider when it comes to driving medical decision using AI. One first question that may arise is how much do we trust a support decision tool based on a digital twin. it is important to point out that not all the decisions are critical but some are already a trial and error process, such as adrenaline injections. This means that it is safe to rely on such system for those decision but further care is needed for more critical ones. The system should have well-fixed margin of error in order to be used as a support tool for decision in ??. Another consistent problem that could affect this task is training models with dataset that may contain hidden biases. This problem comes from the nature of supervised learning since we let the models collect information from the data to make decisions. One of the risk is that there could be a lack of representation of a particular disease group and it would be problematic to say that we can generalize from one population to another. This could result in discriminating gender or ethnics too. In this thesis we did not included the gender in the dataset as information in order to mitigate any gender discrimination but the starting presets from Aplysia are mostly based on men patient. Although, the equations that describes the hemodynamics events that we have analyzed are mainly based on the body composition, hence we can consider them gender-neutral. Age bias is another bias we should be aware of, especially in this field. When comes to medical decision age is an important factor. For instance, older patient usually requires different types of treatment since they could have different non-cardiovascular diseases. Potentially, having a model trained on an unbalanced dataset (with respect to the age) can mislead the support system in making much more drastic decisions then needed. In our analysis the age was partially taken into account since 4 different age groups were generated from the healthy presets but this is not enough to affirm that we have taken measure to avoid any type of hidden bias. If we take a particular patient state the model should behave correctly disregarding the age of the patient, but we already saw that there is a group of control patient (adult 60y) that presents lower performances compared to the other control patients.

6.2.1 Sustainability

The models developed for this thesis work could be used as a part of a bedside decision-support framework. Those tool can both increase the efficiency of the decisions by giving insights and suggest the best treatments for a specific patient, as well as the use of specific medical devices [52]. If we look from the hospital perspective, treatments and medical devices are often expensive and different

criteria needs to be taken into account simultaneously. By automating this process, decision-support models can lower cost on the overall health system.
Other saving can be seen in energy, time and resources: AI-driven decision can automate processes that may have required human labour. Reducing these processes reduce human action and the waste with ensuring a clean and sterile environment for the healthcare standards. Also testing treatments virtually on a digital twin reduces the waste of drugs.

6.3 Conclusions

The purpose of this thesis was to explore different approaches to estimate a set of model parameters starting from time-dependant simulation output. The problem was formulated as a MTR task with MTS as input. To perform this work, a first literature study was conducted to find possible approaches for multi-input and multi-output problems in the medical field. The experiments then focused on comparing deep learning and machine learning models from the literature after adapting them to this specific task. One of the most important constraint was that no pre-existing dataset could be found in the literature and we had to generate the dataset by ourselves. This aspect limited the possible deep learning architectures that could have been deployed. After all the experiments and results evaluation, we can directly address the research questions. We can state that:

- The best approach that provided the smallest residual error from Aplysia output was the the machine learning features extraction and filtering approach. Other approaches that used the raw time-series did not produced any better results;

- The best framework in terms of MTR scores was the machine learning pipeline based on a multi-output gradient boosting regressor. It performed best both in terms of MSE and aRRMSE. We can appreciate the goodness in prediction especially from the distribution visualization in figure 4.6. In general, all the model developed are generally better than a random baseline regressor: this aspect tell us that all the model evaluated were purposfully implemented and produced useful results;

- The robustness of the algorithm was tested by varying the set of input monitoring data adding the weight and length information. This analysis was led to prove that the best model was actually predicting the output model parameters or it was basing its prediction on an estimation of the body dimension.

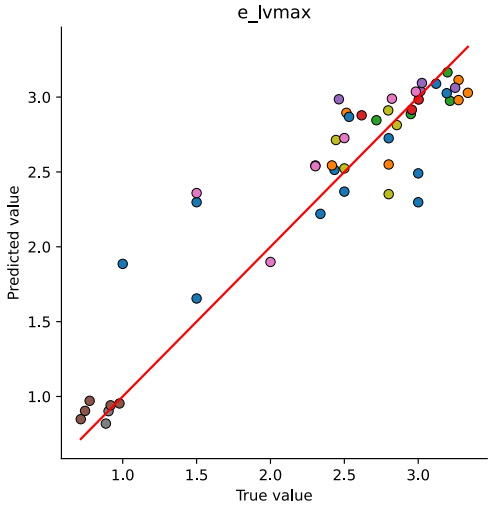
The multi-output gradient boosting regressor by itself can be profitable but in order to be deployed in this field further studies on reliability needs to be done since we

need wide safe margin when it comes to critical decision. The work of this thesis can be considered a good starting point for future research and improvements as described in section 6.1.

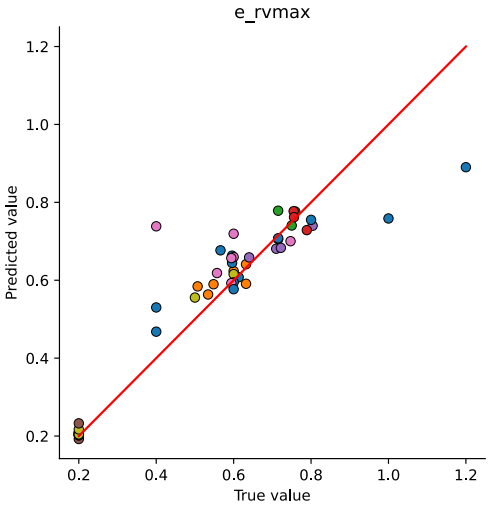
Appendix A

Visualizations

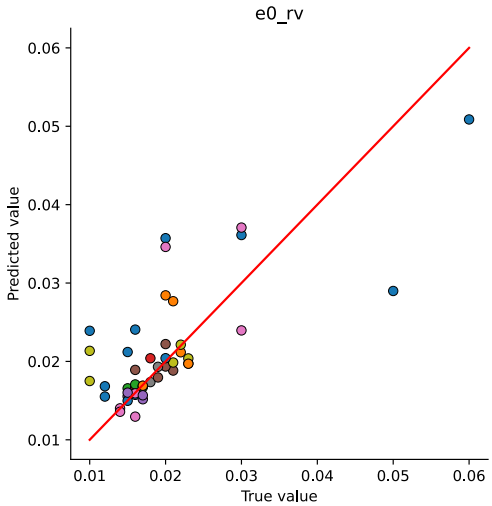
The following section contains the performance visualizations of the Gradient Boosting Regressor model compared with the KNN regressor model on the test data on the five parameters left out during the results discussion.



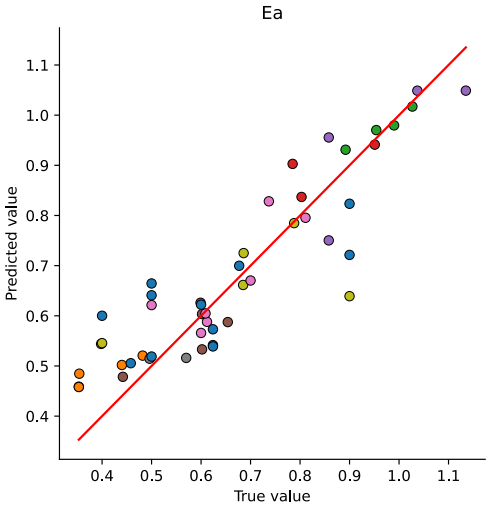
(a) Left ventricular contractility.



(b) Right ventricular contractility.

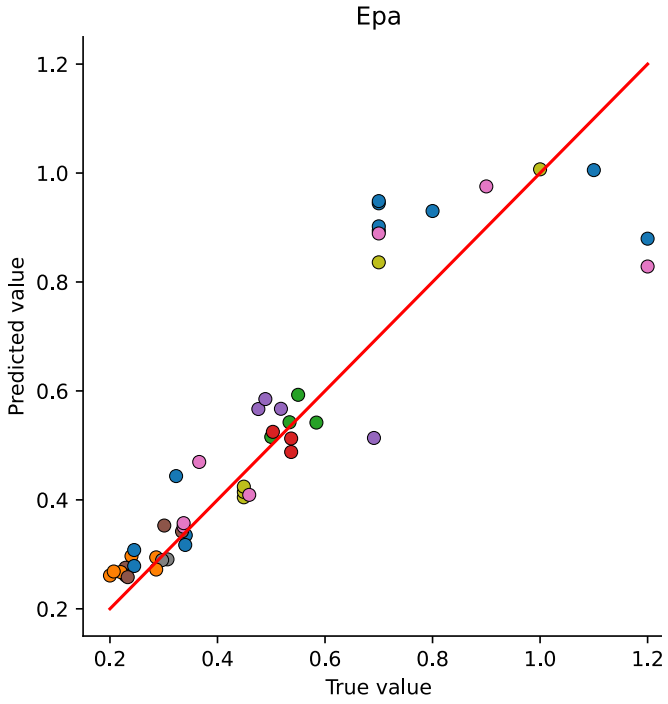


(c) Right ventricular stiffness.



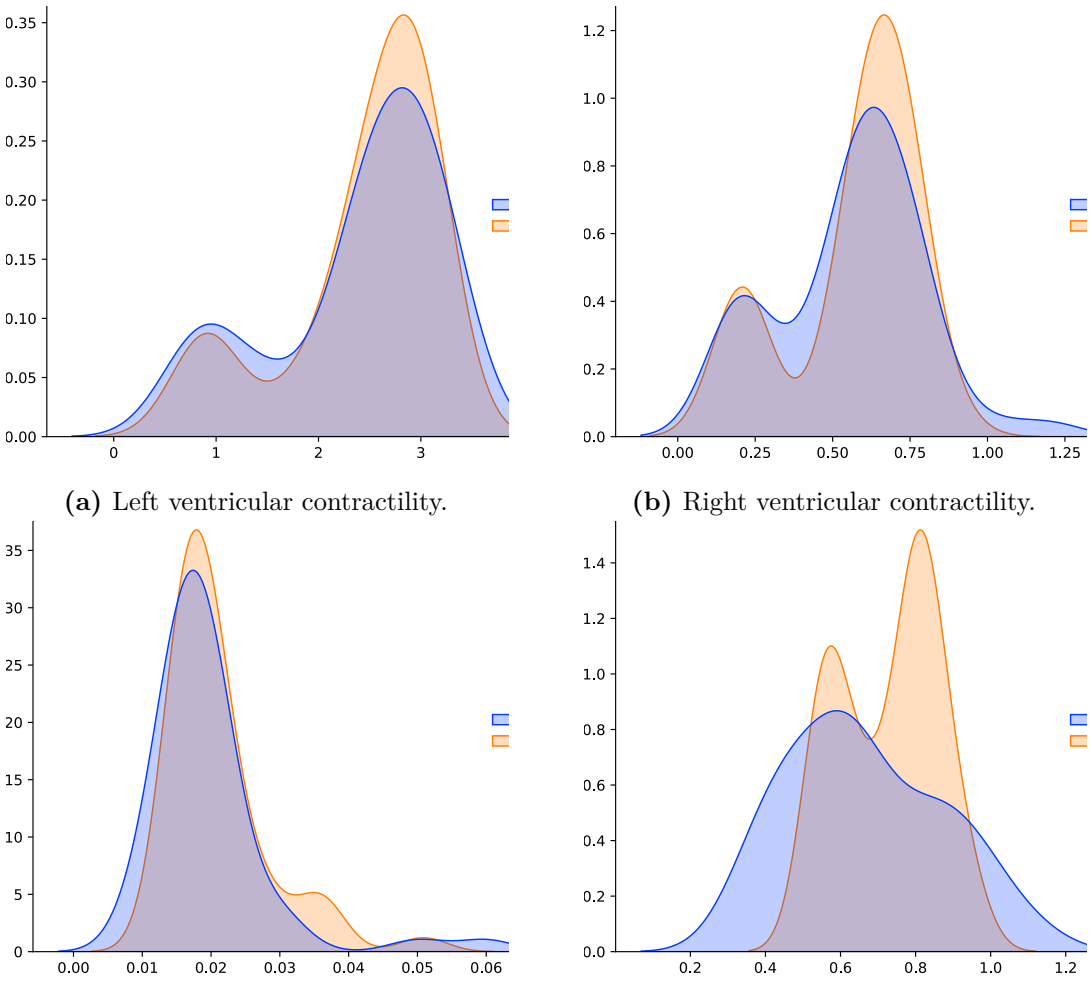
(d) Systemic arterial stiffness.

Figure A.1: Scatter plots of the test data, color coded by starting preset. Gradient boosting regressor. Legend can be found in 4.4.



(a) Pulmonary arterial stiffness.

Figure A.2: Scatter plots of the test data, color coded by starting preset. Gradient boosting regressor. Legend can be found in 4.4.



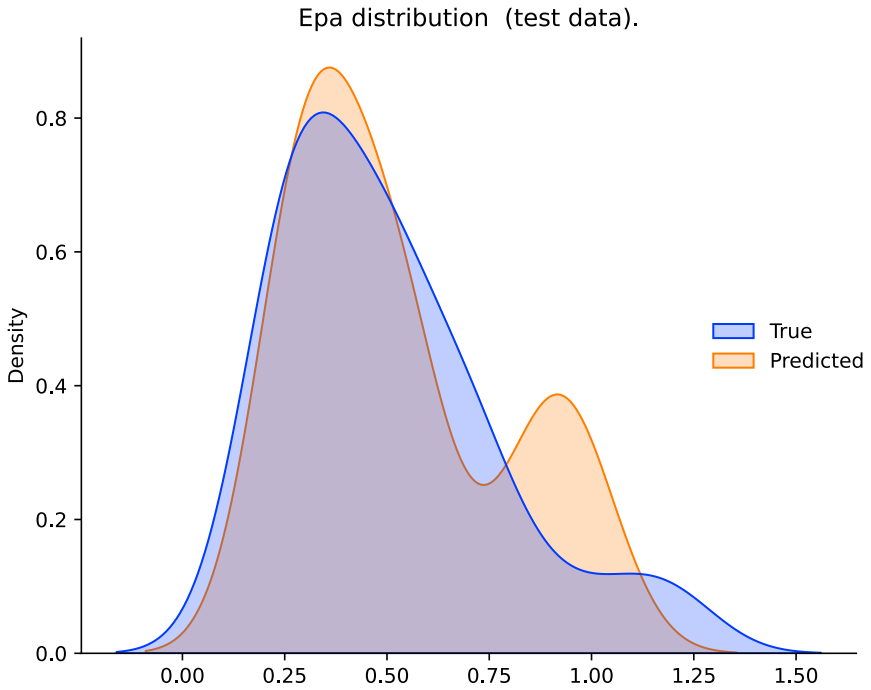
(a) Left ventricular contractility.

(b) Right ventricular contractility.

(c) Right ventricular stiffness.

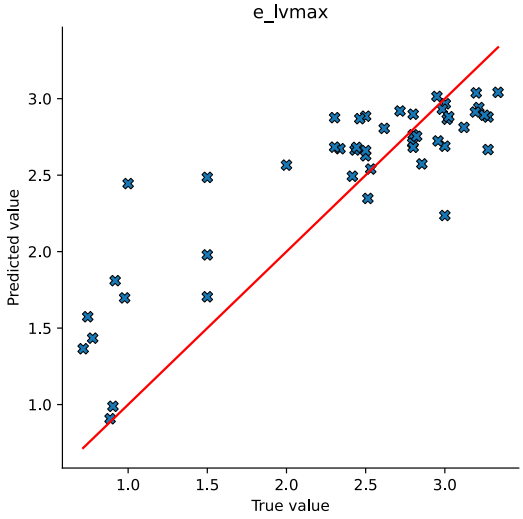
(d) Systemic arterial stiffness.

Figure A.3: Distribution plots of the test data displaying the true (blue) and predicted (orange) distribution.

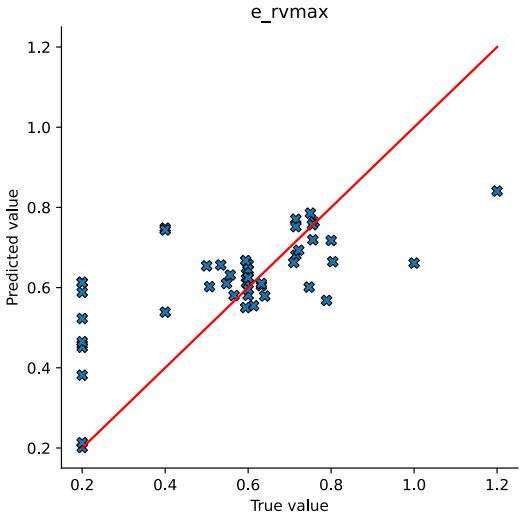


(a) Pulmonary arterial stiffness.

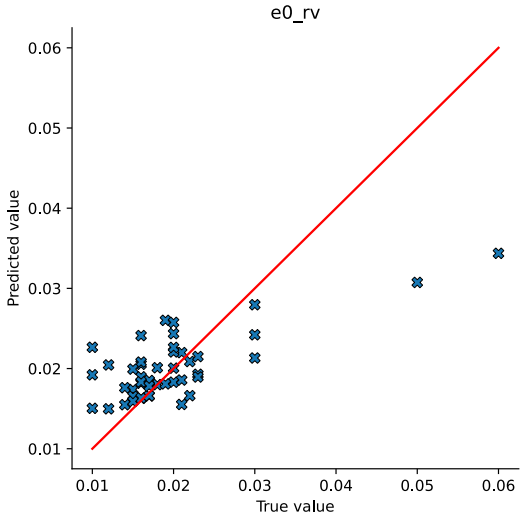
Figure A.4: Distribution plots of the test data displaying the true (blue) and predicted (orange) distribution.



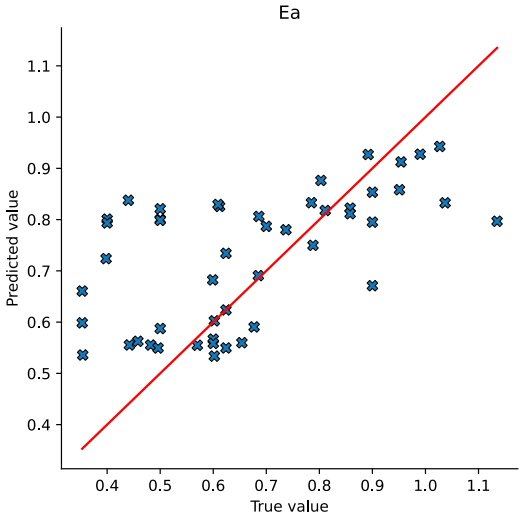
(a) Left ventricular contractility.



(b) Right ventricular contractility.

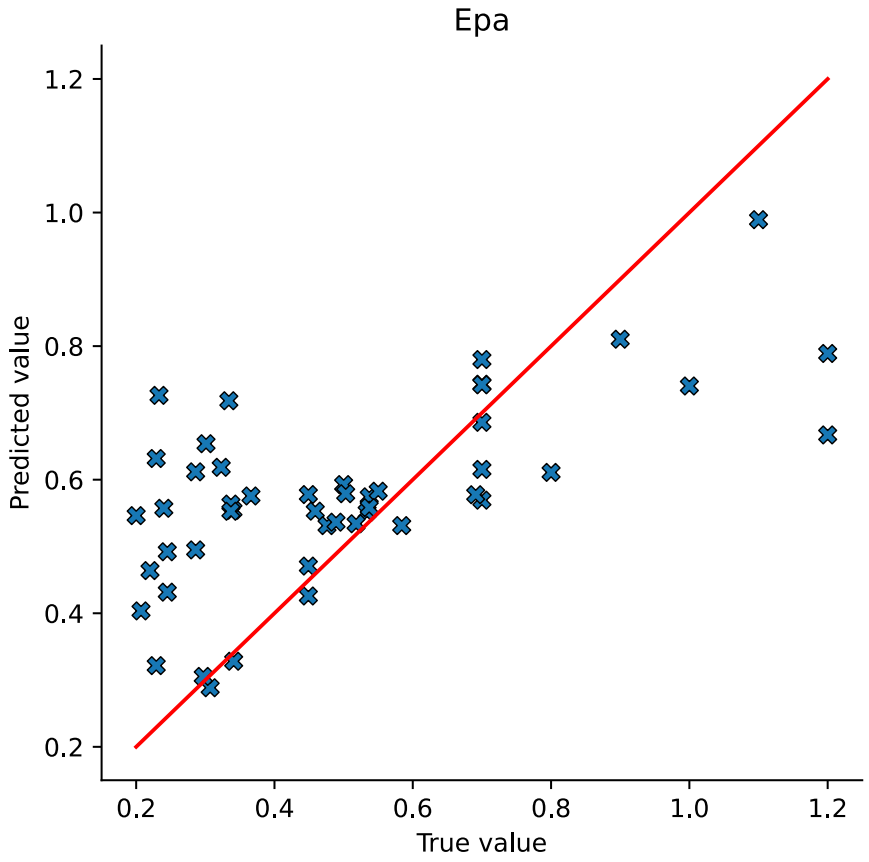


(c) Right ventricular stiffness.



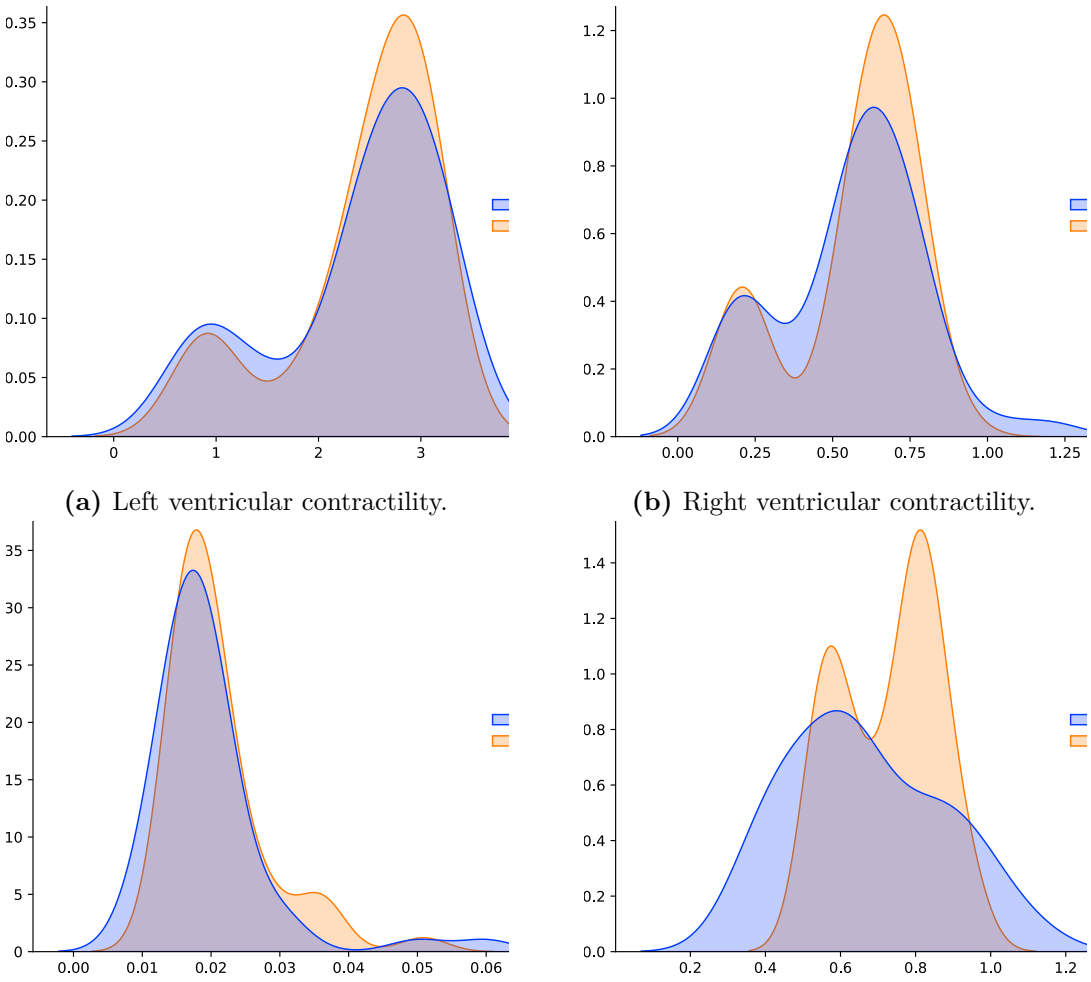
(d) Systemic arterial stiffness.

Figure A.5: Scatter plots of the test data, KNN Regressor.



(a) Pulmonary arterial stiffness.

Figure A.6: Scatter plots of the test data, KNN Regressor.



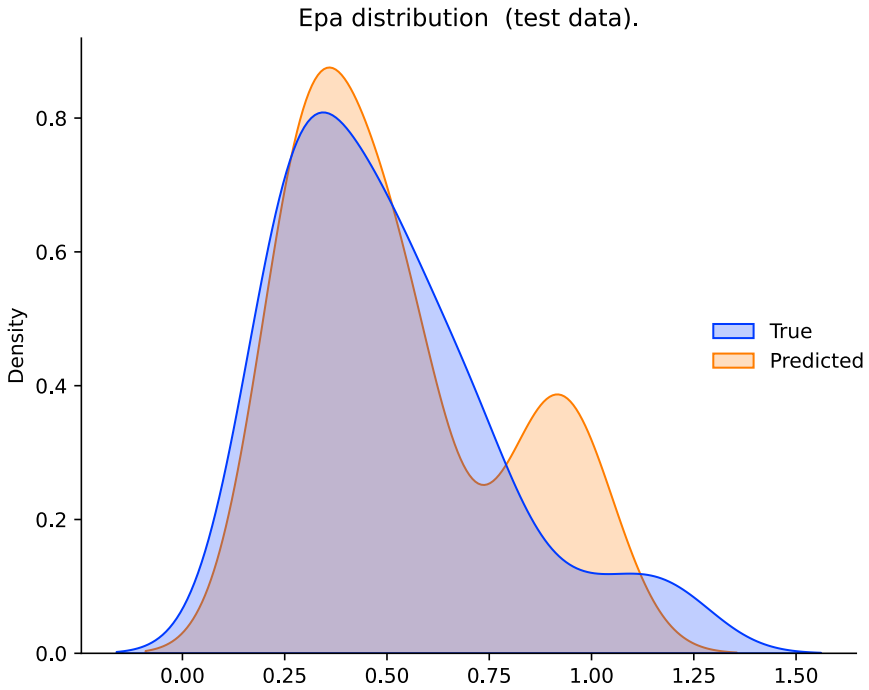
(a) Left ventricular contractility.

(b) Right ventricular contractility.

(c) Right ventricular stiffness.

(d) Systemic arterial stiffness.

Figure A.7: Distribution plots of the test data displaying the true (blue) and predicted (orange) distribution for the KNN regressor.



(a) Pulmonary arterial stiffness.

Figure A.8: Distribution plots of the test data displaying the true (blue) and predicted (orange) distribution for the KNN regressor.

Bibliography

[1] L. Fresiello et al. «A Cardiovascular Simulator Tailored for Training and Clinical Uses». In: *Journal of Biomedical Informatics* 57 (Oct. 2015), pp. 100–112. ISSN: 1532-0480. DOI: 10.1016/j.jbi.2015.07.004 (cit. on pp. 1, 8).

[2] Michael Broomé, Elira Maksuti, Anna Bjällmark, Björn Frenckner, and Birgitta Janerot-Sjöberg. «Closed-loop real-time simulation model of hemodynamics and oxygen transport in the cardiovascular system». eng. In: *Biomedical Engineering Online* 12 (July 2013), p. 69. ISSN: 1475-925X. DOI: 10.1186/1475-925X-12-69 (cit. on p. 1).

[3] Evgeniya Korneva and Hendrik Blockeel. «Towards Better Evaluation of Multi-target Regression Models». In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2020, pp. 353–362 (cit. on p. 4).

[4] Nicolaas Westerhof, Nikolaos Stergiopulos, Mark I.M. Noble, and Berend E. Westerhof. *Snapshots of Hemodynamics: An Aid for Clinical Research and Graduate Education*. Cham: Springer International Publishing, 2019. ISBN: 978-3-319-91931-7 978-3-319-91932-4. DOI: 10.1007/978-3-319-91932-4 (cit. on p. 6).

[5] Alistair E. W. Johnson et al. «MIMIC-III, a Freely Accessible Critical Care Database». In: *Scientific Data* 3.1 (May 2016), p. 160035. ISSN: 2052-4463. DOI: 10.1038/sdata.2016.35 (cit. on p. 8).

[6] Anthony Bagnall, Jason Lines, Aaron Bostrom, James Large, and Eamonn Keogh. «The Great Time Series Classification Bake off: A Review and Experimental Evaluation of Recent Algorithmic Advances». In: *Data Mining and Knowledge Discovery* 31.3 (May 2017), pp. 606–660. ISSN: 1573-756X. DOI: 10.1007/s10618-016-0483-9 (cit. on pp. 9, 11, 37).

[7] Peter J. Brockwell and Richard A. Davis. «Introduction to Time Series and Forecasting». In: *Introduction to Time Series and Forecasting*. Ed. by Peter J. Brockwell and Richard A. Davis. Springer Texts in Statistics. Cham: Springer International Publishing, 2016, pp. 227–257. ISBN: 978-3-319-29854-2. DOI: 10.1007/978-3-319-29854-2_8 (cit. on p. 10).

[8] Ruofeng Wen, Kari Torkkola, Balakrishnan Narayanaswamy, and Dhruv Madeka. «A Multi-Horizon Quantile Recurrent Forecaster». In: (Nov. 2017). DOI: 10.48550/arXiv.1711.11053 (cit. on p. 10).

[9] Bryan Lim, Sercan O. Arik, Nicolas Loeff, and Tomas Pfister. «Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting». In: *arXiv:1912.09363 [cs, stat]* (Sept. 2020). arXiv: 1912.09363 [cs, stat] (cit. on pp. 10, 15).

[10] Christian Bock, Michael Moor, Catherine R. Jutzeler, and Karsten Borgwardt. «Machine Learning for Biomedical Time Series Classification: From Shapelets to Deep Learning». In: *Artificial Neural Networks*. Ed. by Hugh Cartwright. Methods in Molecular Biology. New York, NY: Springer US, 2021, pp. 33–71. ISBN: 978-1-07-160826-5. DOI: 10.1007/978-1-0716-0826-5_2 (cit. on p. 10).

[11] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh. «Addressing Big Data Time Series: Mining Trillions of Time Series Subsequences Under Dynamic Time Warping». In: *ACM Transactions on Knowledge Discovery from Data* 7.3 (Sept. 2013), 10:1–10:31. ISSN: 1556-4681. DOI: 10.1145/2500489 (cit. on p. 11).

[12] André Maletzke, Carlos Ferrero, Chris Tibes, Everton Cherman, and Willian Zalewski. «Medical Time Series Classification Using Global and Local Feature Extraction Strategies». In: *Journal of Health Informatics* 9 (Aug. 2017), p. 73 (cit. on p. 11).

[13] Pei-Yuan Zhou and Keith C. C. Chan. «A Feature Extraction Method for Multivariate Time Series Classification Using Temporal Patterns». In: *Advances in Knowledge Discovery and Data Mining*. Ed. by Tru Cao, Ee-Peng Lim, Zhi-Hua Zhou, Tu-Bao Ho, David Cheung, and Hiroshi Motoda. Vol. 9078. Cham: Springer International Publishing, 2015, pp. 409–421. ISBN: 978-3-319-18031-1 978-3-319-18032-8. DOI: 10.1007/978-3-319-18032-8_32 (cit. on p. 11).

[14] Martin Längkvist, Lars Karlsson, and Amy Loutfi. «A Review of Unsupervised Feature Learning and Deep Learning for Time-Series Modeling». In: *Pattern Recognition Letters* 42 (June 2014), pp. 11–24. ISSN: 0167-8655. DOI: 10.1016/j.patrec.2014.01.008 (cit. on p. 12).

[15] Deepta Rajan and Jayaraman J. Thiagarajan. «A Generative Modeling Approach to Limited Channel ECG Classification». In: *arXiv:1802.06458 [cs, stat]* (June 2018). arXiv: 1802.06458 [cs, stat] (cit. on pp. 12, 14).

[16] Hanen Borchani, Gherardo Varando, Concha Bielza, and Pedro Larrañaga. «A Survey on Multi-Output Regression: Multi-output Regression Survey». In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 5.5 (Sept. 2015), pp. 216–233. ISSN: 19424787. DOI: 10.1002/widm.1157 (cit. on p. 12).

[17] M. Sanchez-Fernandez, M. de-Prado-Cumplido, J. Arenas-Garcia, and F. Perez-Cruz. «SVM Multiregression for Nonlinear Channel Estimation in Multiple-Input Multiple-Output Systems». In: *IEEE Transactions on Signal Processing* 52.8 (Aug. 2004), pp. 2298–2307. ISSN: 1941-0476. DOI: 10.1109/TSP.2004.831028 (cit. on p. 13).

[18] Dragi Kocev, Sašo Džeroski, Matt D. White, Graeme R. Newell, and Peter Griffioen. «Using Single- and Multi-Target Regression Trees and Ensembles to Model a Compound Index of Vegetation Condition». In: *Ecological Modelling* 220.8 (Apr. 2009), pp. 1159–1168. ISSN: 0304-3800. DOI: 10.1016/j.ecolmodel.2009.01.037 (cit. on p. 13).

[19] Esmaeil Hadavandi, Jamal Shahrabi, and Shahaboddin Shamshirband. «A Novel Boosted-neural Network Ensemble for Modeling Multi-Target Regression Problems». In: *Engineering Applications of Artificial Intelligence* 45 (Oct. 2015), pp. 204–219. ISSN: 0952-1976. DOI: 10.1016/j.engappai.2015.06.022 (cit. on p. 13).

[20] Maksat Ashyraliyev, Yves Fomekong-Nanfack, Jaap A. Kaandorp, and Joke G. Blom. «Systems Biology: Parameter Estimation for Biochemical Models». In: *The FEBS journal* 276.4 (Feb. 2009), pp. 886–902. ISSN: 1742-4658. DOI: 10.1111/j.1742-4658.2008.06844.x (cit. on p. 13).

[21] Kenji Kawaguchi. «Deep Learning without Poor Local Minima». In: *Advances in Neural Information Processing Systems*. Vol. 29. Curran Associates, Inc., 2016 (cit. on p. 14).

[22] Arun U. Nair, David G. Taggart, and Frederick J. Vetter. «Optimizing Cardiac Material Parameters with a Genetic Algorithm». In: *Journal of Biomechanics* 40.7 (Jan. 2007), pp. 1646–1650. ISSN: 00219290. DOI: 10.1016/j.jbiomech.2006.07.018 (cit. on p. 14).

[23] Andrea Degasperi, Dirk Fey, and Boris N. Kholodenko. «Performance of Objective Functions and Optimisation Procedures for Parameter Estimation in System Biology Models». In: *npj Systems Biology and Applications* 3.1 (Dec. 2017), p. 20. ISSN: 2056-7189. DOI: 10.1038/s41540-017-0023-2 (cit. on pp. 14, 20).

[24] Hugo Alonso, Teresa Mendonça, and Paula Rocha. «A Hybrid Method for Parameter Estimation and Its Application to Biomedical Systems». In: *Computer Methods and Programs in Biomedicine*. The 6th IFAC Symposium on Modelling and Control in Biomedical Systems 89.2 (Feb. 2008), pp. 112–122. ISSN: 0169-2607. DOI: 10.1016/j.cmpb.2007.10.014 (cit. on p. 14).

[25] Hubert Cardot. *Recurrent Neural Networks for Temporal Data Processing*. Feb. 2011. ISBN: 978-953-307-685-0. DOI: 10.5772/631 (cit. on p. 14).

[26] Sepp Hochreiter and Jürgen Schmidhuber. «Long Short-term Memory». In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735 (cit. on p. 14).

[27] Saeed Saadatnejad, Mohammadhosein Oveisi, and Matin Hashemi. «LSTM-Based ECG Classification for Continuous Monitoring on Personal Wearable Devices». In: *IEEE Journal of Biomedical and Health Informatics* 24.2 (Feb. 2020), pp. 515–523. ISSN: 2168-2208. DOI: 10.1109/JBHI.2019.2911367 (cit. on p. 14).

[28] Alaa Sagheer and Mostafa Kotb. «Unsupervised Pre-training of a Deep LSTM-based Stacked Autoencoder for Multivariate Time Series Forecasting Problems». In: *Scientific Reports* 9.1 (Dec. 2019), p. 19038. ISSN: 2045-2322. DOI: 10.1038/s41598-019-55320-6 (cit. on p. 14).

[29] Madhuri Panwar, Arvind Gautam, Dwaipayan Biswas, and Amit Acharyya. «PP-Net: A Deep Learning Framework for PPG-Based Blood Pressure and Heart Rate Estimation». In: *IEEE Sensors Journal* 20.17 (Sept. 2020), pp. 10000–10011. ISSN: 1558-1748. DOI: 10.1109/JSEN.2020.2990864 (cit. on p. 15).

[30] Xixi Li, Yanfei Kang, and Feng Li. «Forecasting with Time Series Imaging». In: *Expert Systems with Applications* 160 (Dec. 2020), p. 113680. ISSN: 09574174. DOI: 10.1016/j.eswa.2020.113680. arXiv: 1904.08064 (cit. on pp. 15, 57).

[31] Zhiguang Wang and Tim Oates. «Imaging Time-Series to Improve Classification and Imputation». In: (), p. 7 (cit. on p. 15).

[32] YueZijie, DingShuai, ZhaoLei, ZhangYoutao, CaoZehong, TanveerM, JolfaeiAlireza, and ZhengXi. «Privacy-Preserving Time-series Medical Images Analysis Using a Hybrid Deep Learning Framework». In: *ACM Transactions on Internet Technology (TOIT)* (June 2021). DOI: 10.1145/3383779 (cit. on p. 15).

[33] S. Raghu, Natarajan Sriraam, Yasin Temel, Shyam Vasudeva Rao, and Pieter L. Kubben. «EEG Based Multi-Class Seizure Type Classification Using Convolutional Neural Network and Transfer Learning». In: *Neural Networks* 124 (Apr. 2020), pp. 202–212. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2020.01.017 (cit. on p. 16).

[34] Eleftherios Spyromitros-Xioufis, Grigorios Tsoumakas, William Groves, and Ioannis Vlahavas. «Multi-Target Regression via Input Space Expansion: Treating Targets as Inputs». In: *Machine Learning* 104.1 (July 2016), pp. 55–98. ISSN: 0885-6125, 1573-0565. DOI: 10.1007/s10994-016-5546-z. arXiv: 1211.6581 (cit. on p. 17).

[35] Yi Zheng, Qi Liu, Enhong Chen, Yong Ge, and J. Leon Zhao. «Time Series Classification Using Multi-Channels Deep Convolutional Neural Networks». In: *Web-Age Information Management*. Ed. by Feifei Li, Guoliang Li, Seungwon Hwang, Bin Yao, and Zhenjie Zhang. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2014, pp. 298–310. ISBN: 978-3-319-08010-9. DOI: 10.1007/978-3-319-08010-9_33 (cit. on p. 18).

[36] Kang Gu, Soroush Vosoughi, and Temiloluwa Prioleau. «Feature Selection for Multivariate Time Series via Network Pruning». In: *2021 International Conference on Data Mining Workshops (ICDMW)* (Dec. 2021), pp. 1017–1024. DOI: 10.1109/ICDMW53433.2021.00132. arXiv: 2102.06024 (cit. on p. 18).

[37] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. «Deep Residual Learning for Image Recognition». In: *arXiv:1512.03385 [cs]* (Dec. 2015). arXiv: 1512.03385 [cs] (cit. on p. 18).

[38] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. «Data Augmentation Using Synthetic Data for Time Series Classification with Deep Residual Networks». In: *arXiv:1808.02455 [cs]* (Aug. 2018). arXiv: 1808.02455 [cs] (cit. on p. 18).

[39] Zhiguang Wang, Weizhong Yan, and Tim Oates. «Time Series Classification from Scratch with Deep Neural Networks: A Strong Baseline». In: *arXiv:1611.06455 [cs, stat]* (Dec. 2016). arXiv: 1611.06455 [cs, stat] (cit. on pp. 18, 35).

[40] Sabine Geiβ and Jürgen Einax. «Multivariate Correlation Analysis - a Method for the Analysis of Multidimensional Time Series in Environmental Studies». In: *Chemometrics and Intelligent Laboratory Systems* 32.1 (Feb. 1996), pp. 57–65. ISSN: 0169-7439. DOI: 10.1016/0169-7439(95)00067-4 (cit. on pp. 21, 54).

[41] Laurens van der Maaten and Geoffrey Hinton. «Visualizing Data Using T-SNE». In: *Journal of Machine Learning Research* 9.86 (2008), pp. 2579–2605. ISSN: 1533-7928 (cit. on p. 26).

[42] F. Pedregosa et al. «Scikit-learn: Machine Learning in Python». In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on p. 27).

[43] *Time Series FeatuRe Extraction on Basis of Scalable Hypothesis Tests (Tsfresh – A Python Package) - ScienceDirect* (cit. on p. 28).

[44] Yoav Benjamini and Daniel Yekutieli. «The Control of the False Discovery Rate in Multiple Testing under Dependency». In: *The Annals of Statistics* 29.4 (Aug. 2001), pp. 1165–1188. ISSN: 0090-5364, 2168-8966. DOI: 10.1214/aos/1013699998 (cit. on p. 29).

[45] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/ (cit. on p. 33).

[46] Adam Paszke et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. Dec. 2019. DOI: 10.48550/arXiv.1912.01703. arXiv: 1912.01703 [cs, stat] (cit. on p. 33).

[47] Ignacio Oguiza. *tsai - A state-of-the-art deep learning library for time series and sequential data*. Github. 2022. URL: https://github.com/timeseriesAI/tsai (cit. on p. 33).

[48] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. «Dropout: A Simple Way to Prevent Neural Networks from Overfitting». In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. ISSN: 1533-7928 (cit. on p. 34).

[49] Dina Q. Goldin and Paris C. Kanellakis. «On Similarity Queries for Time-Series Data: Constraint Specification and Implementation». In: *Principles and Practice of Constraint Programming — CP '95*. Ed. by Ugo Montanari and Francesca Rossi. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1995, pp. 137–153. ISBN: 978-3-540-44788-7. DOI: 10.1007/3-540-60299-2_9 (cit. on p. 35).

[50] Qingsong Wen, Liang Sun, Fan Yang, Xiaomin Song, Jingkun Gao, Xue Wang, and Huan Xu. «Time Series Data Augmentation for Deep Learning: A Survey». In: *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*. Aug. 2021, pp. 4653–4660. DOI: 10.24963/ijcai.2021/631. arXiv: 2002.12478 [cs, eess, stat] (cit. on p. 53).

[51] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H. Lampert. *iCaRL: Incremental Classifier and Representation Learning*. Apr. 2017. DOI: 10.48550/arXiv.1611.07725. arXiv: 1611.07725 [cs, stat] (cit. on p. 58).

[52] Nicolas Martelli, Paul Hansen, Hélène van den Brink, Aurélie Boudard, Anne-Laure Cordonnier, Capucine Devaux, Judith Pineau, Patrice Prognon, and Isabelle Borget. «Combining Multi-Criteria Decision Analysis and Mini-Health Technology Assessment: A Funding Decision-Support Tool for Medical Devices in a University Hospital Setting». In: *Journal of Biomedical Informatics* 59 (Feb. 2016), pp. 201–208. ISSN: 1532-0480. DOI: 10.1016/j.jbi.2015.12.002 (cit. on p. 59).