Politecnico di Torino

ScuDo
Scuola di Dottorato ~ Doctoral School
WHAT YOU ARE, TAKES YOU FAR

Doctoral Dissertation
Doctoral Program in Metrology (34^{th}cycle)

Deep Learning Methods for Industry and Healthcare

By

Maria Amodeo

******

Supervisor(s):
Prof. M. Parvis, Supervisor
Prof. P. Arpaia, Co-supervisor
Prof. R. Prevete, Co-supervisor

Doctoral Examination Committee:
Prof.ssa Cristaldi Loredana, Referee, Politecnico di Milano
Dr. Petrone Carlo, Referee, CERN
Prof.ssa Grassini Sabrina, Politecnico di Torino
Prof. Isgrò Francesco, Università di Napoli Federico II
Ing. Zorzetti Silvia, Fermilab

Politecnico di Torino
2022

Declaration

I hereby declare that, the contents and organization of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

Maria Amodeo
2022

* This dissertation is presented in partial fulfillment of the requirements for **Ph.D. degree** in the Graduate School of Politecnico di Torino (ScuDo).

To you, who are part of me.

Acknowledgements

A special acknowledgement is addressed to my supervisor, Professor Marco Parvis, for his availability, ideas and precious advice. He was for me an important point of reference in the metrology field.
I would like to express my gratefulness to Professor Pasquale Arpaia who supervised my research activities with precious ideas and suggestions. I thank him to be an incomparable guide, always available to support me.
A special acknowledgement to Professor Roberto Prevete who tightly supervised the progress of my research, providing me the keys to better comprehend deep learning techniques.
I would like to thank all the people of CERN Magnetic Measurement section and, in particular, Dr. Marco Buzio, who supported my research with his impressive knowledge of magnetic measurements. I thank him for his invaluable contribution to my research activities. I would also like to reserve a special thanks to the Btrain team, which has allowed me to grow both professionally and personally. I also thank all the colleagues from the different CERN departments I collaborated with.
I thank the ARHeMLab and Impalab group for the fruitful discussions that have been the starting point of ideas, contributing enormously to the realization of this manuscript.
I would like to thank Dr. Francesco Donnarumma for sharing his invaluable knowledge in the field of deep learning, always offering interesting food for thought.
Special gratitude is reserved to the medical team with whom I collaborated for the realization of the maxillofacial fracture detection system. In particular, I would like to thank Dr. Renato Cuocolo and Dr. Lorenzo Ugga, for their availability and their technical support.
I would like to thank all my dear friends, near and far, who have always supported me and never made me feel the weight of the distance.

Finally, I would like to thank my lovely family and Federico for always being by my side, for supporting my decisions and for teaching me to believe in myself.

Abstract

The research activity discussed in the following manuscript is based on the design, implementation and testing of deep learning architectures for two different contexts: Industry and Healthcare. In the first case, the research activity was carried out at the European Organization for Nuclear Research (CERN) with the aim to model the magnetic hysteresis in the context of electromagnets. The main goal of physicists and engineers working at CERN is the study of high-energy physics. In order to understand the fundamental structure of matter, the particles are made to collide together at a very high speed, close to the speed of light. At this aim, a chain of interconnected accelerators allows the acceleration of charged particles up to 6.5 TeV (Tera electron Volt) per beam in the Large Hadron Collider (LHC). In particle accelerators, the beam circulates in a ring of magnets and is accelerated by the radio frequency cavities. The magnets produce a bending magnetic field that rises in proportion to the beam momentum. Hence, a precise knowledge of magnetic field values is crucial for various subsystems, especially for beam control. Consequently, there is a strong motivation to investigate new models to complement or even replace measurements, when the latter is not available. At this aim, a deep neural network architecture has been proposed for predicting the magnetic field in iron-dominated magnets. According to the results, the proposed architecture is the one that fits the measured field most closely, with a percent error below 0.02 %. These outcomes outperform the current approaches present in the literature and are very encouraging for forthcoming real-time applications.
The second case study regards the healthcare context and it was carried out in collaboration with the University Hospital Federico II, which has provided the dataset consisting of 170 anonymized exams of patients with fracture and 38 anonymized exams of patients without fracture. The assessment of exams, consisting of computed tomography images in trauma patients, is fundamental to adopt the pertinent therapy and conduct them towards highly specialized centers. Often, patients are directed to

the regional reference center specialized in maxillofacial trauma even if they do not need surgical treatment. This causes incongruous hospitalizations, leading patients to a condition of discomfort and stress. For this purpose, a deep neural network was designed to detect fractures in the maxillofacial region of injured patients. The tests show the network capability of distinguishing between fractured and normal bone with an accuracy of 80 %. Even if the automated detection system cannot replace medical staff, it can contribute to validate radiologists' doubts and decisions, representing a useful tool to prevent patient injury by minimizing diagnostic delays and incongruous hospitalization.

Contents

List of Figures

List of Tables

Nomenclature

Acronyms / Abbreviations

AD Antiproton Decelerator

AI Artificial Intelligence

AIC Akaike Information Criterion

ANOVA ANalysis Of VAriance

AUC – ROC Area Under the Receiver Operating Characteristic Curve

BC Bridge Criterion

BIC Bayesian Information Criterion

CNAO National Centre for Oncological Hadrontherapy

CNN Convolutional Neural Network

CT Computed Tomography

CTR Central Timing Receiver

ELENA Extra Low ENergy Antiproton deceleration ring

FEC Front End Computer

FESA Front End Software Architecture

FIRESTORM Field In REal-time STreaming from Online Reference Magnets

FMC FPGA Mezzanine Card

HIT Heidelberg Ion-Beam Therapy Centre

ICML International Conference on Machine Learning

ILSVRC ImageNet Large Scale Visual Recognition

LEIR Low Energy Ion Ring

LHC Large Hadron Collider

LM Levenberg-Marquardt

LR Linear Regression

LSA LHC Software Architecture

LSTM Long Short-Term Memory

MAE Maximum Absolute Error

MFDS Maxillofacial Fracture Detection System

MLP Multilayer Perceptron

MPE Maximum Percent Error

MRI Magnetic Resonance Imaging

NARX Nonlinear Autoregressive Exogenous Neural Network

NeurIPS Conference on Neural Information Processing Systems

NRMSE Normalized Root Mean Square Error

OBR Optical Barcode Recognition

OCR Optical Character Recognition

OHWR Open Hardware Repository

PPM Pulse-to-Pulse Modulation

PS Proton Synchrotron

PSB Proton Synchrotron Booster

ReLU Rectified Linear Unit

ResNet Residual Network

RGB Red, Green and Blue

RMSE Root Mean Square Error

RNN Recurrent Neural Network

RNN Recurrent Neural Network

SGD Stochastic Gradient Descent

SPS Super Proton Synchrotron

TDNN Time Delay Neural Network

WR White Rabbit

Introduction

Nowadays deep learning techniques are widely used both in Industry and Healthcare fields and they are one of the main sources of success for artificial intelligence systems. Deep learning is a sub-category of machine learning and indicates a branch of Artificial Intelligence (AI) that refers to algorithms inspired by biological neural networks, therefore called artificial neural networks. By means of sophisticated algorithms, which are capable of automatically examining data such as audio, video, images or time series, deep learning is experiencing years of rapid progress, allowing, in some cases, to exceed the performance of human beings. Deep learning techniques are widely used in fields such as image classification (e.g. by means of objects identification in the image), autonomous driving (e.g. vehicles are able to identify dangerous situations in the surrounding area), language recognition and processing (e.g. real-time translation), security (e.g. facial recognition and video surveillance), medical diagnosis (e.g. cancer cell, fracture or drug detection).

In this thesis, two cases studies are presented. As regards Industry, deep learning methods are introduced to model the hysteretic behaviour of iron-dominated magnets, available at CERN. The modelling of hysteresis loops is a challenging theme in the field of computational magnetism, as a consequence of the non-linearity and historical dependence of ferromagnetic materials. In this context, a multi-layered neural network based on past values of input and output feedback is proposed to model the hysteretic behaviour of iron-dominated magnets. Test results prove that the presented NARX model is able to fit the quadrupole measured field, guaranteeing a percent error below 0.02 %.
The second case study regards the healthcare context. In this case, deep learning techniques are introduced to realize an automated detection system of fractures in patients with maxillofacial trauma. The possibility of having a fracture detection

system based on AI capable to find out the presence of fractures in injured patients would be of great help in clinical practice. What often happens is that, if there is no need for urgent treatment in the craniofacial region, injured patients are transferred from the emergency room to the nearest center specialized in maxillofacial traumatology. Here, the reassessment of the clinical case in a specialist setting frequently points out the incongruity of hospitalization with often a lack of indications for possible surgery. These patients necessitate only home medical treatments. In this context, an automated detection system would therefore reduce treatment costs and physical and mental distress for patients. The proposed system for maxillofacial fracture detection is able to recognize fractured and normal bone in computed tomography scans of harmed patients with an accuracy of 80 %. This result is particularly encouraging given the thinness of the bones and the anatomical complexity of the splanchnocranium area and it demonstrates the feasibility of using deep learning techniques for the detection of maxillofacial fractures on computed tomography images.
Part of the work described in this thesis was also previously published in [1] and [5].
The manuscript is divided in the following four chapters:

- in Chapter I, an overview on the impact of deep learning techniques in Industry and Healthcare context has been presented;
- in Chapter II, two main application contexts in which deep learning models can be employed have been analyzed; then, the learning process of a neural network architecture has been described; furthermore, particular attention has been paid to the architectures that play a fundamental role in deep learning. Finally, transfer learning, of fundamental importance in particular scenarios, has been presented together with the residual neural network that can be adopted in this context as a starting point for a new model dedicated to a specific task;
- in Chapter III, the research activity carried out at CERN has been presented. A multi-layered neural network architecture has been proposed to model magnetic hysteresis in the context of electromagnets;
- in Chapter IV, in the healthcare context, a deep learning model has been proposed to automatically detect fractures in patients with maxillofacial trauma.

The anonymized images have been obtained from the U.O.C. of Maxillofacial Surgery of the University Hospital “Federico II”.

Chapter 1

Background: the impact of Deep Learning in Industry and Healthcare context

Everyone benefits from deep learning even if they often do not realize it. Facial recognition technology allows social media platforms to help users tag or share photos of friends [6]; search engines that suggest which movies to watch or which items to buy based on user preferences [7]; digital home assistants equipped with a spoken language interface that retrieves the weather or time, plays music, sets alarms [8] and so on. As anticipated in the Introduction, deep learning is a sub-category of neural networks, which is a sub-category of machine learning, which in turn is a sub-category of AI, where learning is based on the assimilation of data representation. The relationship between artificial intelligence and its subcategories is shown in Fig. 1.1.

AI involves all the computational machines capable of performing tasks characteristic of human intelligence. Examples of these tasks include language understanding, planning, learning and problem solving, object and sound recognition [9].
Machine learning is essentially a way to implement artificial intelligence; an AI subgroup that focuses on the ability of machines to receive a series of data and learn on their own. Specifically, machine learning is the ability of a machine to learn without being explicitly programmed. It is therefore a way to "train" an algorithm so that it can learn from various environmental situations. A huge amount of data is required

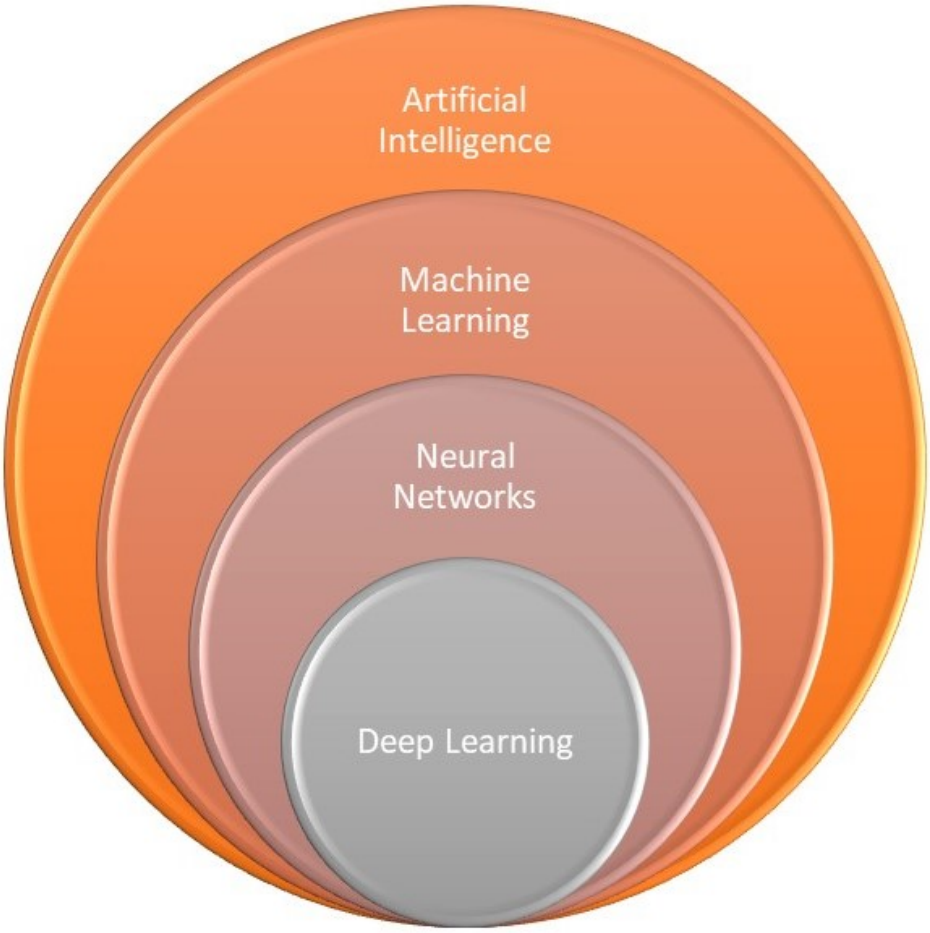

Fig. 1.1 Artificial intelligence and its subcategories.

for the training in order to allow the algorithm to adapt and improve according to the situations that occur [10]. Machine learning automates the construction of the analytical model, using neural network methods and statistical modeling to find hidden information in data.
Neural networks are computing systems consisting of interconnected units, called neurons, that process information by responding to external inputs, transmitting the related information between different units [11]. In particular, the neural networks process the incoming information, select only the necessary ones and on the basis of this information arrive at a conclusion.
Deep learning uses huge models of neural networks with various processing units, capable of learning complex models through a huge amount of data. In particular, neural networks with more than 3 layers (including the input layer and the output layer) constitute deep learning models [12].

1.1 Deep Learning in Industrial Context

By means of neural networks and their ability to analyze data such as audio, time series, video and images, deep learning is attracting more and more interest and is used in a lot of different contexts. One of the main fields of application is Cyber Security [13]. By means of deep learning, the effectiveness of IT system security measures can be significantly increased and much more robust security systems can be deployed. Systems based on deep learning techniques are able to identify not only known but also unknown threats, which are detected by the system as anomalies of the neural network recognition model. Attacks can be of various types such as spam, malware, network intrusions, insider threats, fake data injection and so on. Deep learning architectures are also designed to detect possible fraud [14]. These systems are able to identify serial scammers who modify their habits to evade controls or, for example, credit card fraud. Roy et al. in their work [15] highlight that credit card fraud caused a loss of $3 billion for North American financial institutions in 2017. They evaluated a set of deep learning topologies able to detect credit card fraud and proposed a framework for parameter tuning to allow financial institutions to minimize losses by preventing fraudulent behavior.
Another application of deep learning that is very popular is the chatbot that is a program that simulates human conversations, allowing users to interact with dig-

ital systems as if they were speaking with a real person [16]. These systems are mainly used for customer assistance both for purchased products and to guide them during the purchase of products or services [17]. This solution is mainly used in the insurance, banking and telecommunications fields. Chatbots are able to communicate instantly with customers, consumers, followers. Many companies have had the opportunity to connect with a potentially unlimited number of customers, even proactively, that is, without necessarily having to wait for an initiative from the user himself. Other advantages consist in providing news and information in a timely manner based on parameters provided by customers or users; answering questions in real-time, directly solving many logistical problems; cost reduction for online interaction with the customer; implementing targeted promotional and marketing activities. So a well-crafted chatbot makes business life easier. However, the disadvantages are not lacking. Chatbots are often considered to be complicated and take a long time to understand what customers want; they may not understand a question if it does not refer to something they have previously learned, leading the customer to leave the chat and not to make purchases [18]. In the assistance field, deep learning has also given rise to voice assistants such as Microsoft's Cortana, Google's Assistant, and Amazon's Alexa [19]. In this case, deep learning algorithms are used not only to recognize people's natural language but also to improve the voice assistants' performance according to the requests and behavior of the user, so as to interact with him in an increasingly timely and relevant manner. Virtual assistants aim to make people's lives easier. For example, virtual assistants can buy cinema tickets online, gather information on an event, make appointments, check train times. Furthermore, given their ability to interact not only with people but also with different types of wireless devices, virtual assistants are able also to control home appliances and other connected tools, thus creating smart homes [20].
Another context in which deep learning techniques can be used with great success is the content creation [21] such as articles, captions, essays and so on. By using deep learning, it is possible to create systems able to write in a specific language, learning its punctuation, grammar, spelling and even different styles of writing. In addition, it is also possible to create systems that can replicate human handwriting [22]. Finally, other two main fields of interest for deep learning techniques are mobility and, more generally, industry. By means of the possibilities offered by deep learning, self-driving vehicles have made great strides [23]. These vehicles use computer vision to reproduce human vision and are able to recognize objects present

in the environment around them, such as lost cargo, to move safely. Deep learning and computer vision techniques are also used in industry [24] for the realization of collaborative robots that support operators in production and assembly chains and in predictive maintenance.
Specifically, computer vision aims to reproduce the tasks and functions of the human visual apparatus [25]. In artificial vision systems, the image is acquired through sensors that send the signal to the computer, which in turn digitizes and stores it. The image is then analyzed by the neural network architecture that, on the base of its current knowledge, can make decisions. These systems are based on machine learning and machine training through supervised and unsupervised learning algorithms. The most ambitious challenge of artificial vision concerns the ability of the system to start from a two-dimensional image and process and reconstruct the 3D context in which it is inserted. The three-dimensional object recognition approaches are based on neural networks, in particular, convolutional neural networks [26]. Computer vision also has multiple fields of application but it is in industry and manufacturing that in recent years it is spreading more and more [27]. The diffusion of these systems is mainly due to their ability to improve product quality controls, optimize the production process and increase workplace safety. More specifically, one of the main applications based on deep learning and computer vision techniques that have spread in industry concerns the predictive maintenance [28]. In this context, machine learning and artificial vision approaches allow robots to capture images of the various equipment in the surrounding environment and the enormous amount of data relating to the operating parameters (such as operating temperature, vibration, noise and so on). All this information supports operators to assess the status of the devices in real-time and helps to reduce unplanned downtime, preventing breakdowns. Furthermore, based on production data (such as number of pieces produced, downtime machine, production time, alarms and so on), machine learning algorithms are able to extract patterns that the eye and the human mind could not capture, generating models able to understand if a product will leave the production line with sufficient quality (Quality Check and Anomaly Detection) [29]. The quality analysis of the products is carried out both at the end of the production process on the completed product and along the production line. In the latter case, the systems send the data and any alerts to report defects to the operators.
Another area in which deep learning finds application in the industrial sector is autonomous driving for vehicles moving in the workplace, used, for example, to

relocate the various components along the line, in complete safety [30]. In addition, to improve safety in the workplace, these systems are able to analyze the facility and the behavior of the operators to report potentially dangerous situations, contributing to the prevention of accidents [31]. Furthermore, by equipping robots with artificial intelligence and vision systems, it is possible to replace operators in particularly dangerous situations, improving their working conditions.
Finally, a further industrial application worth mentioning is the barcode scanning and reading [32]. Hand-held scanners can be replaced with computer vision systems integrated with features such as optical character recognition (OCR) or optical barcode recognition (OBR) . In this way, it is possible to optimize the production process and assure that all components are at the right time and in the right place along the production line.

1.2 Deep Learning in Healthcare Context

In addition to the industrial context, the AI-based technologies and, in particular, deep learning applications have penetrated all sectors of the healthcare world, from the improvement of the healthcare process management to the discovery of new drugs [33, 34]. Although it is still improbable that the AI-based tool can completely replace doctors and nurses, modern technologies have already started transforming the healthcare industry.
The identification and diagnosis of diseases are the main fields for which solutions based on machine learning and, in particular, on deep neural networks are used. It is very difficult for a medical practitioner to call to mind all the information necessary to make a rapid and accurate diagnosis, taking into account all the knowledge, including experiments and academic papers. In this context, it is possible to feed an AI-based system with relevant data and let the computer sift through the huge database of information, instead of relying on much more limited human knowledge. The diseases identification was therefore one of the first healthcare fields of application of this type of technology, especially where precise diagnoses are required in a short time with a high rate of criticality, such as oncology [35, 36]. Computer vision technologies together with hundreds of thousands of diagnostic images have improved the recognition of breast cancer [37], melanoma [38], ocular retinopathies [39], but

these are just a few examples of cancer that can be detected through deep learning architectures. In this context, the AI applications are increasingly widespread and reliable, less and less "experimental" or for the benefit of a few cutting-edge centers. Another area where machine learning is spreading faster and faster is the realization of personalized therapies [40–42]. The latter is able to provide more effective treatment by means of a combination of individual health data and predictive analysis, representing an area of research and experimentation of great scientific interest. By means of huge amounts of data and diagnoses, the AI-based tool can estimate patient risk based on the similarity of symptoms and genetic information. In fact, by means of the analysis of big data, it is possible to predict how a patient may respond to a certain treatment, thus turning therapeutics decisions away from a trial and error approach and reducing health diseases and financial burdens [43]. With the convergence of all the sensitive patients' data towards unique government providers, in the future it will be possible to obtain more easily individualized anamnesis in line with the habits and lifestyle of the patient, medical records, real-time data, family inheritance, stress, allergies and so on. Furthermore, valuable support to personal data collection is offered by wearable devices [44]. In addition to data collection, these devices also allow the monitoring of the user's vital parameters (for example, saturation and heartbeat) and can provide suggestions on healthier lifestyles, behavior and posture [45]. Furthermore, applications based on deep learning can be used for the creation of automatic reporting that might potentially save the doctors significant amounts of time [46].

Deep learning is also spreading in the study of new drugs [47, 48]. It is used at different stages of the development process: from the initial screening of chemical and organic compost to the prediction of the success rate based on biological factors. Even in this case, the availability of large volumes of data is fundamental to guarantee the development of AI-based drug research, since most of the current pharmaceutical data is proprietary [49]. However, several partnerships have been established among pharmaceutical companies, technology companies and academia that allow AI research projects based on proprietary data. One of the most significant examples is the company Atomwise which has established collaborations with the world's leading pharmaceutical companies and academic entities, gaining access to information that allows the identification of an innovative candidate drug molecule to be used for multiple sclerosis treatments [50]. Another important project is Microsoft's Project Hanover which, in partnership with the Knight Cancer Institute, is developing AI

technologies for precision cancer treatment using molecular pharmacology [51]. The project consists of a platform capable of reviewing the literature to automatically search for effective anticancer treatments based on patients' specific profile [52]. In this context, machine learning techniques have several potential applications that may help to guide clinical trial research [53]. For example, the application of advanced predictive analysis in the identification of candidates for clinical trials could offer a much wider range of data, including medical visits and social media, as well as genetic information on target populations. The introduction of better sampling procedures would lead to smaller, cheaper and faster studies [54]. Machine learning approaches can also be used for remote monitoring [55] and access to data in real-time [56], ensuring constant monitoring of biological signals to identify any anomalies in time. A different possibility for these kinds of applications include the use of electronic health records to reduce data errors and extract diagnoses and investigation results [57], such as the machine learning-based framework to identify type 2 diabetes subjects, realized by Zheng et al. [58].

Another field of application of deep learning algorithms is planning a radiotherapy treatment [59]. Planning a radiotherapy treatment is an extremely complex activity that involves various professionals and is strongly based on data analysis. The radiotherapist oncologists outline the contours of the volume occupied by the tumor in the images collected by the Computed Tomography (CT) scan and also the contours of the surrounding risk organs that must be safeguarded during treatment. Medical physicists draw up different treatment plan proposals, based on the CT images and dose-volume objectives suggested by oncologists. The definition of a treatment plan requires not only the indication of the direction and intensity of the several high-energy x-ray beams generated by the machine but also the relative position of the patient on the couch. Consequently, it is a problem with a large number of variables (for example, define the trajectory of the arcs that the beams can follow in order to distribute the irradiation over different healthy regions to minimize the damage to the healthy organs and at the same time concentrate the crossing of the beams precisely on the tumor to be affected) and different constraints (for example, the amount of radiation to be delivered in the volume occupied by the tumor as well as that acceptable to surrounding healthy tissues and organs). From this procedure, a series of plans are proposed and are subjected to the evaluation of the oncologists who have the final decision on the treatment. The possibility offered by machine learning algorithms is to propose a radiotherapy treatment plan, using a database

containing a large number of patients and related treatment plans. McIntosh et al., in their work [60], propose a machine learning algorithm that provides radiotherapy treatment plans for the treatment of prostate cancer. The input of the model consists of the images collected from the CT scan of a new patient; the algorithm compares them with a database of patients with the related radiotherapy plans and outputs the new treatment plan. The results of the study show that this AI-based tool may be a useful technology in clinical practice as long as it is subjected to all quality controls that are already performed on the plans developed by human experts to foster trust in these tools and ensure the best treatments possible for patients.

Another application of deep learning technologies is the monitoring and prediction of epidemics around the world [61]. Data sources include data collected from satellites, real-time social media updates, historical web information and so on. For example, Punn et al. in their work [62] explored deep learning models to understand the everyday exponential behaviour of COVID-2019 along with the prediction of future spread across the nations, by utilizing real-time data from the Johns Hopkins dashboard. Another example is provided by Kalipe et al. that in their work [63] focus on deep learning to help predict malaria outbreaks based on climatic factors, such as temperature and humidity. Predicting the severity of epidemics is particularly crucial in third world countries, which often lack medical infrastructure, education opportunities and access to medical care. Therefore, an approach based on deep learning could be used in the future to predict malarial epidemics and enable the necessary preventive measures to be taken to avoid the loss of life due to malaria or, more generally, epidemics.

"Virtual" medical assistants represent another important application of machine learning technologies in the Healthcare world [64]. The categories of users who would benefit most from the virtual assistants are patients with chronic diseases, the elderly and patients in rural areas where access to care is limited [65–67]. One of the advantages of using virtual medical assistants is that patients may feel less embarrassed and more honest when sharing private details with them rather than a human for fear of being judged. Furthermore, the nursing staff is often overburdened with responsibility, which damages patients' well-being and increases the probability of patient safety incidents. In this context, by automating repetitive tasks, virtual nurses can substitute human staff, reducing the risk of accidents while performing their work. Finally, virtual nurses can monitor post-operative patients to make sure their condition does not get worse. They can answer patient questions, collect vital

signs and make an appointment with the doctor. They are available day and night every day of the week and are programmed to show empathy. This approach leads to economic savings for the healthcare system and an improvement of the quality of services provided [68].
Alongside the benefits, there is a certain number of limitations that deep learning has to face in the healthcare context. First of all, there is the issue of the availability of data. One of the principal aspects in the construction of a deep learning model is to be able to draw on a dataset that is representative of all possible subjects, so that it is as heterogeneous as possible [69]. The best case would be represented by training the models on a set of data that are very similar, if not identical, to those listed in electronic medical records. Unfortunately, it may happen that only small datasets are available, often collected from small clinical centers, sometimes of bad quality (characterized by noise, generated by erroneous data) [70].
Another problem that deep learning faces is privacy. As mentioned above, having datasets consisting of correctly compiled medical records would be ideal. However, these data are protected by the legislation that considers them as sensitive information. Therefore, it is more and more difficult to find real data in this context due to strict ethical and legal requirements to safeguard patient privacy [71], making machine learning models more difficult to define. A solution could be to hand over the clinical data to the patient himself, who will then decide what use to make of it [72].
Another important aspect is the impact of potential biases on the deep learning algorithms [73]. The biases can be generated by missing data and patients not detected by algorithms, misclassification and measurement error, sample size and underestimation. These biases and gaps in the dataset might contribute to socioeconomic disparities in the healthcare context. Therefore, during the development of a machine learning system, it is essential to understand how much these biases, represented by the data, will influence the final model and which tools to use to cope with this issue [74].
An important point in using machine application in the healthcare context consists in the sufficient experience needed for the final evaluation of the system's output. Doctors and patients must understand the limitations of these tools, such as the impossibility of a certain model to be able to generalize about another type of problem. It can be very dangerous to blindly rely on machine learning models since it may lead to erroneous decisions. For example, a doctor could underestimate a delicate

case if the algorithm returns a wrong result, below a certain alarm threshold or if the virtual assistant answers the questions of patients and doctors incorrectly [75]. Finally, it is important to have a strong interaction between different teams of doctors, biologists and computer scientists. They must operate together so that they can realize usable models in their respective fields. If there is no collaboration between the teams, there is the risk that computer scientists build systems unsuitable for doctors. Finally, it must take into account not only research works published in traditional scientific journals but also computer manuscripts presented in conferences such as ICML (International Conference on Machine Learning) and NeurIPS (Conference on Neural Information Processing Systems) , which bring important innovations in the machine learning field for the healthcare context.
In conclusion, machine learning, and in particular deep learning, is increasingly infiltrating the medical field, playing an important role in health improvement. Machine learning systems introduce numerous advantages such as the streamlining of medical and bureaucratic procedures, more accurate and rapid diagnosis, the acquisition of global therapeutic knowledge, greater specialization of medical personnel. However, the availability of quality data is necessary (not always possible); it is necessary to address the lack of shared data collection procedures and, finally, the presence of the human component in the final evaluation of the diagnosis remains fundamental to avoid any errors in the system outcomes.

Chapter 2

Deep Learning and the power of Transfer Learning

As anticipated in the previous chapter, deep learning uses artificial neural networks for learning, by means of multi-layer architectures that are capable to process information and identifying features to solve complex problems. In the following sections, the two principal kinds of problems that machine learning, and in particular neural network models, can cope with are described. Then, the learning process of a model is described and the main deep learning architectures, feed-forward and recurrent, are introduced. Finally, the possibility of adopting cross-validation techniques to perform hyperparameters optimization is explored and, to conclude, the transfer learning approach is presented.

2.1 Application Contexts

As anticipated in Chapter 1, there are a lot of different tasks for which machine learning, and in particular neural networks and deep learning, can be adopted successfully. In this section, two main tasks are presented: classification and regression tasks.

2.1.1 Classification Problems

In classification tasks, the machine learning algorithm has to identify the category or class, k, to which a certain input belongs. In order to perform the task, it generates a function f: $\mathbb{R}^n \rightarrow \{1, ..., k\}$. In the field of machine learning, this is fundamentally reflected in the content of the dataset, a collection of input data and relative targets that can be used by a machine learning algorithm for learning and prediction purposes, and on the choice of the loss function. In this case, the dataset targets are discrete values while the loss function generally corresponds to the Cross-Entropy or its variants. Object recognition is an example of classification problems: the image, typically identified by a matrix of pixel brightness values, represents the input, while the discrete value, representing the object in the picture, is the output of the model. For example, Affonso et al. in their work [76] used deep learning for biological image classification, in particular to classify the quality of the wood board. They compared deep learning techniques with traditional machine learning algorithms (such as nearest neighbors [77], support vector machines [78]). The results showed that deep learning techniques have reached predictive performance higher than traditional classification techniques, especially in the most complex scenarios. Another example is provided by Lecun et al. [79] that used deep learning techniques for document recognition, in particular for handwritten character identification. Also in this case, deep learning algorithms outperformed all the other traditional techniques. Another application context in which deep learning has been applied successfully is face recognition, as shown in [80], where the authors proposed deep neural networks to perform facial landmark recognition and unrestricted face detection. This technology is the basis for automatic image tagging [81]. The research studies presented above are just a few examples of the many application contexts where deep learning can be successfully adopted, outperforming the traditional approaches.

2.1.2 Regression Problems

In the case of regression problems, machine learning, and in particular neural networks and deep learning, has to forecast a continuous value, given a certain input. In order to perform the task, it generates a function f: $\mathbb{R}^n \rightarrow \mathbb{R}$. The type of this task is analogous to a classification problem, apart from the format of the output, which is a continuous value and not a discrete one, and the loss function that generally

corresponds to the Square Error or its variants.
Regression problems are present in a great variety of domains. An example is offered by Sreehari et al. in the context of environmental monitoring. In their work [82], they used multiple linear regression to forecast climate variables, in particular rainfall prediction, that can be useful for farmers to make appropriate planting decisions or to evaluate the possibility of floods or droughts. Another application field is finance and marketing. In this context, Gopalakrishnan et al. [83] proposed a method to examine the sales of a big supermarket and forecast future sales in order to improve their incomes. Deep learning techniques for regression problems are also used in the energy field as shown in Ref. [84], where an ensemble of deep learning algorithms has been presented to estimate the electricity load demand, which is crucial for implementing the energy policies. Finally, deep learning for regression tasks has been applied also to medical imaging. Suter et al. in their work [85] used deep learning algorithms to predict the survival time for patients with high-grade brain tumors.

2.2 How does the learning process work?

Deep learning operates with artificial neural networks in such a way as to allow machines to learn more "deeply" by means of the use of multiple layers. Artificial neural networks exploit hidden layers to create multiple layers of abstraction. Each layer adds more and more detailed information so that the network is able to provide reliable output and can solve complex pattern recognition problems. As the number of hidden layers and input data increases, the output of the neural network will be more and more precise: deep learning systems improve their performance as the data received increases.
As described in Sec. A.1, the neuron can be identified as the nonlinear function of a weighted linear combination. The parameters of the linear combination are the weights, W, and they represent the parameters of the model. The learning (or training) of the network aims to estimate the weights and once they have been estimated the network can be used for prediction.

There are four main types of learning processes:

- *Supervised learning* [86]: in this case, the model is built from training data, consisting of input data and the relative targets. Therefore, in this type of learning, during the training process, the desired output signals are already known. However, the training data do not provide information regarding the behavior of the other intermediate layers. These layers are used by the learning algorithm to generate the desired output, but what each intermediate layer should do is not specified by the training data. Therefore, it is the role of the learning algorithm to determine how to employ these layers.
  From the analysis of the input data and the relative target, the network learns associations and rules and uses them to make predictions in the future. Classification and regression techniques are based on this type of learning;

- *Unsupervised learning* [87]: in this type of learning the inputs are provided without the relative targets. The machine learning algorithm will have to find a structure within the data and extrapolate significant information to make predictions;

- *Semi-supervised learning* [88]: considering supervised and unsupervised learning, semi-supervised learning is a hybrid model in which of all the data present in the training set, only some of them are paired with their respective targets. The network will find the connections between the remaining inputs and outputs;

- *Reinforcement learning* [89]: in this case, there are no associations between incoming and outgoing inputs and the network learns to improve its performance through interaction with the environment. Once a certain goal to be achieved has been established, actions that approach the goal are reinforced, while those that move away from it are penalized. Through this mechanism, the network learns to distinguish the right actions from the wrong ones. An example in which reinforcement training is used successfully is the training of a system for the game of chess.

In the case of supervised learning, the basic idea of the training process relies on the comparison between the target and the predicted output of the neural network that gives rise to an error, used to better adjust the weights with an optimization technique. The goal of the neural network is to find the best weights in order to minimize errors.

2.2.1 Gradient Descent Optimization

The optimization process consists in minimizing a function $f(x)$, acting on x. In this case, the function to be minimized is called error function or loss function or even cost function.
The derivative of a function $y = f(x)$, with both x and y real numbers, is identified by $f'(x)$ or $\frac{dy}{dx}$ and provides the slope of the function $f(x)$ at x. It provides information on how to change the input x to have a corresponding variation in the output y, since $f(x+\varepsilon) \approx f(x) + \varepsilon f'(x)$. Consequently, it is possible to reduce $f(x)$ by applying iteratively little variations to x with the opposite sign of $f'(x)$. This procedure is known as gradient descent and it is the optimization technique that is mainly used in neural networks [90].
If $f'(x) = 0$, there is no information regarding the direction in which to move. In this case, x values are called stationary points (local minimum/maximum). For a local minimum, it is not possible to further minimize the function with infinitesimal steps. Instead, if $f(x)$ reaches the absolute lowest value, then the corresponding x value is called global minimum. In deep learning architectures, there may be multiple local minima, which are not optimal, making the optimization process very difficult, particularly when x is multidimensional.
Often, it can happen that the function has $n > 1$ arguments, that is f: $\mathbb{R}^n \to \mathbb{R}$, with still one scalar output. In this case, partial derivatives are used for the optimization procedure. $\frac{\partial}{\partial x_i} f(x)$ is the partial derivative that evaluates how the function $f(x)$ changes when only the x_i variable increases at x. In this context, the term gradient is introduced to represent the derivative with respect to a vector. In fact, the gradient of function $f(x)$, $\nabla_x f(x)$, contains all the partial derivatives, generalizing the concept of derivative. The i-th element of the gradient represents the partial derivative of function $f(x)$ with respect to the variable x_i. In this case, stationary points are the points for which all the elements of the gradient are zero. The directional derivative with direction u, that is a unit vector, is defined as the slope of $f(x)$ in direction u. In order to minimize $f(x)$, the direction in which the function decreases in the fastest way should be chosen and it can be demonstrated that it happens when u is oriented in the opposite direction of the gradient [91]. This means that the function can be minimized by moving along the orientation of the negative gradient. In this case, the gradient descent algorithm identifies the next point x' as $x' = x - \varepsilon \nabla_x f(x)$, with ε the learning rate, which is a positive scalar that identify the step size and can be

chosen in many different ways [92]. The choice of learning rate is, in fact, crucial for the optimization process. If the learning rate is too large there is the risk to jump over minima; on the other hand, if the learning rate is too small, then the algorithm will take a long to converge with the possibility to get stuck in a local minimum. To avoid these issues, usually, the learning rate is varied over the course of the training process.
In this case, the gradient descent will converge if all the elements of the gradient are zero or, practically close to zero.
So, gradient descent is an iterative algorithm and at each iteration i, the next value of θ is given by Eq. 2.1:

$$\theta^{i+1} = \theta^i - \varepsilon^i \nabla f(\theta^i) \tag{2.1}$$

where θ contains all the weights of the artificial neural network, ε is the learning rate and $\nabla f(\theta^i)$ is the gradient with regards to the values of the weights at iteration i. The technique used for estimating the gradient of the error function is the *back-propagation rule*. It is the chain rule [93], in the sense that the derivative of a composed function with regards to the input, $f(g(x))$ can be computed by multiplying two derivatives: the derivative of the external function with regards to the internal one ($\frac{df}{dg}$) and the derivative of the internal function to regards to the input ($\frac{dg}{dx}$), as shown in Eq. 2.2:

$$\frac{d}{dx} f(g(x)) = \frac{df}{dg}\frac{dg}{dx} \tag{2.2}$$

In a multi-layer neural network (see Section A.1 in Appendix A for more details about multi-layer neural networks), the output y can be represented as a function of the previous layers $y = f^n(f^{n-1}(...(f^1(x))))$. It is a composed function, where the more external function, f^n, represents the last layer of the network, while the innermost function is linked with the first layer of the network. The derivative of the output with regards to the input is given by Eq. 2.3:

$$\frac{dy}{dx} = \frac{df^n}{df^{n-1}}\frac{df^{n-1}}{df^{n-2}}...\frac{df^1}{dx} \tag{2.3}$$

Since the computation of the derivative of the output with regards to the input, or to intermediate values, is performed from the last layer of the network to the first layer, the information is back-propagated, hence the name of back-propagation algorithm.

Depending on how much data are used to calculate the gradient of the cost function, three variants of the gradient descent optimization exist: batch gradient descent; stochastic gradient descent and mini-batch gradient descent. On the basis of the quantity of data chosen, there will be a trade-off between the time needed for the parameters update and the accuracy of the update itself.
During the batch gradient descent optimization, the estimation of the gradient of the cost function is performed for the whole training dataset. This means that the weights update is executed after the presentation to the network of all the data of the training dataset. The presentation of the entire dataset to the network together with the weights update represents one epoch. Since the gradients for the total dataset must be computed to perform only one update, this type of optimization can be slow and hardly practicable for huge datasets. Furthermore, it is not possible to add new data on the fly to the training dataset, updating the model online.
On the contrary, the stochastic gradient descent (SGD) executes a weights update for each sample of the training set. This means that after the presentation of each sample of the dataset, the weights of the network are changed. SGD avoids redundant samples, executing one updating at a time, contrary to the batch gradient descent that executes redundant calculations for huge datasets, recalculating gradients for similar samples, which do not contribute much to the learning, before each update. Consequently, this optimization technique is faster than the previous one and it can be adopted to online updates of the training dataset.
Finally, the last optimization algorithm is the mini-batch gradient descent. In this case, it performs an update for each mini-batch containing n training samples. So, instead of using too many samples for one epoch, as in the case of the batch gradient descent, it uses fewer samples but more than one, as in the SGD optimization. The idea is to shuffle the training data set to avoid pre-existing order of samples and, then, to partition the training dataset into mini-batches of a certain batch size. Usually, the size of the mini-batches varies from 50 up to 256, but it depends on the specific application.

2.3 Deep Learning Architectures

Multi-layer neural networks with more than one hidden layer are called deep neural networks. The number of layers represents the depth of a network hence the term "deep learning". In the following sections, some examples of deep neural networks are presented.

2.3.1 Convolutional Neural Networks

Convolutional Neural Network (CNN) is an example of multi-layer neural networks, finding application in a lot of domains. CNNs can be used in classification problems [94], meaning that given an image the network's outcome is the probability to belong to one class; in image retrieval problems [95], in which the network receives in input an image and returns information about that image (such as Google's engine research); for detection problem [96], in which the network is able to recognize the presence of a particular object; in segmentation problem [97], in which the network is able to recognize the elements present in the image; in image captioning application [98], where the CNN receives the image in input and its output is the caption of the image, which try to explain the content of the image; in image colorization [99], where the neural network, given a grayscale photograph as input, returns a plausible color version of the image; in self-driving cars, the CNN can support autonomous driving by identifying, for example, road cracks and objects along the way [100]. CNN can be also fed with input videos [101], considering them as a sequence of images or, in other applications, CNN can be used to provide a 3D representation of an image, starting from its 2D representation [102]. In this case, the neural network learns to infer 3D representations of the world by means of a training dataset of 3D movies [103].
CNN is a deep learning algorithm that learns complex representations of visual data by means of a vast amount of data. Its architecture is inspired by the human visual system: single neurons react to stimuli only in the narrow area of the visual range that is the receptive field; the overlapping of these fields covers the whole visual field. The CNN learns a large number of layers of transformations, applied one on top of the other to extract an increasingly more sophisticated representation of the input data. Assigning importance, through learnable weights and biases, to the different features and elements present in the image, it is able to distinguish one image from

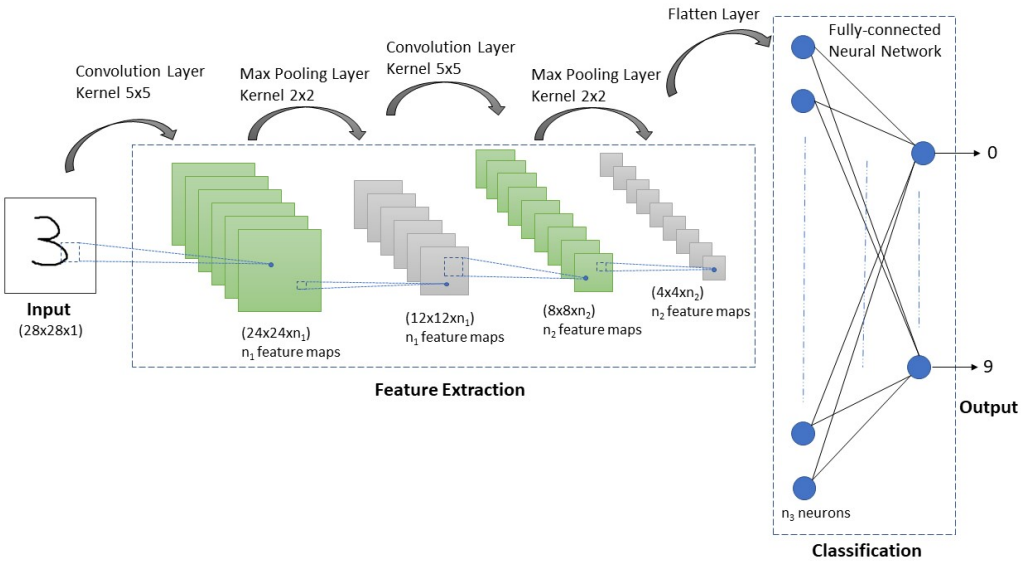

Fig. 2.1 A CNN architecture to recognize handwritten digits.

another. A general CNN architecture used to classify, for example, handwritten digits is shown in Fig. 2.1.

There are four main elements in a CNN architecture: Input Data, Convolutional Layer, Pooling Layer and Classification Layer. As shown in Fig. 2.1, the network is essentially made up of a chain of Convolutional and Pooling Layers, alternating one after the other. In particular, the early convolutional layers are responsible for catching low-layer features, while the final convolutional layers are able to capture high-layer features.
In the following paragraphs, each layer type is described in detail.

Input Data

The input layer consists of a data vector representing the pixels of the input image. For example, in the case of a color image of 28x28 pixels, the input vector must have a length of 28x28x3, where 3 is the number of color channels. In fact, for each pixel of the 28x28 size image, there will be 3 values that represent the three colors of the image in RGB format (Red, Green and Blue), as shown in Fig. 2.2.

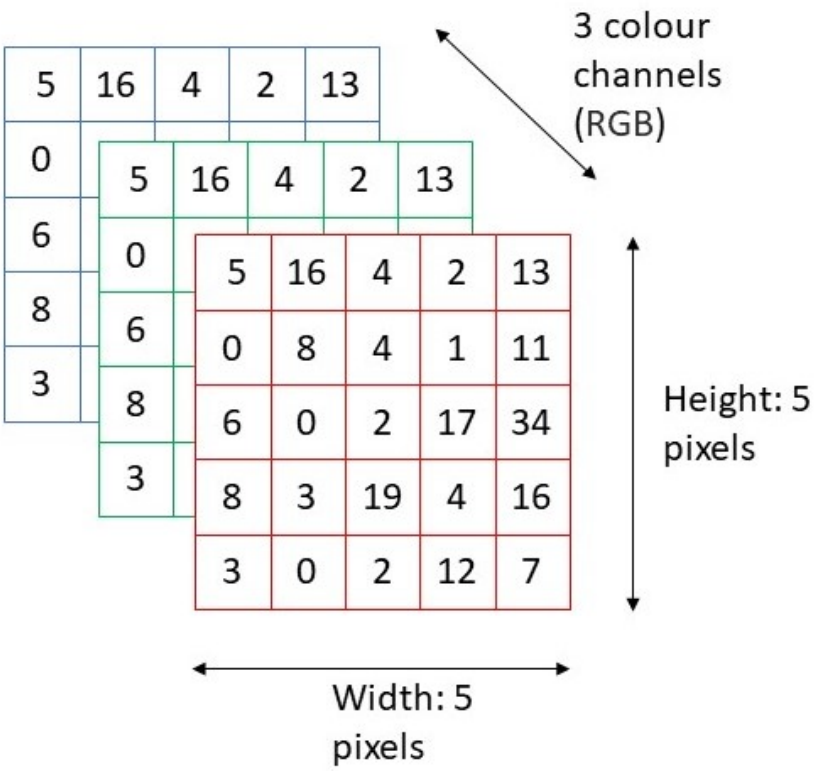

Fig. 2.2 Example of input data: a 5x5x3 RGB image.

One might think of using a fully connected network, in which there is no loss of information. The problem is that a fully connected network leads to a combinatorial explosion in the number of nodes and connections required. With a very small 28x28 pixel color image, it would lead to already have 2 352 nodes in the first layer (28x28 pixels x 3 color channels), with 2 352 connections each. An image with a more reasonable size of 1 000×1 000 would lead to one million nodes with one million connections for each node. Therefore, this solution would not be scalable and totally unrealistic to implement. Conversely, by using a CNN it is possible to scale down the images into a way that is easier to handle while retaining features that are crucial to making a good prediction. Consequently, the CNN represents the best architecture not only for learning features but also for being scalable to huge datasets, typical of classification problems.

Convolutional Layer

The convolutional layer is the main layer of the network. Its goal is to identify patterns, such as vertical and horizontal lines, corners, curves, circles or squares represented in the input image. In each convolutional layer one or more filters can be applied: the greater their number, the greater the complexity of the features that can be identified. The main component of the convolutional layer is the digital filter, or kernel, a small mask that is scrolled over the different positions of the input image; for each position, an output value is generated by executing the scalar product between the mask and the portion of the input covered (Fig. 2.3).

In Fig. 2.3, the filter is represented by a 3×3 matrix. Therefore, the filter will examine only a portion (receptive field) of the 5x5 input image. Moving on, the filter shifts to the right with a given *stride* value until it scans the complete width of the image. The stride controls how a filter convolves surrounding the input volume and represents the number of pixels by which the filter shifts (in this case, stride is 1, which means the filter convolves surrounding the input volume by shifting one pixel at a time). Then, it moves to the beginning (left) of the next line of the image and, with the same stride value, it moves, repeating the process until the whole image is explored.
In the case of images with more than one channel (for example, RGB), the filter must have a depth equal to that of the input image. Therefore, matrix multiplication will be performed between the *n*-th filter and *n*-th slice of the input image ([F1, I1]; [F2,

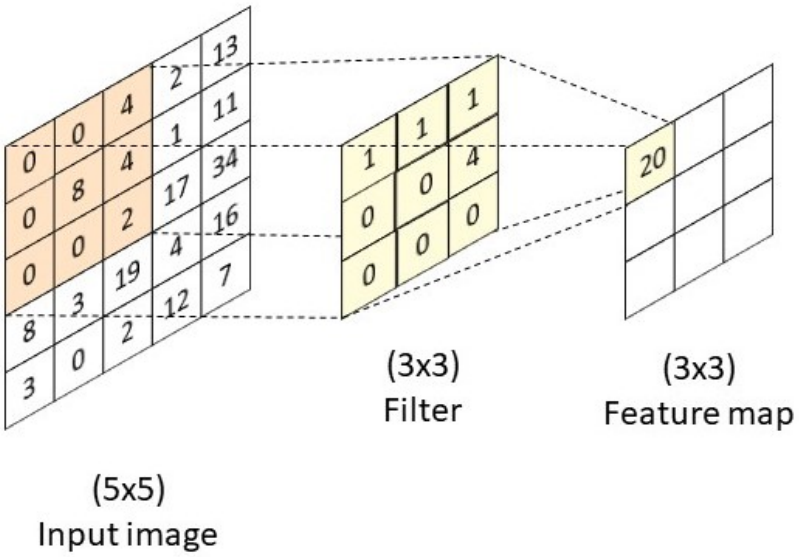

Fig. 2.3 Scalar product between a 3x3 filter and a portion of the input covered.

I2]; [F3, I3]). After that, all the scalar products are summed with the bias to have a final squashed one-depth channel feature matrix output.

So in the end, the image will be scanned piece by piece by the filter, obtaining a smaller matrix (in this case, 3x3), known as feature map, that is the output of the scalar products between the mask and the portions of the input covered. In this case, the feature map size is 3x3 and, in general, it can be computed by Eq. 2.4:

$$\text{Output size} = \frac{(I-F)}{S} + 1 \tag{2.4}$$

where I is the height/width of the input image (in this case, it is 5), F is the filter size (in this case, it is 3) and S is the stride.
In the previous case, the resulting feature map has reduced dimension with respect to the input size. However, it is possible to have a feature map of the same size as the original image by using *zero padding*. Zero padding pads the input with zeros around the border. If stride is 1, in order to have the result of the convolution of the same input dimension, the zero padding must be set to:

$$\text{Zero padding} = \frac{(F-1)}{2} \tag{2.5}$$

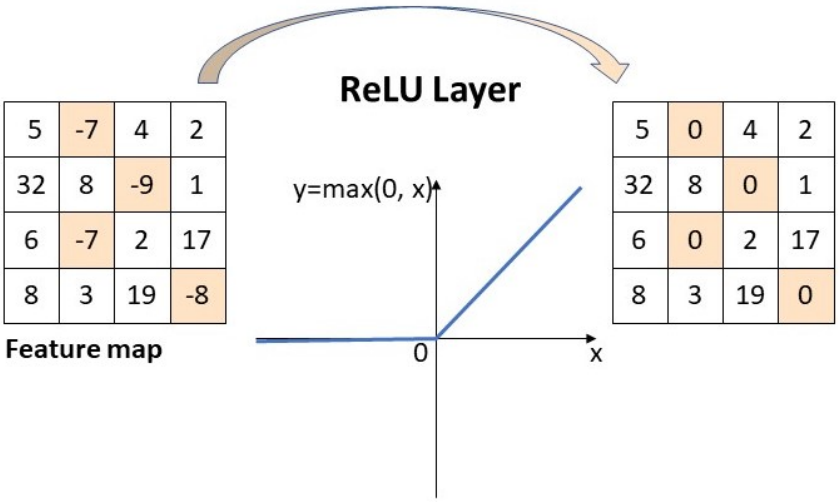

Fig. 2.4 Feature map transformation after a ReLU layer.

In this case, a single pad is sufficient to have a feature map of 5x5 size. Zero padding can be used, for example, in the initial layers of the network to retain as much information regarding the original input volume in order to extract low-level features. By adding zero padding, the general formula to compute the feature map size is given by Eq. 2.6:

$$\text{Output size} = \frac{(I - F + 2P)}{S} + 1 \tag{2.6}$$

where P is the padding.

Typically, after a convolutional layer a nonlinear layer, or activation layer, is added immediately thereafter and it can be considered an integral part of the convolutional layer (that's the reason why in Fig. 2.1 there is only the convolutional block). The goal of this nonlinear layer is to include nonlinearity in a network that essentially has just been performing linear operations in the convolutional layers. The Rectified Linear Unit (ReLU) is used as a nonlinear function because of its computational efficiency [104] and aims to remove previously obtained non-useful values. In fact, applying the function y=max(0, x) to all of the input values, this layer just changes all the negative values of the activation map into zero, as shown in Fig. 2.4.

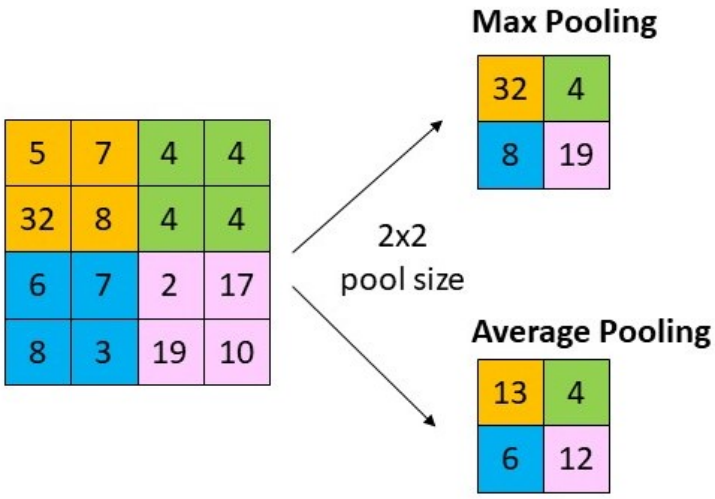

Fig. 2.5 Max and Average Pooling.

Pooling Layer

Typically, each convolutional layer is followed by a pooling layer which aims to reduce the input size [105], making the representations smaller and more manageable. There are two kinds of pooling: max pooling and average pooling. The former considers a filter and a stride of the same length. Then, it applies the filter to the input matrix and outputs the maximum value from the portion of the input covered by the filter. Meanwhile, the average pooling returns the average of all the values contained in the part of the image covered by the filter. In Fig. 2.5 is shown an example of max and average pooling.

Classification Layer

The classification layer is the one that actually performs the classification of the input images. It is able to learn nonlinear combinations of high-level features. After the last pooling layer, the input image is flattened into a column vector to be in a suitable form for the Multilayer Perceptron (MLP), (see Section A.1 in Appendix A for more details about MLP) that characterizes the last part of the network. The flattened output is therefore sent to a feed-forward neural network and the back-propagation is applied to each iteration of the training. After a series of epochs, the network is able to recognize dominating and specific low-level features in the input images that allow it to classify them, using the softmax activation function of the last layer. The

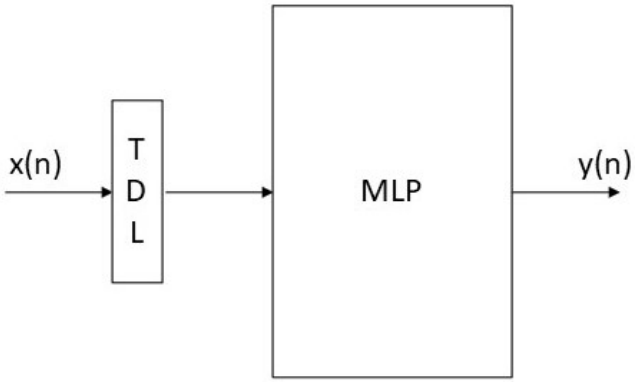

Fig. 2.6 TDNN block diagram.

softmax function [106] transforms a vector of N real values (they can be positive, negative, zero or greater than one) into a vector of N real values, between 0 and 1, that sum to 1. So they can be interpreted as probabilities and the CNN will output the class to which the input image belongs with the highest probability.

2.3.2 Time Delay Neural Network and Nonlinear Autoregressive Exogenous Neural Network

In this paragraph, other two examples of deep neural networks are outlined. The first example is represented by the Time Delay Neural Network (TDNN), shown in Fig. 2.6. In this architecture, past observations of the input are added to the current input data to model long term temporal dependencies with training times comparable to standard feed-forward deep neural networks [107].

In a TDNN architecture, the basic unit of a neural network, which calculates the weighted sum of the inputs to pass it throughout a nonlinear function, includes also delayed values of the input x, x^{t-1} until x^{t-N} [108]. If there are m input values, each input value will be considered together with its N delayed values, as shown in Fig. 2.7.

In this case, there will be different weights for each input and its delayed values. For instance, if $N = 2$ and $m = 16$, then 48 different weights are needed to calculate the weighted sum. By means of the possibility to introduce past values of the inputs, in TDNN architectures the neurons are able to make a comparison between the

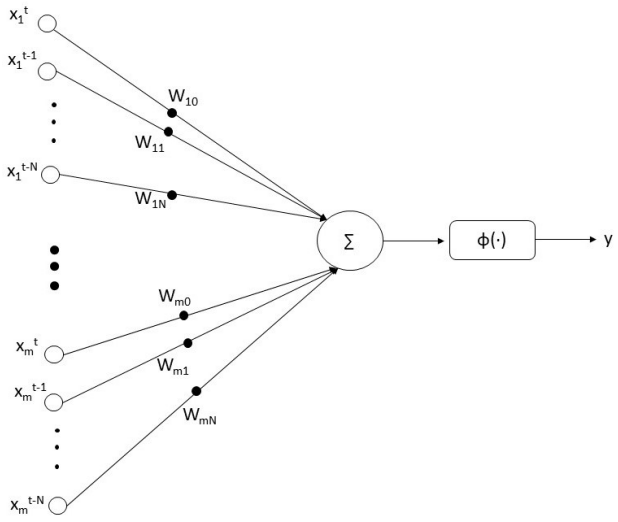

Fig. 2.7 TDNN basic unit.

current input and its past history.

The second example is the Nonlinear Autoregressive Exogenous Neural Network (NARX) [109], shown in Fig. 2.8. This type of architecture relies not only on the past observation of input data but also on temporal output feedback.

Compared to other recurrent neural models (see Section A.2 in Appendix A for more details about recurrent neural networks), which have feedback from hidden neurons [110], the NARX architecture has feedback that comes only from the output layer. Nevertheless, it has been demonstrated that NARX networks can be used instead of conventional recurrent networks without any computational loss and that they are computationally powerful at least as powerful as Turing machines [111].

2.4 Hyperparameters Optimization through Cross-Validation techniques

Almost all the machine learning models are characterized by a wide number of parameters, called hyperparameters, that need to be specified [112]. The tuning of the hyperparameters is not trivial and no hard-and-fast rules exist for guaranteeing the optimal result on a certain dataset.

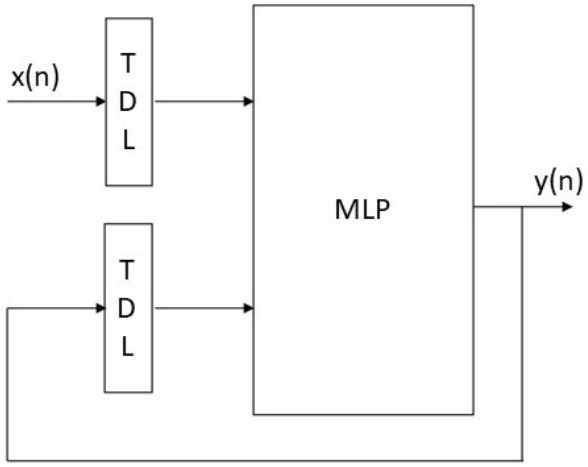

Fig. 2.8 NARX block diagram.

Examples of hyperparameters for deep learning architectures are the number of layers, the number of nodes per layer, the size of filters, learning rate, described in Sec. 2.2.1, weight decay and dropout. Weight decay [113] consists in adding a term to the error function of the model to penalize the network's weights by shrinking them during the back-propagation. This mechanism helps the network to generalize better, preventing the model from fitting the noise of training data. Dropout [114] is another technique used to improve the network's generalization. In this case, neurons are randomly dropped out together with their connections during the training process. This mechanism prevents neurons from co-adapting and consequently resulting in an overly specialized model on a specific training dataset (overfitting problem [115]).
Different sets of hyperparameters may give rise to different models and a model selection process can be used to select the best performing model. There are different model selection techniques that can be adopted but the most common procedure is the K-Fold Cross Validation.
The term cross-validation refers to the fact that a crossing between the training and validation sets is performed in subsequent rounds. In fact, the idea is to test each sample of the dataset and, therefore, is especially used in the case of small datasets. On the other side, the term K-fold refers to the fact that the iteration over a dataset is repeated K times. During each round, the dataset is split into K sets: one set is employed as validation set, while the remaining $K-1$ sets are used together as training set.
In order to understand how the K-Fold Cross Validation can be used as a model

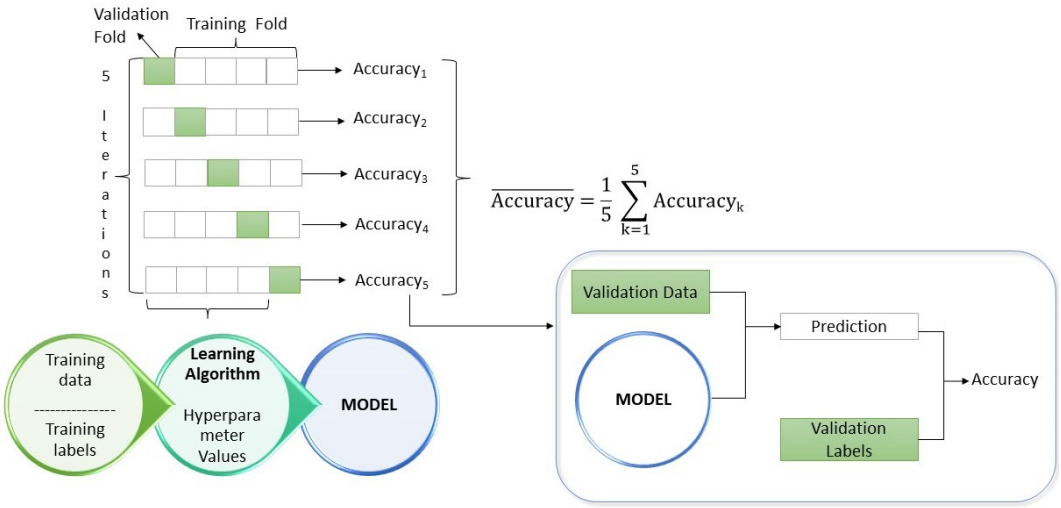

Fig. 2.9 K-Fold Cross Validation process for model evaluation with K=5.

selection technique, it may be useful to understand before how the K-Fold Cross Validation can be used for model evaluation, as shown in Fig. 2.9.

In this case, the 5-Fold Cross Validation is used with a fixed hyperparameters set. In each round, the learning algorithm trains the models on the training folds, giving rise to five different models, fitted on distinct but partially overlapping training sets. On the contrary, the five resulting models were evaluated on non-overlapping validation sets. The final cross-validation performance can be calculated by means of the arithmetic mean of the K different performances resulting from each iteration. After having presented the K-Fold Cross Validation for model evaluation, it is possible to proceed to describe the K-Fold Cross Validation process for model selection, which is the most common case this technique is used for.
The idea is to keep a separate set for test that do not contain samples already included in the training set. In Fig. 2.10, the K-Fold Cross Validation for model selection is presented.

The first step of the process consists in evaluating the models characterized by *N* different hyperparameters sets. Hence, for each hyperparameters set, the K-Fold Cross Validation procedure is performed on the training set, which consists of training data and training labels, generating *N* models with different performances. In the second step, the hyperparameters set that returned the highest performance during the K-Fold Cross Validation is selected and the model is trained on the entire

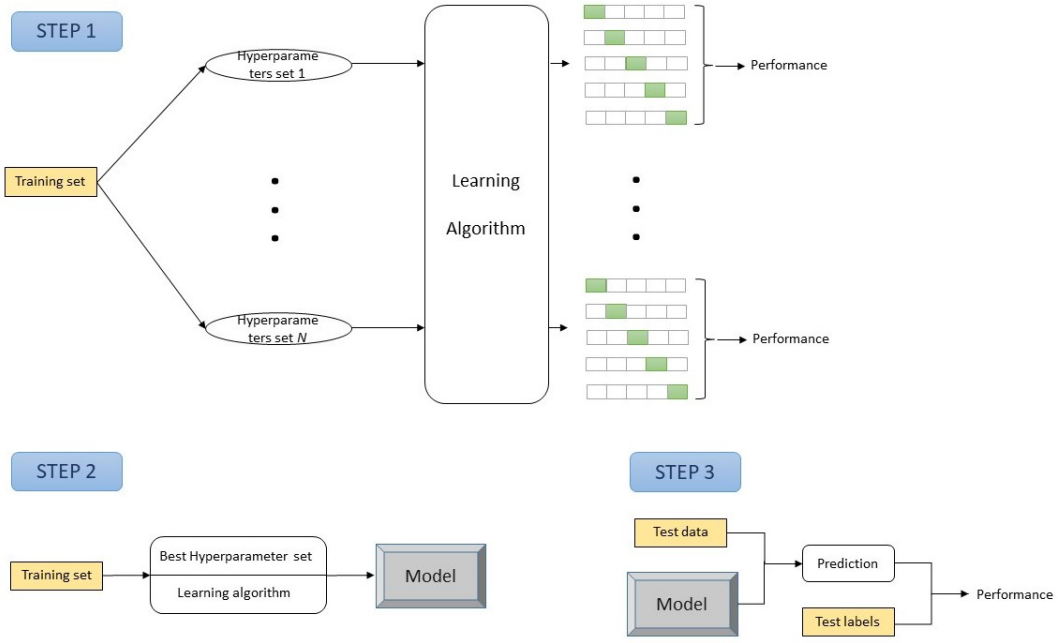

Fig. 2.10 K-Fold Cross Validation for the model selection process.

training set using the selected hyperparameters set. Finally, during the last step, the test dataset is used for the final evaluation of the model obtained in the previous step.

2.5 Transfer Learning

Building and training neural networks from scratch necessitates large datasets as well as enormous computational resources. For example, CNNs for image classification are trained on billions of images, using several servers running for multiple weeks [116]. This process is not always practicable especially for most medical researchers. Transfer learning allows to overcome this difficulty, adopting the extremely fine-tuned characteristics of CNNs, trained on billions of data, as a baseline for a new model [117]. The idea, shown in Fig. 2.11, is to keep the weights of a network that has been trained on a huge amount of data and to "fine-tune" the model with the specific dataset of a certain case study. In order to fine-tune the network, the last layer of the pre-trained network will be replaced with a new one characterized by random weights, to let it adapt to the specific domain and target task. At this point,

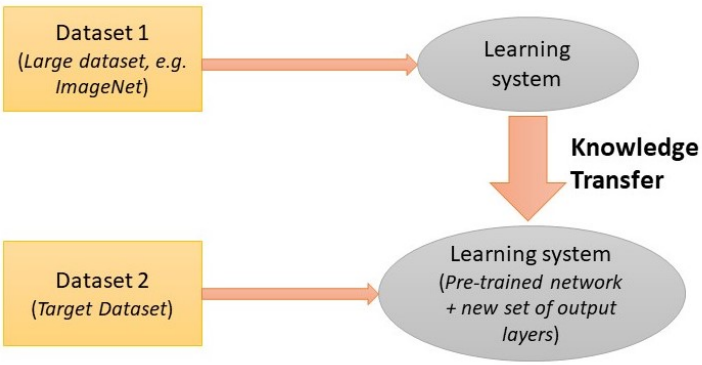

Fig. 2.11 Block diagram representing the concept of transfer learning.

all the weights of the other layers are freezed (in order to not change them during the training of the network) and the network is trained normally.

The idea on which transfer learning is based is that the first layers of a network extract low-level features, such as edges and curves, which are present in all the images, while the next layers are able to extract increasingly complex features up to the last layers of the network that extract high-level features to carry out the classification of the image. Since every network needs to detect curves and edges, rather than using random initialization of weights for training the whole network, it is appropriate to use the weights of the pre-trained model, freezing them and focusing the training on the more important layers for the final image classification, which are the last ones.

A relevant aspect in using the transfer learning approach is the choice of the pre-trained model, which is crucial to reach good performances in the new domain. For instance, in case of image classification problems, several pre-trained models are available (like Resnet [118], AlexNet [119], Inception-v3 [120]), trained on huge datasets such as ImageNet [121]. Conversely, for other types of tasks, different models must be chosen: for example, for the text classification task, pre-trained architecture such as Word2Vec [122] and BERT [123] can be employed.

Residual Network (ResNet) is an example of CNN architecture that can be used as pre-trained network for image classification. The deep CNN won in 2015 the Challenge "ImageNet Large Scale Visual Recognition" (ILSVRC) [124]. There are several ResNet variants that differ not only for the number of layers but are often novel architectures, such as ResNeXt [125], or Densely Connected CNN [126]. In particular, the training of ResNet50 was performed on a dataset consisting of more

than a million pictures from the ImageNet database [127]. The neural network has 50 layers and is able to classify images of 1 000 different object categories, such as mushroom, castle, apron and several species of animals (dog, spider, cat, etc.). Consequently, the neural network has learned a huge quantity of features over a large set of images. The images used as input of the network have a size of 224-by-224. The ResNet50 architecture is characterized by 4 stages:

- Initial convolution with kernel size of 7x7 and max-pooling with kernel size of 3x3;
- 9 convolutional layers:
  - first convolutional layer with kernel size of 1x1 and 64 different kernels;
  - second convolutional layer with kernel size of 3x3 and 64 different kernels;
  - third convolutional layer with kernel size of 1x1 and 256 different kernels;

  These three convolutional layers are repeated in block three times;
- 12 convolutional layers:
  - first convolutional layer with kernel size of 1x1 and 128 different kernels;
  - second convolutional layer with kernel size of 3x3 and 128 different kernels;
  - third convolutional layer with kernel size of 1x1 and 512 different kernels;

  These three convolutional layers are repeated in block four times;
- 18 convolutional layers:
  - first convolutional layer with kernel size of 1x1 and 256 different kernels;
  - second convolutional layer with kernel size of 3x3 and 256 different kernels;
  - third convolutional layer with kernel size of 1x1 and 1 024 different kernels;

  These three convolutional layers are repeated in block six times;
- 9 convolutional layers:

  - first convolutional layer with kernel size of 1x1 and 512 different kernels;
  - second convolutional layer with kernel size of 3x3 and 512 different kernels;
  - third convolutional layer with kernel size of 1x1 and 2 048 different kernels;

  These three convolutional layers are repeated in block three times;

- Average Pooling layer and a fully connected layer characterized by 1 000 neurons with a softmax function in the end.

Using pre-trained models for the learning process offers several advantages such as reducing the training time (deep learning network's training takes a very long time in case of millions of data); saving computational resources (deep learning network's training often needs powerful resources) and less data is needed for training. Although transfer learning has its advantages, it also has its limits. One of the major limitations in using transfer learning is the issue of negative transfer [117]. If a model performs less well after transfer learning, then a negative transfer has taken place. It is possible to use transfer learning only if the target problem and the initial problem are sufficiently similar. If the training dataset of the new domain is not enough related to the dataset the pre-trained network has been trained on, the model may have worse performance than the expected one. At the moment, no explicit standards regarding what kind of tasks are sufficiently related (or how algorithms identify related tasks) exist, making it difficult to prevent negative transfer.

Chapter 3

Deep Learning for Industry: Hysteresis Modelling in Iron-Dominated Magnets

3.1 Background and Motivation

The non-linearity and historical dependence characterizing ferromagnetic materials cause hysteresis modelling to be a complex aspect of computational magnetism [128–134]. Such problem is typically evaluated in presence of sinusoidal current waveforms, that characterize electrical machines [135–137], while it is relatively unexplored in presence of more complex excitation current waveforms that characterize the cyclical operation of particle accelerators. An example of this kind of waveform is shown in Fig. 3.1.

Such waveforms consist of almost periodic sequences that include pulses with trapezoidal shapes. The slopes, corresponding to the ramp-rates of the cycle, vary from one cycle to another as well as plateaux (flat-top and flat-bottom) levels. The latter corresponds to the hysteresis loop reversal points; their succession widely influences the relationship $B(I)$, between the magnetic field and the excitation current. In such circumstances, $B(I)$ is very complicated and difficult to forecast. Within particle accelerators the radio frequency cavities accelerate the beam that circulates in a ring of magnets. These magnets produce bending magnetic fields that increase proportionally to the beam momentum. Consequently, “the precise knowledge of

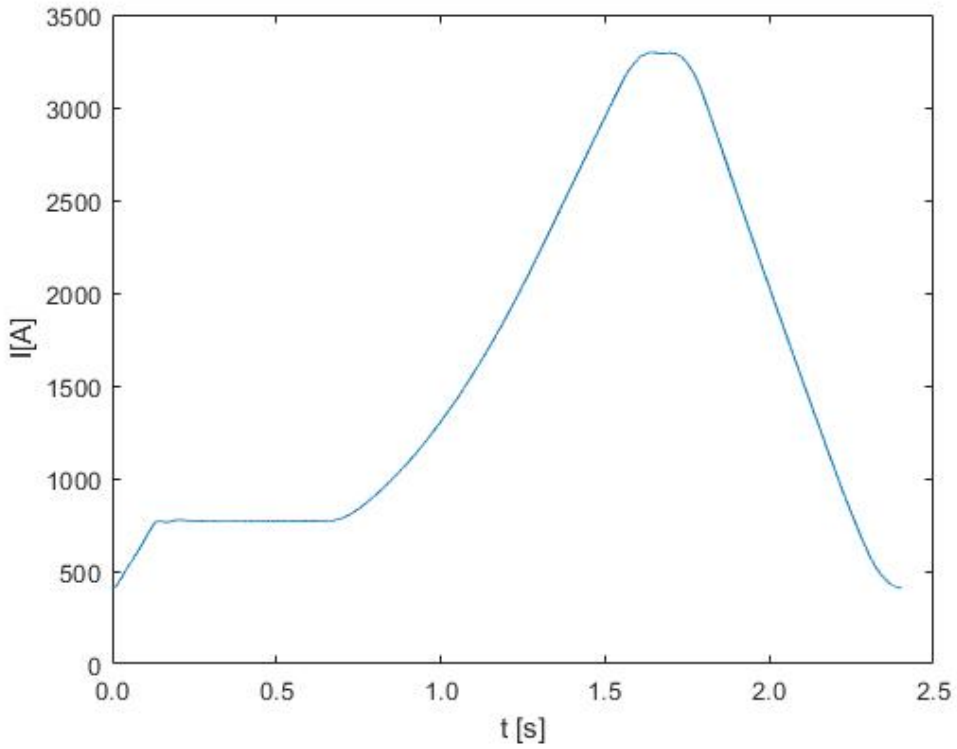

Fig. 3.1 Typical excitation waveform in the Low Energy Ion Ring (LEIR) accelerator.

the magnetic field, $B(t)$, at any instant of time during a magnetic cycle is critical not only for beam and power supply control but also for beam diagnostics and qualitative feedback to various operators. The required Normalized Root Mean Square Error (NRMSE) for the correct functioning of the accelerators complex is typically 0.01 % [138]" [1].

The superconducting dipoles of the LHC produce high field, reaching 8.4 T, but "it is relatively unaffected by perturbations" [139]. Then, a semi-empirical model (FiDeL) [139] is able to model the relationship between magnetic field and excitation current, $B(I)$, accurately. However, in iron-dominated magnets, which represent the most common case, the iron core gives rise to nonlinear effects such as hysteresis, eddy currents that may be of the order of the percent or more. In these cases, the problem of the modelling of the $B(I)$ relationship becomes more difficult [140, 141]. Recent attempts using Preisach models [142, 143] could not achieve an accuracy better than 0.2 %, while Jiles-Atherton differential models [144] are shown to be unsuitable for their difficulty in dealing with "minor hysteresis loops". Alternatively, measurements may be performed in real-time in reference magnets, powered in series with the accelerator rings. For example, at CERN, six of the synchrotrons operate by means of the feedback from measurements performed by real-time magnetic measurement systems (the so-called "B-train" [145]). However, these kinds of measurement systems are often expensive and complex to realize, and in some cases difficult to deploy (for instance, due to absence of sufficient space to install sensors, which are used for the measurements, near the vacuum chamber crossed by the beam). Consequently, there exist a solid motivation for exploring new types of models to support or rather substitute measurements. Moreover, even if it is possible to implement and deploy real-time measurement systems, such models may be useful, for example, during periods of hardware maintenance or shut-down of the machines, when the real-time measurement is not available.

Recently particular interest has been addressed to artificial neural networks, currently employed in a lot of fields relative to time-series forecast with impressive results [146–148, 111], while they are still relatively uninvestigated in magnetic applications. For example, in Ref. [149], a hybrid architecture, including Preisach model and Neural Network, is shown to model hysteresis in the case of "ARMCO" pure iron. The proposed model is able to reach an NRMSE of the order of 0.7 %. In fact, artificial neural networks are increasingly used to model magnetic hysteresis coupled with classical approaches such as Wlodarski, Preisach, Jiles-Atherton and Chua-

Stronsmoe models [135, 137, 150]. In Ref. [135], one of the first attempts at describing the memory mechanism is proposed in systems with rate-independent hysteresis using a combination of Preisach state updating rules and a feed-forward neural network. The model is identified by tuning the weights of the network architecture through a back-propagation based algorithm. Developments following the approach in Ref. [135] are also reported in Refs. [136, 151, 152]. In Refs. [153] and [154], a different hybrid approach is presented by combining an artificial neural network with a Fourier Descriptor to assess simulated hysteresis loops. The method is suitable when a distorted sinusoidal magnetic field H excites, in steady state, the ferromagnetic core of a device and allows to handle problems that appear in classical approaches when static hysteresis, eddy currents and anomalous losses should be considered.

An interesting modelling perspective is proposed in Ref. [155]. A neural network architecture to model hysteresis is reported by using an array of neural networks, in which each neural network is specialized to model a specific part of the hysteresis loop. Then, the total hysteresis loop is rebuilt by combining the different neural networks' predictions. The authors simulate a synthetic material's behavior by means of a Jiles–Atherton model, using simulated curves, such as sinusoidal waveforms. Another study, focused on the hysteretic behavior of a transformer core, is presented by Du et al. [137]. Their study is dedicated to forecasting a single hysteresis cycle and, generally, it is not capable to analyze the magnetic field for a long time.

Therefore, a completely different approach is proposed to fit the magnet response, based on deep neural networks. In order to realize a network able to meet the requirement in terms of NRMSE of 0.01 %, different models are investigated and chosen on the basis of a trade-off between the prediction capability of the network and the model complexity. In particular, the simplest architecture, that is the MLP, is taken as a starting point and fine-tuned to realize more complex multi-layered neural networks, able to model the magnet response. Specifically, the results show that the NARX, relying on input and output past values to understand the underlying physics, is the architecture that best models the hysteretic behaviour of iron-dominated magnets.

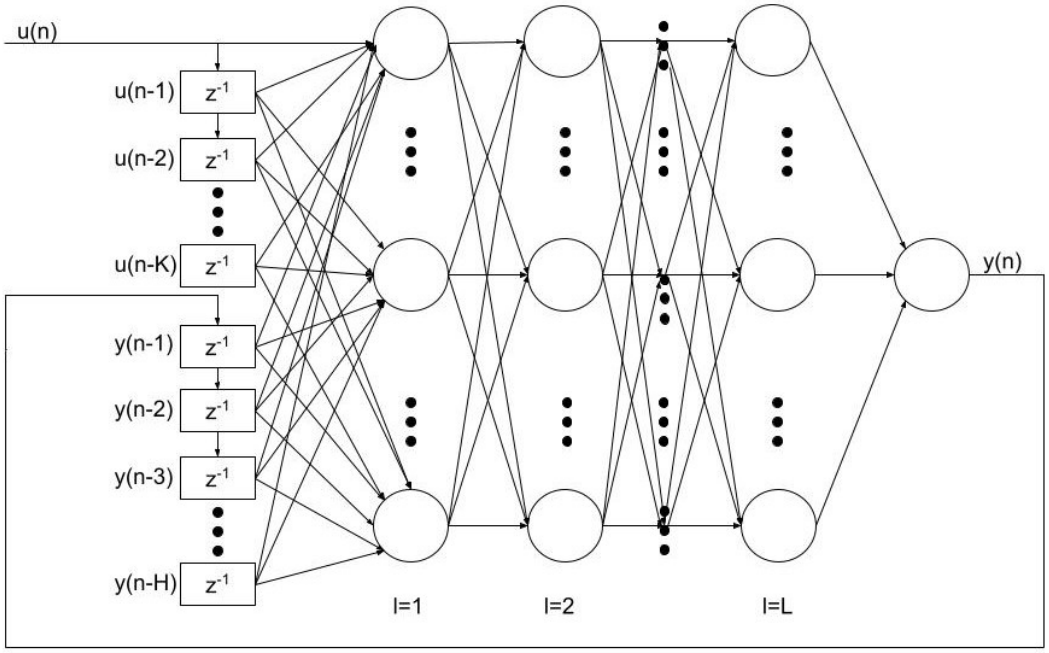

Fig. 3.2 NARX general architecture. The neurons' computation is forward: it starts from the first layer; then, it updates intermediate layers; finally, it computes the output of the network, $y(n)$.

3.2 Proposal

The phenomenon consists in the evaluation of the output y (that is the magnetic field) as a function, f, of the input u (that is the excitation current), K previous inputs and H previous outputs:

$$\begin{aligned} y(n) = f(u(n), u(n-1), ..., u(n-K), \\ y(n-1), ..., y(n-H)) \end{aligned} \tag{3.1}$$

with n a discrete time index. The past values of the magnetic field and excitation current will allow the system to model the history-dependent feature that is the magnetic hysteresis. The aim of the work is to describe Eq. 3.1 through a NARX [156]. The NARX architecture is shown in Fig. 3.2.

The general architecture consists of L layers, each one characterized by a certain number of neurons. The input of the network goes through all the layers up to the last one. The outputs a_j^1 of the A_1 neurons from the first layer $l = 1$ can be calculated by Eq. 3.2:

$$a_j^1(n) = \phi^{A_1}\left(\sum_{h=1}^{H} W_{jh}^O y(n-h) + \sum_{k=0}^{K} W_{jk}^I u(n-k)\right) \tag{3.2}$$

with ϕ^{A_1} the activation function for neurons belonging to the first layer; W_{jh}^O the weights that represent the connection's strength from the output h to the j-th neuron of the first layer, while the weights W_{jk}^I represent the connection's strength from the input k to the j-th neuron of the first layer. A similar equation (Eq. 3.3) can be used to calculate the neurons' output for the next layers:

$$a_j^l(n) = \phi^{A_l}\left(\sum_{i=0}^{A_l} W_{ji}^l a_i^{l-1}(n)\right) \tag{3.3}$$

with $l \in \{2,\ldots,L\}$ the index of the layer; ϕ^{A_l} the activation function for the neurons of the layer l; W_{ji}^l the weights representing the connection's strength from the i-th neuron of the layer $l-1$ to the j-th neuron of the layer l and the term a_0^{l-1} is set to 1 to reproduce the bias W_{j0}^l. Finally, the output neuron y is calculated by Eq. 3.4:

$$y(n) = \sum_{i=0}^{A_{L-1}} W_i^E a_i^L(n) \tag{3.4}$$

where the identity was used as activation function and the weights W_i^E represent the connection's strength from the i-th neuron of the L-th layer to the output neuron, y. To summarize, the set of hyperparameters characterizing the neural network is $\theta = (L, \mathbf{A}, K, H)$, while all the layers, $l \in \{1, L\}$, use the hyperbolic tangent as activation function $\phi^{A_l}(x)$.

3.2.1 Case Study

Experimental Setup

The measurement campaign was carried out at CERN, by using a Reference quadrupole magnet (Fig. 3.3).

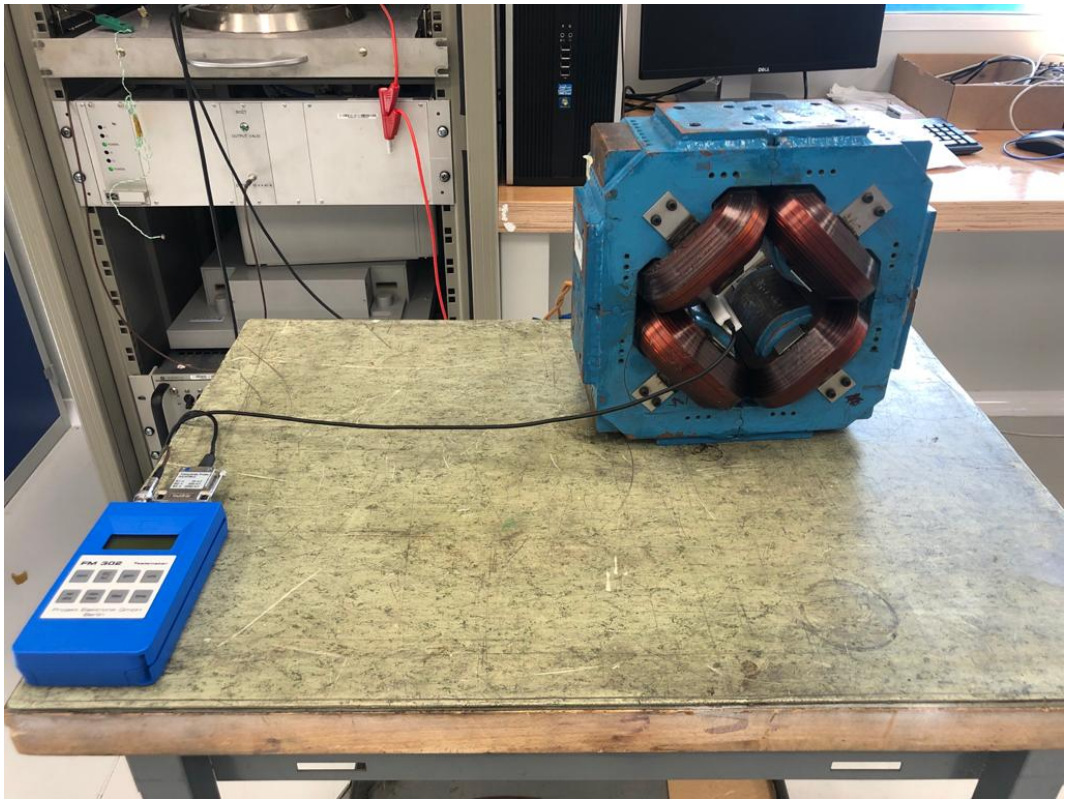

Fig. 3.3 “Quadrupole used for the measurement campaign. At the tip of the pole, the Hall probe based FM302 teslameter measures the magnetic field”.

“The magnet was fed by a power supply (A&D AG BIP1540) that is able to provide an output current up to 40 A and an output voltage of 10 V. An NI PXI 461 card, driven by custom C++ software, was used to control the power supply, while the current was measured by means of a LEM IT60 ULTRASTAB DCCT with an accuracy of 3 ppm. The magnet was excited with ten sequences of seven different cycles”. An example of some of the waveforms used is shown in Fig. 3.4. These kinds of waveforms were specifically conceived to operate with curves analogous to the ones employed in particle accelerators [157].

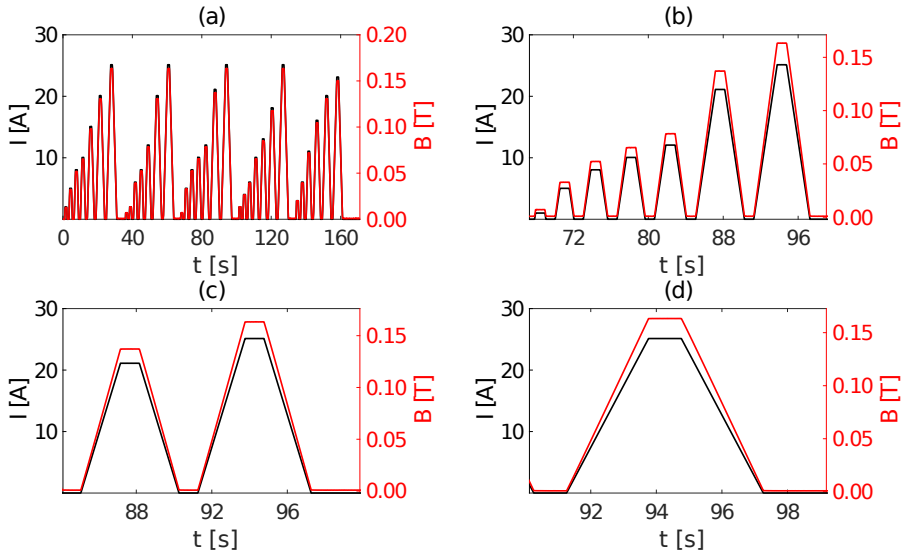

Fig. 3.4 “Sample of the excitation current sequence, $I(t)$, in black and the magnetic field sequence, $B(t)$, in red from the collected dataset. The ordinate axis has two different scales: on the left, in black, for the current, $I(t)$, and on the right, in red, for the magnetic field, $B(t)$. In Panel (a) the shape of an entire sequence of data collected is shown; while Panels (b), (c) and (d) represent a zoom of a set of seven cycles, last two and one cycles of the set of seven cycles, respectively” [1].

The two sequences of excitation current consist of seven trapezoidal cycles, each lasting approximately 5 seconds. The current starting value is zero and reaches 25 A on the higher flat-top, to which the major hysteresis loop corresponds. Before starting the acquisitions, a degaussing of the magnet was performed. The flat-top levels, representing the inversion points in the magnetic history, are all different in the different sequences, while the field ramp-rate, i.e. the slope of the cycle ramp, is constant for both test and training cycle sequences.

“A Hall probe-based FM302 teslameter (from Projekt Elektronic GmbH), characterized by a sensitivity of 1 V/T, was used to measure the magnetic field. The Hall probe was positioned at the pole of the magnet in order to measure the peak field in the gap. The same PXI card used for the power supply was used to acquire the output voltage at a raw sampling rate of 2.5 kS/s with a resolution of 24 bits. The maximum magnetic field measured is $B_{\max} = 0.16$ T, while the noise level of the measurement, computed from its standard deviation on the plateaus, is around 13 μT, i.e. $8.1 \cdot 10^{-5}$ relative to the maximum”.

The hysteresis graph is represented by the relationship, $B(I)$, between the magnetic field, B, and the current, I. This relationship can be described as the sum of two contributions (Eq. 3.5): the predominant one ($B_0 + G \cdot I$) that is a linear contribution and a smaller one ($\hat{B}(I)$) that is the nonlinear part.

$$\text{“}B(I) = B_0 + G \cdot I + \hat{B}(I)\text{”} \tag{3.5}$$

Since the relationship is essentially linear, a Linear Model [158] can be adopted for modelling the magnetic response, in the first approximation setting aside its nonlinear part. As a result, the offset B_0 and the gain G of the least-square linear regression can be computed and they correspond to $7.79 \cdot 10^{-4}$ T and $6.50 \cdot 10^{-3}$ T/A, respectively.

The residual of the Linear Regression (LR), $\hat{B}(I)$, is crucial in the construction of datasets used to train and test the networks. In fact, in opposition with existing hybrid models [149, 135, 153], the learning of the networks is devoted to the nonlinear component of magnetic hysteresis, while cutting out the linear part and modelling it with a separate LR.

Fig. 3.5 shows another representation of the magnetic behaviour in terms of transfer function T_f, defined by Eq. 3.6 as the field-to-current ratio:

$$\text{“}T_\mathrm{f}(I) = \frac{B(I)}{I} = \frac{B_0}{I} + G + \frac{\hat{B}(I)}{I}\text{”} \tag{3.6}$$

In this figure, it is possible to appreciate “how the magnetic response moves discontinuously from the lower branch of the hysteresis loop to the upper one”.

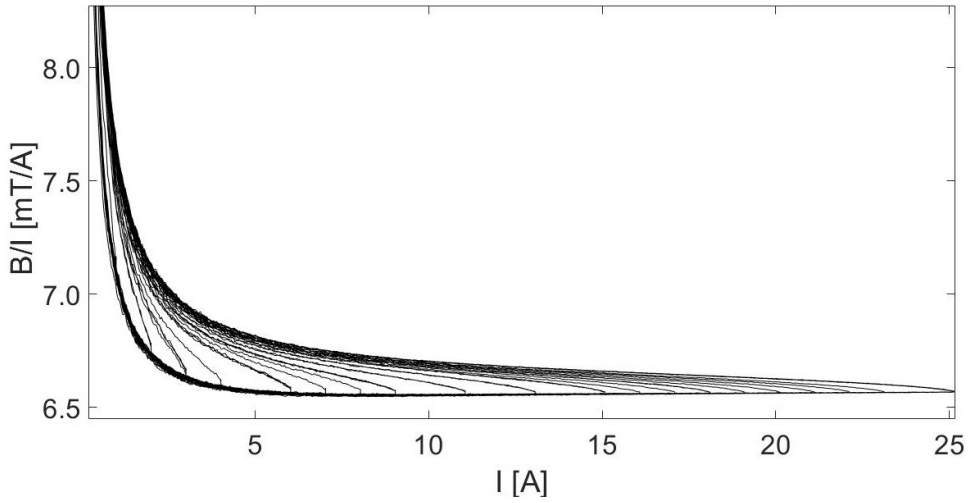

Fig. 3.5 Plot of the transfer function of the magnet ($T_{\mathrm{f}}(I)$), defined by Eq. (3.6) as "the ratio between the magnetic field and the excitation current".

Preparation of the Dataset

"The raw dataset $\bar{D}$ consists of ten sequences of excitation current and magnetic field waveforms and each sequence is composed of 7 magnetic cycles. The acquisition rate is 2.5 kS/s (sampling time 400 μs) for a total of 871 440 samples. The network architecture selection and training phases were performed on a reduced subset D that includes 87 144 samples at the sample rate of 250 S/s (corresponding to a sampling interval of 4 ms). The subsampled dataset, D, was split into two sets: D_{L} was used for the architecture's training and the remaining half, D_{E}, was used for the test and statistical error evaluation". For the first dataset, D_{L}, the data were organized in couples $[I_{\mathrm{L}}(n), B_{\mathrm{L}}(n)]$, where $I_{\mathrm{L}}(n)$ is the excitation current and $B_{\mathrm{L}}(n)$ is the magnetic field. For the dataset D_{E}, the same type of parameters in input/output of the neural architecture are used, $[I_{\mathrm{E}}(n), B_{\mathrm{E}}(n)]$. Hence, the data in D_{L} was split into training $D_{\mathrm{L}}^{\mathrm{train}}$, validation $D_{\mathrm{L}}^{\mathrm{val}}$ and test $D_{\mathrm{L}}^{\mathrm{test}}$ sets with a 60:20:20 split ratio, respectively. The training dataset contains only a few of all possible combinations of different flat-top levels. "Each combination is related to a different branch of the hysteresis loop. The accuracy of the prediction performed on the test combinations measures the interpolating power of the neural network architecture under test. In addition, another version of the datasets, $\hat{D}_{\mathrm{L}} = [\hat{I}_{\mathrm{L}}(n), \hat{B}_{\mathrm{L}}(n)]$ and $\hat{D}_{\mathrm{E}} = [\hat{I}_{\mathrm{E}}(n), \hat{B}_{\mathrm{E}}(n)]$, was considered by replacing the measured field with its nonlinear component (Eq. 3.5). The total magnetic field can be calculated by adding back the network output, $\hat{y}$, to

Table 3.1 Summary of dataset usage.

Magnetic field Input	Model Selection Dataset	Training Dataset	Test Dataset	Magnetic field Estimation
As measured	D_{L} @ 250 S/s	$D_{\mathrm{L}}^{\mathrm{train}}, D_{\mathrm{L}}^{\mathrm{val}}$ @ 250 S/s	D_{E} @ 250 S/s D_{E} @ 2.5 kS/s	$B \approx y$
Nonlinear component only	$\hat{D}_{\mathrm{L}}$ @ 250 S/s	$\hat{D}_{\mathrm{L}}^{\mathrm{train}}, \hat{D}_{\mathrm{L}}^{\mathrm{val}}$ @ 250 S/s	$\hat{D}_{\mathrm{E}}$ @ 250 S/s $\hat{D}_{\mathrm{E}}$ @ 2.5 kS/s	$B \approx B_0 + GI + \hat{y}$

the previously subtracted LR. The reason for this decomposition is to train the network on a dataset that has a much smaller dynamic range, considering the physically interesting part of the magnet's response, which is the nonlinear part". A schematic summary of the way these datasets have been used in the course of this work is given in Table 3.1.

All the architectures were designed and tested with the "neural network Toolbox in Matlab 2018b". Both simulations and training were executed on a computer with Ram 8 GB, Clock 3.2GHz and an Intel Core i5.

Model Selection and Evaluation

The overall method used to select the network architecture is illustrated schematically in Fig. 3.6.

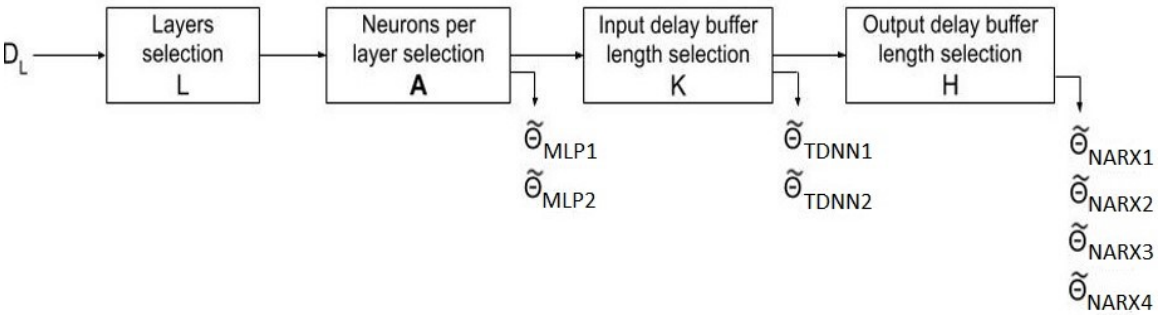

Fig. 3.6 Block diagram of the model selection. The procedure starts with the layer selection that takes as input the dataset D_{L}; once chosen the number of the layers, there is the second step consisting in the choice of the number of neurons per layer, **A**. In the third and fourth steps, there is the input (K) and output (H) delay buffer length selection, respectively. The results of the best architecture models selected during the second ($\tilde{\theta}_{MLP1}$, $\tilde{\theta}_{MLP2}$), third ($\tilde{\theta}_{TDNN1}$, $\tilde{\theta}_{TDNN2}$) and fourth ($\tilde{\theta}_{NARX1}$, $\tilde{\theta}_{NARX2}$, $\tilde{\theta}_{NARX3}$, $\tilde{\theta}_{NARX4}$) steps are shown in Tab. 3.3

Table 3.2 Definition of the hyperparameters for the model selection.

Hyperparameter	**Multilayer Perceptron**	**Time Delay**	**Autoregressive Exogenous**
L	$\{1, ..., 15\}$	$\tilde{L}_{MLP}$	$\tilde{L}_{MLP}$
$\mathbf{A}$	$\{1, ..., 10\}^L$	$\tilde{\mathbf{A}}_{MLP}$	$\tilde{\mathbf{A}}_{MLP}$
K	0	$\{1, ..., 35\}$	$\tilde{K}_{TDNN}$
H	0	0	$\{1, ..., 35\}$

Table 3.3 Hyperparameters $\tilde{\theta}$ selected by Alg. 1 and used for the tests evaluation. In the last column, the corresponding number of neuron connections $|W|$ are listed.

Hyperparameters	L	$\mathbf{A}$	K	H	$\|W\|$
$\tilde{\theta}_{MLP1}$	2	(10, 9)	0	0	129
$\tilde{\theta}_{MLP2}$	4	(1, 1, 1, 10)	0	0	37
$\tilde{\theta}_{TDNN1}$	2	(10, 9)	26	0	379
$\tilde{\theta}_{TDNN2}$	4	(1, 1, 1, 10)	31	0	67
$\tilde{\theta}_{NARX1}$	2	(10, 9)	26	31	689
$\tilde{\theta}_{NARX2}$	2	(7, 8)	26	17	381
$\tilde{\theta}_{NARX3}$	4	(1, 1, 1, 10)	31	31	98
$\tilde{\theta}_{NARX4}$	4	(1, 8, 5, 4)	31	31	153

The approach of the model selection can be summarized in four steps, corresponding to the choice of the four hyperparameters of the network: the first step, set the number of the neurons to ten per layer, consists in the choice of the number of the layers, L; in the second step, fixed the number of the layers, chosen in the previous step, the number of neurons for each layer, A_l, is selected. Therefore, for these two steps, static structures without feedback are considered. Next, the feedback on the input, K, is introduced and the third step is dedicated to choosing it. Finally, as the last step, there is the choice of the output feedback, H, to reach the NARX architecture. The intervals of values used for the selection of L, A_l, K and H are listed in Tab. 3.2.

The optimal hyperparameters found during the model selection process are shown in Tab. 3.3. Afterwards, they were used to test the models trained on the entire dataset, D_{L}.

The pseudo-code representing the selection process is presented in Algorithm 1.

Algorithm 1 contains the pseudo-code that describes the selection process. The dataset D in input to the algorithm represents both the full and the nonlinear com-

Algorithm 1 *"Model Evaluation and Selection* $(\mathscr{M}, params, D)$" [1]

Require: "set $\mathscr{M}$ of hyperparameters θ_j for each model to evaluate (see Table 3.2)
set *params* of simulation parameters (see Table 3.4 in Sec. 3.2.1)
the dataset D"
Ensure: "set *Score* of evaluations for each model in $\mathscr{M}$"

$[D_{\mathrm{L}}^{\mathrm{train}}, D_{\mathrm{L}}^{\mathrm{val}}, D_{\mathrm{L}}^{\mathrm{test}}] = Split(D_{\mathrm{L}}, 60:20:20)$
for $\theta_j \in \mathscr{M}$ **do** ▷ "$Loop1$: iterate over the set of different models"
  for $i = 1 : R$ **do** ▷ "$Loop2$: repeat learning R times"
    repeat
      $W = train(\theta_j, params, D_{\mathrm{L}}^{\mathrm{train}}, D_{\mathrm{L}}^{\mathrm{val}})$
    until *term_condition (params)* ▷ $Loop3$: training instance
    $[B_{\mathrm{L}}^{\mathrm{test}}, I_{\mathrm{L}}^{\mathrm{test}}] \leftarrow D_{\mathrm{L}}^{\mathrm{test}}$
    $y_i \leftarrow y(\theta_j, W; I_{\mathrm{L}}^{\mathrm{test}})$
    $Error(i) \leftarrow RMSE(y_i, B^{\mathrm{test}})$ ▷ as defined in Eq. 3.7
  end for
  $Score(j) = model\_evaluate(Error, \theta_j)$ ▷ calculate BC scores, Eq. 3.8
end for

ponent only dataset. The pseudo-code includes three loops. $Loop1$ is the first loop and iterates over $\mathscr{M}$, that is a "set of models each one characterized by its own hyperparameter vector (θ_j)". The model with the best prediction capability and the lowest complexity will be chosen. $Loop2$ is the second loop and contains the train function and provides an estimation of the prediction error on the test dataset $D_{\mathrm{L}}^{\mathrm{test}}$ for R times, improving the results' statistical significance. Finally, $Loop3$ represents a training instance performed according to the training dataset $D_{\mathrm{L}}^{\mathrm{train}}$, the validation dataset $D_{\mathrm{L}}^{\mathrm{val}}$ and the training parameters shown in Tab. 3.4.

Table 3.4 "Set of parameters in input to Algorithm 1" [1]

params	**Value**	**Description**
epochs	200	Max. training epochs
maxFail	6	Max. validation failures
minGrad	$1 \cdot 10^{-7}$	Min. gradient
muMax	$1 \cdot 10^{10}$	Max. μ value
R	200	Learning repetitions

The training process allows the weights W to be updated iteratively in order to reduce the prediction error using the Levenberg-Marquardt (LM) method [159],

while the hyperparameters set θ_j is kept fixed. The process will be interrupted when one of the termination criteria is satisfied. The function $term\_condition$ contains the termination criteria that includes:

- the validation error does not decrease within *maxFail* epochs;
- training epochs have reached the maximum number (*epochs*);
- "the LM damping factor, μ, overtakes its maximum value (*muMax*). During the LM algorithm, μ moves continuously between a Newton-like ($\mu \approx 0$) optimization and the steepest gradient descent ($\mu \gg 0$) one". If μ assumes too large values, this means that one is very far from a minimum, so the search has failed.

After the training process, the validation dataset, $D_{\mathrm{L}}^{\mathrm{val}}$, is used to improve the network generalization. The training will be stopped when the generalization stops improving. On the other hand, the test dataset $D_{\mathrm{L}}^{\mathrm{test}}$ does not affect the training and is only used to evaluate the network performance. One of the statistical parameters used to evaluate it is the Root Mean Square Error (RMSE) , calculated as in Eq. 3.7:

$$RMSE(y_i, D^{\mathrm{test}}) = \sqrt{\frac{\sum_{n \in N} (y_i(n) - B^{\mathrm{test}}(n))^2}{|N|}} \tag{3.7}$$

"where $y_i = y(\theta_j, W; I_{\mathrm{L}}^{\mathrm{test}})$ is estimated by Eq. (3.1), using the current set of hyperparameters and weights, and $|N|$ is the number of samples for the test dataset".

The model selection process aims to select the best network architecture by maximizing the model *evidence* $P(D|\,\theta_j)$, that indicates in terms of probability the preference expressed by the data for the $j-th$ model of hyperparameters θ_j. "Since the calculation of this probability is analytically intractable, several approximations of the term have been proposed in literature [158]. These approximations, like Akaike Information Criterion (AIC) or Bayesian Information Criterion (BIC), are based on penalization terms of the model complexity that takes into account the number of weights $|W|$ determined by the choice of hyperparameters set θ_j". For this case study, the Bridge Criterion (BC) was selected as information criterion, since it merges the benefits of both BIC and AIC [160], and it is calculated as in Eq. (3.8):

$$Score(j) = |N| \ln \left(\frac{1}{R} \sum_{i=1}^{R} Error^2(i) \right) + |N|^{2/3} \cdot (1 + 1/2 + ... + 1/|W|) \quad (3.8)$$

Thus, the best architecture is selected taking into account both prediction performance, as given by *Error*, and complexity, as defined by the number of weights, $|W|$, related to the selection of the hyperparameters. For this purpose, the function $model\_evaluate$ attributes to each hyperparameter vector, θ_j, a score based on the BC: by minimizing this term it is possible to maximize the *evidence* of the j-th model.

Static and Dynamic Network Structures

The algorithm starts with the evaluation of static network performance, with no feedback in input. In this case, the neural network under test is an MLP architecture, characterized by the hyperparameters set θ_{MLP} that includes "the number of hidden layers (L) and the number of neurons for each layer (A_l)". An example of static architecture is shown in Fig. 3.7, with two hidden layers characterized by a *tanh* activation function, while the output layer is linear.

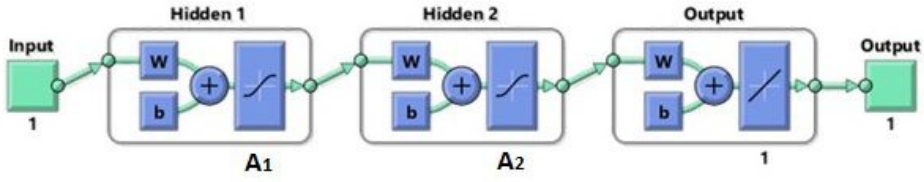

Fig. 3.7 Generic scheme of an MLP. The first green block represents the input layer, where the input is the excitation current *I(n)*; the second and the third blocks represent the two hidden layers with A_1 and A_2 neurons respectively. In each hidden layer, there are two blue boxes representing the weights, W, and the bias, b; the two hidden layers use a *tanh* activation function. The fourth block represents the output layer, which is linear. The output of the neural network is the predicted value $y(n)$ estimating the magnetic field $B(n)$ and corresponding to the last green block.

Starting with the MLP architecture, the model selection process compares different structures with a fixed number of neurons (10) per layer but with different numbers of layers, up to fifteen as shown in Tab. 3.2. This first step of the process

aims at selecting the optimal depth for the neural network architecture. According to the *model_evaluate* function, the architectures with less than 8 hidden layers have the best BC scores. In fact, after the 8-th layer, the BC scores increase, showing a worse behaviour of the network models. Therefore, structures with two and four hidden layers were chosen as suited candidates for the next step of the selection process, providing a good compromise between complexity and performance. During the second step, fixed the number of layers, the process moves on to the selection of the number of neurons (up to 10) for each layer. Therefore, with the two first steps, the process selected static MLP networks by defining the hyperparameters L and **A**, shown in Tab. 3.3. After the choice of the number of layers and neurons, the model selection process moves on to augment the static architectures by adding temporal feedback. First, it starts adding past observations of the excitation current, which means past input values. In this case, the architecture becomes a TDNN, characterized by the hyperparameters set θ_{TDNN}. The scheme of the TDNN structure used is shown in Fig. 3.8 for the case of two layers.

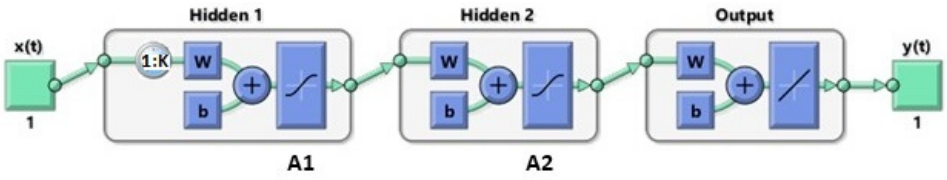

Fig. 3.8 Scheme of the TDNN. The architecture has two hidden layers, with A_1 and A_2 neurons respectively, and a delay of K units for the input. The activation function is *tanh* in the two hidden layers and linear in the output layer.

During this step, the process evaluates the effect of the third hyperparameter that is the input delay buffer length, K. Finally, in the last step, the process includes the evaluation of the last hyperparameter, that is the length of the output buffer, H. In Fig. 3.9 the scheme of a NARX architecture with two layers is shown.

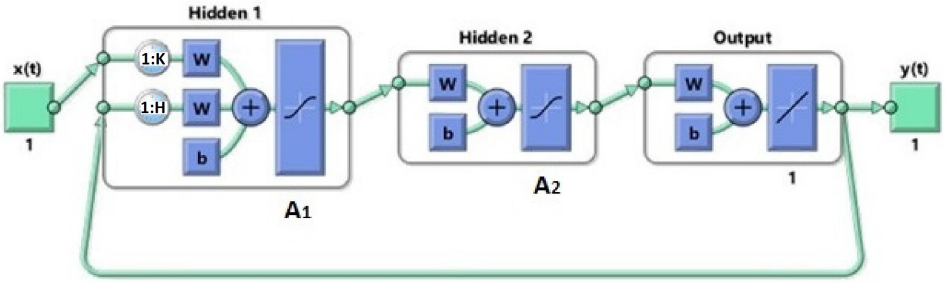

Fig. 3.9 Scheme of a two-layer NARX. The architecture $L = 2$ layers, with A_1 and A_2 neurons respectively, and a delay buffer length of K and H units for the input and output respectively.

In this case, using the optimal hyperparameters set formerly selected, that are L, $\mathbf{A}$ and K, the process only varies the hyperparameter H in the range $\{1,\ldots,35\}$. The resulting network with hyperparameters θ_{NARX} is a NARX architecture that includes the length of the output buffer, H. Tab. 3.3 shows the best hyperparameters sets ($\tilde{\theta}$) chosen by Alg. 1.

Results and Discussions

An entirely new dataset, D_{E}, was used to evaluate the performance of the best models characterized by the hyperparameters listed in Tab. 3.3. In order to calculate the $RMSE(y,D_{\mathrm{E}})$, $y = y(\theta,W;I_{\mathrm{E}})$ was estimated for each model. "To facilitate the comparison with the requirements, the RMSE values were normalized with respect to the maximum measured field. As a result, the results are also provided in terms of NRMSE:

$$NRMSE(y,D_{\mathrm{E}}) = \frac{RMSE(y,D_{\mathrm{E}})}{B_{\max}} \cdot 100 \tag{3.9}$$

In addition to the NRMSE, other two measures of performance were introduced, i.e. the Maximum Absolute Error (MAE) :

$$MAE(y,D_{\mathrm{E}}) = \max\left\{|y(n) - B_{\mathrm{E}}(n)|\right\}_{n\in N} \tag{3.10}$$

and the Maximum Percent Error (MPE) , normalized with respect to the maximum measured field:

$$MPE(y,D_{\mathrm{E}}) = \frac{MAE(y,D_{\mathrm{E}})}{B_{\max}} \cdot 100 \tag{3.11}$$

The test dataset, D_{E}, consists of samples acquired at the same sample rate of the learning dataset, that is 250 S/s. A further performance evaluation is performed on a second test dataset, $\bar{D}_{\mathrm{E}}$, characterized by samples acquired at the raw sample rate of 2.5 kS/s".

The tables containing the results for all the models are presented in Tables 3.5 and 3.6.

Table 3.5 "Comparison of the performances among the different architectures, evaluated on the test datasets D_{E}, considering the decimated data rate of 250 S/s. The evaluation has been performed through the RMSE, NRMSE, MAE and MPE first for the LR alone, then for the neural networks trained on the full dataset D_{L} and finally for the hybrid models LR+*, where the LR is combined with the neural network trained on the nonlinear component only $\hat{D}_{\mathrm{L}}$. In this last case, the reconstructed magnetic field is computed by Eq.3.5: the linear component ($B_0 + G \cdot I$) is computed through the LR coefficients; the nonlinear component ($\hat{B}$) is approximated by the neural network output. The corresponding hyperparameters are listed in Tab. 3.3" [1].

Architecture	Test Dataset	Hyperparameters	RMSE [T]	NMRSE (%)	MAE [T]	MPE (%)
Linear Regression		G, B_0	$9.07 \cdot 10^{-04}$	$5.67 \cdot 10^{-01}$	$2.60 \cdot 10^{-03}$	1.61
MLP1	D_{E}	$\tilde{\theta}_{MLP1}$	$8.70 \cdot 10^{-04}$	$5.44 \cdot 10^{-01}$	$2.60 \cdot 10^{-03}$	1.64
MLP2	D_{E}	$\tilde{\theta}_{MLP2}$	$8.83 \cdot 10^{-04}$	$5.52 \cdot 10^{-01}$	$2.50 \cdot 10^{-03}$	1.55
TDNN1	D_{E}	$\tilde{\theta}_{TDNN1}$	$7.95 \cdot 10^{-04}$	$4.97 \cdot 10^{-01}$	$1.90 \cdot 10^{-03}$	1.21
TDNN2	D_{E}	$\tilde{\theta}_{TDNN2}$	$8.05 \cdot 10^{-04}$	$5.03 \cdot 10^{-01}$	$2.20 \cdot 10^{-03}$	1.39
NARX1	D_{E}	$\tilde{\theta}_{NARX1}$	$2.12 \cdot 10^{-05}$	$1.32 \cdot 10^{-02}$	$3.36 \cdot 10^{-04}$	$2.10 \cdot 10^{-01}$
NARX2	D_{E}	$\tilde{\theta}_{NARX2}$	$2.13 \cdot 10^{-05}$	$1.33 \cdot 10^{-02}$	$3.17 \cdot 10^{-04}$	$1.98 \cdot 10^{-01}$
NARX3	D_{E}	$\tilde{\theta}_{NARX3}$	$2.05 \cdot 10^{-05}$	$1.28 \cdot 10^{-02}$	$3.11 \cdot 10^{-04}$	$1.95 \cdot 10^{-01}$
NARX4	D_{E}	$\tilde{\theta}_{NARX4}$	$2.05 \cdot 10^{-05}$	$1.28 \cdot 10^{-02}$	$3.12 \cdot 10^{-04}$	$1.95 \cdot 10^{-01}$
LR+MLP1	D_{E}	$G, B_0, \tilde{\theta}_{MLP1}$	$8.00 \cdot 10^{-04}$	$5.00 \cdot 10^{-01}$	$1.90 \cdot 10^{-03}$	1.16
LR+MLP2	D_{E}	$G, B_0, \tilde{\theta}_{MLP2}$	$8.00 \cdot 10^{-04}$	$5.00 \cdot 10^{-01}$	$1.90 \cdot 10^{-03}$	1.17
LR+TDNN1	D_{E}	$G, B_0, \tilde{\theta}_{TDNN1}$	$8.05 \cdot 10^{-04}$	$5.03 \cdot 10^{-01}$	$1.90 \cdot 10^{-03}$	1.21
LR+TDNN2	D_{E}	$G, B_0, \tilde{\theta}_{TDNN2}$	$7.97 \cdot 10^{-04}$	$4.98 \cdot 10^{-01}$	$1.90 \cdot 10^{-03}$	1.20
LR+NARX1	D_{E}	$G, B_0, \tilde{\theta}_{NARX1}$	$7.99 \cdot 10^{-04}$	$4.99 \cdot 10^{-01}$	$1.90 \cdot 10^{-03}$	1.19
LR+NARX3	D_{E}	$G, B_0, \tilde{\theta}_{NARX2}$	$7.99 \cdot 10^{-04}$	$5.00 \cdot 10^{-01}$	$1.90 \cdot 10^{-03}$	1.19
LR+NARX3	D_{E}	$G, B_0, \tilde{\theta}_{NARX3}$	$8.00 \cdot 10^{-04}$	$5.00 \cdot 10^{-01}$	$1.90 \cdot 10^{-03}$	1.20
LR+NARX4	D_{E}	$G, B_0, \tilde{\theta}_{NARX4}$	$8.01 \cdot 10^{-04}$	$5.01 \cdot 10^{-01}$	$1.90 \cdot 10^{-03}$	1.19

Table 3.6 "Comparison of the performance among the different architectures, evaluated on the test dataset $\bar{D}_{\mathrm{E}}$, at the full data rate of 2.5 kS/s. The evaluation has been performed through the RMSE, NRMSE, MAE and MPE, first, for the LR alone, then for the neural networks trained on the full dataset D_{L} and, finally, for the hybrid models combining LR with the neural network trained on the nonlinear component only ($\hat{D}_{\mathrm{L}}$). The corresponding hyperparameters are listed in Tab. 3.3" [1].

Architecture	Test Dataset	Hyperparameters	RMSE [T]	NMRSE (%)	MAE [T]	MPE (%)
Linear Regression		G, B_0	$9.07 \cdot 10^{-04}$	$5.67 \cdot 10^{-01}$	$2.60 \cdot 10^{-03}$	1.61
MLP1	$\bar{D}_{\mathrm{E}}$	$\tilde{\theta}_{MLP1}$	$8.70 \cdot 10^{-04}$	$5.44 \cdot 10^{-01}$	$2.60 \cdot 10^{-03}$	1.64
MLP2	$\bar{D}_{\mathrm{E}}$	$\tilde{\theta}_{MLP2}$	$8.83 \cdot 10^{-04}$	$5.52 \cdot 10^{-01}$	$2.50 \cdot 10^{-03}$	1.55
TDNN1	$\bar{D}_{\mathrm{E}}$	$\tilde{\theta}_{TDNN1}$	$1.10 \cdot 10^{-03}$	$6.66 \cdot 10^{-01}$	$2.70 \cdot 10^{-03}$	1.67
TDNN2	$\bar{D}_{\mathrm{E}}$	$\tilde{\theta}_{TDNN2}$	$8.52 \cdot 10^{-04}$	$5.32 \cdot 10^{-01}$	$2.50 \cdot 10^{-03}$	1.56
NARX1	$\bar{D}_{\mathrm{E}}$	$\tilde{\theta}_{NARX1}$	$9.92 \cdot 10^{-06}$	$6.20 \cdot 10^{-03}$	$4.63 \cdot 10^{-05}$	$2.89 \cdot 10^{-02}$
NARX2	$\bar{D}_{\mathrm{E}}$	$\tilde{\theta}_{NARX2}$	$1.27 \cdot 10^{-05}$	$8.00 \cdot 10^{-03}$	$6.90 \cdot 10^{-05}$	$4.31 \cdot 10^{-02}$
NARX3	$\bar{D}_{\mathrm{E}}$	$\tilde{\theta}_{NARX3}$	$9.22 \cdot 10^{-06}$	$5.80 \cdot 10^{-03}$	$3.98 \cdot 10^{-05}$	$2.49 \cdot 10^{-02}$
NARX4	$\bar{D}_{\mathrm{E}}$	$\tilde{\theta}_{NARX4}$	$9.28 \cdot 10^{-06}$	$5.80 \cdot 10^{-03}$	$4.06 \cdot 10^{-05}$	$2.54 \cdot 10^{-02}$
LR+MLP1	$\bar{D}_{\mathrm{E}}$	G, B_0, $\tilde{\theta}_{MLP1}$	$8.00 \cdot 10^{-04}$	$5.00 \cdot 10^{-01}$	$1.90 \cdot 10^{-03}$	1.16
LR+MLP2	$\bar{D}_{\mathrm{E}}$	G, B_0, $\tilde{\theta}_{MLP2}$	$8.00 \cdot 10^{-04}$	$5.00 \cdot 10^{-01}$	$1.90 \cdot 10^{-03}$	1.17
LR+TDNN1	$\bar{D}_{\mathrm{E}}$	G, B_0, $\tilde{\theta}_{TDNN1}$	$8.05 \cdot 10^{-04}$	$5.03 \cdot 10^{-01}$	$1.90 \cdot 10^{-03}$	1.18
LR+TDNN2	$\bar{D}_{\mathrm{E}}$	G, B_0, $\tilde{\theta}_{TDNN2}$	$7.97 \cdot 10^{-04}$	$4.98 \cdot 10^{-01}$	$1.90 \cdot 10^{-03}$	1.18
LR+NARX1	$\bar{D}_{\mathrm{E}}$	G, B_0, $\tilde{\theta}_{NARX1}$	$7.98 \cdot 10^{-04}$	$4.99 \cdot 10^{-01}$	$1.90 \cdot 10^{-03}$	1.17
LR+NARX2	$\bar{D}_{\mathrm{E}}$	G, B_0, $\tilde{\theta}_{NARX2}$	$7.98 \cdot 10^{-04}$	$4.99 \cdot 10^{-01}$	$1.90 \cdot 10^{-03}$	1.17
LR+NARX3	$\bar{D}_{\mathrm{E}}$	G, B_0, $\tilde{\theta}_{NARX3}$	$7.99 \cdot 10^{-04}$	$5.00 \cdot 10^{-01}$	$1.90 \cdot 10^{-03}$	1.17
LR+NARX4	$\bar{D}_{\mathrm{E}}$	G, B_0, $\tilde{\theta}_{NARX4}$	$8.00 \cdot 10^{-04}$	$5.00 \cdot 10^{-01}$	$1.90 \cdot 10^{-03}$	1.17

For each model, the optimal hyperparameter set (listed in Tab. 3.2) and the corresponding error norms are shown in both tables. The output of the LR model gives an "MPE of the order of the percent, which corresponds to the relative width of the hysteresis loop". While in other contexts this type of error might be an indication of a good linearity of the magnet under test, it is unacceptable for this application. As next step, the results of the architectures trained on the D_{L} dataset are analyzed and are reported in the upper half of Tab. 3.5 and Tab. 3.6. The results of "both the static (MLP) and the dynamic neural networks with input feedback (TDNN)" are of the same order of magnitude as the LR. For the NARX models, instead, the performances are much better, gaining two orders of magnitude and reaching an NRMSE of 0.006 % in the best case, while with the reduced dataset, the results are around a factor two worse (0.01 %). Thanks to the output feedback, which consists of the past values of the magnetic field, the NARXs prove to be able to reconstruct the dynamics of the magnet very accurately. Furthermore, it is worth noting "that the maximum length of the output buffer, $K = 35$, corresponds during the training phase to a total duration of 140 ms, which is shorter than the time interval needed to cover

Table 3.7 Performances comparison among different solutions that represent the state of the art for the modelling of hysteresis [1].

Architecture	**Metric**	**Value**
Deep Neural Network (MLP with two hidden layers), Ref. [137]	Root Mean Square Error	0.13 %
Preisach + Feed-forward neural network (one hidden layer), Ref. [135]	Maximum Absolute Error	13 %
Preisach, Ref. [143]	Relative Error	0.2 %
Preisach + Recurrent Neural Network, Ref. [149]	NRMSE	0.7%
Neural Network, Ref. [150]	Relative Error	< 8 %
Genetic Algorithm + Neural Network, Ref. [153]	Mean Square Error	< 5 %

even only two consecutive inversion points of the magnetic cycle. In the classical approaches, such as the Preisach models, the whole sequence of inversion points is indispensable as input to the model to reconstruct accurately the magnetic history. Instead, with the NARX architecture, the information regarding the magnetic history appears to be encoded implicitly by the neural network, despite the shortness of the output delay buffer. Therefore, the role of these input/output buffers might be limited to the reconstruction of the short-term dynamics (for instance, the ripple of a power supply or the decay of eddy currents)". The improved performances of the neural networks, shown in Tab. 3.6, with a shorter duration in the case of a higher sampling rate (buffer duration of 14 ms) seem to confirm this hypothesis.

In Tab. 3.7, a comparison between the results listed in Tables 3.5 and 3.6 and those in literature representing the state of art of similar reconstruction problems are reported.

In Ref. [137], "a Deep Neural Network has been used to reconstruct the magnetization curve with an RMSE of 0.13 %". The network performs as the MLP model ($8.70 \cdot 10^{-4}$ T) and performs worse than the NARX architecture that is able to reach an RMSE of $2.12 \cdot 10^{-5}$ T. In Ref. [135], a hybrid approach consisting of a "Preisach memory block plus a feed-forward Neural Network has been used to model the magnetic hysteresis. In this case, the solution reaches a maximum prediction error of around 13 %, higher than the MAE for a NARX network that is of the order

of 10^{-4}" (Tab. 3.5). In Ref. [143], a classical approach based on Preisach has been used to model the hysteresis of a combined magnet. This method allows obtaining a relative error around 0.2 %, while with a NARX architecture it is possible to achieve a Mean Percent Error of about 0.2 %. In Ref. [149], a combined approach based on Preisach and Recurrent Neural Network has been proposed to model the hysteretic behaviour of the ARMCO pure iron. The results show an NRMSE of about 0.7 % in the prediction of the magnetic flux density, worse respect to the NRMSE reached with a NARX network that is in the order of 10^{-2} (Tab. 3.5). In Ref. [150], a neural network architecture has been presented to model nonlinear hysteretic inductors. The results show a relative error lower than 8 %, higher than the MPE obtained with a NARX network that is around $2 \cdot 10^{-1}$ % (Tab. 3.5). Lastly, in Ref. [153] a hybrid approach based on Genetic Algorithm and Neural Network has been presented for the hysteresis modelling, achieving "a Mean Square Error lower than 5 %. Tab. 3.5 shows an RMSE in the order of 10^{-5} for the NARX network, reaching therefore a better result with respect to the literature".

In Fig. 3.10 and in Fig. 3.11, the nonlinear component of the field, $\hat{B}_E$, predicted by a NARX architecture is presented "as a function of the time or the current", respectively. The measured and the predicted fields match closely, showing the ability of the NARX architecture to model the nonlinear part of the magnetic field with high accuracy.

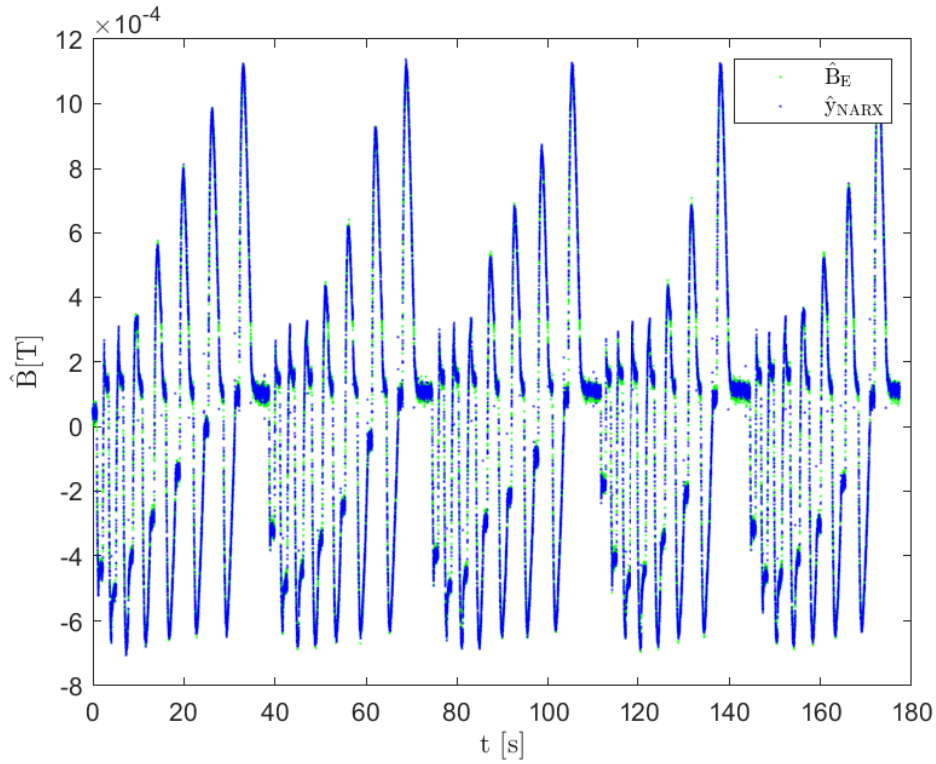

Fig. 3.10 Measured ($\hat{B}_{\mathrm{E}}$, in blue) and estimated ($\hat{y}_{\mathrm{NARX}}$, in green, with $\tilde{\theta}_{NARX1}$ as hyperparameters set) nonlinear part of the magnetic field, $\hat{B}$, as function of the time, t.

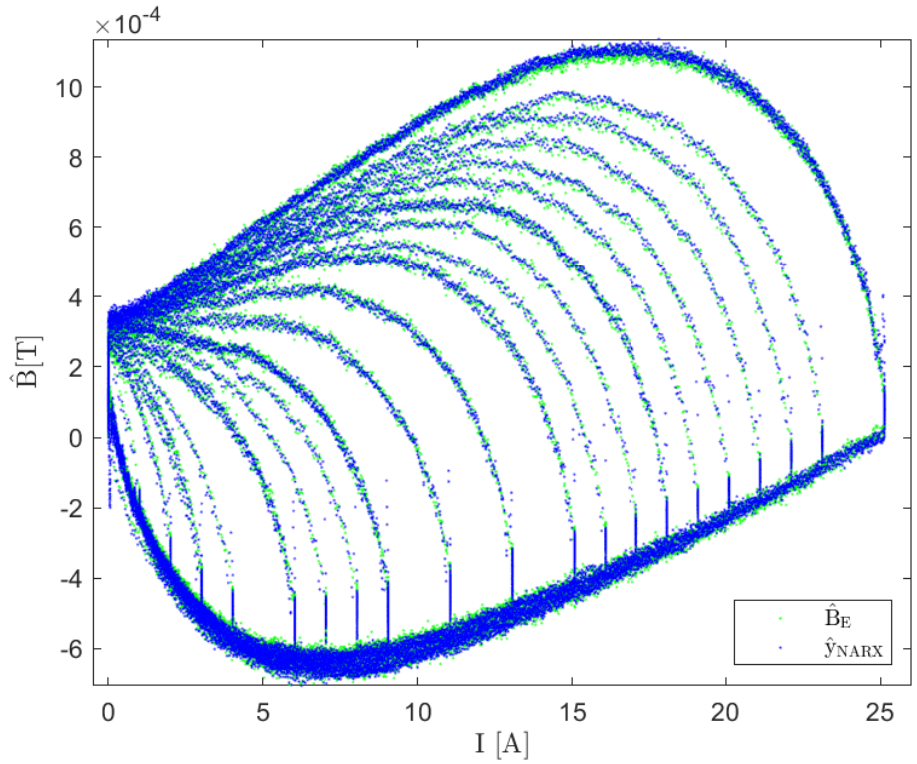

Fig. 3.11 Measured ($\hat{B}_{\mathrm{E}}$, in blue) and estimated ($\hat{y}_{\mathrm{NARX}}$, in green, with $\tilde{\theta}_{NARX1}$ as hyperparameters set) nonlinear part of the magnetic field, $\hat{B}$, as function of the current, I.

Finally, a hybrid approach was considered to predict the magnetic field: the linear part is modeled by the LR while the nonlinear component is modeled by the networks that, in this case, are trained on the nonlinear component dataset, $\hat{D}_{\mathrm{L}}$ (bottom halves of Tables 3.5 and 3.6). With the hybrid modelling, all the architectures learned on $\hat{D}_L$ and combined with the LR perform almost equally. In particular, the results are comparable with those of the MLP and TDNN networks trained on the entire dataset D_{L}, showing the inability of the hybrid architectures to reach the higher performance of the NARX trained on the entire signal contained in D_{L}. As result, the best way to model the dynamic part of the magnetic field is using input and output delay buffers in NARX architectures and avoiding pre-processing data through the extrapolation of the nonlinear component of the signal. In this case, an NRMSE of 0.006 % can be reached in the best case. Furthermore, a statistical performance evaluation by using "one-way analysis of variance (ANOVA)" was performed in terms of Absolute Errors for both datasets D_{E}, reported in Tab. 3.5, and $\bar{D}_{\mathrm{E}}$, reported in Tab. 3.6. Starting with ANOVA on D_{E}, the test showed "a significant

statistical effect on the different groups with $F[16,754 \cdot 10^3] = 13 \cdot 10^3$, $p < 10^7$. Then, the post hoc Tukey's test also revealed a significant difference from all the models to the LR (null hypothesis). The analysis shows that the model TDNN2, the best performing architecture excluding NARXs, is not statistically different from TDNN1, MLPs and LN+TDNN2 ($p > 0.01$). Conversely, the test shows that all the NARX models are statistically different from the other models ($p < 10^{-5}$), while they are not statistically different among them ($p > 0.01$). On the dataset $\bar{D}_{\mathrm{E}}$, the ANOVA analysis shows again a significant statistical effect on the different groups with $F(16,754 \cdot 10^3) = 14 \cdot 10^3$, $p < 10^{-7}$. Post hoc analyses show that the models resulting from a hybrid approach (LR+$*$) are not significantly different among them ($p > 0.01$). Furthermore, the LR+$*$ models' group is not statistically different ($p > 0.01$) from, at least, one of the MLP models group. This result confirms that the hybrid models could at best replicate the performance of the MLP models, and are unable to reach the better performance of the NARX architectures to reconstruct the original signal". As regards the TDNN models, the test shows that TDNN2 is not statistically different ($p > 0.01$) from MLP1, while TDNN1 is a model different from the others, performing as the worst one. This confirms that changing the input buffer length is not enough for the TDNN models "to let them adapt to the signal when a different input rate is given. Fig. 3.12 illustrates the results of the ANOVA analysis performed on the dataset $\bar{D}_{\mathrm{E}}$, showing the Absolute Error together with the 95 % confidence interval for each model (straight line)".

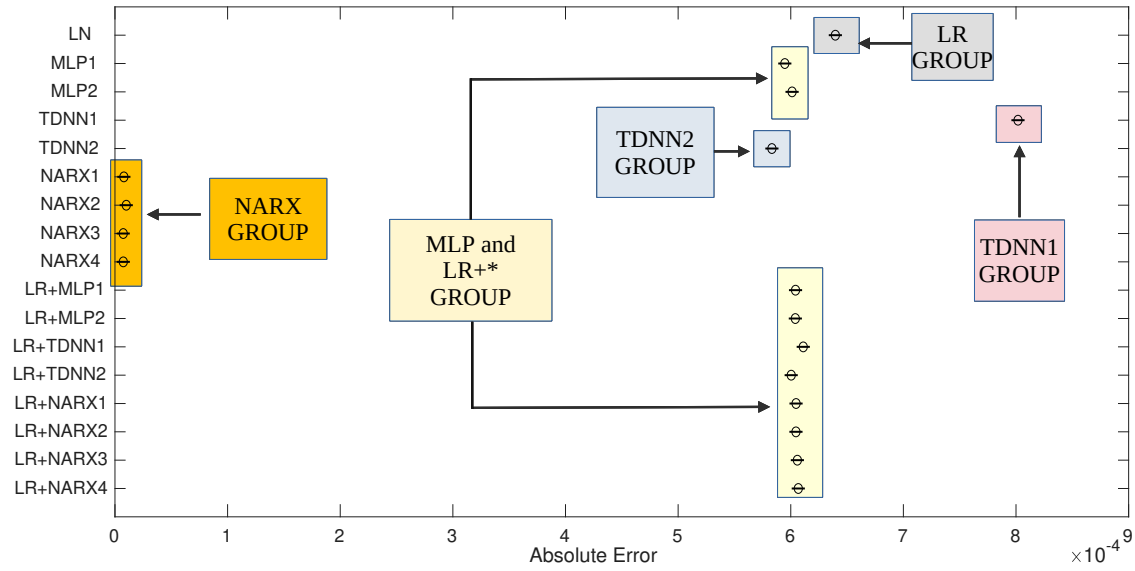

Fig. 3.12 ANOVA analysis results in terms of "Absolute Errors for all the models on Dataset $\bar{D}_{\mathrm{E}}$. For each model, the horizontal lines represent the 95 % confidence interval". All the models can be clustered into five groups: LR group in gray that is the baseline; "MLP and LR+$*$ group in yellow, TDNN1 in pink, TDNN2 in blue and the NARX group in orange" [1].

The models can be partitioned into four groups with the LR as a baseline, in gray. Only TDNN1 (in pink) performs worst of the LR group since it completely fails to model the magnet behaviour at the higher sample rate of $\bar{D}_{\mathrm{E}}$. The behavior of the LR+$*$ group is comparable to the MLPs one, in yellow, and they perform better than the LR group. TDNN2, in blue, performs slightly better than the yellow group, while the NARX group, in orange, has the best performances since it manages to adapt to the new sample rate, significantly overcoming the other groups' performances. Finally, an evaluation of the training and simulation time of the architectures whose hyperparameters are listed in Tab. 3.3 was performed and the results are listed in Tab. 3.8.

Table 3.8 Training and simulation times. The neural networks considered for the times' evaluation are the ones trained on the entire dataset D_{L} [1].

Architecture	**Training time** [s]	**Simulation time** (D_{E}) [s]	**Simulation time** ($\bar{D}_{\mathrm{E}}$) [s]
MLP1	16.88	0.40	1.08
MLP2	14.18	0.44	0.88
TDNN1	46.30	1.73	12.50
TDNN2	22.13	1.50	16.16
NARX1	125.52	2.66	25.23
NARX2	42.96	2.30	22.23
NARX3	19.91	2.91	27.98
NARX4	15.89	2.84	28.67

The first column of Tab. 3.8 shows all the models; the second column shows the training times of the "networks trained on the dataset D_{L}. The training time represents the time needed for the execution of the function Train (line 5 in Alg. 1). The third column contains the simulation time evaluated on the test dataset D_{E}, at the decimated sample rate of 250 S/s (Tab. 3.5), while the fourth column shows the simulation time evaluated on the dataset $\bar{D}_{\mathrm{E}}$) at full sample rate of 2.5 kS/s (Tab. 3.6). The simulation time represents the time needed for the network prediction (see line 8 in Alg. 1)". The results show how the required time of execution of the different steps in Alg. 1 grows with the complexity of the model and the results nicely match with the complexity measure in Tab. 3.3.

Conclusions

An incremental approach to select an optimal deep neural network to model the magnetic field produced by a magnet powered by sequences of cyclic excitation current was presented. The tests were carried out in conditions typical of those characterizing "particle accelerators and other similar pulsed-mode machines". The magnet's response is essentially linear and the work focuses on estimating its residual nonlinear component, dominated by ferromagnetic hysteresis. The training and testing performed on the raw datasets show that networks with feedback based on both input and output delay buffers reach the required level of performance, that is an

NRMSE better than 0.01 %. This level of performance is not reachable with simpler networks with only input buffers, such as TDNN architectures, or without buffers, as the MLP architectures. The NARX architectures' performance is encouraging for future applications in such field. Furthermore, it is worth noting "that the prediction accuracy generally improves if the network is trained on a dataset characterized by a low sample rate (250 S/s) but it is tested on a dataset at a higher sample rate (2.5 kS/s). This result may be related to the fact that the dataset at a low sample rate is less affected by noise, thus allowing the model to better focalize on apprehending the underlying dynamics". As a result, NARX architecture can be implemented in hardware in the context of a revamping of real-time systems used for magnetic measurements, currently taking place at CERN. This upgrade of the system will provide continuous field prediction in addition to the measurement, allowing to collect a large quantity of data regarding cycles sequences in thousands of different variations. Therefore, it will be possible to cover all the relevant dynamic scenarios and fine-tune the hyperparameters of the network, estimating their performance and robustness with high statistical significance in the long term.

Chapter 4

Transfer Learning for Healthcare: Fractures Detection in Patients with Maxillofacial Trauma

4.1 Background

The second case study focuses on deep learning techniques in the healthcare context. Over the past few years, the medical prescriptions for Magnetic Resonance Imaging (MRI), CT scan and all the other radiology services have grown enormously [161]. In this context, AI can support radiologists in the challenging tasks of medical image analysis. Even if the AI-based tools can not replace medical staff, they can confirm or validate radiologists' decisions and doubts, becoming a precious assistive technology. Deep learning has achieved a lot of results and made significant progress in images analysis tasks resulting in a better understanding and representation of complex data. Several work [162–166] deal with solutions based on deep learning for orthopedic traumatology, however, the possibility to identify the presence of maxillofacial fractures in CT scans (3D images) of harmed patients using artificial neural networks based on a transfer learning approach has not been explored yet [167–170].

In their work [171], Kim and MacKinnon focus on transfer learning to use pre-trained CNNs on non-medical images in the radiological field. In particular, they use inception V3 CNN [172] to detect the presence of fracture on wrist radiographs.

By means of the transfer learning approach, they reach an Area Under the Receiver Operating Characteristic Curve (AUC-ROC) of 0.95, on the test dataset. The result shows that a deep neural network pre-trained on a non-medical dataset can be used for medical radiographs with high performances. On the other hand, Chung et al. [173] employ a CNN to reveal and classify proximal humerus fractures by means of plain anteroposterior shoulder radiographs. In this case, the network performs similar to shoulder-specialized orthopedic surgeons but better than general physicians or non-shoulder specialized orthopedic surgeons. So, the results show the possibility to diagnose proximal humerus fractures accurately with CNN on plain radiographs. Tomita et al. [174] carried out another study in this domain, focusing on the detection of osteoporotic vertebral fractures in CT exams. The system includes two blocks: a first block used to extract radiological features from the CT scans; a second block consisting in a Recurrent Neural Network (RNN) unit (see Section A.2 in Appendix A for more details about RNN) used to combine the previously extracted features for the conclusive diagnosis. The results show that the system fits the radiologist practitioners' ability. Therefore, the system could be useful in the medical context for screening the CT exams and determining priorities of potential fracture cases.

4.2 Motivation and Proposal

The anatomical complexity of the maxillofacial region together with the uniqueness of this kind of fracture make the diagnosis complex, with the risk of causing incongruous hospitalizations. In this scenario, an automatic fracture detection system would help medical staff to detect maxillofacial fractures and it would be a decision support tool in clinical practice, minimizing treatment costs and distress for patients. At this aim, a Maxillofacial Fracture Detection System (MFDS), based on CNNs and transfer learning (see Chapter 2 for more details), is proposed to identify traumatic patients' fractures. The architecture, pre-trained on non-medical images, was re-trained and fine-tuned using CT images, that are used as input of the system. The goal of the neural network is the classification of future CTs of injured patients with maxillofacial trauma as either "fracture" or "noFracture". In Fig. 4.1, the block diagram of the system is proposed.

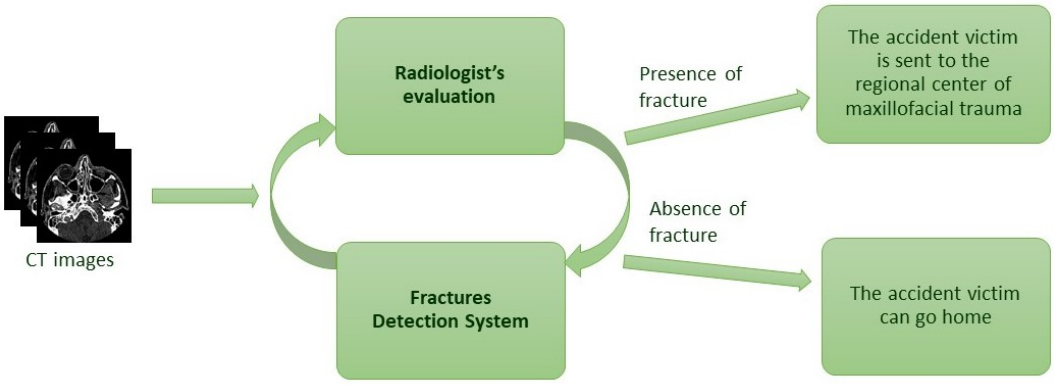

Fig. 4.1 Block diagram of the fracture detection system for injured patients with possible maxillofacial fractures. The system would be a useful tool in assisting radiologists in the CT exam evaluation of an injured patient. The CT images of a harmed patient are the input of the system, while the output of the system points out the presence or not of a fracture.

4.2.1 Case study

Experimental Setup

The dataset consists of anonymized CT exams (a CT exam for each patient) obtained from the U.O.C. of Maxillofacial Surgery of the University Hospital "Federico II" that gathers examinations performed from 2000 to 2020. The research was approved by the Ethics Committee of "Federico II" University in Naples, Italy (Approval number 81/20). The CT examinations of the facial mass were performed by using different devices (CT 16-64 slices) with thickness volumetric acquisition variable in the range of 0.5-2 mm and also variable in-plane resolution (0.5x0.5-1x1 mm). Only the images obtained by means of the bone reconstruction algorithm were gathered for the study. Two radiologists consensually explored, analyzed and classified the CT images according to fracture presence/absence. The dataset also includes CT examinations from patients with non-traumatic facial mass disorder. In particular, the total dataset is made up of 208 patients: 170 patients (11 260 slices of CT images) labeled as with "fracture" and 38 patients (49 762 slices of CT images) labeled as "noFracture". The dataset has been partitioned into training, validation and test dataset. The training dataset is made up of 148 CT scans (120 patients with fracture and 28 patients without fracture). The validation dataset, used for statistical analysis, consists of 30 patients (25 with fracture and 5 without fracture),

as well as the test dataset that is used for final testing. It should be noted that the full dataset is unbalanced on patient-level considering the large majority of patients with a fracture; however, on a slice-level the dataset is unbalanced in favor of images without fracture. Therefore, the total dataset is unbalanced in favor of images with or without fracture depending at which level the study is carried out. The implementation of the system is schematized in Fig. 4.2 and was performed by means of a predictive algorithm. Python v.3.7.6 (available for different Operating Systems) [175] was used as programming language together with PyTorch v.1.4.0 [176] and Fastai v.1.0.60 [177]. Scikit-learn v.0.22.1 [178] was used for the neural architecture, while Pydicom v.1.4.2 [179] was used to treat CT images in Dicom format.

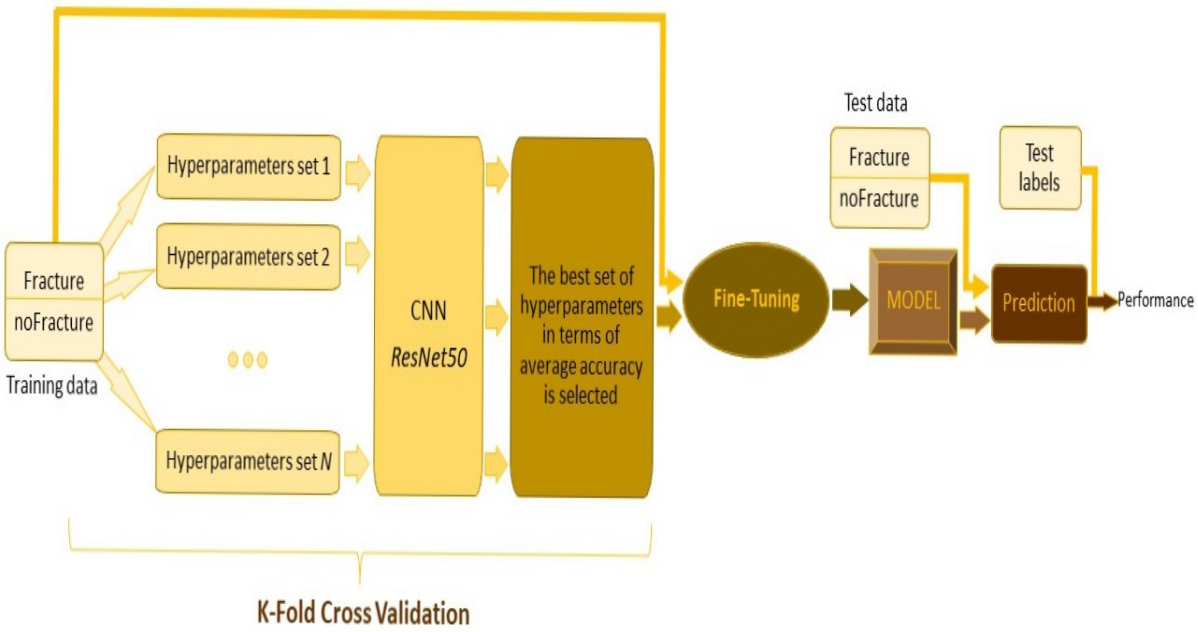

Fig. 4.2 Block diagram of the fracture detection system's implementation. Three main steps can be detected: the first step consists of the K-Fold Cross Validation, used to identify the network's hyperparameters; the second step consists of the fine-tuning of the network and finally, the last step, is represented by the evaluation of the network's performance.

Three main steps can be identified in the block diagram shown in Fig. 4.2:

1. Identification of the hyperparameters (learning rate, weight decay and drop out) by means of the K-Fold Cross Validation. The hyperparameters set that

guarantees the network to have the best performance in terms of accuracy is chosen;

2. Fine-tuning of the neural network with the hyperparameters set chosen in the previous step:
   (a) Training only the last layer
   (b) Unfreezing and training the full model
3. Estimate of the network's performance.

The three main steps listed above are described in detail in the next paragraphs.

K-Fold Cross Validation

The starting point of the system's implementation shown in Fig. 4.2 is the definition of the training dataset to be used in the K-Fold Cross Validation process. In order to keep the two classes ("fracture" and "noFracture") balanced and reduce the computational time, a subset of the whole dataset described in paragraph 4.2.1 was selected. In particular, the dataset used during the K-Fold Cross Validation process includes 359 slices labeled as "fracture" (belonging to 57 different patients) and 362 slices labeled as "noFracture" (belonging to 59 additional patients). It was possible to have 59 patients for the class "noFracture", by selecting only a subset of the "noFracture" slices from some patients. Therefore, these patients will become patients with "noFracture" in this phase.
During the K-Fold Cross Validation, the transfer learning technique was adopted to minimize the training burden for the CNN; at this aim, ResNet50 was used as pre-trained architecture. The K-Fold Cross Validation with K=5, described in Sec. 2.4, was used to choose the most suitable hyperparameters set and the batch size was set at 50. The relevant hyperparameters are learning rate, weight decay and dropout and they were chosen in the following ranges [0.000001; 0.005], [0.0001; 0.0005], [0.1; 0.5], respectively. Precisely, 20 different combinations (N=20 in Fig. 4.2) of the hyperparameters were tested. A random search for hyperparameters optimization was adopted with respect to a grid search. In the case of many hyperparameters, the random search is more effective than the grid search from the computational times' viewpoint and it is able to preserving good performance [180].
During the training process, the early stopping criteria can be adopted to allow the

network generalization and, at the same time, contain the computational costs. In this context, the following criterion was used as Early Stopping criterion: the training process ends if after 3 attempts the accuracy metric does not increase by at least 0.01. For each fold of the K-Fold Cross Validation process, the number of epochs was set to 6, while all the images of the dataset were normalized in line with the ImageNet format and rescaled from 512x512 to 224x224 pixels. A sample of Dicom images is illustrated in Fig. 4.3.

Fig. 4.3 Dicom images samples (8x8 inches) for classes “fracture” and “noFracture”.

After performing tests for the twenty different combinations of the hyperparameters, the set that allowed the model to reach the highest average accuracy (0.86) with

the smallest standard deviation, index of low variability (0.05) has been selected. The hyperparameters set mentioned above has the following characteristics: learning rate of 0.005, drop out of 0.5 and weight decay of 0.0005.

K-Fold Cross Validation Performance Remarks

The choice of using K-Fold Cross Validation for the model selection process has been done taking into account two main factors that characterize the prediction error: the bias error and the variance error [181]. In fact, the prediction error can be expressed as the contribution of three terms [182], as shown in Eq. 4.1:

$$\text{Prediction Error} = \text{Bias Error} + \text{Variance Error} + \text{Irreducible Error} \tag{4.1}$$

The irreducible error depends on the problem under study and can not be reduced, regardless of the algorithm chosen, since it may be produced for example by unknown variables during the mapping of input variables to the output variable. Conversely, the bias and variance errors can be potentially reduced by the choice of the algorithm. When a model uses simplified assumptions so that the target function is easier to learn, these assumptions represent the bias. Therefore, algorithms with a low bias make less simplifying assumptions on the target function, while algorithms with a high bias make more simplifying assumptions. Consequently, algorithms with a high bias are characterized by a lower predictive capability, especially with complex problems where the simplifying assumptions are not verified. On the other hand, variance specifies how much the estimate of the target function will change when different training datasets are used. In principle, the results of training should not change too much if the algorithm is trained on different training datasets of the same type. In this case, the model is able to generalize well, capturing the hidden underlying relationships between the inputs and the output variables. On the contrary, algorithms with a high variance are highly influenced by the specific training dataset it has been trained on and it is unable to generalize. Therefore, a model with high variance indicates significant changes to the target function estimate when the training dataset changes, while a model with low variance indicates small changes to the target function estimate when the training dataset changes. The ideal goal of a machine learning algorithm is to achieve low variance and low bias in order to minimize the prediction error. However, decreasing the bias will increase the variance and vice versa, leading to a bias-variance trade-off. The bias of a Cross

Validation procedure tends to decrease when the size of the training set (n_t) increases [181]. The choice of K for the K-Fold Cross Validation does not only determine the size of the training dataset, $n_t = n(K-1)/K$ where n is the number of total samples, and the number of the splits, influencing the computational cost, but also affects bias and variance. In fact, the bias of the K-Fold Cross Validation decrease with K, since $n_t = n(1-1/K)$ samples constitute the training set. On the other side, the variance of the K-Fold Cross Validation decrease with K for small values of K [181], while Leave-One-Out procedure (where $K = n$) is characterized by a high variance. Furthermore, it has been proved that the optimal choice for K is between 5 and 10 since the statistical performances do not improve so much for larger values of K [183]. Based on these considerations and on the computational complexity, which is roughly proportional to the number of the splits (K for the K-Fold Cross Validation), the K-Fold Cross Validation with $K = 5$ has been chosen for the model selection process.

Fine-Tuning of the Convolutional Neural Network

After having chosen the hyperparameters set, the network was fine-tuned. This process allows pre-trained networks to be able to recognize classes the architecture is not initially trained on. As described in Sec. 2.5, in a pre-trained network the convolutional neural network already contains learned discriminative filters. During the fine-tuning process, the final group of fully connected layers was replaced by a new set of fully connected layers characterized by random weights. Since the new set of fully connected layers acts entirely randomly, if the gradient propagates back from these random values to the entire network, the powerful capabilities of the pre-trained network risk being destroyed. In order to avoid this issue, the network was retrained as shown in Fig. 4.4.

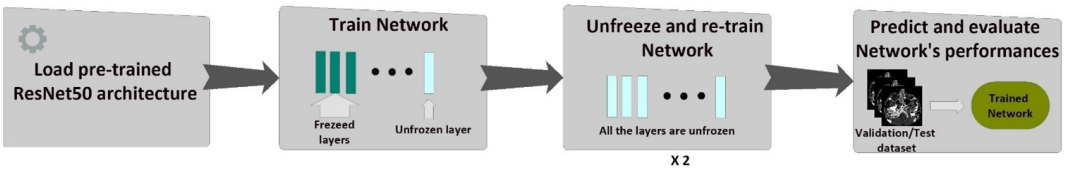

Fig. 4.4 ResNet50 was loaded as pre-trained network; after loading the network, the first step of the fine-tuning consists in freezing all the layers of the network apart from the fully connected layers, responsible for extracting high-level features on the medical dataset. After the fully connected layers have started to learn patterns from the medical dataset, all the architecture layers were unfreezed, allowing all the layers to be fine-tuned. To do so, two training processes, using differential learning rates, were performed.

The process consists of two main steps:

- Training only the last layer of the network (the fully connected set); all the other layers are freezed and the pre-trained model's weights, pre-trained on ImageNet dataset, are used for these layers;
- The previous step allows the last layer to start learning patterns of the medical dataset. So, at this point, all the weights are unfreezed (including the ones of the convolutional layers that have learned discriminative filters) and the entire model is trained with a small learning rate, in order to avoid overturning the convolutional filters radically.

During the fine-tuning process, the total dataset, described in Sec. 4.2.1, was used. Specifically, the training dataset consists of 8 023 slices belonging to the "fracture" class and 34 962 belonging to the "noFracture" class, for a total of 148 patients. Given that the two classes are no longer balanced, the CrossEntropyLoss was used as loss function. Different weights were assigned to the "fracture" and "noFracture" classes. The two weights, w_f for the positive class and w_{nf} for the negative class, are defined as follows:

$$[w_f, w_{nf}] = [\frac{|noFracture|}{|fracture|}, \frac{|noFracture|}{|fracture|}] = [\frac{34\,962}{8023}, \frac{34\,962}{34\,962}] = [4.36, 1.0] \quad (4.2)$$

The validation dataset was used for the evaluation of the network's performances and consists of 1 660 slices for the "fracture" class and 7 910 for the "noFracture" class, for a total of 30 patients.

During this second step, the model was re-trained twice by varying the learning rate to refine the model's performance. Before performing the re-train of the network, the model's weights that returned the highest performance in terms of accuracy were loaded. The value of the learning rate was chosen through the learning rate finder of the Fastai library [184]. Furthermore, the differential learning rates approach, implemented by the Fastai library, was adopted since some features are practically always the same (for example, the image's corners and edges learned in the initial layers of the neural network). By means of this method, different learning rates are assigned to the different network layers. Specifically, using the *slice* function within the *fit* method, learning rate values are assigned as follows: the minimum learning rate value of the selected range for the first layer; the maximum learning rate value of the selected range for the last layer and all learning rate values between the minimum and the maximum are distributed among all the other network's layers in between.

Results and Discussions

The results exposed in this section are attained on the validation dataset, described in the previous section, and on the test dataset, consisting of 1 577 slices belonging to the "fracture" class and 6 890 slices belonging to the "noFracture" class, for a total of 30 patients. The dataset partitioning into training, validation and test set was performed randomly at level-patient; this means that all the slices of one patient were assigned to one of the three datasets (training, validation or test). However, the validation and test datasets are substantially different from each other. First of all, the CT exams were performed on several devices, each one with different characteristics from the others; additionally, the splanchnocranium includes a very large and complex area, hence the fracture can interest any part of the region and, consequently, the CT scans can be completely different from each other. The results obtained from the validation and the test datasets are given in terms of confusion matrix in Fig. 4.5.

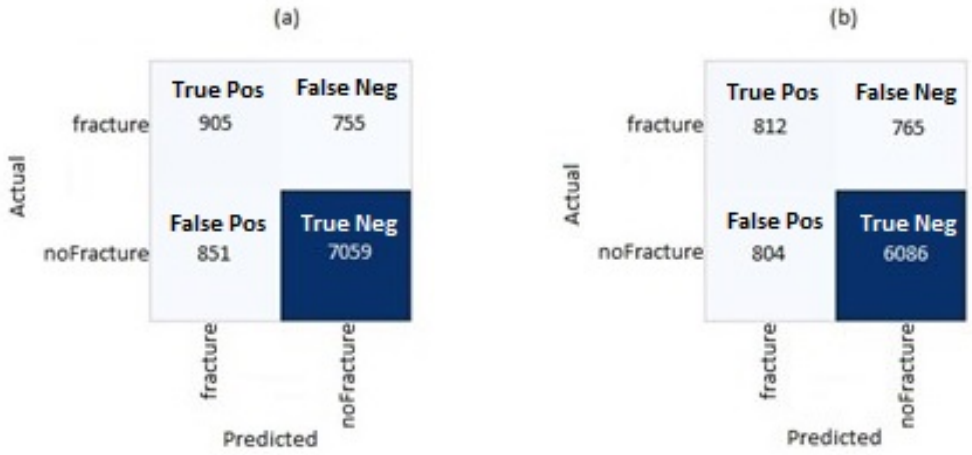

Fig. 4.5 Confusion Matrix for the validation (a) and test (b) datasets.

The results in terms of AUC-ROC for both validation and the test datasets are given in Fig. 4.6 and Fig. 4.7, respectively.

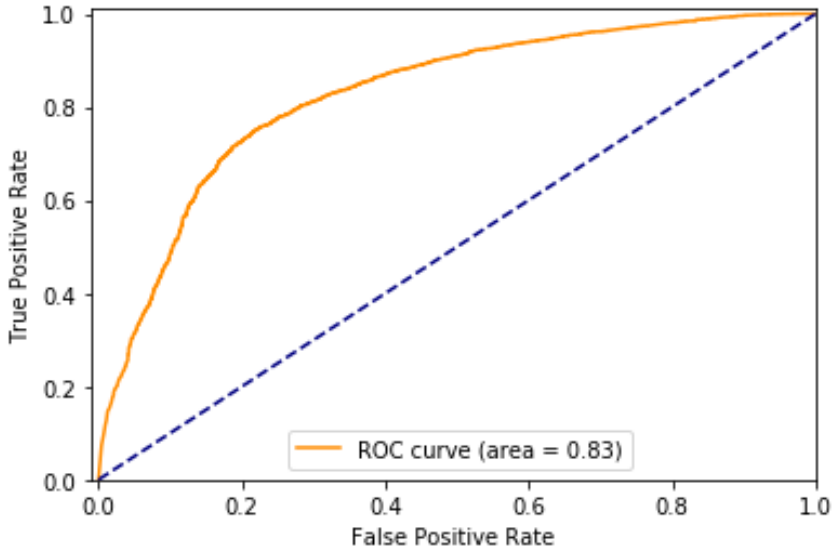

Fig. 4.6 Results in terms of AUC-ROC for the validation dataset.

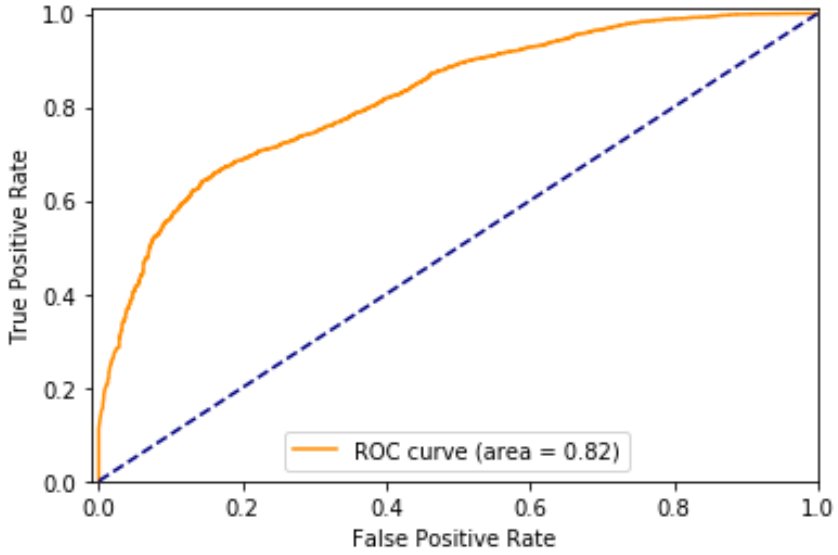

Fig. 4.7 Results in terms of AUC-ROC for the test dataset.

The AUC values for the validation and test datasets are 0.83 (0.82,0.84) and 0.82 (0.81,0.83), respectively. For the AUC values, the 95 % confidence intervals were calculated by means of the analytic method of Hanley and McNeil [185]. For the evaluation of the network's performance, the following metrics are considered:

$$Accuracy = \frac{TP+TN}{P+N} \tag{4.3}$$

$$Recall\ (or\ sensitivity) = \frac{TP}{TP+FN} \tag{4.4}$$

$$Precision\ (or\ positive\ predictive\ value) = \frac{TP}{TP+FP} \tag{4.5}$$

where TP are the number of the true positives, TN the number of the true negatives, P the number of the "positive" cases (it means the slice labeled as "fracture"), N the number of the "negative" cases (it means the slice labeled as "noFracture"), FN the number of false negatives and FP the number of false positives. In Tab. 4.1, the metrics values are shown for both validation and test datasets.

Table 4.1 Accuracy, Recall and Precision for the validation and test dataset with the Clopper-Pearson 95 % confidence intervals.

Metric	**Validation Dataset**	**Test Dataset**
Accuracy	0.83 (0.82, 0.84)	0.81 (0.81, 0.82)
Recall	0.55 (0.52, 0.57)	0.51 (0.49, 0.54)
Precision	0.52 (0.49, 0.54)	0.50 (0.48, 0.53)

The results achieved using the validation and test datasets are very similar and this is very promising for future CT scans, for which similar results in terms of accuracy will occur.

Since the final goal of the system consists in the prediction at patient-level, the performance evaluation is also performed in terms of patient's injury rather than slices. To this aim, all the slices were aggregated by patient with the following assumption: if two consecutive slices of the same patient are classified as slices with "fracture" by the neural network with a probability greater than 0.99, then the CNN has to classify the patient as a patient with fracture. The confusion matrix showing the results for the test dataset is presented in Fig. 4.8.

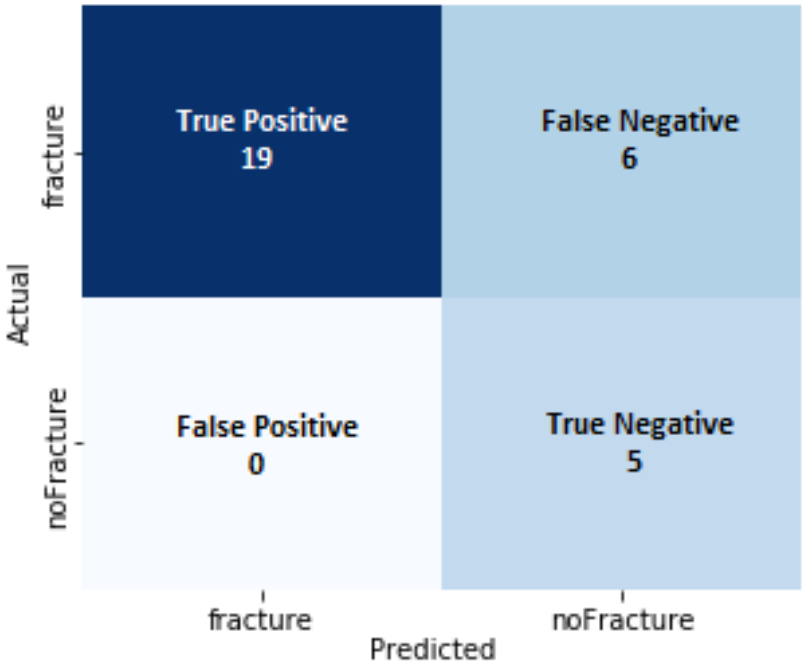

Fig. 4.8 Confusion Matrix in terms of patients for the test dataset.

Table 4.2 Accuracy, Recall and Precision for the test dataset with the Clopper-Pearson 95 % confidence intervals in terms of patients.

Metric	**Test Dataset**
Accuracy	0.80 (0.61, 0.92)
Recall	0.76 (0.55, 0.91)
Precision	1.00 (0.82, 1.00)

The measures of diagnostic accuracy in terms of patients (Accuracy, Recall and Precision) with 95 % confidence intervals for the test dataset are listed in Tab. 4.2.

The results in terms of patients show that the Neural Network ResNet50 can distinguish between fractured and not fractured bone in CT scans of injured patients with relatively high accuracy (80 %). This outcome is particularly encouraging, given the anatomical complexity together with the thinness of bones in the splanchnocranium, proving the effectiveness of transfer learning for medical images. Though a computer-aided decision system with an accuracy of 80 % can not replace the radiologist's decision, it may be very useful in supporting radiologists with immediate diagnosis and treatment. An automated fracture detection system based on the proposed model has the peculiarity of analyzing the CT scan's entire area with equal importance. Consequently, this facilitates the reduction of human errors in missed readings of the full area of the 3D image. Moreover, the proposed detection system may also be helpful to detect small fractures that are often scarcely visible on CT scans, requiring several checks by radiologists. Furthermore, the assessment of CT exams in trauma patients is essential to direct him towards highly specialized units if necessary, reducing medical error risks and unnecessary hospitalization. In fact, there are two main difficulties in the current clinical practice when a patient's trauma arises in an anatomically complex region such as the splanchnocranium. The first issue is the possible failure to identify the existence of a bone fracture; the second one is the erroneous classification of normal anatomical structures (for example sutures, vascular and nerve channels) as traumatic injuries. The diagnostic difficulties mentioned above often translate into higher costs for the healthcare system and a burden for the patient due to unneeded hospitalizations in specialized clinical units. For instance, if it is ascertained that there is no urgency for treatment in a craniofacial area trauma, the patient is transferred from the emergency room to the closest regional reference center that is specialized in maxillofacial trauma. The reassessment of the clinical case in the specialist field many times (about 20 % of

cases) points out not only the incongruity of hospitalization but often also the lack of indications for surgical treatment. In such cases, patients only require home medical therapy.

Conclusions

This research represents a proof of concept for using transfer learning starting from Convolutional Neural Network, pre-trained on non-medical images, and using it in a medical context for the detection of maxillofacial fracture in CT scans. Even though in the literature there are several AI applications in the orthopedic field, the possibility of using transfer learning applied to CT images to identify maxillofacial fractures of trauma patients has not yet been examined. The system is able to predict maxillofacial fractures in injured patients with 80 % of accuracy. The MFDS can become a valuable radiological diagnosis system in assisting radiologists with prompt diagnosis to guide therapeutic choices and treatments, thus minimizing medical error risks and preventing patient stress and harm by reducing maxillofacial trauma's diagnostic delays. Furthermore, the AI-based support tool used in non-specialized clinical wards may be useful to reduce the unnecessary hospitalization's socio-economic burden not only for the patient but also for society and the health system. Nevertheless, it is important to remark that the AI-based system should not replace the radiologist's work but should represent an aid for the medical staff in identifying facial fractures. It should be intended as a second opinion, not an independent one, becoming a precious assistive technology for the medical team leading to a reduction of medical error risks, unneeded patient transportation, hospitalization and socio-economic load for the public health governance [186].

Conclusions

In this thesis, the investigation performed on new deep learning models capable of providing valuable support in the industrial and healthcare context has been presented. The research activity carried out at CERN for the first case study has focused on the design, implementation and testing of new deep learning architectures for the prediction of the magnetic field values. The raw dataset was composed of ten sequences of field and excitation current waveforms. Each sequence consisted "of seven trapezoidal cycles about 5 seconds long, starting from a zero excitation current and reaching increasingly higher flat-top values, in order to scan the entire area of the major hysteresis loop. The network selection process was carried out on a dataset including 87 144 samples with a sample rate of 250 S/s", half of which was used for training and half for testing and statistical error evaluation.
By using an incremental approach, several deep neural networks have been chosen, fine-tuned and tested. The first architecture under test was the simplest one (MLP), a feed-forward network without feedback. For this type of architecture, the static structure hyperparametes (i.e. "the number of layers and the number of neurons for each layer") have been evaluated and selected. However, the results have shown an NRMSE of the order the percent that is not acceptable in this context (the required NRMSE is typically 0.01 %). Starting from the selected hyperparametes of the MLP, past observations of the input have been added to the network, which reduced to a deep TDNN. Nevertheless, the results have shown that the TDNN performs as the MLP architecture. Finally, using the selected hyperparametes of the MLP and the one of the TDNN (number of past input values), past observations of the output have been added to the network to reach the more complex structure of the NARX network. Due to the ability to remember past outputs, this network has been demonstrated to be able to represent the magnet's dynamics very accurately with an NRMSE of two orders of magnitude better, reaching an NRMSE of 0.01 %. All

the further models based on a hybrid approach, where the magnetic field has been reconstructed considering a first module, a linear regression, coupled with a neural network that modeled the nonlinear part of the signal, did not achieve the required level of performance. In conclusion, for the first case study the research activity has shown that, by using networks with feedback based on both input and output delay buffers, it is possible to catch the magnetic field dynamics and meet the required level of performance.

For the second case of study, a maxillofacial fracture detection system, based on convolutional neural networks and transfer learning, has been proposed to detect traumatic fractures. In particular, a convolutional neural network, pre-trained on non-medical images, has been re-trained and fine-tuned on CT images for the realization of a model to classify future CT exams as either "fracture" or "noFracture". The training set consisted of 148 CT scans, one for each patient, split as follows: 120 patients labeled with "fracture" and 28 patients labeled with "noFracture". The validation set was characterized by 30 CT scans (5 with "noFracture" and 25 with "fracture"), that have been used for statistical analysis. Finally, the test set, used for final performance evaluations, was characterized by additional 30 CT scans, with the same images partition of the validation set. The CT exams have been obtained from the internal database of the U.O.C. of Maxillofacial Surgery of the University Hospital "Federico II" and two radiologists consensually examined and classified each CT image, assigning them the proper "fracture"/"noFracture" labels. Tests have been performed both by considering the single slices separately and by grouping the slices for patients, in order to make a prediction at patient-level. A patient has been classified as fractured if two consecutive slices were assigned to the class "fracture" with a probability higher than 0.99. The patients' results have shown a model accuracy in classifying the maxillofacial fractures of 80 %. Therefore, even if the proposed system cannot replace the radiologists' team, it can offer valuable support in assisting radiologist's work, minimizing the risk of human error, diagnostic delays and incongruous hospitalization.

On the basis of these encouraging results, the two case studies show the feasibility of using deep learning models with spectacular success even in fields unexplored so far, such as the prediction of the magnetic field nonlinearities and the detection of fractures in the maxillofacial region.

References

[1] Maria Amodeo, Pasquale Arpaia, Marco Buzio, Vincenzo Di Capua, and Francesco Donnarumma. Hysteresis modeling in iron-dominated magnets based on a multi-layered narx neural network approach. *International Journal of Neural Systems*, page 2150033, 2021.

[2] Esma Mobs. The cern accelerator complex-august 2018. Technical report, 2018.

[3] Joseph Vella Wallbank, Maria Amodeo, Anthony Beaumont, Marco Buzio, Vincenzo Di Capua, Christian Grech, Nicholas Sammut, and David Giloteaux. Development of a real-time magnetic field measurement system for synchrotron control. *Electronics*, 10(17):2140, 2021.

[4] P Koopman. Bresenham line-drawing algorithm. *Forth Dimensions*, 8(6):12–16, 1987.

[5] Maria Amodeo, Vincenzo Abbate, Pasquale Arpaia, Renato Cuocolo, Giovanni Dell'Aversana Orabona, Monica Murero, Marco Parvis, Roberto Prevete, and Lorenzo Ugga. Transfer learning for an automated detection system of fractures in patients with maxillofacial trauma. *Applied Sciences*, 11(14):6293, 2021.

[6] Kristine Hamann and Rachel Smith. Facial recognition technology. *CRIM. JUST*, page 9, 2019.

[7] Umut Ozertem, Olivier Chapelle, Pinar Donmez, and Emre Velipasaoglu. Learning to suggest: a machine learning framework for ranking query suggestions. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, pages 25–34, 2012.

[8] Reinhold Haeb-Umbach, Shinji Watanabe, Tomohiro Nakatani, Michiel Bacchiani, Bjorn Hoffmeister, Michael L Seltzer, Heiga Zen, and Mehrez Souden. Speech processing for digital home assistants: Combining signal processing with deep-learning techniques. *IEEE Signal processing magazine*, 36(6):111–124, 2019.

[9] Stephanie Dick. Artificial intelligence. 2019.

[10] Xin Yao and Yong Liu. Machine learning. In *Search Methodologies*, pages 477–517. Springer, 2014.

[11] Sun-Chong Wang. Artificial neural network. In *Interdisciplinary computing in java programming*, pages 81–100. Springer, 2003.

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

[13] Daniel S Berman, Anna L Buczak, Jeffrey S Chavis, and Cherita L Corbett. A survey of deep learning methods for cyber security. *Information*, 10(4):122, 2019.

[14] Pradheepan Raghavan and Neamat El Gayar. Fraud detection using machine learning and deep learning. In *2019 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE)*, pages 334–339. IEEE, 2019.

[15] Abhimanyu Roy, Jingyi Sun, Robert Mahoney, Loreto Alonzi, Stephen Adams, and Peter Beling. Deep learning detecting fraud in credit card transactions. In *2018 Systems and Information Engineering Design Symposium (SIEDS)*, pages 129–134. IEEE, 2018.

[16] Richard Csaky. Deep learning based chatbot models. *arXiv preprint arXiv:1908.08835*, 2019.

[17] Mohammad Nuruzzaman and Omar Khadeer Hussain. A survey on chatbot implementation in customer service industry through deep neural networks. In *2018 IEEE 15th International Conference on e-Business Engineering (ICEBE)*, pages 54–61. IEEE, 2018.

[18] Sonali Chandel, Yuan Yuying, Gu Yujie, Abdul Razaque, and Geng Yang. Chatbot: efficient and utility-based platform. In *Science and Information Conference*, pages 109–122. Springer, 2018.

[19] Atieh Poushneh. Humanizing voice assistant: The impact of voice assistant personality on consumers' attitudes and behaviors. *Journal of Retailing and Consumer Services*, 58:102283, 2021.

[20] Noura Abdi, Kopo M Ramokapane, and Jose M Such. More than smart speakers: security and privacy perceptions of smart home personal assistants. In *Fifteenth Symposium on Usable Privacy and Security ({SOUPS} 2019)*, pages 451–466, 2019.

[21] Lara Martin, Prithviraj Ammanabrolu, Xinyu Wang, William Hancock, Shruti Singh, Brent Harrison, and Mark Riedl. Event representations for automated story generation with deep neural nets. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[22] Chaouki Boufenar, Adlen Kerboua, and Mohamed Batouche. Investigation on deep learning for off-line handwritten arabic character recognition. *Cognitive Systems Research*, 50:180–195, 2018.

[23] Qing Rao and Jelena Frtunikj. Deep learning for self-driving cars: Chances and challenges. In *Proceedings of the 1st International Workshop on Software Engineering for AI in Autonomous Systems*, pages 35–38, 2018.

[24] Quan Le, Luis Miralles-Pechuán, Shridhar Kulkarni, Jing Su, and Oisín Boydell. An overview of deep learning in industry. *Data Analytics and AI*, pages 65–98, 2020.

[25] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, and Eftychios Protopapadakis. Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*, 2018, 2018.

[26] Kailas Vodrahalli and Achintya K Bhowmik. 3d computer vision based on machine learning with deep neural networks: A review. *Journal of the Society for Information Display*, 25(11):676–694, 2017.

[27] J Alison Noble. From inspection to process understanding and monitoring: a view on computer vision in manufacturing. *Image and vision computing*, 13(3):197–214, 1995.

[28] Kesheng Wang and Yi Wang. How ai affects the future predictive maintenance: a primer of deep learning. In *International Workshop of Advanced Manufacturing and Automation*, pages 1–9. Springer, 2017.

[29] Björn Kroll, David Schaffranek, Sebastian Schriegel, and Oliver Niggemann. System modeling based on machine learning for anomaly detection and predictive maintenance in industrial plants. In *Proceedings of the 2014 IEEE emerging technology and factory automation (ETFA)*, pages 1–7. IEEE, 2014.

[30] Abdullahi Chowdhury, Gour Karmakar, Joarder Kamruzzaman, and Syed Islam. Trustworthiness of self-driving vehicles for intelligent transportation systems in industry applications. *IEEE Transactions on Industrial Informatics*, 17(2):961–970, 2020.

[31] Elena Quatrini, Francesco Costantino, Giulio Di Gravio, and Riccardo Patriarca. Machine learning for anomaly detection and process phase classification to improve safety and maintenance activities. *Journal of Manufacturing Systems*, 56:117–132, 2020.

[32] Justin Hawks. *Intelligent UPC-A Barcode Scanning Using Machine Learning*. PhD thesis, University of Pittsburgh, 2018.

[33] K Shailaja, B Seetharamulu, and MA Jabbar. Machine learning in healthcare: A review. In *2018 Second international conference on electronics, communication and aerospace technology (ICECA)*, pages 910–914. IEEE, 2018.

[34] Lu Zhang, Jianjun Tan, Dan Han, and Hao Zhu. From machine learning to deep learning: progress in machine intelligence for rational drug discovery. *Drug discovery today*, 22(11):1680–1685, 2017.

[35] Hideyuki Shimizu and Keiichi I Nakayama. Artificial intelligence in oncology. *Cancer science*, 111(5):1452, 2020.

[36] Renato Cuocolo, Martina Caruso, Teresa Perillo, Lorenzo Ugga, and Mario Petretta. Machine learning in oncology: a clinical appraisal. *Cancer letters*, 481:55–62, 2020.

[37] Ayelet Akselrod-Ballin, Michal Chorev, Yoel Shoshan, Adam Spiro, Alon Hazan, Roie Melamed, Ella Barkan, Esma Herzel, Shaked Naor, Ehud Karavani, et al. Predicting breast cancer by applying deep learning to linked health records and mammograms. *Radiology*, 292(2):331–342, 2019.

[38] Sara Hosseinzadeh Kassani and Peyman Hosseinzadeh Kassani. A comparative study of deep learning architectures on melanoma detection. *Tissue and Cell*, 58:76–83, 2019.

[39] Filippo Arcadu, Fethallah Benmansour, Andreas Maunz, Jeff Willis, Zdenka Haskova, and Marco Prunotto. Deep learning algorithm predicts diabetic retinopathy progression in individual patients. *NPJ digital medicine*, 2(1):1–9, 2019.

[40] Klaus Donsa, Stephan Spat, Peter Beck, Thomas R Pieber, and Andreas Holzinger. Towards personalization of diabetes therapy using computerized decision support and machine learning: some open problems and challenges. In *Smart Health*, pages 237–260. Springer, 2015.

[41] Teresa Infante, Marco Francone, Maria L De Rimini, Carlo Cavaliere, Raffaele Canonico, Carlo Catalano, and Claudio Napoli. Machine learning and network medicine: a novel approach for precision medicine and personalized therapy in cardiomyopathies. *Journal of Cardiovascular Medicine*, 22(6):429–440, 2021.

[42] Thiraphat Tanphiriyakun, Sattaya Rojanasthien, and Piyapong Khumrin. Bone mineral density response prediction following osteoporosis treatment using machine learning to aid personalized therapy. *Scientific Reports*, 11(1):1–16, 2021.

[43] Darren Plant and Anne Barton. Machine learning in precision medicine: lessons to learn. *Nature Reviews Rheumatology*, 17(1):5–6, 2021.

[44] De Arriba-Pérez, Manuel Caeiro-Rodríguez, Juan M Santos-Gago, et al. Collection and processing of data from wrist wearable devices in heterogeneous and multiple-user scenarios. *Sensors*, 16(9):1538, 2016.

[45] Charlotte Kerner and Victoria A Goodyear. The motivational impact of wearable healthy lifestyle technologies: a self-determination perspective on fitbits with adolescents. *American Journal of Health Education*, 2017.

[46] Steven Alexander Hicks, Sigrun Eskeland, Mathias Lux, Thomas de Lange, Kristin Ranheim Randel, Mattis Jeppsson, Konstantin Pogorelov, Pål Halvorsen, and Michael Riegler. Mimir: an automatic reporting and reasoning system for deep learning based analysis in the medical domain. In *Proceedings of the 9th ACM Multimedia Systems Conference*, pages 369–374, 2018.

[47] Natalie Stephenson, Emily Shane, Jessica Chase, Jason Rowland, David Ries, Nicola Justice, Jie Zhang, Leong Chan, and Renzhi Cao. Survey of machine learning techniques in drug discovery. *Current drug metabolism*, 20(3):185–193, 2019.

[48] Kristina Preuer, Günter Klambauer, Friedrich Rippmann, Sepp Hochreiter, and Thomas Unterthiner. Interpretable deep learning in drug discovery. In *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*, pages 331–345. Springer, 2019.

[49] Adam Zielinski. Ai and the future of pharmaceutical research. *arXiv preprint arXiv:2107.03896*, 2021.

[50] Jeanne-Françoise Williamson. Clinical technology. *Applied Clinical Trials*, 26(6):19, 2017.

[51] Stephen E Kurtz, Christopher A Eide, Andy Kaempf, Vishesh Khanna, Samantha L Savage, Angela Rofelty, Isabel English, Hibery Ho, Ravi Pandya, William J Bolosky, et al. Molecularly targeted drug combinations demonstrate selective effectiveness for myeloid-and lymphoid-derived hematologic malignancies. *Proceedings of the National Academy of Sciences*, 114(36):E7554–E7563, 2017.

[52] Sid Kiblawi, Deborah Chasman, Amanda Henning, Eunju Park, Hoifung Poon, Michael Gould, Paul Ahlquist, and Mark Craven. Augmenting subnetwork inference with information extracted from the scientific literature. *PLoS computational biology*, 15(6):e1006758, 2019.

[53] Felipe Feijoo, Michele Palopoli, Jen Bernstein, Sauleh Siddiqui, and Tenley E Albright. Key indicators of phase transition for clinical trials through machine learning. *Drug discovery today*, 25(2):414–421, 2020.

[54] Steve Rodriguez, Clemens Hug, Petar Todorov, Nienke Moret, Sarah A Boswell, Kyle Evans, George Zhou, Nathan T Johnson, Bradley T Hyman, Peter K Sorger, et al. Machine learning identifies candidates for drug repurposing in alzheimer's disease. *Nature communications*, 12(1):1–13, 2021.

[55] Ka-Chun Un, Chun-Ka Wong, Yuk-Ming Lau, Jeffrey Chun-Yin Lee, Frankie Chor-Cheung Tam, Wing-Hon Lai, Yee-Man Lau, Hao Chen, Sandi Wibowo, Xiaozhu Zhang, et al. Observational study on wearable biosensors and machine learning-based remote monitoring of covid-19 patients. *Scientific reports*, 11(1):1–9, 2021.

[56] Prableen Kaur, Manik Sharma, and Mamta Mittal. Big data and machine learning based secure healthcare framework. *Procedia computer science*, 132:1049–1059, 2018.

[57] Zhuoran Wang, Anoop D Shah, A Rosemary Tate, Spiros Denaxas, John Shawe-Taylor, and Harry Hemingway. Extracting diagnoses and investigation results from unstructured text in electronic health records by semi-supervised machine learning. *PLoS One*, 7(1):e30412, 2012.

[58] Tao Zheng, Wei Xie, Liling Xu, Xiaoying He, Ya Zhang, Mingrong You, Gong Yang, and You Chen. A machine learning-based framework to identify type 2 diabetes through electronic health records. *International journal of medical informatics*, 97:120–127, 2017.

[59] Philippe Meyer, Vincent Noblet, Christophe Mazzara, and Alex Lallement. Survey on deep learning for radiotherapy. *Computers in biology and medicine*, 98:126–146, 2018.

[60] Chris McIntosh, Leigh Conroy, Michael C Tjong, Tim Craig, Andrew Bayley, Charles Catton, Mary Gospodarowicz, Joelle Helou, Naghmeh Isfahanian, Vickie Kong, et al. Clinical integration of machine learning for curative-intent radiation treatment of patients with prostate cancer. *Nature Medicine*, 27(6):999–1005, 2021.

[61] Lijing Wang, Jiangzhuo Chen, and Madhav Marathe. Defsi: Deep learning based epidemic forecasting with synthetic information. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 9607–9612, 2019.

[62] Narinder Singh Punn, Sanjay Kumar Sonbhadra, and Sonali Agarwal. Covid-19 epidemic analysis using machine learning and deep learning algorithms. *MedRxiv*, 2020.

[63] Godson Kalipe, Vikas Gautham, and Rajat Kumar Behera. Predicting malarial outbreak using machine learning and deep learning approach: a review and analysis. In *2018 International Conference on Information Technology (ICIT)*, pages 33–38. IEEE, 2018.

[64] Sebastian Spänig, Agnes Emberger-Klein, Jan-Peter Sowa, Ali Canbay, Klaus Menrad, and Dominik Heider. The virtual doctor: An interactive clinical-decision-support system based on deep learning for non-invasive prediction of diabetes. *Artificial intelligence in medicine*, 100:101706, 2019.

[65] Ahmed Fadhil. Beyond patient monitoring: Conversational agents role in telemedicine & healthcare support for home-living elderly individuals. *arXiv preprint arXiv:1803.06000*, 2018.

[66] Daniel Flores-Martin, Javier Rojo, Enrique Moguel, Javier Berrocal, and Juan M Murillo. Smart nursing homes: Self-management architecture based

on iot and machine learning for rural areas. *Wireless Communications and Mobile Computing*, 2021, 2021.

[67] Theresa Schachner, Roman Keller, and Florian Von Wangenheim. Artificial intelligence-based conversational agents for chronic conditions: systematic literature review. *Journal of medical Internet research*, 22(9):e20701, 2020.

[68] Ishank Agarwal. Virtual healthcare assistant. *Available at SSRN 3833836*, 2021.

[69] Nitin Gupta, Shashank Mujumdar, Hima Patel, Satoshi Masuda, Naveen Panwar, Sambaran Bandyopadhyay, Sameep Mehta, Shanmukha Guttula, Shazia Afzal, Ruhi Sharma Mittal, et al. Data quality for machine learning tasks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 4040–4041, 2021.

[70] Hussein Ibrahim, Xiaoxuan Liu, Nevine Zariffa, Andrew D Morris, and Alastair K Denniston. Health data poverty: an assailable barrier to equitable digital health care. *The Lancet Digital Health*, 2021.

[71] Georgios A Kaissis, Marcus R Makowski, Daniel Rückert, and Rickmer F Braren. Secure, privacy-preserving and federated machine learning in medical imaging. *Nature Machine Intelligence*, 2(6):305–311, 2020.

[72] Alvin Rajkomar, Jeffrey Dean, and Isaac Kohane. Machine learning in medicine. *New England Journal of Medicine*, 380(14):1347–1358, 2019.

[73] Milena A Gianfrancesco, Suzanne Tamang, Jinoos Yazdany, and Gabriela Schmajuk. Potential biases in machine learning algorithms using electronic health record data. *JAMA internal medicine*, 178(11):1544–1547, 2018.

[74] Alvin Rajkomar, Michaela Hardt, Michael D Howell, Greg Corrado, and Marshall H Chin. Ensuring fairness in machine learning to advance health equity. *Annals of internal medicine*, 169(12):866–872, 2018.

[75] Shuo Tian, Wenbo Yang, Jehane Michael Le Grange, Peng Wang, Wei Huang, and Zhewei Ye. Smart healthcare: making medical care more intelligent. *Global Health Journal*, 3(3):62–65, 2019.

[76] Carlos Affonso, André Luis Debiaso Rossi, Fábio Henrique Antunes Vieira, André Carlos Ponce de Leon Ferreira, et al. Deep learning for biological image classification. *Expert Systems with Applications*, 85:114–122, 2017.

[77] Zhongheng Zhang. Introduction to machine learning: k-nearest neighbors. *Annals of translational medicine*, 4(11), 2016.

[78] Shan Suthaharan. Support vector machine. In *Machine learning models and algorithms for big data classification*, pages 207–235. Springer, 2016.

[79] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[80] Weihong Wang, Jie Yang, Jianwei Xiao, Sheng Li, and Dixin Zhou. Face recognition based on deep learning. In *International Conference on Human Centered Computing*, pages 812–820. Springer, 2014.

[81] Jianlong Fu and Yong Rui. Advances in deep learning approaches for image tagging. *APSIPA Transactions on Signal and Information Processing*, 6, 2017.

[82] E Sreehari and Satyajee Srivastava. Prediction of climate variable using multiple linear regression. In *2018 4th International Conference on Computing Communication and Automation (ICCCA)*, pages 1–4. IEEE, 2018.

[83] T Gopalakrishnan, Ritesh Choudhary, and Sarada Prasad. Prediction of sales value in online shopping using linear regression. In *2018 4th International Conference on Computing Communication and Automation (ICCCA)*, pages 1–6. IEEE, 2018.

[84] Xueheng Qiu, Le Zhang, Ye Ren, Ponnuthurai N Suganthan, and Gehan Amaratunga. Ensemble deep learning for regression and time series forecasting. In *2014 IEEE symposium on computational intelligence in ensemble learning (CIEL)*, pages 1–6. IEEE, 2014.

[85] Yannick Suter, Alain Jungo, Michael Rebsamen, Urspeter Knecht, Evelyn Herrmann, Roland Wiest, and Mauricio Reyes. Deep learning versus classical regression for brain tumor patient survival prediction. In *International MICCAI Brainlesion Workshop*, pages 429–440. Springer, 2018.

[86] Pádraig Cunningham, Matthieu Cord, and Sarah Jane Delany. Supervised learning. In *Machine learning techniques for multimedia*, pages 21–49. Springer, 2008.

[87] Happiness Ugochi Dike, Yimin Zhou, Kranthi Kumar Deveerasetty, and Qingtian Wu. Unsupervised learning based on artificial neural network: A review. In *2018 IEEE International Conference on Cyborg and Bionic Systems (CBS)*, pages 322–327. IEEE, 2018.

[88] Dong-Hyun Lee et al. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *Workshop on challenges in representation learning, ICML*, volume 3, page 896, 2013.

[89] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016.

[90] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[91] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[92] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pages 437–478. Springer, 2012.

[93] Vasily E Tarasov. On chain rule for fractional derivatives. *Communications in Nonlinear Science and Numerical Simulation*, 30(1-3):1–4, 2016.

[94] Nadia Jmour, Sehla Zayen, and Afef Abdelkrim. Convolutional neural networks for image classification. In *2018 International Conference on Advanced Systems and Electric Technologies (IC_ASET)*, pages 397–402. IEEE, 2018.

[95] Adnan Qayyum, Syed Muhammad Anwar, Muhammad Awais, and Muhammad Majid. Medical image retrieval using deep convolutional neural network. *Neurocomputing*, 266:8–20, 2017.

[96] Hokuto Kagaya, Kiyoharu Aizawa, and Makoto Ogawa. Food detection and recognition using convolutional neural network. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 1085–1088, 2014.

[97] Seunghoon Hong, Junhyuk Oh, Honglak Lee, and Bohyung Han. Learning transferrable knowledge for semantic segmentation with deep convolutional neural network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3204–3212, 2016.

[98] Saloni Kalra and Alka Leekha. Survey of convolutional neural networks for image captioning. *Journal of Information and Optimization Sciences*, 41(1):239–260, 2020.

[99] Jeff Hwang and You Zhou. Image colorization with deep convolutional neural networks. In *Stanford University, Tech. Rep.* 2016.

[100] Lei Zhang, Fan Yang, Yimin Daniel Zhang, and Ying Julie Zhu. Road crack detection using deep convolutional neural network. In *2016 IEEE international conference on image processing (ICIP)*, pages 3708–3712. IEEE, 2016.

[101] Rui Hou, Chen Chen, and Mubarak Shah. Tube convolutional neural network (t-cnn) for action detection in videos. In *Proceedings of the IEEE international conference on computer vision*, pages 5822–5831, 2017.

[102] Meng Wang, Lingjing Wang, and Yi Fang. 3densinet: A robust neural network architecture towards 3d volumetric object prediction from 2d image. In *Proceedings of the 25th ACM international conference on Multimedia*, pages 961–969, 2017.

[103] Junyuan Xie, Ross Girshick, and Ali Farhadi. Deep3d: Fully automatic 2d-to-3d video conversion with deep convolutional neural networks. In *European conference on computer vision*, pages 842–857. Springer, 2016.

[104] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Icml*, 2010.

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

[106] Gian Carlo Cardarilli, Luca Di Nunzio, Rocco Fazzolari, Daniele Giardino, Alberto Nannarelli, Marco Re, and Sergio Spanò. A pseudo-softmax function for hardware-based high speed image classification. *Scientific Reports*, 11(1):1–10, 2021.

[107] Vijayaditya Peddinti, Daniel Povey, and Sanjeev Khudanpur. A time delay neural network architecture for efficient modeling of long temporal contexts. In *Sixteenth annual conference of the international speech communication association*, 2015.

[108] Alex Waibel, Toshiyuki Hanazawa, Geoffrey Hinton, Kiyohiro Shikano, and Kevin J Lang. Phoneme recognition using time-delay neural networks. *IEEE transactions on acoustics, speech, and signal processing*, 37(3):328–339, 1989.

[109] Hang Xie, Hao Tang, and Yu-He Liao. Time series prediction based on narx neural networks: An advanced approach. In *2009 International conference on machine learning and cybernetics*, volume 3, pages 1275–1279. IEEE, 2009.

[110] Larry R Medsker and LC Jain. Recurrent neural networks. *Design and Applications*, 5:64–67, 2001.

[111] Hava T Siegelmann, Bill G Horne, and C Lee Giles. Computational capabilities of recurrent narx neural networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 27(2):208–215, 1997.

[112] Sebastian Raschka. Model evaluation, model selection, and algorithm selection in machine learning. *arXiv preprint arXiv:1811.12808*, 2018.

[113] Anders Krogh and John Hertz. A simple weight decay can improve generalization. *Advances in neural information processing systems*, 4, 1991.

[114] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[115] Douglas M Hawkins. The problem of overfitting. *Journal of chemical information and computer sciences*, 44(1):1–12, 2004.

[116] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[117] Lisa Torrey and Jude Shavlik. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pages 242–264. IGI global, 2010.

[118] Zifeng Wu, Chunhua Shen, and Anton Van Den Hengel. Wider or deeper: Revisiting the resnet model for visual recognition. *Pattern Recognition*, 90:119–133, 2019.

[119] Wajahat Nawaz, Sagheer Ahmed, Ali Tahir, and Hassan Aqeel Khan. Classification of breast cancer histology images using alexnet. In *International conference image analysis and recognition*, pages 869–876. Springer, 2018.

[120] Jin Li, Peng Wang, Yanzhao Li, Yang Zhou, Xiaolong Liu, and Kuan Luan. Transfer learning of pre-trained inception-v3 model for colorectal cancer lymph node metastasis classification. In *2018 IEEE International Conference on Mechatronics and Automation (ICMA)*, pages 1650–1654. IEEE, 2018.

[121] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[122] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[123] Yifan Qiao, Chenyan Xiong, Zhenghao Liu, and Zhiyuan Liu. Understanding the behaviors of bert in ranking. *arXiv preprint arXiv:1904.07531*, 2019.

[124] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[125] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.

[126] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[127] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[128] Augusto Visintin. Mathematical models of hysteresis. In *Modelling and Optimization of Distributed Parameter Systems Applications to Engineering*, pages 71–80. Springer, 1996.

[129] Vahid Hassani, Tegoeh Tjahjowidodo, and Thanh Nho Do. A survey on hysteresis modeling, identification and control. *Mechanical systems and signal processing*, 49(1-2):209–233, 2014.

[130] Isaak D Mayergoyz. *Mathematical models of hysteresis and their applications*. Academic Press, 2003.

[131] Edward Della Torre. Hysteresis modeling. *COMPEL-The international journal for computation and mathematics in electrical and electronic engineering*, 1998.

[132] J Takács. A phenomenological mathematical model of hysteresis. *COMPEL-The international journal for computation and mathematics in electrical and electronic engineering*, 2001.

[133] Monia Ferjani Jaafar. Magnetic hysteresis modeling and numerical simulation for ferromagnetic materials. In *2013 International Conference on Control, Decision and Information Technologies (CoDIT)*, pages 516–523. IEEE, 2013.

[134] David C Jiles and David L Atherton. Theory of ferromagnetic hysteresis. *Journal of applied physics*, 55(6):2115–2120, 1984.

[135] Claudio Serpico and Ciro Visone. Magnetic hysteresis modeling via feed-forward neural networks. *IEEE Transactions on Magnetics*, 34(3):623–628, 1998.

[136] Dimitre Makaveev, Luc Dupré, Marc De Wulf, and Jan Melkebeek. Modeling of quasistatic magnetic hysteresis with feed-forward neural networks. *Journal of Applied Physics*, 89(11):6737–6739, 2001.

[137] A Du, J Pan, and A Guzzomi. Modeling the hysteresis characteristics of transformer core under various excitation level via on-line measurements. *Electronics*, 7(390), 2018.

[138] Eike Feldmeier, Th Haberer, Andreas Peters, C Schoemers, Rudolf Steiner, et al. Magnetic field correction in normal conducting synchrotrons. *Proc. IPAC*, 2010.

[139] M. Di Castro, D. Sernelius, L. Bottura, L. Deniau, N. Sammut, S. Sanfilippo, and W. Venturini Delsolaro. Parametric field modeling for the lhc main magnets in operating conditions. In *2007 IEEE Particle Accelerator Conference (PAC)*, pages 1586–1588, 2007.

[140] N Sammut, L Bottura, and J Micallef. Mathematical formulation to predict the harmonics of the superconducting large hadron collider magnets. *Phys. Rev. Accel. Beams*, 9(1):012402, 2006.

[141] P Arpaia, M Buzio, F Caspers, G Golluccio, and C Petrone. Static metrological characterization of a ferrimagnetic resonance transducer for real-time magnetic field markers in particle accelerators. In *2011 IEEE International*

Instrumentation and Measurement Technology Conference, pages 1–4. IEEE, 2011.

[142] Ferenc Preisach. Über die magnetische nachwirkung. *Zeitschrift für physik*, 94(5-6):277–302, 1935.

[143] Valentin Pricop. *EFECTELE HISTEREZISULUI DIN MATERIALELE FOLOSITE PENTRU CIRCUITELE MAGNETICE ALE ACCELERATOARELOR DE PARTICULE, HYSTERESIS EFFECTS IN THE CORES OF PARTICLE ACCELERATOR MAGNETS*. PhD thesis, Școala Doctoral Interdisciplinară, Departament: Inginerie Electrică și Fizică Aplicată, Transilvania U., Brașov, Romania, 2016.

[144] David C Jiles and David L Atherton. Theory of ferromagnetic hysteresis. *Journal of magnetism and magnetic materials*, 61(1-2):48–60, 1986.

[145] Christian Grech, Maria Amodeo, Anthony Beaumont, Marco Buzio, Vincenzo Di Capua, David Giloteaux, Nicholas Sammut, and Joseph Vella Wallbank. Error characterization and calibration of real-time magnetic field measurement systems. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, page 164979, 2020.

[146] Pedro Lara Benítez, Manuel Carranza García, and José C. Riquelme. An experimental review on deep learning architectures for time series forecasting. *International Journal of Neural Systems*, 31(03):2130001, 2021.

[147] Spyridon Plakias and Yiannis S. Boutalis. Lyapunov theory-based fusion neural networks for the identification of dynamic nonlinear systems. *International Journal of Neural Systems*, 0(0):17, 2019.

[148] Tsungnan Lin, Bill G Horne, Peter Tino, and C Lee Giles. Learning long-term dependencies in narx recurrent neural networks. *IEEE Transactions on Neural Networks*, 7(6):1329–1338, 1996.

[149] Christian Grech, Marco Buzio, Mariano Pentella, and Nicholas Sammut. Dynamic ferromagnetic hysteresis modelling using a preisach-recurrent neural network model. *Materials*, 13(11):2561, 2020.

[150] Silvano Cincotti, Michele Marchesi, and Antonino Serri. A neural network model of parametric nonlinear hysteretic inductors. *IEEE transactions on magnetics*, 34(5):3040–3043, 1998.

[151] AA Adly and SK Abd-El-Hafiz. Using neural networks in the identification of preisach-type hysteresis models. *IEEE Transactions on Magnetics*, 34(3):629–635, 1998.

[152] Maurizio Cirrincione, Rosario Miceli, Giuseppe Ricco Galluzzo, and Marco Trapanese. Preisach function identification by neural networks. *IEEE Transactions on Magnetics*, 38(5):2421–2423, 2002.

[153] Alessandro Salvini, F Riganti Fulginei, and Christian Coltelli. A neuro-genetic and time-frequency approach to macromodeling dynamic hysteresis in the harmonic regime. *IEEE Transactions on Magnetics*, 39(3):1401–1404, 2003.

[154] Alessandro Salvini, Francesco Riganti Fulginei, and Giuseppe Pucacco. Generalization of the static preisach model for dynamic hysteresis by a genetic approach. *IEEE transactions on Magnetics*, 39(3):1353–1356, 2003.

[155] Francesco Riganti Fulginei and Alessandro Salvini. Neural network approach for modelling hysteretic magnetic materials under distorted excitations. *IEEE Transactions on Magnetics*, 48(2):307–310, 2012.

[156] Design time series narx feedback neural networks. https://it.mathworks.com/help/deeplearning/ug/design-time-series-narx-feedback-neural-networks.html;jsessionid=d6b668bf9c76b4023681339d944d. Accessed: 22-07-2021.

[157] Simone Gilardoni, Django Manglunki, Jean-Paul Burnet, Christian Carli, Michel Chanel, Roland Garoby, Massimo Giovannozzi, Steven Hancock, Helmut Haseroth, Kurt Hübner, et al. Fifty years of the cern proton synchrotron: Volume 2. *arXiv preprint arXiv:1309.6923*, 2013.

[158] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[159] Sam Roweis. Levenberg-marquardt optimization. *Notes, University Of Toronto*, 1996.

[160] Jie Ding, Vahid Tarokh, and Yuhong Yang. Model selection techniques: An overview. *IEEE Signal Processing Magazine*, 35(6):16–34, 2018.

[161] Pishtiwan HS Kalmet, Sebastian Sanduleanu, Sergey Primakov, Guangyao Wu, Arthur Jochems, Turkey Refaee, Abdalla Ibrahim, Luca v Hulst, Philippe Lambin, and Martijn Poeze. Deep learning in fracture detection: a narrative review. *Acta orthopaedica*, 91(2):215–220, 2020.

[162] Andre Esteva, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *nature*, 542(7639):115–118, 2017.

[163] Varun Gulshan, Lily Peng, Marc Coram, Martin C Stumpe, Derek Wu, Arunachalam Narayanaswamy, Subhashini Venugopalan, Kasumi Widner, Tom Madams, Jorge Cuadros, et al. Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs. *Jama*, 316(22):2402–2410, 2016.

[164] June-Goo Lee, Sanghoon Jun, Young-Won Cho, Hyunna Lee, Guk Bae Kim, Joon Beom Seo, and Namkug Kim. Deep learning in medical imaging: general overview. *Korean journal of radiology*, 18(4):570–584, 2017.

[165] Jakub Olczak, Niklas Fahlberg, Atsuto Maki, Ali Sharif Razavian, Anthony Jilert, André Stark, Olof Sköldenberg, and Max Gordon. Artificial intelligence for analyzing orthopedic trauma radiographs: deep learning algorithms—are they on par with humans for diagnosing fractures? *Acta orthopaedica*, 88(6):581–586, 2017.

[166] An Tang, Roger Tam, Alexandre Cadrin-Chênevert, Will Guest, Jaron Chong, Joseph Barfett, Leonid Chepelev, Robyn Cairns, J Ross Mitchell, Mark D Cicero, et al. Canadian association of radiologists white paper on artificial intelligence in radiology. *Canadian Association of Radiologists Journal*, 69(2):120–135, 2018.

[167] Min-Suk Heo, Jo-Eun Kim, Jae-Joon Hwang, Sang-Sun Han, Jin-Soo Kim, Won-Jin Yi, and In-Woo Park. Artificial intelligence in oral and maxillofacial radiology: what is currently possible? *Dentomaxillofacial Radiology*, 50(3):20200375, 2021.

[168] Kuofeng Hung, Carla Montalvao, Ray Tanaka, Taisuke Kawai, and Michael M Bornstein. The use and performance of artificial intelligence applications in dental and maxillofacial radiology: A systematic review. *Dentomaxillofacial Radiology*, 49(1):20190107, 2020.

[169] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen Awm Van Der Laak, Bram Van Ginneken, and Clara I Sánchez. A survey on deep learning in medical image analysis. *Medical image analysis*, 42:60–88, 2017.

[170] Ravleen Nagi, Konidena Aravinda, N Rakesh, Rajesh Gupta, Ajay Pal, and Amrit Kaur Mann. Clinical applications and performance of intelligent systems in dental and maxillofacial radiology: A review. *Imaging Science in Dentistry*, 50(2):81, 2020.

[171] DH Kim and T MacKinnon. Artificial intelligence in fracture detection: transfer learning from deep convolutional neural networks. *Clinical radiology*, 73(5):439–445, 2018.

[172] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[173] Seok Won Chung, Seung Seog Han, Ji Whan Lee, Kyung-Soo Oh, Na Ra Kim, Jong Pil Yoon, Joon Yub Kim, Sung Hoon Moon, Jieun Kwon, Hyo-Jin Lee, et al. Automated detection and classification of the proximal humerus fracture by using deep learning algorithm. *Acta orthopaedica*, 89(4):468–473, 2018.

[174] Naofumi Tomita, Yvonne Y Cheung, and Saeed Hassanpour. Deep neural networks for automatic detection of osteoporotic vertebral fractures on ct scans. *Computers in biology and medicine*, 98:8–15, 2018.

[175] Python. Available online:. https://www.python.org/, Accessed on 24 June 2021.

[176] PyTorch. Available online:. https://pytorch.org/, Accessed on 3 July 2020.

[177] Fastai. Available online:. https://docs.fast.ai/, Accessed on 3 February 2021.

[178] Scikit learn. Available online:. https://scikit-learn.org/stable/, Accessed on 6 July 2020.

[179] Pydicom. Available online:. https://pydicom.github.io/, Accessed on 8 July 2020.

[180] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

[181] Sylvain Arlot and Alain Celisse. A survey of cross-validation procedures for model selection. *Statistics surveys*, 4:40–79, 2010.

[182] Gareth James and Trevor Hastie. Generalizations of the bias/variance decomposition for prediction error. *Dept. Statistics, Stanford Univ., Stanford, CA, Tech. Rep*, 1997.

[183] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

[184] Jeremy Howard and Sylvain Gugger. Fastai: a layered api for deep learning. *Information*, 11(2):108, 2020.

[185] James A Hanley and Barbara J McNeil. The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology*, 143(1):29–36, 1982.

[186] Monica Murero. Building artificial intelligence for digital health. a socio-tech-med approach, and a few surveillance nightmares. *Etnografia e ricerca qualitativa*, 13(3):374–388, 2020.

[187] Murat H Sazli. A brief review of feed-forward neural networks. *Communications Faculty of Sciences University of Ankara Series A2-A3 Physical Sciences and Engineering*, 50(01), 2006.

[188] Andrea Apicella, Francesco Donnarumma, Francesco Isgrò, and Roberto Prevete. A survey on modern trainable activation functions. *Neural Networks*, 138:14–32, 2021.

[189] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

[190] Himansu Das, Ajay Kumar Jena, Janmenjoy Nayak, Bighnaraj Naik, and HS Behera. A novel pso based back propagation learning-mlp (pso-bp-mlp) for classification. In *Computational intelligence in data mining-volume 2*, pages 461–471. Springer, 2015.

[191] Douglas M Kline and Victor L Berardi. Revisiting squared-error and cross-entropy functions for training neural network classifiers. *Neural Computing & Applications*, 14(4):310–318, 2005.

[192] EJ Gilroy, RM Hirsch, and TA Cohn. Mean square error of regression-based constituent transport estimates. *Water Resources Research*, 26(9):2069–2077, 1990.

[193] Greg Van Houdt, Carlos Mosquera, and Gonzalo Nápoles. A review on the long short-term memory model. *Artificial Intelligence Review*, 53(8):5929–5955, 2020.

[194] Mohamed Akram Zaytar and Chaker El Amrani. Sequence to sequence weather forecasting with long short-term memory recurrent neural networks. *International Journal of Computer Applications*, 143(11):7–11, 2016.

[195] Hafed Zarzour, Yaser Jararweh, Mahmoud M Hammad, and Mohammed Al-Smadi. A long short-term memory deep learning framework for explainable recommendation. In *2020 11th International Conference on Information and Communication Systems (ICICS)*, pages 233–237. IEEE, 2020.

[196] Thomas Fischer and Christopher Krauss. Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, 270(2):654–669, 2018.

[197] Mercy M Moila and Thipe I Modipa. The development of a sepedi text generation model using long-short term memory. In *Proceedings of the 2nd International Conference on Intelligent and Innovative Computing Applications*, pages 1–5, 2020.

[198] Haşim Sak, Andrew Senior, and Françoise Beaufays. Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. *arXiv preprint arXiv:1402.1128*, 2014.

[199] P. Bryant and K. Johnsen. *The Principles of Circular Accelerators and Storage Rings*. Cambridge University Press, Cambridge, January 2005.

[200] Christian Grech, Marco Buzio, and Nicholas Sammut. A magnetic measurement model for real-time control of synchrotrons. *IEEE Transactions on Instrumentation and Measurement*, 69(2):393–400, 2019.

[201] R Assmann, M Giovannozzi, Y Papaphilippou, F Zimmermann, A Caldwell, and G Xia. Generation of short proton bunches in the cern accelerator complex. *Proc. PAC09*, page 4542, 2009.

[202] Vardan Khachatryan, Albert M Sirunyan, Armen Tumasyan, Wolfgang Adam, E Asilar, Thomas Bergauer, Johannes Brandstetter, Erica Brondolin, Marko Dragicevic, Janos Erö, et al. Measurement of long-range near-side two-particle angular correlations in p p collisions at s= 13 tev. *Physical review letters*, 116(17):172302, 2016.

[203] Michel Chanel. Leir: the low energy ion ring at cern. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 532(1-2):137–143, 2004.

[204] G Yu Drobychev, U Gendotti, I Boscolo, H Walters, M Büchner, André Rubbia, MK Oberthaler, P Nédélec, S Zavatarelli, C Carraro, et al. Proposal for the aegis experiment at the cern antiproton decelerator (antimatter experiment: Gravity, interferometry, spectroscopy). Technical report, 2007.

[205] M Buzio, L Walckiers, A Beaumont, C Petrone, D Giloteaux, S Gilardoni, P Galbraith, and G Golluccio. Development of upgraded magnetic instrumentation for cern real-time reference field measurement systems. Technical report, 2010.

[206] Klaus Hanke, Julie Coupard, Heiko Damerau, Anne Funken, Brennan Goddard, Alessandra Lombardi, Django Manglunki, Simon Mataguez, Malika Meddahi, Bettina Mikulec, et al. The lhc injectors upgrade (liu) project at cern: proton injector chain. In *Proceedings, 8th International Particle Accelerator Conference (IPAC 2017): Copenhagen, Denmark, May 14-19*, 2017.

[207] G Franzini, O Coiro, D Pellegrini, M Serio, A Stella, M Pezzetta, and M Pullia. Final design and features of the b-train system of cnao. *Proc. IPAC*, pages 1–3, 2010.

[208] Open Hardware Repository. Available online:. https://ohwr.org/, Accessed on 20 January 2022.

[209] Carles Kishimoto Bisbe and Tony Cass. Cern technical network upgrade. Technical report, 2019.

[210] Igal Galili, Dov Kaplan, and Yaron Lehavi. Teaching faraday's law of electromagnetic induction in an introductory physics course. *American journal of physics*, 74(4):337–343, 2006.

[211] Anthony Beaumont, Marco Buzio, and Giovanni Boero. Ferrimagnetic resonance field sensors for particle accelerators. *Review of Scientific Instruments*, 90(6):065005, 2019.

[212] Patrick Loschmidt, Georg Gaderer, Natasa Simanic, Arshad Hussain, and Pedro Moreira. White rabbit-sensor/actuator protocol for the cern lhc particle accelerator. In *SENSORS, 2009 IEEE*, pages 781–786. IEEE, 2009.

[213] PPM Jansweijer, HZ Peek, and E De Wolf. White rabbit: Sub-nanosecond timing over ethernet. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 725:187–190, 2013.

[214] C Roderick and R Billen. The lsa database to drive the accelerator settings. In *12th International Conference On Accelerator And Large Experimental Physics Control Systems, Kobe, Japan*, pages 12–16, 2009.

[215] About CentOS. Available online:. https://centos.org/about/, Accessed on 20 January 2022.

[216] Michel Arruat, Leandro Fernandez, Stephen Jackson, Frank Locci, Jean-Luc Nougaret, Maciej Peryt, Anastasiya Radeva, Maciej Sobczak, and Marc Vanden Eynden. Front-end software architecture. In *ICALEPCS*, volume 7, page 310. Citeseer, 2007.

[217] Julian Lewis, Pablo Alvarez, Jean-Claude Bau, Stephane Deghaye, Ioan Kozsar, and Javier Serrano. The cern lhc central timing, a vertical slice. *ICALEPCS07*, 2007.

[218] Julian Lewis, Jean-Claude Bau, Javier Serrano, David Dominguez, Pablo Alvarez Sanchez, et al. The evolution of the cern sps timing system for the lhc era. In *Proceedings of ICALEPCS*, pages 125–129, 2003.

Appendix A

Neural Network Architectures

Neural network architectures can be divided into two main groups: multi-layer neural networks, wherein no cycle is created by the connections between nodes, and recurrent neural networks, which on the contrary contain loops. In the following paragraphs, the two different categories of neural networks are described in detail.

A.1 Multi-layer Neural Networks

A multi-layer neural network, also known as MLP, aims to approximate a function f^* [91], defining a mapping $y = f(x;\theta)$ and learning the parameters θ that return the best function estimation. For example, in case of classification problem, the mapping $y = f^*(x)$ maps the input, x, to a certain category y. Multi-layer neural networks represent the most widely used model of feed-forward neural networks [187].
They consist of N basic computing units, called neurons, that are organized in $L > 1$ ordered layers [188]. The first layer consists of m input variables, $x_1, ..., x_m$; the last layer is the output layer, which provides the output of the model, while the layers between the input and the output layer are called hidden layers. Each neuron of a certain layer l (with $1 \leq l \leq L$) receives connections from all the units, coinciding with the input variables if $l = 1$, of the previous layer and sends its output to all the neurons of the next layer, as shown in Fig. A.1.

The units in each layer are independent of each other, which means that there are no connections between the neurons of a single layer. Each connection is associated with a real value, called weight. The computation flow departs from the input layer

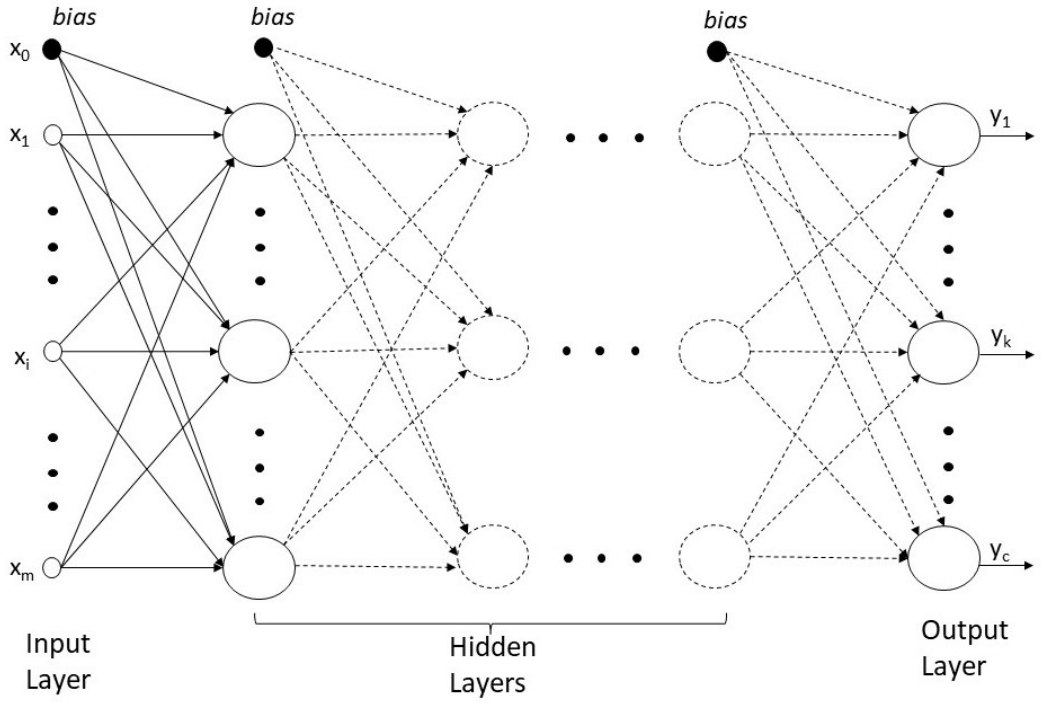

Fig. A.1 Multi-layer neural network general architecture.

until the output layer without any feedback (forward propagation). When feedbacks are present in the neural network, it becomes a recurrent neural network, as described in Sec. A.2.
The neuron computation can be distinguished in two steps, as shown in Fig. A.2: the computation of the neuron input and, subsequently, the computation of its output.

The input a_i^l of a generic neuron i belonging to a generic layer l is a weighted linear combination of the incoming values, which represent the previous layer's output:

$$a_i^l = \sum_j W_{ij}^l z_j^{l-1} + b_i^l \tag{A.1}$$

where W_{ij}^l corresponds to the weight of the connection from the j-th neuron of the layer $l-1$ to the i-th neuron of the layer l; b_i^l represents a parameter called bias and z_j^{l-1} is the output of the j-th neuron of the layer $l-1$ or the input variables in case of $l=1$. The index j varies over the indexes of the neurons belonging to the layer $l-1$ from which the connections to the i-th neuron depart. The corresponding matrix notation of the neuron's input is given by $a_i^l = \mathbf{W}_i^l \mathbf{z}^{l-1} + b_i$, with $\mathbf{W}_i^l$ representing

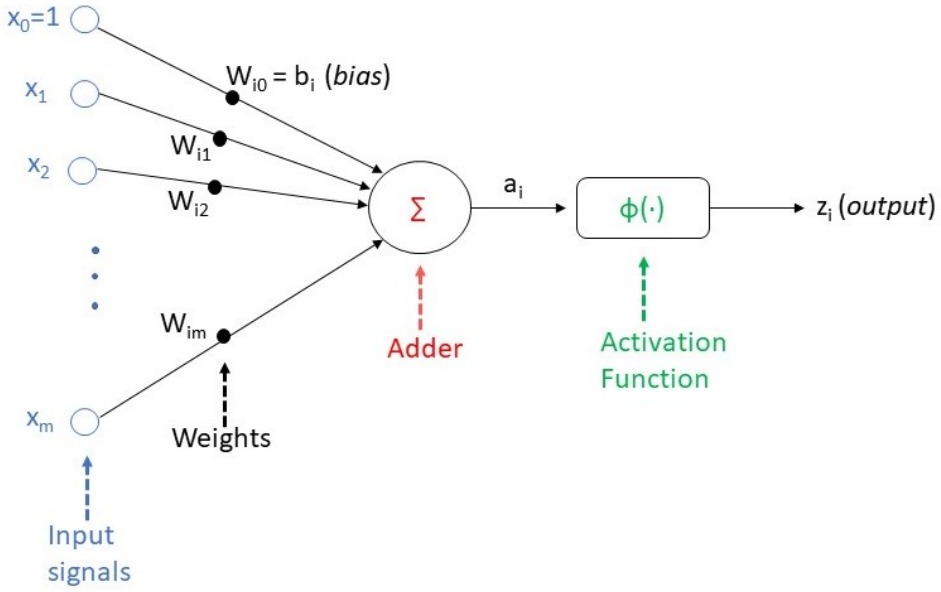

Fig. A.2 Representation of a generic neuron computation with m input signals.

the weight column vector of the i-th neuron of the l-th layer, while $\mathbf{z}^{l-1}$ corresponds to the output column vector of the neurons of the previous layer, $l-1$. In case of $l=1$, $\mathbf{z}^{l-1}$ represents the input variables. On the other side, the output z_i^l of a generic neuron i belonging to a generic layer l is usually calculated by a nonlinear function $\phi()$, called activation function, as $z_i^l = \phi(a_i^l)$. In general, the activation function used in the multi-layer neural networks belongs to the family of rectified functions, such as ReLU [189], but they can be also classic functions such as logistic sigmoid or tanh.

Multi-layer neural networks represent a broad class of different neural networks, from the simplest ones, like the shallow or vanilla network with a single hidden layer (Fig. A.3), to the deep and more complex ones, like the convolutional ones, described in Sec. 2.3.1.

The mathematical expression of the generic output y_k for a shallow neural network is given by Eq. A.2:

$$y_k = \tilde{\phi}\Big(\sum_{j=0}^{n} W_{kj} \underbrace{\phi\Big(\sum_{j=0}^{m} W_{ji} x_i\Big)}_{z_j}\Big) \tag{A.2}$$

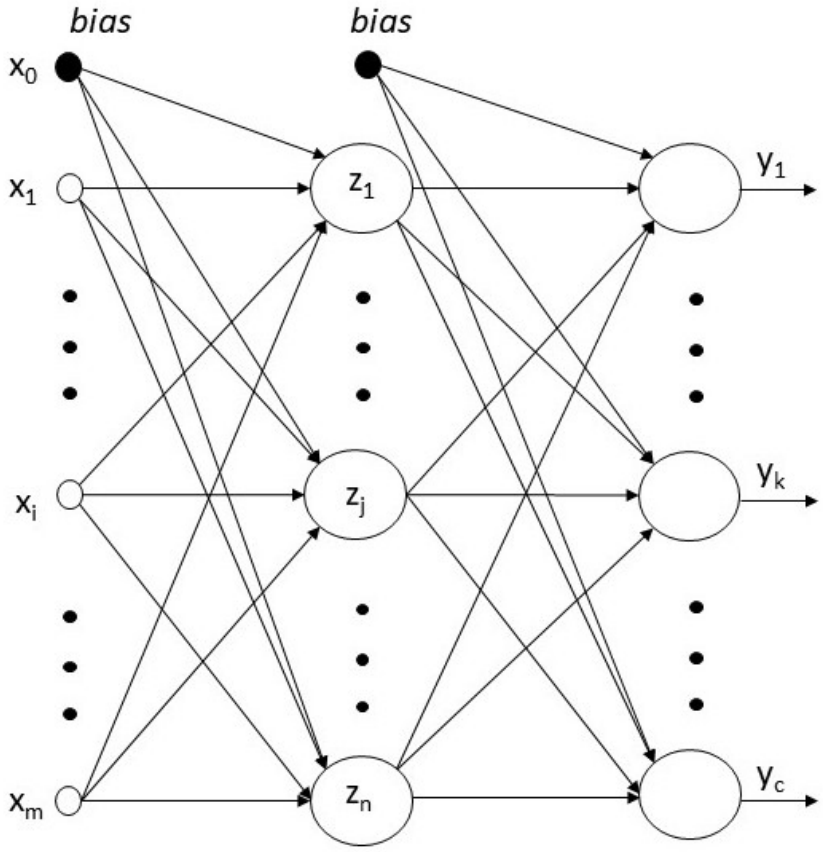

Fig. A.3 Shallow or vanilla architecture characterized by a single hidden layer.

In order to allow the convergence of the function $f(x)$, implemented by the network, to $f^*(x)$, the multi-layer neural network is trained with an update rule generally based on the derivative calculation performed by the back-propagation algorithm [190], as described in detail in Sec. 2.2. Starting with random weights, in a multi-layer neural network, they are continually updated to minimize the loss function. Depending on the problem type, a multi-layer neural network may use different loss functions. For example, the loss function used for classification problem is Cross-Entropy [191], while for regression problem multi-layer neural network makes use of the Square Error loss function [192]. After computing the loss, the latter is propagated back from the output layer to the previous layers through a backward pass, which gives each weight parameter an update value to reduce the loss.

The popularity of multi-layer neural network is due to the capability to learn non-linear models and, in particular, to the capability to approximate arbitrarily well any continuous functional mapping defined on a compact input domain, provided that the number of hidden neurons is sufficiently large and the activation functions of the hidden neurons satisfy some suitable properties [158]. In this sense, multi-layer networks are universal approximators. On the other hand, it is important to note that a multi-layer neural network is sensible to the input data scaling and non-convex loss function may contain more than one local minimum. Therefore, it is possible to reach a different validation accuracy starting from different random weight initializations.

A.2 Recurrent Neural Networks

Recurrent neural networks (RNNs) are a category of neural networks used for handling sequence of data $x(1), ..., x(n)$ [91].

The idea that allowed to pass from feed-forward neural networks to RNNs arose in machine learning research of the 1980 and deals with the sharing of parameters among the several parts that constitute a model. By means of this mechanism, it is possible to employ the model in different situations, such as different sequence lengths, being capable to generalize. This would not be possible if the model used different parameters for every value of the time index. Furthermore, parameters sharing is also crucial when there is the possibility that certain information can appear in different positions in the sequence. For instance, if there is the need to

extract the year contained in a sentence, an RNN will be able to easily identify the year regardless of its position within the sentence, thanks to the sharing of the same weights through several time instances.
Convolution with a 1-D temporal sequence has similar aspects to an RNN, from the parameters sharing point of view, and it represents the base for the time-delay neural networks [108]. Parameters are shared in a convolutional operation in the sense that it outputs a sequence where each element is a function of a certain number of input neighboring elements. This is due to the application of the same convolutional filter at every time instance. In the case of RNNs, parameters sharing occurs in a different way. In fact, in this case, each element of the output is a function of the previous output elements and is generated with the same update rule used for the previous outputs.
The time index in a sequence of values does not necessarily have to be related to the passing of time in the real environment, but often it is related only to the position of the value in the sequence. A way to represent graphically an RNN is through a computational graph including cycles. The latter indicates that the current value of a variable affects the value of the same variable at a future time instance.

The expression that describes a dynamical system is shown by Eq. A.3:

$$s^t = g(s^{t-1}; \theta) \tag{A.3}$$

with s^t the state of the system. Eq. A.3 is recurrent since the state of the system, s, at time t, depends on the state of the system at time $t-1$. In case of finite number (n) of time steps, the computational graph can be "unfolded" employing the definition, shown in Eq. A.3, $n-1$ times. For instance, unfolding Eq. A.3 with n=3, the state of the system at time 3 is given by Eq. A.4:

$$s^3 = g(s^2; \theta) = g(g(s^1; \theta); \theta) \tag{A.4}$$

The result of repeatedly employing the definition on a recurrent expression leads to an equation without recurrence, that can be described by a directed computational graph without cycles, as shown in Fig. A.4.

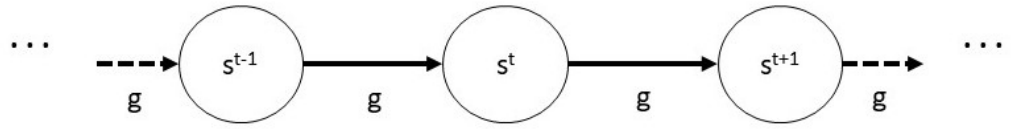

Fig. A.4 Unfolded graph representation of a dynamical system, where the nodes contain the state of the system at a certain time while the function g maps the system's state at t time to the system's state at $t+1$. In all the time instances, the same parameters, θ, employed to parametrize g, are used.

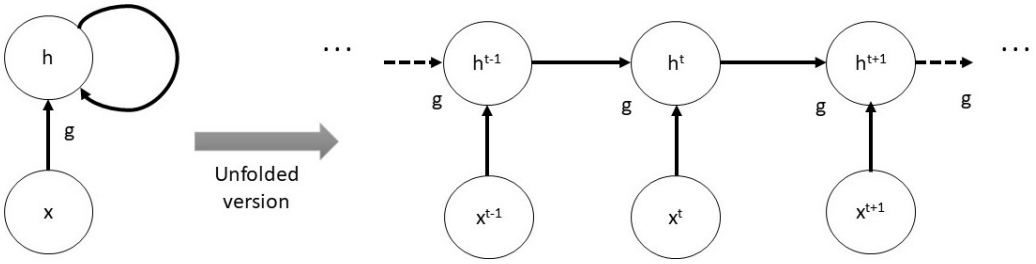

Fig. A.5 Computational graph representation of an RNN without output. The RNN receives an external input, x, and integrates it into the state, h, that flows forward in time. On the right of the figure, the unfolded version of the computational graph is presented. In this case, each node is linked to a specific time step.

There are several variants of RNNs since basically any function including recurrence can be classified as an RNN. Eq. A.5 is typically used to define the hidden units of the network:

$$h^t = g(h^{t-1}, x^t; \theta) \tag{A.5}$$

where x^t is an external signal. The corresponding computational graph is shown in Fig. A.5.

During the training of an RNN, that allows the network to predict future values starting from past values, the state, $h(t)$, represents a kind of summary of the most relevant features of the past inputs values up to the time step t. Depending on the network's task and on the training process, $h(t)$ may keep some characteristics of the sequence of the past values with more precision with respect to other characteristics. Starting from Eq. A.5, two different graphical representations can be obtained for the RNN, as shown in Fig. A.5. The first approach is to introduce one node for each

element that may be present in a physical implementation of the architecture, as shown on the left side of Fig. A.5. In this case, the model represents a circuit working in real-time, where the current state of the physical parts can affect their future state. The second approach is to represent the RNN by means of an unfolded graph, where each element is associated with different variables: one variable per time instance, which describes the state of the element at that time instance. Therefore, the unfolded graph consists of separate nodes for each variable, as shown on the right side of Fig. A.5. Thus the unfolding can be seen as the process that maps a circuit diagram (left of Fig. A.5) to a computational graph with parts that are repeated (right of Fig. A.5). The size of an unfolded graph relies on the length of the sequence.
The unfolded recurrence after t time instances can be expressed by Eq. A.6:

$$h^t = f^t(x^t, x^{t-1}, ..., x^1) = g(h^{t-1}, x^t; \theta) \tag{A.6}$$

Function f^t considers the total past sequence $(x^t, x^{t-1}, ..., x^1)$ as input, generating the current state, h^t. However, f^t can also be factorized, applying the g function repeatedly. In this case, the same input size characterizes the learned model, since the latter is defined with regard to the passage from a certain state to another one instead of defined with regard to the variable length history of the states. Moreover, the same function g can be used with the same parameters for the transition between the different states. These aspects allow a better generalization of the model that will be able to analyze sequence lengths that are not included in the training set.
Finally, as anticipated before, there are a lot of variants of RNNs and some examples are the following:

- RNNs that generate an output at every time instance and contain recurrent connections among hidden units, as shown in Fig. A.6;
- RNNs that generate an output at every time instance and contain recurrent connections from the output unit at a certain time instance to the hidden unit at the next time instance, as shown in Fig. A.7;
- RNNs that contain recurrent connections among hidden units that scan the whole sequence and after that it generates a single output, as shown in Fig. A.8.

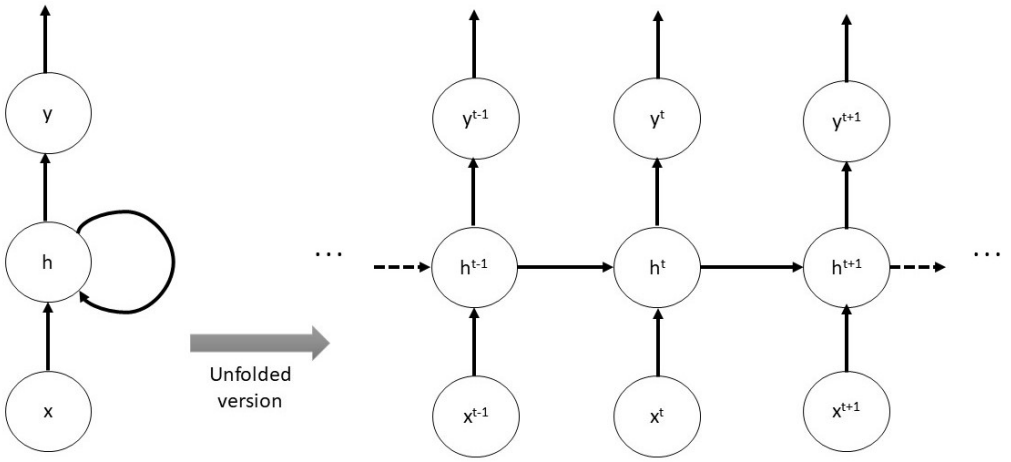

Fig. A.6 RNN architecture with recurrent connections among hidden units and an output at every time instance.

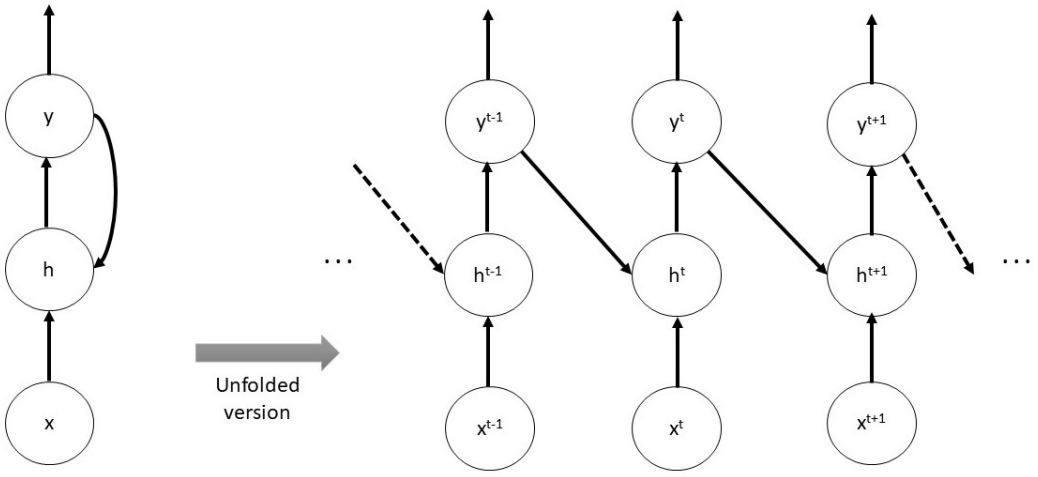

Fig. A.7 RNN architecture with recurrent connections from the output unit at a certain time instance to the hidden unit at the next time instance and an output at every time instance.

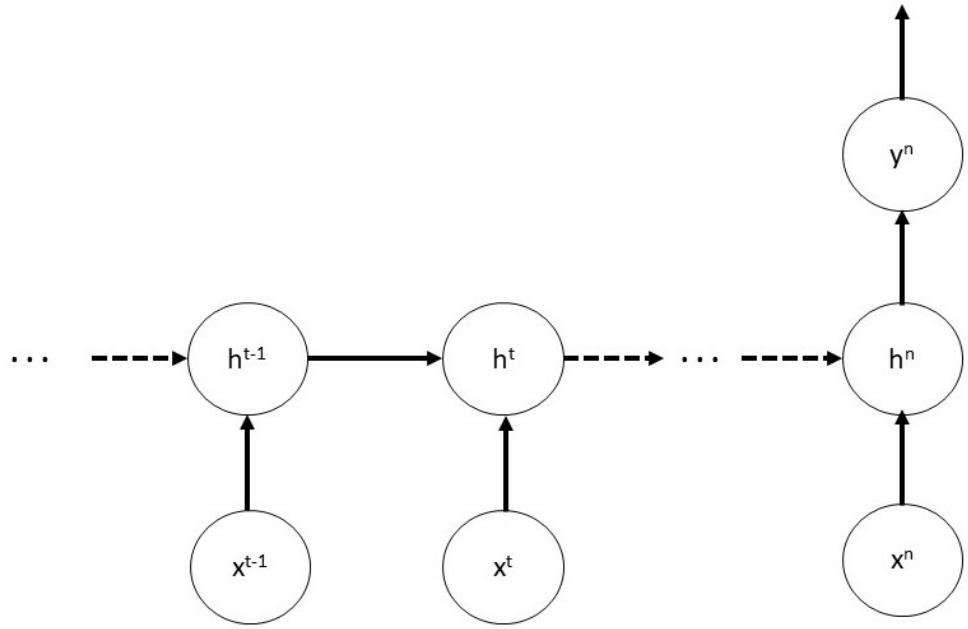

Fig. A.8 RNN architecture with recurrent connections among hidden units and a single output at the end of the sequence scan.

Particular interest is addressed to Long Short-Term Memory (LSTM) that is another type of RNN architecture [193], used in deep learning field for handling sequences of data. For example, LSTM has been used in weather prediction problems [194], product recommendation [195], stock market forecasting problems [196], text generation [197] or speech recognition problems [198].
The neurons of an LSTM architecture are called cells (there can be several layers and several cells per layer) and contain gates and weights. The gate is the peculiarity of the LSTM models. There are three types of gates: the input gate, the forget gate and the output gate. The forget gate is responsible for eliminating unnecessary information, similarly to the human behavior that does not consider information or events that are not necessary or relevant to make a decision. On the other hand, the input gate is responsible for adding new information to the cell, just as humans consider new information in addition to the information already acquired. Finally, the output gate chooses the useful information on the basis of the previous cell output, the cell state and the new data, producing the cell output which feeds the next cell.

Appendix B

B-train: a Real-Time Magnetic Measurement System for Synchrotron Control

In this Appendix, an overview of the "B-train" systems is presented. Part of the work described in this section was also previously published in [3].
The accurate knowledge of the magnetic field values generated by the dipole magnets is crucial for the proper functioning of a synchrotron [199]. At this aim, the B-train system has been realized and is currently in operation in six machines [200]: it performs real-time magnetic field measurements, in iron-dominated electromagnets characterized by strong non-linear effects, and provides them to various users, such as radiofrequency cavities, beam control and diagnostics, power converter control. The six machines that are using the Btrain system to determine the dipole field are the LEIR, the Proton Synchrotron (PS), the Proton Synchrotron Booster (PSB), the Super Proton Synchrotron (SPS), the Antiproton Decelerator (AD) and the Extra Low ENergy Antiproton deceleration ring (ELENA). PSB, PS, SPS are part of the chain of injectors delivering protons to the LHC, as shown in Fig. B.1.

In particular, each machine in the chain of particle accelerators complex increases the energy of the particles by a factor typically about 20 to deliver them to the next more powerful accelerator or to experiments. Four main experiments are present in the LHC ring (CMS, ATLAS, LHCb and ALICE) and dozen of smaller experiments, also known as fixed-target. PSB accelerates the protons to 1.4 GeV to inject the beam

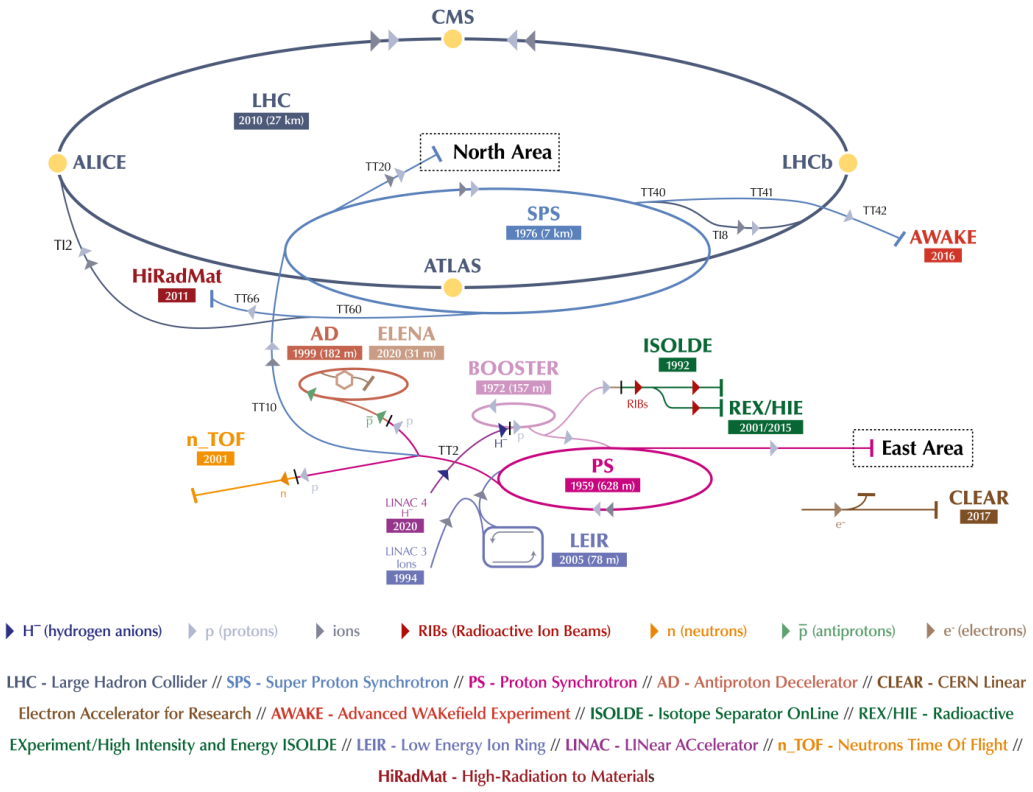

Fig. B.1 CERN's Accelerator Complex. The LHC is the last ring of the particle accelerators complex, along which the main four experiments are located (ALICE, CMS, LHCb and ATLAS). The smaller machines in the chain increase the energy of particles and provide beams not only to the LHC for the main experiments but also to a whole set of smaller experiments, known as fixed-target. (Image: © 2018 CERN [2]).

into the PS that accelerates the protons to the energy of 26 GeV. Finally, protons are sent to the SPS with an energy of 450 GeV to reach finally the LHC, where an energy of 6.5 TeV per beam is reached [201]. Here, the two beams circulate in opposite directions and collide inside four detectors (CMS, ATLAS, LHCb and ALICE), where a center-of-mass energy of 13 TeV at the collision point is reached [202].
In addition to protons, it is also possible to accelerate ions. For this purpose, LEIR has been introduced in the chain of the accelerators complex. It receives long bunches of ions from LINAC 3 and, splitting them into four shorter bunches, accelerates these bunches up to 72 MeV to deliver them to the PS [203]. Then, the pathway continues as in the case of protons.
Finally, AD and ELENA are responsible for the production of low-energy antiprotons to study antimatter [204].

The name "B-train" derives from the operating principle of the legacy systems [205], developed since the 1950s, where the discrete positive and negative pulse "trains" were used to propagate incrementally the measured magnetic field. The actual "B-train" system is called FIRESTORM (Field In REal-time STreaming from Online Reference Magnets), realized to replace the legacy systems in the context of a long-term consolidation project. It has been developed to deal with the High-Luminosity LHC upgrade, which requires higher beam intensity and an improved beam control in the whole injector chain [206]. After a considerable and successful test campaign, the deployment of the systems has been successfully completed for all six machines and the new systems are expected to be fully operational for the next physical run of LHC.
B-train systems are used not only for high energy physics research but are also present in medical contexts as in the case of the Heidelberg Ion-Beam Therapy Centre (HIT) [138] or the National Centre for Oncological Hadrontherapy (CNAO) [138] [207], a hospital center specialized in the treatment of tumors by hadrontherapy.

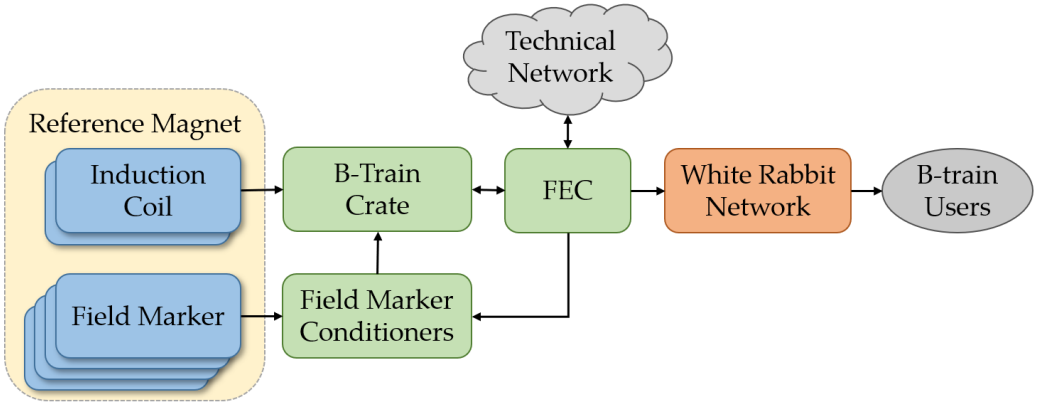

Fig. B.2 Block diagram representing the architectural layout of the new FIRESTORM system [3].

B.1 Hardware and Software Architecture of the New FIRESTORM system

In Fig. B.2 the architectural scheme of the new system is shown.

Unlike the legacy B-train, characterized by a large number of custom components with few small differences, the new system is based on a modular architecture consisting of parametric firmware and common hardware. This solution allows the FIRESTORM system to adapt efficiently to the different setups that characterize the six machines. In fact, each machine has different sensors configurations according to the needs of each synchrotron. The simplest case is represented by the SPS, in which the system is equipped with a long integral coil with one low-field marker that sends input to a single integration channel. In ELENA instead, the system is equipped with one integral coil but two field markers, each used according to the type of magnetic cycle: the high-field marker is used with cycles where antiprotons are decelerated, while the low-field marker is used for some special test cycles, where protons and H^- ions are accelerated. In the PS machine, there are particular magnets that consist of two parts: half with a focusing gradient and the other half with a defocusing one. The two halves must basically be handled as two independent magnets and, therefore, each part needs an integration channel and a dedicated coil. For the time being, there is only one low-field marker on each integration channel, but in the upcoming operating scenarios a second pair of high-field markers will be added to reach higher

accuracy both at the injection and extraction of the beam.
Therefore, a flexible architecture is required to effectively address these diverse requirements. The modular approach adopted by the design, along with the diagnostic capabilities and the possibility to configure the system remotely, made possible by the software integration within the accelerator control system, should enhance both longevity and maintenance of the system.
The main functionalities of the FIRESTORM B-train are implemented in a set of modules based on standard PCIe FPGA carrier cards (Simple PCI Express Carrier or SPEC card), contained in an industrial Front End Computer (FEC). The SPEC card includes a custom-made FPGA Mezzanine Card (FMC), which implements digital and analog I/O. The modular architecture allows separating various functionalities with a good level of scalability, improving both the maintainability and the adaptability of the system. The PCB layouts and firmware as well as all the other design elements are released on the Open Hardware Repository (OHWR) [208], a platform for electronics designers working at experimental physics facilities to stimulate collaboration on open hardware designs and the commercialization of the latter by industrial partners. The B-train crate acts as a central hub, allowing the link between the cards and the magnetic sensors. The access to the FECs occurs through the Technical Network [209], an isolated Ethernet network secured against malicious attacks, that is used to monitor and control the accelerators complex.
The combination of the induction coils and the field markers, installed inside the reference magnets of the specific machines, allows magnetic field measurements. In particular, the induction coil measures the rate of change of the magnetic field according to Faraday's law [210], while the field marker provides the necessary integration constant and its output passes through a conditioner before reaching the crate. The field marker conditioner removes the DC component, allowing the subsequent comparison with a known threshold, and then, if necessary, amplifies the signal [211].
Finally, the White Rabbit (WR) is a fiber-optic Ethernet-based network [212], that is used to propagate the measured magnetic field with noise immunity and high speed [213]. The Ethernet frames allow not only the transmission of the measured field but also other three versions of the magnetic field: the field provided by the legacy system, if available; the *simulated field*, that is a copy of the nominal field extracted from the magnetic cycle database (LHC Software Architecture (LSA) database [214]) as shown in Fig. B.3; the *predicted field*, that is a model of the

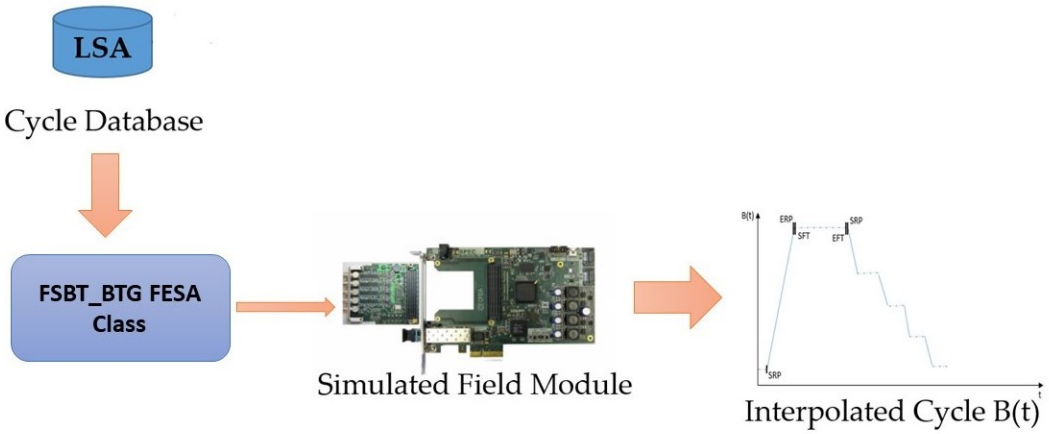

Fig. B.3 Schematic of the simulated Btrain. The FSBT_BTG FESA class (see Sec. B.1.1) downloads the cycle data vectors from the LSA database onto the FPGA of the simulated field module. A Bresenham line-drawing algorithm [4] generates the simulated field, thus providing the interpolated magnetic cycle.

magnetic field based on the magnet excitation current by means of neural network (see Chapter 3). The availability of these synchronized versions of the field facilitates system diagnostics and the improvement of the operational flexibility in particular situations (for example, when the measured field is not available).

B.1.1 Front End Computer and Real-Time Software

The FEC is the main element of the FIRESTORM system. It is an industrial diskless rack-mounted PC that hosts the principal electronic components. At CERN there are about two thousand FECs that interface with devices involved in synchrotron control (e.g. beam diagnostics instrumentation, vacuum and radio frequency control systems as well as B-train systems). The generation of FEC currently in use at CERN is the Siemens SIMATIC IPC847E with up to eleven free PCIe slots with 64-bit CentOS7 Linux as operating system [215]. As shown in Fig. B.4, the FEC hosts Device Driver and software based on a distributed and real-time framework named Front End Software Architecture (FESA) , that is the main component of accelerators control complex at CERN and GSI [216]. FESA is a framework dedicated to equipment specialists for the design, development, deployment and testing of their equipment software, called FESA class. The C++ FESA classes are deployed on the Btrain FECs.

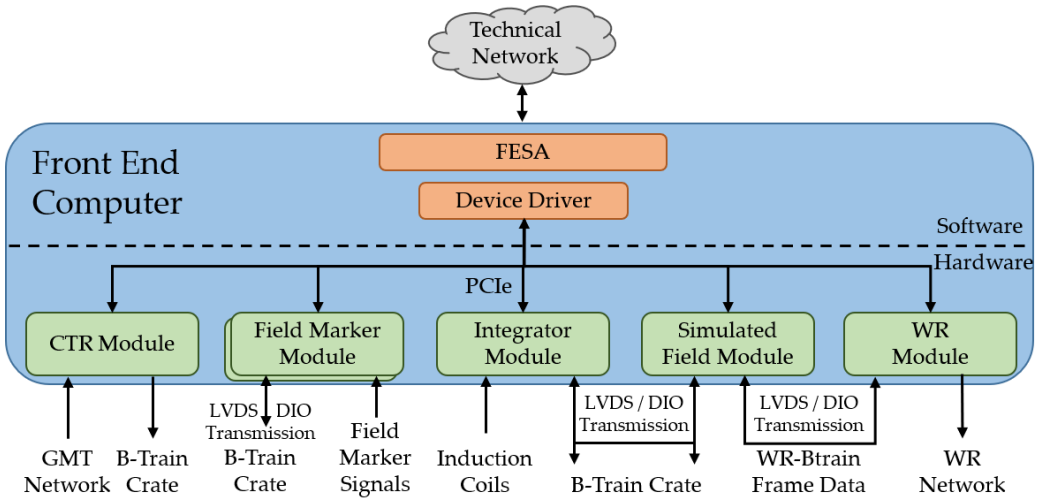

Fig. B.4 FEC's architecture, depicting the internal modules and the flow of data through the FEC [3].

FESA acts as an interface between the high-level infrastructure for accelerator control and the local hardware, accessed through user-written device drivers. FESA classes are closely integrated with the timing system [217], through the Central Timing Receiver (CTR) module, used for the synchronization of the accelerators and their subsystems, within a few microseconds [218]. The timing system comprises a network of coaxial cables that are independent for each accelerator and distribute hundreds of trigger pulses, representing significant timing events for beam or for the equipment control. In particular, a serial channel (the Machine Telegram) for each accelerator propagates relevant information regarding the beam, such as the magnetic cycle's type being run, the beam's destination and the type of the next cycle that will be run.

FESA framework has recently been fully equipped with the so-called *Pulse-to-Pulse Modulation* (PPM) properties, which enable/disable specific actions such as setting properties' items, based on the cycle type. PPM is a crucial new feature that allows automatic adaptation of parameters for sensor calibration according to the cycle type. This aspect is particularly important, for example, for the field marker level that for the best accuracy must be calibrated separately for each cycle type. As a result, with respect to the manual updating performed in the older B-train systems, by using this mechanism, the configuration process becomes much more flexible and reliable.

During the research activity carried out at CERN, six different FESA classes, be-

longed the FIRESTORM software, have been designed, developed, tested, deployed and maintained:

- B-train FESA class: it is the main class that provides the measured magnetic field to all the other subsystems;
- FSBT_BTG FESA class: it collects the nominal field from the LSA database and distributes the simulated field to all the other subsystems;
- CosmosCheckWRS FESA class: it monitors the status of the White Rabbit network;
- Comet_EVM FESA class: it is used for the environmental monitoring to control, for example, environmental temperature and humidity;
- BtrainSelection FESA class: it controls the Btrain Operational/Spare commutation (for each machine, there are two identical Btrains: one is the Operational and is sent to the various subsystems; the other is the Spare and is used as a backup system, replacing the Operational Btrain in case of malfunctions);
- BtrainDiagnostics FESA class: a supervisor class for Btrain diagnostics (for example, it shows both the field values measured by the operating system and the spare one to check for any inconsistencies).

The FIRESTORM FESA classes allow the tuning of more than two-hundred different configuration parameters, related to the B-train operation, as well as to access to more than 100 acquisition parameters, inherent to internal registers and measurement values. In Figs. B.5, B.6, B.7, B.9, B.13, B.15 and B.17, the FIRESTORM FESA class interfaces are shown, while in Figs. B.8, B.10, B.14, B.16 and B.18 the relationships existing between the properties for each class are shown. Finally, in Fig. B.19 an example of a Btrain FESA class property for the SPS accelerator is shown through the FESA Navigator, that is a graphical user interface to access to the class properties of a certain instance. In particular, from the property *Signals*, it is possible to plot the measured magnetic field through the item *bCycle*. In this case, the MD1 magnetic cycle of the operational system is displayed.

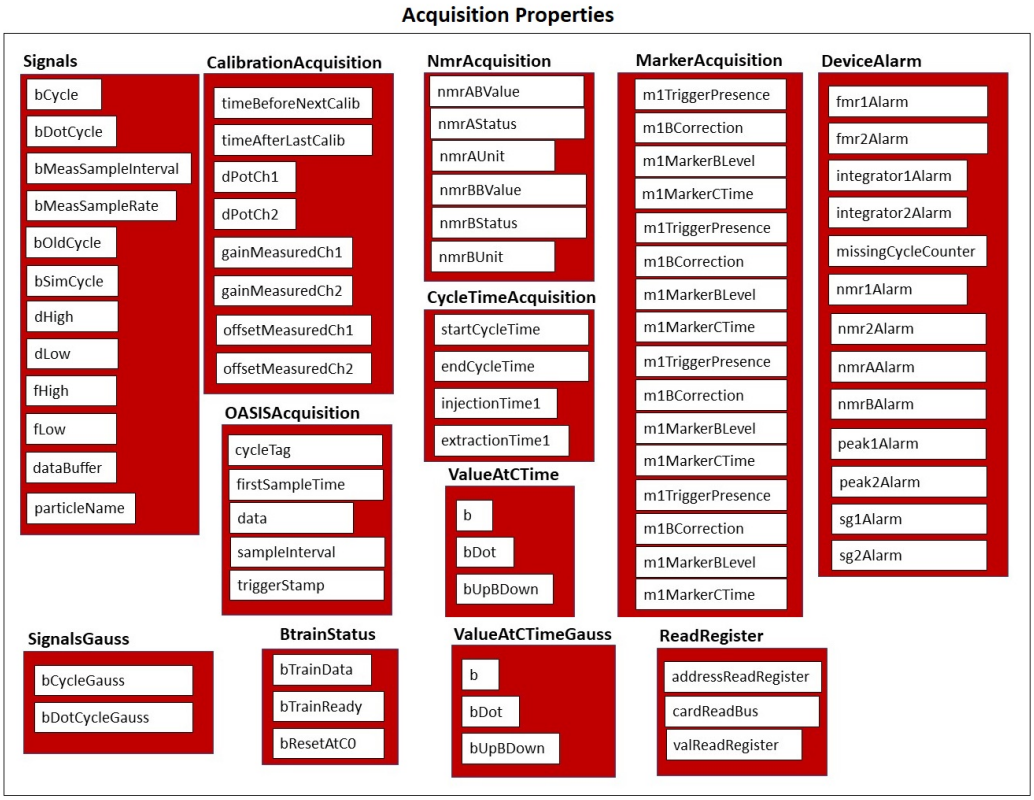

Fig. B.5 Btrain FESA class interface: Acquisition Properties.

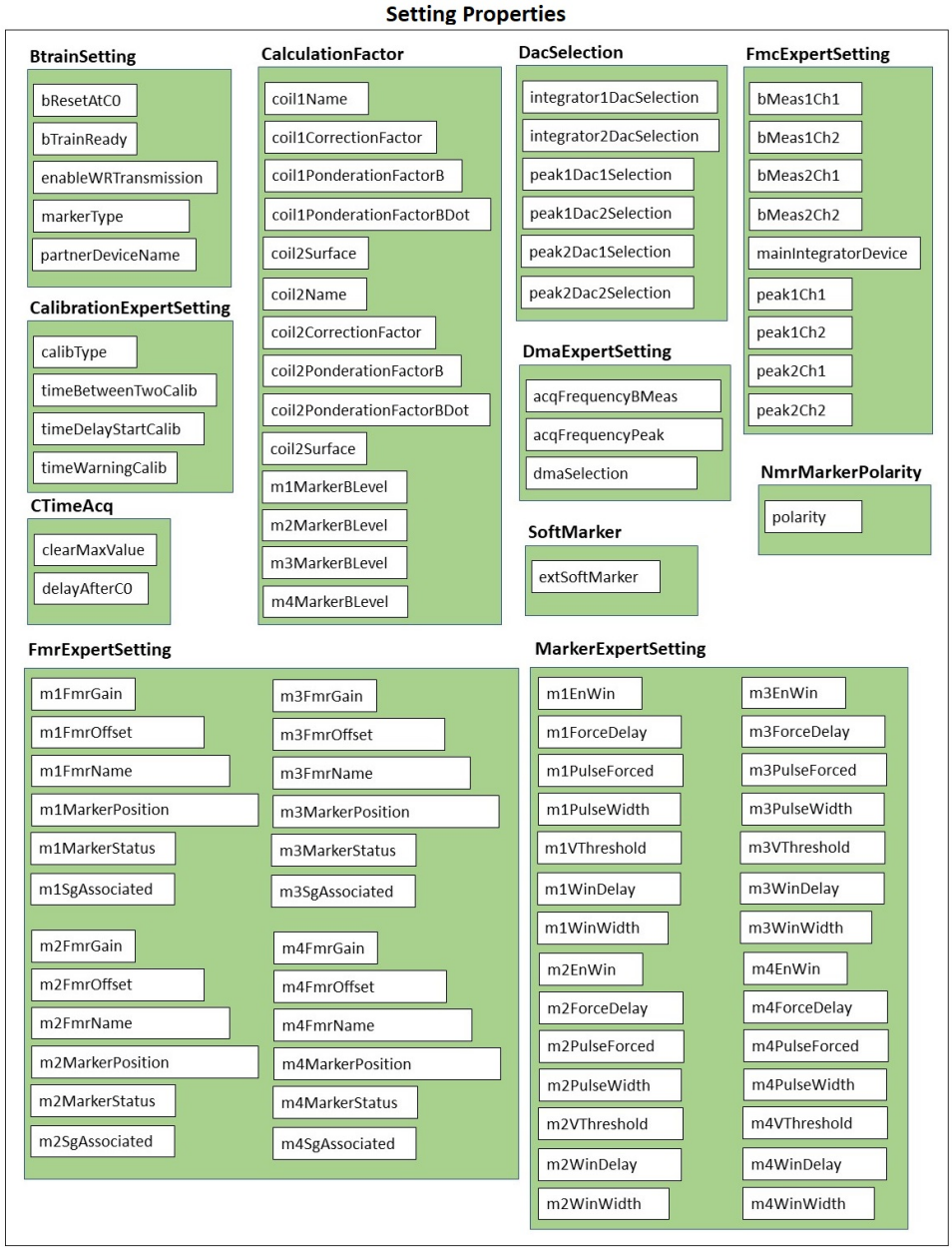

Fig. B.6 Btrain FESA class interface: Setting Properties - Part 1.

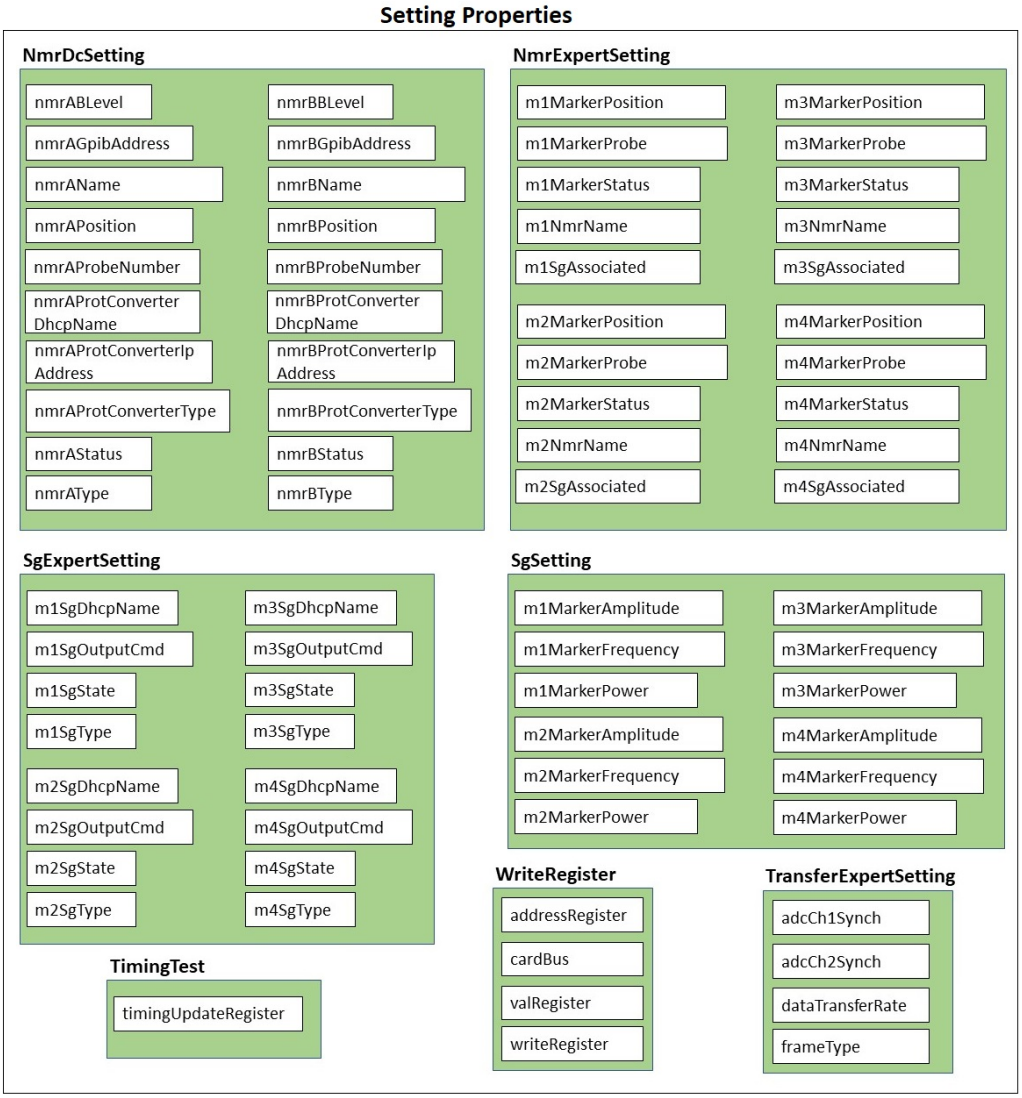

Fig. B.7 Btrain FESA class interface: Setting Properties - Part 2.

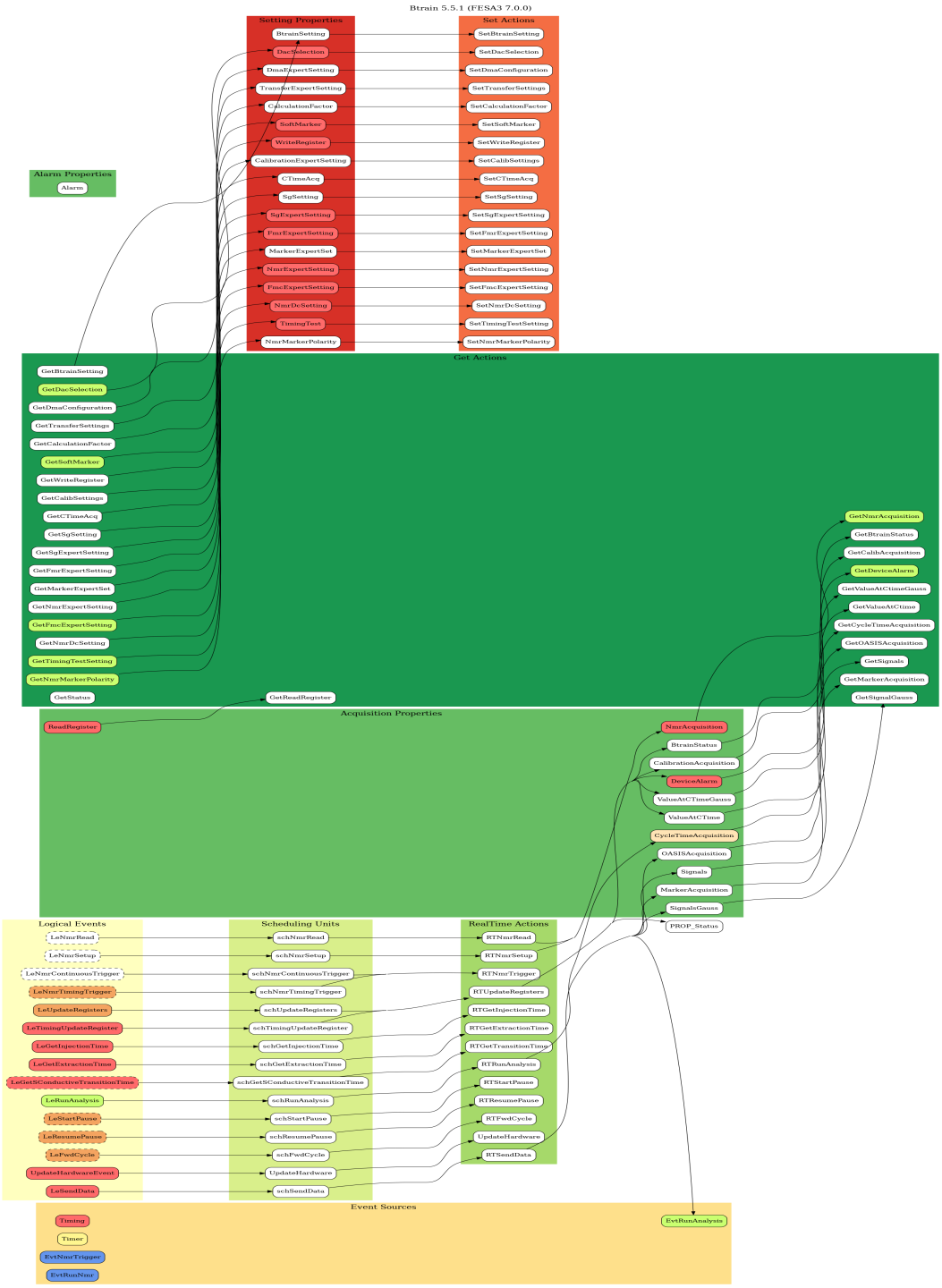

Fig. B.8 Btrain FESA class properties relationships.

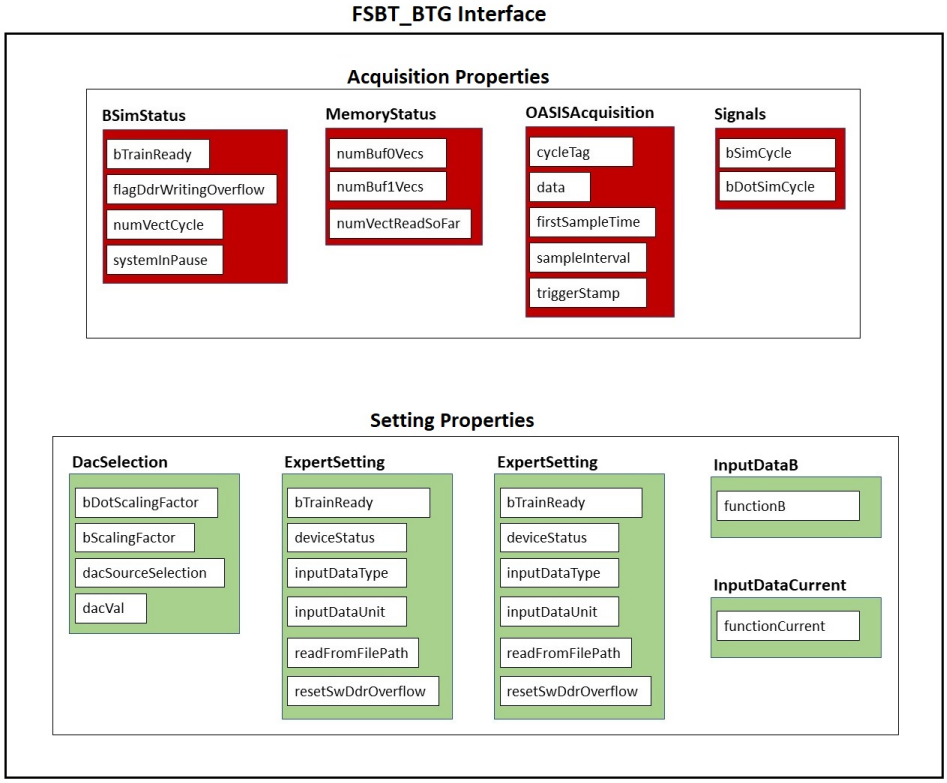

Fig. B.9 FSBT_BTG FESA class interface: Acquisition and Setting Properties.

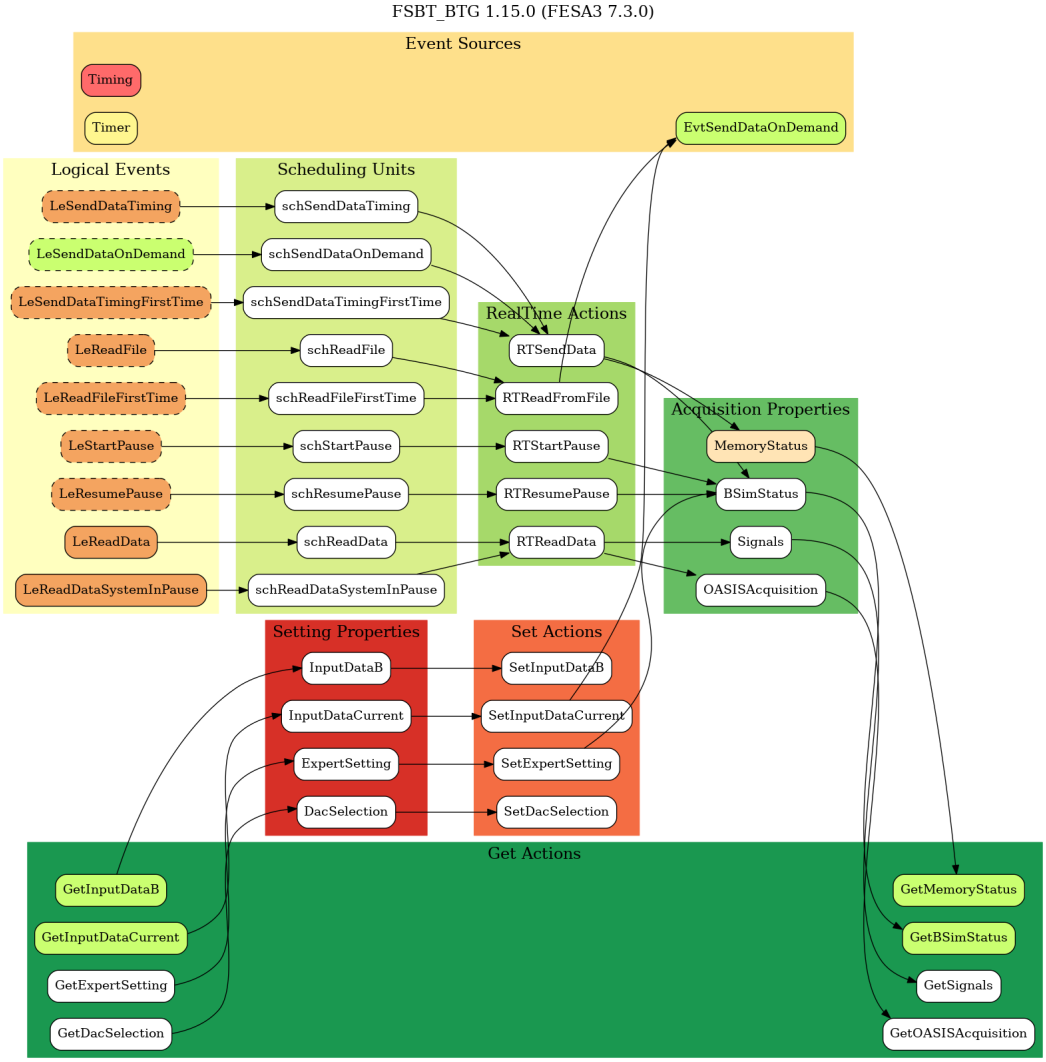

Fig. B.10 FSBT_BTG FESA class properties relationships.

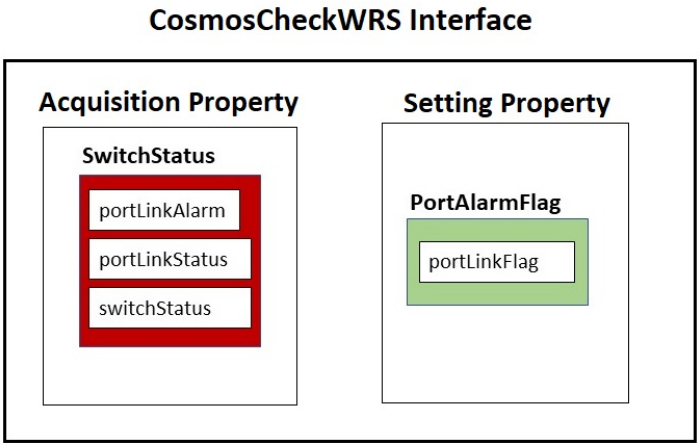

Fig. B.11 CosmosCheckWRS FESA class interface: Acquisition and Setting Properties.

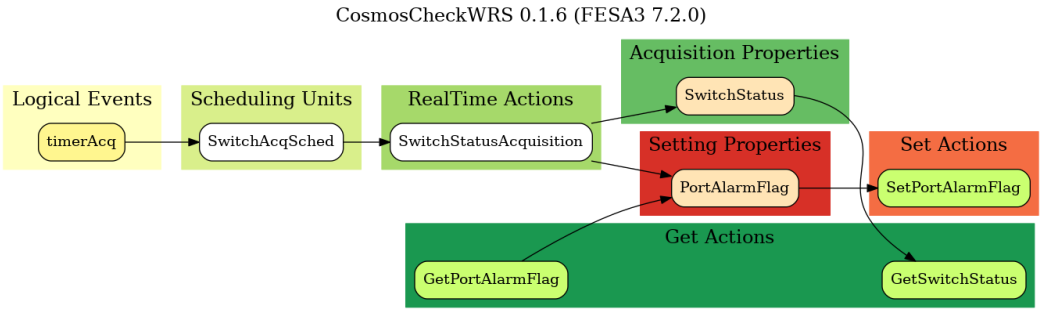

Fig. B.12 CosmosCheckWRS FESA class properties relationships.

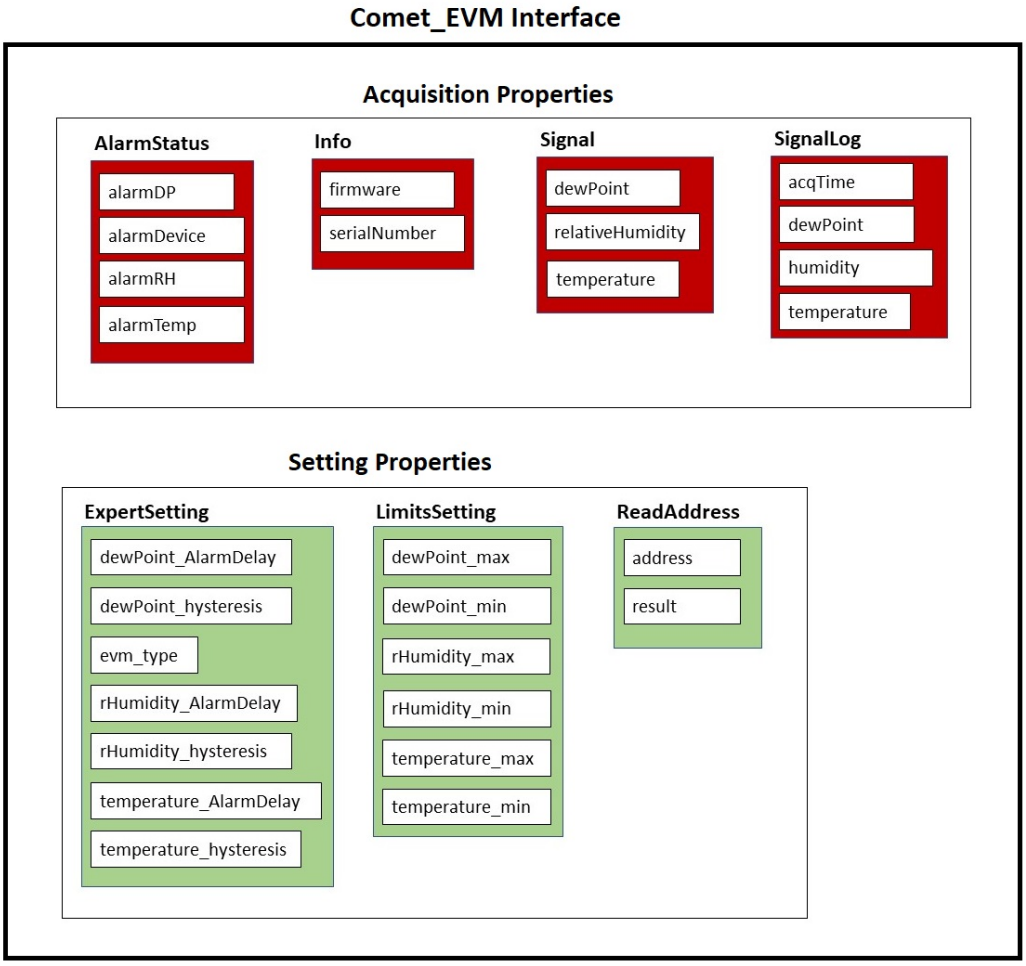

Fig. B.13 Comet_EVM FESA class interface: Acquisition and Setting Properties.

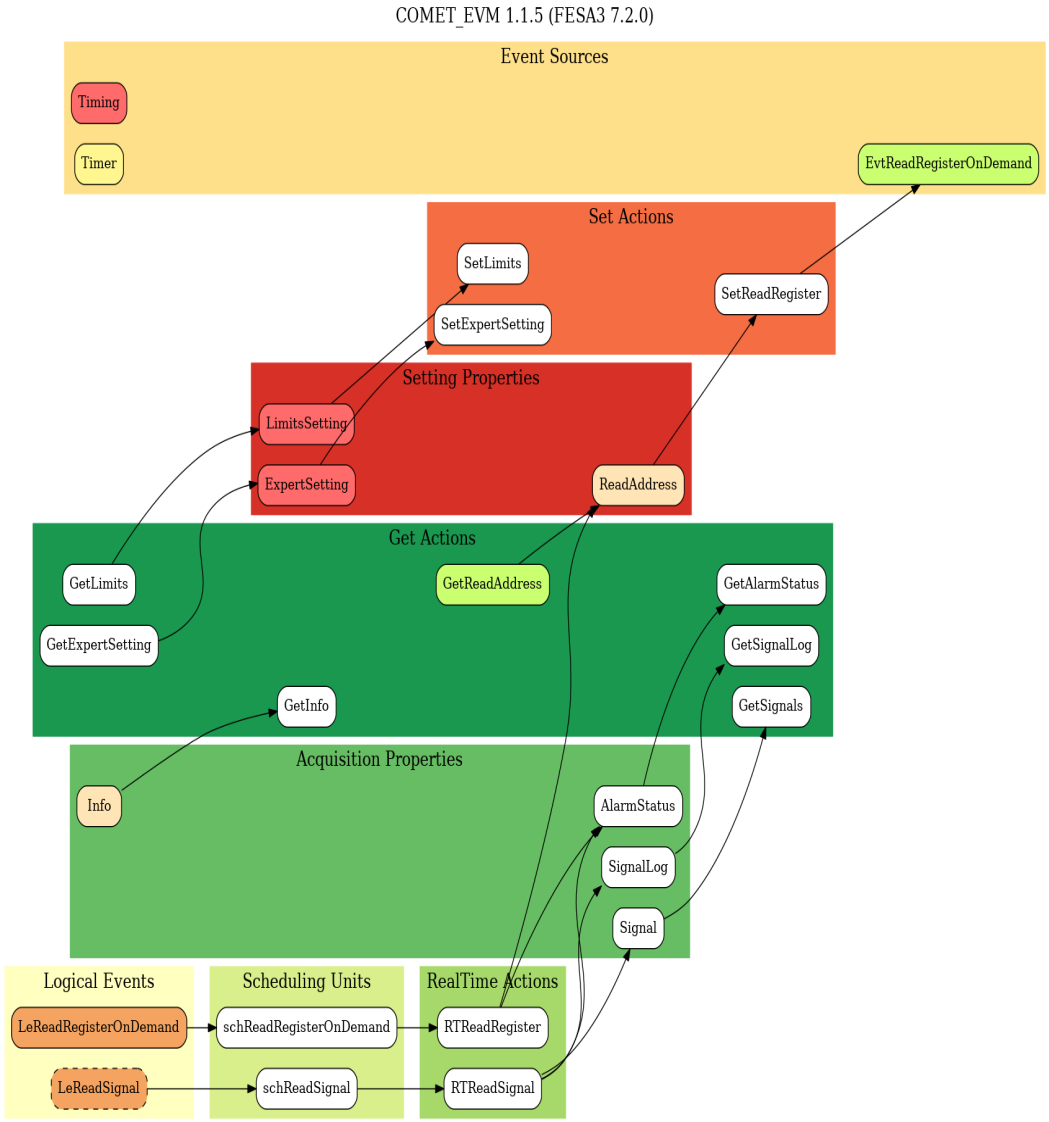

Fig. B.14 Comet_EVM FESA class properties relationships.

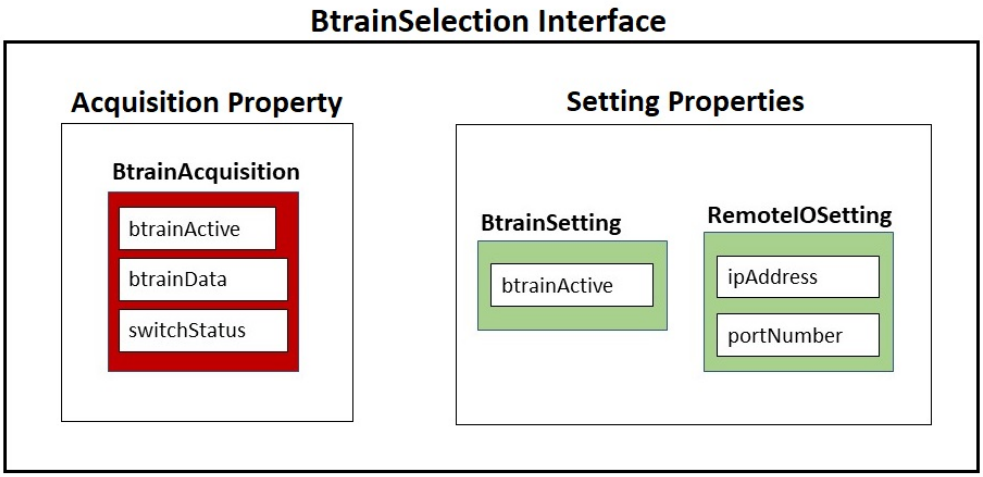

Fig. B.15 BtrainSelection FESA class interface: Acquisition and Setting Properties.

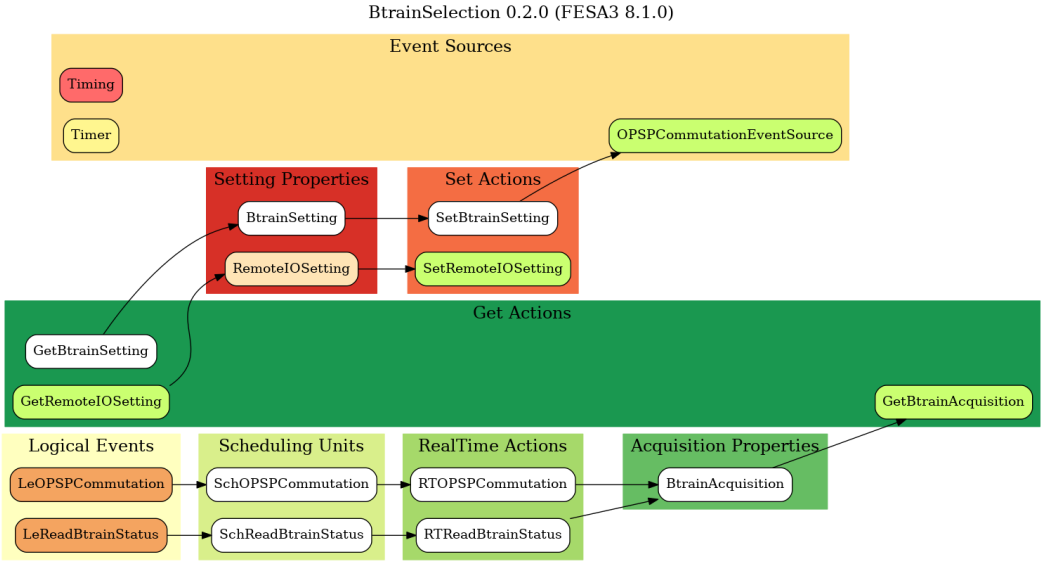

Fig. B.16 BtrainSelection FESA class properties relationships.

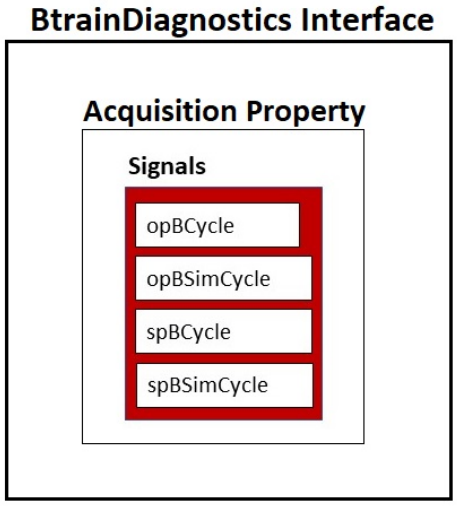

Fig. B.17 BtrainDiagnostics FESA class interface.

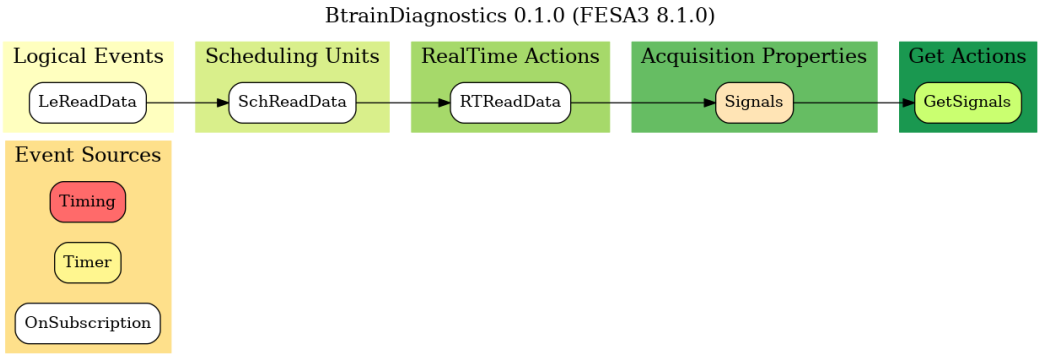

Fig. B.18 BtrainDiagnostics FESA class properties relationships.

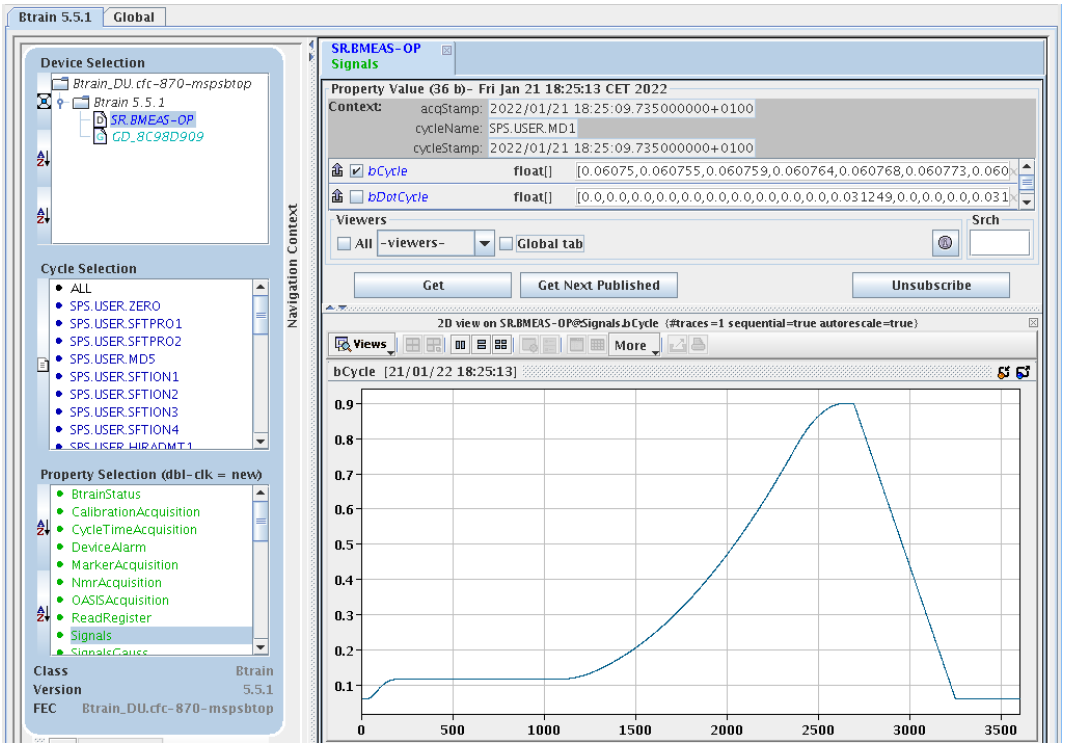

Fig. B.19 Example of Btrain FESA class property shown by means of the FESA Navigator. In this particular case, the measured magnetic cycle for the SPS operational system is shown.