Università degli Studi di Ferrara

DOTTORATO DI RICERCA IN
SCIENZE DELL'INGEGNERIA

CICLO XXXV

COORDINATORE Prof. Trillo Stefano

Explainable Deep Learning

Settore scientifico disciplinare ING-INF/05

Dottorando

Dott. Michele Fraccaroli

Tutori

Prof.ssa Evelina Lamma
Prof. Fabrizio Riguzzi

Anni 2019/2022

Abstract

The great success that Machine and Deep Learning has achieved in areas that are strategic for our society such as industry, defence, medicine, etc., has led more and more realities to invest and explore the use of this technology. Machine Learning and Deep Learning algorithms and learned models can now be found in almost every area of our lives. From phones to smart home appliances, to the cars we drive. So it can be said that this pervasive technology is now in touch with our lives, and therefore we have to deal with it. This is why eXplainable Artificial Intelligence or XAI was born, one of the research trends that are currently in vogue in the field of Deep Learning and Artificial Intelligence. The idea behind this line of research is to make and/or design the new Deep Learning algorithms so that they are interpretable and comprehensible to humans. This necessity is due precisely to the fact that neural networks, the mathematical model underlying Deep Learning, act like a black box, making the internal reasoning they carry out to reach a decision incomprehensible and untrustable to humans. As we are delegating more and more important decisions to these mathematical models, it is very important to be able to understand the motivations that lead these models to make certain decisions. This is because we have integrated them into the most delicate processes of our society, such as medical diagnosis, autonomous driving or legal processes.

The work presented in this thesis consists in studying and testing Deep Learning algorithms integrated with symbolic Artificial Intelligence techniques. This integration has a twofold purpose: to make the models more powerful, enabling them to carry out reasoning or constraining their behaviour in complex situations, and to make them interpretable. The thesis focuses on two macro topics: the explanations obtained through neuro-symbolic integration and the exploitation of explanations to make the Deep Learning algorithms more capable or intelligent. The neuro-symbolic integration was addressed twice, by experimenting with the integration of symbolic algorithms with neural networks. A first approach was to create a system to guide the training of the networks themselves in order to find the best combination of hyper-parameters to automate the design of these networks. This is done by integrating neural

networks with Probabilistic Logic Programming (PLP). This integration makes it possible to exploit probabilistic rules tuned by the behaviour of the networks during the training phase or inherited from the experience of experts in the field. These rules are triggered when a problem occurs during network training. This generates an explanation of what was done to improve the training once a particular issue was identified. A second approach was to make probabilistic logic systems cooperate with neural networks for medical diagnosis on heterogeneous data sources. The second topic addressed in this thesis concerns the exploitation of explanations. In particular, the explanations one can obtain from neural networks are used in order to create attention modules that help in constraining and improving the performance of neural networks.

All works developed during the PhD and described in this thesis have led to the publications listed in Chapter 14.2.

Sommario

Il grande successo che il Deep Learning ha ottenuto in ambiti strategici per la nostra società quali l'industria, la difesa, la medicina etc., ha portanto sempre più realtà a investire ed esplorare l'utilizzo di questa tecnologia. Ormai si possono trovare algoritmi di Machine Learning e Deep Learning quasi in ogni ambito della nostra vita. Dai telefoni, agli elettrodomestici intelligenti fino ai veicoli che guidiamo. Quindi si può dire che questa tecnologia pervarsiva è ormai a contatto con le nostre vite e quindi dobbiamo confrontarci con essa. Da questo nasce l'eXplainable Artificial Intelligence o XAI, uno degli ambiti di ricerca che vanno per la maggiore al giorno d'oggi in ambito di Deep Learning e di Intelligenza Artificiale. Il concetto alla base di questo filone di ricerca è quello di rendere e/o progettare i nuovi algoritmi di Deep Learning in modo che siano affidabili, interpretabili e comprensibili all'uomo. Questa necessità è dovuta proprio al fatto che le reti neurali, modello matematico che sta alla base del Deep Learning, agiscono come una scatola nera, rendendo incomprensibile all'uomo il ragionamento interno che compiono per giungere ad una decisione. Dato che stiamo delegando a questi modelli matematici decisioni sempre più importanti, integrandole nei processi più delicati della nostra società quali, ad esempio, la diagnosi medica, la guida autonoma o i processi di legge, è molto importante riuscire a comprendere le motivazioni che portano questi modelli a produrre determinati risultati.

Il lavoro presentato in questa tesi consiste proprio nello studio e nella sperimentazione di algoritmi di Deep Learning integrati con tecniche di Intelligenza Artificiale simbolica. Questa integrazione ha un duplice scopo: rendere i modelli più potenti, consentendogli di compiere ragionamenti o vincolandone il comportamento in situazioni complesse, e renderli interpretabili. La tesi affronta due macro argomenti: le spiegazioni ottenute grazie all'integrazione neuro-simbolica e lo sfruttamento delle spiegazione per rendere gli algoritmi di Deep Learning più capaci o intelligenti. Il primo macro argomento si concentra maggiormente sui lavori svolti nello sperimentare l'integrazione di algoritmi simbolici con le reti neurali. Un approccio è stato quelli di creare un sistema per guidare gli addestramenti delle reti stesse in modo da trovare la

migliore combinazione di iper-parametri per automatizzare la progettazione stessa di queste reti. Questo è fatto tramite l'integrazione di reti neurali con la Programmazione Logica Probabilistica (PLP) che consente di sfruttare delle regole probabilistiche indotte dal comportamento delle reti durante la fase di addestramento o ereditate dall'esperienza maturata dagli esperti del settore. Queste regole si innescano allo scatenarsi di un problema che il sistema rileva durate l'addestramento della rete. Questo ci consente di ottenere una spiegazione di cosa è stato fatto per migliorare l'addestramento una volta identificato un determinato problema. Un secondo approccio è stato quello di far cooperare sistemi logico-probabilistici con reti neurali per la diagnosi medica da fonti di dati eterogenee. La seconda tematica affrontata in questa tesi tratta lo sfruttamento delle spiegazioni che possiamo ottenere dalle rete neurali. In particolare, queste spiegazioni sono usate per creare moduli di attenzione che aiutano a vincolare o a guidare le reti neurali portandone ad avere prestazioni migliorate.

Tutti i lavori sviluppati durante il dottorato e descritti in questa tesi hanno portato alle pubblicazioni elencate nel Capitolo 14.2.

Acknowledgements

First of all, I would like to thank my supervisors Evelina Lamma and Fabrizio Riguzzi that guided and helped me through this journey and let me explore many real interesting research topics.

I thank my family, Claudia, Maurizio, and Marcello, for their unconditional support in all my activities. I thank also my friends for giving me wonderful moments of fun and care.

I also thank the Emilia-Romagna region for my PhD scholarship funded under POR FSE 2014-2020 program, the “SUPER: Supercomputing Unified Platform—Emilia-Romagna” project, financed under POR FESR 2014 – 2020 and the "National Group of Computing Science (GNCS-INDAM)".

Special thanks go to my girlfriend Anna who encouraged and supported me in my journey, sharing the darkest and brightest moments with me. Thanks so much for everything.

This work is dedicated to my dear friend Fabio.
See you soon my friend.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

AD	Anomaly Detection
AE	Autoencoder
AI	Artificial Intelligence
ANN	Artificial Neural Network
AnoSeg	Anomaly Segmentation Network
AOI	Area of Interest
AUC	Area Under the Curve
AutoML	Automated Machine Learning
BO	Bayesian Optimization
CAM	Class Activation Mapping
CART	Classification And Regression Tree
CEOD	Cross Entropy Overlap Dispance
CNN	Convolutional Neural Network
CT	Computed Tomography
CV	Computer Vision
DARTS	Differentiable Architecture Search
DFN	Deep Feedforward Network

DICOM Digital Imaging and COmmunications in Medicine

DL Deep Learning

DNN Deep Neural Network

DT Decision Tree

ECG Electrocardiogram

EI Expected Improvement

ELU Exponential Linear Unit

EM Expectation–maximization

ENAS Efficient Neural Architecture Search

FLOPS Floating Point Operations per Second

FOL First-order Logic

GAN Generative Adversarial Network

GP Gaussian Process

Grad-CAM Gradient-weighted Class Activation Mapping

HP Hyper-Parameter

HPLP Hierarchical Probabilistic Logic Programming

HPL program Hierarchical Probabilistic Logic program

HPO Hyper-Parameters Optimization

HU Hounsfield

ICU Intensive Care Unit

ILP Inductive Logic Programming

KBANN Knowledge-Based Artificial Neural Networks

KNN K-Nearest Neighbours

KG Knowledge Graph

LFI Learning From Interpretation

LIME Local Interpretable Model-Agnostic Explanations

LRNN Lifted Relational Neural Networks

LRP Layer-wise Relevance Propagation

LTN Logic Tensor Network

Mask R-CNN Mask Region Based Convolutional Neural Networks

MDA Mean Decrease Accuracy

MDI Mean Decrease in Impurity

ML Machine Learning

MLP Multilayer Perceptron

NAS Neural Architecture Search

NeSy Neuro-Symbolic Integration

NNI Neural Network Intelligence

OD Overlap Distance

PLP Probabilistic Logic Programming

PR Precision-Recall

ReLU Rectified Linear Units

RF Random Forest

RNN Recurrent Neural Network

ROC Receiver Operating Characteristic

SDD	Sentential Decision Diagram
SELU	Scaled Exponential Linear Unit
SHAP	SHapley Additive exPlanations
SLEAHP	Structure LEArning of Hierarchical Probabilistic logic programming
SLF	Semantic Loss Function
SMBO	Sequential Model-Based Optimization
SMOTE	Synthetic Minority Over sampling Technique
SOFA	Sequential Organ Failure Assessment
SPOS	Single Path One-Shot
StarAI	Statistical Relational Artificial Intelligence
STR	Symbolic Tuning Rule
SVM	Support-Vector Machine
TA	Tuning Action
VAE	Variational Autoencoder
XAI	eXplainable Artificial Intelligence
XGB	eXtreme Gradient Boosting

Part I

Introduction

Chapter 1

Motivation and Goal of the Thesis

1.1 Motivation

To benefit from the amount of data produced by people, industries and more generally by companies, in recent years, Artificial Intelligence (AI) technologies have been more and more developed. In carrying out some tasks, these technologies have shown that they have arrived at the *superhuman* level. Just think of the classification of temporal data or images. Given the pervasiveness of these techniques, we can find uses for neural networks or more general models of Artificial Intelligence also in the most strategic sectors of our society such as the medical, military, manufacturing, economic and financial industries. Precisely for this reason, we can no longer trust these intelligent agents as if they were oracles. We need to understand what drives them to make certain decisions. This is used to evaluate whether their behaviour is guided by the correct motivations or, in the most optimistic case, to be able to learn from them.

Neuro-Symbolic Integration is a research field that tries to achieve the explainability task of AI by integrating neural networks with symbolic and/or logical algorithms. This integration aims to equip deep neural networks (DNNs) with methodologies and systems capable of being interpretable by humans. By exploiting the natural explainability of logical and symbolic algorithms, it is possible to integrate them within the DNN training process or structure to create systems capable of justifying and generating increasingly interpretable

answers. This integration also leads to the creation of more resilient, robust and more performing systems.

It is possible to integrate these two worlds of AI in two ways: through designing an AI model to be transparent or by post-hoc explainability methods. The *transparent AI models* are understandable by themselves (i.e., Linear Regression, Decision Trees, K-Nearest Neighbours, etc.). Post-hoc explainability methods, instead, are techniques for retrieving an explanation from AI models that are not transparent. These methods are a set of extensions to be applied to AI models to explain their decisions.

1.2 Goal of the Thesis

The goal of this thesis is to provide new types of Neuro-Symbolic algorithms or a new way to obtain the explainability of Machine Learning models.

First, we propose a new framework to tune the hyper-parameters of a DNN and, at the same time, tune the network architecture. This system, called Symbolic DNN-Tuner, exploits Probabilistic Logic Programming (PLP) and Bayesian Optimization to optimize the networks. This system is equipped with a set of a *probabilistic tuning rules* that are triggered with a certain probability if a problem is identified during the training phase of the networks. The probability of applying these rules is determined by how efficient they have been in optimizing the network in the previous steps of the optimization made by Symbolic DNN-Tuner. These probabilities are refined through a Parameter Learning algorithm called Learning From Interpretation described in Section 5. This new framework is tested on benchmarking datasets and a real-case industrial dataset, demonstrating excellent performance compared to classic Bayesian Optimization (state-of-art in optimizing computationally expensive functions). Subsequently, Symbolic DNN-Tuner has been modified to design and optimize DNNs for edge devices with power and computational constraints. Also in this use case, Symbolic DNN-Tuner has proven to be better than state-of-the-art Neural Architecture Search (NAS) algorithms.

The second proposed system is a multimodal learning algorithm that integrates Random Forest and Decision Trees with DNNs through Hierarch-

ical Probabilistic Logic Programming (HPLP). HPLP is an extension of Liftable PLP, which is an AI approach for integrating symbolic and sub-symbolic systems. This system, called Neural HPLP, exploits these different Machine Learning methods to learn from heterogeneous data sources like structured and unstructured data, and merge them with HPLP. This system has been tested in the medical field obtaining good results in terms of prediction/diagnosis accuracy and interpretability. In detail, Neural HPLP exploits a pipeline of Random Forest and Decision Trees to extract knowledge from clinical data and DNNs to extract relevant patterns from clinical images (i.e., CT scans or X-Ray). This knowledge, thanks to the HPLP program, is exploited to perform a final prediction and generate an interpretation for the decision generated.

The last proposed contribution achieved is a new loss function that works like an attention module. This new loss, called Cross Entropy Overlap Distance (CEOD), is used to force a DNN to focus on relevant parts of an image. To do that, we identify the most important pixels for classification according to the DNN by exploiting an explainability algorithm called Gradient-weighted Class Activation Mapping (Grad-CAM). After that, we calculate how much these pixels overlap with a mask generated with a segmentation network that is provided, for each image, during the training phase. The computed overlap value is added to the loss of the network, to force the network to recognize the most important pixels, only within the area marked by the masks.

Chapter 2

Structure of the Thesis

This thesis is divided into five parts: after the introduction (first part), the second part introduces the background concepts of Deep Learning, Probabilistic Logic Programming and Neuro-Symbolic Integration. The third part discusses the Explainable Deep Learning algorithms and systems created during the PhD programme and the fourth part introduces the applications of the methods described in part three and the results obtained in benchmarking and real-cases datasets. The fifth part concludes this dissertation with several possible directions and future work.

The Deep Learning chapter of the second part summarizes this publication:

- Riccardo Zese, Elena Bellodi, Michele Fraccaroli, Fabrizio Riguzzi, and Evelina Lamma. Neural Networks and Deep Learning Fundamentals, pages 23–42. Springer International Publishing, Cham, 2022.

The third part is described in an exhaustive and in-depth way in these publications:

- Michele Fraccaroli, Evelina Lamma, and Fabrizio Riguzzi. Exploiting Parameters Learning for Hyper-parameters Optimization in Deep Neural Networks. Proceedings of the 36th International Conference on Logic Programming, 2075-2180, 10.4204/EPTCS.364, 2022.

- Michele Fraccaroli, Evelina Lamma, and Fabrizio Riguzzi. Symbolic dnn-tuner. Machine Learning, 111(2):625–650, 2022.[1]

[1]Reproduced with permission from Springer Nature

- Michele Fraccaroli, Evelina Lamma, and Fabrizio Riguzzi. Symbolic dnn-tuner: a python and problog-based system for optimizing deep neural networks hyperparameters. SoftwareX, 17:100957, 2022.

- Michele Fraccaroli, Evelina Lamma, and Fabrizio Riguzzi. Automatic setting of DNN hyper-parameters by mixing Bayesian Optimization and tuning rules. In Giuseppe Nicosia, Varun Ojha, Emanuele La Malfa, Giorgio Jansen, Vincenzo Sciacca, Panos Pardalos, Giovanni Giuffrida, and Renato Umeton, editors, Machine Learning, Optimization, and Data Science, 6th International Conference, LOD 2020, Siena, Italy, July 19–23, 2020, Revised Selected Papers, Part I, volume 12565 of Lecture Notes in Computer Science, pages 477–488, Cham, 2020. © Springer, Springer International Publishing.

The experiments and application of the fourth part are described more extensively also in these publications:

- Michele Fraccaroli, Giulia Mazzucchelli, and Alice Bizzarri. Machine learning techniques for extracting relevant features from clinical data for COVID-19 mortality prediction. In 2021 Symposium on Computers and Communications (ISCC): 26th IEEE Symposium on Computers and Communications - Workshop on ICT Solutions for eHealth (ICTS4eHealth) (ICTS4eHealth2021), pages 1–7, Athens, Greece, September 2021.

- Michele Fraccaroli, Alice Bizzarri, Paolo Casellati, and Evelina Lamma. Cross entropy overlap distance. Accepted and Presented at ITAL-IA 2022, workshop on AI for Industry, feb 2022.[2]

- Fadja, A.N., Fraccaroli, M., Bizzarri, A. et al. Neural-Symbolic Ensemble Learning for early-stage prediction of critical state of Covid-19 patients. Med Biol Eng Comput (2022). https://doi.org/10.1007/s11517-022-02674-1.[2]

[2]Reproduced with permission from Springer Nature

2.1 How to Read this Thesis

We tried to make this work as understandable and logically smooth as possible. The main topic of this work is the *explainability* of Deep Learning models. Following the Figure 2.1, Chapters 3 to 5 compose the background. Chapter 6 and 7 are important for correctly understanding the part of the thesis that compose the right branch of Figure 2.1 (Chapter 8). Chapters 9 and 13 compose the second approach to explainability for AI systems.

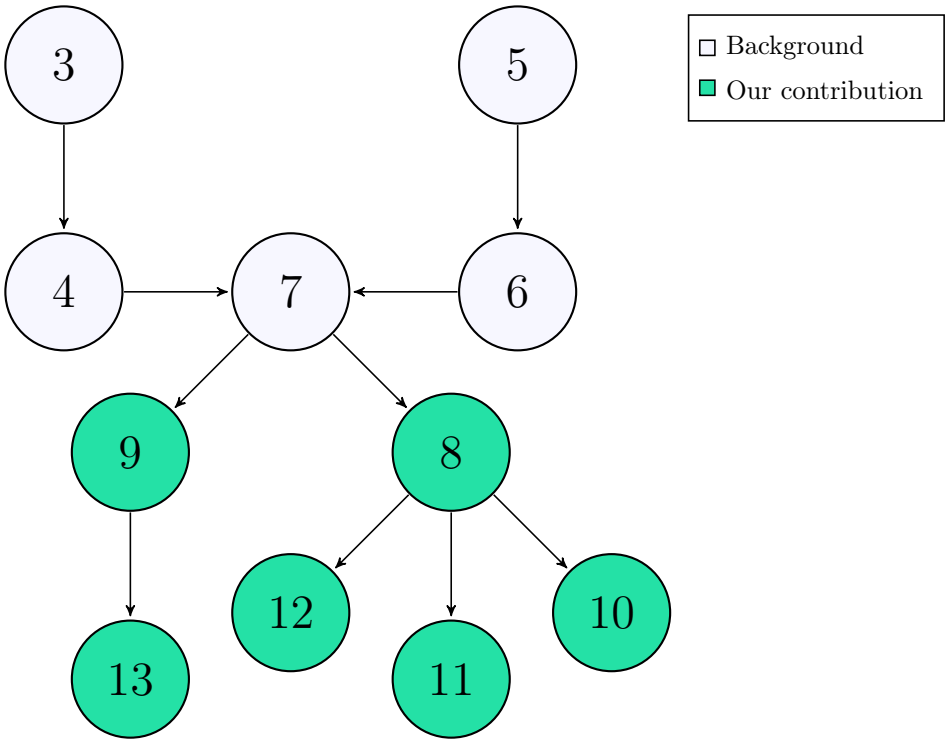


Figure 2.1: Chapter dependency graph. For instance, to understand Chapter 9, you have to read Chapter 8.

5

Part II

Background

Chapter 3

Deep Learning

Deep Learning (DL) is a Machine Learning (ML) technique inspired from neuroscience. It uses a computational model called Artificial Neural Network (ANN) which is inspired by the real biological neural network [3]. DL involves computational models, called neural network that is composed of multiple hierarchical levels to be able to learn data representations with multiple levels of abstraction where, each level, learns concepts based on the previous levels. This stacking of layers and units (neurons) per layer can allows neural networks to represent functions of increasing complexity.

DL architectures such as Deep Feedforward Network (DFN), Recurrent Neural Network (RNN) and CNN have been applied to fields including computer vision, speech and audio recognition, natural language processing, social network filtering, bioinformatics and many other sectors.

These systems learn in two main ways: *supervised* and *unsupervised*. *Supervised Learning* is a type of learning that starts from the examples provided during the training phase. Every example is composed by an input object and a desiderated output value (label). This value is what we want to get form the network.

In *Supervised Learning*, we need to supply the network with the right amount of the examples. With these data, the network builds an *interpretation* of the input, that enables it to correctly classify the given examples. In other words, it analyses the given data and produces an inferred function which can be used for mapping new examples.

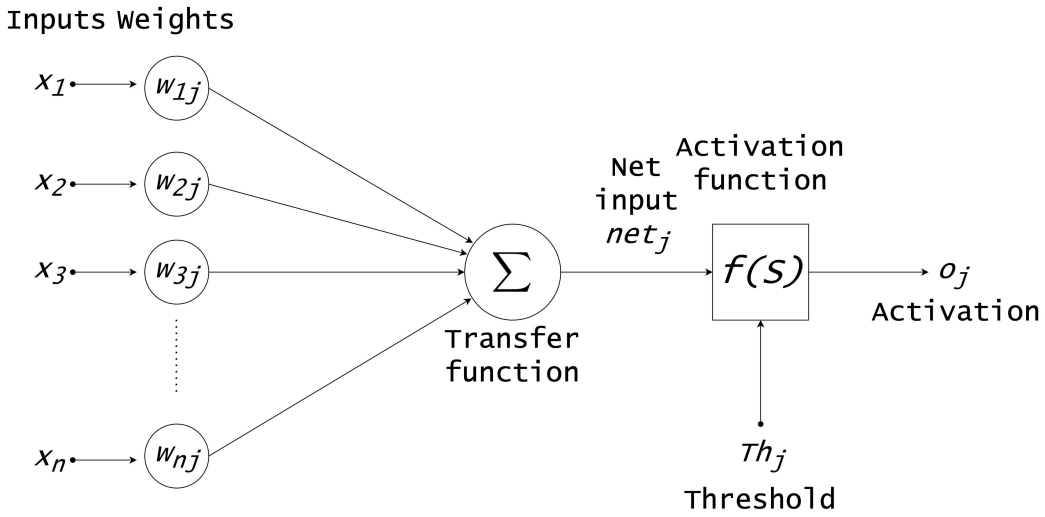


Figure 3.1: Representation of a structure of an artificial neuron.

Unsupervised Learning is like a Supervised Learning but, the examples provided to the network do not have an output label. The network uses the examples for learning common features for trying to reason and predict future input.

The core example of deep learning model is the *Multilayer Perceptron (MLP)* [4].This is an extension of the *perceptron* [5]. The MLP consists in at least three layers: input layer, hidden layer and output layer. Each node of this network (excluding the input) is a *neuron* that use a non-linear activation function. The activation function define the output of a neuron given a set (or single) inputs. This output represent the input of the next neurons and so on. A representation of a structure of an artificial neuron can be seen in Figure 3.1.

3.1 Deep Feed-forward Networks

DFN, also called MLP, have many intermediate layers between the input and output's layers. A feed-forward network defines a mapping $y = f(X, \theta)$ where X is the input, Y is the category or output and θ represents the parameter vector that the network has to learn [4]. Then, the objective is to approximate a function $f : \mathbb{R}^{n \times m} \rightarrow \mathbb{R}^k$ by means of other functions such that

$f(X) = f_n(f_{n-1}(...(f_1(f_0(X)))))$, where X is the input tensor (a tensor is a n-dimensional vector with $n \in \mathbb{N}$) of size $n \times m$. Then, the MLP is a neural network with n layers, each representing a function f_j with j the number of the layer. The information flows through the network form the input X to the output Y without feedback. DFN are extremely important for deep learning applications. since they are the common architecture for popular network such as RNN [6] or CNN. The CNN is the fundamental basic network used in the techniques explained in this work.

3.2 Convolutional Neural Network

CNN is the base network for image analysis. *Convolution* is a mathematical operation between two functions of a variable, it's the integration of products between the first function and the second translated of a certain value:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(a)g(t-a)da \tag{3.1}$$

$$(f * g)[n] = \sum_{a=-\infty}^{\infty} f(a)g(n-a) \tag{3.2}$$

Equation 3.1 defines convolution between continuous function while Equation 3.2 adapt it to discrete data. Moving from mathematics to CNN, the function f is the *input*, the function g is the *kernel* and the output is called *feature map*.

CNN are inspired by biological processes [7], indeed, the connection between neurons is similar to that of the animal visual cortex: every neuron responds to a stimuli of a specific region of the visual field, the receptive field. Receptive fields partially overlap and cover the entire visual field. The design of CNN is composed by an input, an output and a multiple hidden layer. These layers typically consist in a series of:

- Convolutional layer

- Activation function
- Pooling layer

The **convolutional layer** is parametrized by a tensor with shape *(image width)* x *(image height)* x *(image depth)* (or channels). Image depth is the number of colour channels (for a colour images channels is 3, red, green and blue). Then, give an input matrix, eg, an image, this layer perform a filtering with a filter called *kernel* and the region of the matrix (or image) covered by a kernel is called **receptive field**. The filtering operation consists of a multiplication of the filter values with the pixel values of the input tensor, generating a unique number in output that represents the single receptive field. This operation is repeated moving the filter over the whole image by a predefined unit. At the end of this operation, we have the *feature map* as output, that is a smaller matrix than the input that represents an extraction of significant features found in the original image. The graphical situation is represented in Figure 3.4. After the convolutional layer, there is a non-linear layer or *activation layer*. The utility of this layer is to introduce a non-linearity. There are different types of activation functions such as: sigmoid or tanh but the best choice for hidden layers is the *Rectified Linear Units (ReLU)* [8, 4]. The three principal functions are described by figures 3.2 and 3.3. ReLU and Exponential Linear Unit (ELU) are the best choice because the training time of the network is shorter then other activation functions and still maintains good accuracy. Function ELU, unlike ReLU, tries to make the mean activations closer to zero. ELU can obtain an higher classification accuracy than ReLU [9].

The mathematical functions applied at every input is:

$$y = x^+ = max(0, x) \tag{3.3}$$

for ReLU and

$$y = \begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & \text{otherwise} \end{cases} \tag{3.4}$$

for ELU where the parameter α is a hyper-parameter to be tuned.

After the activation layer, there is the *pooling layer*. Pooling can be global

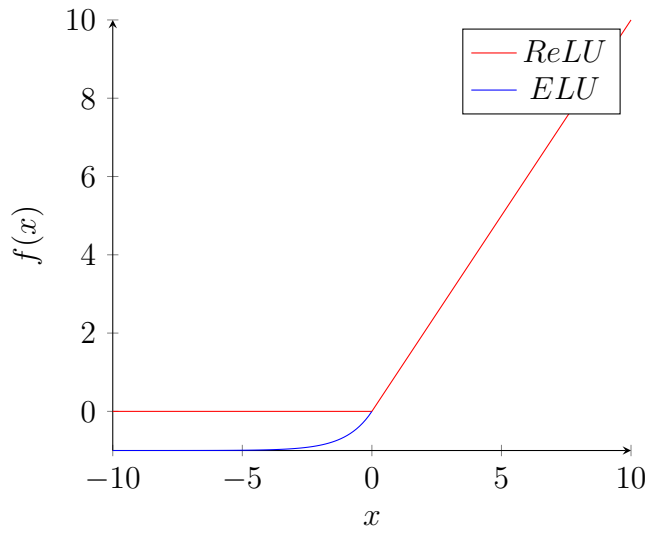


Figure 3.2: ReLU and ELU function.

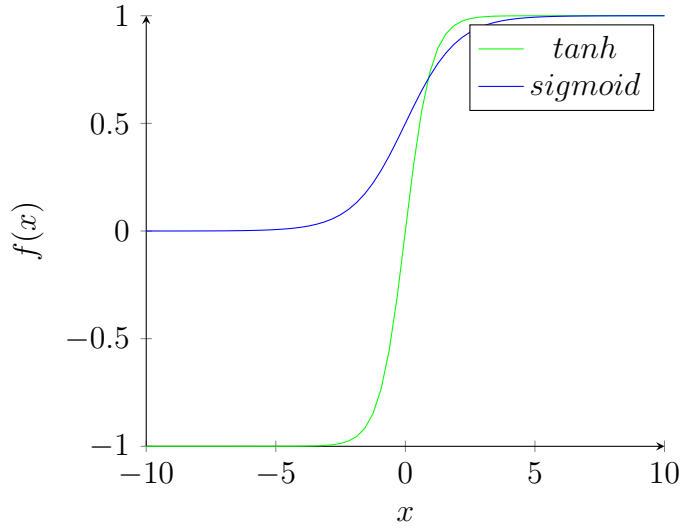


Figure 3.3: Sigmoid and tanh functions.

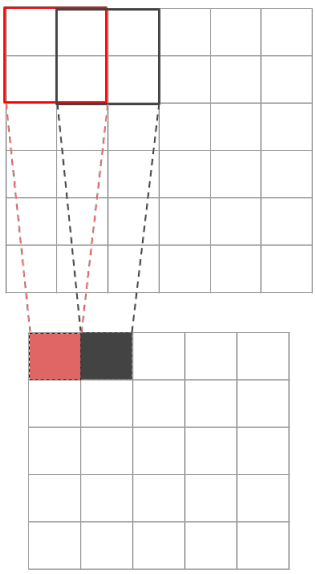


Figure 3.4: Input tensor (top grid) with kernel 2x2 (red square) in the first position and the respective point or result on feature map (bottom grid). The grey square represent the second step of filtering.

or local. This layer reduces the dimensions of the matrix using a kernel and a stride of the same length. The global pooling acts on all the neurons of the previous convolutional layer while the local combine small clusters as can be seen in Figure 3.5. The operation applied by pooling layer can be a max or an average. Max-pooling use the maximum value from each cluster, instead, average pooling use the average value from each cluster. For each filtering, the layer maintain the highest (or the mean) value of parameters thus obtaining a reduced matrix. The second purpose of pooling is to control overfitting, this happens because, for every pooling layer, many values are deleted, keeping only the most significant.

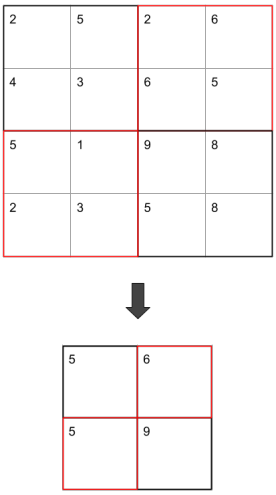


Figure 3.5: This is an example of max-pooling. The output of convolutional layer (top grid) is the input of pooling layer with kernel 2x2 and stride 2 and the bottom grid is the result of max-pooling.

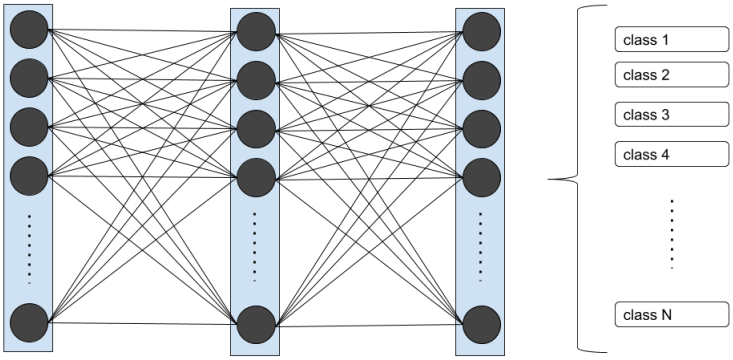


Figure 3.6: Some layers of fully connected neurons terminated with classification.

The last layer and also the output layer of this network is the **fully connected** layer. This layer connect every neuron of a layer to every neuron of layer above. This is used for classification and returns the effective output of the whole CNN as shown in Figure 3.6.

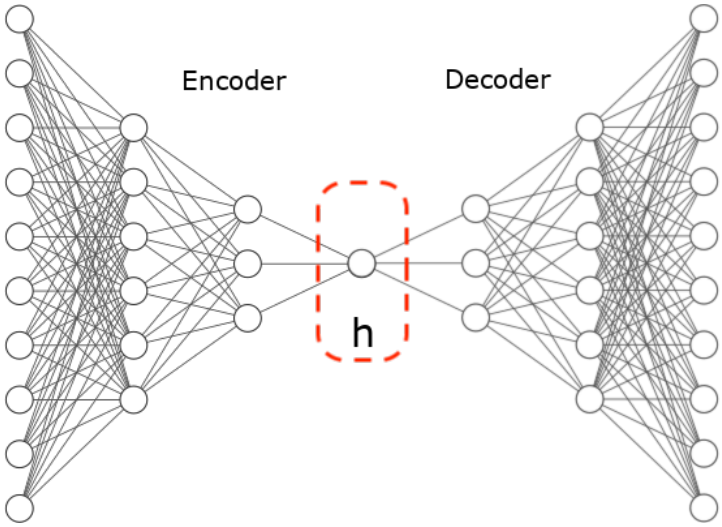


Figure 3.7: Representation of AE.

3.3 Autoencoders

An AE [4] is a type of neural network that is trained to try to replicate its input to its output. This network can be decomposed into two parts: an encoder (E) that compresses the input into latent space h and a decoder (D) that produces a reconstruction of the input starting from the latent space h as shown in Figure 3.7. Then, if x is the input the given to the network, the E can be defined as $h = f(x)$ where f is the encoder function, D can be defined as $r = g(h)$ where g is the decoder function and r is the reconstruction of x. Then, an AE learns to set $g(h) \rightarrow g(f(x)) = x$.

The potential of AE lies in the input copying task and in the possibility of constraining h to be smaller than x (in this case, we talk about *undercomplete* AE). Learning an undercomplete representation forces the network to identify the most important features of the input data. This process can be performed minimizing the function $l(x, g(f(x)))$ where l is the loss function that penalizes the network when $g(f(x))$ is distant from x.

In addition to the mentioned AE, there are other types and variations of AE: Sparse, Denoising and Variational AE. A Sparse AE is an AE that exploits a sparse penalty on the latent space as a training criterion. This type of AE is used to perform tasks like classification. Denoising AE is an AE used to remove corruption (or noise) from the input data. Instead of applying penalty like the

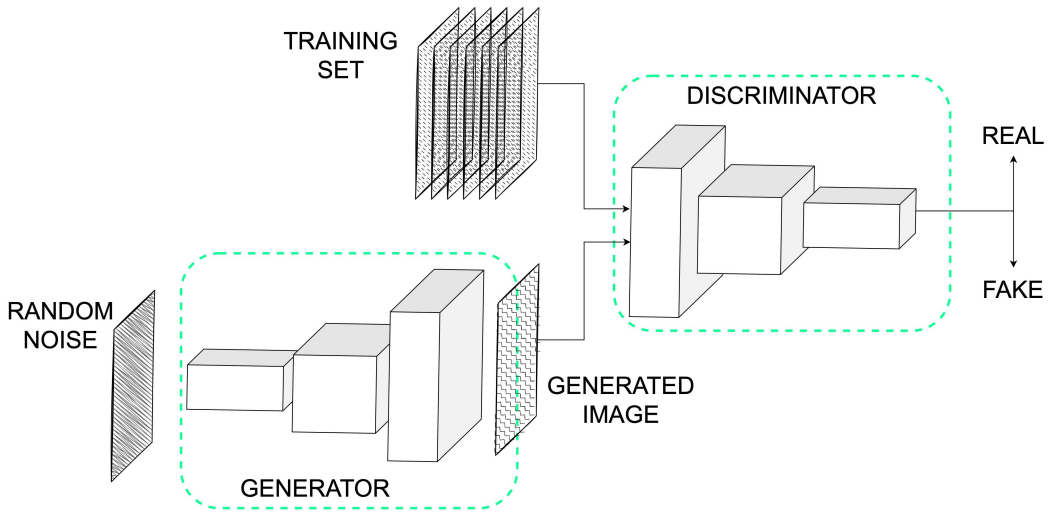


Figure 3.8: Representation of GAN.

Sparse AE, in this case, the network minimizes the function $l(x, g(f(\tilde{x})))$ where $\tilde{x}$ is a noisy version of x. A Variational Autoencoder (VAE) is a generative network, that is a type of network which, starting from a compressed vector or from random noise, is able to generate a realistically truthful data. VAE it is related to AE because of its architectural affinity. VAE differs from standard AE in the latent space representation. Instead of encoding an input as a single embedding, VAE encodes it as a distribution over the latent space. Then, the encoder of VAE defines an approximate posterior distribution $q(h|x)$, which takes in input x and outputs a set of parameters to describe a conditional distribution of h (typically μ and σ). The decoder defines the conditional distribution over $p(x|h)$. It takes in input h and outputs the parameters for a conditional distribution of the observation. There are other many types of AE. Please, see [4] for more detailed descriptions.

3.4 Generative Adversarial Networks

GANs is a generative modelling approach based on different generator neural networks. The foundation of GANs is game theory. It includes a *generator* that competes against the *discriminator* network. as shown in Figure 3.8.

The generator produces samples starting from random noise. The discriminator tries to classify if the generated image is a real or fake image. Then, the

learning process of a GAN can be formulated as a zero-sum game. If ϕ^g and ϕ^d are the payoffs of generator and discriminator respectively, during training, each network tries to minimize its payoff at convergence. The payoff for a player is a value that expresses the evaluation of the result obtained by that player, following the choices made by all the players involved. In this way, the generator learns how to fool the discriminator, by creating more and more realistic images. Instead, the discriminator learns how to distinguish real and fake samples. At convergence, the generator is able to create samples that are indistinguishable from the real data.

Chapter 4

Computer Vision

Computer Vision (CV) is a field of computer science that focuses on creating systems and algorithms that can process and analyse images or videos. This field includes many techniques typically based on classical image processing and Artificial Intelligence (AI) algorithms. In the era of Industry 4.0 and thanks to the availability of enormous amounts of data, much research has focused on the part of the CV that involves AI techniques. Thanks to this research advance, the applications of CV range from tasks such as an industrial vision for anomaly detection and inspection to medical diagnosis and analyses. The techniques involved in these tasks include the methodologies explained in Chapter 3, also extending them to tackle more complicated tasks such as object detection or segmentation by exploiting CNNs or advanced anomaly detection by integrating GAN and AE. Some applications that take benefit from CV are automatic industrial inspection, visual surveillance, navigation systems, autonomous vehicles, robotics, medical diagnosis, etc.

In specific, in the medical environment, one of the most prominent applications of CV techniques based on AI systems is medical image processing. This task consists of the extraction of information from patients' unstructured data like Computed Tomography (CT) or X-ray scans for diagnostic purpose [10, 11, 12]. An example of a successful application of these techniques is the detection of tumours [13, 14] or, concerning recent events, for analysing or detecting problems related to the Covid-19 infection [15, 16].

In the industrial environment, the applications of CV are mainly focused in

supporting production processes or quality control. One example of a quality control task performed by CV is the automatic inspection of the final product at the end of the production line to find defects or anomalies. A real example can be wafer inspection [17] in which every single wafer is being measured and inspected for inaccuracies or defects to prevent a computer chip from coming to market in an unusable manner.

In the military industry, CV is used for many tasks such as enemy soldiers detection, and missile or vehicle guidance. More modern applications in this field that involves AI are exploiting image sensors that provide a rich set of information for *battlefield awareness* [18].

All these applications highlight the importance of CV based on AI and specifically, based on DL. But the pervasive use of DL has raised a problem. ANNs are considered a *black box* due to their non-linear structure: this means that despite the good performance of the ANNs, we can't understand the internal reasoning of these models. This peculiarity can bring different problems, mostly in fields like medicine and autonomous driving.

Chapter 5

Probabilistic Logic Programming

Probabilistic Logic Programming (PLP) is a tool for reasoning on uncertain relational domains that is gaining popularity in Statistical Relational Artificial Intelligence (StarAI) due to its expressiveness and intuitiveness. PLP has been successfully applied to a variety of fields, such as natural language processing [19, 20, 21], bio-informatics [22, 23, 24], link prediction in social networks [25], entity resolution [26] and model checking [27].

Various approaches have been proposed for combining logic programming with probability theory. One of the most known language for PLP is ProbLog [23]. ProbLog is a probabilistic extension of Prolog and it is a PLP language under the distribution semantic [28]. A program that adopts a semantics of this type defines a probability distribution over normal logic programs called *worlds*. To define the probability of a query, this distribution is extended to a joint distribution of the query and the *worlds* and the probability of the query is obtained from the joint distribution by marginalization [29].

A ProbLog program consists of a set of clauses. Every clause c_i is labelled with the probability p_i. Listing 5.1 shows an example of ProbLog code.

Listing 5.1: ProbLog example

```
% Probabilistic facts:
0.5::heads1.
0.6::heads2.
% Rules:
twoHeads :- heads1, heads2.
```

```
% Queries:
query(heads1).
query(heads2).
query(twoHeads).
```

A ProbLog program $P = \{p_1 :: c_1, p_2 :: c_2, .., p_i :: c_m\}$ defines a probability distribution over logic program $L = \{c_1, c_2, ..., c_m\}$ and the aim of inference in ProbLog is to calculate the probability that the query will succeed [23]. With reference to Listing 5.1, the three queries return the probabilities of 0.5, 0.6 and 0.3 for `heads`, `heads2` and `twoHeads` respectively.

Listing 5.2: Another ProbLog example

```
% Probabilistic facts:
0.8::stress(ann).
0.6::influences(ann,bob).
0.2::influences(bob,carl).
% Rules:
smokes(X) :- stress(X).
smokes(X) :- influences(Y,X), smokes(Y).
% Query:
query(smokes(carl)).
```

Listing 5.2 show another ProbLog code example. In this example, there are three probabilistic facts reporting how people Ann, Bob and Carl are affected by stress or influence another person all weighed by the probabilistic weights at the top of the facts. The rules describe that if a person is stressed, then he smoke. If a people X is influenced by Y and Y smokes, then X smokes. Then, given facts and these rules, if we run the query `smokes(carl)` we obtain the probability of 0.096.

ProbLog inference [30] works by generating all ground instances of clauses in the program the query depends on and transforming the clauses to a propositional formula. After that, the logic formula is compiled into a Sentential Decision Diagram (SDD) [31]. SDDs are a representation language for propositional knowledge bases. Then ProbLog evaluates the SDD bottom-up to calculate the success probability of the given query.

5.1 Parameter Learning

The ProbLog system [32] includes a Parameter Learning algorithm (Learning From Interpretation (LFI)-ProbLog algorithm) [33] that learns the parameters of ProbLog programs from partial interpretations. Generally, one is interested in the maximum likelihood parameters given the training data. Then, given a ProbLog program P containing probabilistic facts with unknown parameters (probabilistic weights) and a set $E = \{\mathcal{I}_1, \ldots, \mathcal{I}_T\}$ of partial interpretations (the training examples), find the value of the parameters $\mathbf{\Pi}$ of P that maximize the likelihood of the examples, i.e., solve:

$$\arg\max_{\mathbf{\Pi}} P(E) = \arg\max_{\mathbf{\Pi}} \prod_{t=1}^{T} P(q(\mathcal{I}_t)) \tag{5.1}$$

where partial interpretation $\mathcal{I}$ can be written like $\mathcal{I} = \langle I_t, I_f \rangle$ where atoms in I_t are true and those in I_f are false. $\mathcal{I}$ can be associated with conjunction $q(\mathcal{I}) = \wedge_{a \in I_t} a \vee \wedge_{a \in I_f} \sim a$. So, given a ProbLog program and a sets of partial interpretations $\mathcal{I}$, the goal is to find the maximum likelihood parameters. Now, we need to pay attention to interpretations (the training examples) when computing $\arg\max_{\mathbf{\Pi}} P(E)$. If we have complete interpretations, the parameters (probabilistic weights) can be computed by relative frequency. If some interpretations in E are partial, instead, an Expectation–maximization (EM) algorithm [34] must be used [33, 29]. For a more complete and treatment of PLP and PLP Parameter Learning, see [29].

Chapter 6

Neuro-Symbolic Integration

Neuro-Symbolic Integration (NeSy) is a research field that aims to combine the two most influential aspects of AI: the ability to percept and learn from data and environment and the capacity to reason on the learned concepts [35, 36, 37]. NeSy has three purposes: to extend the capabilities of AI as mentioned above, to create a system capable of perceiving the outside world and making complex reasoning about what is perceived and to provide a possible explanation from these complex systems. This research field boasts several works carried out over the years but in these years of the explosion of deep learning, the need to obtain self-explanatory, interpretable and reliable systems becomes ever greater. Then, the main characteristics of NeSy allow the combination of robust learning and efficient inference in neural networks, along with the interpretability offered by symbolic knowledge extraction and reasoning with logical systems [38]. We can refer to the systems that integrate symbolic and sub-symbolic worlds as *hybrid*. These hybrid methods can be divided into two families, based on how the blending of symbolic and sub-symbolic approaches is realised: integration and combination methodologies.

6.1 Integration methods

In this category, we describe the hybrid systems that combine logic and symbolic knowledge with neural networks. Typically, this integration exploits logic rules for constraining the behaviour of the network via loss extension through regularization terms or other constraints. The networks are then trained as

usual via backpropagation.

One of the earliest works in this sub-filed is Knowledge-Based Artificial Neural Networks (KBANN) [39]. KBANN, starting from a (Prolog-like) knowledge base, aims to create a network that semantically reflects the symbolic knowledge base from which it was created. Another method to constrain the behaviour of DNNs is DNN with Logic Rules [40]. This method allows the neural network to be trained at the same time on data and logic rules. This is made possible by a procedure for translating the symbolic knowledge encoded as symbolic rules into network parameters. This approach exploits two networks: the *teacher* and the *student*. Differently from other approaches of the same category, only the teacher net has a rule-based regularization term on the loss function. Instead, the student net is trained with a balance of emulating the teacher net output and predicting the exact output expected by the labels. Logic Tensor Network (LTN) [41] integrates learning based on Neural Tensor Network [42] with reasoning based on First-order Logic (FOL). Then, data in form of vectors, constraints and relations that apply to a certain subset of vectors, can be specified using FOL. In this way, reasoning on constraints helps to improve learning and vice versa. The logic formula is used in LTN to build the loss function. This aims to train the network for approximating the truth value of the formula given as input. This is done by exploring the best possible representation for symbolic constructs in a vector space so that the satisfiability of the network is as close to 1. LTN is an example of explainability by design where the logic is directly encoded into the sub-symbolic model and is exploited for improving it. An extension of LTN is Lyrics [43]. Lyrics improves the way symbolic knowledge is declaratively enforced while training the sub-symbolic part of the system. An ulterior interesting approach for bridging DNNs and symbolic constraints is Semantic Loss Function (SLF) [44]. The idea is to constrain the training of the network via a propositional logic formula that is encoded as part of the loss function. This formula is composed of boolean variables that represent the input and output of neurons of the network to be constrained. Slightly changing approach, CILP++ [45] integrates Inductive Logic Programming (ILP) via bottom clause propositionalisation and neural networks. ILP is used to find new rules starting from the input dataset. CILP++ exploits neural nets to make ILP faster and pro-

positionalisation to make the construction of neural networks out of arbitrary logic theories possible. Then, the first step is to use propositionalisation to convert the data (clauses) into real vectors and background knowledge into an MLP. In this way, the structure of the network reflects the rules into the background knowledge and the first (input) layer contains a neuron for each possible atom used in the knowledge base. Thanks to the fact that ILP works on positive and negative examples, in CILP++ the network is trained so that the weights of the network are optimized to select the atoms of the rule to be induced. Strictly correlated to ILP there is Differentiable ILP (δILP) [46]. δILP aims to mimic logic deduction on definite clauses with neural networks using forward chaining. Another possible approach is that proposed by Luc De Raedt et al. [47] called DeepProbLog, which exploits PLP instead of ILP. DeepProbLog is based on ProbLog and exploits neural networks for computing the probabilities of facts used by logic predicated (called *neural predicates*). A DeepProbLog program could include different neural classifiers and is translated into a tensorial computational graph to be trained and optimized with gradient descent. The training simultaneously sets the probabilities of facts and the internal weights of the network. The program is designed for probabilistic inference. One more recent system for blending neural networks and relational learning is Lifted Relational Neural Networks (LRNN) [48]. Similar to DeepProbLog, LRNN exploits sets of weighted formula as a template for building a neural network to be trained. This approach requires the logic knowledge base to be grounded before the network construction.

6.2 Combination methods

The combination methodologies try to synergistically *combine* the symbolic models - like rule sets or trees - with neural networks. In this category, we can identify other two sub-categories: symbolic extraction and injection methods.

The main assumption of *symbolic extraction models* is that the DNN has been trained and reached a good performance. For this sub-category, the two principal techniques used to extract knowledge from the neural predictor are *rule* and *decision tree* extraction.

Rule extraction aims to retrieve a flat list of rules from a neural network if

the form of:

IF $condition^1$ **THEN** $output^1$
ELSE IF $condition^2$ **THEN** $output^2$
...
ELSE $output^n$

Where $condition^i$ can be conjunction (or disjunction) of Boolean predicates, constraints or M-of-N rules. M-of-N rules are a compact way of representing knowledge that has a strong intuitive connection to the structure of neural networks [49]. For rule extraction, we can identify a further division that is: *pedagogical*, *decompositional* and *eclectic* methods. The pedagogical approach views the ANN as an *oracle* in the sense that it does not exploit any internal information of the network to perform the extraction. The decompositional approach is capable of extracting symbolic knowledge from a network by looking at the internal parts of the neural predictors. Finally, the eclectic methods, combine the two previous methods by analysing the internal structure of the network and extracting rules via training instead of analysing weights [50]. The first pedagogical approach was proposed by Saito et al. [51]. Their method extracts M-of-N rules from any "black-box" classifier. It did not support regression tasks but, despite its limitations, this work has been shown to obtain good results in expert system diagnosis support [51]. ALPA [52] is the first rule extraction system that is fully applicable to any "black-box" predictor with no limitations or constraints. For the decompositional approach, one of the first works is MofN [53]. MofN and other similar methods like RuleNet [54] and the method of Giles et al. [55] impose some strong restrictions that limit the neural networks that they can manage. For example, MofN extracts rules in the form of M-of-N only from the networks obtained from KBANN algorithm described above. The decision tree extractions methods are quite similar to rule extraction methods. The difference lies in the hierarchical structure of the knowledge extracted. One of the main approaches of this sub-category is TREPAN [56]. TREPAN is a pedagogical algorithm for extracting symbolic representations from trained neural networks. It queries the DNN to induce a decision tree that describes the concept represented (learned) by the DNN itself. Following the same line, Krishnan et al. [57] proposed another method for extracting decision trees from neural networks. Differently from

TREPAN, in [57] the internal structure of the network is taken into account in the process of tree generation. Finally, the eclectic approach to knowledge extraction can be described as the combination of the pedagogical and decompositional approaches. For example, the DEDEC [58] system works in two steps: it identifies functional dependencies between the input and output of an ANN by analysing the architecture and the weights. The second step is purely pedagogical. Symbolic learning is performed based on the weights analyses. Another eclectic method is HERETIC [50]. The idea behind this system is to perform a symbolic training at each neuron. So, each node on an ANN generates a decision tree trained with the input-output of the neuron. The output of the trees of the previous layer become the input of the trees of the next layer. After the generation of the trees, each tree is converted into a rule in disjunction normal form.

Knowledge injection is specular to knowledge (rule) extraction. One of the most typical injections is Knowledge Graph (KG) injection. KG represent a knowledge base with entities and relations, the idea of KG injection is to represent this type of information in a continuous vector space to allow a DNN to use this information during training. Then, a KG, is represented in a continuous vector space and, after that, the embeddings of entities and relations are computed by maximizing the plausibility of the observed facts. The main limitation is that these techniques perform the embedding task only based on observed facts. The noteworthy approaches in this field are Kale [59] and Oscar [60]. Kale represents rules as a complex formula modelled by the t-norm fuzzy logic [61]. Embedding thus becomes minimizing an overall loss on both atomic and complex formulas. In this way, the embeddings are learned as compatible with the rules. Instead, Oscar is a pre-training regularization technique for injecting knowledge and ontologies into a DNN. The knowledge is exploited as a regulariser for the network.

Chapter 7

Explainable Artificial Intelligence

eXplainable Artificial Intelligence (XAI) can be defined as an AI that can be understood by humans in the results and the internal reasoning. The theme of XAI is directly in contrast with the actual trend of ML and especially DL. Nowadays, the state-of-the-art of AI is represented by ANNs. These models can be considered as a *black box* where nobody (including the designers themselves) can explain why an ANN arrived at a specific decision. The problem is that these *black box* models are the ones that have given the best results and that have the potential to be applied to the most important and delicate areas of our economy, for example, the medical industry, self-driving vehicles, the judicial and financial sectors. Most of these sectors are particularly sensitive. Just think about the medical diagnosis or the financial sector. In these fields, we cannot use *oracle* systems and simply trust their decision. We need at least transparent and self-explanatory systems capable of justifying what led them to make certain decisions [62, 63].

Starting from this point, we can talk about the two main approaches to the XAI: *transparency* models and post-hoc explainability. [64]. The term *transparency* indicates the human-level understanding of the internal reasoning of an AI model. *Post-hoc explanability* indicates models that can be explained through external XAI techniques.

The *transparency* models are interpretable and explainable by design. There are different levels of transparency:

- Simulatability: it is a level of transparency that indicates the model's

ability to be simulated by human;

- Decomposability: it is another level of transparency that indicates the ability to divide the models into parts and explain each part independently;

- Algorithmic transparency: it is a deeper level of transparency and indicates the possibility of understanding the internal procedure of the model which leads from the inputs to obtaining a decision.

Post-hoc explanation techniques try to extract an explanation from models that are not interpretable by design. These techniques are:

- Text explanations: brings explainability by learning to generate text explanations that help explain the results from the model;

- Visual explanations: aims to visualize the model's behaviour. There are many approaches for this scope, for example, dimensionality reduction or heatmap generation algorithms;

- Local explanations: segments the solution space and gives explanations to less complex solution subspaces that are relevant for the whole model;

- Explanations by example: extracts examples that are related to the result generated by the model. This is done to try to get a better understanding of the model itself;

- Explanation by simplification: a new system is built based on the model to be explained. This new simplified model tries to resemble the main model by reducing its complexity and consequently facilitating its interpretability;

- Feature relevance explanation: explains the model by computing the relevance score for its variables. These scores quantify the sensitivity a feature has upon the output of the model.

7.1 Transparent AI Models

A model can be considered transparent if it is understandable by itself. Among these models, we can mention Linear Regression, Decision Trees, K-Nearest Neighbours, Rule Based Learners and Bayesian Models.

Logistic Regression [65] takes the assumption of linear dependence between the predictors and the predicted variables, impeding a flexible fit to the data. This stiffness of the model is the reason that put Logistic Regression inside the set of transparency models. A Decision Tree (DT) is a ML algorithm that builds a tree-like structure used for regression and classification problems [66]. DTs have long been used in the field of decision-making due to their intrinsic transparency. DTs recursively split a dataset into smaller groups until it reaches sets that are small enough to be described by a specific class or class label. Each node of the tree structure represents a test on a specific feature and each leaf node is associated with a class label. The splitting process of DTs relies on a set of splitting policies which depend on the features [67]. The branches created during tree generation represent conjunctions of features that lead to a specific class. Thus, a path from the root to a leaf represents a whole decision path. K-Nearest Neighbours (KNN) is a ML algorithm for classification. It classifies a new sample by voting the classes of its k nearest neighbours. KNN can also be used for regression problems. For classification, the concept of neighbours is induced by the distance between samples, in the case of regression, the voting system used for the prediction is replaced by an average of the target values associated with the k nearest neighbours. The explainability in the KNN is based on the notion of distance or similarity between samples and on the possibility to evaluate the reasons for a new sample's classification inside a group. Every model that produces rules to classify the data it is meant to learn from is referred to as Rule Based Learner. Rules can take the shape of basic conditional *if-then* rules or more complicated combinations of simple rules. Rules Based Learners are naturally explainable by their ability to generate rules in a highly human-understandable form. They are often used to explain more complex models, exploiting the rules generated to explain their predictions. Bayesian models take the form of a probabilistic graphical model that links together a set of variables with conditional dependencies.

Due to their graphical form, Bayesian models give a good explanation of the relationship between features and targets.

For more details, we recommend reading the works of Arrieta et al. [64] and Vilone et al [68]. In this thesis, the explanations of the ensemble and the neural models are highlighted precisely because they are the families of models mainly used in this work. In addition to the neural models described above, ensemble models are a machine learning approach to combine multiple other models in the prediction process

7.2 Post-hoc Explainability Methods

Post-hoc explainability methods are techniques for retrieving an explanation from ML models that are not transparent by design. We can consider these techniques as a set of extensions or external methods to be applied to ML models to explain their decisions. We can divide these techniques into two families: model-agnostic and model-specific methods.

Model-agnostic methods can be applied to any ML model and the specific post-hoc techniques are model simplification, feature relevance extraction and model visualization. An example of a model simplification method is Local Interpretable Model-Agnostic Explanations (LIME) [69]. It builds an arbitrary number of surrogate simplified models. These surrogate models are trained on the prediction of the base (opaque) model. Then, LIME interprets individual model predictions by locally approximating the model around a given prediction. Feature relevance extraction techniques aim to explain the decision (or the result) of opaque models by measuring the importance of each feature in the output of the model to be explained. A significant example of this type of explanation method is SHapley Additive exPlanations (SHAP) [70]. SHAP is a game-theoretic approach to explaining the output of different machine learning models. This explainer framework introduced by Lundberg et al. is based on the Shapley value [71]. The Shapley value is a concept from coalitional game theory. It assigns a reward to each player in a coalition, depending on the marginal contribution that they make to it. Visual explanations methods are another important part of model-agnostic techniques. Most of the visual explanation methods [72, 73] work along with feature relevance techniques thus

providing the information that is eventually displayed to the end user, because it is not easy to create a model-agnostic visualization technique based only on the input and the output of the model.

7.2.1 Post-hoc Methods for Ensemble Models

Ensemble models are among the most accurate ML models used nowadays. Their success is due to their good learning capability and their natural predisposition to explainability. In this family of ML models, we can cite models like Extra Trees, Random Forest (RF) and eXtreme Gradient Boosting (XGB). For these models, the two main approaches for post-hoc explanation are model simplification and feature relevance extraction. In the literature, different works exploit model simplification techniques for explaining tree ensembles like such as the method proposed by Pedro Domingos [74] that proposes to train a single less complex model on a dataset labelled by the ensemble model. Another interesting approach is the work of Hara Satoshi et al. [75] which uses a simple and a full model. The simple model makes the interpretations and the ensemble model the predictions exploiting Expectation-Maximization [76] and Kullback-Leibler divergence [77]. For feature relevance explanation methods, there is an important work by Breiman Leo et al. [78]. They have studied the variable importance in RF. Their methods are based on the Mean Decrease Accuracy (MDA) of the RF when a certain variable is randomly permuted in the out-of-bag samples [79]. Another interesting approach to extracting an explanation based on the feature importance is the measure of Mean Decrease in Impurity (MDI) [80]. This method calculates each feature's importance as the sum over the number of splits (across all trees) that include the feature, proportionally to the number of samples it splits.

7.2.2 Post-hoc Methods for Deep Learning Models

Feature relevance techniques are the best method for explaining neural networks. In this Section, we focus mainly on the explanation of MLPs and CNNs.

For MLP, one of the most representative and pioneering works is [81]. Gregoire Montavon et al. created a method called *deep Taylor decomposition*

that decomposes the classification of the MLP into contributions of its inputs. One significant problem in the explanation of deep networks is the verification of the correctness of the obtained explanations. Following this theme, some interesting approaches are *integrated gradients* experimented in [82], PatterNet [83] and PatternAttribution [83].

The works for explaining CNNs are divided into two branches: those works that try to understand the process of mapping the output of the network in the input space to see the salient part of the input that are discriminative for the output and those that try to understand the internal processes by analysing the single intermediate layers. For the first category of methods, one is the work of Matthew D Zeiler et al. [84] that uses the Deconvnet [85] for reconstructing the maximum activations for each internal layer output. Finally, to visualize these strong activations in the input image, the authors use an occlusion sensitivity method to generate a saliency map [86]. A further step into improving explanation of CNNs is proposed by Sebastian Bach et al. [87]. Here, the authors propose to visualize the contribution of each pixel in the input images for the prediction in the form of a heatmap. They use a technique called Layer-wise Relevance Propagation (LRP) that is based on the Taylor series close to the prediction point [87]. To improve again visual explanation methods like heatmaps, class activation maps and saliency maps are used. Notable is the work of Grad-CAM [88] that computes the gradient of the output score for the target class, calculated before the last activation w.r.t. the feature map activations of the last convolutional layer. This produces a localization map that highlights the most important parts in the input image for predicting that specific target class. For the second category of methods, one important work is the techniques proposed by [89]. The authors proposed a method for reconstructing images from CNN's internal representation. They showed that different layers have accurate information about the image. Similar work is proposed by Anh Nguyen et al. [90] that introduce a Deep Generative Network to generate the most representative image for a specific class. Most of the methods of this category, exploit the post-hoc technique of *local explanation* for both obtaining a local interpretation of the internal process of a CNN and for obtaining a global model explanation. In the literature, there are many other methods for MLPs, CNNs and other types of ANNs explanation.

based on different approaches like the previously mentioned LIME or based on adversarial techniques [64].

Part III

Explainable Deep Learning

Chapter 8

Explanation by Neuro-Symbolic Integration

This Chapter present a paradigm of NeSy that integrates DNNs with symbolic AI where the two parts work together but without a reciprocal and direct integration. Section 8.1 describe Symbolic DNN-Tuner[1] [91, 92]. This is a Neural-Symbolic Neural Architecture Search (NAS) with hyper-parameters optimization based on Bayesian Optimization (BO). This system can be used to guide the training of a DNN in constrained situations. For example, if one wants to design and train a network by binding it to hardware power and resource consumption constraints as described in Section 8.2. Section 8.3 describe a NeSy ensemble learning system for multimodal learning [93] called Neural HPLP. This system is developed for learning from heterogeneous data sources (structured data and 2D or 3D images) and provides a human-understandable classification in terms of explainability of the AI model. Neural HPLP was developed for use in the medical field, specifically for analyses and forecasts of the state of patients affected by COVID-19. This system leverages both clinical data and patients' lung CT scans to predict patient mortality or status.

[1]Symbolic DNN-Tuner website: https://ml.unife.it/software/symbolic-dnn-tuner/

8.1 Symbolic DNN-Tuner

DNNs are very sensitive to the tuning of their HPs. Different tunings of the same neural network can lead to completely different results. For this reason, Hyper-Parameters Optimization (HPO) algorithms play an important role in building Deep Learning models. These algorithms have shown good performance [94], comparable with human experts.

Here we aim at creating an algorithm to drive the training of DNNs, automatizing the choice of HPs and analysing the performance of each training experiment to obtain a network with better performance. The algorithm combines an automatic tuning approach with some tricks usually used in manual approaches [95]. For the automatic approach we use BO [96]. This choice is motivated by the fact that tuning DNNs is computationally expensive and the BO algorithm limits the evaluations of the objective function (DNNs training and validation in this case) by spending more time in choosing the next set of HPs values.

The tricks used in manual approaches to solve problems are mapped into (non-deterministic, and probabilistic) STRs. These rules identify Tuning Actions (TAs), which have the purpose of editing the HPs search space, adding new HPs or updating the network structure without human intervention. All this is aimed at avoiding network problems like overfitting, underfitting or incorrect learning rate values and driving the whole learning process to better results.

Symbolic DNN-Tuner is composed by two main parts: a Neural Block that manages the neural network, the HPs search space and the application of the TAs, and a Symbolic Block (developed with PLP, and STRs in particular) that, on the basis of the network performance and computed metrics after each training, diagnoses problems and identifies the (most probable) TA to be applied on the network architecture. In the beginning, probabilistic weights of STRs are set manually, and then they are refined, after each training, via Learning from Interpretations (an inference available in PLP) based on the improvements obtained or not, for each TA applied in previous training.

8.1.1 Bayesian Optimization

BO is an approach to optimize objective functions (f) which are very expensive and/or slow to optimize [96]. The main idea behind this approach is to limit the time spent in the evaluation of f by spending more time choosing the new set of HPs values. BO builds a surrogate model of the objective function, quantifies the uncertainty in the surrogate using a regression model (*e.g., Gaussian Process Regression*) and uses an acquisition function to decide where to sample the new set of HPs [97]. The focus of BO is solving the problem:

$$\max_{x \in D} f(x) \tag{8.1}$$

where the input x is in $\mathbb{R}^d$, d is the number of HPs and D is a search space which can be seen as a hyper-cube where each dimension is a HP. Then, BO builds a probabilistic model for $f(x)$ and exploits this model to decide where to sample the next set of HPs values. The idea is to use all the information derived from previous evaluations of $f(x)$ as a memory to make the next decision.

BO consists of two crucial components: the *probabilistic regression model* (e.g., Gaussian Process, see below) and the *activation function*. The first component provides a posterior probability distribution that captures the uncertainty in the surrogate model and the second determines the next point to evaluate. This is done by measuring the value that would be generated by the $f(x)$ at this new point based on the posterior distribution [97]. This *activation function* is also used for finding a good balance between Exploration and Exploitation. Exploration aims at selecting samples that eliminate the parts of the input search space that do not include the maximizer of the $f(x)$, while Exploitation aims at selecting the sample closest to the optimum with a high probability [98].

Gaussian Processes (GPs) are stochastic processes and prior distributions on functions. GPs offers a non-parametric approach in that it finds a distribution over the possible functions $f(x)$ that are consistent with the observed data. A GPs can be used for regression problems. Any finite set $X = \{x_1, x_2, ..., x_n\}$ induces a multivariate Gaussian distribution with n dimensions and a GP is completely determined by the mean function μ and the covariance matrix K,

$f \sim \mathcal{GP}(\mu, K)$ [99, 100]. The covariance matrix K is created by evaluating a *covariance function* (*kernel*) k at each pair of points x_i, x_j, $K = k(x_i, x_j)$. The kernel is chosen so that points x_i and x_j that are closer in the input space, have a larger correlation. In this way, it can be obtained that these points should have more similar function values than points that are far apart. For convenience, we assume that the prior mean is the zero function $\mu = 0$ and, for covariance, a very popular choice is the squared exponential function

$$k(x_i, x_j) = \sigma^2 exp\Big(-\frac{1}{2l^2}\|x_i - x_j\|^2 \Big) \tag{8.2}$$

where parameters σ determines the variation of function values from their mean and l describes how smooth a function is. Given vectors $X = [x_1, x_2, ..., x_n]^T$ and $Y = [y_1, y_2, ..., y_n]^T$ of observed values, the aim is to predict the y value for a new point x. It can be shown that y is Gaussian distributed with mean $\mu = \mathbf{k}^T C^{-1} Y$ and variance $\sigma^2 = k(x, x) - \mathbf{k}^T C^{-1} \mathbf{k}$ [101, 29]. $\mathbf{k}$ is the column vector with elements $k(x_i, x)$ and C is composed by elements $C_{ij} = k(x_i, x_j) + s^2 \delta_{ij}$ where s^2 is the variance of the random noise in the linear regression model $y = f(x) + \epsilon$ and δ_{ij} is the *Kronecker delta* ($\delta_{ij} = 1$ if $i \neq j$, 0 if $i = j$). So, if $s^2 = 0$, $C = K$. In this way, it is possible to define a prior distribution over parameters instead of choosing values. For a complete and more precise overview of the GP and its application to ML, see [99].

The Acquisition function determines the next point to evaluate or, in our case, the next set of HPs values. There are many types of acquisition function but, the most commonly used is *Expected Improvement (EI)* [97, 102]. EI defines a non-negative expected improvement over the best previously observed target value at a given point x. Since we observe f, we can say that $f_n^* = \max_{m \leq n} f(x_m)$ is the optimal choice, i.e., the previously evaluated point with the largest value, where n is the number of times that we have evaluated f thus far. Now suppose that if we evaluate our function f at point x, we will observe $f(x)$. After the evaluation at the point x, the value of the best observed point will be either $f(x)$ or f_n^* (clearly, it depends on who is greater than the other). The improvement in the value of the best observed point is:

$$best = \max\{0, f(x) - f_n^*\} \tag{8.3}$$

for simplicity we can write this like $[best]^+$. We would like to choose an x that leads to maximum improvement, and we take the expected value of this improvement and choose x to maximize it. We can formalize this as follows:

$$EI_n(x) = E_n\left[[best]^+\right] \tag{8.4}$$

$$EI_n(x) = E_n\left[[f(x) - f_n^*]^+\right] \tag{8.5}$$

where $E_n[\cdot]$ is the expected value taken under the posterior distribution of f after having observed $x_1, ..., x_n$ [97].

8.1.2 DNNs Training Problems and Countermeasures

Experts in DNN training have identified several problems that can be encountered, see [95, 103]. Here we list them together with the appropriate countermeasures mapped into TAs. Table 8.1 shows the association between symptoms and diagnosed problems. Table 8.2 shows the association between the problems and the corresponding TAs with the acronyms used in Section IV Chapter 10 for the description of the experiments.

8.1.2.1 Overfitting

Overfitting is the lack of generalization ability of the model. This happens when the model adapts too much to the training data, not generalizing and therefore not working correctly on the validation data [103]. Diagnosing over-

Table 8.1: Symptoms and related problems

Symptoms	Problem
Gap between accuracy in training and validation	Overfitting
Gap between loss in training and validation	Overfitting
High loss	Underfitting
Low accuracy	Underfitting
Loss trend analysis	Increasing loss
Fluctuation of the loss	Fluctuating loss
Evaluation of the shape of the loss	Low learning rate
	High learning rate

Table 8.2: Problem - TA associations

Problem	TAs	Acronyms
Overfitting	Regularization and Batch Normalization	reg_l2 & batch_norm
	Increase dropout	inc_dropout
	Data augmentation	data_augm
Underfitting	Decrease the learning rate	decr_lr
	Increase the number of neurons	inc_neurons
	Addition of fully connected layers	new_fc_layer
	Addition of convolutional blocks	new_conv_layer
Increasing loss	Decrease the learning rate	decr_lr_inc_loss
Fluctuating loss	Increase the batch size	inc_batch_size
	Decrease the learning rate	decr_lr_fl
Low learning rate	Increase learning rate	inc_lr
High learning rate	Decrease learning rate	dec_lr

fitting is relatively easy by monitoring the performance of the network during both training and validation. Symbolic DNN-Tuner checks the difference between training and validation for both the accuracy and the loss, in order to identify any possible gap. A significant gap between the two phases is a clear symptom of overfitting. When overfitting is diagnosed, Symbolic DNN-Tuner applies two possible TAs, as shown in Table 8.2: Regularization [104] and Batch Normalization [105], Increase Dropout [106] and Data Augmentation [107].

8.1.2.2 Underfitting

Underfitting happens when the model is not able to learn and fails in both the training and validation phases [103]. In order to detect underfitting, Symbolic DNN-Tuner measures the accuracy and the loss in the validation phases. If, at a certain iteration, the loss is greater than a manually predetermined threshold and accuracy is lower than another manually predetermined threshold, Symbolic DNN-Tuner diagnoses underfitting. These two thresholds are dynamically increased as the algorithm progresses. This allows Symbolic DNN-Tuner to become increasingly demanding as iterations progress. For fixing underfitting, there are different TAs (Table 8.2). Apart from decreasing the learning rate, the remaining TAs aim at increasing the learning capability of the network. Besides increasing the number of neurons and the addition of fully connected layers which are self explicative, we may add a convolutional block composed

of a sequence of two convolutional layers followed by a pooling and dropout layer.

8.1.2.3 Increasing Loss

Symbolic DNN-Tuner uses Early Stopping, so, if at a certain iteration the training loss starts growing, this means that the learning rate is probability too high. In this case, Symbolic DNN-Tuner applies a TA that aims at reducing the learning rate's search space by eliminating all values that are larger than the last chosen.

8.1.2.4 Fluctuating Loss

The oscillation in the loss is usually due to the batch size. When the batch size is 1, oscillation can occur and the loss will be noisy. When the batch size is the complete dataset, the oscillation will be minimal because each update of the gradient should improve the loss function monotonically (unless the learning rate is set too high). When this behaviour is detected, a TA that aims at shrinking the batch size's search space is applied, to make sure larger values are selected in the next iterations.

8.1.2.5 Management of the Learning Rate

For the correct management of the learning rate, we exploit the relation between the loss and the learning rate [108], as can be seen in Figure 8.1. From the trend of the loss, Symbolic DNN-Tuner can diagnose if the learning rate is too high or too low. For doing this, the algorithm computes the integral of the loss and the line between the initial and the final loss, that is AUL and $AULL$ respectively. Then the absolute difference between the two is computed. The next step is to check whether the difference is greater or less than two thresholds as you can see in Equation 8.8.

$$AUL = \int loss \qquad AULL = \int line \tag{8.6}$$

$$R = |AULL - AUL| \tag{8.7}$$

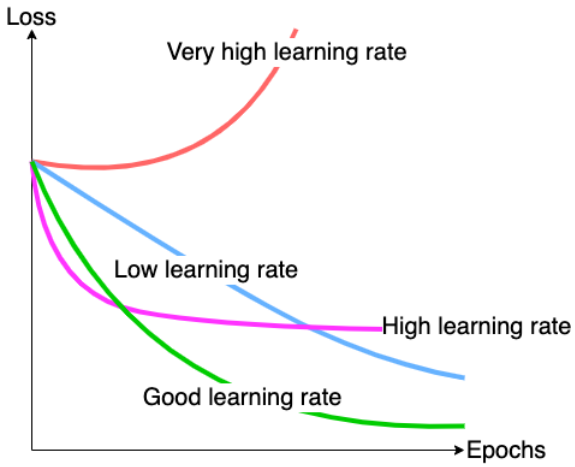


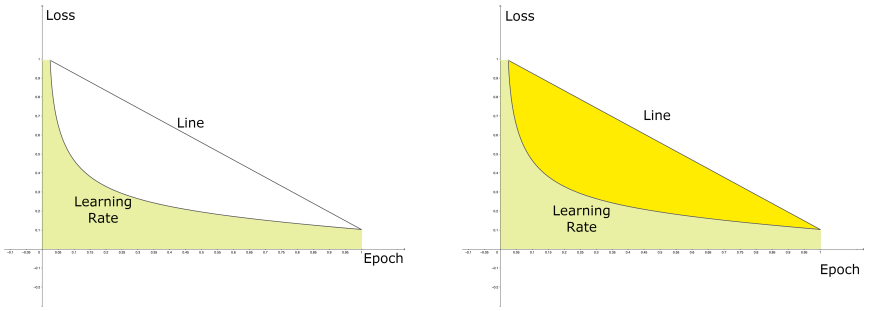
Figure 8.1: Relation between loss and learning rate.

$$Problems = \begin{cases} too\_large\_lr & \text{if } R > \frac{3AULL}{4} \\ too\_small\_lr & \text{if } R < \frac{AULL}{4} \\ good\_lr & \text{otherwise} \end{cases} \tag{8.8}$$

Figures 8.2, 8.3 and 8.4 shows the difference between the AUL and AULL in the three main cases of *good*, *high* and *low* learning rate. R changes considerably depending on the shape of the loss and therefore depending on the learning rate. When the diagnosis is too_large_lr, a TA that removes large values from the learning rates search space is applied. On the contrary, when the diagnosis is too_small_lr, a TA that removes small values from the learning rates search space is applied.

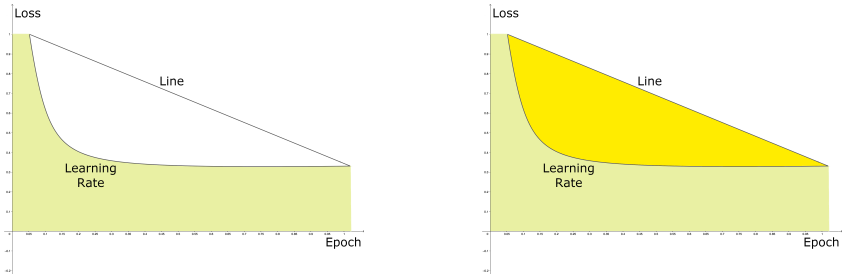
8.1.3 Framework Description

Symbolic DNN-Tuner is a system developed and implemented during the PhD programme to drive the training of a DNN, analysing the performance of each training experiment and automatizing the choice of HPs to obtain a network with better performance. It only requires an initial definition of the network architecture, a space of values for the HPs to be optimized and the datasets for training and validation. The system starts with a given set of rules with



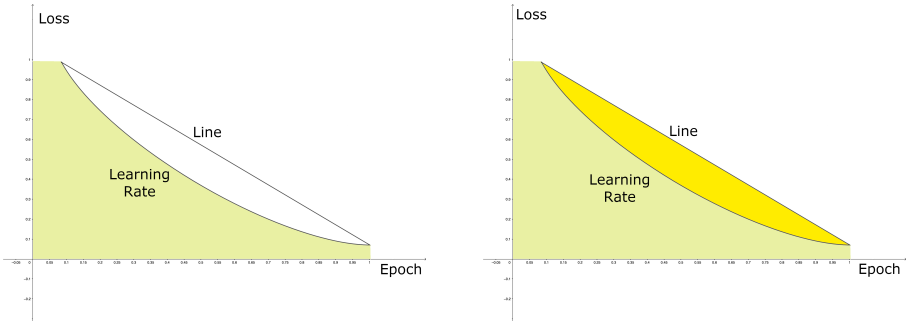
(a) AUL (b) $AULL$

Figure 8.2: AUL (light yellow area) and difference displayed in Equation 8.7 (yellow area) when learning rate is *good*.



(a) AUL (b) $AULL$

Figure 8.3: AUL (light yellow area) and difference displayed in Equation 8.7 (yellow area) when learning rate is *high*.



(a) AUL (b) $AULL$

Figure 8.4: AUL (light yellow area) and difference displayed in Equation 8.7 (yellow area) when learning rate is *low*.

default weights (which change by LFI, after each training and diagnosis phase). These rules are written in PLP, and implement Table 8.2. A sample of these rules is given in Listing 8.1 and Listing 8.3.

Symbolic DNN-Tuner exploits BO for the choice of HPs and applies a performance analysis at the end of each training and validation session in order to identify possible problems such as overfitting, underfitting or incorrect learning rate configurations as described in Section 8.1.2.

By analysing the behaviour of the network, it is possible to identify some problems (e.g., overfitting, underfitting, etc.) that BO is not able to avoid because it works only with a single metrics (validation loss or accuracy, training loss or accuracy). When Symbolic DNN-Tuner diagnoses these problems, it changes the search space of HP values or the architecture of the network by applying TAs to drive the DNN to a better solution.

8.1.3.1 Architecture

Symbolic DNN-Tuner is composed by two main parts: a *Neural Block* that manages the neural network, the HPs search space and the application of the TAs, and a *Symbolic Block* (where STRs are implemented in PLP) that, on the basis of the network performance and computed metrics after each training, diagnoses problems and identifies the (most probable) TAs to be applied on the network architecture.

In the beginning, the probabilistic weights of STRs are set manually, and then they are refined, after each training, via LFI on the basis of the improvements obtained or not, for each TA applied in previous training. The schema of Symbolic DNN-Tuner and of its blocks is shown in Figure 8.5.

STRs are probabilistic rules that map problems into resolutive actions (TAs) as defined in Table 8.2, and those weights are learned by LFI [33].

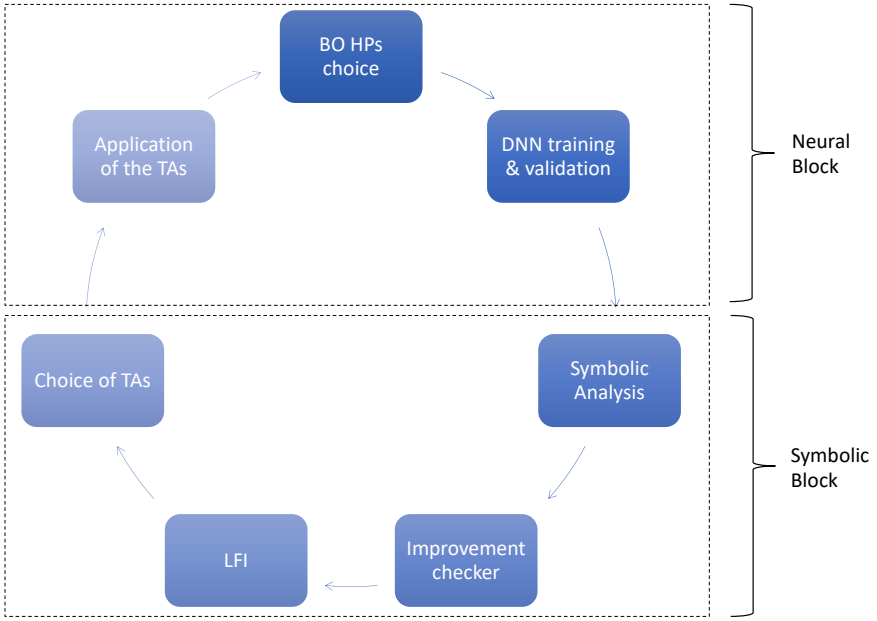


Figure 8.5: Symbolic DNN-Tuner execution pipeline with *Neural Block* and *Symbolic Block*. In the Figure is shown the Symbolic DNN-Tuner's pipeline and the steps between Neural and Symbolic block.

8.1.3.2 Algorithm

At each iteration, Symbolic DNN-Tuner trains and validates the neural network means of BO. Once training and validation have finished, the algorithm checks if training and validation have achieved better results than the previous training (the training of the neural network performed in the previous iteration of Symbolic DNN-Tuner) in terms of accuracy and loss (line 12 in Algorithm 1). This improvement check is used to build the training set for LFI. In fact, at each iteration, a new point is added to the training set for LFI and parameter learning is rerun. Then, the new weights of the STRs are placed in the symbolic program and the diagnosis starts. The symbolic analysis (line 22 in Algorithm 1) returns the TAs to be applied to the network architecture and/or HPs search space.

After each training, the BO status is saved, so that we can resume it for the new training of the network in the next iteration of Symbolic DNN-Tuner. The importance of starting a new training with a resumed BO status is that BO works by maintaining some kind of memory of the experience of past training

and, in this way, it will choose the HPs values in an optimized way. This is possible only if the HPs search space or the neural network architecture has not changed.

Algorithm 1 Symbolic DNN-Tuner

procedure SYMBOLIC_DNN_TUNER(S, M, n, PM)
  $Iteration \leftarrow 0$
  $Ckpt \leftarrow \emptyset$
  $(Ckpt, R, H) \leftarrow BO_s(S, M)$
  $(NewM, NewS) \leftarrow$ANALYSIS_TUNING(R, H, PM)
  while $Iteration \leq n$ **do** ▷ Symbolic DNN-Tuner main loop
    if $NewM \neq M$ **then**
      $(Ckpt, R, H) \leftarrow BO_s(S, NewM)$ ▷ New training with BO from scratch
    else
      $(Ckpt, R, H) \leftarrow BO_r(NewS, NewM, Ckpt)$ ▷ New training with a restored checkpoint
    end if ▷ - status of BO
    $Improve \leftarrow ImprovementChecker(R, DB)$
    $PM \leftarrow LearnFromInt(Improve, SymbolicDiagnosis, SymbolicTuning)$
    $(M = NewM, S = NewS) \leftarrow$ANALYSIS_TUNING$(R, H, PM)$
    $Iteration \leftarrow (Iteration + 1)$
  end while
end procedure

function ANALYSIS_TUNING(R, H, PM)
  $DB \leftarrow saveResult(DB, R)$
  $(AUL, AULL) \leftarrow Areas(H)$
  $(SymDiagnosis, SymTuning) \leftarrow SymbolicAnalysis([H, R, AUL, AULL], PM)$
  $(NewM, NewS) \leftarrow Tuning(SymDiagnosis, SymTuning)$
  return $(NewM, NewS)$
end function

Due to the sequential application of the TAs, there are some possible dangerous feedback loops. This problem can occur specifically with the TAs for the management of the learning rate (increasing and decreasing of the learning rate) or with the TAs for fixing underfitting. Thanks to the management described in Section 8.1.2.5, possible loops on learning rate are avoided. For TAs to fix underfitting, thresholds have been set to prevent loops that lead to too large networks.

Algorithms 1 encapsulates the whole Symbolic DNN-Tuner's process. With BO_s and BO_r we refer respectively to the functions that applies BO from scratch and BO with resumed status respectively. With S, M, n and PM we refer respectively to the search space of HPs values, initial neural net-

work, number of cycles of the algorithm and the *Probabilistic Model* coded into Symbolic DNN-Tuner. $Iteration$ is the counter of the iteration of Symbolic DNN-Tuner. $Ckpt$ is a checkpoint of the Bayesian Algorithm that can be used to restore the state of the BO.

BO_s, receiving S and M or $NewM$, returns the checkpoint of the BO, the results of the evaluation of the trained network in terms of loss and accuracy R (R is the tuple $(Acc, Loss)$) and the history H of the loss and accuracy in both training and validation. BO_r, receiving the new restricted search space $NewS$, the new neural network model $NewM$ and $Ckpt$, perform the same computation as BO_s but starting from a checkpoint rather than from scratch.

The results R are stored in the database DB. $Improve$ is a boolean value indicating whether there was an improvement over the previous iteration. PM is updated by the *Learning From Interpretation* ($LearnFromInt$) function after each iteration of the algorithm. This step can be applied after the first iteration of the whole algorithm because we need to have the $SymbolicDiagnosis$ and $SymbolicTuning$ to performing the $LearnFromInt$ function and learn the weights of the STRs. These two variables are obtained from the previous tuning step of the algorithm. The $LearnFromInt$ and $SymbolicAnalysis$ functions from line 19 of Algorithm 1 are fully developed exploiting PLP. will be explained in detail in the next section. AUL and $AULL$ will be useful for the analysis of the learning rate. $NewM$ and $NewsS$ are the new model and the new restricted search space obtained after the application of the TA, respectively.

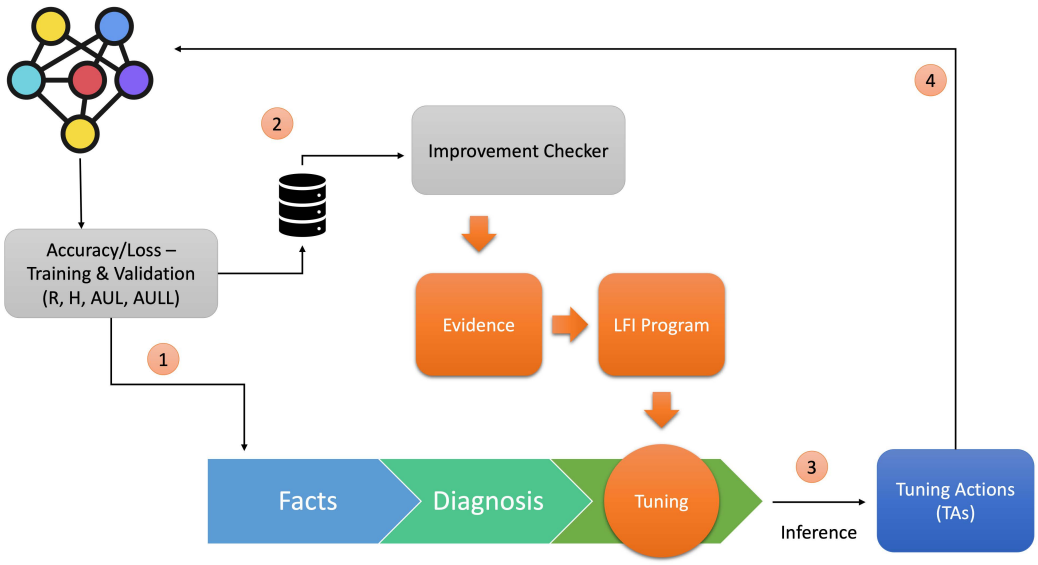


Figure 8.6: Symbolic Block of Symbolic DNN-Tuner. The numbers mark the order of execution.

8.1.3.3 Symbolic Section

The Symbolic Block performs the $LearnFromInt$ and $SymbolicAnalysis$ functions in Algorithm 1. This block analyses the network metrics (R, H, AUL and $AULL$ in Algorithm 1), producing a diagnosis and, from this, returns the TAs to be applied. In the following, we describe in more detail how STRs have been implemented in PLP and how their weights are calibrated by exploiting LFI.

The Symbolic Block, is composed of a PLP program with three parts: *Facts*, *Diagnosis* and *Tuning* (`FACTS`, `DIAGNOSIS` and `TUNING` sections in Listing 8.3). A sample program is shown in Listing 8.3. The whole logic program is dynamically created by the union of these three parts at each iteration of the Algorithm 1, as can be seen in Figure 8.6. *Facts* memorizes R, H, AUL and $AULL$ obtained from the Neural Block (arc 1 in Figure 8.6). The *Diagnosis* section encapsulates the code for diagnosing the DNNs behaviour problems. Finally, the *Tuning* section is composed by the STRs. *Facts*, *Diagnosis* and *Tuning* form the symbolic program. Thanks to ProbLog inference, we can query this program and obtain the TAs (arc 3 in Figure 8.6). And finally, TAs are passed to the Neural Block and applied to the DNN structure or the HPs

search space.

Each STR encapsulates a TA associated with a problem (see the associations in Table 8.2). TAs and problems of Table 2 are mapped into arguments of atoms of symbolic tuning rules, occurring in their head and body, respectively, as shown in Listing 8.1. Each STR has a weight which determines the probability of application of its TA, in case the associated problem is diagnosed. In the *Tuning* section, each STR is a rule such as those described in Listing 8.1.

Listing 8.1: STRs in the symbolic part of Symbolic DNN-Tuner

```
0.7::action(data_augment):- problem(overfitting).
0.3::action(decr_lr):- problem(underfitting).
0.8::action(inc_neurons):- problem(underfitting).
0.4::action(new_conv_layer):- problem(underfitting).
```

Listing 8.1 shows a subset of all STRs (see *Tuning* section in Listing 8.3 for a complete version of Listing 8.1). The `problem(...)` predicate is defined in the *Diagnosis* section of the Symbolic Block, see Listing 8.3.

The probabilistic weights are learned from the *experience* (evidences) gained from previous iterations. This *experience* becomes the set of *training examples* for the LFI program. Then, the LFI program is composed of two parts: the program and the evidences obtained from the *ImprovementChecker* module, as shown in Listing 8.2:

Listing 8.2: Learning From Interpretation part of Symbolic DNN-Tuner

```
% Program
t(0.5)::action(data_augment).
t(0.2)::action(decr_lr).
t(0.85)::action(inc_neurons).
t(0.3)::action(new_conv_layer).
- - - - - - - - - - - - - - - - - - - - -
% Evidence
evidence(action(data_augment), True).
evidence(action(decr_lr), False).
```

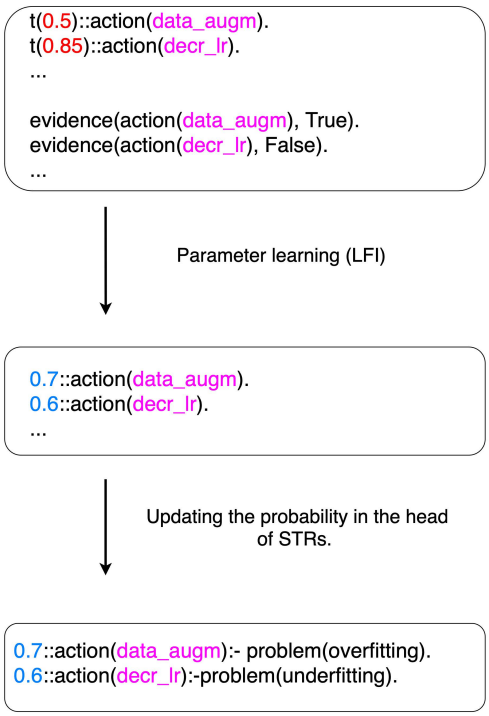


Figure 8.7: Learning From Interpretation pipeline. In the middle rectangle, in blue, we can see the learned parameters after the LFI.

This file is built dynamically at each iteration of Algorithm 1. After each training, Symbolic DNN-Tuner checks the improvement of the network with the *ImprovementChecker* module. The improvement (*Improve* in the Algorithm 1) is a Boolean value and it is used to build the evidence. The aim is to reward with greater probability those TAs that have led to improvements. In detail, in Figure 8.7 we can see that, starting from a program like Listing 8.2, learning the parameters of this program, we can obtain the new values of probability of applying the TA. After that, we can update the probability in the head of the STRs. In this way, Symbolic DNN-Tuner can learn which TA was better and consequently favours it over the others.

Finally, with the complete and updated symbolic program, we can use ProbLog inference and query the program for the probabilities of given query atoms, say, `query(problem(_))` and `query(action(_))`. For clarity, an extract of ProbLog code is provided in Listing 8.3.

In the rest of this Section, we show an extract of ProbLog code used in Symbolic DNN-Tuner.

Listing 8.3: Extract of Logic section of Symbolic DNN-Tuner

```
% FACTS ----------------------------------------------------------
a([0.0529399998486042, 0.0710360012948513,
0.6266616476927525, 0.6298289950701192, ... ]).
va([0.0191, 0.0593, 0.1797, 0.2304, 0.2512, 0.28,
0.5261, 0.5339, 0.5273, ... ]).
l([4.776382889623642, 4.218988112640381, 3.960466429057121,
1.8257129939079284, ... ]).
vl([5.670237278938293, 4.4710672222614285,
2.000358765614033, 1.9812814263105392, ...]).
itacc(0.10625000000000001).
itloss(0.4125).

% DIAGNOSIS ------------------------------------------------------
abs2(X,Y) :- Y is abs(X).
isclose(X,Y,W) :- D is X - Y, abs2(D,D1), D1 =< W.
gap_tr_te_acc :- a(A), va(VA), last(A,LTA), last(VA,ScoreA),
Res is LTA - ScoreA, abs2(Res,Res1), Res1 > 0.2.
gap_tr_te_loss :- l(L), vl(VL), last(L,LTL), last(VL,ScoreL),
Res is LTL - ScoreL, abs2(Res,Res1), Res1 > 0.2.
low_acc :- va(A), itacc(Tha), last(A,LTA),
Res is LTA - 1.0, abs2(Res,Res1), Res1 > Tha
high_loss :- vl(L), itloss(Thl), last(L,LTL), \+isclose(LTL,0,Thl).

% PROBLEMS -------------------------------------------------------
problem(overfitting) :- gap_tr_te_acc; gap_tr_te_loss.
problem(underfitting) :- high_loss; low_acc.

% TUNING ---------------------------------------------------------
action(reg_l2) :- problem(overfitting).
```

```
0.545454545454545::action(inc_dropout):- problem(overfitting).
0.0::action(data_augment):- problem(overfitting).
0.3::action(decr_lr):- problem(underfitting).
0.0::action(inc_neurons):- problem(underfitting).
0.545454545454545::action(new_fc_layer):- problem(underfitting).
0.4::action(new_conv_layer):- problem(underfitting).

% QUERY ------------------------------------------------------------
query(problem(_)).
query(action(_)).
```

Listing 8.3 shows a portion of the logic program of Symbolic DNN-Tuner. The first part contains the *Facts*. They describe the history of the accuracy and the loss during training phase and validation phase (`a([])`, `l([])` and `va([])` and `vl([])` respectively). `itacc()` and `itloss()` are two threshold used to diagnose underfitting (see Section 8.1.2).

The *Diagnosis* section contains some utility functions and examples of clauses used for the analysis applied to *Facts*. The clauses are used to catch some gaps between training and validation phases of accuracy and loss, and to identify if loss is too high or accuracy too low. With these clauses, we can identify the problems encountered through the clauses found in the *Problems* section.

At the end, there are the STRs containing the TAs (*Tuning* section), each with its probability in the head of the clauses. For example, if `gap_tr_te_acc` or `gap_tr_te_loss` is **true**, this means that there is a gap between training and validation accuracy or loss larger than 0.2 (e.g., training accuracy is 0.8 and validation accuracy is 0.5). This means that `problem(overfitting)` is **true**, then *overfitting* is diagnosed and the clauses with body true are the following in the Listing 8.4:

Listing 8.4: Clauses with body true

```
action(reg_l2) :- problem(overfitting).
0.545454545454545::action(inc_dropout):- problem(overfitting).
0.0::action(data_augment):- problem(overfitting).
```

By querying the program with `query(problem(_))` and `query(action(_))` we retrieve the problems and the TAs. In this case we retrieve overfitting with `query(problem(_))` and L2 regularization, increment dropout and data augmentation each with its probability of application with `query(actions(_))`.

8.1.3.4 Tracking and Monitoring Experiments

To monitoring the experiments, Symbolic DNN-Tuner has a dashboard developed with the Dash framework [2] and Netron [3][4]. Figures 8.8 and 8.9 show the four pages of the dashboard. During the execution, all events are logged. All trained models, together with their probabilistic weights, the diagnosis, the TAs applied, are saved. The Neural Component can also log the training information to be monitored using TensorBoard[5] if more details are needed on the performance of the networks.

Figure 8.8 shows the tabs of metrics and HPs of the dashboard. Figure 8.8 (a) displays metrics such as accuracy and loss during the training and validation phases. Figure 8.8 (b) shows the HPs used in each iteration of the software. The orange point is the last iteration with its HPs. To see the whole history of the HPs used in each iteration of Symbolic DNN-Tuner, you can follow the little red arrow in the HPs graph.

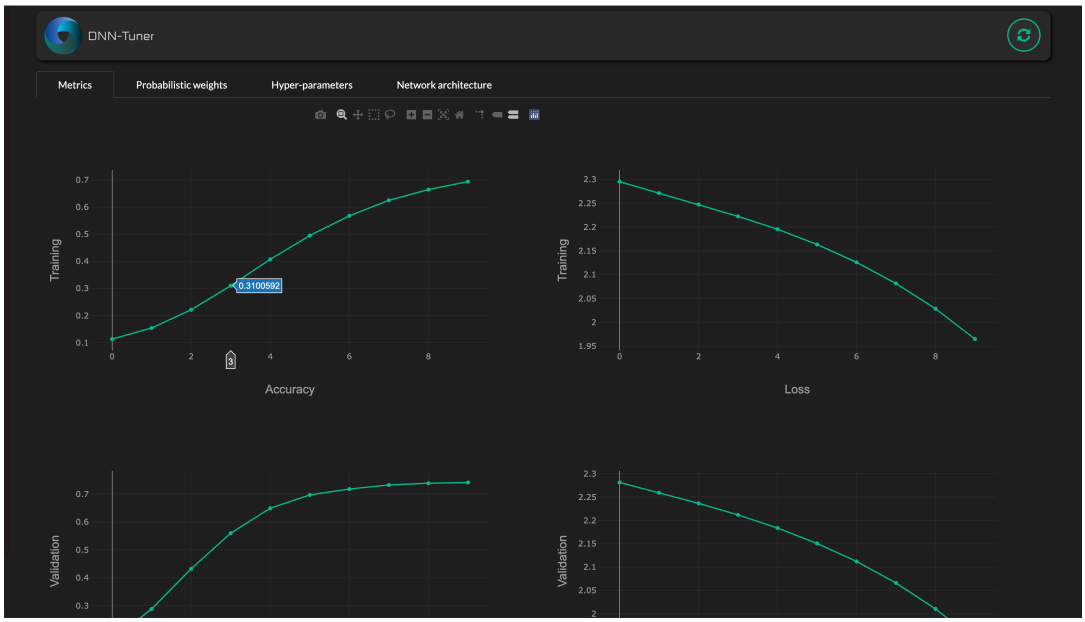
Figure 8.9 shows the tabs of probabilistic weights and network architecture of the dashboard. Figure 8.9 (a) shows the page with the probabilistic weights of the STRs. At each iteration of the Symbolic DNN-Tuner, it is possible to see which TA was found to be more effective during the HP optimization process. Figure 8.9 (b) shows the integration with Netron for visualizing the network architecture. Thanks to the integration with Netron, it is possible to interact with the network and see more details for each layer (e.g., activation, kernel size, kernel regularization, etc.).

8.1.4 Related Work

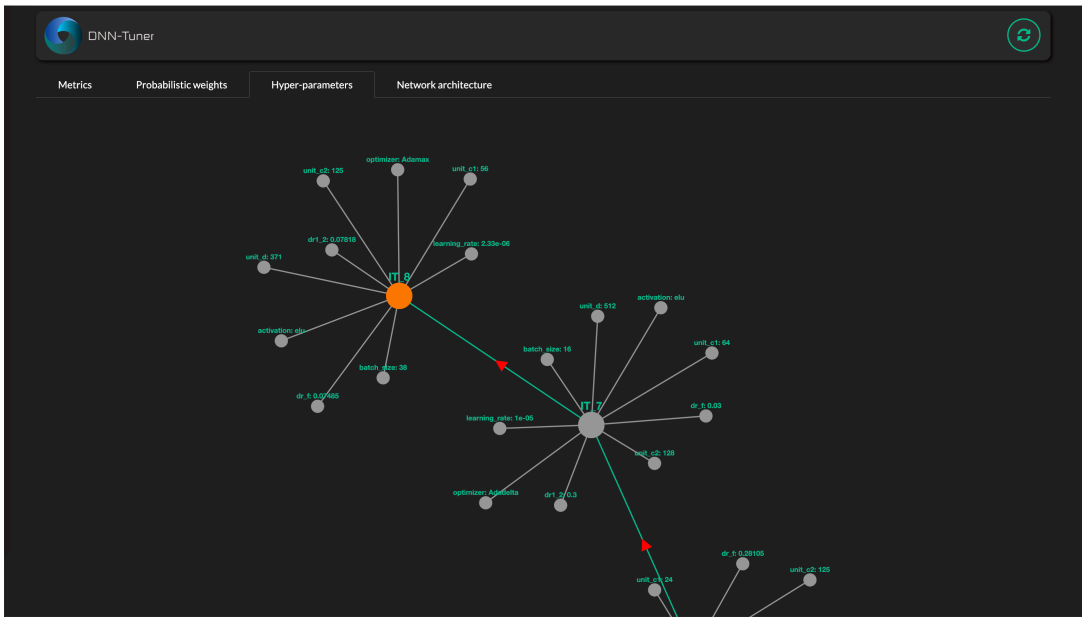
In the ML automation scenario, we can distinguish two main work areas: the HPO algorithm and the NAS algorithm [109]. The first works only on the HPs that govern the main settings of ML systems (including DL) for the training phases. The latter works mainly on the DNNs architecture.

In the field of DL, the state-of-the-art of HPO algorithm are: Grid Search,

[2]Dash framework website: https://plotly.com/dash/
[3]Netron website: https://lutzroeder.github.io/netron/
[4]Netron GitHub repository: https://github.com/lutzroeder/netron
[5]TensorBoard website: https://www.tensorflow.org/tensorboard



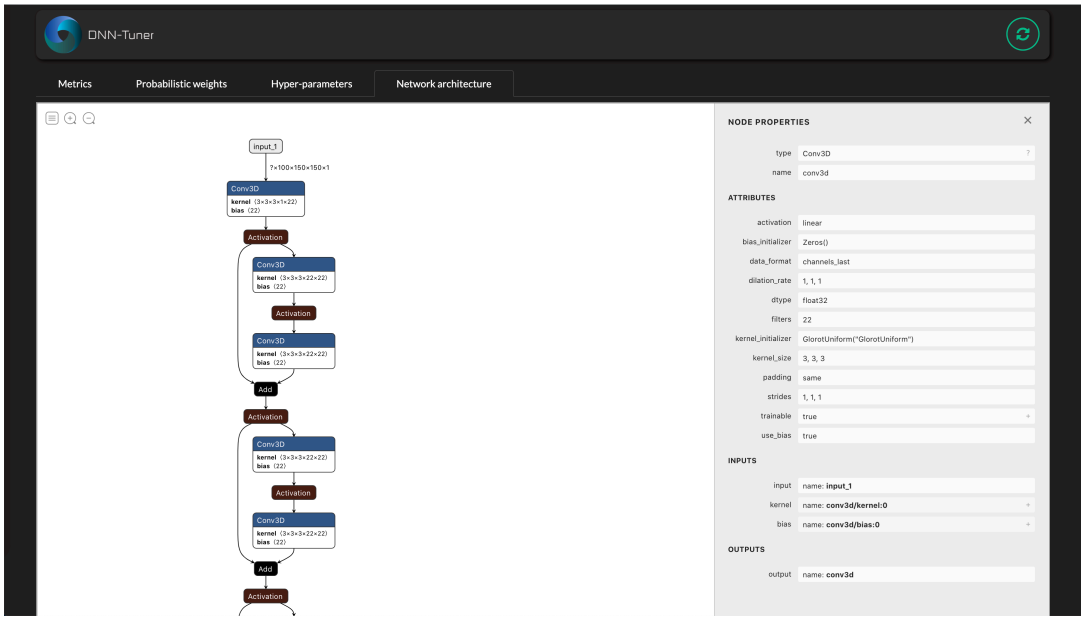
(a) Metrics of current training



(b) Hyper-Parameters

Figure 8.8: Symbolic DNN-Tuner's dashboard for monitoring the whole optimization process: (a) page with the metrics, (b) page with the HPs used in each iteration of the software.

(a) Probabilistic weights



(b) Network architecture

Figure 8.9: Symbolic DNN-Tuner's dashboard: (a) page with the probabilistic weights of the STRs of the Symbolic Parts, (b) visualization of the network's architecture using Netron.

Random Search and BO. Grid search [110] is the basic method for the HPO. It performs exhaustive research (also called brute-force research) on the user-specified HPs search space. This algorithm performs new training for each combination of the HPs and each training is independent of the others. This allows it to run in parallel and guarantees to find the optimal configuration but, Grid Search suffers from the *curse of dimensionality*. This problem arises because the computational resources increase exponentially with the number of hyper-parameters to set [110]. The application of this algorithm with the actual DNNs is correlated with the huge amount of HPs to set (then a huge number of possible configuration) and to the dimension of the modern DNNs architecture which could take a long time to complete the training phase. This rise a time problem.

Random Search [111] performs a random search over the used defined HPs search space. Random search leads to better results than the previous algorithm due to the predetermined budget (the searching process stops when this budget is reached). Random search may perform better especially when some HPs are not uniformly distributed [110]. Unlike the Grid Search, this algorithm does not guarantee to achieve the optimum, but it requires less computational time while finding a reasonably good model in most cases [111].

BO is a Sequential Model-Based Optimization (SMBO) algorithm aimed at finding the global optimum with the minimum number of trials. Its success in optimizing the HPs of DNNs is because BO limits the number of training of DNNs spending more time choosing the next set of HPs to try. In literature, there are works that apply this kind of HPO algorithm to DNNs [112, 100, 113].

NAS is the process of automating the design of DNNs architectures. It is strictly correlated to HPO and Automated Machine Learning (AutoML). NAS methods have outperformed manually designed architectures [114, 115]. The state-of-the-art NAS are: Efficient Neural Architecture Search (ENAS) [116], Differentiable Architecture Search (DARTS) [117], Single Path One-Shot (SPOS) [118] and ProxylessNAS [119]. We will explore this topic later.

8.2 Symbolic DNN-Tuner in DNN's Design for Edge Devices

The task of designing DNNs for edge devices is a problem that is placed in a new growing sub-field of ML called Tiny ML. Tiny ML deals with hardware and algorithms capable of performing on low-power and with low-computing capabilities devices. The aim is to find a systematic way of creating and adapting a neural network on a device with power and consumption constraints. We focused on the use of NAS [120] algorithms to design a DNN, the aim is to modify a NAS by adding constraints that help it to create a network respecting certain prerequisites. For this reason we have chosen to modify the optimization formula of Symbolic DNN-Tuner to take into account hardware constraints during the optimization process. In detail, as a hardware constraint, we use the number of FLOPS that the hardware can manage. Therefore, the goal is to create an optimization function that aims to maximize the accuracy of the created models and that tries to have a number of FLOPS that is as close as possible to the maximum threshold of FLOPS that the hardware can handle.

8.2.1 Neural Architecture Search

NAS is a technique for automating the design of DNN architectures. It is strictly correlated to AutoML [121]. The three main elements that compose a NAS are *Search Space, Search Strategy* and *Performance Estimation Strategy*. *Search Space* refers to all possible architectures that can be generated by the NAS. *Search Strategy* refers to the methods to explore the search space with the canonical exploration-exploitation trade-off. *Performance Estimation Strategy* refers to the methods to measure the performance of the neural network [109]. Given a *Search Space*, there are many *Search Strategies* that can be used to explore this space. These strategies include: random search, BO [122], reinforcement learning [116], gradient-based methods [117] and evolutionary algorithms like genetic algorithm[123] [109]. We can group NAS in two branches: the standard and the one-shot NAS [124]. Standard NAS follows the traditional search approach also used by Grid Search, where each generated DNN runs as an independent trail. One-shot NAS use weight sharing among models

in neural architecture search space to train a super-net and uses this to select better models. This type of algorithm reduces computational resources compared to the classical NAS algorithm. The state-of-the-art of one-shot NAS are: ENAS [116], DARTS [117], SPOS [118] and ProxylessNAS [119].

8.2.2 Symbolic DNN-Tuner's Constrained Optimization

The optimization process of Symbolic DNN-Tuner is based on the optimization performed by BO:

$$\min_{x \in D} f \tag{8.9}$$

where the input x is in $\mathbb{R}^d$, d is the number of HPs and D is the search space which can be seen as a hyper-cube where each dimension is a hyper-parameter. Then, BO builds a probabilistic model for f and exploits this model to sample the next set of HPs values. The aim is to use the information derived from previous evaluations of f as a memory bank to make the next decision in a non-random way. In the specific case of this work, the aim is to maximize the accuracy of the model and obtain a network with the FLOPS as close as possible to the FLOPS threshold. We can accept a model that slightly exceeds the FLOPS threshold because later a weight pruning pipeline can be applied. Thus it is possible to optimize the model by maximizing the accuracy and trying to obtain a network that uses the most of the power of the hardware in terms of FLOPS. In this way, the optimization formula became:

$$\min_{x \in D} f_{flops} \tag{8.10}$$

$$f_{flops} = -|A_i - |FLOPS_{th} - FLOPS_i|\epsilon| \tag{8.11}$$

where A_i and $FLOPS_i$ are the accuracy and the number of FLOPS of the ith model generated by Symbolic DNN-Tuner. $FLOPS_{th}$ is the threshold of the FLOPS that the hardware can manage and that forms the power constraint on which to optimize the network. ϵ is an HP used to balance and adjust the contribution of the calculated gap between the FLOPS on the objective

function. $FLOPS_i$ and $FLOPS_{th}$ are normalized between 0 and 1.

In order for Symbolic DNN-Tuner to optimize DNNs taking into account the FLOPS constraint, it is necessary to create a new STR that activates when the created DNN exceeds the FLOPS limit. To address this problem, we defined a new problem called *latency* and two new TA. The first TA decreases the number of neurons in the various layers and the second removes the last convolutional layer of the DNN and any other layers related to it (for example, pooling layer, dropout etc.). This is to maintain consistency in the neural network architecture after removing the convolutional layer. Listing 8.5 shows the two new STRs with their respective probabilistic weights.

Listing 8.5: STR for FLOPS constraint

```
0.4::action(dec_neurons,latency):- problem(latency).
0.7::action(dec_layers,latency):- problem(latency).
```

8.3 Neural-Symbolic Ensemble Learning for Medical Environment

The medical environment is a fertile ground for the design and implementation of neural-symbolic or multimodal ML algorithms. This because, in this filed, there are different sources of heterogeneous data that lead at the same concept or diagnosis. Then, exploiting DL algorithms on unstructured data (i.e., CT or X-Ray scans) it is possible to extract concepts that could be integrated on other structured data (i.e., clinical data) and therefore exploit both perceptive and reasoning algorithms to correlate this data e get to deeper deductions. This integration would also lead to the construction not only of more powerful but also more interpretable and safe systems. The aim of this work is to design and implement a system that is able to perform an accurate, reliable and interpretable diagnosis on patients. The system is easily adapted to any form of medical task that involves the intersection of visual and clinical data.

Due to this historical period conditioned by the pandemic, this work focused on the cases of application in the field of patient analysis and the infection from Covid-19, hoping to be able to make a contribution not only to the

scientific community but also help to the medical community to face this great challenge.

The global emergency caused by the spread of Covid-19 has highlighted the necessity for early-stage identification of the complications and risk status of patients caused by the Covid-19 infection. Early diagnosis is vital for Covid-19 positive patients. With the advent of the pandemic and Medicine 4.0, much research has focused on healthcare. In fact, AI techniques are increasingly applied to this field with the aspiration to make machines able to perform typical human activities [125, 126]. In sensitive areas such as medicine, it is necessary to build systems that are able to provide clear explanations of their decisions [63, 127, 128]. For these reasons, we chose to exploit both neural and symbolic models in order to detect Covid-19 infection from patients' clinical data and pulmonary CT scans. More importantly, it is necessary to motivative medical diagnoses or decisions with detailed reasoning and explanations. Then, DNNs are used to analyse unstructured data like CT scans and symbolic models are used to analyse structured clinical data. The aim of this work is to design and implement a NeSy model that is able to predict the severity of Covid-19 patients from clinical data and lung CT scans, and enable the model to provide an explanation of its prediction. The idea is to extract relevant patterns from heterogeneous data collected from patients to produce a more comprehensive analysis. Therefore, we trained a 3D-CNN for predicting the severity of lung lesions and a DT to predict the probability of a patient's death during hospitalisation. The output of these two systems is combined to generate the dataset for the final part of the system which integrates the neural and the symbolic approaches through HPLP.

8.3.1 System Architecture

In order to predict the health state of Covid-19 patients arriving at the hospital, we propose a novel Neural-Symbolic method shown in Figure 8.10, that integrates both symbolic and neural systems. The neural-symbolic block is based on HPLPs [129, 130], which is a ML model that is able to build scalable, reliable and explainable AI systems. HPLP receives as input the integration of the outputs of a DT system that predicts the severity state of Covid-19

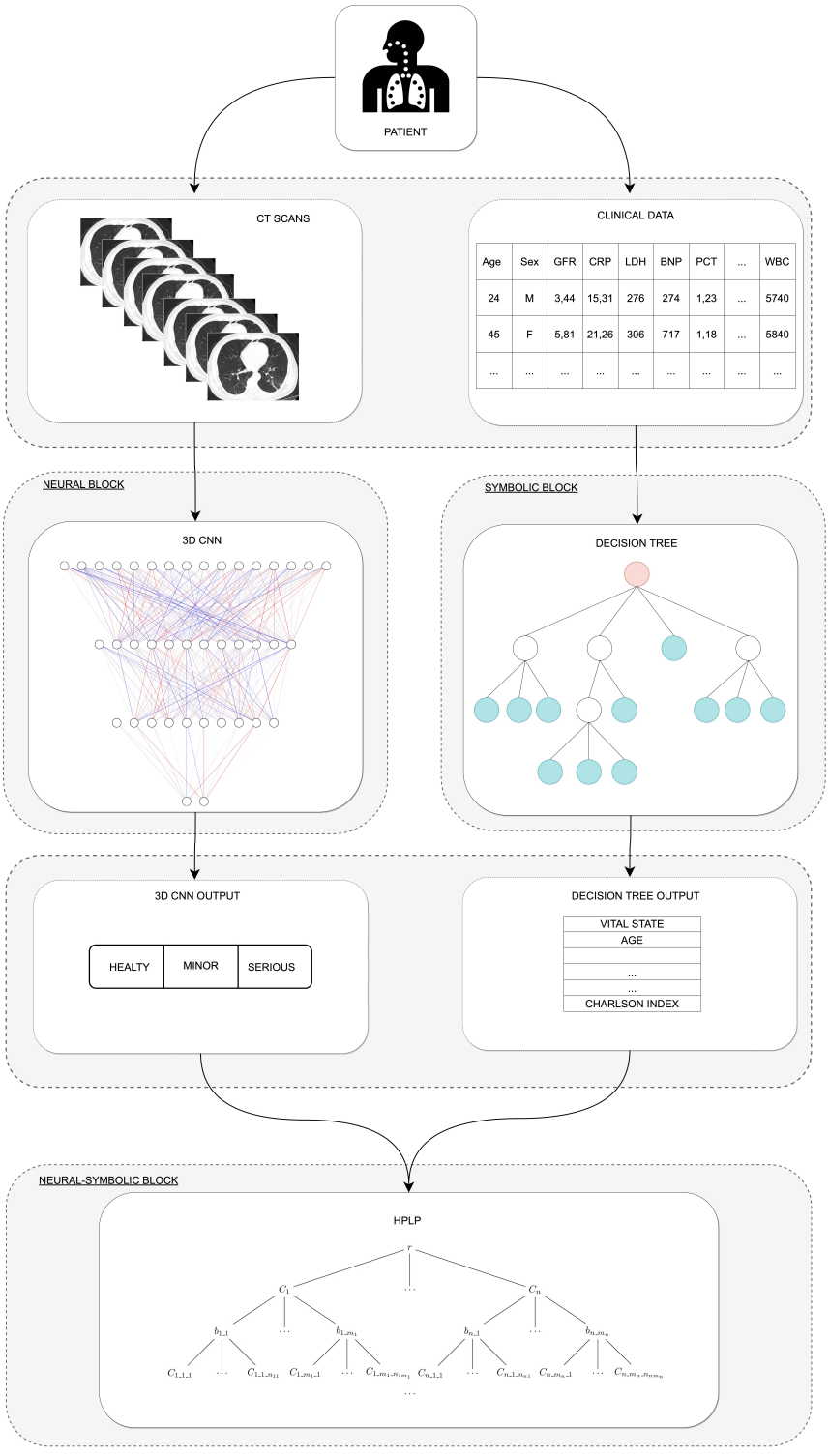


Figure 8.10: Overview of the proposed system. From top to bottom you can see the two different types of data used in this project that feed the two different ML systems and the integration of the two systems through HPLP.

patients from clinical data and a 3D-CNN that predicts the patients' lungs state using lung CT scans. Then, HPLP learns a set of probabilistic rules that predicts, at an early stage, if a Covid-19 patient arriving at the hospital will end in a critical condition. Therefore, we trained a 3D-CNN for predicting the severity of lung lesions and a DT to predict the probability of a patient's death during hospitalization. The output of these two systems are combined to generate the dataset for the final part of the system which integrates the neural and the symbolic approaches through HPLP.

8.3.1.1 Machine Learning models for Clinical Data

For the clinical data, DTs [66] and RFs [131, 132] are used. DT is a ML algorithm that recursively split a dataset into smaller groups until it reaches sets that are small enough to be described by a specific class or class label. It builds a tree-like structure in which each node represent a test on a specific feature (in our case a clinical feature) and each leaf node is associated with a class label. The splitting process of DTs relies on a set of splitting policies which depend on the features [67]. The branches created during tree generation represents conjunctions of features that lead to a specific class. Thus, a path from the root to a leaf represents a whole *decision path* (or *classification rule*). A decision path is composed of the leaf node and the conditions along the path (from root to leaf) that forms a conjunction of predicates as follows:

$$\textbf{if } condition_1 \wedge ... \wedge condition_n \textbf{ then } outcome \tag{8.12}$$

There are different DT algorithms. In this work we used the Classification And Regression Tree (CART) [133] algorithm.

RFs consist of different DTs that operates as an ensemble. Each tree in the ensemble produces its own class prediction, and the most frequent class among the DTs is the RF outcome. One of the most important aspects of RFs is the possibility of measuring the importance of each feature with respect to the prediction. This is made by measuring how much a tree node k using a particular feature reduce impurity across all trees in the forest as described in

Equation 8.13:

$$N_k^i = w_k C_k - w_{left(k)} C_{left(k)} - w_{right(k)} C_{right(k)} \tag{8.13}$$

where N_k^i is the importance of feature i in node k, w_k is the weighted number of samples in node k as a fraction of the total weighted number of samples, C_k is the impurity in node k and $left(k)$ and $right(k)$ are its respective children nodes. To retrieve a map of which features are important for the final classification result, we computed N_k^i for each node and each feature and normalize the result as displayed in Equation 8.14:

$$FI_i = \frac{\sum_{k:\ node\ k\ splits\ on\ feature\ i} N_k^i}{\sum_{j \in all\ nodes} N_j^i} \tag{8.14}$$

8.3.1.2 Deep Learning models for CT Scans

For unstructured data like CT scans, 3D-CNNs are used. Specifically, the developed 3D-CNN predicts the gravity of lung injuries from patient's CT scans. The typical format of images produced by CT machines is DICOM. So, a CT scan can be seen as a set of images that dissect the patient in slices to form a 3D image. For this reason we used a CNN with 3D filters. Before training, CT scans were pre-processed using a segmentation that creates a lung's binary mask followed by an application of a mask to eliminate unnecessary parts of the images as can be seen in Figure 8.11 . The segmentation was done using the Hounsfield (HU) scale. The HU scale is a quantitative scale for describing radiodensity in medical CT. On HU scale, air is represented by a value of -1000 and bone between $+700$ to $+3000$. As bones are much denser than the other soft tissues, they show up much better in CT scans. Using this information, it was possible to identify which part of the image contains lungs and create a binary mask (lungs are represented by a value between -700 to -600 in the HU scale). After the segmentation and after the application of the binary mask, the images were normalized between 0 and 1. As mentioned before, we work with 3D images, and to analyse three-dimensional data such as video and 3D images, where the volumetric or temporal context is important, it is necessary to use DNN with 3D convolutional layers. 3D-CNNs apply a three-dimensional filter to data that lies along the 3 dimensions (x, y, z). Their

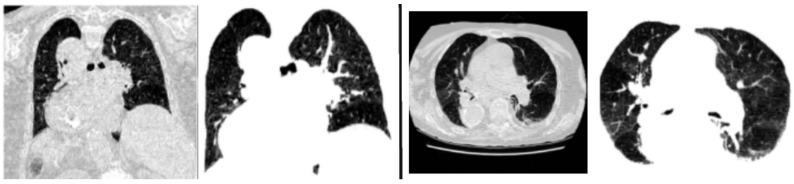


Figure 8.11: Segmentation of CT scans. The odd images represent an original slice of a DICOM voxel that depict the lungs of the patient. The even images represents the binary masks obtained after the pre-processing.

output shape is a three-dimensional volume space such as a cube. This is useful because, in addition to the 2D spatial characteristics of the images, we also want to learn the volumetric characteristics of the CT scans.

8.3.1.3 Neural HPLP

The proposed NeSy integration exploit HPLP [129, 130], an extension of Liftable PLP [134], which is an AI approach for integrating symbolic (e.g probabilistic logic program) and sub-symbolic (neural networks) systems. The proposed system, named Neural HPLP, learns a predicate, also called *target predicate* using a set of examples called *interpretations*. Each interpretation is associated with each patient and is composed of the outputs of the DT and the 3D-CNN described in Sections 8.3.1.1 and 8.3.1.2 respectively. The target predicate is, for a Covid-19 patient, that of being in a critical state.

Now, suppose we want to compute the probability of atoms[6] for a target predicate r using a PLP. In particular, we want to compute the probability of a ground atom $r(\vec{t})$[7], where $\vec{t}$ is a vector of terms[8]. We consider a specific form of Probabilistic Logic program that defines r in terms of *input predicates* (their definition is given as input and is certain) and *hidden predicates*, defined by clauses of the program. A discrimination is done between input predicates, which encapsulate the input data and the background knowledge, and the target predicate, which is the predicate we are interested in predicting, i.e. in our case Covid-19 patient in a critical state. We introduce the notion of hidden predicates which are disjoint from input and target predicates. Each

[6] An *atom* is a predicate, p, applied to some terms

[7] An expression (atom, literal, term or formula) is *ground* if it does not contain any variable

[8] A *term* is a variable, a constant, or a functor, f, applied to terms, $f(t_1, t_2, \ldots, t_n)$.

clause in the program has a single head atom annotated with a probability. Furthermore, the program is hierarchically defined so that it can be divided into layers. Each layer defines a set of hidden predicates in terms of predicates of the layer immediately below or in terms of input predicates. A generic clause C is of the form

$$C = p(\vec{X}) : \pi :- \phi(\vec{X}, \vec{Y}), b_1(\vec{X}, \vec{Y}), \ldots, b_m(\vec{X}, \vec{Y})$$

where $\phi(\vec{X}, \vec{Y})$ is a conjunction of literals[9] for the input predicates. The vector $\vec{X}$ represents variables appearing in the head of C and $\vec{Y}$ represents the variables introduced by input predicates. $b_i(\vec{X}, \vec{Y})$ for $i = 1, \ldots, m$ is a literal built on a hidden predicate. Variables in $\vec{Y}$ are existentially quantified with scope the body. Only literals for input predicates can introduce new variables into the clause. Moreover, all literals for hidden predicates must use the whole set of variables of the predicate in the head $\vec{X}$ and of input predicates $\vec{Y}$. Moreover, we require that the predicate of each $b_i(\vec{X}, \vec{Y})$ does not appear elsewhere in the body of C or in the body of any other clause, i.e each hidden predicate literal is unique in the program. We call Hierarchical PLP (HPLP) the language that admits only programs of this form [129]. A generic hierarchical program is defined as follows:

$$\begin{aligned}
C_1 = r(\vec{X}) : \pi_1 \quad &:- \quad \phi_1, b_{1_1}, \ldots, b_{1_m_1} \\
&\ldots \\
C_n = r(\vec{X}) : \pi_n \quad &:- \quad \phi_n, b_{n_1}, \ldots, b_{n_m_n} \\
C_{1_1_1} = r_{1_1}(\vec{X}) : \pi_{1_1_1} \quad &:- \quad \phi_{1_1_1}, b_{1_1_1_1}, \ldots, b_{1_1_1_m_{111}} \\
&\ldots \\
C_{1_1_n_{11}} = r_{1_1}(\vec{X}) : \pi_{1_1_n_{11}} \quad &:- \quad \phi_{1_1_n_{11}}, b_{1_1_n_{11}_1}, \ldots, b_{1_1_n_{11}_m_{11n_{11}}} \\
&\ldots \\
C_{n_1_1} = r_{n_1}(\vec{X}) : \pi_{n_1_1} \quad &:- \quad \phi_{n_1_1}, b_{n_1_1_1}, \ldots, b_{n_1_1_m_{n11}} \\
&\ldots \\
C_{n_1_n_{n1}} = r_{n_1}(\vec{X}) : \pi_{n_1_n_{n1}} \quad &:- \quad \phi_{n_1_n_{n1}}, b_{n_1_n_{n1}_1}, \ldots, b_{n_1_n_{n1}_m_{n1n_{n1}}} \\
&\ldots
\end{aligned}$$

[9] A *literal* is an atom or its negation

where r is the target predicate and $r_{1\_1\ldots\_n}$ is the predicate of $b_{1\_1\ldots\_n}$, e.g. r_{1_1} and r_{n_1} are the predicates of b_{1_1} and b_{n_1} respectively. The bodies of the lowest layer of clauses are composed only of input predicates and do not contain hidden predicates. Note that here the variables were omitted except for rule heads. A generic program can be represented by a tree, see Figure 8.12 with a node for each clause and literal for hidden predicates. Each clause (literal) node is indicated with $C_{\vec{p}}$ ($b_{\vec{p}}$) where $\vec{p}$ is a sequence of integers encoding the path from the root to the node. The predicate of literal $b_{\vec{p}}$ is $r_{\vec{p}}$ which is different for every value of $\vec{p}$.

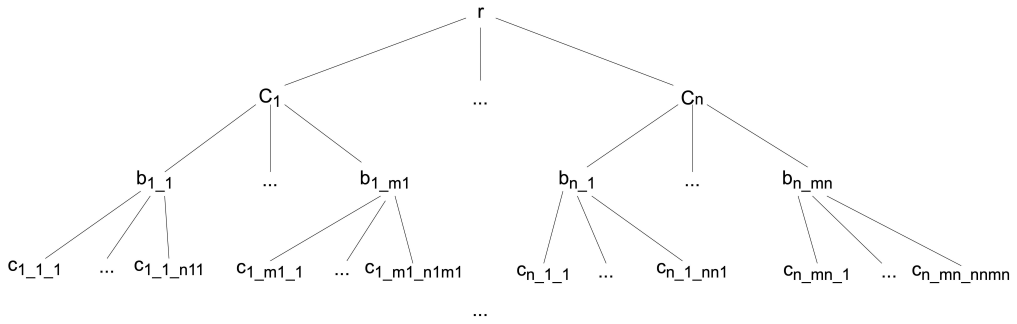


Figure 8.12: Generic HPLP Program

Given the target predicate to learn, Neural HPLP learns from data a Hierarchical Probabilistic Logic program (HPL program) which consists of a set of logical clauses annotated with probabilities. The learned program is able not only to predict whether a patient arriving at the hospital will end in a critical state but it is also able to give a useful explanation of its prediction. To learn a HPL program, an algorithm entitled Structure LEArning of Hierarchical Probabilistic logic programming (SLEAHP) generates a set of clauses called *bottom clauses* from examples called interpretations. An *interpretation* is a whole description of a particular example. In our case it contains all clinical information concerning a patient, see Example 1. Then, an initially large HPL program is randomly generated from the bottom clauses. This large HPL program is converted into a deep neural network and algorithms such as Gradient Descent/Backpropagation, see [135], and EM, see [136], are applied to learn the probabilities associated with the clauses. Finally, clauses with very small values of probabilities are removed. For a detailed description on HPLP, see [129, 130].

Since Neural HPLP takes as input a set of interpretations which consists of the whole description of information regarding a single Covid-19 patient, we generated as many interpretations as the number of available patients by applying the following criteria: each interpretation is annotated with a predicate that defines the critical state of the corresponding patient: *a patient is in a critical state if the DT classifies him/her as subject to death soon (dead) or if the 3D-CNN classified its lung as in serious condition.* Two more predicates are added in the interpretation which corresponds to the output of the DT (*dead* or *alive*) and the 3D-CNN (state of its lung, *serious*, *minor* or *healthy*) respectively. In order to enrich each interpretation, we also added in each the decision path, i.e the set of predicates the DT applies to take its decision, see Example 1.

8.3.2 Related Work

Different studies demonstrate that early diagnosing of Covid-19 considerably decreases its mortality rate [137]. Our work introduces an explainable AI system, Neural HPLP, that predicts if a Covid-19 will end in a severe condition and therefore will need intensive care or more intensive treatment. Predicting if a Covid-19 patient will end in a critical condition is useful in managing the pandemic and save human lives. In the peak of the crisis with numerous Covid-19 patients in severe conditions, managing the limited number of intensive care in any hospital becomes vital. Knowing early that a Covid-19 patient could end in serious conditions has many advantages: it allows doctors to gain early knowledge on patients and provide special treatment to those predicted to finish in severe conditions. Moreover, it allows doctors to predict the future number of patients in intensive care and therefore enable an optimal distribution of those places with respect to other critical diseases. Finally, by providing a rules-based explanation of its prediction, e.g the clinical attributes relevant to detect the severity condition, Neural HPLP not only guides doctors to provide special treatments to those patients, but appears to be a more interpretable and reliable predictive model.

Based on the format of the medical data such as structured clinical data, CT, radiographs, Electrocardiogram (ECG) etc., in the literature, it is pos-

sible to find different approaches and applications of ML and DL algorithms that analyse and create predictive models on Covid-19 positive patients. As far as it concerned work on clinical data, we can cite Chansik An et al. [138]. In [138] the authors suggest different ML models to diagnose Covid-19 patients based on socio-demographic information and medical status, for the nation-wide cohort of South Korea. Dan Assaf et al. [139] used DL, RF and DTs to improve the management of the pandemic through the optimization of both medical resources allocation and triage procedures. An italian study conduced by Augusto Di Castelnuovo et al. [140] used ML algorithms to analyse clinical data of about 3000 Covid-19 patients. The work aim at identifying the underlying characteristics affecting Covid-19 patients who died during hospitalization. Another study conduced by Yan Li, et al. [141] uses XGB and DTs to find some decision rules to detect patients at the highest risk of mortality.

Concerning work on CT scans and/or chest X-Ray, Ardakani et al. built an ML system that evaluates radiological features of CT images collected from patients with Covid-19 and non-Covid-19 disease. They used different ML algorithms to find the computer-aided diagnosis system with the best performance in distinguishing Covid-19 patients from non-Covid-19 pneumonia. Alsharman et al. [142] used a CNN to detect Covid-19 on CT scans in the early stage of disease course. Albahli [143] highlighted the high performance of DNNs in detecting Covid-19 patients. His model reach 89% of accuracy on synthetic data produced by GAN-based model. Parnian Afshar et al. [144] try an alternative framework based on Capsule Networks [145] called COVID-CAPS that is capable of handling small datasets. COVID-CAPS achieved an accuracy of 95.7%, sensitivity of 90%, specificity of 95.8%, and Area Under the Curve (AUC) of 0.97. In [146], the authors propose an interesting approach, similar to Neural HPLP, that works on both clinical and images data for predicting Covid-19 severity. The paper developed a ML model to predict Covid-19 severities and a model to predict progression to critical disorder. These models were trained on radiomics features and clinical variables. The work accurately predict Covid-19 severity and progression to critical illness from radiomics features joined with clinical attributes. Differently from Neural HPLP, the proposed models do not provide a clear explanation of its prediction.

Other work addressing Covid-19 thematic is being done. For example, based on the Intensive Care Unit (ICU), the work of Cheng, Fu-Yuan et al. [147] exploits ML to create a risk prioritization tool that predicts the ICU transfer within 24 hours. Another interesting work done by Montomoli et al. [148] exploits XGB algorithm to predict the increase or decrease in patients' Sequential Organ Failure Assessment (SOFA) score on day 5 after ICU admission.

The novelty of Neural HPLP mainly lies in the possibility of obtaining an explanation from the whole system thanks to the HPLP. In systems that exploit a different form of data, when using neural networks, it is almost difficult to provide an explainable interpretation of the results due to their black box nature. This differentiates Neural HPLP from the other works.

Chapter 9

Exploiting Explanation

This Chapter present a new paradigm of explainability, in this thesis called *Exploiting Explanation*. The concept of this paradigm is to exploit the explanation to improve the performance of the AI model itself. Therefore, the explanation given to us by the model itself is exploited to try to specialize it or to speed up learning. Section 9.1 presents a new loss function to be applied on the training phase of a CNN that acts as an attention module and it is used to make the CNN focus only on a specific area of the image. This procedure is useful in industrial cases where random parts of the image must be excluded from the analysis. This technique exploit a mask-guided attention module for enhancing the salient part of the image. There are different works in literature that exploit and implement this approach. We cite some works that applies this method to the task of pedestrian detection [149, 150, 151].

9.1 Cross Entropy Overlap Distance

Anomaly detection in industrial image data is of the highest importance for many tasks in the field of computer vision [152]. In the task of surface analysis, very often, the images acquired in an industrial environment contain some sections that are not part of the surface to be inspected [153]. Just think of images of products running on conveyor rollers or connected to other components not subject to inspection or simply images of the edge of the product which inevitably incorporates part of the background. In many cases, if we know the shape of a product to be inspected, we can simply use some traditional image

processing techniques to remove the useless parts from the images. But, in other cases, we don't know the exact shape of our product or the background appears in the image.

In order to focus on relevant points of an image, this work proposes and tests a new approach to identify a systematic way to train a CNN that focuses only on the area of interest. To do that, we identify the most important pixels in the images for classification according to a CNN. After that, we calculate how much these pixels overlap with the mask that is provided, for each image, during the training phase. The computed overlap value is added to the loss of the network, to force the network to recognize the most important pixels, only within the area marked by the masks.

The idea is to add an *Overlap Coefficient* to the standard cross-entropy. In this way, the more the identified anomaly is outside the AOI, the greater is the loss. We call the resulting loss CEOD. The advantage of adding the masks in the training phase is that the network is forced to learn and recognize defects only in the area circumscribed by the mask. The added benefit is that, during inference, these masks will no longer be needed. Therefore, there is no difference, in terms of execution times, between a standard CNN and a network trained with this loss. In some applications, the masks themselves are determined at run-time through a trained segmentation network like Mask Region Based Convolutional Neural Networks (Mask R-CNN) [154].

The contribution of the approach described in this section it the use of Grad-CAM to add a penalty when the network finds an anomaly outside the AOI. Our method differs from those previously mentioned in that the neural network focuses on distinguishing defects in a specific area of the image and not on the whole image. In this way, CNN can learn to distinguish faults that are in the area of interest from noise given by a heterogeneous background.

9.1.1 Problem Description

The problem is due to the heterogeneity that can occur in the images to be analysed in an industrial environment. In some cases, it is not possible to perfectly isolate the piece of the surface to be analysed due to the shape of the object or to the environment in which the image of the object is acquired.

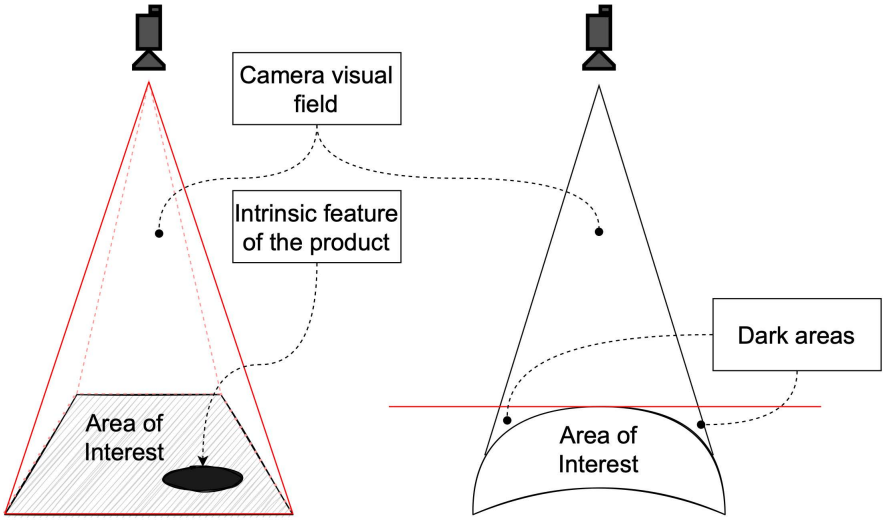


Figure 9.1: Visual examples of possible problems encountered during surface analysis.

These problems can bring a lot of useless information into the dataset which should still be processed by vision systems (neural networks in this case). Figure 9.1, on the left, shows a representation of the surface with intrinsic features like a tapped hole for a screw, which could be recognized as a defect since this feature is not present in all images and not always in the same location. On the right, there is a representation of a curved surface that, due to the curvature itself, can present dark areas that could have light refractions and might lead a neural network to mistake them for defects [155]. This useless information could alter the result of the network making it inefficient or unusable. Figure 9.2 shows an example that well describes the problem. To solve this problem, we need to focalize the vision system on a specific *Area of Interest* (AOI) making sure to weigh more the contribution of the information contained in this area than in the rest of the image.

This problem is like the task of instance/image segmentation but, unfortunately, in the industrial environment and the anomaly detection field, we don't know a priori the defect. Thus, we can't generate the right masks to train a segmentation network. For this reason, we focus on standard CNN for performing a binary classification to classify the images with anomalies.

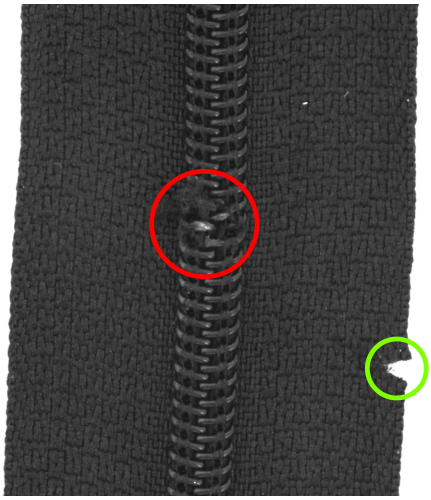


Figure 9.2: Example of image take from the MVTec AD dataset [1, 2] with two defects: one inside (red circle) and one outside (green circle) of the area of interest respectively. The damage outside of the area of interest, for classification purposes, must be considered not a defect.

9.1.2 Cross-Entropy Overlap Distance

The idea is to get an *overlap value* between objects to minimize during the training of the CNN. As an overlap value, we use the *Overlap Coefficient* (aka Szymkiewicz - Simpson coefficient) [156]. The objects for which we are going to calculate the overlap will be the mask provided during training (present in the dataset) and the region of the image that the CNN believes is most significant for the recognition of that image. To do this, we exploit an *explanation algorithm* called Grad-CAM [88]. Then, at the end of each forward pass of the network's training phase, we calculate an *heatmap* that highlights the most important pixels in the input image (the hottest pixels) with Grad-CAM. After that, we extract the hottest pixels (see Figure 9.3) and we calculate how much these hottest pixels overlap with the mask.

9.1.3 Visual Explanation by Grad-CAM

Grad-CAM [88] is a localization technique based on the Class Activation Mapping (CAM) algorithm [157] that generates visual explanations for any CNN without requiring changes or re-training. In order to generate a class-discriminative heatmap, Grad-CAM computes the gradient of the output score

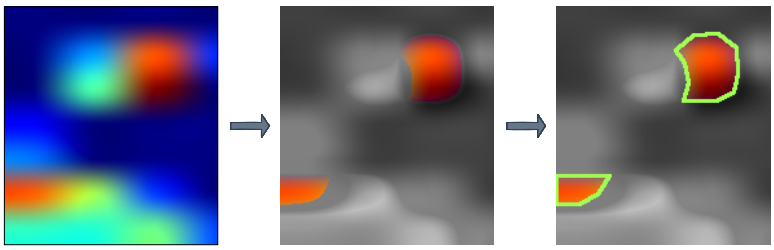


Figure 9.3: Extraction of the hottest pixels.

for class cls, the output (out_{cls}) calculated before the last (softmax) activation w.r.t. the feature map activations of the last convolutional layer. The global average pooling of these gradients is calculated to obtain the neuron importance weights α_{cls}:

$$\alpha_{cls} = \frac{1}{P} \sum_i \sum_j \frac{\partial out_{cls}}{\partial A} \tag{9.1}$$

where the $\sum_i \sum_j$ represent the global-average pooling, P represents the number of pixels in the feature map and A represent the feature map activations of the last convolutional layer. Finally, Grad-CAM performs a weighted combination of activation maps, followed by a ReLU to obtain the heatmap:

$$L_{cls}^{HeatMap} = ReLU(\sum_k \alpha_{cls} A) \tag{9.2}$$

For more details, see the work of R.R. Selvaraju et al. [88].

9.1.4 Mathematical Formulation

The equation of *Overlap Coefficient* is:

$$overlap_c(A_d, A_{gt}) = \frac{|A_d \cap A_{gt}|}{min(|A_d|, |A_{gt}|)} \tag{9.3}$$

where A_d and A_{gt} are the areas obtained through Grad-CAM and the segmentation mask (or area of the ground truth) respectively. A_{gt} is obtained with a manual segmentation or using a previously trained *segmentation neural network* [158, 159, 160]. Figure 9.4 shows a graphical representation of A_d and A_{gt}. In this case, if A_d is a subset of A_{gt} or the converse, the *Overlap Coefficient*

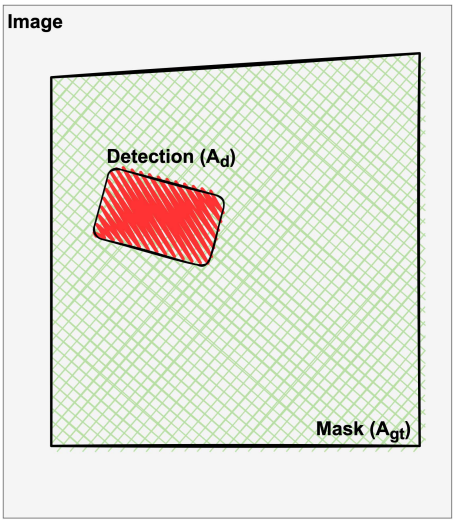


Figure 9.4: Stylized sample image with A_d and A_{gt} as area of the network detection and mask respectively.

is 1. If we want to add this term to the loss function of the neural network, we need to negate the Overlap Coefficient. Applying the negation of the logarithm we obtain a new value that we have called Overlap Distance (OD), expressed by Equation 9.4:

$$OD(A_d, A_{gt}) = -\ln(\frac{|A_d \cap A_{gt}|}{min(|A_d|, |A_{gt}|)}) \tag{9.4}$$

In this way, when A_d is a subset of A_{gt}, we obtain the $-\ln(1)$ and OD becomes 0, giving no contribution to the loss. The logarithm was introduced because it offers less penalty for small differences between predicted and corrected values. When the difference is large, the penalty will be higher. To optimize our CNN also w.r.t. this further aspect, we need to add this new term to the *Cross-Entropy* loss [161], as described by the following equation:

$$CE = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(p(y_i)) \tag{9.5}$$

where N is the number of examples, y_i and $p(y_i)$ are the label and the output of the network for the *i-th* example respectively. This is necessary in order to take into account the contribution of the classification task to the overall loss.

We thus obtain the CEOD that is:

$$CEOD = CE + OD(A_d, A_{gt}) \tag{9.6}$$

$$CEOD = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(p(y_i)) + \omega y_i(-\ln\big(\frac{|A_d^i \cap A_{gt}^i|}{min(|A_d^i|, |A_{gt}^i|)}\big)) \tag{9.7}$$

The term ω in Equation 9.7 is a new hyper-parameter to be set which represents the degree of impact of the new term on the overall loss. This term depends on the order of magnitude and on the difference of the two parts of the loss. In our experiments, after different tests, we have set ω to 0.001. The OD part of the loss is also multiplied by y_i to take into account the label of the images and it can take the value of 0 or 1. This is because, in the anomaly detection task, the defect-free images (*good* images) do not have specific areas with the hottest pixels but their heat map is rather uniform and with low-intensity levels.

A_d is obtained by extracting the pixels with the largest values obtained from the heatmap generated with Grad-CAM. Looking at the 3D representation of the heatmap displayed in Figure 9.5 (bottom left image), by filtering the heatmap for extracting the hottest pixels, we can obtain the object (or the area) (then the A_d term) used in Equation 9.1.

9.1.5 Algorithm

For exploiting the OD in the training of a CNN, we need to create a custom training loop for obtaining the feature extracted in the last convolutional layer, generating the heatmap and then using this in CEOD. For obtaining both the classification output and the features extracted from the last convolutional layer, the output of the CNN was modified. Algorithm 2 shows the process behind the custom training loop and the CEOD loss.

As can be seen in line 3 of Algorithm 2, we have applied a convolution with filter $\left(\begin{smallmatrix} .5 & .5 & .5 \\ .5 & .5 & .5 \\ .5 & .5 & .5 \end{smallmatrix}\right)$ to incorporate a *distance* concept in the OD computation. The transformation of the mask after convolution is visible in Figure 9.6. Note how, after convolution, there are no more clear differences in height (between

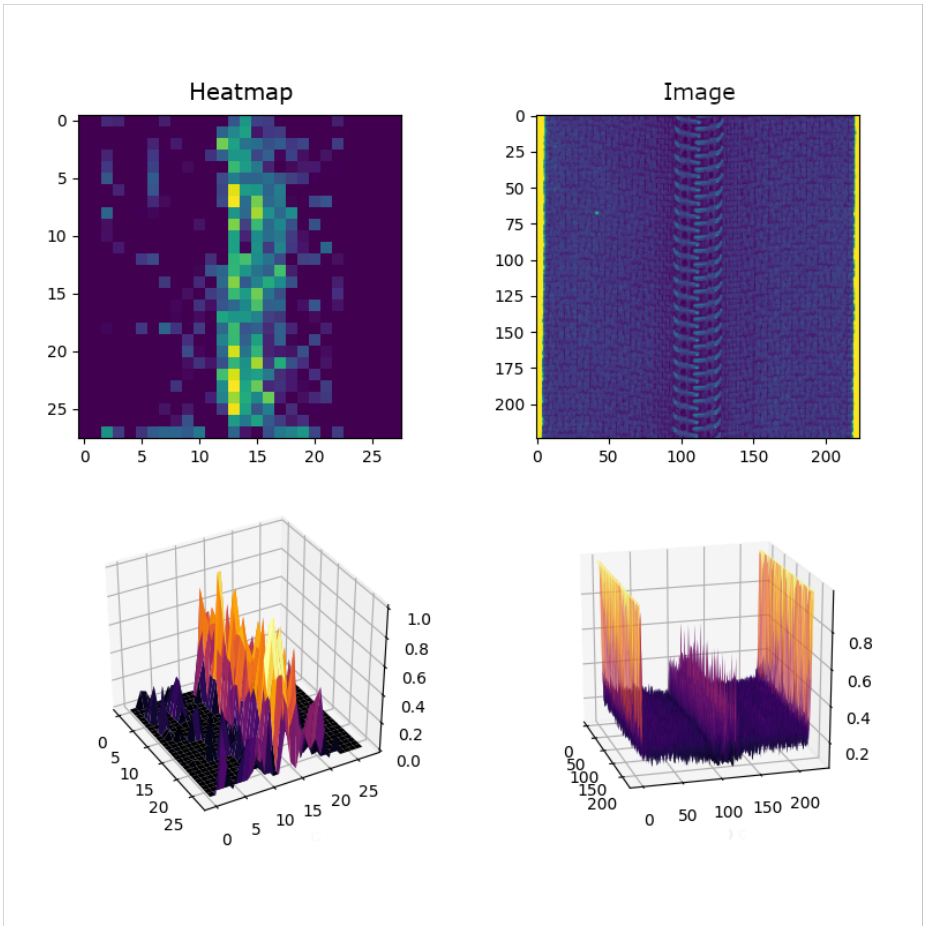


Figure 9.5: 2D and 3D heatmap (top left and bottom left) obtained with Grad-CAM from an image (top right and bottom right).

Algorithm 2 CEOD loss calculation and custom training loop

Input: $x, y, mask$

1: $output, feature\_tensor \leftarrow CNN(x, y)$
2: $heatmap \leftarrow GradCAM(output, feature\_tensor)$
3: $mask_{filtered} \leftarrow mask * kernel\left(\begin{smallmatrix} .5 & .5 & .5 \\ .5 & .5 & .5 \\ .5 & .5 & .5 \end{smallmatrix}\right)$
4: $OD \leftarrow OD(mask_{filtered}, heatmap)$
5: $loss \leftarrow crossentropy(output, y) + OD$
6: *Perform backpropagation*

the values 1 and 0, see the bottom right and bottom left 3D representations in Figure 9.6) but the AOI of the mask becomes more gradual, widening the AOI and allowing us to implement the distance computation so that it can be differentiated as the rest of the loss. The *distance* is important to help the network to understand when the detection is far or near the AOI. Following the example of Figure 9.7, focusing on the original mask (left side of 9.7) we can see that the detections marked with A and B have two different distances (Da and Db) and the distance of B (Db) from the AOI is larger than Da. If we use the original mask in the CEOD, these two detections give the same results because both A and B are multiplied by the same mask value which is zero. Instead, if we exploit the filtered masks, we can see that $A1$ is partially over the AOI, so its contribution in the calculation of the CEOD will be greater than $B1$. The closer the detection is to the AOI, the smaller the distance. Clearly, in case of overlap, the distance will be zero. This contribution leads the network to understand that the further the detection is from the AOI, the worse it is.

To make the OD part of the loss differentiable, the formulation of the OD became as follows:

$$OD = -\log\left[clip_{\epsilon}\left(\frac{\sum A_d A^*_{gt}}{min(\sum A_d, \sum A^*_{gt})}\right)\right] \tag{9.8}$$

where $\sum$ is calculated over the pixels of the images and represent the intersection $\cap$ displayed in Equations 9.3, 9.4 and 9.7, A^*_{gt} represent the convoluted mask and $clip_{\epsilon}$ is a function that clips the value of the Overlap Coefficient in the interval $[\epsilon, 1-\epsilon]$ to avoid the logarithm returning unacceptable values.

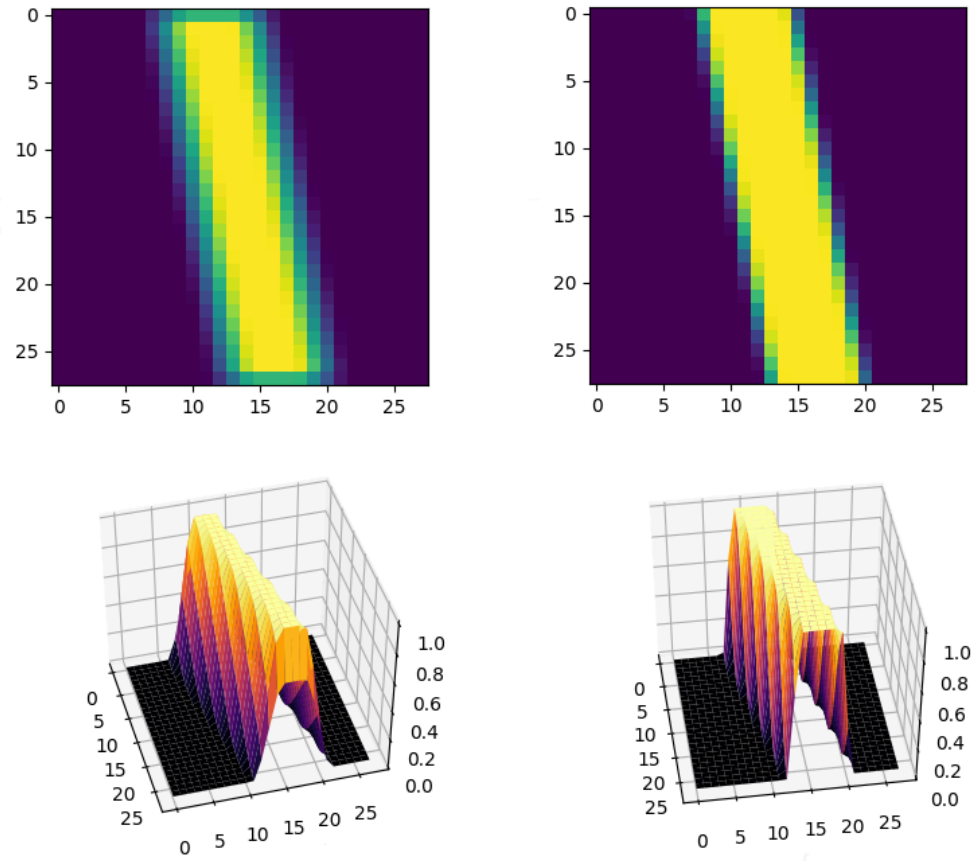


Figure 9.6: 2D and 3D representation of the filtered mask (top left and bottom left) and the mask inside the dataset (top right and bottom right).

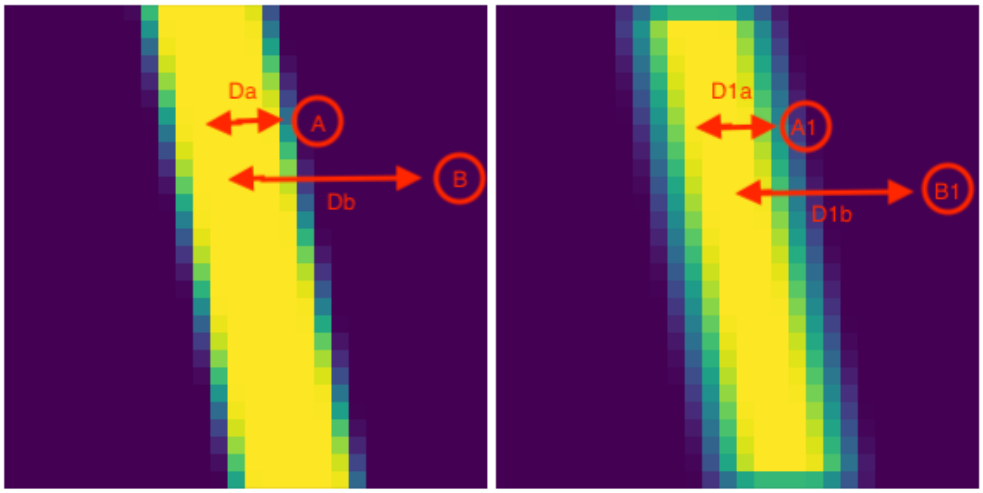


Figure 9.7: Original mask (left) and filtered mask (right) with two different detection: A, B, $A1$ and $B1$ respectively. Da, Db, $D1a$ and $D1b$ represent the distance between the detection and the AOI for the original and filtered mask respectively.

9.1.6 Related Work

In ML, anomaly detection has long been an issue of great interest, especially in Industry 4.0 where identifying defects is one of the major tasks of computer vision. Various articles survey anomaly detection in the literature [162, 163]. The aim of our work focuses primarily on the use of CNNs for structural defect detection during the monitoring of manufacturing line.

The vast majority of the CNN-based approaches have been used to study anomalies in the whole image area. Weimer D, et al. in [164] investigate CNNs in order to overcome the difficulties of manually redefining a specific feature representation for each new industrial inspection problem. In [165] the authors show how CNN with Triplet Loss [166] can be used to identify anomalies in the industrial environments.

In their study, Samet Akcay et al. [167], employ a Generative Adversarial Network (GAN) [168] for anomaly detection when the sample size of the classes of interest (usually the anomaly class) is insufficient to be modelled effectively. An and Cho in their study [169] use a VAE [170] for anomaly detection. In these methods, GAN (or VAE) learns the distribution of a certain class and use the difference between a reconstructed image and an input to detect the anomalies.

However, these approaches have proven effective in the reconstruction of simple anomalies.

In [171] Yong Moon et al. show the importance of using CAMs to check if the neural network focuses on the area of interest. The authors analyse the CNN architecture in detail using CAM images along with several evaluation metrics to optimize the CNN. Recently, path imaging has been shown to be effective in the segmentation and recognition of anomalies [172, 173]. In [174], the authors introduce the use of Grad-CAM to construct a self-supervised method to remove image noise for robust anomaly detection. Venkataramanan et al. [175] use the activation map to guide autoencoder training by reducing network attention in abnormal areas and increasing attention in normal areas in order to strengthen anomaly detection. Song et al. [176] propose an Anomaly Segmentation Network (AnoSeg) that can generate an anomaly map to segment the anomaly region.

Part IV

Applications of Explainable Deep Learning

Chapter 10

Symbolic NAS & HPO

This chapter presents the applications of Symbolic DNN-Tuner [91][1][2] to benchmarking and real-case datasets. Symbolic DNN-Tuner was tested on three different datasets: CIFAR10 [177], CIFAR100 [178] and CIMA, and compared to classic BO. On CIFAR10, the software was also compared with ENAS [116], DARTS [117] and Autokeras[3] which is one of the most widely used AutoML systems [122]. The ENAS experiments exploit the *micro* search space and *macro* search space [116]. All NAS algorithms were implemented with the Neural Network Intelligence (NNI)[4] toolkit provided by Microsoft, except Autokeras. In these experiments, CNNs were used as DNNs.

CIFAR10 contains 60,000 32x32 colour images divided in 10 classes. In this experiment we have used CIFAR10 with 50,000 training images and 10,000 validation images. CIFAR100 is the same of CIFAR10 but divided in 100 classes instead of 10. CIMA is a dataset provided by CIMA S.P.A[5] with 3,200 training images and 640 validation images of size 256x128. These images are facsimiles of Euro-like banknotes. CIMA has 16 classes that represent the denomination and orientation of the banknote (e.g., 5_front, 5_rear, 10_front, 10_rear, etc).

[1]Symbolic DNN-Tuner website: `https://ml.unife.it/software/symbolic-dnn-tuner/`

[2]Symbolic DNN-Tuner code: `https://github.com/micheleFraccaroli/Symbolic_DNN-Tuner`

[3]Autokeras website: https://autokeras.com/

[4]NNI website: https://www.microsoft.com/en-us/research/project/neural-network-intelligence/

[5]CIMA website: http://www.cima-cash-handling.com/it/

All experiments were performed on the GALILEO cluster provided by Cineca[6], equipped with Intel(R) Xeon(R) E5-2630 v3 @ 2.40GHz CPUs and Nvidia K80 GPUs. For each experiment, early stopping was set. Each experiment had a fixed duration of 8 hours. This means that Symbolic DNN-Tuner, BO and the NASs algorithms run for 8 hours consecutively at most.

10.1 Experiments on Benchmarking Datasets

The first experiment was performed using the CIFAR10 dataset. Symbolic DNN-Tuner and BO start with a neural network with two blocks composed by two convolutional layers, a max-pooling layer, a dropout and two fully connected layers separated by a dropout layer, the second fully connected layer is the output. The initial HPs to be set by the algorithm are the number of the neurons in the convolutional and fully connected layers, the values of the Dropout layers, the learning rate, the batch size, the activation functions and the optimizer. The size of the search space depends on the HPs. The domains of HPs are as follows. The size of the first and second convolutional layers are between 16 and 64 and between 64 and 128 respectively. The domains of the rate of dropout for the first and second convolutional layers are [0.002,0.3] and [0.03,0.5] respectively. The size of the fully connected layers is between 256 and 512. The domain of the learning rate is $[10^{-5},10^{-1}]$. The choice for the activation functions are: ReLU, ELU and Scaled Exponential Linear Unit (SELU). The choice for optimizers are: *Adam, Adamax, RMSprop* and *Adadelta*. For ENAS ([116] Sect. 3.2) and DARTS ([117] Appendix A.1.1), NNI uses the default search space in the original paper for CIFAR10 [177]. Autokeras starts with a three-layers CNN. Each convolutional layer is actually a convolutional block of a ReLU layer, a batch-normalization layer, the convolutional layer, and a pooling layer. All the convolutional layers are with kernel size equal to three, stride equal to one, and number of filters equal to 64 [122].

In Table 10.1 (in bold the best result in terms on validation accuracy) we show the results of the execution of Symbolic DNN-Tuner for every iteration: accuracy and loss in both phases, the diagnosis of this iteration and the TAs for the next iteration (i.e., the results of step 2 is the result of the application of TA

[6]Cineca website: https://www.cineca.it/

Table 10.1: CIFAR10 - Symbolic DNN-Tuner's results

Step	Training		validation		Diagnosis	TA
	Accuracy	Loss	Accuracy	Loss		
1	0.9315	0.1958	0.793	0.7004	overfitting	reg_l2 & batch_norm data_augm
					underfitting	inc_neurons
2	0.943	0.1648	0.8812	0.3658	overfitting	reg_l2 & batch_norm data_augm
					floating_loss	inc_batch_size
3	0.8715	0.365	0.8364	0.4968	underfitting	inc_neurons
					floating_loss	decr_lr
4	0.9565	0.1287	0.8578	0.4742	overfitting	reg_l2 & batch_norm data_augm
					floating_loss	decr_lr
5	0.8967	0.2946	0.8759	0.3711	floating_loss	decr_lr
6	0.8902	0.3129	0.8655	0.3916	floating_loss	decr_lr
7	0.9435	0.1678	0.8692	0.3900	overfitting	reg_l2 & batch_norm data_augm
					underfitting	inc_neurons
					floating_loss	decr_lr
8	0.9549	0.1348	0.8768	0.4021	overfitting	reg_l2 & batch_norm data_augm
					floating_loss	decr_lr
9	0.9699	0.0999	0.8731	0.408	overfitting	reg_l2 & batch_norm data_augm
					underfitting	inc_neurons
					floating_loss	decr_lr
10	**0.9741**	**0.07971**	**0.8836**	**0.3925**	**overfitting**	**reg_l2 & batch_norm data_augm**
					underfitting	**inc_neurons**
					floating_loss	**decr_lr**

Table 10.2: CIFAR10 - Bayesian Optimization's results

Step	Training		Validation	
	Accuracy	Loss	Accuracy	Loss
1	0.9762	0.06377	0.7776	1.046
2	0.09708	2.307	0.1	2.3078
3	0.1008	4.107	0.1	3.72
4	0.9133	0.2379	0.7655	0.9343
5	0.09874	2.303	0.1	2.303
6	0.7834	0.618	0.7404	0.7616
7	0.09842	2.303	0.1	2.303
8	0.1008	8.641	0.1	3.377
9	0.7599	0.6915	0.718	0.8171
10	**0.888**	**0.3272**	**0.7824**	**0.8229**
11	0.9609	0.1094	0.7758	1.036
12	0.6628	0.9813	0.6419	1.037
13	0.102	2.337	0.1	2.329
14	0.7411	0.7453	0.7327	0.7729
15	0.8633	0.378	0.7809	0.7518

Table 10.3: CIFAR10 - Comparison of algorithms

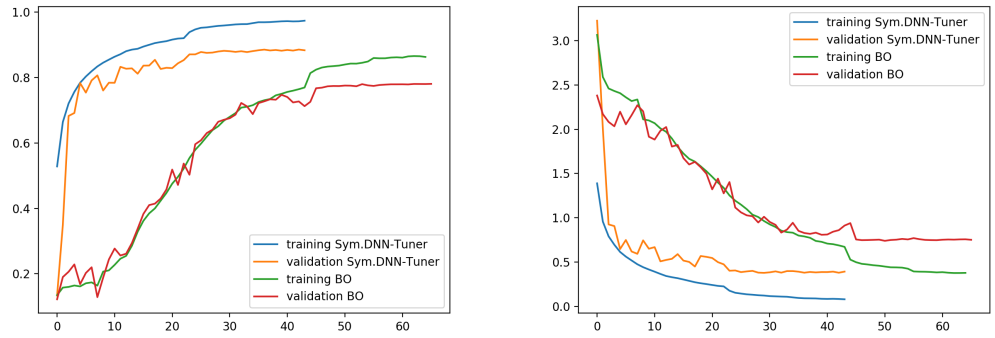
Algorithms	Val. accuracy	Val. loss
Symbolic DNN-Tuner	0.8836	0.3925
Bayesian Opt.	0.7824	0.8229
ENAS Macro search	0.4520	1.6540
ENAS Micro search	0.4887	1.6711
DARTS	0.9554	0.1526
Autokeras	0.9458	0.2507

of step 1). As can be seen, the TA `reg_l2 & batch_norm` to fix the overfitting is always applied regardless of other TA once overfitting is diagnosed. This is because using a regularization and a batch normalization in any case does not lead to worsening.

In Table 10.2 (in bold the best result in terms on validation accuracy) we show the results of the application of standard BO on the same network and same dataset of the previous experiment. Figure 10.1 compares the best results of Symbolic DNN-Tuner and BO on CIFAR10 graphically.

Table 10.3 show a recap of the experiments performed on the dataset CIFAR10. It compares Symbolic DNN-Tuner with standard BO, ENAS with both Macro and Micro search space, DARTS and Autokeras. Only DARTS and Autokeras outperform Symbolic DNN-Tuner in 8 hours of execution on CIFAR10.

From the CIFAR10 experiment, we can see that Symbolic DNN-Tuner does



(a) Accuracy (b) Loss

Figure 10.1: Best results of Symbolic DNN-Tuner and Bayesian Optimization on CIFAR10.

f

not always obtain the networks with the best performances but, unlike the BO and NAS methods, Symbolic DNN-Tuner guarantees an explanation of the actions it performs during its operation as shown in Table 10.1. At each iteration, we can see that a certain TA is applied to address a certain problem that is identified during the diagnosis phase.

The second experiment was performed on the CIFAR100 dataset. The initial network is the same as in the previous experiment. Tables 10.4 and 10.5 show the progression of Symbolic DNN-Tuner and BO respectively (in bold the best results in terms on validation accuracy), and Figure 10.2 compares the best results of Symbolic DNN-Tuner and BO graphically. Table 10.4 shows that the increase in the number of classes w.r.t. the number of images in the dataset leads to a worsening of the quality of the training in terms of accuracy and loss. By comparing Table 10.4 and Table 10.5 it can be seen how Symbolic DNN-Tuner outperform BO in terms of both accuracy and loss on CIFAR100.

10.2 Experiments on an Industrial Dataset

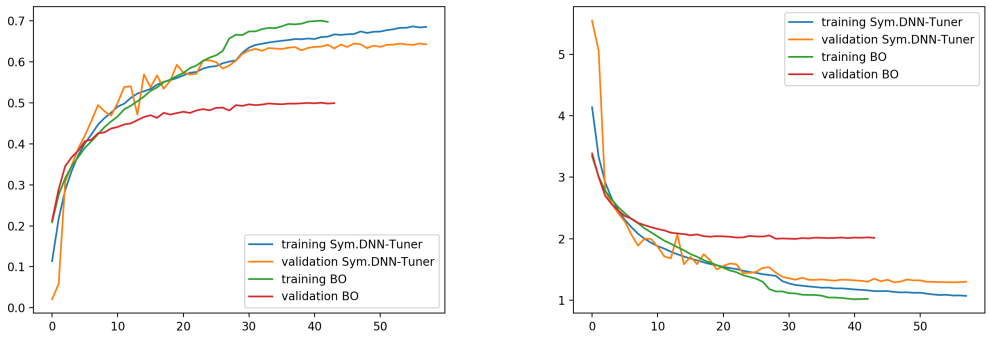
The last two experiments were performed on the CIMA dataset. This was used to test Symbolic DNN-Tuner on an industrial real case. To make this experiment as similar as possible to a production environment, the CIMA validation dataset has been degraded. In this experiment, in 8 hours, both

Table 10.4: CIFAR100 - Symbolic DNN-Tuner's results

Step	Training		validation		Diagnosis	STR
	Accuracy	Loss	Accuracy	Loss		
1	0.7121	1.039	0.4067	2.624	overfitting	reg_l2 & batch_norm
					underfitting	data_augm
						inc_neurons
2	0.7199	0.98	0.6238	1.443	overfitting	reg_l2 & batch_norm
					underfitting	data_augm
					floating_loss	inc_neurons
						inc_batch_size
3	0.7953	0.6752	0.6496	1.33	underfitting	inc_neurons
					underfitting	data_augm
					floating_loss	inc_neurons
						inc_batch_size
4	0.8353	0.5786	0.6056	1.548	overfitting	reg_l2 & batch_norm
					underfitting	data_augm
						inc_neurons
5	0.6656	1.157	0.6324	1.309	underfitting	inc_neurons
					floating_loss	inc_batch_size
6	**0.6852**	**1.068**	**0.6425**	**1.299**	**overfitting**	**reg_l2 & batch_norm**
					underfitting	**data_augm**
						inc_neurons
					floating_loss	**inc_batch_size**
7	0.663	1.156	0.6362	1.324	underfitting	inc_neurons
					floating_loss	inc_batch_size

Table 10.5: CIFAR100 - Bayesian Optimization's results

Step	Training		Validation	
	Accuracy	Loss	Accuracy	Loss
1	$9.62e-3$	5.628	$1e-2$	5.691
2	0.475	2.017	0.4065	2.391
3	$9.62e-3$	4.935	$1e-2$	4.768
4	$9.86e-3$	5.087	$1e-2$	5.035
5	0.4634	2.061	0.4247	2.26
6	0.4981	1.911	0.4107	2.365
7	0.9612	0.13	0.4487	3.446
8	$9.82e-3$	4.698	$1e-2$	4.685
9	0.8432	0.5023	0.4594	2.524
10	0.7943	0.7538	0.4216	2.633
11	0.8226	0.5973	0.3923	2.956
12	0.8304	0.5655	0.3869	3.729
13	**0.692**	**1.056**	**0.4901**	**2.079**
14	0.8143	0.6264	0.4472	2.692
15	0.9771	0.08094	0.4483	3.766

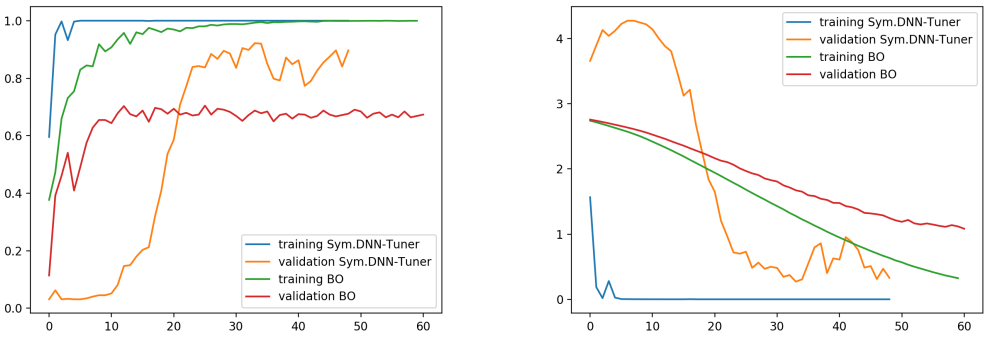


(a) Accuracy (b) Loss

Figure 10.2: Best results of Symbolic DNN-Tuner and Bayesian Optimization on CIFAR100.

Symbolic DNN-Tuner and BO have performed more than fifty trainings, then, for brevity, only the best trainings from the experiment will be shown for both systems.

In the first experiment on the CIMA dataset, both Symbolic DNN-Tuner and BO starts with a small neural network with only one convolutional block and, at the end of the network, two fully connected layers divided by a dropout layer, the second fully connected layer being the output. The results are shown in Figure 10.3. In this experiment, Symbolic DNN-Tuner starts with the network previously described and ends with a network with two convolutional blocks (two convolutional layers, a max-pooling layer and dropout at the end) and at the end of the network, four fully connected layers plus a dropout layer and a fully connected layer for the output. BO, instead, keeps the same network because is not able to modify the architecture of the network. The network obtained with Symbolic DNN-Tuner shows a behaviour in the first part of training that is more fluctuating than the one obtained with BO. After the 25th training cycle, we note how the network obtained by Symbolic DNN-Tuner shows a higher accuracy and a lower loss than that obtained by BO (Figure 10.3). In the second experiment on the CIMA dataset, Symbolic DNN-Tuner produces the same network as the first experiment. BO, instead, starts with a neural network with two blocks composed by two convolutional layers, a max-pooling layer, a dropout and two fully connected layers separ-

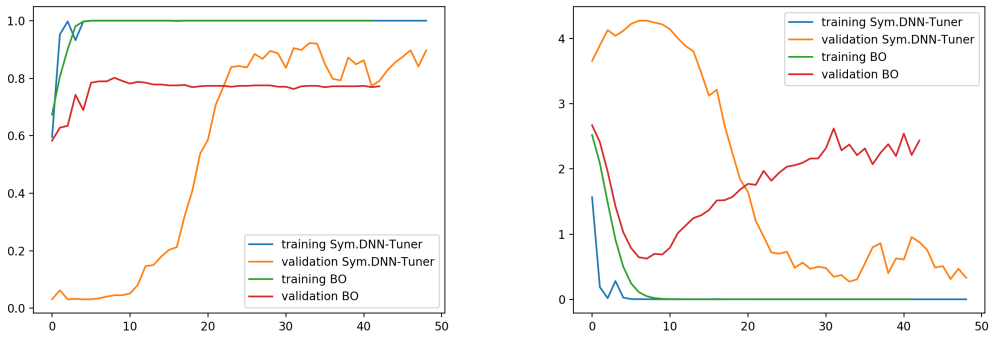


(a) Accuracy (b) Loss

Figure 10.3: Best results of Symbolic DNN-Tuner and Bayesian Optimization on the CIMA dataset in the first experiment.

ated by a dropout layer, the second fully connected layer being the output. The results are shown in Figure 10.4. As in the previous experiment, Symbolic DNN-Tuner shows a behaviour in the first part of training that is more fluctuating than the one obtained with BO. The network obtained by BO shows the best results around the 10th training cycle. After that, it begins to deteriorate showing evident signs of overfitting (Figure 10.4).

In all the performed experiments, it can be seen that Symbolic DNN-Tuner shows better results by analysing the performance of each network that it trains. See Figures 10.3 and 10.4.



(a) Accuracy (b) Loss

Figure 10.4: Best results of Symbolic DNN-Tuner and Bayesian Optimization on the CIMA dataset in the second experiment.

Chapter 11

Symbolic NAS in Tiny Machine Learning

This chapter explain the results of the application of Symbolic DNN-Tuner with the update described in Section 8.2 for creating and adapting DNNs to run on hardware with power and consumption limits. This new version of the framework has been compared with an ENAS [116] implemented in Autokeras [122].

11.1 Autokeras

Autokeras is an AutoML[179] system based on Keras [180]. This framework develops a neural network kernel and a tree-structured acquisition function optimization algorithm to efficiently explore the search space. Autokeras is an ENAS with network morphism [181] which utilizes BO [96, 97] to guide through the search space by selecting the most promising operations each time.

Many of the after-mentioned NAS approaches require a huge amount of time and computational power to reach good results. Also, each of the neural networks is trained from scratch which is very slow. Autokeras is an ENAS that integrates network morphism [181] and the key idea is to explore the search space via morphing the neural architectures guided by the BO algorithm. This is a technique for morphing the network architecture (e.g., inserting a layer or adding a skip-connection) and at the same time keeping its functionality. The

principal problem of the network morphism-based NAS methods is the selection of operations. This method is not quite efficient due to the amount of data necessary for the convergence of the algorithm [182] and for the inefficiency in exploring the large search space [183]. For this reason, Autokeras exploits BO for network morphism. In this context, the BO algorithm iteratively performs three steps: train the underlying Gaussian process model with the existing architectures and their performance, generate the next architecture to observe by optimizing a suitably defined acquisition function and obtain the actual performance by training the generated neural architecture.

In the context of this work, Autokeras has been modified to respect the constraints imposed by the hardware to ensure that the generated networks could fit with the hardware itself. As said in Section 8.2, as a hardware constraint, we use the number of FLOPS that the hardware can manage. The modification made to Autkeras consists in calculating the number of FLOPS of a new generated DNN and discarding it if it exceeds the FLOPS limit, forcing the algorithm to regenerate another. This process is iterated until a network is generated that respects the imposed limits.

11.2 Experiments

The experimental setup consists in 8 different experiments performed by each software. Each of these experiments is characterized by a different FLOPS threshold, that is: 10M, 30M, 40M, 80M, 90M, 100M, 110M and 120M FLOPS respectively. Then, we ran Symbolic DNN-Tuner and Autokeras 8 times, each with one of the different FLOPS thresholds. All experiments were performed on CIFAR-10 dataset [184]. This dataset contains 60000 32x32 colour images divided in 10 classes. In these experiments we have used CIFAR10 with 50000 training images and 10000 validation images. Both Autokeras and Symbolic DNN-Tuner start each experiment with a neural network consisting of two blocks composed of two convolutional layers and a last block with the two dense layers.

11.2.1 Results

The experiments show two very different behaviours between Autokeras and Symbolic DNN-Tuner. The first shows a much more accentuated exploration phase, creating and training neural networks that are very far from a possible optimal. This causes a considerable amount of time to train networks that will never reach acceptable performance (even if they are within the limits of the constraints). Symbolic DNN-Tuner, on the other hand, exhibits different behaviour. Apart from a first quick exploratory phase, the framework tends to focus almost immediately on an acceptable solution, optimizing it until satisfactory results are obtained. Figure 11.1 shows these behaviours. As can be seen from the comparison shown in Figure 11.1, it can be seen that the crosses (i.e., the iterations of Autokeras) are more sparse. Many of the executions of this framework are positioned in the lower part of the graph, showing results of low accuracy. Instead, Symbolic DNN-Tuner shows less sparsity. Once a value of FLOPS is found, optimized DNNs are generated with the same number of operations. The point cloud represented in Figure 11.1 aims to show the distribution of executions and training done by the two frameworks. Figure11.2 clarifies the results of the best accuracies achieved by the two frameworks for each FLOPS limit. As can be seen from Figure 11.2, the accuracy trend found by Symbolic DNN-Tuner exceeds that identified by Autokeras in the same FLOPS thresholds. Figure 11.3 shows the results of the execution times of the two frameworks. As you can see, Symbolic DNN-Tuner turns out to be the fastest system on every tested FLOPS limit. This is due to the Autokeras issue mentioned above. By generating and training networks that with high probability will never reach the optimum, the execution time of the framework is unnecessarily lengthened. This behaviour highlights one of the potentialities of Symbolic DNN-Tuner for which it was designed. Indeed, one of the important points of this system is the symbolic analysis that allows you to dynamically modify the search space. By dynamically excluding the HPs values that would take the optimization far from a possible optimum, it is possible to save time.

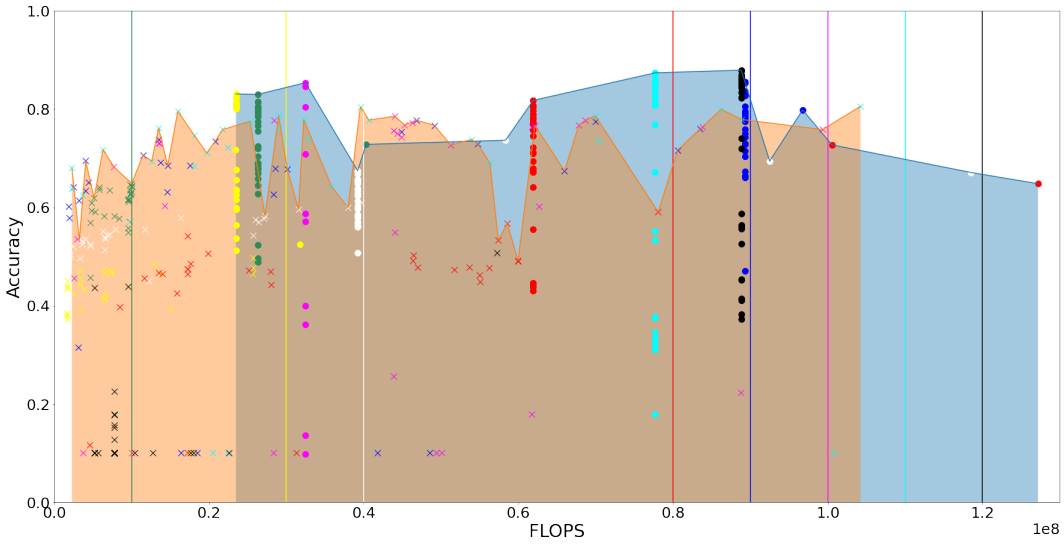


Figure 11.1: Comparison between Autokeras (orange area) and Symbolic DNN-Tuner (blue area) tested on different FLOPS thresholds. The x-axis represents the thresholds and the y-axis the accuracy achieved by the various models. The vertical lines represent the limits of the FLOPS set for each experiment. Autokeras and Symbolic DNN-Tuner results are marked with cross and dot respectively.

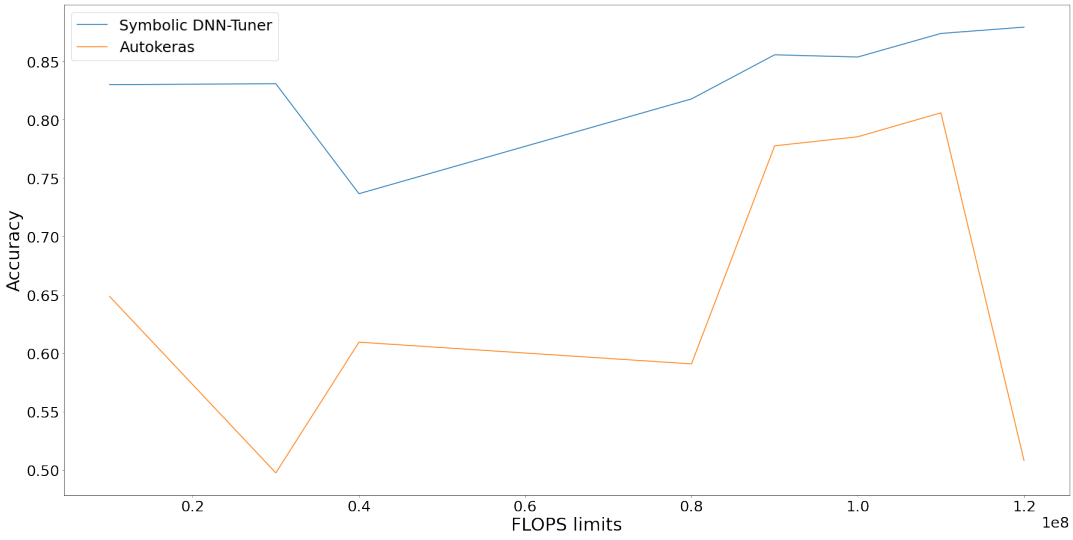


Figure 11.2: Comparison in accuracy between Autokeras (orange line) and Symbolic DNN-Tuner (blue line) tested on different FLOPS thresholds. The x-axis represents the thresholds and the y-axis the accuracy achieved by the best models.

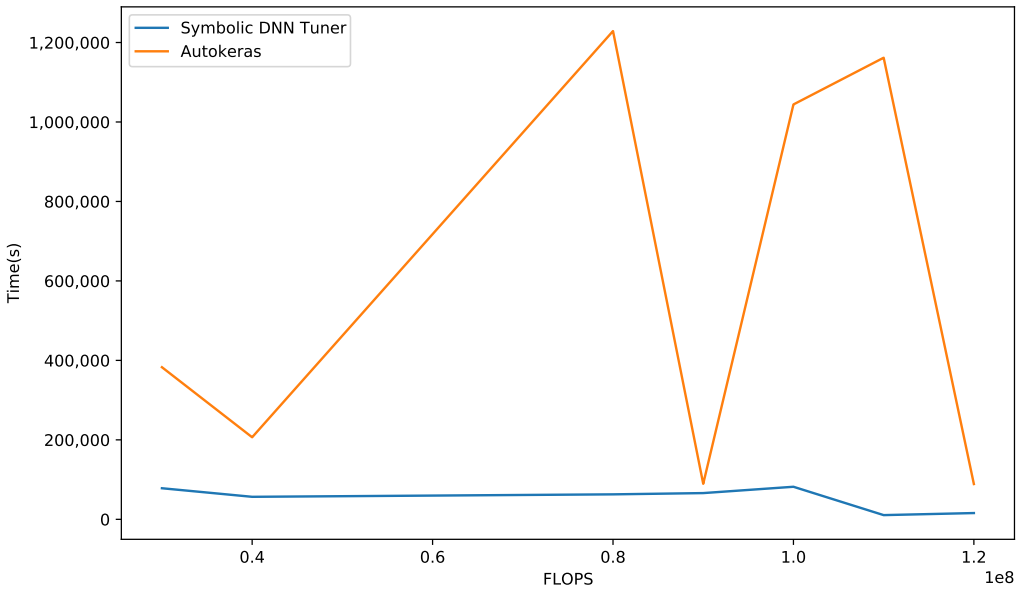


Figure 11.3: Time comparison between Autokeras (orange line) and Symbolic DNN-Tuner (blue line) tested on different FLOPS thresholds. The x-axis represents the thresholds and the y-axis the time taken by each framework to perform 30 iteration with the specific FLOPS constraint.

Table 11.1: Diagnosis & Tuning with 10M and 30M FLOPS respectively (acronyms of the tuning rules are available in Table 8.2 and Listing 8.5

10M FLOPS		
Step	Diagnosis	Tuning rule
1	reg_l2	overfitting
	data_augmentation	overfitting
	inc_lr	low_lr
	dec_layers	**latency**
2	inc_lr	low_lr
	inc_batch_size	floating_loss
	dec_neurons	**latency**
3	inc_lr	underfitting
	inc_batch_size	floating_loss
	dec_neurons	**latency**
4	inc_lr	underfitting
	inc_batch_size	floating_loss
	dec_neurons	**latency**
5	inc_lr	low_lr
	inc_batch_size	floating_loss
	dec_neurons	**latency**

30M FLOPS		
Step	Diagnosis	Tuning rule
1	inc_lr	low_lr
	dec_layers	**latency**
2	reg_l2	overfitting
	data_augmentation	overfitting
	inc_lr	low_lr
3	reg_l2	overfitting
	data_augmentation	overfitting
	decr_lr	underfitting
	inc_batch_size	floating_loss
4	decr_lr	underfitting
5	decr_lr	underfitting

11.2.1.1 Symbolic DNN-Tuner's Inner Results

In the context of this work, it is interesting to see the diagnosis performed by the symbolic analysis made by Symbolic DNN-Tuner and which tuning rules have been activated to solve the encountered problems. Tables 11.1, 11.2, 11.3, 11.4 show the diagnoses and TAs used by Symbolic DNN-Tuner for the eight experiments. In bold you can see every time that the symbolic analysis has detected latency, therefore that the threshold of the maximum FLOPS has been exceeded. As can be seen, also from the trend displayed by Figure 11.4, the exceeding of the FLOPS threshold occurs in experiments ranging from 10M to 90M FLOPS. From the 100M experiments onwards this phenomenon has not been detected anymore. This behaviour is also visible by analysing the placement and distribution of the Symbolic DNN-Tuner iterations in Figure 11.1. This behaviour, at the level of explainability, could also be interpreted as a level of measurement of the basic size (or capacity) of the DNN. This information tells us that networks with less than 10M FLOPS are not usable on this type of dataset or for this specific experiment. DNNs with more than 90 million FLOPS may work best for this task. Thus, we can tackle the problem of how to find the best dimensioning of a DNN given a dataset.

Table 11.2: Diagnosis & Tuning with 40M and 80M FLOPS respectively (acronyms of the tuning rules are available in Table 8.2 and Listing 8.5

40M FLOPS		
Step	Diagnosis	Tuning rule
1	reg_l2	overfitting
	data_augmentation	overfitting
	inc_lr	low_lr
	dec_layers	**latency**
2	decr_lr	underfitting
	inc_batch_size	floating_loss
	dec_layers	**latency**
3	decr_lr	underfitting
	dec_layers	**latency**
4	decr_lr	underfitting
	inc_batch_size	floating_loss
5	decr_lr	underfitting
	inc_batch_size	floating_loss

80M FLOPS		
Step	Diagnosis	Tuning rule
1	decr_lr	underfitting
	dec_layers	**latency**
2	reg_l2	overfitting
	data_augmentation	overfitting
	decr_lr	underfitting
	dec_layers	**latency**
3	decr_lr	underfitting
	inc_batch_size	floating_loss
	dec_layers	**latency**
4	decr_lr	underfitting
5	decr_lr	underfitting
	inc_neurons	underfitting

Table 11.3: Diagnosis & Tuning with 90M and 100M FLOPS respectively (acronyms of the tuning rules are available in Table 8.2 and Listing 8.5

90M FLOPS		
Step	Diagnosis	Tuning rule
1	reg_l2	overfitting
	data_augmentation	overfitting
	inc_lr	low_lr
	dec_layers	**latency**
2	reg_l2	overfitting
	data_augmentation	overfitting
	decr_lr	underfitting
	inc_batch_size	floating_loss
3	decr_lr	underfitting
	decr_lr	floating_loss
4	reg_l2	overfitting
	data_augmentation	overfitting
	decr_lr	floating_loss
5	decr_lr	underfitting
	decr_lr	floating_loss

100M FLOPS		
Step	Diagnosis	Tuning rule
1	decr_lr	underfitting
2	decr_lr	underfitting
	inc_batch_size	floating_loss
3	reg_l2	overfitting
	data_augmentation	overfitting
	decr_lr	underfitting
	inc_batch_size	floating_loss
4	decr_lr	underfitting
	inc_batch_size	floating_loss
5	reg_l2	overfitting
	data_augmentation	overfitting
	decr_lr	underfitting
	inc_batch_size	floating_loss

Table 11.4: Diagnosis & Tuning with 110M and 120M FLOPS respectively (acronyms of the tuning rules are available in Table 8.2 and Listing 8.5

110M FLOPS		
Step	Diagnosis	Tuning rule
1	reg_l2	overfitting
	data_augmentation	overfitting
	decr_lr	underfitting
2	reg_l2	overfitting
	inc_dropout	overfitting
	inc_lr	low_lr
	inc_neurons	underfitting
	inc_batch_size	floating_loss
3	reg_l2	overfitting
	inc_dropout	overfitting
	inc_neurons	underfitting
4	reg_l2	overfitting
	inc_dropout	overfitting
	inc_neurons	underfitting
	inc_batch_size	floating_loss
5	inc_lr	low_lr
	inc_neurons	underfitting

120M FLOPS		
Step	Diagnosis	Tuning rule
1	reg_l2	overfitting
	data_augmentation	overfitting
	dec_lr	underfitting
	inc_batch_size	floating_loss
2	reg_l2	overfitting
	inc_dropout	overfitting
	inc_neurons	underfitting
	decr_lr	floating_loss
3	reg_l2	overfitting
	inc_dropout	overfitting
	inc_neurons	underfitting
4	reg_l2	overfitting
	inc_dropout	overfitting
	decr_lr	floating_loss
	inc_neurons	underfitting
5	inc_lr	low_lr
	inc_neurons	underfitting

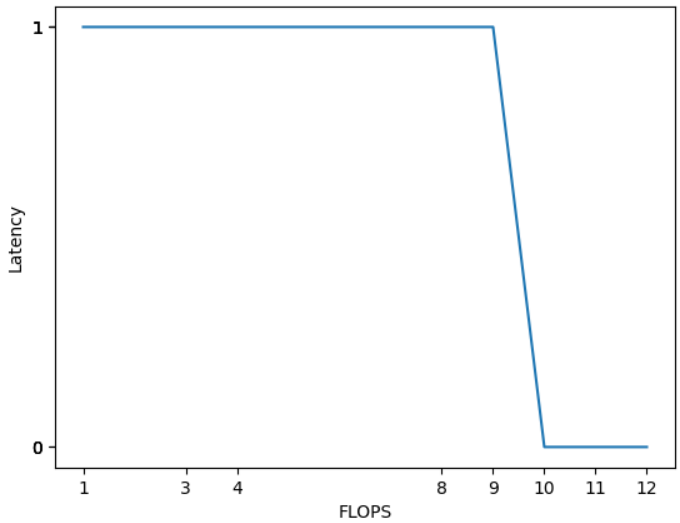


Figure 11.4: Trend of latency in the various experiments. On the x-axis there are the FLOPS thresholds and on the y-axis the presence of latency. 1 means that in the specific experiments, some created DNNs exceeded the FLOPS threshold.

Chapter 12

Neural HPLP in the Medical Environment

In this chapter we describe the application of Neural HPLP to the medical environment. In detail, the system aims to predicts if a Covid-19 patient arriving at the hospital will end up in a critical condition. The dataset used in this experiment is composed of two sub-datasets: clinical data and lung CT scans. A separate experiment was conducted on each of these. The result of the two experiments form the dataset for the last experiment conducted through HPLP. In Sections 12.1 and 12.2 we report the results of the application of Neural HPLP on two different experiments.

12.1 Early-Stage Prediction of Critical State of Covid-19 Patients

12.1.1 Experiments on Clinical Data

The clinical dataset used in this work was provided by an hospital in Ferrara, Italy. It contains data of 502 Covid-19 positive patients collected during spring 2020 of which 126 died during hospitalization. Thus, the dead patients correspond to about 25% of the whole dataset. Each patient in the dataset has 59 clinical attributes. In addition, 96 of these patients also had a CT scan. These 96 patients were kept as the test set. Of these 96 patients, 30 passed away during the hospitalization period. Table A.1 in Appendix A shows the clinical

attributes of each patient with acronyms. In order to balance data, Synthetic Minority Over sampling Technique (SMOTE) [185, 186] was applied. SMOTE selects a minority class instance and picks its n nearest neighbours belonging to the same class. A synthetic instance is then created by choosing one of the n nearest neighbours and connecting with the chosen real instance to form a line segment in the feature space. With SMOTE, we oversampled the class of dead patients since it corresponds to 25% of the dataset until parity is achieved for both classes.

The ML pipeline for predicting the probability of a patient's death during hospitalization is composed of a RF and a DT. We use a RF to analyse the dataset and extract the most important features from the initial clinical dataset. The DT is trained on the dataset obtained by keeping the most important features selected by RF. The result of the DT with its decision paths are used to create the dataset for the HPLP part of Neural HPLP. Both the RF and DT were trained exploiting grid search for finding the best combination of HPs values. The most relevant clinical attributes extracted by the RF are: Age, Sex, Glomerular Filtration Rate (GFR), C-reactive Protein (CRP), Troponin, Creatinine, Lactate Dehydrogenase (LDH), Brain Natriuretic Peptide (BNP), Procalcitonin (PCT), White Blood Cells (WBC) and Charlson Index, with an accuracy of $\sim 80\%$ on the test set. This result is in line with the work done by Yan li et al. [141] which states that LDH, lymphocytes and CRP are crucial predictive biomarkers of disease mortality with an accuracy of 90% on the test set.

After training a DT with the clinical attributes listed above, we achieved about 70% accuracy on the test set (i.e., on the 96 patients described at the beginning of this section).

12.1.2 Experiments on CT Scans

We used the *MosMedData* [187] CT scan dataset. It contains human lung CT scans with Covid-19 related findings, as well as without such findings. The CT scans were collected in 2020 and provided by the municipal hospital in Moscow, Russia. The dataset contains CT scans divided by the severity of lung tissue abnormalities with Covid-19. There are five classes: without injuries (CT-

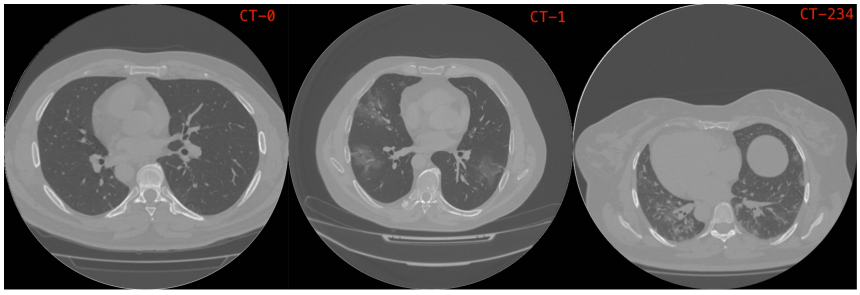


Figure 12.1: Example image of a slide of a DICOM voxel for the three classes of the dataset. We can see an example of image for class CT-0, CT-1 and CT-234 on the left, middle and right respectively.

0), with mild (CT-1), moderate (CT-2) , severe (CT-3) and critical injuries (CT-4) respectively. The dataset contains 254, 684, 125, 45 and 2 patients for the after-mentioned classes respectively. It can be observed that the dataset is unbalanced towards the CT-1 class, the mild injuries class. Due to the small numbers in the last three classes, we merged them into one obtaining the following distribution: CT-0 with 254 (22,8%), CT-1 with 684 (61,6%) and CT-234 with 172 (15,6%) patients respectively. Figure 12.1 shows an example image for each class. These classes correspond to three different level of severity of the lung injures that are: *healthy*, *minor* and *serious*. We used the CT scans of the 96 patients named previously as test set. All images in this dataset are in DICOM format. So, a CT scan in DICOM format can be seen as a set of consecutive images that form a 3D image. For this reason we used a convolutional neural network with 3D filters.

As mentioned in Section 8.3.1.2, the DL pipeline includes a pre-processing phase that consists in a segmentation task based on the HU scale and the training of a 3D-CNN on the segmented dataset. The network trained in this work is a 3D-CNN on the following form: two blocks composed of two 3D convolutional layers with $5 \times 5 \times 5$ kernels and a ReLU-like activation function, followed by a max pooling layer, with 98 and 160 neurons respectively. These two blocks are followed by two fully connected layers. The first with 110 neurons and the second is the output layer with 3 neurons corresponding to the three classes, CT-0, CT-1, CT-234. The 3D-CNN was trained and validated on the MosMedData dataset achieving $\sim 70\%$ accuracy on the validation set. It was also tested on the CT scans of the 96 patients described in the previous

sections achieving $\sim 54\%$ accuracy. This result is heavily conditioned by the small amount of CT scans in the test set.

12.1.3 Multimodal Learning with HPLP

12.1.3.1 Data Generation

As described in Section 8.3.1.3, the dataset for the training of the HPL program is composed by many *interpretations* that include facts describing the critical state that works as a label, the prediction of the death during hospitalization (that is the output of the DT), the lung's status (that is the output of the CNN) and the whole decision path of the DT to enrich the interpretation, see Example 1.

Example 1. *Consider the following interpretation that describes a Covid-19 patient with id 98:*

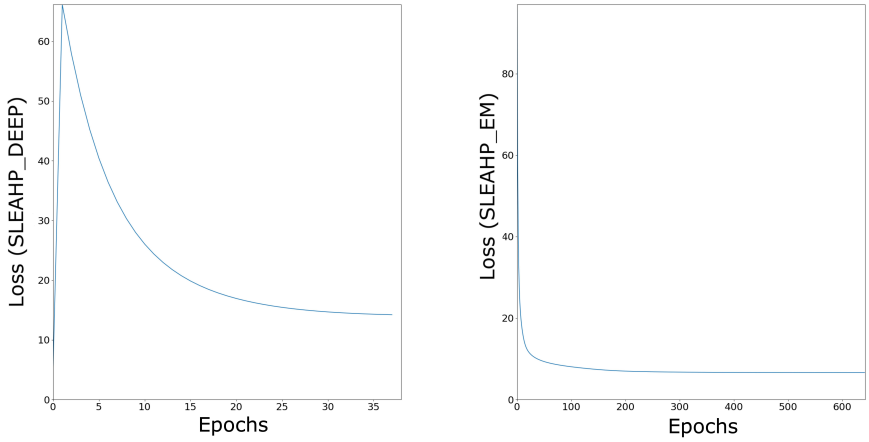
$$
\begin{aligned}
&critic(98).\\
&vital\_state(98, dead).\\
&lung\_injury(98, minor).\\
&age(98, 94).\\
&pcring(98, 13.59).\\
&ldhing(98, 71.89).\\
&troponina(98, 0.0).\\
&pcting(98, 403.0).
\end{aligned}
$$

where the first three facts indicate that the patient was labelled as in critical conditions, the DT classifies him/her as dead and the 3D-CNN classifies his/her lung as in a mild state. The other facts describe the body of the decision path applied by the DT to predict the vital state of the patient, $vital\_state(98, dead)$.

12.1.3.2 Experiments on Neural HPLP

After training the DT and the 3D-CNN, inference was performed on the corresponding test sets (96 patients) as described in the previous sections. Classifications on the test set for both the DT and the 3D-CNN were compared with those given by an expert in the domain, a radiologist in particular. According to the expert, 51 were correctly classified. We then built 51 interpretations using the procedure described in the previous section. Among the interpretations, 20 were labelled as in a critical state and 31 as in a non-critical state. Given the reduced amount of data, the training procedure was done using cross validation, i.e., the dataset was split into three folds with 17 interpretations in each fold. Every fold is balanced in terms of patients criticality. Interpretations in two folds are used for training and the remaining for testing. The procedure is repeated for the three crossed-combinations. Two versions of SLEAHP are applied: SLEAHP_DEEP, that uses Gradient Descent/Back-propagation (specifically with Adam optimizer) for learning the parameters, and SLEAHP_EM, that uses Expectation Maximization as parameter learning. Both versions were trained with L_2 regularisation [188, 189, 190, 191, 192] as described in [129] and, after learning, clauses annotated with probabilities below a certain threshold are dropped. We used 10^{-5} as threshold. Both algorithms were trained for 1000 iterations with early stop. The default Adam HP were used in SLEAHP_DEEP.

Since data used are unbalanced in both categories, we draw, for each test fold, the Receiver Operating Characteristic (ROC) and the Precision-Recall (PR) curves and compute the area under each curve (AUCROC and AUCPR) as described in [193]. The values of the areas, the final loss values and the associated average (over the folds) for both SLEAHP_DEEP and SLEAHP_EM are shown in Tables 12.1 and 12.2 respectively. While these systems provide high performance both in terms of AUCROC and AUCPR, it is worth noting that SLEAHP_EM performs better than SLEAHP_DEEP. The perfect result obtained in Fold 3 was due to the fact that the combination of data included in folds 1 and 2 used to for training was informative and enabled the algorithm to learn a better theory. It could also be observed that the value of the loss function for Fold 3 is better than the ones associated with



(a) SLEAHP_DEEP (b) SLEAHP_EM

Figure 12.2: SLEAHP_DEEP loss function trained using the first two folds and SLEAHP_EM loss function trained using the first two folds.

Fold 1 and Fold 2. It also is worth noting that SLEAHP_EM converges faster than SLEAHP_DEEP as observed in Figure 12.2.

Table 12.1: Areas under the curves and loss for SLEAHP_DEEP

	AUCROC	AUCPR	Loss
Fold 1	0.67347	0.80148	-10.80451
Fold 2	0.83333	0.90110	-8.99724
Fold 3	1.00000	1.00000	-5.57603
Average	**0.83560**	**0.90086**	**-8.45926**

An example of a learned theory is shown in Example 2. From it we can see that the feature *pcting* is one of the clinical attribute most useful to predict if a patient will end in a critical state. The first clause states that a Covid-19 patient is very likely to end in a critical state if his/her lungs are in a serious condition. This explanation is a clear consequence of the criteria for labelling interpretations defined in Section 12.1.3.1. Another interesting explanation can be observed using the combination of rules highlighted in bold: these rules state that if the troponina value of the Covid-19 patient is greater than 14.5, then the patient is very likely to end up in a critical state. Similar explanations can be observed for the other clinical attributes. Based on the present work,

Table 12.2: Areas under the curves and loss for SLEAHP_EM

	AUCROC	AUCPR	Loss
Fold 1	0.95918	0.95876	-4.64181
Fold 2	0.93750	0.94097	-5.45861
Fold 3	1.00000	1.00000	-4.17694
Average	**0.96556**	**0.9665766667**	**-4.75912**

doctors could pay more attention to these clinical values of a Covid-19 patient arriving at the hospital and improve their diagnosis and decision making using the learned explanations.

Example 2. *Learned rules for predicting the critical state of a Covid-19 patient*

$critic : 0.9983201826613162 : -lung\_injury(serious).$
$critic : 0.07404512050456119 : -troponina(B).$
$critic : 0.0031878498774003394 : -bnp(C), hidden\_3(C).$
$critic : 0.009686597460037139 : -age(D).$
$critic : 0.017233160198335595 : -pcring(E).$
$critic : 0.9999999999999978 : -pcting(F).$
$critic : 0.009842737641589272 : -gender\_f\_2(G), hidden\_8(G).$
$critic : 0.31365106628607303 : -age(H), hidden\_9(H).$
critic: 0.9441254441618012:- troponina(I),hidden_ 10(I).
$critic : 0.0037801053824951802 : -ldhing(J), hidden\_12(J).$
$critic : 0.0037629815686686108 : -charlsonindex(K), hidden\_13(K).$
$critic : 0.00843829776843874 : -age(L), hidden_1 4(L).$
$critic : 0.003175373172965623 : -pcring(M), hidden\_15(M).$
$critic : 0.025554356497234587 : -ldhing(N), hidden\_16(N).$
$critic : 0.009720608547637732 : -gender\_f\_2(O), hidden\_17(O).$
$critic : 0.20913363931689946 : -age(P), hidden\_18(P).$
$critic : 0.00023631655161687748 : -pcring(Q), hidden\_19(Q).$
$critic : 0.05738042137315996 : -ldhing(R).$
$critic : 0.010041915381422406 : -gender\_f\_2(S), hidden\_21(S).$
$hidden\_3(C) : 0.00323811617948383 : -greater\_than(C, 393.0).$
$hidden\_8(G) : 0.009921403222347414 : -greater\_than(G, 2.0).$
$hidden\_9(H) : 0.31230871757212836 : -greater\_than(H, 70.0).$

hidden_ 10(I) :0.9441254441618012:- greater_ than(I,14.5).
$hidden\_12(J) : 0.0037799297021227085 : -greater\_than(J, 101.87).$
$hidden\_13(K) : 0.0037630849919179643 : -greater\_than(K, 21.0).$
$hidden\_14(L) : 0.010897346883759151 : -greater\_than(L, 78.0).$
$hidden\_15(M) : 0.0031347295724000745 : -greater\_than(M, 22.79).$
$hidden\_16(N) : 2.911186251403075e-5 : -greater\_than(N, 54.37).$
$hidden\_17(O) : 0.0084854038369232 : -greater\_than(O, 2.0).$
$hidden\_18(P) : 0.20917254607034547 : -greater\_than(P, 85.0).$
$hidden\_19(Q) : 0.043254912362177045 : -greater\_than(Q, 7.82).$
$hidden\_21(S) : 0.009506938720927838 : -greater\_than(S, 2.0).$

12.2 Detection of the Covid-19 Infection

12.2.1 Dataset

The dataset used for the additional experiment was provided by the Huazhong University of Science and Technology [194], Wuhan, China and consists of 1521 patients of which 1126 from the Union Hospital (HUST-UH) and 395 from the Liyuan Hospital (HUST-LH). The dataset includes 894 Covid-19 positive patients and 627 non-Covid-19 patients. All patients had 120 clinical attributes, and 1342 subjects had both CT scans and clinical data. To perform the experiments, patients with normal CT scans (class Normal) and with lung lesions (class Pneumonia) are considered. More precisely, there are 1006 patients with pneumonia and 336 patients with normal lungs. All the examples in the dataset are in the DICOM format. In the experiment, for each image, individual slices were extracted and processed. More precisely, only part of the images containing the lungs was considered. Tables A.2 and A.3 in Appendix A list all clinical attributes. A total of 47260 2D images were obtained and used for the training of a CNN. The dataset, grouped by the patient, was divided into training (75%), validation (10%), and testing (15%) by sampling. Therefore, the test set includes 203 patients.

12.2.2 Experimental Setup

The trained CNN is composed of the following parts: four blocks composed of one convolutional layer with kernels of shape 3×3 and ReLU as activation function followed by a batch normalization layer with 64, 64, 128 and 256 neurons respectively. These blocks are followed by a global average pooling layer, one fully connected layer with 512 neurons and one dropout layer. The output layer consists of 2 neurons associated with the two classes, Normal, Pneumonia.

Regarding clinical data, an approach similar to the one applied in the previous experiment, described in Section 12.1.1, is adopted. The only difference is that the RF and DT were used to predict Covid-19 positive or negative patients instead of the death of a patient during hospitalisation.

12.2.3 Results

This section presents the results of Neural HPLP applied to the second dataset described in the previous section. This further experiment serves to confirm the reusability, the validity and more importantly the efficiency of Neural HPLP. As mentioned before, the target is to identify patients positive to Covid-19. First, we trained a RF on all clinical data classifying whether the patients are positive or negative to Covid-19. As results, we obtained an accuracy of 93.9%, an AUCROC of 0.93 and an AUPRC of 0.86. Then, using the trained RF, the first 10 (most important) clinical attributes are extracted and are the following: *Temperature*, *Coefficient variation of red cell volume distribution width*, *Standard deviation of red cell volume distribution width*, *Age*, *Lymphocyte count*, *Eosinophil percent*, *Eosinophil count*, *Neutrophil percent*, *Hemoglobin* and *Lymphocyte percent*. Considering only these features, a new dataset for training a DT is generated. After training the DT, the following metrics on the set are obtained: an *accuracy of 90.14%, an AUCROC of 0.9045 and an AUCPR of 0.9208*. Experiments on the trained CNN achieved the followings results on the test set: an accuracy of 81.77%, an AUCROC of 0.823 and an AUCPR of 0.8709.

The last part of the experiment is performed using SLEAHP_EM and SLEAHP_DEEP. Using the outcome of the DT and CNN, a dataset consisting

of 203 interpretations (one for each patient in the test set) is generated for training the SLEAHP systems. From the experiments, the following results are obtained: SLEAHP_DEEP achieved an AUCROC of 0.8188 and an AUCPR of 0.7210 while SLEAHP_EM achieved better results with an AUCROC of 0.8956 and an AUCPR of 0.8144. In summary, this last experiment on a consolidated dataset confirms the accuracy and more importantly the effectiveness of Neural HPLP.

Chapter 13

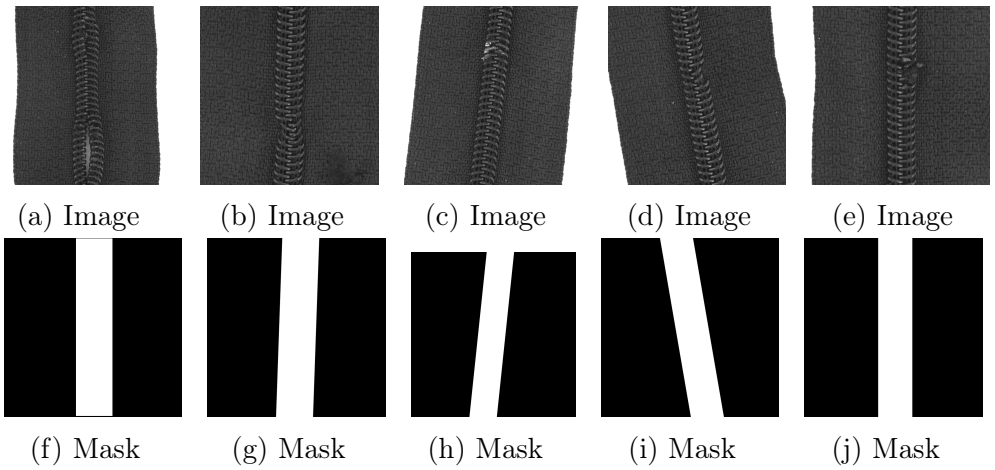
Cross Entropy Overlap Distance in Anomaly Detection

This chapter describe the experiments and the results obtained by applying the techniques used in Chapter 9 to benchmark datasets and industrial datasets.

13.1 Anomaly Detection Datasets

The experiments were performed on two different datasets. The first dataset is a sub-dataset of MVTec Anomaly Detection (AD) [1, 2], specifically, only the zipper images. This sub-dataset was augmented to change its shape and proportions. Figure 13.1 shows some sample images of this dataset. Each image has an associated binary mask. This dataset is composed of 216 *images without defects* and 184 *images with defects*. Then, in total, there are 400 images in the dataset. The training proportion is 70/20/10 for training, validation and testing. This dataset was chosen because it is representative of the problem in question. The images in this dataset have flaws found on both the zipper and the fabric on the outside of the zipper. To comply with the problem, we consider defective only the images that have defects on the zip. Then, all images with no flaws on the zipper but with flaws on the surrounding fabric were relabelled as non-defective.

The second dataset used in these experiments is provided by an Italian company. Unfortunately, it is not possible to give details about this dataset



(a) Image (b) Image (c) Image (d) Image (e) Image

(f) Mask (g) Mask (h) Mask (i) Mask (j) Mask

Figure 13.1: Example of the augmented zip dataset.

due to a Non-Disclosure Agreement signed with the company in the project. But this latter dataset is a real industrial dataset. This dataset is composed of 2818 *images without defects* and 2246 *images with defects*. In total, there are 5064 images in the dataset. The training configuration is always 80/20 for training and validation. The test set has 893 images: 467 *images without defects* and 426 *images with defects*.

For each experiment, EfficientNet-B0 was pre-trained on ImageNet. For our experiments, we have tested both transfer learning and fine-tuning. The experiments show that fine-tuning gives significantly better results than transfer learning. This is due to the fact that the network was trained on ImageNet with a standard loss (categorical cross-entropy). Therefore, re-training only the last dense layer may not be sufficient to be able to enhance the contribution of the new loss. For our experiments, we have performed the fine-tuning by unfreezing the last 20 convolutional layers. The custom CNN, instead, was trained from scratch by adopting the new proposed loss.

13.1.1 Experiments on MVTec AD Dataset

Table 13.1 shows the results of EfficientNet-B0 and the custom CNN trained on the MVTec AD dataset. As can be seen, in both cases, the application of the new OD can bring the networks to achieve better results in the validation phases. Figure 13.2 shows the results of EfficientNet-B0 on the test set. From

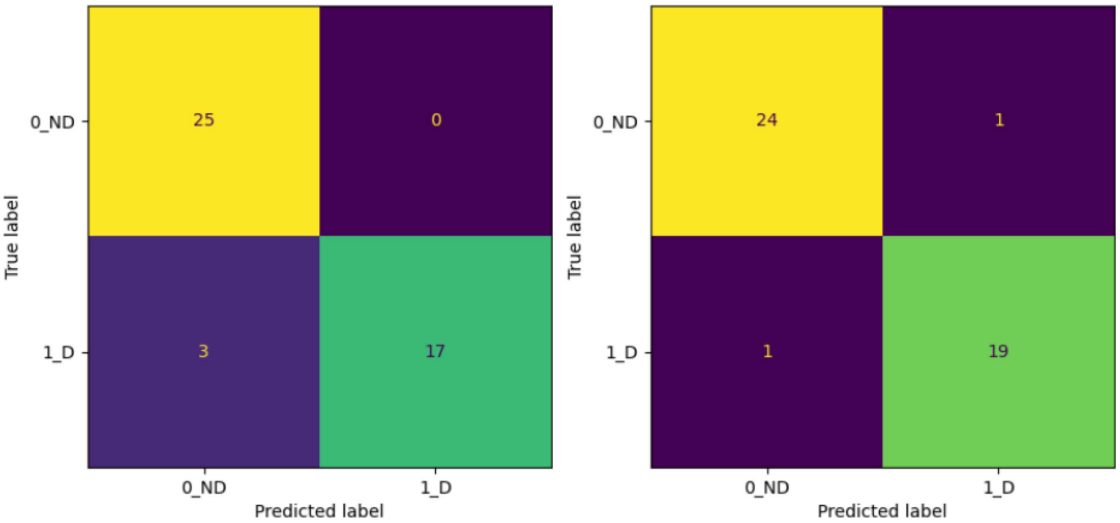
the confusion matrices, we can see that, by the application of OD on the loss (thus using the CEOD loss), the network obtains better results in terms of defect identification at the expense of a slight worsening in the identification of non-defective ones. The network trained with CEOD reaches 95.5% of accuracy and 0.95 of AUCROC. Indeed, EfficientNet-B0 with classical cross-entropy reaches 93.3% of accuracy and 0.925 of AUCROC. Figure 13.3 shows the results of custom CNN on the test set. The custom CNN trained with CEOD obtains 73.3% of accuracy and 0.74 of AUCROC in the same test set. The custom CNN trained with standard cross-entropy reaches 48.8% of accuracy and 0.53 of AUCROC. Figure 13.4 shows an example of a heatmap produced by a network trained classically and the one trained with CEOD. The time spent performing 1K training cycles with EfficientNet-B0 trained with standard cross-entropy is 2h 28m 24s. The time spent performing 1K training cycles with EfficientNet-B0 trained with CEOD is 2h 36m 29s. The time spent with the custom CNN to perform 100 training cycles with standard loss is 4m 30s versus 4m 28s for the custom CNN trained with CEOD.

13.1.2 Experiments on the Industrial Dataset

Table 13.2 shows the results of EfficientNet-B0 and the custom CNN on the industrial dataset. From Table 13.2, we can see that the network trained with CEOD is better in terms of accuracy in the validation phase. We can also note the improvement of the network trained with fine-tuning w.r.t. the network trained with only transfer learning. The slight worsening of the loss is probably due to the fact that, unlike the experiment done on the MVTec AD benchmark dataset, this industrial dataset presents much more difficulties. This is because it is representative of a real use case. The sum of the OD part

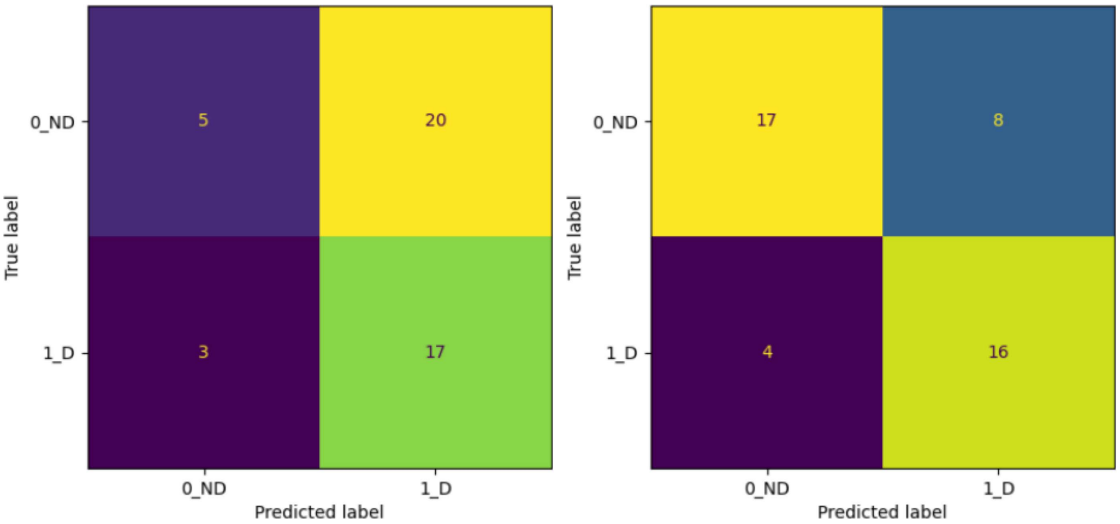
Table 13.1: MVTec AD Dataset.*Cls.* and *Exp.* stand for *classification* and *experiment* respectively

CNN	Exp.	Training			validation		
		Accuracy	Loss	OD	Accuracy	Loss	OD
EfficientNet-B0	Cls.	1.000	0.0049	-	0.992	0.022	-
	CEOD	0.990	0.005	0.0005	**1.00**	**0.010**	0.00032
CustomNet	Cls.	0.999	0.008	-	0.910	0.35	-
	CEOD	0.998	0.010	0.0002	**0.960**	**0.09**	0.0003



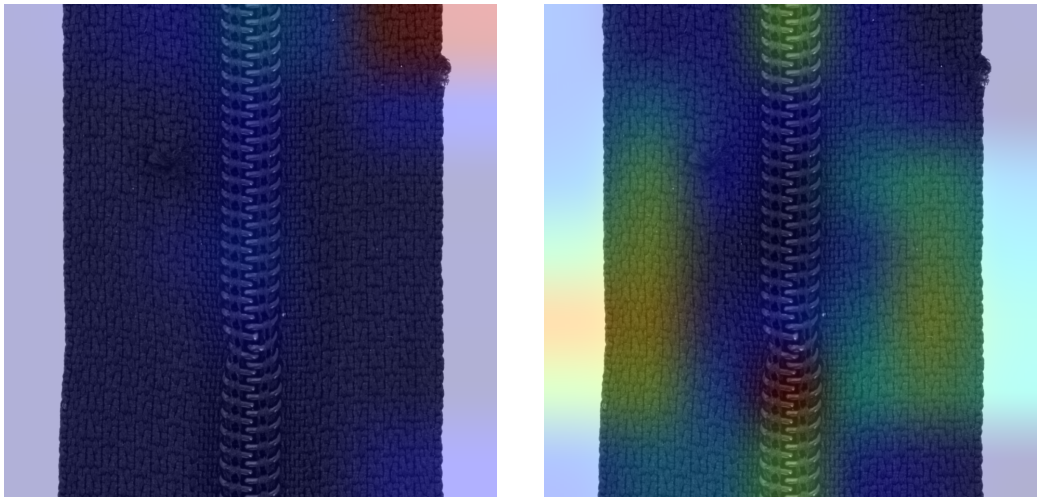
(a) Cls. (b) CEOD

Figure 13.2: Confusion matrices on the test set of the MVTec AD dataset obtained with EfficientNet-B0. *0_ND* and *1_D* represent the class without and with defects respectively.



(a) Cls. (b) CEOD

Figure 13.3: Confusion matrices on the test set of the MVTec AD dataset obtained with the custom CNN. *0_ND* and *1_D* represent the class without and with defects respectively.



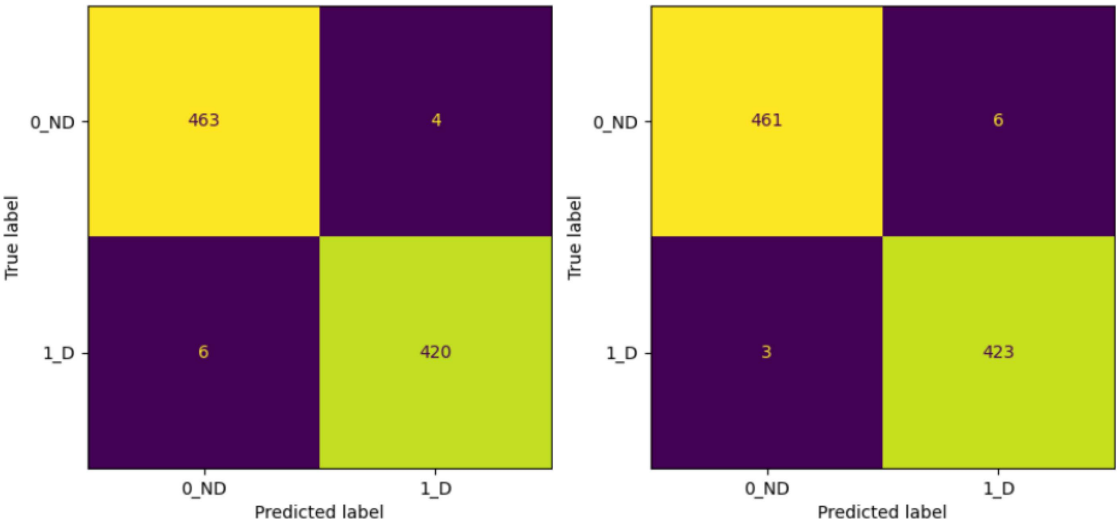
(a) Cls. (b) CEOD

Figure 13.4: In (a) there is the heatmap produced with the classification (Cls.) network. It is possible to see that the defect on the zip is not highlighted unlike the external defect in the upper right. In (b) there is the heatmap produced by the network trained with CEOD. In this case, the defect on the zip is highlighted unlike external defects which are not considered by the network.

to the cross entropy loss leads to this slight deterioration. This phenomenon does not occur in the MVTec AD dataset because, being simpler, the network trained with CEOD can clearly exceed that trained with the standard loss and therefore the effect of the sum of the ODs does not lead to worsening the loss. However, despite the excellent results obtained through the network trained for standard classification, with CEOD we are able to obtain a further increase in performance. Figures 13.5 and 13.6 show the results of EfficientNet-B0 and the custom CNN on the *test set* of the industrial dataset. In Figures 13.5 and 13.6, we can see that the networks trained with CEOD obtain better results in terms of defects identification at the expense of a slight worsening in the identification of non-defective ones. EfficientNet-B0 trained with CEOD reach 98.9% of accuracy and 0.99 of AUCROC compared to EfficientNet-B0 trained with standard cross-entropy that reaches 98.8% of accuracy and 0.98 of AUCROC on the test set. The custom CNN trained with CEOD reaches 95.4% of accuracy and 0.95 of AUCAUC as compared to the CNN trained with the standard loss that reaches 93.3% of accuracy and 0.93 of AUCROC on the test set. The time spent performing 360 training cycles with EfficientNet-B0

Table 13.2: Industrial Dataset. For EfficientNet-B0, results for Transfer Learning (TL) and Fine-Tuning (FT).

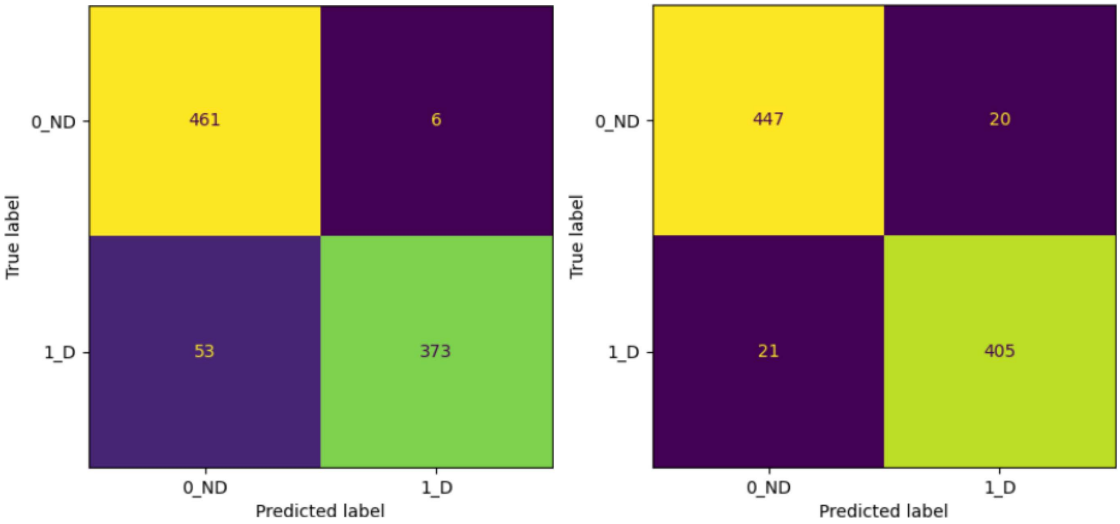
CNN	Exp.	Training			validation		
		Accuracy	Loss	OD	Accuracy	Loss	OD
EfficientNet-B0 - TL	Cls.	0.956	0.11	-	0.976	0.102	-
	CEOD	0.957	0.10	0.00055	**0.977**	**0.110**	0.00036
EfficientNet-B0 - FT	Cls.	0.990	0.00018	-	0.995	0.017	-
	CEOD	1.000	0.0007	0.00038	**0.998**	**0.045**	0.00032
CustomNet - FT	Cls.	1.000	0.00001	-	0.98	0.053	-
	CEOD	1.000	0.00002	0.000017	**0.9965**	**0.013**	0.00002



(a) Cls. (b) CEOD

Figure 13.5: Confusion matrices on the test set of the industrial dataset obtained with EfficientNet-B0. *0_ND* and *1_D* represent the class without and with defects respectively.

trained with standard cross-entropy is 9h 25m 31s. The time spent performing 360 training cycles with EfficientNet-B0 trained with CEOD is 8h 12m 47s. The time spent with the custom CNN to perform 80 training cycles with standard loss is 1h 44m 25s versus 1h 44m 5s for the custom CNN trained with CEOD.



(a) Cls. (b) CEOD

Figure 13.6: Confusion matrices on the test set of the industrial dataset obtained with the custom CNN. *0_ND* and *1_D* represent the class without and with defects respectively.

Part V

Conclusions and Outlooks

Chapter 14

Conclusions and Future Works

14.1 Conclusions

The work presented in this thesis is focused on the study and the exploitation of the concept of *eXplainable Artificial Intelligence* (XAI). In Part II we introduce the needed background knowledge for the correct understanding of the thesis. Part II describes the main deep learning concepts used in the rest of the thesis, the Neuro-Symbolic integration techniques used to combine deep networks with symbolic/logic human-interpretable algorithms and the background for the XAI concepts. Part III theoretically explains the various works developed in the contest of this dissertation. Finally, Part IV describes the experiments performed with the systems developed in Part III in different working scenarios.

In summary, in this work, we have developed two different approaches to XAI. All these approaches are widely described in Part III. The first approach, is described in Chapter 8. The work associated with this approach is a neuro-symbolic Neural Architecture Search (NAS) with Hyper-Parameter Optimization (HPO) algorithm. By exploiting Probabilistic Logic Programming (PLP), we can drive the training of a neural network by analysing the performance of each training and automatizing the choice of hyper-parameters to obtain a network with better performance. Thanks to PLP, we can get a diagnosis of the malfunctions and improvements that have been made to get to the final network. This system, called Symbolic DNN-Tuner, has been tested on both benchmark and industrial datasets obtaining good results compared with

Bayesian Optimization (BO) which is the state-of-the-art for hyper-parameters optimization. The second work belonging to this approach is Neural HPLP (Chapter 8.3). This is a neuro-symbolic ensemble learning system for learning from heterogeneous data streams in the medical environment. This system has been developed for facing the pandemic crisis caused by Covid-19. Neural HPLP combines deep networks with symbolic algorithms to learn from clinical data and unstructured data like CT scans or X-Ray and the results of these two different learnings are merged with Hierarchical Probabilistic Logic programming (HPLP). This is an approach for integrating symbolic (e.g probabilistic logic program) and sub-symbolic (neural networks) systems. The second approach described in Chapter 9 aims to exploit the explanation to improve the network performance. This approach is mainly focused on the task of anomaly detection. The idea of the work described in Chapter 9.1 is to create an attention module to focus a Convolutional Neural Network (CNN) only on the area of interest of the images. To achieve this goal, we identify the most important pixels in the images for classification according to a CNN through an XAI algorithm called Grad-CAM. After that, we calculate how much these pixels overlap with the mask that is provided, for each image, during the training phase. The computed overlap value is added to the loss of the network, to force the network to recognize the most important pixels, only within the area marked by the masks.

Part IV describes the applications of the after-mentioned approaches. In Chapter 10 and 11, we describe the application of Symbolic DNN-Tuner as NAS and HPO applied on benchmark datasets and industrial datasets and an updated version of the software for creating and adapting a neural network on a device with power and consumption constraints in a systematic way. Chapter 12 describes the application of Neural HPLP on two different tasks: the early-stage prediction of the critical state of Covid-19 patients and the detection of Covid-19 infection. Finally, Chapter 13 describes the application of the Cross Entropy Overlap Distance loss in the task of the classification and detection of anomalies on the surface of industrial products.

14.2 Future Work

The XAI field is a research area in constant growth. In these years, much work has been done on the theme of the explainability of Artificial Intelligence algorithms due to the importance that these algorithms are having in our lives. Despite this, this research field is still under-explored.

A future direction for Symbolic DNN-Tuner could be to remove the Bayesian Optimization (BO) from the optimization process, replacing it with other types of algorithms like Reinforcement Learning algorithms. Another future work is to advance the application of Symbolic DNN-Tuner for creating and adapting deep neural networks (DNNs) to an edge device with power constraints. The work described in this thesis concerning this field is the first step in this direction. A possible future step could be to add new constraints to be met. These constraints will be integrated as new tuning rules and the software will be compared with other recent software or software that will be developed in the context of Tiny Machine Learning (Tiny ML).

For Neural HPLP, we plan to build an end-to-end training process for Neural HPLP based on a customized optimization function. Furthermore, we also plan to integrate multiple other machine learning algorithms using hierarchical probabilistic logic programming. Finally, we plan to investigate the scalability of Neural HPLP by applying it to a large amount of clinical data obtained from different hospitals in different countries.

In the field of Exploiting Explanation (see Chapter 9), the next step is to try to apply this new type of loss to an unsupervised learning framework. We are studying a way to implement this approach into Generative Adversarial Networks (GANs) for anomaly detection. This is motivated by the high suitability of GANs for anomaly detection task in the industrial sector.

Appendix

Appendix A

Clinical Data

Table A.1 reports the list of the attributes of the first clinical dataset described in Section 12.1. Tables A.2 and A.3 report the list of attributes of the second dataset for the additional experiment described in the Section 12.2.

Table A.1: Attributes for the main experiment

Clinical attribute	Acronym
Age	-
Gender	-
Organization Cost Centre	CdcoUO
Intensification of care	-
Pneumology department	-
Anesthesia and resuscitation department	-
Clinical onset with fever	-
Hospitalization day	-
Discharge day	-
In-hospital days	-
Symptoms cardiopulmonary onset	-
Gastrointestinal onset symptoms	-
Systolic Blood Pressure at the entrance	SBP
Diastolic Blood Pressure at the entrance	DBP
Heart rate	-
Breath frequency	-
Body temperature	-
Modified Early Warning Score	MEWS
Partial pressure of oxygen in a gaseous environment	pO2
PO2 / FiO2 ratio	PF
High Resolution TC	HRTC
High Resolution TC per ground glass	HRTCpergrpoundglass
White blood cells	WBC
Lymphocytes	-
C-reactive Protein	CRP
Procalcitonin	PCT
Creatinine	-
Glomerular Filtration Rate	GFR
Lactate Dehydrogenase	LDH
Brain Natriuretic Peptide	BNP
Fibrinogen	-
D-Dimero	-
Isoamylase	-
Alanine Aminotransferase	ALT
Creatine Phosphokinase	CPK
Ferritin	-
Troponin	-
Smoking habit	-
Hypertension	-
Ischemic heart disease	-
Heart failure	-
IRCIIIIVV	-
ICTUSoTIA	-
Chronic Peripheral Obliterative Arteriopathy	AOCP
Chronic Obstructive Pulmonary Disease	COPD
Mild liver disease	-
Moderate liver disease	-
Peptic ulcer	-
AIDS	-
Hemiplegia	-
Localized or haematological neoplasm	-
Metastasis	-
Dementia	-
Charlson index	-
Microcythemia	-
Inflammatory Bowel Disease	IBD
Diabetes	-
Diabetes without organ damage	-
Diabetes with organ damage	-

Table A.2: Attributes for the additional experiment

Clinical attribute	Clinical attribute
Age	Alkaline phosphatase
Sex	Alanine aminotransferase
Temperature	Aspartate aminotransferase
malattie pregresse	Urea nitrogen
covid	Calcium
CT	Chlorine
Morbidity	Total carbon dioxide
Mortality	Creatinine
Erythrocyte sedimentation rate	Latitude-glutamyltransferase
C-reactive protein	Globulin
Procalcitonin	Potassium
Mean corpuscular hemoglobin concentration	Magnesium
Mean corpuscular hemoglobin	Sodium
Mean corpuscular volume	Phosphorus
Hematocrit	Total bilirubin
Hemoglobin	Serum total protein
Red blood cell	Uric acid
Platelet distribution width	Total cholesterol
Plateletcrit	Creatine kinase
Mean platelet volume	High density lipoprotein cholesterol
Platelet count	Lactate dehydrogenase
Basophil count	Triglyceride
Eosinophil count	Anion gap
Monocyte count	Direct bilirubin
Lymphocyte count	Glucose
Neutrophil count	Low density lipoprotein cholesterol
Basophil percent	Osmotic pressure
Eosinophil percent	Prealbumin
Monocyte percent	Total bile acids
Lymphocyte percent	Pseudo-hydroxybutyrate dehydrogenase
Neutrophil percent	Cystatin C
White blood cell	Leucine aminopeptidase
Platelet larger cell ratio	5'nucleotidase
Standard deviation of red cell volume distribution width	Homocysteine
Coefficient variation of red cell volume distribution width	Serum amyloid protein A
D-Dimer	Small density low density lipoprotein
Thrombin time	CD3+ T cell
Fibrinogen	CD4+ T cell
Activated partial thromboplastin time	CD8+ T cell
International normalization ratio	B lymphocyte
Prothrombin time	Natural killer cell
Albumin/Globulin ratio	CD4/CD8 ratio
Albumin	Interleukin-2
Interleukin-4	White blood cell count
Interleukin-6	Squamous epithelial cell
Interleukin-10	Viscose rayon
TNF-pseudo	Unclassified crystal
IFN-latitude	Specific gravity
Fibrin/fibrinogen degradation products	Complement C1q
Antithrombin III	Hyaline cast
B-type brain natriuretic peptide precursor	Pathological cast

Table A.3: Attributes for the additional experiment - continued

Clinical attribute	Clinical attribute
Indirect bilirubin	pH
Fungi (1-3)-tail-D-glucan	Complement C3
Urea	Immunoglobulin M
High-sensitivity C-reactive protein	Immunoglobulin A
Red blood cell count	Immunoglobulin G
Non-squamous epithelial cell	Yeast
Choline esterase	Complement C4
Sialic acid	Lipase
Pseudo-L-Fucosidase	Anti-streptolysin O
Lipoprotein A	Rheumatoid factor
Apolipoprotein A1	Bacterial count
Apolipoprotein B	Lactic acid
Leukocyte mass	

Bibliography

[1] Paul Bergmann, Kilian Batzner, Michael Fauser, David Sattlegger, and Carsten Steger. The mvtec anomaly detection dataset: a comprehensive real-world dataset for unsupervised anomaly detection. *International Journal of Computer Vision*, 129(4):1038–1059, 2021.

[2] Paul Bergmann, Michael Fauser, David Sattlegger, and Carsten Steger. Mvtec ad–a comprehensive real-world dataset for unsupervised anomaly detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9592–9600, 2019.

[3] S.B. Maind and P Wankar. Research paper on basic of artificial neural network. *International Journal on Recent and Innovation Trends in Computing and Communication*, 2:96–100, 01 2014.

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

[5] Allen Newell. Perceptrons. an introduction to computational geometry. marvin minsky and seymour papert. m.i.t. press, cambridge, mass., 1969. vi + 258 pp., illus. cloth, 12; *paper*,4.95. *Science*, 165(3895):780–782, 1969.

[6] Alex Sherstinsky. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network, 2018.

[7] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, Apr 1980.

[8] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, USA, 2010. Omnipress.

[9] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[10] Andre Esteva, Alexandre Robicquet, Bharath Ramsundar, Volodymyr Kuleshov, Mark DePristo, Katherine Chou, Claire Cui, Greg Corrado, Sebastian Thrun, and Jeff Dean. A guide to deep learning in healthcare. *Nature medicine*, 25(1):24–29, 2019.

[11] Mingyu Kim, Jihye Yun, Yongwon Cho, Keewon Shin, Ryoungwoo Jang, Hyun-jin Bae, and Namkug Kim. Deep learning in medical imaging. *Neurospine*, 16(4):657, 2019.

[12] Andre Esteva, Katherine Chou, Serena Yeung, Nikhil Naik, Ali Madani, Ali Mottaghi, Yun Liu, Eric Topol, Jeff Dean, and Richard Socher. Deep learning-enabled medical computer vision. *NPJ digital medicine*, 4(1):1–9, 2021.

[13] Qi Zhang, Yang Xiao, Wei Dai, Jingfeng Suo, Congzhi Wang, Jun Shi, and Hairong Zheng. Deep learning based classification of breast tumors with shear-wave elastography. *Ultrasonics*, 72:150–157, 2016.

[14] Heba Mohsen, El-Sayed A El-Dahshan, El-Sayed M El-Horbaty, and Abdel-Badeeh M Salem. Classification using deep learning neural networks for brain tumors. *Future Computing and Informatics Journal*, 3(1):68–71, 2018.

[15] Harsh Panwar, PK Gupta, Mohammad Khubeb Siddiqui, Ruben Morales-Menendez, and Vaishnavi Singh. Application of deep learning for fast detection of covid-19 in x-rays using ncovnet. *Chaos, Solitons & Fractals*, 138:109944, 2020.

[16] Shuo Wang, Yunfei Zha, Weimin Li, Qingxia Wu, Xiaohu Li, Meng Niu, Meiyun Wang, Xiaoming Qiu, Hongjun Li, He Yu, et al. A fully automatic deep learning system for covid-19 diagnostic and prognostic analysis. *European Respiratory Journal*, 56(2), 2020.

[17] Jong-Chih Chien, Ming-Tao Wu, and Jiann-Der Lee. Inspection and classification of semiconductor wafer surface defects using cnn deep learning networks. *Applied Sciences*, 10(15):5340, 2020.

[18] Hui Peng, Yifan Zhang, Sen Yang, and Bin Song. Battlefield image situational awareness application based on deep learning. *IEEE Intelligent Systems*, 35(1):36–43, 2019.

[19] Taisuke Sato and Keiichi Kubota. Viterbi training in prism. *Theory and Practice of Logic Programming*, 15(2):147–168, 2015.

[20] Fabrizio Riguzzi, Evelina Lamma, Marco Alberti, Elena Bellodi, Riccardo Zese, Giuseppe Cota, et al. Probabilistic logic programming for natural language processing. In *URANIA@ AI* IA*, pages 30–37, 2016.

[21] Arnaud Nguembang Fadja and Fabrizio Riguzzi. Probabilistic logic programming in action. In *Towards integrative machine learning and knowledge extraction*, pages 89–116. Springer, 2017.

[22] Søren Mørk and Ian Holmes. Evaluating bacterial gene-finding hmm structures as probabilistic logic programs. *Bioinformatics*, 28(5):636–642, 2012.

[23] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. Problog: A probabilistic prolog and its application in link discovery. In *IJCAI*, volume 7, pages 2462–2467. Hyderabad, 2007.

[24] Taisuke Sato and Yoshitaka Kameya. Prism: a language for symbolic-statistical modeling. In *IJCAI*, volume 97, pages 1330–1339, 1997.

[25] Wannes Meert, Jan Struyf, and Hendrik Blockeel. Cp-logic theory inference with contextual variable elimination and comparison to bdd based inference methods. In *International Conference on Inductive Logic Programming*, pages 96–109. Springer, 2009.

[26] Fabrizio Riguzzi. Speeding up inference for probabilistic logic programs. *The Computer Journal*, 57(3):347–363, 2014.

[27] Andrey Gorlin, CR Ramakrishnan, and Scott A Smolka. Model checking with probabilistic tabled logic programming. *Theory and Practice of Logic Programming*, 12(4-5):681–700, 2012.

[28] Taisuke Sato. A statistical learning method for logic programs with distribution semantics. In *In Proceedings of the 12th International Conference On Logic Programming (ICLP'95*. Citeseer, 1995.

[29] Fabrizio Riguzzi. *Foundations of Probabilistic Logic Programming*. River Publishers, 2018.

[30] Daan Fierens, Guy Van den Broeck, Joris Renkens, Dimitar Shterionov, Bernd Gutmann, Ingo Thon, Gerda Janssens, and Luc De Raedt. Inference and learning in probabilistic logic programs using weighted boolean formulas. *Theory and Practice of Logic Programming*, 15(3):358–401, 2015.

[31] Adnan Darwiche. Sdd: A new canonical representation of propositional knowledge bases. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[32] Anton Dries, Angelika Kimmig, Wannes Meert, Joris Renkens, Guy Van den Broeck, Jonas Vlasselaer, and Luc De Raedt. Problog2: Probabilistic logic programming. In *Joint european conference on machine learning and knowledge discovery in databases*, pages 312–315. Springer, 2015.

[33] Bernd Gutmann, Ingo Thon, and Luc De Raedt. Learning the parameters of probabilistic logic programs from interpretations. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 581–596. Springer, 2011.

[34] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

[35] Sebastian Bader and Pascal Hitzler. Integration - a structured survey. *ArXiv*, abs/cs/0511042, 2005.

[36] Ivan Donadello, Luciano Serafini, and Artur D'Avila Garcez. Logic tensor networks for semantic image interpretation. *arXiv preprint arXiv:1705.08968*, 2017.

[37] Artur SD'Avila Garcez, Luis C Lamb, and Dov M Gabbay. *Neural-symbolic cognitive reasoning*. Springer Science & Business Media, 2008.

[38] Artur d'Avila Garcez, Marco Gori, Luis C Lamb, Luciano Serafini, Michael Spranger, and Son N Tran. Neural-symbolic computing: An effective methodology for principled integration of machine learning and reasoning. *arXiv preprint arXiv:1905.06088*, 2019.

[39] Geoffrey G Towell and Jude W Shavlik. Knowledge-based artificial neural networks. *Artificial intelligence*, 70(1-2):119–165, 1994.

[40] Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard Hovy, and Eric Xing. Harnessing deep neural networks with logic rules. *arXiv preprint arXiv:1603.06318*, 2016.

[41] Luciano Serafini and Artur S d'Avila Garcez. Learning and reasoning with logic tensor networks. In *Conference of the Italian Association for Artificial Intelligence*, pages 334–348. Springer, 2016.

[42] Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. Reasoning with neural tensor networks for knowledge base completion. *Advances in neural information processing systems*, 26, 2013.

[43] Giuseppe Marra, Francesco Giannini, Michelangelo Diligenti, and Marco Gori. Lyrics: a general interface layer to integrate ai and deep learning. *arXiv preprint arXiv:1903.07534*, 2019.

[44] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Broeck. A semantic loss function for deep learning with symbolic knowledge. In *International conference on machine learning*, pages 5502–5511. PMLR, 2018.

[45] Manoel VM França, Gerson Zaverucha, and Artur S d'Avila Garcez. Fast relational learning using bottom clause propositionalization with artificial neural networks. *Machine learning*, 94(1):81–104, 2014.

[46] Richard Evans and Edward Grefenstette. Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research*, 61:1–64, 2018.

[47] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. *Advances in Neural Information Processing Systems*, 31, 2018.

[48] Gustav Sourek, Vojtech Aschenbrenner, Filip Zelezny, Steven Schockaert, and Ondrej Kuzelka. Lifted relational neural networks: Efficient learning of latent relational structures. *Journal of Artificial Intelligence Research*, 62:69–100, 2018.

[49] Simon Odense and Artur d'Avila Garcez. Extracting m of n rules from restricted boltzmann machines. In *International Conference on Artificial Neural Networks*, pages 120–127. Springer, 2017.

[50] Ridwan Al Iqbal. Eclectic extraction of propositional rules from neural networks. In *14th International Conference on Computer and Information Technology (ICCIT 2011)*, pages 234–239. IEEE, 2011.

[51] Kazumi Saito and Ryohei Nakano. Medical diagnostic expert system based on pdp model. In *ICNN*, pages 255–262, 1988.

[52] Enric Junque De Fortuny and David Martens. Active learning-based pedagogical rule extraction. *IEEE transactions on neural networks and learning systems*, 26(11):2664–2677, 2015.

[53] Geoffrey Towell and Jude Shavlik. Interpretation of artificial neural networks: Mapping knowledge-based neural networks into rules. *Advances in neural information processing systems*, 4, 1991.

[54] Clayton McMillan, Michael C Mozer, and Paul Smolensky. Rule induction through integrated symbolic and subsymbolic processing. *Advances in neural information processing systems*, 4, 1991.

[55] C Lee Giles and Christian W Omlin. Rule refinement with recurrent neural networks. In *IEEE International Conference on Neural Networks*, pages 801–806. IEEE, 1993.

[56] Mark Craven and Jude Shavlik. Extracting tree-structured representations of trained networks. *Advances in neural information processing systems*, 8, 1995.

[57] R Krishnan, G Sivakumar, and P Bhattacharya. Extracting decision trees from trained neural networks. *Pattern recognition*, 32(12), 1999.

[58] Alan B Tickle, Marian Orlowski, and Joachim Diederich. Dedec: decision detection by rule extraction from neural networks. *QUT NRC*, 1994.

[59] Shu Guo, Quan Wang, Lihong Wang, Bin Wang, and Li Guo. Jointly embedding knowledge graphs and logical rules. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pages 192–202, 2016.

[60] Travis R Goodwin and Dina Demner-Fushman. Bridging the knowledge gap: Enhancing question answering with world and domain knowledge. *arXiv preprint arXiv:1910.07429*, 2019.

[61] Àngel Garcı, Eva Armengol, Francesc Esteva, et al. Fuzzy description logics and t-norm based fuzzy logics. *International Journal of Approximate Reasoning*, 51(6):632–655, 2010.

[62] David Gunning. Explainable artificial intelligence (xai). *Defense advanced research projects agency (DARPA), nd Web*, 2(2):1, 2017.

[63] Erico Tjoa and Cuntai Guan. A survey on explainable artificial intelligence (xai): Toward medical xai. *IEEE transactions on neural networks and learning systems*, 32(11):4793–4813, 2020.

[64] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-López, Daniel Molina, Richard Benjamins, et al. Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information fusion*, 58:82–115, 2020.

[65] Raymond E Wright. Logistic regression. 1995.

[66] J Ross Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.

[67] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[68] Giulia Vilone and Luca Longo. Classification of explainable artificial intelligence methods through their output formats. *Machine Learning and Knowledge Extraction*, 3(3):615–661, 2021.

[69] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

[70] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.

[71] Lloyd S Shapley. A value for n-person games. *Classics in game theory*, 69, 1997.

[72] Paulo Cortez and Mark J Embrechts. Opening black box data mining models using sensitivity analysis. In *2011 IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*, pages 341–348. IEEE, 2011.

[73] Alex Goldstein, Adam Kapelner, Justin Bleich, and Emil Pitkin. Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation. *journal of Computational and Graphical Statistics*, 24(1):44–65, 2015.

[74] Pedro Domingos. Knowledge discovery via multiple models. *Intelligent Data Analysis*, 2(1-4):187–202, 1998.

[75] Satoshi Hara and Kohei Hayashi. Making tree ensembles interpretable. *arXiv preprint arXiv:1606.05390*, 2016.

[76] Todd K Moon. The expectation-maximization algorithm. *IEEE Signal processing magazine*, 13(6):47–60, 1996.

[77] James M Joyce. Kullback-leibler divergence. In *International encyclopedia of statistical science*, pages 720–722. Springer, 2011.

[78] Leo Breiman, Jerome H Friedman, Richard A Olshen, and Charles J Stone. *Classification and regression trees*. Routledge, 2017.

[79] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013.

[80] Xiao Li, Yu Wang, Sumanta Basu, Karl Kumbier, and Bin Yu. A debiased mdi feature importance measure for random forests. *Advances in Neural Information Processing Systems*, 32, 2019.

[81] Grégoire Montavon, Sebastian Lapuschkin, Alexander Binder, Wojciech Samek, and Klaus-Robert Müller. Explaining nonlinear classification decisions with deep taylor decomposition. *Pattern recognition*, 65:211–222, 2017.

[82] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *International conference on machine learning*, pages 3319–3328. PMLR, 2017.

[83] Pieter-Jan Kindermans, Kristof T Schütt, Maximilian Alber, Klaus-Robert Müller, Dumitru Erhan, Been Kim, and Sven Dähne. Learning how to explain neural networks: Patternnet and patternattribution. *arXiv preprint arXiv:1705.05598*, 2017.

[84] Matthew D Zeiler, Graham W Taylor, and Rob Fergus. Adaptive deconvolutional networks for mid and high level feature learning. In *2011 international conference on computer vision*, pages 2018–2025. IEEE, 2011.

[85] Matthew D Zeiler, Dilip Krishnan, Graham W Taylor, and Rob Fergus. Deconvolutional networks. In *2010 IEEE Computer Society Conference on computer vision and pattern recognition*, pages 2528–2535. IEEE, 2010.

[86] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[87] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140, 2015.

[88] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017.

[89] Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5188–5196, 2015.

[90] Anh Nguyen, Alexey Dosovitskiy, Jason Yosinski, Thomas Brox, and Jeff Clune. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. *Advances in neural information processing systems*, 29, 2016.

[91] Michele Fraccaroli, Evelina Lamma, and Fabrizio Riguzzi. Symbolic dnn-tuner. *Machine Learning*, 111(2):625–650, 2022.

[92] Michele Fraccaroli, Evelina Lamma, and Fabrizio Riguzzi. Symbolic dnn-tuner: a python and problog-based system for optimizing deep neural networks hyperparameters. *SoftwareX*, 17:100957, 2022.

[93] Dhanesh Ramachandram and Graham W Taylor. Deep multimodal learning: A survey on recent advances and trends. *IEEE signal processing magazine*, 34(6):96–108, 2017.

[94] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554, 2011.

[95] Grégoire Montavon, Geneviève Orr, and Klaus-Robert Múller. *Neural networks: tricks of the trade*, volume 7700. springer, 2012.

[96] Ian Dewancker, Michael McCourt, and Scott Clark. Bayesian optimization primer, 2015.

[97] Peter I Frazier. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

[98] Ali Jalali, Javad Azimi, and Xiaoli Z. Fern. Exploration vs exploitation in bayesian optimization. *CoRR*, abs/1204.0047, 2012.

[99] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer School on Machine Learning*, pages 63–71. Springer, 2003.

[100] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

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

[102] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

[103] Kian Katanforoosh Andrew Ngn Younes Bensouda Mourri. Improving deep neural networks: Hyperparameter tuning, regularization and optimization. `https://www.coursera.org/learn/deep-neural-network`.

[104] Twan van Laarhoven. L2 regularization versus batch and weight normalization. *CoRR*, abs/1706.05350, 2017.

[105] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[106] 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.

[107] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):60, 2019.

[108] Minghui Ou, Hua Wei, Yiyi Zhang, and Jiancheng Tan. A dynamic adam based deep neural network for fault diagnosis of oil-immersed power transformers. *Energies*, 12(6):995, 2019.

[109] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377*, 2018.

[110] Tong Yu and Hong Zhu. Hyper-parameter optimization: A review of algorithms and applications. *arXiv preprint arXiv:2003.05689*, 2020.

[111] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13(1):281–305, 2012.

[112] Korichi, Mathilde Guillemot, Catherine Heusèle, and Rodolphe. Tuning neural network hyperparameters through bayesian optimization and application to cosmetic formulation data. In *ORASIS 2019*, 2019.

[113] Hadrien Bertrand, Roberto Ardon, Matthieu Perrot, and Isabelle Bloch. Hyperparameter optimization of deep neural networks: Combining hyperband with bayesian model selection. In *Conférence sur l'Apprentissage Automatique*, 2017.

[114] E. Real, A. Aggarwal, Y. Huang, and Quoc V. Le. Aging evolution for image classifier architecture search. In *AAAI 2019*, 2019.

[115] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.

[116] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

[117] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

[118] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. *arXiv preprint arXiv:1904.00420*, 201.

[119] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018.

[120] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019.

[121] Matthias Feurer and Frank Hutter. Towards further automation in automl. In *ICML AutoML workshop*, page 13, 2018.

[122] Haifeng Jin, Qingquan Song, and Xia Hu. Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1946–1956, 2019.

[123] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019.

[124] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture

search. In *International Conference on Machine Learning*, pages 550–559, 2018.

[125] Scott Mayer McKinney, Marcin Sieniek, Varun Godbole, Jonathan Godwin, Natasha Antropova, Hutan Ashrafian, Trevor Back, Mary Chesus, Greg S Corrado, Ara Darzi, et al. International evaluation of an ai system for breast cancer screening. *Nature*, 577(7788):89–94, 2020.

[126] Diego Ardila, Atilla P Kiraly, Sujeeth Bharadwaj, Bokyung Choi, Joshua J Reicher, Lily Peng, Daniel Tse, Mozziyar Etemadi, Wenxing Ye, Greg Corrado, et al. End-to-end lung cancer screening with three-dimensional deep learning on low-dose chest computed tomography. *Nature medicine*, 25(6):954–961, 2019.

[127] Andreas Holzinger, Georg Langs, Helmut Denk, Kurt Zatloukal, and Heimo Müller. Causability and explainability of artificial intelligence in medicine. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 9(4):e1312, 2019.

[128] Isabella Castiglioni, Leonardo Rundo, Marina Codari, Giovanni Di Leo, Christian Salvatore, Matteo Interlenghi, Francesca Gallivanone, Andrea Cozzi, Natascha Claudia D'Amico, and Francesco Sardanelli. Ai applications to medical images: From machine learning to deep learning. *Physica Medica*, 83:9–24, 2021.

[129] Arnaud Nguembang Fadja, Fabrizio Riguzzi, and Evelina Lamma. Learning hierarchical probabilistic logic programs. *Machine Learning*, pages 1–57, 2021.

[130] Arnaud Nguembang Fadja, Evelina Lamma, and Fabrizio Riguzzi. Deep probabilistic logic programming. In *PLP@ ILP*, pages 3–14, 2017.

[131] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[132] Ahmad Taher Azar, Hanaa Ismail Elshazly, Aboul Ella Hassanien, and Abeer Mohamed Elkorany. A random forest classifier for lymph diseases. *Computer methods and programs in biomedicine*, 113(2):465–473, 2014.

[133] Wei-Yin Loh. Classification and regression trees. *Wiley interdisciplinary reviews: data mining and knowledge discovery*, 1(1):14–23, 2011.

[134] Arnaud Nguembang Fadja and Fabrizio Riguzzi. Lifted discriminative learning of probabilistic logic programs. *Machine Learning*, 108(7):1111–1135, 2019.

[135] Arnaud Nguembang Fadja, Fabrizio Riguzzi, and Evelina Lamma. Learning the parameters of deep probabilistic logic programs. In *PLP@ ILP*, pages 9–14, 2018.

[136] Arnaud Nguembang Fadja, Fabrizio Riguzzi, and Evelina Lamma. Expectation maximization in deep probabilistic logic programming. In *International conference of the italian association for artificial intelligence*, pages 293–306. Springer, 2018.

[137] Qin Sun, Haibo Qiu, Mao Huang, and Yi Yang. Lower mortality of covid-19 by early recognition and intervention: experience from jiangsu province. *Annals of intensive care*, 10(1):1–4, 2020.

[138] Chansik An, Hyunsun Lim, Dong-Wook Kim, Jung Hyun Chang, Yoon Jung Choi, and Seong Woo Kim. Machine learning prediction for mortality of patients diagnosed with covid-19: a nationwide korean cohort study. *Scientific reports*, 10(1):1–11, 2020.

[139] Dan Assaf, Ya'ara Gutman, Yair Neuman, Gad Segal, Sharon Amit, Shiraz Gefen-Halevi, Noya Shilo, Avi Epstein, Ronit Mor-Cohen, Asaf Biber, et al. Utilization of machine-learning models to accurately predict the risk for critical covid-19. *Internal and emergency medicine*, 15(8):1435–1443, 2020.

[140] Augusto Di Castelnuovo, Marialaura Bonaccio, Simona Costanzo, et al. Common cardiovascular risk factors and in-hospital mortality in 3,894 patients with covid-19: survival analysis and machine learning-based findings from the multicentre italian corist study. 2020.

[141] Li Yan, Hai-Tao Zhang, Jorge Goncalves, Yang Xiao, Maolin Wang, Yuqi Guo, Chuan Sun, Xiuchuan Tang, Liang Jing, Mingyang Zhang, et al. An

interpretable mortality prediction model for covid-19 patients. *Nature machine intelligence*, 2(5):283–288, 2020.

[142] N Alsharman and I Jawarneh. Googlenet cnn neural network towards chest ct-coronavirus medical image classification. *Journal of Computer Science*, pages 620–625, 2020.

[143] Saleh Albahli. Efficient gan-based chest radiographs (cxr) augmentation to diagnose coronavirus disease pneumonia. *International journal of medical sciences*, 17(10):1439, 2020.

[144] Parnian Afshar, Shahin Heidarian, Farnoosh Naderkhani, Anastasia Oikonomou, Konstantinos N Plataniotis, and Arash Mohammadi. Covid-caps: A capsule network-based framework for identification of covid-19 cases from x-ray images. *Pattern Recognition Letters*, 138:638–643, 2020.

[145] Mensah Kwabena Patrick, Adebayo Felix Adekoya, Ayidzoe Abra Mighty, and Baagyire Y Edward. Capsule networks–a survey. *Journal of King Saud University-computer and information sciences*, 2019.

[146] Subhanik Purkayastha, Yanhe Xiao, Zhicheng Jiao, Rujapa Thepumnoeysuk, Kasey Halsey, Jing Wu, Thi My Linh Tran, Ben Hsieh, Ji Whae Choi, Dongcui Wang, et al. Machine learning-based prediction of covid-19 severity and progression to critical illness using ct imaging and clinical data. *Korean Journal of Radiology*, 22, 2021.

[147] Fu-Yuan Cheng, Himanshu Joshi, Pranai Tandon, Robert Freeman, David L Reich, Madhu Mazumdar, Roopa Kohli-Seth, Matthew A Levin, Prem Timsina, and Arash Kia. Using machine learning to predict icu transfer in hospitalized covid-19 patients. *Journal of clinical medicine*, 9(6):1668, 2020.

[148] Jonathan Montomoli, Luca Romeo, Sara Moccia, Michele Bernardini, Lucia Migliorelli, Daniele Berardini, Abele Donati, Andrea Carsetti, Maria Grazia Bocci, Pedro David Wendel Garcia, et al. Machine learning using the extreme gradient boosting (xgboost) algorithm predicts 5-day delta of sofa score at icu admission in covid-19 patients. *Journal of Intensive Medicine*, 1(02):110–116, 2021.

[149] Yanwei Pang, Jin Xie, Muhammad Haris Khan, Rao Muhammad Anwer, Fahad Shahbaz Khan, and Ling Shao. Mask-guided attention network for occluded pedestrian detection. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4967–4975, 2019.

[150] Tingting Wang, Liang Wan, Lu Tang, and Mingsheng Liu. Mga-yolov4: a multi-scale pedestrian detection method based on mask-guided attention. *Applied Intelligence*, pages 1–17, 2022.

[151] Cheng Chi, Shifeng Zhang, Junliang Xing, Zhen Lei, Stan Z Li, and Xudong Zou. Pedhunter: Occlusion robust pedestrian detector in crowded scenes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 10639–10646, 2020.

[152] Xiaoqing Zheng, Song Zheng, Yaguang Kong, and Jie Chen. Recent advances in surface defect inspection of industrial products using deep learning techniques. *The International Journal of Advanced Manufacturing Technology*, 113(1):35–58, 2021.

[153] Gaokai Liu, Ning Yang, Lei Guo, Shiping Guo, and Zhi Chen. A one-stage approach for surface anomaly detection with background suppression strategies. *Sensors*, 20(7):1829, 2020.

[154] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[155] Awei Zhou, Bobo Ai, Pingge Qu, and Wei Shao. Defect detection for highly reflective rotary surfaces: An overview. *Measurement Science and Technology*, 32(6):062001, 2021.

[156] Vijaymeena M.K and K. Kavitha. A survey on similarity measures in text mining. 2016.

[157] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2921–2929, 2016.

[158] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn, 2018.

[159] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017.

[160] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[161] Shie Mannor, Dori Peleg, and Reuven Rubinstein. The cross entropy method for classification. In *Proceedings of the 22nd international conference on Machine learning*, pages 561–568, 2005.

[162] Singh S. Markou M. Novelty detection: a reviewpart 2: statistical approaches. 2004.

[163] Austin J. Hodge V. A survey of outlier detection methodologies. *Artificial Intelligence Review*, 2004.

[164] Shpitalni M. Weimer D, Scholz-Reiter B. Design of deep convolutional neural network architectures for automated feature extraction in industrial inspection. *CIRP*, 2016.

[165] Michael Freitag Benjamin Staar, Michael Lutjen. Anomaly detection with convolutional neural networks for industrial surface inspection. *Procedia CIRP*, 2019.

[166] Ailon N. Hoffer E. Deep metric learning using triplet network. *International Workshop on Similarity-Based Pattern Recognition*, 2015.

[167] Samet Akcay, Amir Atapour-Abarghouei, and Toby P Breckon. Ganomaly: Semi-supervised anomaly detection via adversarial training. In *Asian conference on computer vision*, pages 622–637. Springer, 2018.

[168] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[169] An Jinwon and Cho Sungzoon. Variational autoencoder based anomaly detection using reconstruction probability. *Special Lecture on IE*, 2(1):1–18, 2015.

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

[171] In Yong Moon, Ho Won Lee, Se-Jong Kim, Young-Seok Oh, Jaimyun Jung, and Seong-Hoon Kang. Analysis of the region of interest according to cnn structure in hierarchical pattern surface inspection using cam. *Materials*, 14(9):2095, 2021.

[172] Niv Cohen and Yedid Hoshen. Sub-image anomaly detection with deep pyramid correspondences. *arXiv preprint arXiv:2005.02357*, 2020.

[173] Jihun Yi and Sungroh Yoon. Patch svdd: Patch-level svdd for anomaly detection and segmentation. In *Proceedings of the Asian Conference on Computer Vision*, 2020.

[174] Daiki Kimura, Subhajit Chaudhury, Minori Narita, Asim Munawar, and Ryuki Tachibana. Adversarial discriminative attention for robust anomaly detection. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2172–2181, 2020.

[175] Shashanka Venkataramanan, Kuan-Chuan Peng, Rajat Vikram Singh, and Abhijit Mahalanobis. Attention guided anomaly localization in images. In *European Conference on Computer Vision*, pages 485–503. Springer, 2020.

[176] Jouwon Song, Kyeongbo Kong, Ye-In Park, Seong-Gyun Kim, and Suk-Ju Kang. Anoseg: Anomaly segmentation network using self-supervised learning. *arXiv preprint arXiv:2110.03396*, 2021.

[177] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research).

[178] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-100 (canadian institute for advanced research).

[179] Xin He, Kaiyong Zhao, and Xiaowen Chu. Automl: A survey of the state-of-the-art. *Knowledge-Based Systems*, 212:106622, 2021.

[180] Antonio Gulli and Sujit Pal. *Deep learning with Keras*. Packt Publishing Ltd, 2017.

[181] Tao Wei, Changhu Wang, Yong Rui, and Chang Wen Chen. Network morphism. In *International conference on machine learning*, pages 564–572. PMLR, 2016.

[182] Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[183] Thomas Elsken, Jan-Hendrik Metzen, and Frank Hutter. Simple and efficient architecture search for convolutional neural networks. *arXiv preprint arXiv:1711.04528*, 2017.

[184] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[185] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[186] Leonardo Rundo, Roberta Eufrasia Ledda, Christian di Noia, Evis Sala, Giancarlo Mauri, Gianluca Milanese, Nicola Sverzellati, Giovanni Apolone, Maria Carla Gilardi, Maria Cristina Messa, et al. A low-dose ct-based radiomic model to improve characterization and screening recall intervals of indeterminate prevalent pulmonary nodules. *Diagnostics*, 11(9):1610, 2021.

[187] S.P. Morozov, A.E. Andreychenko, N.A. Pavlov, A.V. Vladzymyrskyy, N.V. Ledikhova, V.A. Gombolevskiy, I.A. Blokhin, P.B. Gelezhe, A.V. Gonchar, and V.Yu. Chernina. Mosmeddata: Chest ct scans with covid-19 related findings dataset. *medRxiv*, 2020.

[188] Qinghe Zheng, Mingqiang Yang, Xinyu Tian, Nan Jiang, and Deqiang Wang. A full stage data augmentation method in deep convolutional neural network for natural image classification. *Discrete Dynamics in Nature and Society*, 2020, 2020.

[189] Qinghe Zheng, Mingqiang Yang, Jiajie Yang, Qingrui Zhang, and Xinxin Zhang. Improvement of generalization ability of deep cnn via implicit regularization in two-stage training process. *IEEE Access*, 6:15844–15869, 2018.

[190] Qinghe Zheng, Xinyu Tian, Mingqiang Yang, Yulin Wu, and Huake Su. Pac-bayesian framework based drop-path method for 2d discriminative convolutional network pruning. *Multidimensional Systems and Signal Processing*, 31(3):793–827, 2020.

[191] Qinghe Zheng, Penghui Zhao, Yang Li, Hongjun Wang, and Yang Yang. Spectrum interference-based two-level data augmentation method in deep learning for automatic modulation classification. *Neural Computing and Applications*, 33(13):7723–7745, 2021.

[192] Qinghe Zheng, Penghui Zhao, Deliang Zhang, and Hongjun Wang. Mr-dcae: Manifold regularization-based deep convolutional autoencoder for unauthorized broadcasting identification. *International Journal of Intelligent Systems*, 36(12):7204–7238, 2021.

[193] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240. ACM, 2006.

[194] Wanshan Ning, Shijun Lei, Jingjing Yang, Yukun Cao, Peiran Jiang, Qianqian Yang, Jiao Zhang, Xiaobei Wang, Fenghua Chen, Zhi Geng, et al. Open resource of clinical data from patients with pneumonia for

the prediction of covid-19 outcomes via deep learning. *Nature biomedical engineering*, 4(12):1197–1207, 2020.

Author's Publications and Awards

Publications

All works developed during the PhD and described in this thesis have led to the publications listed below.

- International Journals:
  - Michele Fraccaroli, Evelina Lamma, and Fabrizio Riguzzi. Symbolic dnn-tuner. Machine Learning, 111(2):625–650, 2022.
  - Michele Fraccaroli, Evelina Lamma, and Fabrizio Riguzzi. Symbolic dnn-tuner: a python and problog-based system for optimizing deep neural networks hyperparameters. SoftwareX, 17:100957, 2022.
  - Fadja, A.N., Fraccaroli, M., Bizzarri, A. et al. Neural-Symbolic Ensemble Learning for early-stage prediction of critical state of Covid-19 patients. Med Biol Eng Comput (2022). https://doi.org/10.1007/s11517-022-02674-1.
- International Conferences:
  - Michele Fraccaroli, Evelina Lamma, and Fabrizio Riguzzi. Exploiting Parameters Learning for Hyper-parameters Optimization in Deep Neural Networks. Proceedings of the 36th International Conference on Logic Programming, 2075-2180, 10.4204/EPTCS.364, 2022.
  - Michele Fraccaroli, Giulia Mazzucchelli, and Alice Bizzarri. Machine learning techniques for extracting relevant features from clinical data for COVID-19 mortality prediction. In 2021 Symposium on Computers and Communications (ISCC): 26th IEEE Symposium on Computers and Communications - Workshop on ICT Solutions

for eHealth (ICTS4eHealth) (ICTS4eHealth2021), pages 1–7, Athens, Greece, September 2021.

  - Michele Fraccaroli, Evelina Lamma, and Fabrizio Riguzzi. Automatic setting of DNN hyper-parameters by mixing Bayesian Optimization and tuning rules. In Giuseppe Nicosia, Varun Ojha, Emanuele La Malfa, Giorgio Jansen, Vincenzo Sciacca, Panos Pardalos, Giovanni Giuffrida, and Renato Umeton, editors, Machine Learning, Optimization, and Data Science, 6th International Conference, LOD 2020, Siena, Italy, July 19–23, 2020, Revised Selected Papers, Part I, volume 12565 of Lecture Notes in Computer Science, pages 477–488, Cham, 2020. © Springer, Springer International Publishing.

- Book Chapters:

  - Riccardo Zese, Elena Bellodi, Michele Fraccaroli, Fabrizio Riguzzi, and Evelina Lamma. Neural Networks and Deep Learning Fundamentals, pages 23–42. Springer International Publishing, Cham, 2022.

- National Workshop:

  - Michele Fraccaroli, Alice Bizzarri, Paolo Casellati, and Evelina Lamma. Cross entropy overlap distance. Accepted and Presented at ITALIA 2022, workshop on AI for Industry, feb 2022.

- Submitted:

  - Michele Fraccaroli, Alice Bizzarri, Paolo Casellati, and Evelina Lamma. Exploiting CNN's Visual Explanations to drive Anomaly Detection. submitted to Applied Intelligence, Springer.

Award

- National Award in Research Big Data & Artificial Intelligence provided by IFAB and Web Marketing Festival.