Politecnico di Torino

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

Doctoral Dissertation
Doctoral Program in Artificial Intelligence for Industry 4.0 (37^{th}cycle)

Neuro-Symbolic Integration in Artificial Intelligence and its Applications

By

Alice Bizzarri

******

Supervisor(s):
Prof. Evelina Lamma, Supervisor
Prof. Fabrizio Riguzzi, Co-Supervisor

Doctoral Examination Committee:
Prof. Luca Oneto, Referee, University of Genova
Prof. Felix Weitkaemper, Referee, LMU Munich

Politecnico di Torino
2025

Declaration

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

Alice Bizzarri
2025

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

Happiness is only real when shared.

Acknowledgements

I would like to thank my supervisors, Evelina Lamma and Fabrizio Riguzzi, for guiding me throughout my journey. I also extend my gratitude to my colleagues and fellow members of the AI laboratory at the University of Ferrara, as well as the friends who accompanied me during my time in Pensacola, for the immense support they have always provided and for making me feel at home even when I was so far away.

I am deeply grateful to my parents, Monica and Pierluigi, and my siblings for their unconditional love and encouragement. To my friends, thank you for offering me moments of fresh air in nature and for bringing genuine joy and laughter into my life when I needed it most.

To my boyfriend Diego, thank you for all the adventures we've shared, for every moment we've experienced together, and for being by my side through it all.

From the bottom of my heart, thank you all.

Abstract

In recent years, artificial intelligence (AI) has gained an increasingly central role in many fields, including medicine, engineering, and finance. Today, AI is deeply embedded in our daily lives, influencing areas such as recommendation systems, smart agriculture, Industry 4.0, healthcare, and autonomous driving. With growing interest and investments in this field, the performance and effectiveness of AI systems have rapidly evolved over the past few decades.

Currently, AI is primarily structured into two main approaches: symbolic AI and neural AI. Symbolic AI is based on explicit rules and logic for problem-solving, while neural AI, which is built on deep neural networks, learns from data and generalizes complex behaviors. Both paradigms have led to significant advancements, but they also exhibit inherent limitations. Neural systems, while highly effective in terms of performance, are often difficult to interpret and require significant computational power. In contrast, symbolic models are more transparent and interpretable but tend to be less flexible when dealing with new or unforeseen situations.

A potential solution to overcome these limitations lies in the integration of symbolic AI and neural AI, a synergy that can combine the accuracy and learning capabilities of neural models with the transparency and robustness of symbolic systems. This hybrid approach, known as *neurosymbolic AI*, represents a promising avenue for enhancing the explainability, computational efficiency, and overall robustness of AI systems, making them more reliable and versatile across a wide range of applications.

This thesis explores the potential of neurosymbolic integration, focusing on the importance of combining the learning capabilities of deep neural networks with the explicit and interpretable reasoning characteristic of symbolic AI. The primary objective is to develop hybrid models that ensure not only high performance but also interpretability, efficiency, and robustness, with particular attention to critical

domains such as healthcare and cybersecurity, where decision transparency is of paramount importance.

Throughout the research, various types of integration were explored using neurosymbolic frameworks, and tailored solutions were developed. Specifically, neurosymbolic ensemble learning systems were created, capable of combining a symbolic subsystem with a neural subsystem through the use of explainable mechanisms. Furthermore, the integration of constraints during the training process was analyzed, with the aim of creating more precise and robust anomaly detection systems. A hybrid system for the "open set recognition" scenario was also developed, combining neural and symbolic components. In addition to technical aspects, the thesis proposes new benchmarks for the evaluation of neurosymbolic systems, filling a gap due to the lack of shared standards in the field.

Practical applications of the developed systems were discussed, with a detailed presentation of the obtained results. Despite the promising results, the thesis highlights some ongoing challenges, such as the computational complexity of hybrid models and the difficulties in integrating deep learning with symbolic reasoning. Another critical aspect concerns the applicability of these models to unexplored domains, such as robotics or autonomous systems, which require further study.

In conclusion, neurosymbolic integration represents an important step towards the development of AI systems that are more transparent, reliable, and capable of tackling complex challenges in critical contexts. The research presented in this thesis lays the foundation for future developments in AI, aiming to make these systems increasingly efficient and interpretable, while simultaneously improving their adaptability across different application domains.

Contents

List of Figures

List of Tables

List of Acronyms

1D CNN	One-Dimensional Convolutional Neural Network
AC	Arithmetic Circuit
AE	Autoencoder
AI	Artificial Intelligence
ANN	Artificial Neural Network
AUCROC	Area Under the Curve Receiver Operating Characteristic
AUPRC	Area Under the Precision-Recall Curve
BO	Bayesian Optimization
CAM	Class Activation Map
CART	Classification And Regression Trees
CE	Cross-Entropy
CEOD	Cross Entropy Overlap Distance
CNN	Convolutional Neural Network
CN	Constrained Network
CT	Computed Tomography
DEC	Deep Embedding for Clustering
DL	Deep Learning
DNN	Deep Neural Network
DSRL	Deep Symbolic Reinforcement Learning
DT	Decision Tree
ELU	Exponential Linear Unit

FCM	Fuzzy c-means
FLOPS	FLOating Point Operations
FN	False Negative
FP	False Positive
FOL	First Order Logic
FPR	False Positive Rate
FT	Fine-Tuning
GAN	Generative Adversarial Network
GNN	Graph Neural Network
Grad-CAM	Gradient-weighted Class Activation Mapping
GRL	Graph Representation Learning
HPLP	Hierarchical Probabilistic Logic Program
HPO	Hyperparameter Optimization
HU	Hounsfield Units
IoT	Internet of Things
KL	Kullback-Leibler
LSTM	Long Short-Term Memory
LTN	Logic Tensor Network
MACs	Multiply-Accumulate Operations
ML	Machine Learning
MLP	Multilayer Perceptron
MOO	Multi-Objective Optimization
MSE	Mean Square Error
NAD	Neural Annotated Disjunction
NAS	Neural Architecture Search
NDA	Non-Disclosure Agreement
NeSy AI	Neuro-Symbolic AI
NIDS	Network Intrusion Detection Systems

NN	Neural Network
NSCL	Neuro-Symbolic Concept Learner
OD	Outlier Detection
OOD	Out-Of-Distribution
OSR	Open Set Recognition
PCAP	Packet Capture
PHIL	Parameter Learning for Hierarchical Probabilistic Logic Programs
PLP	Probabilistic Logic Programming
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
RF	Random Forest
RNN	Recurrent Neural Network
ROI	Region of Interest
SDD	Sentential Decision Diagram
SLEAHP	Structure Learning of Hierarchical Probabilistic Logic Programming
SMOTE	Synthetic Minority Oversampling Technique
SOFA	Sequential Organ Failure Assessment
TAs	Tuning Actions
TL	Transfer Learning
TN	True Negative
TP	True Positive
TPR	True Positive Rate
VAE	Variational Autoencoder

Chapter I

Introduction

1 Motivation

In recent years, Artificial Intelligence (AI) research has advanced significantly, with Deep Learning (DL) emerging as one of the most powerful techniques for pattern recognition and learning from vast amounts of data. Despite its success, DL models suffer from a critical lack of interpretability. This limitation poses a major challenge, particularly in domains where decision transparency is essential, such as medical diagnosis, industrial automation, and cybersecurity. In these fields, it is not sufficient to simply produce accurate results; there must also be a clear understanding of the underlying reasons behind those results.

On the other hand, symbolic AI, which relies on formal representations and logical rules, offers superior explainability and reasoning capabilities. However, this approach struggles with scalability and adaptability, especially when dealing with large and complex datasets, where pure symbolic methods fall short.

Neuro-symbolic integration seeks to combine the strengths of these two paradigms: the generalization and learning power of DL and the explicit reasoning and interpretability of symbolic AI. This emerging framework presents an opportunity to overcome the limitations of each individual approach, paving the way for more robust, interpretable, and adaptive AI systems.

The core motivation of this thesis lies in the increasing need for AI systems that can operate in high-stakes and critical contexts, such as healthcare and cybersecu-

rity. In these domains, not only must decisions be accurate, but they must also be explainable and understandable to end users, such as doctors or security analysts.

This research aims to explore and develop neuro-symbolic integration techniques to enhance the performance of AI models while ensuring high levels of interpretability.

2 Thesis Aims

The primary goal of this thesis is to investigate and develop neuro-symbolic integration techniques that combine the interpretability of symbolic AI with the learning capabilities of deep learning models. Although neuro-symbolic systems are still in their early stages of development, this research aims to analyze and apply existing systems in real-world contexts, define benchmarks to evaluate proposed solutions, and create new neuro-symbolic models tailored to real-world challenges.

Specifically, the research focuses on implementing hybrid models that enhance transparency, reliability, and performance in critical applications such as medical diagnosis and anomaly detection. To this end, various systems have been proposed to address different application challenges. For instance, one of the developed systems combines probabilistic logic with neural networks to improve the prediction of medical conditions, ensuring both accurate forecasts and the explainability required in the healthcare domain. Additionally, a system for network intrusion detection was developed, capable of identifying previously unknown attacks through the use of deep learning and symbolic logic techniques.

Another important contribution of this thesis is the introduction of new benchmarks for evaluating the performance of neuro-symbolic systems, which currently lack a standardization. Moreover, the thesis explores the optimization of deep learning models for use in resource-constrained environments, such as TinyML, by applying neuro-symbolic systems to ensure computationally and energy-efficient models that are still interpretable.

This research aims to contribute to the evolution of AI towards more robust, transparent, and adaptive systems capable of addressing the challenges of complex application domains, with a particular focus on the explainability of decisions in critical sectors such as healthcare and cybersecurity.

3 Thesis Outline

This thesis is structured into five main chapters, each contributing to the exploration of neuro-symbolic integration in Artificial Intelligence and its practical applications. The chapters follow a progressive path, starting from fundamental concepts and advancing towards specific applications.

The first chapter introduces the thesis by explaining the motivations behind this work and highlighting the growing importance of developing AI systems that are both powerful and interpretable. Additionally, the specific research objectives are outlined, with a focus on critical contexts such as healthcare and cybersecurity, where transparency in AI decision-making is of paramount importance.

In the second chapter, a comprehensive review of the essential foundational concepts for understanding neuro-symbolic integration is provided. The key areas of symbolic AI, deep learning, and probabilistic logic programming are addressed, forming a solid foundation for the hybrid systems discussed in the following chapters. These sections not only introduce the reader to fundamental techniques but also help to contextualize the innovations presented in the rest of the thesis.

The third chapter forms the core of the thesis, where various neuro-symbolic approaches are presented. These approaches demonstrate how neural and symbolic methods can be effectively combined to improve the performance and interpretability of AI systems.

The fourth chapter applies the proposed neuro-symbolic methods to real-world scenarios, with a particular emphasis on healthcare, where these techniques are used for COVID-19 diagnostics and anomaly detection. This chapter builds on the findings presented in the previous chapter and presents results.

These studies exemplify the practical utility of neuro-symbolic systems in addressing complex tasks in medical and industrial settings.

Finally, the thesis concludes with a chapter that reflects on the main contributions of this research and discusses potential directions for future work. The broader impact of neuro-symbolic AI is emphasized, particularly in emerging areas such as network intrusion detection and open-set recognition.

Chapter II

Background

4 Symbolic AI

In AI, the term symbolic AI refers to methods that use high-level symbolic, human-interpretable, representations of problems, logic, and reasoning. These methods are based on symbols, which are discrete, well-defined entities that represent concepts, and on formal rules that describe how symbols can be manipulated to emulate human thought.

In the 1950s and 1960s, the first AI programs were created using symbolic representations to solve mathematical and logical problems. The 1960s saw the birth of expert systems, which aimed to imitate human decision-making in specific fields. In the 1970s and 1980s, expert systems became popular in fields such as medicine and finance [1]. However, symbolic systems had problems with scalability and adaptability due to the manual coding of knowledge. In the 2010s, symbolic AI declined in popularity due to DL and statistical algorithms. In recent years, there has been a renewed interest in symbolic AI in combination with DL [2]. Modern applications include advanced medical diagnosis systems [3, 4], Computer Vision [5, 6], Question Answering [7] and Robotics [8]. The use of languages such as probabilistic logic programming has helped to integrate DL with symbolic techniques.

Symbolic AI offers advantages such as transparency and interpretability, allowing humans to understand and explain the decisions made by the system. It also enables precise logical manipulation, which is beneficial in fields like mathematics and formal

verification. However, symbolic methods have limitations, including scalability issues for large problems and a lack of adaptability to new data or unexpected situations due to rigid rules. In response to these limitations, symbolic AI has evolved and integrated with DL and statistical approaches. This integration has resulted in hybrid methodologies that combine the precision and interpretability of symbolic AI with the generalising and adaptive power of DL.

Some main techniques and methodologies used in the field of symbolic AI will be discussed below, focusing specifically on those I used in my thesis.

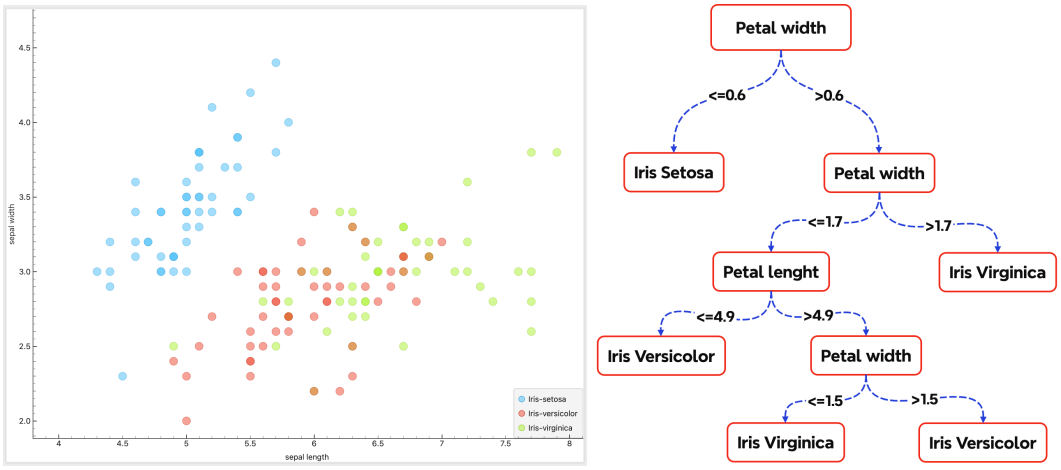
4.1 Decision Trees

Decision Trees (DTs) [9], or Classification And Regression Trees (CART) [10], represent a non-parametric supervised learning technique employed for both classification and regression tasks. The primary objective is to construct a model that predicts the value of a target variable by learning simple decision rules inferred from the data.

DTs are defined through a recursive subdivision of the input space, with the consequent establishment of a local model within each resulting region. The global model is representable as a tree, featuring a leaf for each region [11]. In general, DTs are constructed via an algorithmic approach that identifies ways to split a data set based on different conditions.

At each node, the feature value d_i of the input vector x is compared against a threshold value t_i. If the feature value is higher than the threshold, the input is directed to the right branch; otherwise, it proceeds to the left branch. In the case of discrete features, the splitting process differs. Instead of comparing feature values against a threshold, the input is divided based on the different categories or values that the discrete feature can assume. For each distinct value of the feature, a separate branch is created, directing the input to the appropriate subtree based on the specific value of the discrete feature.

Each terminal node (leaf) of the tree represents a logical expression formed by the conjunction of literals encountered along the path from the root of the tree to the leaf. DTs can be classified into two types: classification trees and regression trees. In classification trees, the predicted outcome is a class (discrete) to which the data belongs, whereas in regression trees, the predicted outcome is a real number.



(a) (b)

Fig. II.1 (a) Scatter plot of Iris dataset. (b) A hypothetical classification tree fitted to Iris dataset

In DTs, each leaf node has an output label, which can be categorical (yes or no) for classification tasks, or numerical for regression tasks. New instances falling in a given leaf will have the same label (classification) or similar numeric output (regression).

Consider, for instance, the Iris dataset [12]. A DT can be constructed to classify the samples. The tree is illustrated in Figure II.1. The initial node tests whether the petal width of sample x_i is less than or equal to the threshold $t_i = 0.6$. If this condition is satisfied, the label of sample x_i is determined to be *Iris-Setosa*. If not, the tree continues to be traversed until a terminal leaf node is reached, which corresponds to the sample's label.

The tree structure is not predetermined but evolves during the learning process according to the problem's complexity. Several algorithms exist for learning a DT structure, with popular methods including CART [10], ID3 [9], and C4.5 [13]. These algorithms use a greedy procedure, iteratively expanding the tree one node at a time to identify the optimal DT structure.

The methodology is based on "divide and conquer" techniques through an iterative process that segments the dataset into smaller subgroups using a series of tests or decisions until each subgroup is as homogeneous as possible concerning the target variable. Initially, the feature that best partitions the dataset is selected, leading to the creation of a decision node representing this choice, which includes a test or condi-

tion based on the chosen feature. The dataset is then split into subgroups according to this condition. This process is repeated iteratively until all instances within a subset belong to the same class (resulting in pure nodes) or until a termination condition is met (e.g., the subset reaches a predetermined minimum size).

Several metrics are employed to measure the *purity* of each node. A node is considered pure if all instances in it belong to the same class C_i, represent a leaf node. Common metrics used to assess the quality of partitioning include:

- Entropy for classification, defined as

$$H(D) = -\sum_{j=1}^{k} P(c_j) \log_2 P(c_j) = E[-\log_2 P(C)] \tag{II.1}$$

  where $P(c_j) = \frac{|D_j|}{|D|}$ with D representing the dataset and D_j the subset of examples from D that belong to class c_j.

- The Gini index for classification, defined as

$$Gini(D) = 1 - \sum_{j=1}^{k} P(c_j)^2 \tag{II.2}$$

- Mean Square Error (MSE) for regression, defined as

$$\text{MSE}(D_i) = \frac{1}{|D_i|} \sum_{n \in D_i} (y_n - \hat{y})^2 \tag{II.3}$$

  where $\hat{y}$ is the mean of the target value in D_i.

Tree models offer several advantages: they are very fast to train and test (highly parallelizable), and, with binary decisions, each decision can eliminate half of the cases. If there are b regions, then in the best case, the correct region can be found in $log_2 b$ steps. Additionally, DTs are easy to interpret and can be converted into a set of IF-THEN rules that are easily understandable, contributing to their popularity.

DTs can fit all datasets with zero bias and high variance, but they often suffer from overfitting. This means that while the model can classify training data with high accuracy, its performance with new, unseen data is often poor. This issue is partly due to the greedy nature of the tree construction algorithm. Another problem is the

instability of trees: small modifications to the input data can result in significant changes to the tree structure, due to the hierarchical nature of the tree growth process. Consequently, errors at the top of the tree can propagate through the remainder of the structure [11].

4.2 Random Forest

As discussed in Section 4.1, DTs are high-variance estimators. A common method to mitigate this variance is ensemble learning, where the outputs of multiple models are combined. Two widely used ensemble techniques are averaging and voting, each with a different way of aggregating the results from the base models.

In the case of averaging, the outputs of the models are averaged, typically used for regression tasks. The resulting model is expressed as follows:

$$f(\mathbf{x}) = \frac{1}{|M|} \sum_{m \in M} f_m(\mathbf{x}) \tag{II.4}$$

where M is a set of models and f_m represents the m-th baseline model. The ensemble will have a similar bias to that of the base models but with reduced variance, often resulting in improved overall performance.

In contrast, voting is typically used for classification tasks. Here, each model in the ensemble votes for a class, and the final prediction is the class that receives the majority of votes. The voting ensemble is expressed as:

$$\hat{y} = \text{mode}\left(\{\hat{y}_m\}_{m \in M}\right) \tag{II.5}$$

where $\hat{y}_m$ is the predicted class from the m-th model, and $\text{mode}(\cdot)$ returns the most frequent class label. Like averaging, voting reduces variance while maintaining the bias of the base models.

One of the ensemble learning techniques is bagging (*bootstrap aggregation*) [14]. In bagging, multiple base models are fitted to different versions of the data sampled randomly, which encourages diverse predictions. The datasets are sampled with replacement, allowing a given example to appear multiple times, resulting in a total of N examples per model (where N is the number of original data points).

The concept of bagging relies on the idea that applying a learning algorithm to different subsets of data will yield sufficiently diverse fundamental patterns. However, this approach does not always enhance performance, as it assumes that excluding certain data can significantly alter the fit of the resulting model. This is the case for DTs, but not for other models, such as nearest neighbour classifiers.

Other techniques have been developed to further improve basic learners. One such technique is Random Forests (RFs) [15]. In RFs, learning occurs through trees that are based on a randomly chosen subset of input variables at each node of the tree.

RFs employ the technique of feature bagging, where a subset of features is selected randomly to reduce the correlation between different DTs. This distinguishes RFs from DTs and bagging, as the latter considers all features, while RFs only consider a random subset of features.

Similar to bagging, each tree in a RF is created using a random sample of the training data, selected with replacement. A third of this data is set aside as an out-of-bag sample, which is used to validate the predictions. Depending on the type of problem, the final prediction is obtained by averaging the predictions of the individual trees (for regression) or by taking the majority vote of the predicted classes (for classification). The out-of-bag sample is also used for cross-validation, providing an accurate estimate of the model's performance.

This technique reduces the risk of overfitting and facilitates the determination of feature importance. However, RF algorithms can be slow in processing data, as they calculate the data for each individual DT, require more resources and are more complex.

4.3 Gradient Boosting

Both the bagging algorithm and the RF algorithm can be described by the formula:

$$f(\mathbf{x}, \theta) = \sum_{m=1}^{M} \alpha_m F_m(\mathbf{x}, \theta_m) \tag{II.6}$$

where F_m is the m-th tree and α_m is the corresponding weight. This concept can be generalised by allowing the functions F_m to be any model, such as NNs, not just

trees. The result is called the additive model [16]. This model can be viewed as a linear model with adaptive basis functions.

Boosting [17] is an algorithm for the sequential adaptation of additive models in which each F_m is a binary classifier that returns $F_m \in \{-1, +1\}$. In the initial step, F_1 is fitted to the original dataset. Then, the data samples are weighted according to the errors made by F_1, ensuring that misclassified examples receive higher weights.

Next, F_2 is fitted to the weighted dataset. This process is repeated until the desired number of components, M, has been reached. It is important to note that the bias inherent to strong learners is reduced by fitting trees that depend on one another, while the variance is reduced by fitting independent trees, as seen in bagging and RFs.

These models, known as weak learners, are iteratively combined into a single strong learner. Gradient boosting is a notable method among boosting techniques. Unlike traditional boosting methods that reweight data points, gradient boosting optimizes the model using gradient descent.

Consider a boosting algorithm with M stages. At each stage m (where $1 < m < M$), we start with an imperfect model F_m. To improve the predictions of F_m, we calculate the derivative of the loss function g_m for each model:

$$g_m(\mathbf{x}_i) = \left[\frac{\partial \ell\left(y_i, F\left(\mathbf{x}_i\right)\right)}{\partial F\left(\mathbf{x}_i\right)} \right]_{F=F_{m-1}} \tag{II.7}$$

where y_i is the observed value for sample $\mathbf{x}_i$. The model is then updated as follows:

$$F_{m+1} = F_m - \beta_m g_m \tag{II.8}$$

where β_m is the step length or learning rate.

In this framework, $g_m(\mathbf{x}_i)$ represents the gradient of the loss function with respect to the model's predictions, evaluated at the current model. The update step involves adjusting the current model F_m by moving in the direction of the negative gradient, scaled by the learning rate β_m. This iterative process continues until the model converges or reaches the predefined number of stages M.

XGBoost [18] stands for "Extreme Gradient Boosting", is an advanced implementation of gradient boosting designed for efficiency, speed, and performance. It

has become widely popular in Machine Learning (ML) competitions and industry applications due to its robust capabilities and versatility.

XGBoost is an advanced implementation of gradient boosting that incorporates regularization based on tree complexity. This regularization reduces overfitting and improves the model's generalization. Additionally, XGBoost automatically handles missing values during training, enhancing the model's robustness. The algorithm uses a tree pruning mechanism called "max depth" to retain only relevant splits and minimize model complexity. XGBoost also supports parallel processing in tree construction, significantly reducing training time and improving scalability. In summary, XGBoost offers regularization, automatic missing value handling, efficient tree pruning, and parallel processing capabilities, making it a powerful and efficient ML algorithm.

4.4 Clustering

Clustering is an ML technique for unsupervised learning, that is, learning from unlabelled data. Clustering learns a new labelling function g from unlabelled data which divides data in subgroups, known as clusters. The function can be a predictive model that can be applied to new data, or can be a descriptive model that only describes the given data.

A clustering algorithm works by assessing the similarity between instances and putting similar instances in the same cluster and 'dissimilar' instances in different clusters. The goal of clustering could be that of finding homogeneous groups in the data or that of finding unusual data objects (Outlier Detection (OD)). There exist different clustering techniques:

- **Exclusive Clustering**: In this type of clustering, each instance belongs to a single, unique cluster. One of the most common techniques for exclusive clustering is *K-means* [19], which operates with numeric data. K-means identifies k unique clusters by assigning each data point to a cluster and computing the centroid of each cluster, denoted as μ_k, which is the mean of the points in that cluster. The algorithm aims to minimize the sum of the squares of the distances of each data point from its nearest cluster center μ_k. The objective function minimized by K-means is:

$$e^2 = \sum_{j=1}^{k} \sum_{i \in \text{cluster}(j)} d^2(x_i, c_j) \tag{II.9}$$

where $d^2(x_i, c_j)$ represents the distance between instance x_i and cluster center c_j.

The K-means algorithm proceeds as follows: initially, it randomly selects k cluster centers. Each data point is then assigned to the nearest cluster center. Subsequently, each cluster center c_j is recomputed as the mean of the points assigned to that cluster. The data points are then reassigned to the nearest cluster center based on the updated means. These steps are iteratively repeated until the algorithm converges, meaning that the cluster centers no longer change significantly between iterations. The quality of the clusters is usually measured using the Sum of Squared Errors, which is the sum of the squared differences between each observation and the mean of its assigned cluster.

While K-means is easy to implement, the problem of finding the global minimum of the objective function is NP-complete. Although K-means converges to a stationary point in finite time, there are no guarantees that this point is the global minimum. Additionally, an unfortunate initialization of the centroids can lead to a suboptimal solution. In practice, it is advisable to run the algorithm multiple times with different initializations and select the solution with the smallest within-cluster scatter. In addition, large datasets increase the time complexity for distance-based methods. If an obvious distance measure doesn't exist, we must "define" it, which is not always straightforward, especially in multi dimensional spaces. The performance of the algorithm on a dataset will be significantly influenced by the chosen distance measure.

- **Overlapping Clustering**: This clustering approach is rooted in fuzzy set theory, where each data point can belong to multiple clusters with varying degrees of membership. Fuzzy c-means (FCM) [20] is an example of an overlapping clustering method. The FCM algorithm aims to partition a finite collection of n elements $X = \{x_1, \ldots, x_n\}$ into c fuzzy clusters based on a specified criterion. It returns c cluster centers $C = \{c_1, \ldots, c_c\}$ and a partition matrix $W = \{w_{ij}\} \in [0,1],\ i = 1, \ldots, n,\ j = 1, \ldots, c$, where each w_{ij} indicates the degree to which element x_i belongs to cluster c_j. The membership values

w_{ij} are calculated based on the relative distance between data points and cluster centers, given by:

$$w_{ij} = \frac{1}{\sum_{k=1}^{c} \left(\frac{\|x_i - c_j\|}{\|x_i - c_k\|} \right)^{\frac{2}{m-1}}}, \tag{II.10}$$

where m controls the fuzziness of the membership. The objective of FCM is to minimize the following function:

$$\arg\min_{C} \sum_{i=1}^{n} \sum_{j=1}^{c} w_{ij}^{m} \|x_i - c_j\|^2, \tag{II.11}$$

where $1 \leq m < +\infty$ is a parameter that controls the degree of fuzziness. As m approaches 1, w_{ij} tends to 0 or 1, resembling hard assignments as in K-means. Larger values of m lead to fuzzier cluster memberships. A key drawback of FCM is its susceptibility to local minima, and its results can vary significantly based on the initial selection of cluster centers and membership weights.

- **Hierarchical Clustering**: In this clustering technique, the input to the algorithm is an $N \times N$ dissimilarity matrix D, and the output is a tree structure in which groups i and j with small dissimilarities are grouped together in a hierarchical manner. A common algorithm used is the *Agglomerative bottom-up technique*. Given a set of N items to be clustered and an $N \times N$ distance (or similarity) matrix, the process begins by assigning each item to its own cluster, resulting in N clusters. The algorithm then finds the closest (most similar) pair of clusters and merge them into a single cluster, so that now we have one cluster less. Next, it computes the distances between the new cluster and each of the remaining clusters. This process is repeated until all items are merged into a single cluster of size N. The sequence of clusters formed during this process creates a hierarchical clustering. The function used to determine the distance between two clusters, known as the linkage function, can be single, complete, or average linkage. Additionally, the formula used to calculate inter-cluster distances can vary.

  The main drawbacks of hierarchical clustering are that it does not scale well, having a time complexity of $O(n^2)$ and memory complexity $O(n)$, where n is the total number of objects, using the SLINK [21] algorithm. However, the use

of a heap can reduce computation time to $O(n^2 log(n))$, but increase memory requirement [22]. The algorithm cannot undo previous steps, meaning once clusters are merged, they cannot be separated. Furthermore, the addition of a single new instance can significantly alter the entire clustering structure.

- **Probabilistic Clustering**: the most common probabilistic clustering algorithm is Clustering as a Mixture of Gaussians. In this approach each cluster is represented by a parametric distribution, such as a Gaussian (continuous) or a Poisson (discrete). The entire data set is then modelled as a mixture of these distributions. The most widely used clustering method of this kind is the one based on learning a mixture of Gaussians. The probability distribution, each for one cluster, describes the distribution of values for members of that cluster. For a given set of data points, the Mixture of Gaussians would identify the probability of each data point belonging to each of these distributions.

5 Probabilistic Logic Programming

Probabilistic Logic Programming (PLP) integrates logical and probabilistic reasoning to handle uncertainty in complex domains. One of the primary theoretical models underlying PLP is the distribution semantics [23], which defines a probability distribution over a set of normal logic programs, each representing a possible world. The probability of a query is computed from the joint distribution by marginalization [24].

One of the most well-known PLP languages under the distribution semantics is ProbLog [25]. ProbLog has been applied to various practical problems, such as natural language processing [26] and bioinformatics [27].

A probabilistic program P consists of a set of clauses, each clause c_i being associated with a probability p_i. The basic version of ProbLog, and all the examples provided, assign probabilities only to facts. In this case, each probabilistic fact is associated with a probability value that represents the likelihood of that fact being present in a possible world. Rules in the program are deterministic, with uncertainty handled exclusively at the fact level. The following is an example of a ProbLog program:

```
% Probabilistic facts:
```

```
0.5::heads1.
0.6::heads2.

% Rules:
someHeads :- heads1.
someHeads :- heads2.

% Query:
query(someHeads).
```

The probability distribution of a probabilistic program is defined over the possible ground atoms that can be true simultaneously and we assume that it has a single model that are generated by selecting subsets of the probabilistic facts, where each fact is either included or excluded based on its associated probability. The probability distribution is thus defined over the set of possible worlds, with each world corresponding to a model. The probability of a world is the product of the probabilities of the facts it includes and the complement of the probabilities of the facts it excludes. In other words, each possible world has an associated probability. The goal of ProbLog is to compute the probability that a given query holds. This probability is computed by summing the probabilities of all the interpretations in which the query is true [24]. For instance, in the example above, the probability of the query *someHeads* is 0.8

Here is another example of ProbLog code:

```
% Probabilistic facts:
0.7::burglary.
0.2::earthquake.
0.9::p_alarm1.
0.8::p_alarm2.
0.1::p_alarm3.

% Rules:
alarm :- burglary, earthquake, p_alarm1.
alarm :- burglary, \+earthquake, p_alarm2.
alarm :- \+burglary, earthquake, p_alarm3.
```

```
% Query:
query(alarm).
```

In this example, there are five probabilistic facts: the occurrence of a burglary (`burglary`), an earthquake (`earthquake`), and three distinct cases where the alarm goes off (`p_alarm1`, `p_alarm2`, `p_alarm3`). The rules define the conditions under which the alarm goes off, such as when both a burglary and an earthquake occur and `p_alarm1` is true. The query `alarm` then computes the probability that the alarm goes off, which is the sum of the probabilities of the worlds $p(w_i)$ where the query is true. In this case, the probability is:

$$\begin{aligned} P(q) &= p(w_1) + p(w_2) + p(w_3) \\ &= (0.7 \times 0.2 \times 0.9) + (0.7 \times 0.8 \times 0.8) + (0.3 \times 0.2 \times 0.1) \\ &= 0.58 \end{aligned}$$

Inference in ProbLog [25] works by generating all ground instances of the clauses in the program that are relevant to the query and transforming these clauses into a propositional formula. This process consists of translating the probabilistic logic program into a purely propositional representation, where each logical clause is expressed as a combination of Boolean variables describing whether a fact is true or false in each possible interpretation. The resulting logical formula is then compiled into a Sentential Decision Diagram (SDD) [28].

SDDs are a data structure that compactly and efficiently represents propositional knowledge bases. For a given query, ProbLog compiles the entire probabilistic program and its relevant clauses into an SDD, which is then evaluated in a bottom-up fashion, starting from the leaf nodes and progressing towards the root. In this way, ProbLog is able to compute the probability of the query by combining the results from all possible interpretations, weighted by their respective probabilities.

One of the main challenges of PLP is the high computational cost of probabilistic inference, due to the possibly exponential number of worlds that must be considered in order to calculate the probabilities of a given query.

In PLP, it is possible to learn either the structure or the parameters of the probabilistic logic program. *Parameter learning* refers to the process of determining the optimal values of the probabilities associated with the facts in the program [29]. This typically involves adjusting the probabilities so that the program's predictions align as closely as possible with the observed data. Techniques like gradient descent or Expectation Maximization (EM) are often used for parameter learning. On the other hand, *structure learning* refers to the process of discovering the optimal set of rules or logical relationships that best describe the underlying data. In structure learning, the task is to determine the logical dependencies (e.g., which predicates depend on which others) and the form of the clauses themselves. This can be seen as constructing the architecture of the program, while parameter learning fine-tunes it. Both forms of learning are essential to creating accurate and interpretable probabilistic logic programs.

5.1 Hierarchical Probabilistic Logic Programming

Hierarchical Probabilistic Logic Programs (HPLPs) provide a solution to the computational cost problem of traditional PLP by introducing a restriction to the syntax of clauses. In an HPLP, predicates and clauses are organized into a hierarchical structure. This structure significantly reduces the computational complexity of inference. An HPLP can be converted into an Arithmetic Circuit (AC) or a deep neural network, simplifying the inference and parameter learning [30].

The distinguishing feature of HPLPs is that logical clauses are constructed in hierarchical levels, with each level depending only on the lower levels; clauses for a predicate are built on predicates of the layer below, or input data. This type of organization promotes clause independence and makes probability calculations more manageable.

To perform inference in an HPLP, the program is transformed into an AC. An AC is a directed acyclic graph where nodes represent arithmetic operations such as sums and products, and the leaves represent probabilities associated with basic logical facts. Each path in the arithmetic circuit corresponds to a particular combination of events that lead to a specific result [31].

In the context of an HPLP, probabilities are propagated through the circuit using the *independent-or* principle, which ensures that the probability of a disjunction

(OR) of events is computed assuming the events are independent. This property allows for the combination of the probabilities of multiple clauses without needing to explore all possible combinations of logical worlds, thus significantly reducing inference time [30].

For parameter learning in HPLPs, an algorithm called Parameter Learning for Hierarchical Probabilistic Logic Programs (PHIL) was designed. The PHIL algorithm is intended to learn the parameters of an HPLP from a dataset. Its operation is based on two main techniques:

- DPHIL: The algorithm employs gradient descent to minimize the difference between the probabilities predicted by the model and the probabilities observed in the data. This is done by iteratively updating the model parameters to reduce overall error.
- EMPHIL: The algorithm uses an Expectation Maximization approach. In this scheme, the algorithm alternates between calculating the expectations of missing data and updating the model parameters, progressively improving the probability estimates.

The learning process in PHIL involves two phases: in the forward phase, the nodes of the arithmetic circuit are evaluated from input to output, calculating probabilities; in the backward phase, the quantities necessary to update the parameters are computed using a backpropagation-like approach, similar to that used in neural networks [30].

The following is an example of a HPLP and the respective generated tree structure:

$$\begin{aligned} C_1 &= works\_with(A,B) : 0.7 : - \\ &\quad worker(A), manager(B), project(A,C), project(B,C), r_{11}. \\ C_2 &= works\_with(A,B) : 0.2 : - \\ &\quad worker(A), manager(B), project(A,C), manages(B,C). \\ C_{111} &= r_{11}(A,B,C) : 0.9 : - \\ &\quad product(X,A,C), product(X,B,C). \end{aligned} \tag{II.12}$$

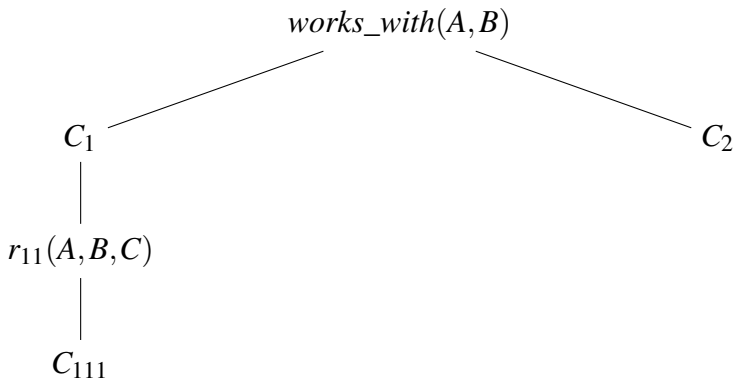


Fig. II.2 The probabilistic program tree for Example II.12.

In this context, *product*(X,A,C) denotes that the product X is a result of the work of A in project C. The predicate *works_with/2* is the target predicate, while *project/2*, *worker/1*, and *manager/1* are input predicates. In this case, the probability of $r = works\_with(A,B)$ is not only dependent on shared projects but also the products generated in those projects.

The clause defining $r_{11}(A,B,C)$ aggregates over the products associated with a project, while the higher-level clause aggregates over the projects. This type of program can be represented by the tree structure depicted in Figure II.2.

To learn the structure of an HPLP, the SLEAHP algorithm was developed. SLEAHP generates a large HPLP from a bottom clause obtained from a language bias as described in [32], and subsequently applies a regularized version of PHIL to the generated HPLP to prune clauses with low parameter values.

The HPLP approach offers many advantages over traditional PLP models. In particular, the ability to perform inference more efficiently and to automatically learn both the structure and parameters of a probabilistic logic model makes it suitable for a wide range of applications, including bioinformatics, natural language processing, and relationship prediction in social networks.

In conclusion, the hierarchical organization of HPLPs, combined with the learning algorithms PHIL and SLEAHP, provides a powerful solution for modeling and learning in contexts characterized by uncertainty and relational complexity.

6 First Order Logic and Fuzzy Logic

Logic plays a fundamental role in knowledge representation and automated reasoning. Various types of logic, such as First Order Logic (FOL) and Real Logic, are key tools for modeling complex problems. This section provides a brief discussion of both, emphatising their importance in the field of AI. FOL is a fundamental tool in knowledge representation and reasoning. It is one of the most powerful and expressive formal logics, capable of precisely modeling the real world through a set of well-defined logical rules. FOL differs from propositional logic by its ability to express statements not only about atomic propositions (as in propositional logic) but also about objects and the relationships between them. This makes it particularly suitable for describing complex phenomena involving multiple entities and their interactions [33].

FOL formulas use predicates, functions, and quantifiers. Predicates represent relationships between objects, while functions map or transform objects into other objects. For instance, the friendship relation between two people can be expressed with a predicate such as *Friend(x, y)* (indicating that x and y are friends).

A distinctive feature of FOL is the use of quantifiers, which allow for the expression of general or existential statements. The universal quantifier ($\forall$) is used to express that a certain property holds for all objects in a set. For example, "All men are mortal" can be formalized as $\forall x(Man(x) \rightarrow Mortal(x))$. The existential quantifier ($\exists$), on the other hand, asserts that at least one object satisfies a given property. For example, "There exists a person who is a doctor" can be expressed as $\exists x(Person(x) \wedge Doctor(x))$.

A crucial aspect of FOL is the notion of satisfiability and validity. A statement is considered satisfiable if there is at least one model (a possible interpretation of the world) in which the statement is true. This concept is essential in formal logic, as it allows for evaluating whether a set of statements or rules is consistent with a given model of the world [33]. As an extension of FOL, Real Logic expands the expressive capabilities of traditional logic by enabling the representation of partial or gradual truths, better reflecting the complexity of real-world scenarios.

Real Logic is particularly suited to represent situations where truth is not binary, but rather nuanced or gradual. While classical FOL relies on absolute truths (either true or false), Real Logic employs a multi-valued system that allows for the expres-

sion of fuzzy truths, with values ranging from 0 to 1. This approach is particularly useful in contexts where truths are not rigid but subject to exceptions or uncertainties. In Real Logic, each statement can be partially true. For example, a proposition might be 70% true instead of being completely true or completely false. This type of representation is crucial in contexts where decisions must be made in the presence of vagueness, or when dealing with complex phenomena that involve degrees of ambiguity.

In conclusion, FOL and Real Logic offer powerful tools for representing and reasoning about complex knowledge. FOL provides a rigorous and formal representation of relationships between objects, while Real Logic extends this representation to handle situations of uncertainty and ambiguity, which are often present in real-world contexts.

7 Deep Learning

In the early development of AI, the field advanced rapidly, enabling systems to solve problems that, while intellectually challenging for humans, were relatively simple for computers. These problems could be described through formal rules, such as playing chess. However, the true challenge for AI lies in addressing tasks that are intuitive and straightforward for humans but difficult to formalize—tasks that we solve instinctively, such as recognizing a voice or identifying objects in images. One approach to overcoming this challenge involved creating machines capable of learning from experience and interpreting the world in terms of hierarchical concepts, where complex ideas are built from simpler ones. This gave birth to Deep Learnings [34].

The difficulty of encoding the knowledge required by AI systems highlighted the need for systems that could autonomously acquire knowledge by detecting patterns in raw data. This capability is referred to as ML. However, the performance of ML algorithms is heavily dependent on the quality of data representation. Some algorithms can learn representations of data automatically, a process known as representation learning. Nevertheless, these systems often face the challenge of factors of varation or external elements that affect the input. DL addresses this issue by introducing representations that are built upon and refined by other representations, thereby enabling more robust and scalable learning models [34].

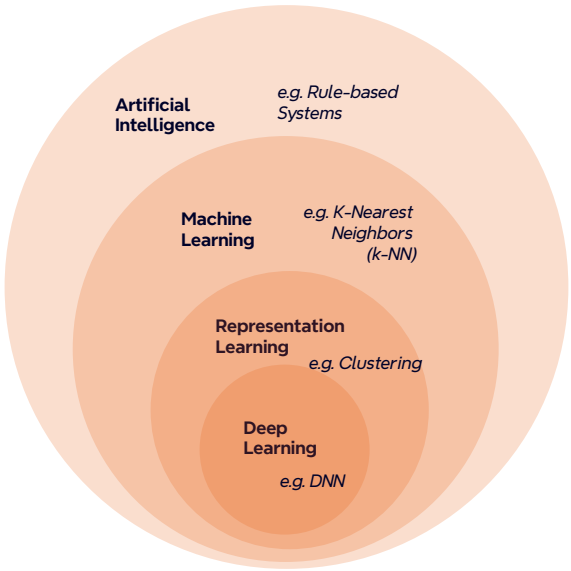


Fig. II.3 Venn Diagram illustrating the hierarchical relationship between AI, ML, Representation Learning, and DL. Each section includes an example.

The introduction of DL has revolutionized the field of AI, enabling systems to outperform traditional ML techniques in a variety of complex tasks such as image recognition, natural language processing, and autonomous systems. Unlike conventional ML models that often require manually designed features, DL models can automatically learn hierarchical representations of data.

Figure II.3 illustrates this relationship by means of a Venn diagram, which shows how DL fits within the broader categories of Representation Learning, ML, and AI.

Deep Learning is inspired by neuroscience, particularly by biological neural networks. It employs models known as Artificial Neural Networks (ANNs) that are conceptualized based on their biological counterparts [35]. DL involves computational models composed of multiple hierarchical layers, which are capable of learning complex representations through various levels of abstraction. Each layer in these models builds upon the representations learned by the preceding layers.

A prominent example of DL is the Multilayer Perceptron (MLP) [34], an extension of the perceptron [36]. An MLP is a mathematical function that maps a set of inputs to an output. Practically, it consists of at least three layers: the input layer, one or more hidden layers, and the output layer. Each node in the network, except those in the input layer, functions as a neuron that utilizes a non-linear activation

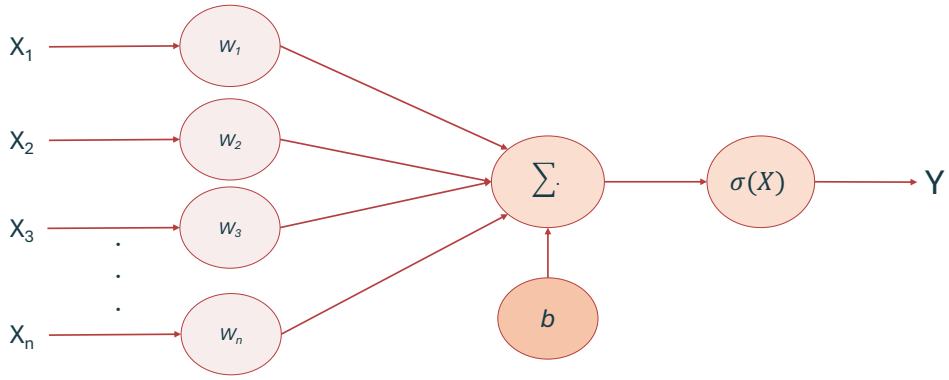


Fig. II.4 An artificial neuron.

function. This function determines the output of a neuron based on its input, and the output of the activation function is used as input to the subsequent layer. Figure II.4 illustrates a neuron.

7.1 Multilayer Perceptrons

An MLP is a type of modern artificial feedforward Neural Network made up of fully connected neurons with non-linear activation functions, organized into at least three layers. It can learn to classify non-linearly separable data [37].

Feedforward NNs, such as MLPs, are trained using the backpropagation algorithm. The MLP was developed to overcome the limitations of single-layer perceptrons, which could only classify linearly separable data.

In a feedforward network, the model maps an input X to an output $Y = f(X,\theta) = W\phi(X) + b$, where $\theta = (W,b)$ represents the network parameters, and ϕ is the activation function [34]. The goal is to approximate the true function that associates each input with its correct output.

This function is still linear in the parameters $\theta = (W,b)$, which simplifies model fitting, as the negative log-likelihood remains convex [11]. However, manually specifying the feature transformation is quite restrictive. A natural extension is to allow the feature extractor to have its own parameters, θ_2, resulting in:

$$f(X,\theta) = W\phi(X,\theta_2) + b \tag{II.13}$$

where $\theta = (\theta_1, \theta_2)$ and $\theta_1 = (W, b)$. This process can be repeated recursively to create more complex functions:

$$f(X, \theta) = f_n(f_{n-1}(\dots(f_1(f_0(X))\dots))) \tag{II.14}$$

where X is an input tensor, and each $f_j(X) = f_j(X, \theta_j)$ is a function corresponding to layer j of the network. An MLP is thus a neural network with n layers, where each layer represents a function f_j. The network learns these functions during training, refining its parameters to improve the approximation of the target function.

During forward propagation, the input passes through the network layers, generating a hypothesis. The loss function then measures the difference between the predicted output and the actual target. Backpropagation computes the gradient of the error with respect to each weight in the network, and gradient descent is used to update the weights and reduce the error. This iterative process continues until the prediction error is minimized.

MLPs form the foundational architecture of widely used networks such as Recurrent Neural Networks (RNNs) [38] and Convolutional Neural Networks (CNNs). CNNs are particularly well-suited for computer vision, where individual input elements (e.g., pixels) often hold little meaning on their own. CNNs serve as the core network in the techniques discussed throughout this work.

7.2 Convolutional Neural Networks

CNNs are foundational architectures for tasks involving image analysis. These networks utilize the convolution operation to preserve spatial relationships within the input data, which is critical in visual tasks where the correlation between neighboring pixels is key to correctly interpreting the content of the image. The convolution process allows CNNs to efficiently extract local features such as edges, textures, and other complex patterns, making them particularly suitable for image recognition and other computer vision applications.

In mathematical terms, the convolution operation can be defined as a sliding window process, where a filter (or kernel) is applied over the input data. Formally, given an input image I and a kernel K, the convolution of I with K, denoted as $(I * K)(i, j)$, is computed as:

$$(I * K)(i,j) = \sum_{m=-k}^{k} \sum_{n=-k}^{k} I(i+m, j+n) \cdot K(m,n) \tag{II.15}$$

Where:

- $I(i,j)$ is the pixel value at position (i,j) in the input image.
- $K(m,n)$ represents the value of the kernel at position (m,n), typically of size $(2k+1) \times (2k+1)$.
- $(I * K)(i,j)$ is the resulting value at position (i,j) in the output feature map.

The convolution operation results in feature maps that emphasize specific aspects of the input image, making it easier for the model to recognize patterns. Additionally, pooling layers often follow convolutional layers to downsample the feature maps, preserving important information while reducing the spatial dimensions.

CNNs are a type of neural network particularly effective in processing images and data with spatial structure. A CNN typically consists of an input, an output, and several hidden layers. Each hidden layer is usually composed of three fundamental components: the convolutional layer, the activation function, and the pooling layer.

The convolutional layer is a core building block of CNNs. Its primary purpose is to apply a set of learnable *filters* (or *kernels*) to the input data, typically images, in order to automatically extract high-level features such as edges, textures, and patterns. Each filter is a small matrix, typically of size 3×3, 5×5, or similar, which slides (or *convolves*) across the input, computing a dot product between the entries of the filter and the input at each spatial position.

The convolution operation results in a new feature map that highlights the presence of features captured by the filter. Each filter in a convolutional layer is applied to the entire input, but only within a small local region at each step (controlled by the receptive field size), drastically reducing the number of parameters compared to fully connected layers. The depth of the convolutional layer is the number of filters applied, allowing the model to capture various features in parallel. The graphical representation of a convolution operation is shown in Fig. II.5

After each convolution operation, the output is passed through a non-linear activation function, such as the *Rectified Linear Unit (ReLU)* [39], defined as:

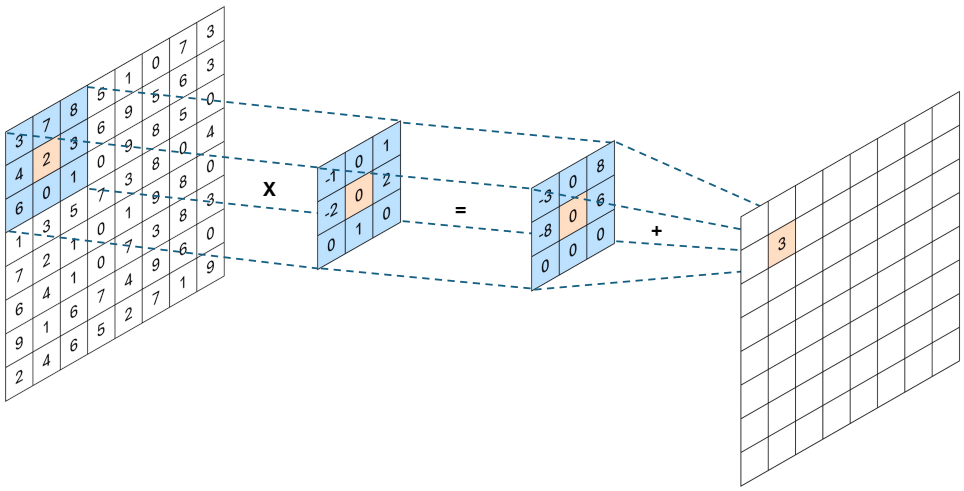


Fig. II.5 Illustration of a 2D convolution operation. The input feature map is convolved with a filter or kernel, resulting in the output feature map. The convolution is computed by sliding the filter across the input, applying an element-wise multiplication, and summing the results to produce each value in the output.

$$f(x) = \max(0, x) \tag{II.16}$$

The ReLU introduces non-linearity into the model, which is crucial because convolution is inherently a linear operation, and DL models must learn complex, non-linear patterns to capture real-world data relationships.

ReLU is widely used because it is computationally efficient and helps mitigate the vanishing gradient problem, a common issue with deep networks where gradients become too small to update the weights effectively. This problem particularly arises with activation functions like *Sigmoid* and *Tanh*, which squash input values into small ranges, causing gradients to vanish during backpropagation [11]. The Sigmoid function maps inputs to the range $(0, 1)$, while Tanh maps inputs to $(-1, 1)$:

Both Sigmoid and Tanh can cause saturation of neurons, where the derivatives of the activations are nearly zero, leading to vanishing gradients, especially in deeper layers. Hence, they are less commonly used in modern DL architectures.

The Exponential Linear Unit (ELU) [40] is another non-linear activation function that addresses some limitations of ReLU, such as the fact that ReLU can cause "dead neurons" (neurons that never activate). ELU allows for negative outputs, which improves the model's learning capacity by shifting the mean of activations closer to zero. ELU is defined as:

$$f(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \alpha(e^x - 1), & \text{if } x < 0 \end{cases} \tag{II.17}$$

where α is a hyperparameter controlling the saturation for negative inputs. ELU tends to perform better in some tasks because it introduces smoother gradients for negative values, which leads to more robust training, especially for deep models.

As shown in Figure II.6, ReLU has a sharp cutoff at zero, while ELU has a smoother curve for negative inputs, and Sigmoid and Tanh both squash the inputs into narrow ranges, leading to gradient saturation issues.

The pooling layer is another essential component that reduces the spatial dimensions (height and width) of the feature maps, thereby reducing the computational cost and the number of parameters while retaining the most relevant information. The

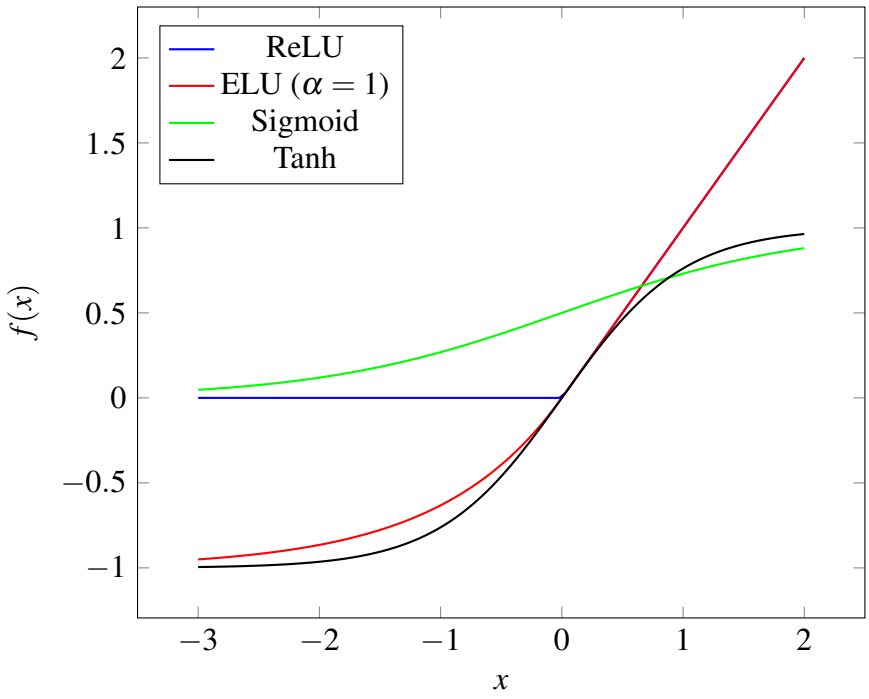


Fig. II.6 Common activation functions

pooling layer performs a downsampling operation, typically through *max pooling* or *average pooling*. These two type of pooling are shown in Fig. II.7.

Max pooling selects the maximum value within a local window (e.g., a 2×2 window). Average pooling, on the other hand, computes the average value within a local window. By retaining only the maximum or average value, pooling layers reduce the output size, making the network more computationally efficient and reducing the risk of overfitting. Additionally, pooling introduces translation invariance, meaning that small translations or distortions in the input (e.g., slight movements of an object within an image) will have minimal impact on the output, thus enhancing the robustness of the model.

Overall, the combined effect of convolution, activation, and pooling layers allows CNNs to efficiently learn hierarchical representations of data, from low-level features (e.g., edges) in the initial layers to more abstract concepts (e.g., objects or faces) in the deeper layers.

Fully Connected Layers are typically found at the end of CNNs. After a series of convolutional and pooling layers, the feature maps generated by these layers are flattened into a one-dimensional vector, which is then fed into one or more

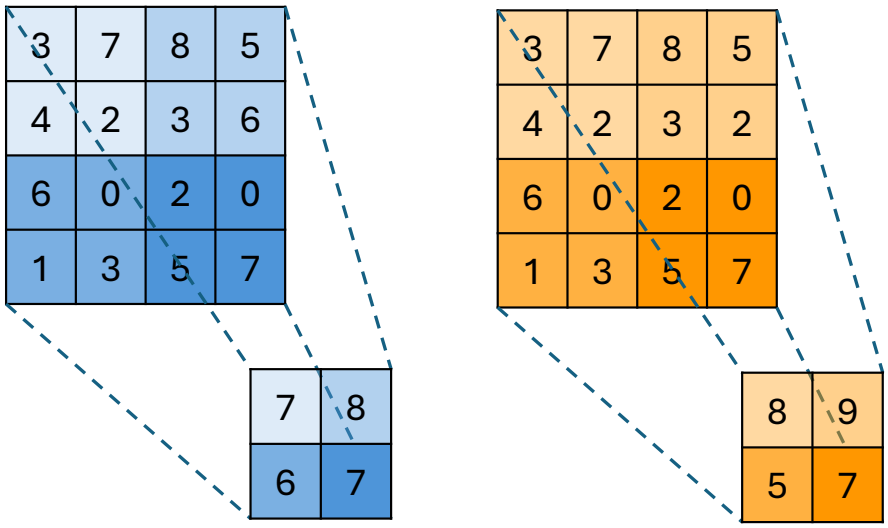


Fig. II.7 (Left) 2D max pooling operation: the input matrix is divided into regions, and the maximum value from each region forms the output. (Right) 2D average pooling operation: the average value from each region forms the output, reducing spatial dimensions while preserving key features.

fully connected layers. The purpose of the fully connected layers is to combine the features learned by the earlier layers to make a final prediction. Each neuron in a fully connected layer is connected to every neuron in the previous layer, enabling the model to learn complex combinations of features. This part of the network operates in a manner similar to traditional feedforward neural networks. The final fully connected layer often uses a softmax activation function to output class probabilities for classification tasks:

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}} \tag{II.18}$$

where z_i represents the input to the softmax function for the i-th class, and N is the number of classes. The fully connected layers play a crucial role in transforming the learned feature representations into a format suitable for classification or regression tasks. While the convolutional layers excel at feature extraction, the fully connected layers integrate this information to produce the final output of the network.

7.3 Autoencoder

Autoencoders (AEs) are a class of neural networks designed to learn an efficient and compressed representation of data while maintaining the ability to faithfully reconstruct the original input. This ability makes autoencoders powerful tools for tasks such as dimensionality reduction, data compression, denoising, and even the generation of new data similar to that in the training set.

An AE is composed of two primary components: the encoder e and the decoder d. The encoder is responsible for the first stage of the network, where it compresses the input into a lower-dimensional representation, known as the *latent space*. The decoder, in turn, reconstructs the original input from this compact latent representation. The structure of an AE is illustrated in Fig. II.8.

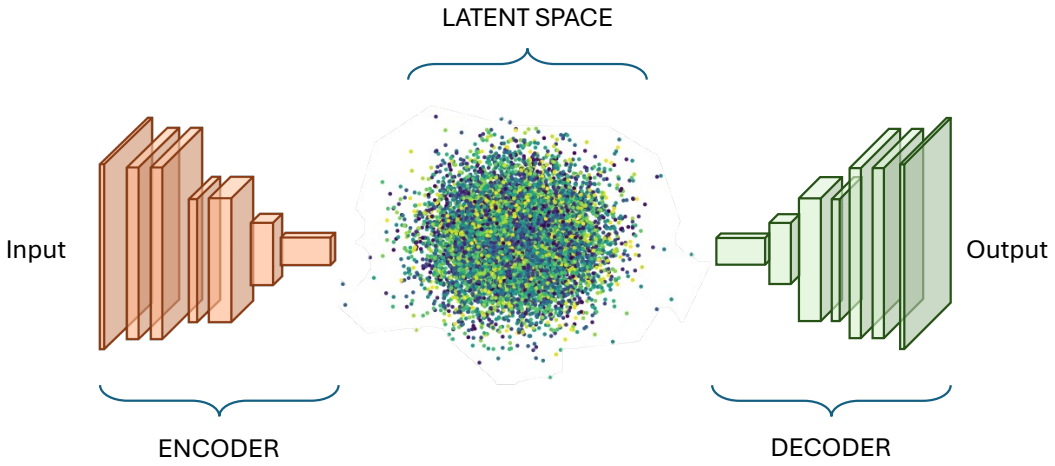


Fig. II.8 Diagram of an AE. The encoder reduces the dimensionality of the input, generating a compact latent representation, while the decoder reconstructs the original input from this latent space.

The encoder can be composed of one or more hidden layers that progressively reduce the dimensionality of the input, transforming it into a reduced representation, $\mathbf{h}$. In the case of a single hidden layer, the encoder can be described by the following function:

$$\mathbf{h} = f_{\text{encoder}}(\mathbf{x}) = \sigma(\mathbf{W}_e\mathbf{x} + \mathbf{b}_e) \tag{II.19}$$

where $\mathbf{x}$ is the input vector, $\mathbf{W}_e$ represents the encoder weights, $\mathbf{b}_e$ are the biases, $\sigma(\cdot)$ is the activation function (such as ReLU or sigmoid), and $\mathbf{h}$ is the vector representing the latent space.

However, if the encoder is composed of multiple layers, the transformation is progressively applied across each layer. In the case of L layers, the encoder can be described as a composition of transformations for each layer l:

$$\mathbf{h}^{(l)} = \begin{cases} \sigma_1(\mathbf{W}_l\mathbf{x} + \mathbf{b}_l) & \text{if } l = 1, \\ \sigma_l(\mathbf{W}_l\mathbf{h}^{(l-1)} + \mathbf{b}_l) & \text{if } l \in \{2, 3, \ldots, L\}. \end{cases} \tag{II.20}$$

where $\mathbf{h}^{(l)}$ represents the output of the l-th layer, $\mathbf{W}_l$ and $\mathbf{b}_l$ are the weights and biases for the l-th layer, and $\sigma_l(\cdot)$ is the activation function applied at the l-th layer.

Thus, in the case of multiple layers, the final output of the encoder $\mathbf{h}^{(L)}$ is obtained through a series of transformations applied sequentially to the input $\mathbf{x}$.

The dimension of $\mathbf{h}$ is usually much smaller than that of the original input, making the representation $\mathbf{h}$ a compressed and abstract version of the input. The latent space is a reduced and dense space where the autoencoder encodes the most relevant information from the input. This space captures the main characteristics of the input, allowing for data compression. The size of the latent space represents a trade-off between the amount of information lost and the level of compression: a latent space that is too small may result in an inaccurate reconstruction of the input, while one that is too large reduces the effectiveness of compression.

The second component, the decoder, is responsible for reconstructing the original input from the latent space $\mathbf{h}$. The decoder uses an inverse process compared to the encoder, gradually expanding the vector $\mathbf{h}$ back to a size equal to that of the original input. The decoder can be described with the following function:

$$\hat{\mathbf{x}} = f_{\text{decoder}}(\mathbf{h}) = \sigma(\mathbf{W}_d\mathbf{h} + \mathbf{b}_d) \tag{II.21}$$

Where $\mathbf{W}_d$ are the decoder weights, $\mathbf{b}_d$ are the biases, and $\hat{\mathbf{x}}$ is the reconstructed output, which approximates the original input $\mathbf{x}$.

The autoencoder is trained by minimizing the difference between the original input $\mathbf{x}$ and its reconstruction $\hat{\mathbf{x}}$. This is typically done using a loss function such

as the MSE. During training, the autoencoder learns the weights $\mathbf{W}_e$ and $\mathbf{W}_d$ that minimize this loss, optimizing its ability to compress and reconstruct the data.

Beyond standard autoencoders, several variants exist: Denoising, Sparse, and Variational Autoencoders. Sparse autoencoders impose a sparsity penalty on the latent space as a training criterion. This type of autoencoder performs well in tasks such as classification. Denoising autoencoders are used for removing noise from data. During training, noise is added to the input, but the model is trained to reconstruct the "clean" version of the input. This helps the model learn more robust representations. Variational Autoencoders (VAEs) are a type of autoencoder that learns a probability distribution in the latent space, rather than a single deterministic representation. This allows for the generation of new data by sampling from the latent space, making VAEs generative models. The loss function for VAEs includes a term that measures the difference between the learned distribution and a multivariate normal distribution.

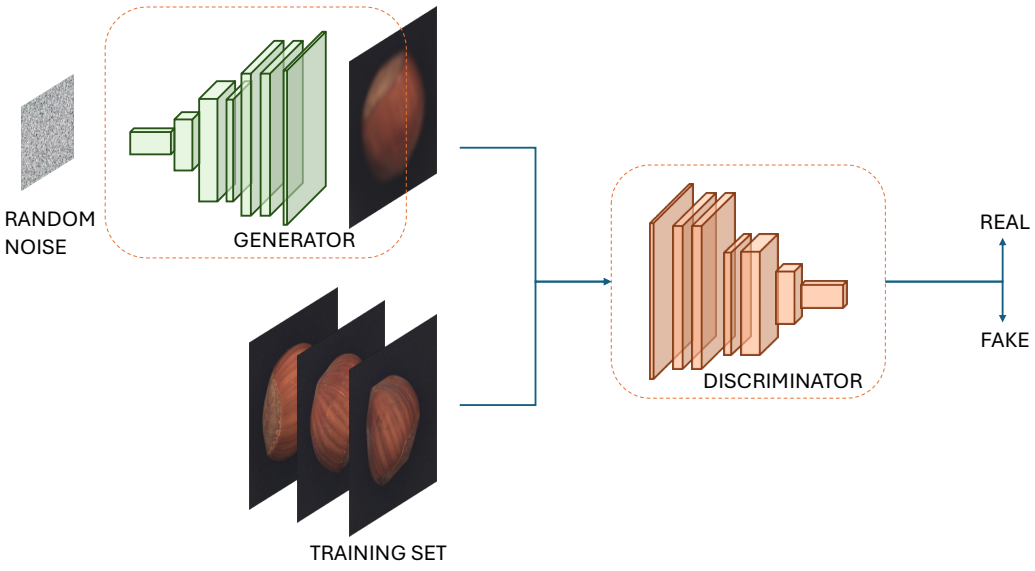


Fig. II.9 Diagram of a GAN. The Generator creates synthetic data from random noise, while the Discriminator evaluates both real data from the training set and generated data, classifying them as *real* or *fake*

While autoencoders focus on compressing and reconstructing data, another powerful class of generative models, known as Generative Adversarial Networks (GANs), takes a different approach to data generation. Introduced by Goodfellow et al. [41], GANs are designed to generate entirely new data that mimics the distribution

of the training set. Unlike autoencoders, which learn to reproduce their input, GANs consist of two neural networks—the *generator* and the *discriminator*—that engage in a competitive process. The generator creates synthetic data by sampling from a latent space, while the discriminator tries to differentiate between real and generated data. Fig II.9 shows a GAN schema. The generator's objective is to produce data so realistic that the discriminator cannot distinguish it from real samples. This training dynamic is framed as a minimax game, where the generator seeks to minimize the classification error of the discriminator, and the discriminator aims to maximize it. GANs have shown exceptional performance in generating high-quality images, video synthesis, and even data augmentation, making them a versatile tool in modern ML applications.

7.4 Deep Clustering

Clustering plays a pivotal role in ML and data mining, offering insight into the inherent structure of data without requiring labels. As discussed in Se. 4.4, in traditional clustering methods, the success largely depends on the choice of handcrafted features and distance metrics. However, these approaches often struggle when dealing with high-dimensional, complex datasets. To address this, deep clustering has emerged, combining DL's ability to automatically learn hierarchical representations with conventional clustering algorithms. This hybrid approach allows for feature learning directly from raw data, enabling Deep Neural Networks (DNNs) to simultaneously optimize feature representations and cluster assignments. By leveraging gradient-based optimization techniques, deep clustering enhances intra-cluster similarity and maximizes inter-cluster separation, making it highly adaptable to a variety of datasets.

Later in this thesis, we illustrate work based on Deep Embedding for Clustering (DEC), introduced by Xie et al. [42], which integrates deep neural networks to jointly learn feature representations and cluster assignments. DEC projects data into a low-dimensional feature space where clustering objectives are optimized iteratively. The model consists of two primary components: the encoder, which learns a latent representation, and the clustering layer, which computes a soft assignment q for each sample, indicating its probability of belonging to a specific cluster. A schema of DEC is shown in Fig. II.10 The training loss is defined as the Kullback-Leibler (KL)

divergence between the soft assignment q and a target distribution p, facilitating the refinement of both the feature space and the clustering.

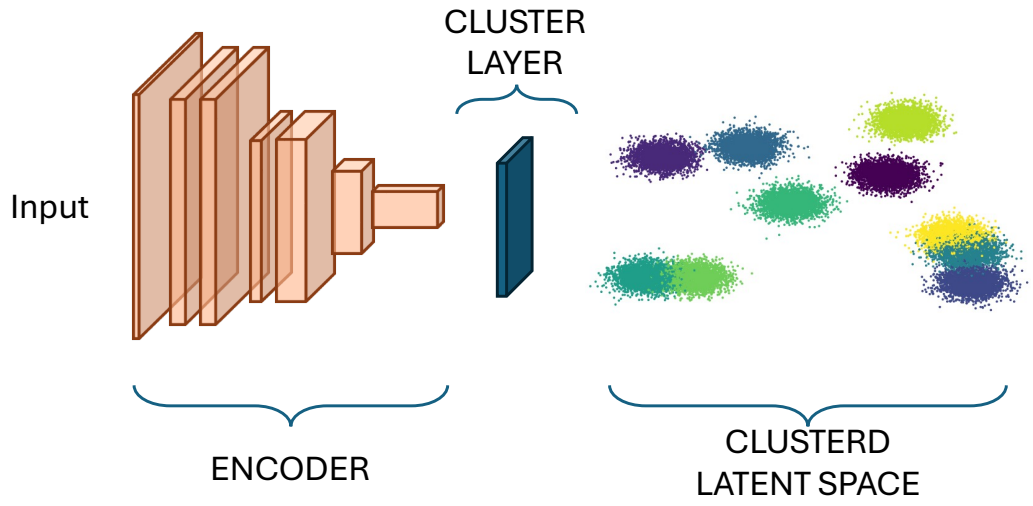


Fig. II.10 Schematic of a DEC model. The encoder maps input data into a low-dimensional latent space, where a clustering layer organizes the data into clusters. This approach jointly optimizes the embedding and cluster assignments for unsupervised learning tasks.

To initialize the DEC, an AE is first pre-trained to establish the initial parameters. This pre-training significantly reduces the computational effort required to achieve clustering. The AE's encoder forms the backbone of the DEC, and a clustering layer is added on top to generate the soft assignments. Both the centroids and the encoder parameters, Θ, are trainable using stochastic gradient descent, allowing the feature space to adapt dynamically as clustering progresses. In [42], KL divergence was employed to guide the optimization.

The clustering process in DEC consists of two phases. First, a soft assignment is computed between each embedded point and each cluster, using a Student's t-distribution kernel [43] to measure similarity:

$$q_{ij} = \frac{\left(1 + \left\|z_i - \mu_j\right\|^2 / \alpha\right)^{-\frac{\alpha+1}{2}}}{\sum_{j'} \left(1 + \left\|z_i - \mu_{j'}\right\|^2 / \alpha\right)^{-\frac{\alpha+1}{2}}} \tag{II.22}$$

where, z_i represents the embedded feature point for sample i, and μ_j is the centroid of cluster j. Variable j' refers to all possible cluster centroids, and α is the parameter controlling the degrees of freedom.

The second phase involves updating both the cluster centroids and the deep mapping parameters, by learning from high-confidence assignments via an auxiliary target distribution. The target distribution p, defined in Equation II.23, aims to emphasize data points with higher confidence:

$$p_{ij} = \frac{q_{ij}^2 / f_j}{\sum_{j'} q_{ij'}^2 / f_{j'}} \tag{II.23}$$

where $f_j = \sum_i q_{ij}$ represents the soft cluster frequency. The network is then trained to minimize the KL divergence between the soft assignments q and the target distribution p:

$$\mathscr{L}_{kld} = \mathrm{KL}(P\|Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{II.24}$$

By iteratively refining the cluster assignments, DEC enhances clustering performance, making it more resilient to varying datasets and hyperparameters, as demonstrated in [42].

8 Metrics

In both ML and DL, selecting appropriate metrics to evaluate model performance is crucial. These metrics help to assess how well a model is performing, and different tasks (e.g., classification, regression) require specific types of metrics. In this section, we will explore some of the most commonly used metrics in ML and DL, particularly for classification tasks. Additionally, we will discuss two common issues in model performance: overfitting and underfitting, which can significantly impact the effectiveness of these metrics.

Confusion Matrix The confusion matrix is a table that summarizes the performance of a classification model by showing the actual versus predicted labels. For a binary classification problem, the confusion matrix consists of four cells:

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

This matrix serves as the basis for calculating various metrics like Accuracy, Precision, Recall, and F1 Score.

Accuracy Accuracy is one of the most straightforward metrics and measures the proportion of correctly predicted instances out of the total number of instances. It is defined as:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{II.25}$$

While Accuracy provides a quick snapshot of model performance, it can be misleading in the case of imbalanced datasets, where the number of instances in one class far exceeds the other.

Precision Precision measures the proportion of correctly predicted positive instances out of all instances predicted as positive. It is defined as:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{II.26}$$

Precision is particularly useful when the cost of false positives is high, such as in medical diagnoses where a false positive could lead to unnecessary treatments.

Recall Recall, also known as Sensitivity or True Positive Rate (TPR), measures the proportion of actual positives that are correctly identified by the model. It is given by:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{II.27}$$

Recall is crucial when missing a positive instance (i.e., a false negative) is costly, as in cases like fraud detection or disease screening.

F1 Score The F1 Score is the harmonic mean of Precision and Recall, providing a balanced metric that accounts for both false positives and false negatives. It is particularly useful when there is an uneven class distribution and one needs to balance

the importance of Precision and Recall. The F1 Score is defined as:

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{II.28}$$

A higher F1 Score indicates a better balance between Precision and Recall.

Area Under the Curve Receiver Operating Characteristic (AUCROC) The AUCROC is a popular metric for binary classification problems. It measures a model's ability to distinguish between positive and negative data points across various classification thresholds. The ROC curve plots the TPR (Recall) against the False Positive Rate (FPR), where FPR is defined as:

$$\text{FPR} = \frac{FP}{FP + TN} \tag{II.29}$$

The AUC represents the area under the ROC curve, and it ranges from 0 to 1, with 1 being a perfect classifier. A higher AUC indicates that the model has better discriminatory power.

Area Under the Precision-Recall Curve (AUPRC) For highly imbalanced datasets, the AUCROC might not be sufficient, as it may still present an overly optimistic picture. In such cases, the AUPRC is a more appropriate metric. The Precision-Recall curve plots Precision against Recall, focusing more on the performance with respect to the minority class. A high AUPRC value indicates a better trade-off between Precision and Recall.

While both AUCROC and AUPRC provide valuable insights, they serve different purposes. For example, in a dataset where only 5% of the instances belong to the positive class, the ROC curve may still show a reasonable AUC because the model performs well with the majority class. However, the Precision-Recall curve may reveal that the model struggles to identify positive cases, as it provides a more focused view on Precision for the positive class.

Overfitting and Underfitting In the process of evaluating model performance using these metrics, it is crucial to address overfitting and underfitting, two common issues that can hinder a model's ability to generalise. Fig. II.11 shows the differences between underfitting, good fit, and overfitting in polynomial regression models.

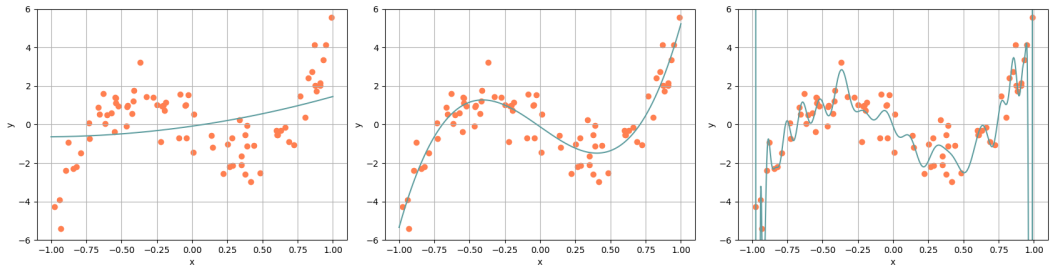


Fig. II.11 Demonstration of underfitting (left), good fit (center), and overfitting (right) in polynomial regression models.

Overfitting occurs when a model becomes too complex and learns the noise in the training data along with the actual patterns, leading to high performance on the training data but poor performance on unseen data. Overfitting can be mitigated by reducing model complexity, using regularization techniques (such as L1 or L2 regularization), applying dropout in neural networks, or increasing the size of the training dataset. Cross-validation is also a useful tool for ensuring the model generalises well to new data.

Underfitting, on the other hand, happens when a model is too simplistic and fails to capture the underlying patterns in the data. This results in poor performance on both the training and validation datasets. Underfitting can be addressed by increasing model complexity, improving feature engineering, or training the model for a longer period.

Balancing these two phenomena is key to developing a model that generalises well, which ultimately ensures that the metrics discussed earlier provide a true reflection of the model's performance.

Evaluating the performance of ML and DL models requires careful selection of metrics tailored to the problem at hand. While metrics like Accuracy are simple and intuitive, they may not always provide a complete picture, especially in cases of imbalanced datasets. Metrics like Precision, Recall, and the F1 Score offer a more comprehensive evaluation. Furthermore, advanced metrics such as AUCROC and AUPRC provide deeper insights into a model's ability to handle trade-offs between classes. Finally, addressing overfitting and underfitting is crucial for ensuring that the selected metrics provide a reliable assessment of the model's generalization performance.

9 Neuro-Symbolic AI

Up to now we explore various aspects and algorithms of symbolic AI and neural networks. Symbolic AI excels at generating logical conclusions, works best with small, well-defined datasets, and often requires human intervention for effective implementation. It leverages symbols that are generally understandable to both experts and non-experts, allowing it to reason and represent knowledge in a way that is transparent. On the other hand, neural AI employs a layered structure, capable of extracting increasingly complex information from raw data, working most effectively with large datasets by learning patterns and associations. Inspired by the biological workings of the human brain, neural AI consists of interconnected layers that activate with varying intensity depending on the task. This makes it highly efficient, but its processes are often difficult to interpret, even for experts.

Symbolic AI encompasses all techniques that rely on high-level symbolic representations, understandable by humans, to address problems through logic and reasoning. The reasoning here refers to drawing conclusions from known information using logical principles and rules. Common approaches in symbolic AI include rule-based systems (e.g., IF-THEN rules) or FOL using predicates. In contrast, neural networks are non-linear statistical models capable of capturing complex relationships between inputs and outputs that are difficult to express using traditional analytical functions.

To overcome the limitations of both paradigms, Neuro-Symbolic AI (NeSy AI) was developed, aiming to harness the strengths of both symbolic reasoning and neural learning. This approach seeks to combine the reasoning and explainability of symbolic AI with the powerful capabilities of neural networks, reducing the data-intensive nature of training by incorporating pre-existing knowledge and logical reasoning. NeSy AI harmonizes the ability to learn from data (neural aspect) with the capacity to reason on the acquired knowledge (symbolic aspect), striving to merge the robustness of neural networks with the interpretability and structured reasoning of symbolic AI [44].

For this reason, NeSy AI has garnered significant attention as a promising approach for advancing AI research [45, 46]. The integration of these two paradigms remains an active area of research, offering many challenges but also exciting poten-

tial. In the following sections, we will explore state-of-the-art approaches in the field of NeSy AI.

9.1 Relational Embeddings

In this subsection, we explore approaches that map objects or concepts into continuous vector spaces, where the distances or similarities between vectors reflect semantic or logical relationships between the objects themselves. These methods utilize various ML techniques, particularly neural networks, to learn such distributed representations of data. The goal is to preserve semantic or logical relationships throughout the embedding process. By embedding logical or semantic relationships within these distributed representations, models can process and infer such relationships. Embedding techniques have been prominently featured in recent NeSy AI research [47–49]. Training for these systems often uses backpropagation, leading to the development of systems referred to as *relational embeddings*, which represent relational predicates within neural networks. In this section, we will focus on one popular approach from recent years: Logic Tensor Network (LTN)[50].

Logic Tensor Networks LTNs represent a neuro-symbolic framework that combines symbolic AI with neural computation, providing a unified formalism for addressing various AI tasks [50]. LTNs introduce a differentiable form of FOL, known as Real Logic (see Secrion 6), into DL models. This integration enables the use of logical operators within neural network computations, allowing the model to reason about relationships between inputs [50].

One of the key strengths of LTNs is their ability to integrate symbolic and neural representations within a single framework. This makes them particularly suited to handling both structured and unstructured data, with applications in fields like natural language processing and computer vision [51]. LTNs extend the Neural Tensor Network model [52], which is a state-of-the-art method for relational embeddings, by representing complex logical structures using FOL formulas. LTNs adopt Real Logic as their formalism, an infinite-valued fuzzy logic language. Real Logic operates within a first-order language $\mathscr{L}$, which includes $\mathscr{C}$ (a set of constant symbols representing objects), $\mathscr{F}$ (a set of functional symbols), $\mathscr{P}$ (a set of relational symbols), and $\mathscr{X}$ (a set of variable symbols).

In LTNs, the loss function is built upon the concept of satisfiability within Real Logic. Specifically, the model is trained to maximize the truth value of logical formulas by minimizing a loss term that quantifies the degree of violation of these formulas. This loss term, commonly referred to as the satisfiability loss, measures how far the current state of the neural network's output is from satisfying the logical constraints encoded in the knowledge base [51]. The satisfiability loss can be defined as,

$$\textbf{\textit{SAT loss}} = 1 - \text{SatAgg}(\phi) \tag{II.30}$$

where SatAgg represents an aggregation of the truth values of all formulas in the knowledge base ϕ. Thus, the training process involves optimizing the parameters of the neural network to minimize this loss, ensuring the network learns representations that adhere to both the data and the logical constraints.

In real-world applications, degrees of truth and exceptions are common, which makes the use of fuzzy semantics essential. In this form of logic, domains are interpreted by tensors in the real field. The process known as grounding involves replacing variables in formulas with constants or variable-free terms, sometimes referred to as *instantiation* [44]. When grounded, terms become tensors of real numbers, and formulas are mapped to real numbers in the range $[0,1]$, representing truth values.

By employing Real Logic, objects are represented as points within a feature space, and features and predicates can be learned. LTNs are trained to approximate optimal satisfiability [51], making inference efficient through feedforward propagation. Figure II.12 illustrates an example of an LTN representing the formula $\forall(x,z)(P(x) \wedge Q(z))$. In summary, LTNs mark a significant step towards developing more flexible and interpretable DL models that can perform reasoning tasks akin to human cognition [51].

9.2 End-to-End Approaches

In this section, we explore end-to-end approaches, which refer to systems or architectures that address the entire sequence of operations from input data to final output in a self-contained manner, without relying on external components. These approaches represent a unified workflow that integrates the entire process, both during training

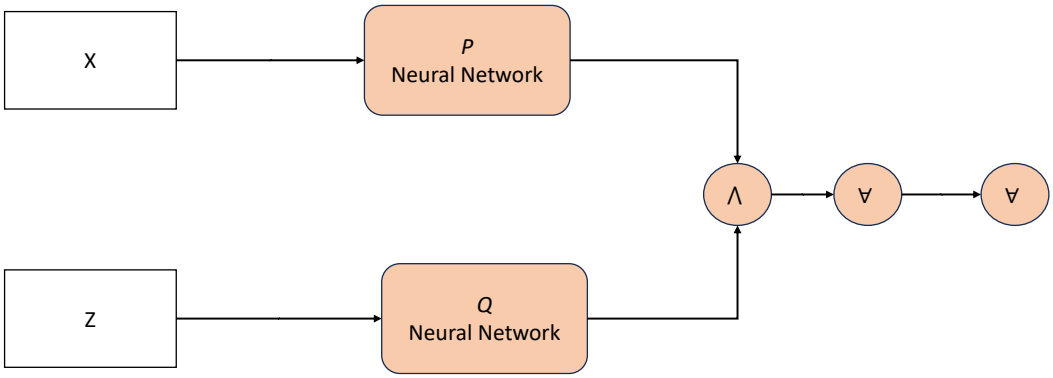


Fig. II.12 Logic tensor network for $\forall(x,z)(P(x) \wedge Q(z))$

and inference. We discuss one end-to-end system: DeepProbLog that integrates probabilistic approaches, ProbLog, with neural networks.

DeepProbLog Among the most notable neuro-symbolic models are those that integrate PLP with DL, particularly the frameworks like DeepProbLog [53, 54] and DeepStochLog [55]. These models have demonstrated substantial versatility across several domains, including image interpretation, probabilistic reasoning, and the handling of complex structured representations. By combining the strengths of probabilistic logic with neural networks, these systems are able to learn from data while also reasoning about uncertainty and managing complex relationships within the data.

DeepProbLog extends traditional logic programming by incorporating probabilistic annotations, allowing uncertainty to be modeled within a logical framework. At the core of DeepProbLog is the integration of Neural Annotated Disjunctions (NADs), which allows neural networks to be linked directly with logical predicates. A NAD takes the following form:

Definition 1 ***Neural Annotated Disjunction (NAD).*** *A NAD is expressed as:*

$$nn(m_r, \boldsymbol{I}, O, \boldsymbol{d}) :: r(\boldsymbol{I}, O), \tag{II.31}$$

where nn is a reserved predicate, m_r refers to a neural network with k inputs and n outputs, and $\boldsymbol{I} = I_1, \ldots, I_k$ denotes the inputs to the network. O is the output variable, and $\boldsymbol{d} = d_1, \ldots, d_n$ represents the set of possible output classes. The predicate r refers to the so-called neural predicate [53].

In practice, a ground NAD refers to the specific instance of this disjunction, where the neural network's output probabilities are applied to grounded inputs. For example, in a card game application, such as determining if a hand contains a specific combination (e.g., a poker hand), DeepProbLog can process images of cards using neural networks. A predicate like $game([C_0, C_1, C_2, C_3, C_4], Outcome)$ can be defined, where the list $[C_0, C_1, C_2, C_3, C_4]$ represents card images, and $Outcome$ is a binary value that indicates whether the hand meets the target condition (e.g., poker hand or not). Neural networks can be employed to classify the individual card images.

For instance, a NAD can be defined as follows:

$$nn(m\_card, [X], Y, [jack, queen, king, ace]) :: rank(X, Y), \tag{II.32}$$

where the network m_card classifies individual cards, and X represents the image of a card, while Y corresponds to the predicted rank (e.g., jack, queen, etc.). Given a specific card image, such as  , the following ground NAD would be generated:

$$\begin{aligned} &nn(m\_card, [\,\text{  }\,], jack) :: rank(\text{}, jack)\ ; \\ &nn(m\_card, [\,\text{  }\,], queen) :: rank(\text{}, queen)\ ; \\ &\ldots\ ;\ nn(m\_card, [\,\text{  }\,], ace) :: rank(\text{}, ace). \end{aligned} \tag{II.33}$$

Upon evaluation, this would generate a grounded probabilistic program:

$$p_0 :: rank(\text{  }, jack)\ ;\ \ldots\ ;\ p_3 :: rank(\text{}, ace), \tag{II.34}$$

where $[p_0, \ldots, p_3]$ represents the output probabilities for each class produced by the neural network m_card.

DeepProbLog works by transforming logic programs into probabilistic programs during inference. The program is grounded with respect to the query, neural networks generate predictions for the relevant inputs, and the resulting grounded NADs are integrated into a ProbLog program. The program is compiled into a propositional formula and then into an arithmetic circuit, which is evaluated to compute the final probabilities for each query.

Regarding learning, DeepProbLog optimizes both neural parameters and logic program parameters in a unified manner. The loss function typically involves minimizing the negative log probability of the correct answers:

$$\arg\min_{\theta} \mathscr{L} = \arg\min_{\theta} \frac{1}{|E|} \sum_{(q,1)\in E} -\log P_{\theta}(q), \tag{II.35}$$

where $\mathscr{L}$ represents the loss, θ is the vector of parameters (both neural and symbolic), and $P_{\theta}(q)$ is the probability assigned to query q by the DeepProbLog program. This optimization is carried out using gradient descent, ensuring the neural network and probabilistic parameters are adjusted together.

End-to-end systems like DeepProbLog offer robust solutions for managing uncertainty and handling complex reasoning tasks by integrating probabilistic logic programming with DL. These systems enable reasoning over both data-driven and symbolic knowledge, and they allow for more interpretable decision-making processes. However, such models also face challenges, particularly in terms of scalability and the computational cost of integrating symbolic logic with DL. Furthermore, these approaches require specialized techniques to balance the complexity of neural and symbolic learning, ensuring that they are applicable to real-world settings.

9.3 Hybrid Integration

Hybrid integration in NeSy AI refers to approaches that combine different components or methods within a system, such as ML models working alongside logical rules or neural networks incorporating symbolic constraints. These systems maintain distinct neural and symbolic sub-components that collaborate to enhance overall performance, rather than fully integrating the two paradigms. For istances, in the work of Fadja et al. [56], a CNN is combined with a random forest through Hierarchical Probabilistic Logic Programming [30] to create an interpretable system for early-stage prediction of critical states in COVID-19 patients. Various frameworks employ hybrid integration strategies [57, 58]. The Neuro-Symbolic Concept Learner (NSCL) [57] is a model that merges neural and symbolic reasoning for tasks such as visual reasoning and unsupervised learning of visual concepts, words, and sentences. NSCL combines object-based scene representations with the translation of natural language queries into executable symbolic programs, which are applied to a latent scene representation. It is commonly used in tasks involving the CLEVR dataset [59], which features images of geometric objects paired with reasoning-based questions. The NSCL model consists of a visual perception component, responsible

for generating object masks and extracting features; neural operators for mapping object representations to attribute spaces; and a quasi-symbolic executor that produces probabilistic predictions from parsed queries, enabling the learning of object concepts without explicit labeling.

The NS-DR [60] model combines separate neural and symbolic submodels to address dynamic reasoning and semantic parsing tasks. It has been applied successfully to the CLEVRER video reasoning dataset, which extends CLEVR to videos instead of static images. The model includes several key components: a video frame parser, a neural dynamics predictor using PropNet [61] to model object interactions and predict motion, a question parser based on a Seq2Seq model, and a symbolic program executor. Unlike NSCL, NS-DR's symbolic executor uses fully non-differentiable operations and relies on supervised training to learn concepts. Hybrid NeSy AI systems, such as NS-DR, offer advantages by combining the pattern recognition abilities of neural networks with the interpretability of symbolic reasoning. However, they introduce challenges in complexity, training, and interpretability, which must be carefully addressed to realize their potential in practical applications.

Hybrid NeSy AI systems, which combine neural networks with symbolic reasoning, offer a promising approach to integrating expert knowledge into tasks. By leveraging the strengths of neural networks to detect complex patterns and symbolic logic to provide interpretability and rule-based reasoning, hybrid systems can uncover relationships and insights that may be difficult for purely neural models to detect. Additionally, hybrid approaches allow for greater adaptability, making it easier to update models with new knowledge.

However, hybrid systems also introduce complexity. The integration of neural and symbolic components requires specialized skills and resources to implement effectively, and balancing the neural and symbolic components during training can be challenging. While hybrid systems improve interpretability compared to purely neural models, understanding the final decisions made by the system can still be difficult due to the combination of symbolic logic and neural processing.

9.4 Neural-Symbolic Reinforcement Learning

Neural-Symbolic Reinforcement Learning combines the strengths of Reinforcement Learning (RL) with symbolic reasoning to address complex tasks such as game

playing and robot navigation, demonstrating the ability to learn and reason over sophisticated behaviors [62]. This approach leverages the adaptability of RL while incorporating the interpretability and rule-based reasoning of symbolic methods, creating models that can learn from interactions with their environment and reason about those interactions to improve performance.

One of the significant challenges in this field is the continuous validation of neural networks within a learning loop, particularly in the context of ensuring safe deep RL [63]. To address this, researchers have developed the *Estimate and Replace* strategy, which incorporates external symbolic reasoning or pre-existing applications into DL architectures. This method, as demonstrated by the EstiNet model's success in table-based question-answering tasks, allows the system to learn effectively from fewer data samples [64].

Another advancing area in neuro-symbolic programming is the integration of DL with program synthesis, where neural networks are trained to generate programs from data. Techniques such as neural program synthesis and neural program induction enable models to create programs based on input/output examples or latent program representations, further bridging the gap between DL and symbolic reasoning [65–67].

While deep RL has achieved notable success across various domains, one of its limitations is its lack of reasoning capabilities [68]. To address this, researchers are increasingly exploring ways to incorporate symbolic knowledge into RL frameworks. Deep Symbolic Reinforcement Learning (DSRL), proposed by Garnelo et al. [69], is a key approach to tackling this issue. DSRL integrates a neural back-end, which processes raw sensor data, with a symbolic front-end that uses this information to generate high-level strategies and plans.

Building on the foundations of DSRL, Symbolic RL with Common Sense, introduced by Garcez et al. [70], enhances both the learning and decision-making phases of RL by incorporating common-sense knowledge.

The integration of RL with symbolic reasoning offers several advantages. RL provides the flexibility to make sequential decisions in dynamic environments, while symbolic logic contributes precision and the ability to incorporate expert rules.

Chapter III

Neuro-Symbolic Integration

In this chapter, we delve into the integration of neural and symbolic methods to address complex tasks requiring both predictive accuracy and interpretability. The neuro-symbolic approach provides a unique framework that combines the strengths of neural networks—such as deep learning for pattern recognition—with the explainability and reliability of symbolic reasoning systems.

Several aspects discussed in this chapter have been explored in our previous works. For instance, techniques for integrating symbolic reasoning with neural networks, particularly using neuro-symbolic systems based on hybrid architectures such as HPLP, have been introduced in the papers *Machine Learning Techniques for Extracting Relevant Features from Clinical Data for COVID-19 Mortality Prediction* [71], *Neural-Symbolic System for Predicting COVID-19 Positivity* [72], and *Neural-Symbolic Ensemble Learning for Early-Stage Prediction of Critical State of COVID-19 Patients* [56].

Furthermore, methodologies related to the integration between constrained optimization and deep networks are reviewed in *Integration between Constrained Optimization and Deep Networks: A Survey* [73]. Additional contributions that influence the topics of this chapter include our work on leveraging visual explanations from CNNs for improving decision-making in complex systems, as explored in *Exploiting CNN's Visual Explanations to Drive Anomaly Detection*.

Lastly, the neuro-symbolic approaches for enhancing decision processes in various domains are further developed from insights in *A Neuro-Symbolic AI Network Intrusion Detection System* [74] and *Neuro-symbolic Integration for Open Set Recog-*

nition in Network Intrusion Detection [75]. These works form the foundation for the neuro-symbolic integration approaches discussed in this chapter.

10 Neuro-Symbolic Ensemble Learning

In domains such as healthcare, law and security, reliability and explainability are of utmost importance. A clear example is diagnosing Covid-19. In response tpo the Covid-19 emergency, computer scientists worked closely with healthcare professionals to develop various ML models for diagnosing Covid-19 patients using Computed Tomography (CT) scans and clinical data. In this case, it is important not only to diagnose the disease accurately bat also to provide explanations for the decisions made.

In this Chapter, we present *Neural HPLP*, a NeSy approach that emerged from the need for an explainable system during the Covid-19 emergency.

The architecture of this system is outlined below, in Chapter 14 we will discuss the medical case study and the results.

The architecture of Neural HPLP is shown in Figure III.1. The system consists of three components: a symbolic, a neural, and a neuro-symbolic part. The first component includes DTs and RFs; the second involves a CNN; and finally, the third is a HPLP that combines the outputs of the other two components to generate explainable rules.

The DT analyses tabular data and provides a prediction (e.g. the severity of the patient's condition), while of the neural network analyses images and returns the prediction (e.g. 3D convolutional networks on CT scans predicting the severity of lung elisions). Using this information, HPLP generates probabilistic rules that can not only predict but also provide an explanation.

HPLP [30, 76] are an extension of Liftable PLP [77]. Neural HPLP, learns a target predicate using a set of examples called interpretations.

To compute the probability of istances of a target predicate r using a PLP, we focus on the probability of a ground atom $r(t)$, where t is a vector of terms. We use a specific form of probabilistic logic program that defines r in terms of *input predicates*, whose definition is given as input and is certain, and *hidden predicates*

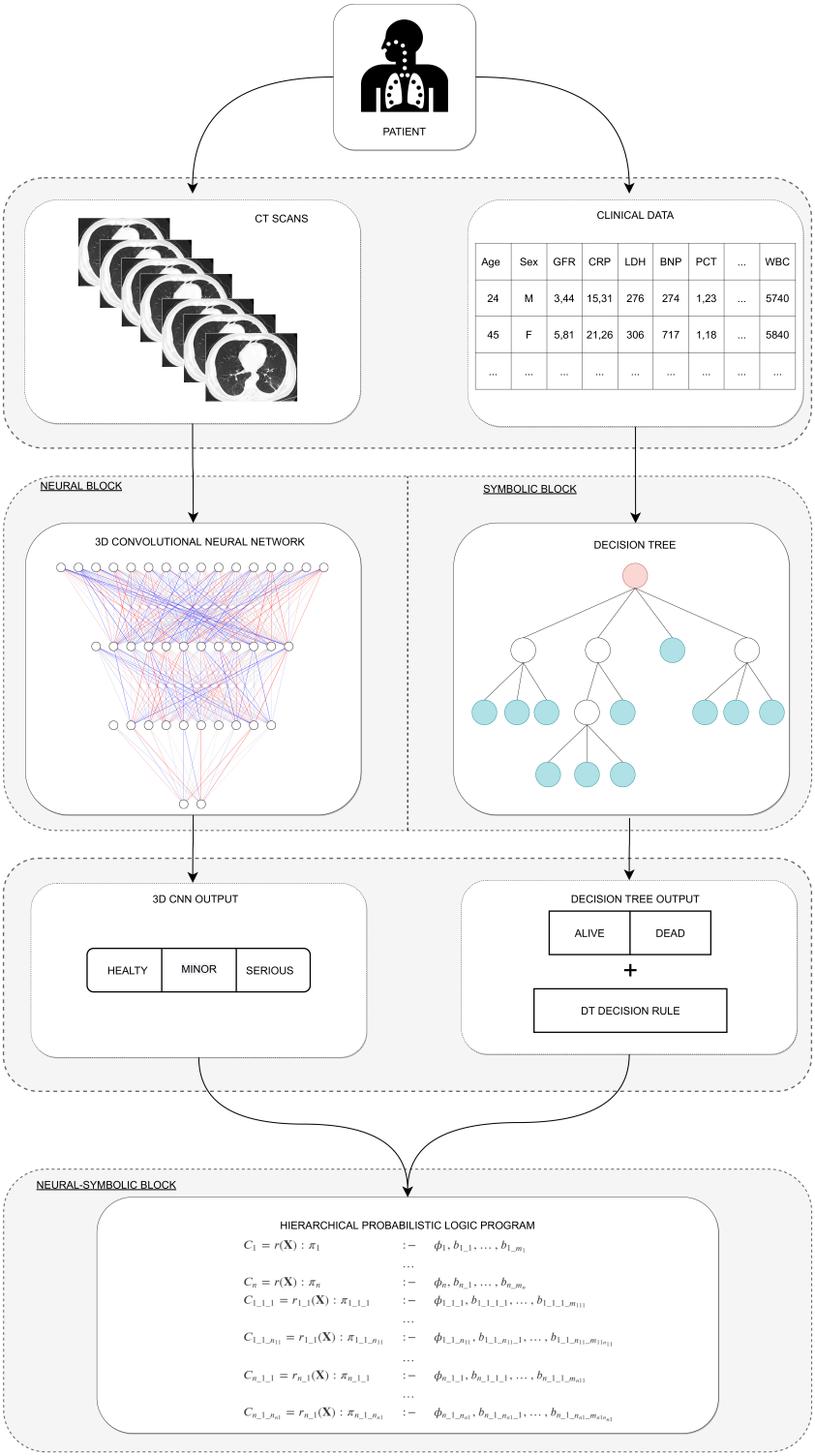


Fig. III.1 Neuro-Sylmolic system: DT and 3D-CNN are integrated using HPLP

that are defined by program clauses. Input predicates encapsulate input data and background knowledge, while the target predicate is the one we aim to predict. Hidden predicates are distinct from input and target predicates. Each clause in the program has a single head atom annotated with a probability. The program is hierarchically structured so that each layer defines a set of hidden predicates based on predicates from the layer immediately below or input predicates.

A generic clause C in this hierarchical program can be represented as follows:

$$C = P(X) : \pi : -\phi(X,Y), b_1(X,Y), b_2(X,Y), \ldots, b_m(X,Y) \tag{III.1}$$

where $\phi(X,Y)$ is a conjunction of literals for input predicates, X represents variables in the clause head, and Y represents variables introduced by input predicates. The literals $b_1(X,Y), b_2(X,Y), \ldots, b_m(X,Y)$ are built on hidden predicates. Variables in Y are existentially quantified within the clause body. Only literals for input predicates can introduce new variables, and all literals for hidden predicates must use the entire set of variables X and Y. Additionally, each hidden predicate literal is unique within the program.

A hierarchical program can be represented as a tree, see Fig. III.2, where each node represents a clause or literal for hidden predicates.

Given the target predicate to learn, Neural HPLP learns a HPLP, consisting of logical clauses annotated with probabilities.

To learn an HPLP, the SLEAHP algorithm follows a structured procedure, generating bottom clauses from examples known as interpretations. These interpretations encapsulate all the relevant information from the domain, serving as the foundation for learning. Initially, a large HPLP is randomly generated using these bottom clauses. This program is then transformed into a graph structure, which facilitates the application of learning algorithms such as Gradient Descent [78] (SLEAHP_DEEP) and Expectation-Maximization (EM) [79] (SLEAHP_EM).

The learning process involves iteratively refining the probabilities associated with each clause in the program. The goal is to maximize the likelihood of the observed data, which is achieved by adjusting the clause weights. Clauses with very low probability values are pruned from the program to enhance the model's interpretability and performance. The overall workflow of the SLEAHP algorithm is shown in Algorithm 1:

Algorithm 1 SLEAHP Learning Algorithm for HPLP

Input: Set of interpretations $\mathscr{I}$, Learning method (*Gradient Descent* or *EM*)
Output: Learned HPLP with associated probabilities
Generate initial bottom clauses from interpretations $\mathscr{I}$
Randomly create an initial HPLP program $\mathscr{P}$
Convert program $\mathscr{P}$ into a graph structure G
Initialize clause probabilities π in $\mathscr{P}$
repeat
    if Learning method is *Gradient Descent* **then**
        Adjust clause weights using Gradient Descent to optimize the probability distribution
    else if Learning method is *EM* **then**
        Apply Expectation-Maximization to update clause probabilities
    end if
    Prune clauses with low probability values from $\mathscr{P}$
until Convergence criteria are met
Return the learned HPLP program $\mathscr{P}$

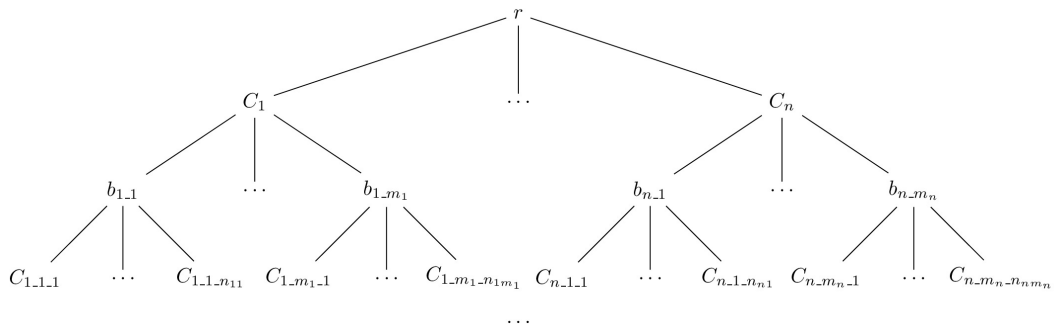


Fig. III.2 A HPLP program

This pseudo-code outlines the key steps in the SLEAHP learning process:

- Initialization: Start by generating bottom clauses from the given set of interpretations. These clauses provide a comprehensive representation of the domain knowledge.
- Program Generation: Construct an initial HPLP by randomly combining these bottom clauses. This initial program serves as the starting point for learning.
- Graph Conversion: Transform the HPLP into a graph structure to facilitate the application of learning algorithms.
- Learning Iteration: Depending on the chosen method (Gradient Descent or EM), adjust the probabilities of the clauses iteratively:
  - Gradient Descent: Optimize the probability distribution by adjusting clause weights, aiming to maximize the likelihood of the data.
  - EM Algorithm: Iteratively apply the expectation-maximization process to update the probabilities based on the observed data.
- Clause Pruning: After each iteration, prune clauses with very low probability values to simplify the model and enhance interpretability.
- Convergence: Repeat the learning and pruning process until convergence criteria are met (e.g., changes in probabilities fall below a threshold).

In conclusion, Neural HPLP integrates symbolic and neural components within an explainable framework, aiming to enhance the accuracy, reliability, and interpretability of predictive models. By leveraging both the strengths of symbolic methods and the processing power of neural networks, this system strives to offer a robust, interpretable decision-support tool applicable in various domains where transparency is crucial. The iterative learning process of HPLP, guided by SLEAHP, ensures that the model evolves to provide both accurate predictions and meaningful explanations.

11 Integration Between Constrained Optimization and Deep Networks

The integration of constraints within NNs can be approached from two perspectives that, although seemingly distinct, share several common characteristics: Constrained Neural Architecture Search (NAS) and Constrained Networks (CNs) the application of constraints during neural network training. Both paradigms aim to embed constraints and prior knowledge into the process of constructing and training NNs, with the goal of enhancing efficiency, interpretability, and adherence to specific application requirements. In both cases, knowledge integration and constrained optimization play a crucial role, though they serve different purposes.

In Constrained NAS, constraints are introduced during the architecture search phase to design NNs optimized for specific contexts. This approach is particularly relevant in areas like TinyML, where physical limitations such as computational resources are critical. On the other hand, CN focuses on improving network performance in specialized tasks, such as dealing with imbalanced data or leveraging domain knowledge to guide learning. By embedding such constraints, these methods seek to improve both the network's adaptability and its effectiveness in meeting the needs of specific application scenarios.

If we consider the broader impact of the two paradigms, they also offer potential advancements in areas such as explainability and generalization, where the integration of domain-specific constraints can help bridge the gap between neural and symbolic reasoning, opening up new opportunities for Neuro-Sylmolic AI.

We now show, we will illustrate a case of CN. Then, in Part IV, we will discuss real world examples of Constrained NAS and CN, demonstrating their practical utility across different domains.

11.1 Constrained Networks

The growing success of DL has prompted many researchers to explore methods for enhancing it through the integration of constraint-based domain knowledge. In certain scenarios, purely data-driven models may fall short, particularly when data is scarce or the learning tasks are highly complex. By leveraging domain knowledge,

NNs can benefit from problem-specific insights—such as the structure of the output, data generation processes, or expert knowledge—ultimately simplifying the training process and improving model performance. This approach avoids the need to start from scratch when addressing difficult learning challenges for NNs.

Below, we present two approaches for network training with the imposition of constraints, both implemented via penalty terms. The first method [80] introduces a penalty to the loss function when the network identifies elements outside a predefined region of interest. This approach encourages the network to focus on specific areas of the image, thereby leveraging domain knowledge to enhance performance in image-based tasks. The second approach [74] involves a LTN that, in contrast to the original formulation, incorporates logical loss as a regularization term. This penalty enables the network to utilize domain knowledge to improve both performance and robustness.

11.2 Enhancing NNs with Region-based Overlap Distance

In many applications, ensuring that deep learning models focus on the correct regions of an image is essential. Objects captured in industrial environments often include multiple components, not all of which are relevant to the task at hand. Consequently, it is important to guide NNs toward concentrating on the areas of interest, ensuring that irrelevant portions of the image do not influence the model's decision.

Our approach addresses this issue by identifying the polygon that outlines the image region most influential in the network's decision-making process. We then compute an overlap index with a predefined mask that marks the actual area of interest. This overlap index is integrated into the loss function as an Overlap Coefficient, where a smaller overlap results in a larger loss, and a larger overlap to a smaller loss. This mechanism encourages the network to focus specifically on the relevant areas of the image.

The Overlap Coefficient is incorporated into the Cross-Entropy (CE) loss function, which we refer to as Cross Entropy Overlap Distance. By leveraging a mask during training to calculate the overlap index, the network is guided to recognize and prioritize elements within the areas of interest. Importantly, these masks are only required during training for the overlap computation and are not needed during inference. As a result, inference time does not increase with respect to a standard

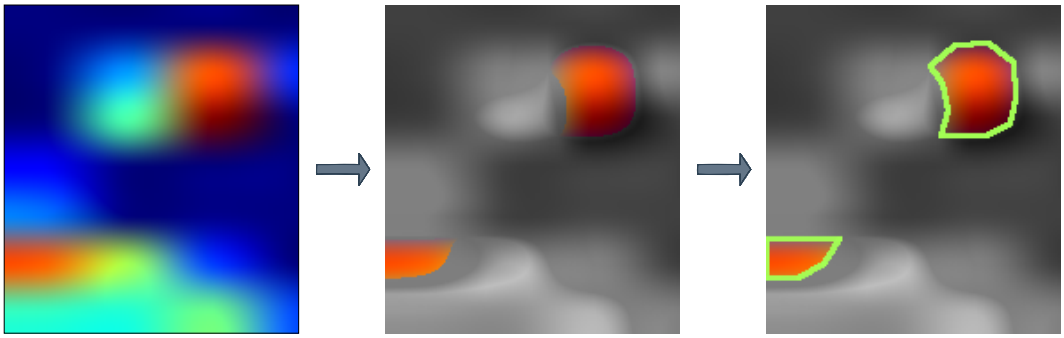


Fig. III.3 Example of a feature map extracted from a convolutional layer.

CNN. The masks are generated using Mask Region-Based Convolutional NNs (Mask R-CNN) [81].

To identify regions of the image that activate the network's decision-making process, we use Gradient-weighted Class Activation Mapping (Grad-CAM) [82]. The information provided by Grad-CAM is used to penalize the network when it detects elements outside the designated areas of interest, ensuring that the model's attention is consistently focused where it is most relevant.

The goal is to obtain a value that measures the overlap between two areas: the Region of Interest (ROI) and the one identified by the network. To achieve this, we utilize an Overlap Coefficient, also known as the Szymkiewicz-Simpson coefficient [83]. This index is calculated from the ROI, obtained through a mask present in the dataset, and the area of the image that the CNN considers most significant for the decision made. To identify this region, we use Grad-CAM.

A feature map is an intermediate output of a convolutional layer in a CNN. It contains information about the features detected in the input image after applying convolutional filters. For instance, in early layers, feature maps might represent edges, while in deeper layers, they represent more complex structures like shapes or objects. Figure III.3 shows an example of a feature map extracted from a convolutional layer.

On the left, a rough feature map is shown, where different colours, ranging from blue to red, represent varying levels of activation in response to the applied convolutional filters. Regions highlighted in red and orange indicate strong activations, suggesting that significant features have been detected in these areas, while the blue and green regions correspond to lower activation, indicating fewer or no relevant features. In the middle, less relevant activations have been suppressed, highlighting

only the strongest activations, which remain in orange and red, while the rest of the map appears in grayscale. Finally, on the right presents a more refined representation, where the most salient areas are outlined with green contours.

Grad-CAM is a localization technique derived from the Class Activation Map (CAM) algorithm [84], designed to generate visual explanations for any CNN without the need for modifications or re-training. To produce a class-discriminative heatmap, Grad-CAM computes the gradient of the output score for a specific class *cls* with respect to the feature map activations of the last convolutional layer, using the output out_{cls} obtained prior to the final softmax activation. The neuron importance weights, denoted as α_{cls}, are determined by performing a global average pooling of these gradients:

The indices i and j represent the spatial dimensions (height and width) of the feature map, and the gradient is summed across these dimensions to compute the importance of each feature map for the class of interest:

$$\alpha_{cls} = \frac{1}{P}\sum_{i}\sum_{j}\frac{\delta out_{cls}}{\delta A_{ij}} \tag{III.2}$$

where $\sum_i \sum_j$ represent the global-average pooling, P represents the number of pixels in the feature map, and A_{ij} represents the activation of the feature map at spatial location (i, j).

The final step involves a weighted combination of the activation maps, followed by a ReLU activation to produce the heatmap.

$$L_{cls}^{HeatMap} = ReLU(\sum_{k}\alpha_{cls}A_k) \tag{III.3}$$

For further details, refer to [82].

At the end of each forward pass during training, a heatmap is generated using Grad-CAM, highlighting the areas that most strongly influence the decision on the input image (the hottest pixels). These pixels are then extracted and compared with the mask; the greater the overlap, the lower the penalty. Figure III.4 illustrates the training phase with Cross Entropy Overlap Distance (CEOD).

The Overlap Coefficient is defined as:

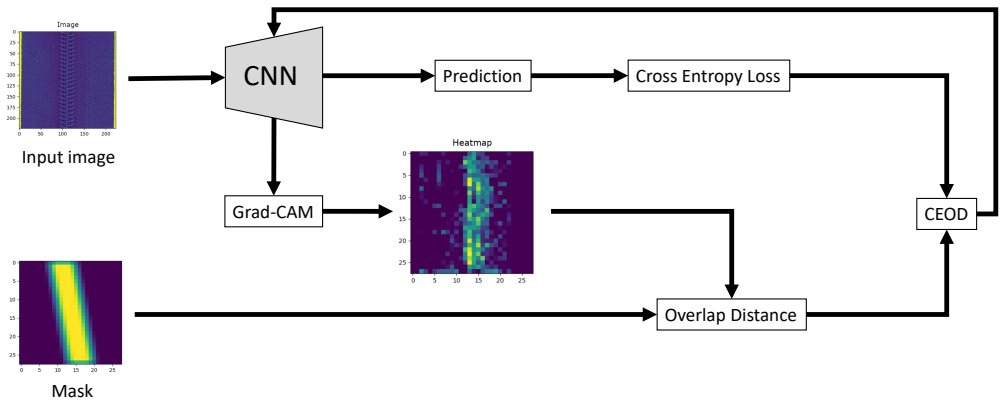


Fig. III.4 CE Overlap Distance training phase.

$$overlap_c(A_d, A_{gt}) = \frac{|A_d \cap A_{gt}|}{\min(|A_d|, |A_{gt}|)} \tag{III.4}$$

Here, the absolute value represents the number of pixels in the intersected region between A_d and A_{gt}. where A_d and A_{gt} represent the areas obtained via Grad-CAM and the segmentation mask (or ground truth mask), respectively. A_{gt} is derived either through manual segmentation or by using segmentation networks [81, 85, 86].

To integrate this term into the loss function, we define an overlap distance as follows:

$$\mathrm{OD}(A_d, A_{gt}) = -\ln(overlap_c(A_d, A_{gt})) = -\ln(\frac{|A_d \cap A_{gt}|}{\min(|A_d|, |A_{gt}|)}) \tag{III.5}$$

This formulation ensures that when the areas perfectly overlap, OD equals zero. This term is then incorporated into the loss function, specifically a CE loss [87], defined as:

$$ce = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(p(y_i)) \tag{III.6}$$

where N is the number of examples, each true label $y_i \in \{0, 1\}$ represents the ground truth of the i-th example, and $p(y_i)$ represents the predicted probability from the network for the i-th example.

This results in the CEOD, defined as:

$$\text{CEOD} = -\sum y_i \log(p(y_i)) + \omega(-\ln(\frac{|A_d \cap A_{gt}|}{\min(|A_d|, |A_{gt}|)})) \tag{III.7}$$

Here, ω is a new hyperparameter that allows for adjusting the penalty, which depends on the magnitude and difference between the two components of the loss.

By filtering the heatmap to extract the hottest pixels (Figure III.5, bottom left), we can isolate the object or area of interest, thereby defining the A_d term used in Equation (III.7).

We have introduced a novel method for enhancing CNN performance in image-based detection tasks by integrating domain knowledge through the use of overlap coefficients. By leveraging Grad-CAM to identify the regions of the image most relevant to the network's decision-making process, and comparing them with pre-defined ROIs, we ensure that the network is encouraged to focus on the correct areas during training. The resulting CEOD loss function effectively balances classification performance with spatial accuracy, providing a more robust and interpretable model.

11.3 Constraint-Based Neural Network Training

In many fields, the application of constraints during neural network training has proven effective in boosting both model performance and reliability. For example, in case of imbalanced datasets, constraints can be used to impose higher penalties for misclassifications of minority classes [88]. In anomaly detection systems, constraints can help by penalizing the misclassification of normal instances as anomalies. Similarly, in medical diagnostic systems, the incorporation of clinical constraints ensures that the network's predictions adhere to established medical evidence and guidelines. By enforcing these domain-specific constraints, we can significantly improve the neural network's accuracy and robustness. Furthermore, these constraints ensure that the model adheres to domain logic and principles, thereby enhancing its overall validity and trustworthiness in critical applications.

Logic Tensor Networks [50] offer a framework for integrating domain knowledge into neural network training by employing logical constraints. The core idea behind LTNs is to represent knowledge through logical predicates and use fuzzy logic to approximate logical operations in a differentiable manner. To guide the network,

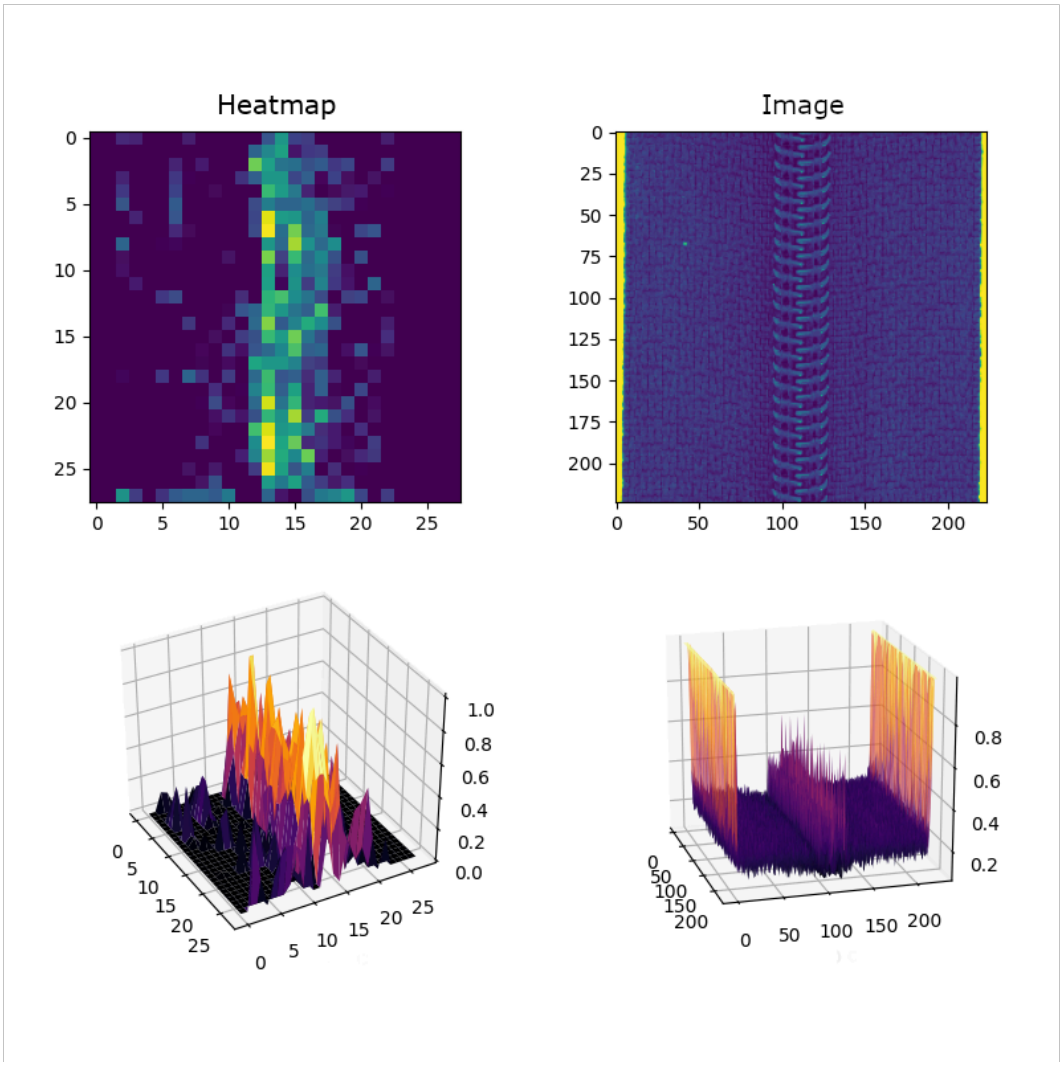


Fig. III.5 2D and 3D heatmaps (top left and bottom left) generated using Grad-CAM from an image (top right and bottom right).

we introduce a **SAT loss** (Satisfiability Loss) derived from the logical satisfaction of these constraints. The term SAT is used here to indicate the degree to which the neural network satisfies the logical constraints imposed on it. This loss acts as a regularization term, penalizing the network based on its violation of domain-specific rules. For instance, in a Network Intrusion Detection Systems (NIDS), we could impose a higher penalty for misclassifying benign instances because the primary objective in a NIDS is to accurately distinguish between benign and malicious traffic, rather than focusing on correctly identifying the specific type of attack. This prioritization ensures that the system can effectively detect any potential threat, minimizing the risk of overlooking a malicious event. Similarly, in the medical domain, where false negatives can have serious consequences, the network could be guided to prioritize minimizing them over false positives. By combining this ***SAT loss*** with a traditional training loss, such as cross-entropy, we can leverage domain knowledge to shape the network's learning process. During training, the ***SAT loss*** regularizes the model, while at inference, the network functions as a standard deep learning model, making it easily integrable with existing systems.

To optimize the DNN with respect to the additional focus provided by logical constraints, we combine the CE loss with the **SAT loss**. The **SAT loss**, which measures the degree of satisfaction of the logical constraints, is defined separately in Eq. II.30 (refer to Section 9.1 for its detailed formulation). The **CE loss**, which we define here, is expressed as follows:

$$\mathbf{L}_{CE} = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log\left(p\left(\widehat{y}_i\right)\right) \tag{III.8}$$

In this equation, $p(\widehat{y}_i)$ represents the predicted probability for the true class y_i, capturing the network's confidence in its predictions. Here, N is the number of samples, $\widehat{y}_i$ and y_i are the network's predicted output and the true label (as a one-hot vector) for the i-th sample, respectively. The logarithm is applied element-wise, and $\cdot$ denotes the dot product.

The final loss function, referred to as the *Hybrid-Loss*, is obtained by combining the cross-entropy loss and the **SAT loss**:

$$\textit{Hybrid-Loss} = \mathbf{L}_{CE} + \textbf{SAT Loss} \tag{III.9}$$

$$\textit{Hybrid-Loss} = -\frac{1}{N}\sum_{i=1}^{N} y_i \log\left(p\left(\widehat{y}_i\right)\right) + \left(1 - \text{SatAgg}(\phi)\right) \tag{III.10}$$

This combination allows the model to be guided not only by the standard classification objective but also by the logical constraints encoded in the **SAT loss**, improving the network's ability to adhere to domain-specific rules and logical consistency.

Figure III.6 illustrates how this hybrid loss function is used by the system.

During the testing and validation phases, we employ only the encapsulated DNN, $\text{MLP}_\theta(x)$, as a conventional model.

The ***SAT Loss*** acts as a regularization term that penalizes the network for violation of constraints. Our approach encapsulates an MLP or other DNN within an LTN structure, incorporating a regularization term that reflects the network's ability to satisfy a constraint.

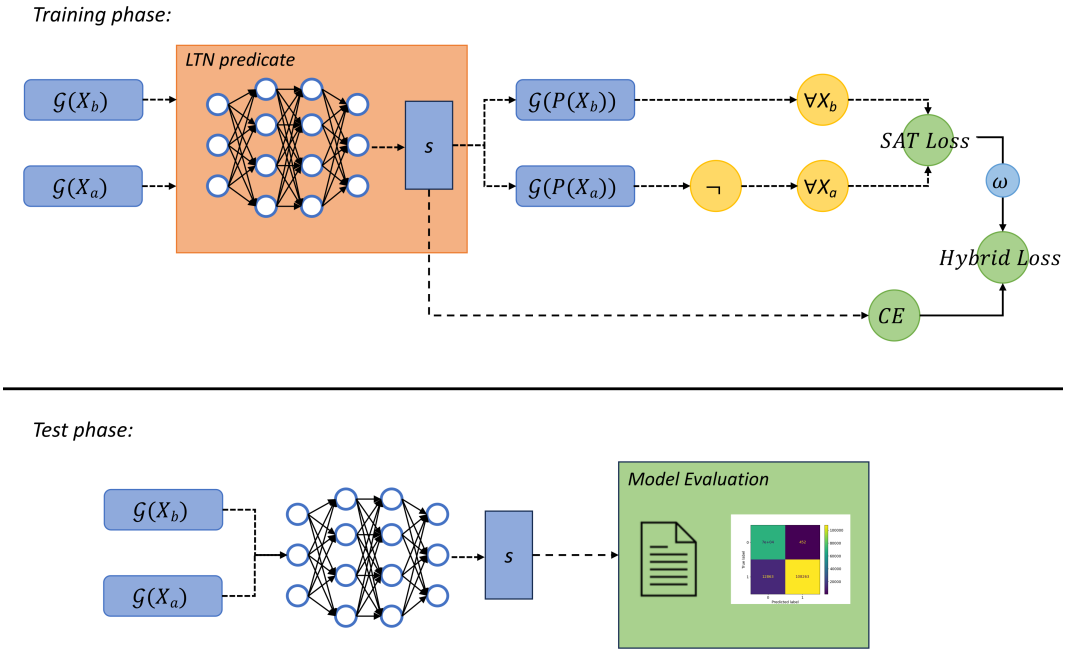


Fig. III.6 *Training phase (top):* the LTN predicate $\mathcal{G}(P)$ takes $\mathcal{G}(X_b)$ and $\mathcal{G}(X_a)$ as inputs and produces $\mathcal{G}(P(X_b))$ and $\mathcal{G}(P(X_a))$ as outputs. *Test phase (bottom):* the network takes $\mathcal{G}(X_b)$ and $\mathcal{G}(X_a)$ as inputs and produces score s as output. s is used to evaluate model's prediction.

12 Open-Set Recognition with Neuro-Symbolic AI

ML systems are typically developed under the closed-world assumption, which assumes that all test classes are represented in the training data [89–91]. However, there has been a growing focus on enhancing these systems' ability to recognize and manage unknown classes. This effort has been especially evident in the areas of anomaly detection, Out-Of-Distribution (OOD) detection, and Open Set Recognition (OSR). While anomaly detection was traditionally the main focus, recent trends have shifted attention towards OOD detection and OSR.

The key differences between OOD detection and OSR are twofold. First, OOD detection deals with a wider semantic gap between the data considered inside and outside the distribution. In contrast, OSR involves categorizing data subsets as either within or outside the distribution within the same dataset. Second, OOD detection primarily focuses on distinguishing between external and internal samples, whereas OSR also evaluates the classification performance on known classes within a closed-world context [92]. As discussed in [93], there is a distinction between semantic shift and covariate shift. Semantic shift refers to OOD samples that belong to entirely different classes, while covariate shift pertains to samples that come from different domains.

To address OSR problem, we propose a system that combines Tree Extreme Gradient Boosting with DEC.

We are given a dataset D comprising pairs (x, y), where $x \in X$ and $y \in C$. Here, X represents the input space and C represents the label set. D is divided into a training set, D_{tr}, and a test set, D_{test}. Additionally, we define two subsets of C: C_k, containing the known classes, and C_u, containing the unknown classes. The objective is to construct a function $f : X \rightarrow \{known, novelty\}$ that assigns each input x to one of two categories: *known* if $y \in C_k$, and *novelty* if $y \in C_u$.

The ability to accurately detect these novel classes is crucial for ensuring computer network security, particularly as new and sophisticated cyber threats continue to emerge.

Tree EXtreme Gradient Boosting with DEC (**TEX-DEC**) [75] exploits DEC and XGBoost, as shown in Figure III.7.

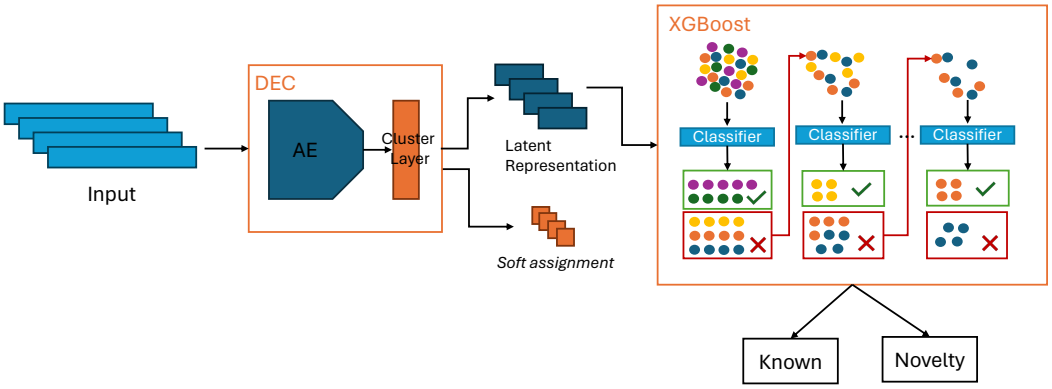


Fig. III.7 DEC first extracts the latent representation and performs soft assignment. XGboost then uses the latent representation to detect Known and Novel sample.

We divided the dataset D into three distinct subsets: *Known*, *Novelty 1*, and *Novelty 2*, as depicted in Figure III.8. The *Novelty 1* and *Novelty 2* subsets correspond to examples from novel classes, with *Novelty 1* being utilized for training the XGBoost model and *Novelty 2* reserved exclusively for the testing phase. The *Known* subset is further segmented into three parts: *Known Tr1*, *Known Tr2* and *Known Test*. The first for training the DEC model, the second for training XGBoost along with *Novelty 1*, and the third, together with *Novelty 2*, for testing the TEX-DEC system.

We adopt a one-vs-all strategy by using one class present in *Novelty 2* at a time during testing, which allows for the training of a separate XGBoost model for each class. For example, if classes A, B, and C are treated as novelties, three distinct XGBoost models are trained, each focusing on a different novelty class. If the test class *Novelty 2* is A, the other classes (B and C) are treated as *Novelty 1*.

We employ contrastive learning to enhance the separation of clusters generated by DEC. By integrating the strengths of contrastive learning with traditional clustering methods, our approach aims to improve data representation and clustering effectiveness. Our primary contribution is the incorporation of a contrastive loss and a classification loss alongside the KL divergence typically used in DEC. This enhancement helps the second stage of the architecture better differentiate between known and novel samples. Below, we detail the loss components used during training:

- *KL Divergence:* As employed in DEC, see Section *background*.

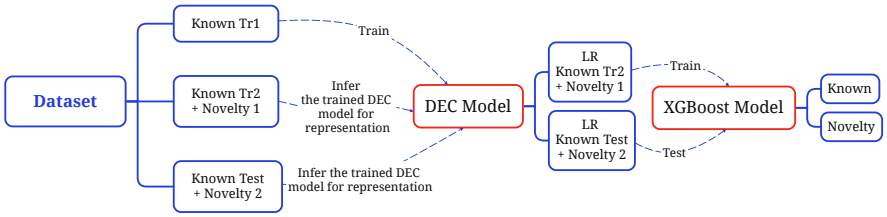


Fig. III.8 The dataset is divided into six subsets: *Known Tr1, Known Tr2, Known Test, Novelty 1* and *Novelty 2*. The *Known Tr1* subset is used to train DEC. A latent representation (LR) is obtained from each of subsets (note: ML DEC parameters are not changed at this stage). These LRs are then used to train (*Known Tr2* and *Novelty 1*) and test (*Known Test* and *Novelty 2*) an XGBoost classifier for sample classification.

- *Contrastive Loss:* We calculate the average centroid distance, $C_{distance}$, using the Euclidean distance between each pair of centroids:

$$C_{distance} = \frac{\sum_{i,j} \|\mu_i - \mu_j\|_2}{k(k-1)} \tag{III.11}$$

  where k is the number of clusters and μ_i is the centroid of cluster i. This loss measures the mean Euclidean distance between cluster centroids, rather than focusing on individual sample pairs as in traditional contrastive losses. To our knowledge, this specific formulation has not been introduced in the literature. The contrastive loss is then defined as:

$$\mathscr{L}_{contrastive} = \frac{1}{C_{distance}} = \frac{k(k-1)}{\sum_{i,j} \|\mu_i - \mu_j\|_2} \tag{III.12}$$

  This loss function is designed to maximize the distance between centroids.

- *Classification Loss:* To ensure that clusters accurately represent the true classes of known samples, we employ the Cross Entropy loss (L_{CE}), defined as follows:

$$L_{CE} = -\frac{1}{N} \sum_{i=1}^{i=N} y_i \cdot \log\left(p(\widehat{y}_i)\right) \tag{III.13}$$

  where $\widehat{y}_i$ and y_i represent the predicted and actual labels, respectively.

The overall loss function is then formulated as:

$$\mathscr{L} = \alpha \cdot \mathscr{L}_{kld} + \beta \cdot \mathscr{L}_{contrastive} + \omega \cdot L_{\text{CE}} \tag{III.14}$$

After DEC compresses the input into a more compact latent representation, XGBoost is employed to differentiate between known and novel instances. This approach uses both known examples and a limited selection of novel samples for training XGBoost. Our dataset is divided into three distinct subsets: the first is used solely for training DEC with known instances, the second for training XGBoost with a combination of known and novel examples, and the third for testing, which includes a balanced mix of both known and novel instances. Importantly, the novelty classes used during XGBoost training are distinct from those evaluated in the final test set, ensuring a reliable and consistent assessment of the novelty detection performance.

13 Benchmarking activity

The lack of standardized benchmarks in NeSy AI represents a significant challenge, preventing the comparison and objective evaluation of the various solutions developed in this field. Unlike more mature areas of AI such as machine learning and deep learning, which benefit from well-established datasets and performance metrics, NeSy AI still lacks shared benchmarks that can uniformly test the integration of symbolic reasoning and neural learning. This gap not only limits the ability to measure progress but also hinders the development of more robust, efficient, and scalable methods. Consequently, identifying the most promising techniques for specific applications is currently difficult. Establishing a set of benchmarks is therefore essential to foster both research and the practical adoption of Neuro-Symbolic systems.

To address this issue, we developed two datasets: the first for the classification of tic-tac-toe board states and the latter for determining the optimal next move in tic-tac-toe. These datasets are part of a wider activity for developing a foundational framework for comparing NeSy systems [94], that includes datasets such as CLEVR-Hans [95], Kandinsky Patterns [96], and CLE4EVR [97]. In the following sections, we will introduce the tic-tac-toe datasets and demonstrate their usage through an example implementation with DeepProbLog. This approach seeks to not only encourage standardization but also provide a practical tool for advancing research in this growing area.

13.1 Tic-Tac-Toe Dataset Generation

In this section, we describe the generation of two distinct Tic-Tac-Toe datasets: one for predicting the winner of a complete game and the other for determining the optimal next move in an ongoing game.

The first dataset consists of all possible valid Tic-Tac-Toe board configurations, where either player X wins, player O wins, or the game ends in a draw. We utilized Python to systematically generate these board configurations, exploring all potential end-game states. The following Python function was employed to create these configurations:

```
def generate_random_board():
    # X starts, alternating X and O
    symbols = ['X', 'O'] * 4 + ['X']
    # Empty 3x3 board
    board = [['b'] * 3 for _ in range(3)]
    # Shuffle to explore different move sequences
    random.shuffle(symbols)

    for i in range(9):
        row, col = divmod(i, 3)
        board[row][col] = symbols.pop()
        if winner := check_winner(board):
            # Stop if a winner is found
            return board, winner

    # Return 'b' for a draw
    return board, 'b' if valid_game(board) else None

def valid_game(board):
    count_x = sum(row.count('X') for row in board)
    count_o = sum(row.count('O') for row in board)
    # Ensure a valid state
    return count_x == count_o or count_x == count_o + 1
```

This approach ensures that each board state is complete, valid, and reflective of all potential end-game scenarios. These configurations were exported to CSV format for subsequent use in model training and evaluation.

For the second dataset, we focused on predicting the optimal next move for a given Tic-Tac-Toe board state. This dataset is more complex, requiring the model to anticipate the best strategic move based on the current board configuration. We used the following Prolog code to generate random board configurations and determine the next move:

```
% Prolog code snippet for generating the next move dataset
generate_random_board(Board):-
    length(Board,9),
    place_elements(Board),
    valid(Board), !.
generate_random_board(Board):-
    generate_random_board(Board).



% Determine the next move
next(Column,Row) :- player(Player), line(Column,Row,Player).
next(Column,Row) :- player(Player), opponent(Player,Opponent),
                line(Column,Row,Opponent).


pair_(Board,L):-
    generate_random_board(Board),
    assert_board(Board),
    pair(Board,L),
    retractall(cell(_,_,_)), !.

pair_(_,_):-
    retract(player(_)),
    retractall(cell(_,_,_)).
```

```
pair(Board,[Column,Row]):-
    next_player(Board,Player),
    opponent(Player,Opponent),
    assertz(player(Player)),
    \+win(Player),\+win(Opponent),
    \+full_board,
    next(Column,Row),
    retract(player(Player)).

pair(Board,[]):-
    writeln(Board),
    win(Player),
    display_game(Board).
```

This Prolog code defines the logic for generating random Tic-Tac-Toe board states and predicting the optimal next move for the current player. The board configurations are generated randomly, ensuring each board is valid before proceeding. The code then uses logical rules to evaluate possible moves, considering both winning strategies and blocking the opponent's potential wins. By simulating various board states and selecting moves accordingly, this approach generates a dataset that encapsulates strategic decision-making in Tic-Tac-Toe. This dataset can then be used to train models that can learn to predict optimal moves in similar board scenarios.

In addition to generating board states, we utilized images to represent the board cells. Two sets of images were used: digits from the MNIST dataset and a custom set of hand-drawn images. In this context, the digit '1' represents player X, '2' represents player O, and '3' indicates an empty cell. These images were processed to fit into the Tic-Tac-Toe grid cells, providing a visual representation of the game state. Figure III.9 shows examples of the images used.

The combination of board configurations and visual data creates a rich dataset that challenges models to learn both visual perception (recognizing the board symbols) and reasoning (determining the game outcome or next move).

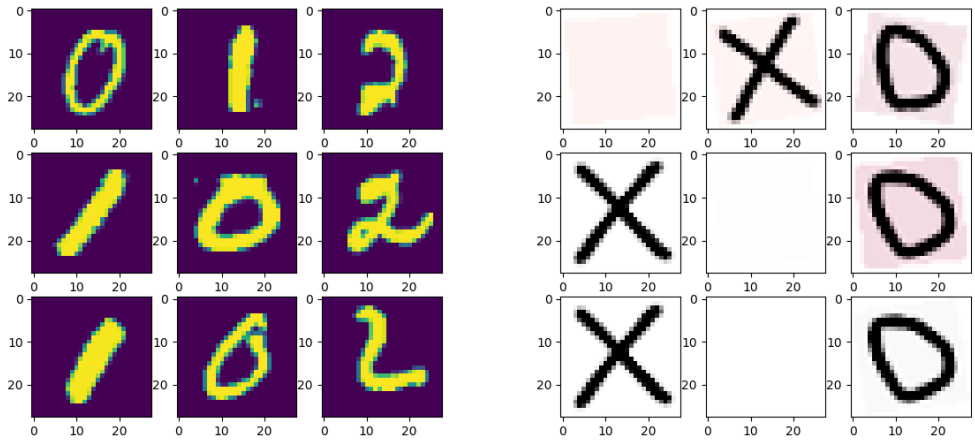


Fig. III.9 The MNIST dataset board (left) and the custom hand-drawn board (right).

13.2 DeepProbLog Implementation

For the implementation with DeepProbLog, we first define a neural predicate to classify the board's cells based on the input images. The neural network, trained on the MNIST and custom hand-drawn images, outputs probabilities for each cell being 'X', 'O', or empty. The following DeepProbLog code snippet outlines how we define the neural predicate and integrate it with the logical rules:

```
% DeepProbLog code snippet for defining the neural predicate
nn(cell/2, [x, o, empty], NeuralNet).

% Logical rules for evaluating the board state
win(X) :- row(X); column(X); diagonal(X).

% Define the rows, columns, and diagonals
row(X) :- cell(1,X), cell(2,X), cell(3,X).
row(X) :- cell(4,X), cell(5,X), cell(6,X).
row(X) :- cell(7,X), cell(8,X), cell(9,X).
column(X) :- cell(1,X), cell(4,X), cell(7,X).
column(X) :- cell(2,X), cell(5,X), cell(8,X).
column(X) :- cell(3,X), cell(6,X), cell(9,X).
diagonal(X) :- cell(1,X), cell(5,X), cell(9,X).
```

```
diagonal(X) :- cell(3,X), cell(5,X), cell(7,X).
```

In the code above, the neural predicate ‘cell/2‘ employs the trained neural network to classify each cell on the board. This classification information is then used by the logical rules to evaluate the board's state, either to determine the winner of the game or to identify the optimal move in an ongoing game. DeepProbLog thus leverages the neural network's perceptual capabilities to interpret visual input, while employing symbolic reasoning to make higher-level decisions.

This application of DeepProbLog to the Tic-Tac-Toe game exemplifies the effective integration of DL with symbolic reasoning. The system successfully processes visual inputs, interprets game states, and makes informed decisions about the game's outcome. The ability to handle complete board suggests potential for broader applications in more complex games scenarios where visual and logical information must be seamlessly combined. Moreover, this work contributes to the growing field of NeSy, which seeks to combine NNs with symbolic reasoning. Despite its potential, the NeSy field currently suffers from a lack of consistent benchmarks that can rigorously evaluate new approaches. By developing a structured and challenging benchmark involving Tic Tac Toe, this project not only showcases the practical capabilities of DeepProbLog but also provides a valuable resource for the NeSy community. This benchmark can serve as a foundational tool for future research, helping to establish a more robust and comprehensive literature in this rapidly evolving domain.

Chapter IV

Application of Neuro-Symbolic Integration

In recent years, NeSy AI has gained increasing attention for its ability to combine the power of NNs with the transparency and interpretability of symbolic systems. This approach is particularly valuable in fields where both accuracy and explainability are crucial. In this part, we will illustrate how several NeSy AI techniques can enhance both the effectiveness and the interpretability of the proposed solutions.

In the medical field, we will explore an application of Neural HPLP exploiting image analysys of CT scans and tabular clinical data for Covid-19 diagnosis. The combination of these results through the use of a HPLP enables the development of a highly accurate and explainable model, which is critical in contexts such as medical diagnosis.

In the industrial domain, we will present a system that uses information about regions of interest in images to increase the focus of the NN on relevant portions of the image. This is achieved through the introduction of a new loss function, called CEOD, which guides the model to focus more on the significant areas of the image, thereby improving the system's precision.

Regarding Neural Architecture Search, we will discuss various approaches that implement NAS under physical constraints, such as memory and energy consumption. Additionally, we will introduce the use of a symbolic NAS, which effectively manages these limitations by balancing hardware resources with model performance.

Finally, in the field of Network Intrusion Detection Systems, we will address the problem of Open Set Recognition. Two approaches will be illustrated to improve the robustness of classification models. The first, *Hybrid LTN*, uses a Logic Tensor Network framework with a hybrid loss function that combines Cross Entropy with logical constraints. The second approach, *TEX-DEC*, leverages deep clustering and XGBoost to detect unseen examples during training, enhancing the system's ability to handle new threats.

Together, these applications showcase the broad applicability of NeSy AI across diverse fields, emphasizing its potential to address intricate challenges with solutions that are both accurate and transparent.

In Section 14, we explore three significant works related to Covid-19 diagnosis and prediction using NeSy AI approaches: *Machine Learning Techniques for Extracting Relevant Features from Clinical Data for COVID-19 Mortality Prediction* [71], *Neural-Symbolic System for Predicting COVID-19 Positivity* [72], and *Neural-Symbolic Ensemble Learning for Early-Stage Prediction of Critical State of COVID-19 Patients* [56]. These studies demonstrate the effectiveness of combining neural and symbolic methods to improve both model accuracy and interpretability in the medical field.

A portion of this research has been published in our survey *Integration between Constrained Optimization and Deep Networks: A Survey* [73], discussed in Section 15. This survey provides a thorough overview of the integration between deep networks and constrained optimization, laying the groundwork for the techniques examined in this chapter.

Additionally, in Section 16, we revisit concepts explored in our work *Exploiting CNN's Visual Explanations to Drive Anomaly Detection*. This work highlights the importance of explainability in CNNs for anomaly detection.

Finally, Section 17 delves into our research on NIDS, featuring works like *A Neuro-Symbolic AI Network Intrusion Detection System* [74] and *Neuro-symbolic Integration for Open Set Recognition in Network Intrusion Detection* [75]. These contributions underscore the value of NeSy AI approaches in enhancing cybersecurity by improving the detection of previously unknown threats, including zero-day attacks. This further demonstrates the practical significance of NeSy AI in real-world applications such as cybersecurity.

14 Neural HPLP in the Medical Environment

In this section, we describe the application of Neural HPLP in a medical context. Specifically, the system predicts whether a Covid-19 patient admitted to the hospital will progress to a critical condition using both clinical data and CT scan images. Additionally, an experiment will be presented that aims to identify Covid-19 infections through the analysis of CT scans and clinical data.

The global emergency has underscored the necessity for early identification of complications and the assessment of risk status in patients infected with Covid-19. Early diagnosis is critical for Covid-19 positive patients to ensure timely and appropriate interventions [98]. Given the vast amount of data and extensive research in healthcare, particularly in Medicine 4.0, AI technologies are increasingly being applied in the medical field [99–101]. However, predicting complications from clinical data presents challenges, including the difficulty of identifying patterns in structured clinical data, handling missing values, and dealing with lack of annotations.

Currently, while DL algorithms demonstrate high analytical accuracy, they often lack interpretability, which undermines their acceptance. Therefore, it is essential to develop systems that can provide clear explanations for their decisions [102, 103], especially in sensitive areas such as medicine. The objective of this work is to employ DL techniques to analyze lung images (CT scans) and clinical data through explainable ML methods to predict disease severity while simultaneously offering explanations for these predictions.

Neural HPLP leverages 3D-CNN to analyze CT lung scans of Covid-19 patients, DTs to predict mortality risk based on clinical data, and a neural system that integrates these components using HPLP [30, 76]. The goal is to extract relevant patterns from the diverse data collected from patients to provide a more comprehensive analysis. This integrated approach aims to combine the predictive power of DL with the interpretability of symbolic AI methods to enhance both accuracy and trustworthiness in medical decision-making.

14.1 Methodology

The objective of our study is to predict the health status of patients infected with Covid-19 upon their arrival at the hospital. To accomplish this, we propose the

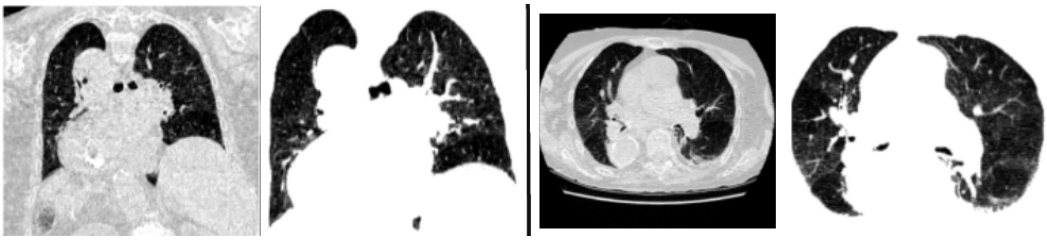


Fig. IV.1 Images (a) and (c) show an original DICOM slice. Images (b) and (d) present the result of pre-processing.

application of Neural HPLP, described in Section 10, which integrates both symbolic and neural components. The neuro-symbolic block is based on HPLPs, an ML model designed to create a scalable, reliable, and interpretable AI system.

The neural, symbolic, and neuro-symbolic components are detailed as follows. The first is a 3D convolutional network that processes CT scans, the second is a DT that analyzes clinical data, and finally, the neuro-symbolic block integrates the outputs of these components using HPLP.

3D-CNN for CT scans DL has the ability to process large amounts of complex data and detect patterns. This makes it extremely suitable for analyzing large images. In particular, CNNs are able to identify topographical patterns within images. In our case, we used 3D-CNNs to analyze lung CT images. These are data in DICOM format. They can be seen as image sets that dissect the patient and form a 3D model. Before the training, the images were pre-processed by applying a threshold to isolate the lung areas based on Hounsfield Unit (HU) values, as shown in Fig. IV.1. This was achieved by using a range of HU values typically associated with lung tissue, between -700 and -600, to create a binary mask that isolates the lungs from other structures. This is a simple yet effective method to enhance the quality of the input data.

The segmentation process uses the Hounsfield Units (HU) scale, a quantitative measure for describing radiodensity in medical CT images. Using this scale, we identified the lung regions, which have HU values between -700 and -600, and isolated the lungs.

It is important to note that using a fixed threshold for lung segmentation can be influenced by different scanners and acquisition conditions [104]. This issue can be

addressed using unsupervised Fuzzy c-means clustering techniques [105], such as spatial FCM [106].

When working with 3D scans, it is essential to understand the volumetric characteristics of CT scans in addition to their 2D features. The volumetric aspect is fundamental to the analysis of scans, as it plays a crucial role in accurately interpreting and assessing the data.

DT for Clinical Data The clinical data consist of structured tabular data with 59 clinical attributes per patient, which include numerical and categorical variables such as age, sex, laboratory test results, and comorbidities. Before applying the DT, the dataset was pre-processed to handle missing values and ensure consistency. We used DT and RF for clinical data. Prior to training, the dataset needed to be balanced; hence, Synthetic Minority Oversampling Technique (SMOTE) [107] was applied. SMOTE works by selecting a minority class instance and identifying its n-nearest neighbors within the same class. In this study, SMOTE was used to oversample the class of deceased patients, which comprised 25% of the dataset.

First, a RF was employed to identify the most relevant clinical features influencing a patient's likelihood of death during hospitalization. This was done by measuring the importance of each feature based on how much they reduced impurity in the trees, also known as the Mean Decrease in Impurity (MDI) method [**?**]. This produced a new version of the dataset with the same number of patients but with only 10 clinical parameters. This refined dataset was then used to train a DT.

A DT was selected due to its ability to provide transparent decision paths. The entire decision path can be extracted as a rule, which offers an initial explanation for the model's predictions.

Neural HPLP We use Neural HPLP, as previously discussed, which represented a target predicate based on a set of examples referred to as interpretations. Each interpretation pertains to a specific patient and includes outputs from both the DT and the 3D-CNN. The target predicate in this context is whether a Covid-19 patient will progress to a critical condition.

The learned program is capable of predicting whether a patient arriving at the hospital will deteriorate into a critical state and can provide explanations for these predictions.

Neural HPLP operates on a set of interpretations, each representing comprehensive information about an individual Covid-19 patient. For each patient, we generated an interpretation that includes an atom indicating the patient's critical state. A patient is considered critical if the DT predicts imminent death or if the 3D-CNN identifies a severe lung condition. Each interpretation also contains atoms reflecting the outputs from the DT (e.g., *dead* or *alive*) and from the 3D-CNN (lung condition: *serious, minor*, or *healthy*).

To enhance the interpretations, we incorporated the decision path from the DT, which outlines the features and criteria used in its decision-making process. This additional information provides greater transparency and understanding of how predictions are made, thereby improving both the interpretability and reliability of the system.

14.2 Dataset

The dataset is composed of two subsets: clinical data and lung CT scans. The clinical dataset was provided by a hospital in Ferrara, Italy, and includes records from 502 Covid-19 positive patients collected during the spring of 2020, of whom 126 died during hospitalization. Thus, deceased patients account for approximately 25% of the entire dataset. Each patient record contains 59 clinical attributes. Additionally, 96 of these patients also had corresponding CT scans, which were retained as the test set. Among these 96 patients, 30 passed away during the hospitalization period. Table IV.1 lists the clinical attributes of each patient along with their respective acronyms.

The CT scan dataset is described in MosMedData [108]. It includes human lung CT scans with and without Covid-19-related findings. The scans were collected in 2020 and provided by the municipal hospital in Moscow, Russia. The dataset categorizes CT scans according to the severity of lung tissue abnormalities related to Covid-19, with five classes: no injury, mild, moderate, severe, and critical injuries. The distribution of the dataset is as follows: CT-0, CT-1, CT-2, CT-3, and CT-4 contain 254, 684, 125, 45, and 2 patients, respectively. It is evident that the dataset

Table IV.1 Clinical data for the main experiment

Clinical attribute	Acronym	Clinical attribute	Acronym
Age	-	Gender	-
Organization Cost Centre	CdcoUO	Intensification of care	-
Pneumology department	-	Glomerular Filtration Rate	GFR
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	HRTCpergroundglass	White blood cells	WBC
Lymphocytes	-	C-reactive Protein	CRP
Procalcitonin	PCT	Creatinine	-
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	-		

is imbalanced, with a greater concentration in the CT-1 (mild injuries) class. Due to the limited number of cases in the last three classes, they were merged into a single class, resulting in the following distribution: CT-0 with 254 patients (22.8%), CT-1 with 684 patients (61.6%), and CT-234 with 172 patients (15.6%). Figure IV.2 shows an example image for each class. These classes correspond to three levels of lung injury severity: no injury (CT-0), minor (CT-1), and serious (CT-234).

For the test set, we used the CT scans from the aforementioned 96 patients from the Ferrara hospital dataset. All images in this dataset are in DICOM format, meaning that each CT scan can be viewed as a sequence of consecutive images forming a 3D representation. Consequently, we employed a convolutional NN with 3D filters for analysis.

The dataset used for the Covid-19 Positivity Prediction experiment was provided by the Huazhong University of Science and Technology [109], Wuhan, China, and consists of 1,521 patients, with 1,126 from the Union Hospital HUST-UH and 395 from the Liyuan Hospital HUST-LH. The dataset comprises 894 Covid-19 positive patients and 627 non-Covid-19 patients. Each patient is described by 120

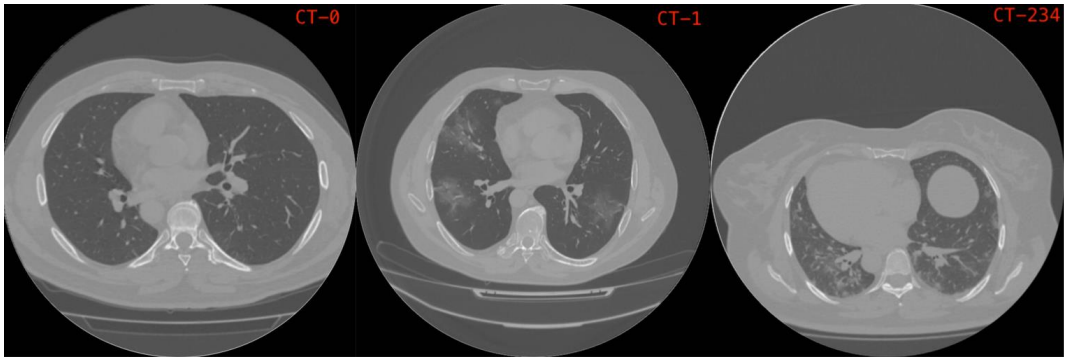


Fig. IV.2 Here is an example of a DICOM voxel slice illustrating the three dataset classes: CT-0 on the left, CT-1 in the middle, and CT-234 on the right.

clinical attributes, and 1,342 subjects had both CT scans and clinical data. For the experiments, patients with normal CT scans (class Normal) and those with lung lesions (class Pneumonia) were considered. Specifically, 1,006 patients have pneumonia and 336 patients have healthy lungs. All scans in the dataset are in DICOM format. In the experiment, individual slices from each scan were extracted and processed, focusing only on the sections containing the lungs. Table IV.2 lists all the clinical attributes.

A total of 47,260 2D images were obtained and used to train a CNN. The dataset, grouped by patient, was divided into training (75%), validation (10%), and testing (15%) sets. The test set, therefore, includes 203 patients.

14.3 Related Work

Numerous studies indicate that early diagnosis of Covid-19 significantly reduces its mortality rate [98]. Predicting whether a Covid-19 patient will suffer a severe condition requiring intensive care or more aggressive treatment is crucial for managing the pandemic and saving lives. During peak crisis times, with many patients in serious condition, effectively managing the limited intensive care resources in hospitals is vital. Early identification of patients likely to become critically ill allows healthcare providers to gain advanced insight and administer specialized treatment to those at high risk. Additionally, it helps to predict the future demand for intensive care, facilitating optimal allocation of resources among various critical illnesses. By providing rule-based explanations for its predictions, such as identifying clinical

Table IV.2 Clinical data for the Covid-19 Positivity Prediction

Clinical attribute	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
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		

attributes relevant to disease severity, Neural HPLP not only aids in targeted patient care but also serves as a more interpretable and acceptable predictive model.

Various ML and DL algorithms have been applied to different types of medical data, such as structured clinical data, CT scans, radiographs, and electrocardiograms, to create predictive models for Covid-19. For example, several studies have utilized ML models for diagnosing Covid-19 using sociodemographic and medical data [110]. Other research has employed DL, RF, and DT to optimize resource allocation and triage procedures during the pandemic [111], in contrast to studies that used ML algorithms to analyze clinical data with the goal of identifying factors influencing mortality during hospitalization [112]. Additionally, Yan Li et al. [113] applied XGBoost and DTs to establish decision rules for identifying patients at high risk of mortality.

In terms of CT scans and chest X-rays, Alsharman et al. [114] employed CNNs to detect Covid-19 in early-stage CT scans. Albahli [115] demonstrated the high performance of DNNs in identifying Covid-19 patients, achieving 89% accuracy on synthetic data generated by a GAN-based model. Parnian Afshar et al. [116] proposed a framework using Capsule Networks (COVID-CAPS) [117], which handled small datasets, achieving 95.7% accuracy, 90% sensitivity, 95.8% specificity, and an AUC of 0.97. Purkayastha et al. [118] introduced an approach similar to Neural HPLP that combined clinical and imaging data to predict Covid-19 severity. This study developed models for predicting both the severity and progression to critical illness using radiomics features and clinical variables but lacked explainable predictions.

Additional research addressing Covid-19 includes the work by Cheng et al. [119], that used ML to develop a risk prioritization tool predicting ICU transfers within 24 hours, and Montomoli et al. [120], who used the XGB algorithm to forecast changes in patients' Sequential Organ Failure Assessment (SOFA) scores five days after ICU admission.

The primary innovation of Neural HPLP lies in its ability to provide explainable predictions through HPLP. Unlike other systems that process different types of data, where NNs often function as black boxes, Neural HPLP offers a transparent interpretation of results, distinguishing it from other approaches.

14.4 Experiments

14.4.1 Experiments with Clinical Data

We first analyzed clinical data. The ML models used in these experiments were DTs [13] and RFs [15]. The experiment was conducted in two stages. In the first stage, RFs were employed to identify the most influential clinical features that determine patient mortality during hospitalization. The feature importance analysis performed with RF enabled the identification of key clinical attributes that are most useful for classification.

To extract the most relevant features using the RF, we utilized the Mean Decrease in Impurity (MDI) technique, also known as Gini importance. This technique measures the importance of each feature based on how much it decreases the weighted impurity in a tree. The importance scores for each feature are computed by averaging the decrease in impurity brought by that feature across all trees in the forest [15]. Features with the highest mean decrease in impurity are considered the most significant.

Using the feature extraction results from the RF, a new version of the dataset was created, containing the same number of patients but reduced to only 10 clinical parameters. This refined dataset was then used to train a DT, which achieved an accuracy comparable to that of the RF, with the added benefit of enabling the extraction of a decision rule (decision path) for the classification. A DT was chosen because it allows for the extraction of the entire decision path in the form of a rule, as illustrated by Rule (IV.1), providing a clear explanation of the DT's prediction. The generated DT is depicted in Figure IV.3.

$$\textbf{if } condition_1 \wedge \ldots \wedge condition_n \textbf{ then } outcome \tag{IV.1}$$

The most relevant clinical attributes identified 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.

These clinical attributes were used to train a DT, which achieved an accuracy of approximately 70% on the test set (i.e., the 96 patients described at the beginning of this section).

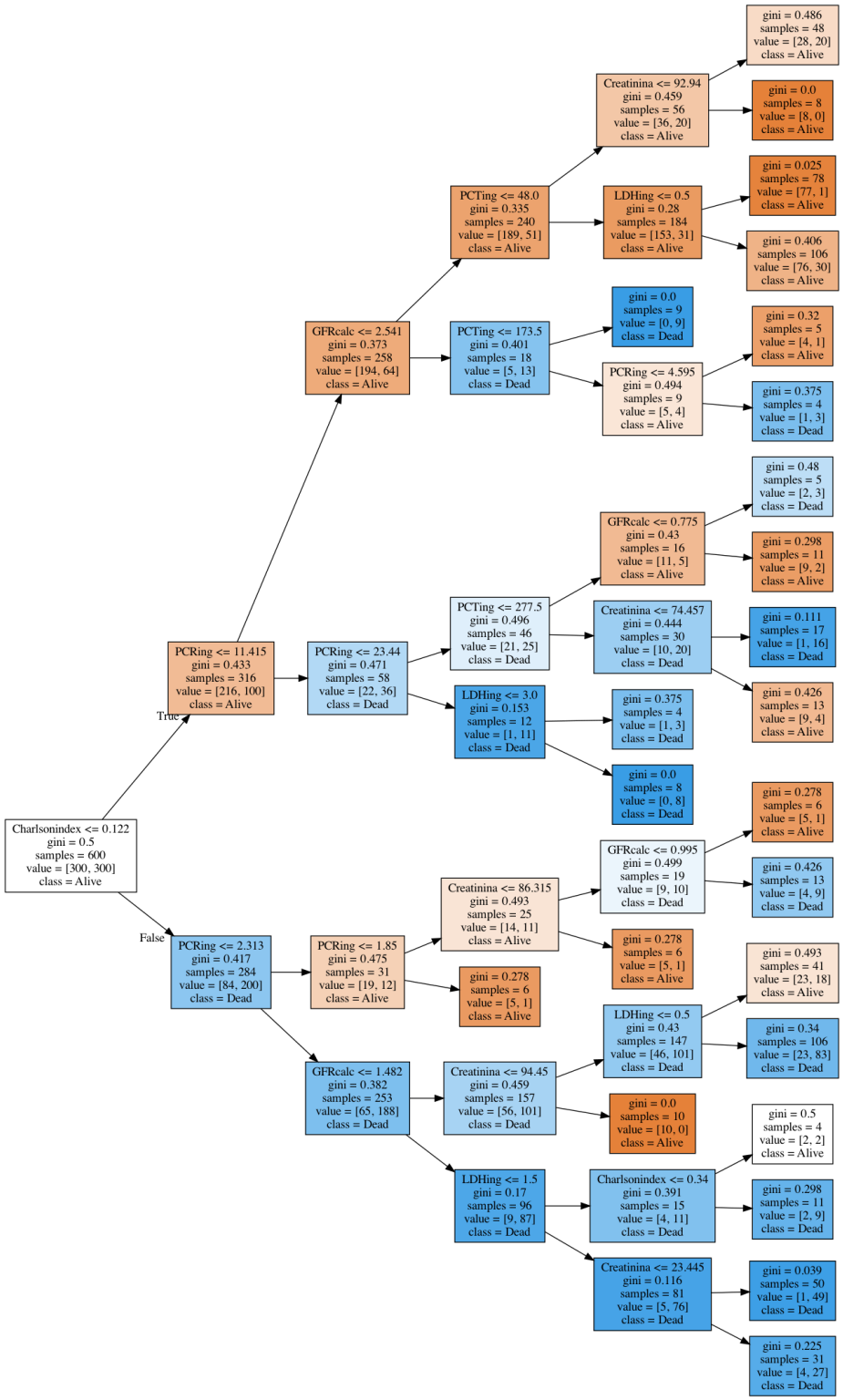


Fig. IV.3 The DT created from the dataset is based on clinical features that were extracted using the RF algorithm.

14.4.2 Experiments on Lung CT Scans

The second experiment was conducted using lung CT scans from patients. Before training, the CT scans underwent pre-processing, which involved segmentation to isolate the lungs. Segmentation was performed using the HU scale, a quantitative scale for describing radiodensity in medical CT imaging. On the HU scale, air is represented by a value of -1000 and bone by values ranging from $+700$ to $+3000$. Since bones are significantly denser than other soft tissues, they are more prominently displayed in CT scans. Using this information, it was possible to identify the regions containing the lungs, which are represented by values between -700 and -600 on the HU scale, and create a binary mask. After segmentation and applying the binary mask, the images were normalized, as each pixel is assigned a value on the HU scale, necessitating the adjustment of these values to a range between 0 and 1.

When working with 3D scans, it is important to capture not only the 2D features of the images but also the volumetric aspects of the CT scans. The network trained in this experiment consists of two blocks, each containing two 3D convolutional layers with a 5×5 kernel and a ReLU-like activation function, followed by max pooling layers with 98 and 160 neurons, respectively. These two blocks are followed by two fully connected layers: the first with 110 neurons and the second, the output layer, with 3 neurons.

The 3D-CNN was trained and validated on the MosMedData dataset, achieving approximately 70% accuracy on the validation set. It was also tested on the CT scans of the 96 patients on the Ferrara's dataset, where it achieved approximately 54% accuracy.

14.4.3 Neural Hierarchical Probabilistic Logic Programming (Neural HPLP)

As detailed in Section 10, the dataset used for training the HPLP consists of multiple interpretations. Each interpretation includes facts that describe the patient's critical state (serving as the label), the prediction of mortality during hospitalization (output of the DT), the condition of the lungs (output of the CNN), and the complete decision path of the DT to provide additional context, as illustrated in Example 14.1.

Example 14.1 *Consider the following interpretation that describes a Covid-19 patient with ID 98:*

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

The first three facts indicate that the patient was labeled as being in a critical condition, classified as deceased by the DT, and classified as having mild lung injury by the 3D-CNN. The remaining facts are those involved in the decision path used by the DT to predict the patient's vital state, $vital\_state(98, dead)$.

In the context of Neural HPLP, an interpretation is a collection of atoms, and each interpretation can be represented as a set of Prolog facts. This representation is crucial for encoding both the patient's data and the model's predictions in a structured manner.

Each interpretation in our framework is composed of:

- **Critical state label**: Indicates whether the patient is in a critical state, represented by the atom `critic(patient_id)`.
- **Mortality prediction**: The output of the Decision Tree (DT) model predicting the patient's vital state, represented as `vital_state(patient_id, state)` where `state` can be `dead` or `alive`.
- **Lung condition**: The output of the Convolutional Neural Network (CNN) indicating the severity of lung injury, represented by the atom `lung_injury(patient_id, severity)` where `severity` can be `minor`, `moderate`, or `severe`.
- **Clinical data**: Facts about clinical measurements and patient attributes used by the DT in its decision-making process. For example, `age(patient_id, value)`, `pcring(patient_id, value)`, `ldhing(patient_id, value)`, and so on.

Each fact is a Prolog atom of the form `predicate(arguments)`. These atoms are grouped together to form a single interpretation. For instance, the interpretation shown in Example 14.1 for patient ID 98 represents:

- The atom `critic(98)` labels the patient as being in a critical condition.
- The atom `vital_state(98,dead)` represents the DT's prediction of the patient's mortality.
- The atom `lung_injury(98, minor)` indicates the severity of lung injury as assessed by the CNN.
- Additional atoms like `age(98,94)` and `pcring(98,13.59)` provide clinical context used in the DT's decision path.

This Prolog-based representation allows for an interpretable and structured encoding of patient data, predictions, and model explanations within the Neural HPLP framework.

After training the DT and 3D-CNN models, inference was carried out on their respective test sets, which consisted of the 96 Ferrara patients. The results from these models were then compared to those provided by a domain expert, specifically a radiologist. According to the expert, 51 of the classifications were accurate. Based on these 51 correctly classified cases, 51 interpretations were constructed following the procedure detailed earlier. Among these interpretations, 20 were labeled as representing a critical state, while 31 were labeled as non-critical. The labeling was performed by the domain expert based on the clinical data and CT scan analysis.

Due to the limited amount of data, cross-validation was employed for the training process. The dataset was divided into three folds, each containing 17 interpretations, with stratified sampling. Two folds were used for training, while the remaining fold was used for testing. This process was repeated for all three possible fold combinations.

Two versions of the SLEAHP algorithm were applied: SLEAHP_DEEP, which uses Gradient Descent/Backpropagation (specifically with the Adam optimizer) to learn the parameters, and SLEAHP_EM, which uses Expectation Maximization for parameter learning. Since both versions produced only a few clauses, no regularization was applied. Both algorithms were trained for 10,000 iterations with early stopping. The default Adam hyperparameters were used in SLEAHP_DEEP.

14.4.4 Covid-19 Positivity Prediction

The trained CNN consists of the following components: four blocks, each containing one convolutional layer with 3×3 kernels and ReLU as the activation function, followed by a batch normalization layer. These blocks have 64, 64, 128, and 256 neurons, respectively. After these blocks, the network includes a global average pooling layer, a fully connected layer with 512 neurons, and a dropout layer. The output layer features 2 neurons corresponding to the two classes: Normal and Pneumonia.

For the clinical data, a similar approach to the one described in Section 14.4.1 was used. The key difference is that this time, RF and DT models were employed to predict whether patients were Covid-19 positive or negative, instead of predicting mortality during hospitalization.

The experiments for Covid-19 positivity prediction were conducted on the MosMedData dataset. For the purpose of this experiment, we focused on the binary classification task of distinguishing between normal lungs and those with pneumonia, grouping the severity levels into two main classes.

14.5 Results

14.5.1 Results on Clinical Data and Neural HPLP Analysis

The following presents the results of Neural HPLP on the dataset from Ferrara's hospital, including both clinical data and CT scans. Given that the dataset is imbalanced across categories, we generated the ROC and PR curves for each test fold and calculated the area under each curve (AUCROC and AUCPR), as outlined in [121]. The results were analyzed to assess the effectiveness of Neural HPLP in predicting the critical state of patients.

Table IV.3 displays the results along with the final loss values and their associated averages (across folds) for both SLEAHP_DEEP and SLEAHP_EM.

While both systems demonstrate strong performance in terms of AUCROC and AUCPR, it is important to note that SLEAHP_EM outperforms SLEAHP_DEEP. The perfect result achieved in Fold 3 can be attributed to the fact that the combination of data in Folds 1 and 2, which was used for training, provided enough informative

Table IV.3 Areas under the curves and loss for SLEAHP

SLEAHP_DEEP	AUCROC	AUCPR	Loss
Fold 1	0.637	0.801	-10.805
Fold 2	0.833	0.901	-8.997
Fold 3	1.000	1.000	- 5.576
Avg.	**0.836**	**0.901**	**-8.460**
SLEAHP_EM	AUCROC	AUCPR	Loss
Fold 1	0.959	0.959	-4.642
Fold 2	0.938	0.941	-5.459
Fold 3	1.000	1.000	-4.177
Avg.	**0.966**	**0.967**	**-4.759**

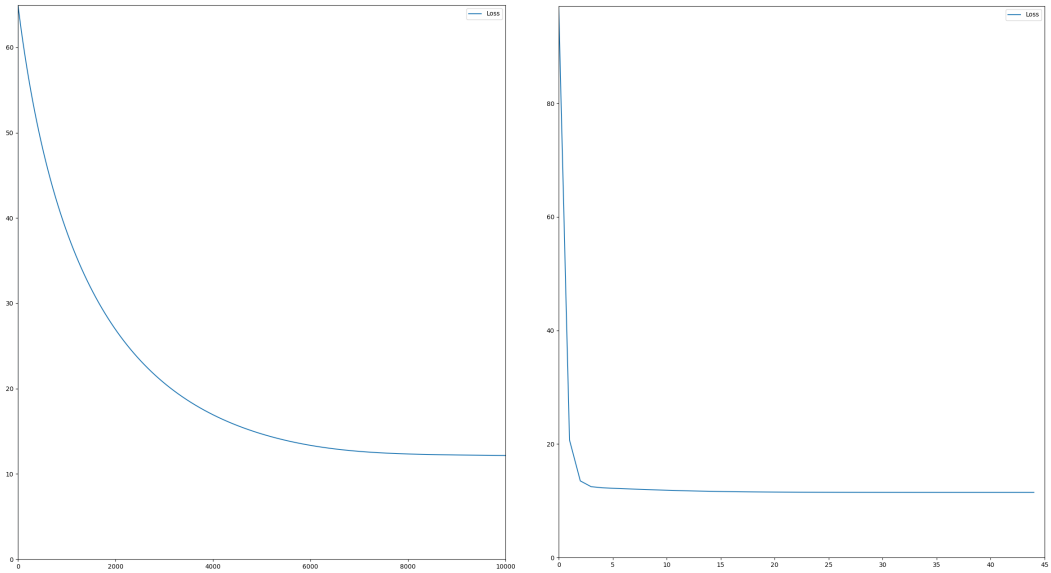


Fig. IV.4 Loss functions during training for the SLEAHP models. (a) SLEAHP_DEEP, (b) SLEAHP_EM.

content, allowing the algorithm to learn a more accurate theory. Additionally, the loss function value for Fold 3 is better compared to those for Folds 1 and 2. It is also noteworthy that SLEAHP_EM converges more quickly than SLEAHP_DEEP, as illustrated in Figure IV.4. An example of the learned rules is presented in Example 14.2.

Example 14.2 *Learned rules for predicting a Covid-19 patient's critical state.*

$critic : 0.998320 : -lung\_injury(severe).$
$critic : 0.074045 : -troponina(B).$
$critic : 0.003187 : -bnp(C), hidden\_3(C).$
$critic : 0.009686 : -age(D).$
$critic : 0.017233 : -pcring(E).$
$critic : 0.999999 : -pcting(F).$
$critic : 0.009842 : -gender\_f\_2(G), hidden\_8(G).$
$critic : 0.313651 : -age(H), hidden\_9(H).$
critic: 0.944125:- troponina(I),hidden_10(I).
$critic : 0.003780 : -ldhing(J), hidden\_12(J).$
$critic : 0.003762 : -charlsonindex(K), hidden\_13(K).$
$critic : 0.008438 : -age(L), hidden\_14(L).$
$critic : 0.003175 : -pcring(M), hidden\_15(M).$
$critic : 0.025554 : -ldhing(N), hidden\_16(N).$
$critic : 0.009720 : -gender\_f\_2(O), hidden\_17(O).$
$critic : 0.209133 : -age(P), hidden\_18(P).$
$critic : 0.000236 : -pcring(Q), hidden\_19(Q).$
$critic : 0.057380 : -ldhing(R).$
$critic : 0.010041 : -gender\_f\_2(S), hidden\_21(S).$
$hidden\_3(C) : 0.003238 : -greater\_than(C, 393.0).$
$hidden\_8(G) : 0.009921 : -greater\_than(G, 2.0).$
$hidden\_9(H) : 0.312308 : -greater\_than(H, 70.0).$
hidden_10(I) :0.944125:- greater_than(I,14.5).
$hidden\_12(J) : 0.003779 : -greater\_than(J, 101.87).$
$hidden\_13(K) : 0.003763 : -greater\_than(K, 21.0).$
$hidden\_14(L) : 0.010897 : -greater\_than(L, 78.0).$
$hidden\_15(M) : 0.00313 : -greater\_than(M, 22.79).$
$hidden\_16(N) : 2.911e-5 : -greater\_than(N, 54.37).$

$hidden\_17(O) : 0.008485 : -greater\_than(O, 2.0).$
$hidden\_18(P) : 0.209172 : -greater\_than(P, 85.0).$
$hidden\_19(Q) : 0.043254 : -greater\_than(Q, 7.82).$
$hidden\_21(S) : 0.009506 : -greater\_than(S, 2.0).$

This example clearly highlights that the feature *pcting* is one of the most significant clinical attributes for predicting whether a patient will progress to a critical state because it has the highest probabilistic parameter. The first clause indicates that a Covid-19 patient is highly likely to reach a critical state if their lungs are in serious condition, which directly correlates with the criteria for labeling interpretations defined in Section 14.4.3. Another notable explanation can be derived from the combination of rules highlighted in bold: these rules suggest that if a Covid-19 patient's *troponina* level exceeds 14.5, the patient is highly likely to become critical. Similar insights can be observed for other clinical attributes. Based on these findings, doctors could focus more on these clinical values when examining Covid-19 patients in the hospital, potentially enhancing their diagnosis and decision-making processes by relying on the learned explanations.

14.5.2 Results on Covid-19 Positivity Prediction

This section details the results of applying Neural HPLP to a more established dataset, as described in the previous section. This experiment aims to validate and assess the effectiveness of Neural HPLP in identifying patients who test positive for Covid-19. Initially, an RF was trained on the full set of clinical data to classify patients as either positive or negative for Covid-19. The RF achieved an accuracy of 93.9%, an AUCROC of 0.93, and an AUPRC of 0.86.

Next, using the trained RF, the top 10 most important clinical attributes were identified: Temperature, Coefficient of 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. A new dataset, including only these features, was created for training a DT. The DT achieved an accuracy of 90.14%, an AUCROC of 0.9045, and an AUCPR of 0.9208.

The trained CNN was evaluated on the test set, yielding an accuracy of 81.77%, an AUCROC of 0.823, and an AUCPR of 0.8709.

The final phase of the experiment involved using systems SLEAHP_EM and SLEAHP_DEEP. A dataset of 203 interpretations (one for each patient in the test set) was generated based on the results from the DT and CNN. The outcomes were as follows: SLEAHP_DEEP achieved an AUCROC of 0.8188 and an AUCPR of 0.7210, while SLEAHP_EM performed better with an AUCROC of 0.8956 and an AUCPR of 0.8144.

In summary, this experiment with a consolidated dataset reaffirms the accuracy and, more importantly, the effectiveness of Neural HPLP in predicting Covid-19 positivity and providing interpretable results.

15 Symbolic DNN-Tuner in Tiny Machine Learning

Tiny ML is an emerging subfield of ML that focuses on the development of algorithms and hardware tailored for low-power devices with limited computational capacity. In this chapter, we specifically address the use of NAS algorithms to design a DNN that adheres to specific hardware constraints. To achieve this goal, we have modified the optimization formula of Symbolic DNN-Tuner to incorporate the number of FLOating Point Operations (FLOPS) that the hardware can handle as a hardware constraint. The objective is to create an optimization function that maximizes the accuracy of the generated models while aiming to approach as closely as possible the maximum number of FLOPS that the hardware can manage. Essentially, the goal is to design an efficient DNN capable of operating effectively on edge devices with power and consumption limitations.

15.1 Related works

This section aims to explore various NAS approaches and their potential integration to design NNs that respect the physical constraints of embedded systems, such as latency, memory usage, and energy consumption. In [122], Elsken et al. provide a comprehensive overview of the current research in NAS.

The first set of NAS systems discussed in [122] is that of morphism-based NAS systems that begin with a simple network and iteratively modify its architecture by adjusting elements such as depth, width, kernel size, and even entire subnetworks. We focus on three types of morphism-based NAS systems, each represented by a specific example:

a An AutoML system that offers multiple search strategies [123].

b A symbolic tuner that leverages symbolic rules and Bayesian Optimization (BO) to explore the search space [124, 125].

c Tuners designed for microcontroller systems that utilize Multi-Objective Optimization (MOO) [126–128].

Auto-Keras [123], an open-source system built on Keras, exemplifies an AutoML system with a variety of available search strategies. It is designed to enable domain experts with limited ML knowledge to easily apply ML techniques. Auto-Keras offers several tools for defining and exploring the architecture search space using different algorithms (e.g., BO, random search, grid search) and strategies (e.g., penalization, rejection).

Symbolic DNN-Tuner [125] employs BO, which is a state-of-the-art Hyperparameter Optimization (HPO) algorithm for DL. BO tracks past results to build a probabilistic model, thereby constructing a probability density over the hyperparameter space. Symbolic DNN-Tuner aims to enhance BO when applied to DNNs by analyzing the network's performance on training and validation datasets. The system uses symbolic tuning rules, implemented in PLP [24], to logically assess the results and adjust the network architecture and hyperparameters for better performance.

Figure IV.5 illustrates the architecture of Symbolic DNN-Tuner. The neural block outputs values from the trained network, which are then processed by two components: the Improvement Checker (2), which evaluates the network's progress, and the symbolic program (1), which consists of three parts:

1. Facts, that store the data obtained from the neural block.
2. Diagnosis, that analyzes the neural network's behavioral issues.
3. Tuning, that contains the Symbolic Tuning Rules.

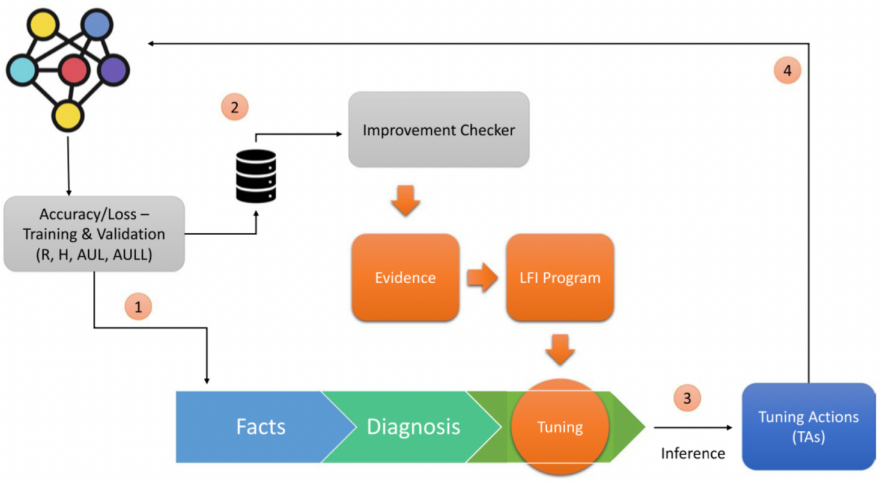


Fig. IV.5 Symbolic DNN-Tuner architecture.

Using ProbLog [25] inference, the symbolic program can be queried to produce Tuning Actions (TAs) (3). These TAs are then applied to the neural block, modifying the DNN structure or the hyperparameter search space (4).

μNAS [126] serves as an example of a NAS system tailored for microcontrollers. This system focuses on integrating DL capabilities into small personal Internet of Things (IoT) devices, enabling local computations and ensuring that user data remains on the device, thereby enhancing privacy and autonomy.

IoT devices are powered by microcontroller units (MCUs), which are ultra-small computers with highly limited resources housed within a single chip. This design makes MCUs more cost-effective and energy-efficient compared to desktop devices or smartphones. However, these benefits come at the cost of significantly reduced computing power.

In [126], the authors propose a NAS approach that operates within a constrained search space, designed to accommodate the limited resources of MCUs. They employ a MOO function to explore this search space, aiming to identify the optimal parameters α^*, defined as follows:

$$\begin{aligned}\alpha^* &= \arg\min_{\alpha \in \mathscr{S}} \mathscr{L}(\alpha) \\ &= \arg\min_{\alpha \in \mathscr{S}} \Big(1.0 - \mathit{ValAccuracy}(\alpha), \\ &\qquad \mathit{ModelSize}(\alpha), \\ &\qquad \mathit{PeakMemUsage}(\alpha), \\ &\qquad \mathit{Latency}(\alpha)\Big)\end{aligned} \tag{IV.2}$$

Here, *ValAccuracy* measures the network's accuracy on the validation set, *ModelSize* refers to the network's size, *PeakMemUsage* denotes the maximum memory usage during operation, and *Latency* is the time required to perform a single inference. The algorithm developed by Liberis and Lane [129] was used to calculate the maximum number of stored parameters. To optimize this multi-objective function, the authors applied the method introduced by Paria et al. [130].

The results presented in [126] demonstrate that with a carefully designed search space and explicit consideration of physical constraints, it is possible to develop a NAS system that efficiently discovers resource-constrained models for various image classification tasks. Table IV.4 compares the key results of μNAS with other constrained NAS approaches. The authors used Multiply-Accumulate Operations (MACs) as a metric to quantify latency in relation to model size.

Dataset	Model	Acc. (%)	Model size	MACs
MNIST	SpArSe [131]	98.64	2770	-
	BonsaiOpt [132]	94.38	490	-
	ProtoNN [133]	95.88	63′900	-
	µNAS	99.19	480	28.6 K
CIFAR-10	LEMONADE [134]	≈ 91.77	10K	-
	µNAS	86.49	11.4 K	384 K
Speech Commands	RENA [135]	94.04	47 K	$\approx 700M$
	DS-CNN [136]	94.45	$<$ 38.6 K	$\approx 2.7M$
	MCUNet [137]	91.20	$<$ 1 M	-
	µNAS	95.36	37 K	1.1 M

Table IV.4 Pareto-optimal architectures discovered by *µNAS* vs other Resource - Constrained ML NAS

Problem	Symptoms	TAs
Overfitting	Gap between accuracy or loss in training and validation	Regularization and Batch Normalization Increase dropout Data augmentation
Underfitting	High loss Low accuracy	Decrease the learning rate Increase the number of neurons Addition of fully connected layers Addition of convolutional blocks
Increasing loss	Loss trend analysis	Decrease the learning rate
Fluctuating loss	Fluctuation of the loss	Increase the batch size Decrease the learning rate
Low learning rate	Evaluation of the shape of the loss	Increase learning rate
High learning rate	Evaluation of the shape of the loss	Decrease learning rate
Peak Memory Usage	High number of Parameters for layer	Decrease the number of neurons
Model Size	High number of total Parameters	Decrease the number of layers or the number of neurons
Latency	High FLOPS	Decrease the number of layers or the number of neurons

Table IV.5 Problem, Symptoms and TAs

The methods discussed above highlight the differences between various NAS approaches. However, this raises important challenges, such as the integration of MOO methods with one other, and the use of constraints without MOO.

One potential direction for future research is to incorporate a multi-objective function into the Symbolic DNN-Tuner [125], similar to the approach used in the μNAS model. In this case, the optimization function would be multi-objective, aiming to optimize the NN while taking physical constraints into account. After optimization, TAs could be applied to adjust the network whenever it violates these constraints. Table IV.5 shows the TAs used by the Symbolic DNN-Tuner [125], along with potential additional rules for addressing new constraints, listed in the lower part of the table.

Another approach to enforcing constraints is to incorporate a penalty into the metric used to evaluate the networks. For example, AutoKeras introduces a penalty

term to the network loss function, where the penalty increases as a constraint is violated. In the context of physical constraints, such as an upper limit on FLOPS, the penalty would increase as the network approaches the limit. As a result, the NAS system would tend to favor smaller networks, as larger networks would incur higher loss values due to the penalty.

15.2 Experiments

As discussed above, the optimization process in Symbolic DNN-Tuner is built upon the optimization techniques employed by BO.

In contrast to the original tool, here we use these optimization methods to refine the neural architecture with respect to both network performance and device-specific objectives.

More specifically, we introduce the number of FLOPS as a key constraint during model tuning. FLOPS serve as an approximate metric to quantify the available computational power of the underlying execution platform.

Similar to other NAS approaches [135], the focus at this early stage of tool development is not to model hardware costs with absolute precision. Instead, the goal is to demonstrate that by considering approximate metrics, the proposed NAS can effectively optimize them. This approach leads to a clear cost quantification in terms of fundamental operations and yields easily interpretable results without over-specializing the NAS framework for a particular hardware platform.

The overall objective is to maximize model accuracy while designing a network architecture suitable for deployment on resource-constrained platforms, with a number of FLOPS that closely approximates a predetermined threshold.

Consequently, the BO approach is focused on solving the following problem:

$$\min_{x \in D} f_{flops}(x) \tag{IV.3}$$

where f_{flops} is the objective function, x represents the input in $\mathbb{R}^d$, d is the number of hyperparameters, and D is the search space, conceptualized as a hyper-cube. Specifically, f_{flops} is defined as:

$$f_{flops} = -|Acc_i - |FLOPS_{th} - FLOPS_i| \times \varepsilon| \tag{IV.4}$$

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

Fig. IV.6 STRs for imposing the constraint on the number of FLOPS.

Here, Acc_i and $FLOPS_i$ represent the accuracy and the number of FLOPS for the ith model, respectively, while $FLOPS_{th}$ denotes the FLOPS threshold that the hardware can support. ε is a hyperparameter used to balance and adjust the impact of the FLOPS gap in the objective function and has been empirically set to 0.33 through a parameter sweep. Both $FLOPS_i$ and $FLOPS_{th}$ are normalized between 0 and 1.

To enable Symbolic DNN-Tuner to optimize DNNs with the FLOPS constraint in mind, we introduced a new Symbolic Tuning Rule that activates when the DNN exceeds the FLOPS limit. Specifically, we defined a new `problem(thr_exceeded)` literal, which is triggered when the number of FLOPS surpasses the threshold, and introduced two new TAs. The first TA reduces the number of neurons in various layers, while the second removes the last convolutional layer, together with any associated layers (e.g., pooling, dropout, etc.) to maintain architectural consistency. Figure IV.6 illustrates these two new STRs with their respective probabilistic weights, which are randomly initialized and adaptively tuned during system execution.

In the extended Symbolic DNN-Tuner, the designer's expertise is embedded in the rules used to iteratively mutate a hyperparameter that is likely causing issues in the current network model. This approach is fundamentally different from that of AutoKeras, where network mutations occur without awareness of potential issues. As demonstrated by the experimental results, this methodology enhances the rate of improvement in the quality of the solutions produced.

We evaluated the extended Symbolic DNN-Tuner and the modified version of AutoKeras, both enhanced with hardware awareness, on image classification tasks using the CIFAR10 dataset [138]. The evaluation involved eight distinct experiments, each characterized by progressively relaxed FLOPS thresholds: 10M, 30M, 40M, 80M, 90M, 100M, 110M, and 120M FLOPS.

This series of experiments represents a design space exploration similar to what edge device designers might undertake to assess the trade-off between accuracy and

computational cost. High-end NN models are prone to exceeding resource budgets, while smaller models may fail to achieve adequate levels of accuracy.

15.3 Results

The experiments conducted with both AutoKeras and Symbolic DNN-Tuner reveal two markedly different behaviors. In the case of AutoKeras, the exploration phase generates numerous NNs with very low accuracy, often requiring significant computation time to evaluate accuracy, as each network must be fully trained. Conversely, Symbolic DNN-Tuner, through intelligent exploration, is able to identify a strong initial model, which is subsequently refined through hyperparameter tuning. This behavior is illustrated in Figure IV.7 where the x-axis reports the FLOPS and the y-axis the accuracy.

Each cross represents a model trained by AutoKeras, while each circle represents a model trained by Symbolic DNN-Tuner. The colors of the crosses and circles indicate to the FLOPS thresholds, which are depicted as vertical lines. We can see that the crosses are more scattered, indicating that AutoKeras tests a wide variety of models. A significant number of crosses are located in the lower portion of the graph, corresponding to models with low accuracy. In contrast, Symbolic DNN-Tuner quickly identifies a strong model and incrementally improves it, rather than testing widely varying architectures.

The accuracy trends of the two frameworks for each FLOPS threshold are shown in Figure IV.8, where Symbolic DNN-Tuner consistently outperforms AutoKeras, with an average accuracy improvement of 31%.

Execution times are presented in Figure IV.9, showing that Symbolic DNN-Tuner is consistently faster for every threshold, with an average time reduction of up to 78%. This advantage is attributed to AutoKeras's approach, which unnecessarily extends execution time by constructing and training networks with a high likelihood of failing to reach an optimal solution. This behavior underscores one of the key capabilities of Symbolic DNN-Tuner: its symbolic block allows for adaptive modification of the search space, saving time by dynamically excluding hyperparameter values that would not lead to a potential optimum.

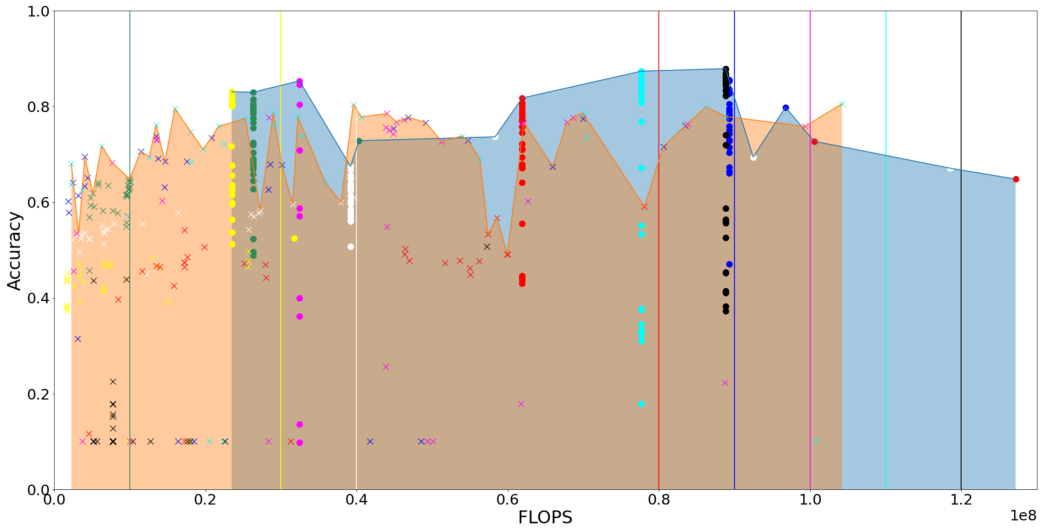


Fig. IV.7 A comparison of accuracy between the Symbolic DNN-Tuner (blue area) and Autokeras (orange area) across different FLOPS thresholds. The x-axis represents MFLOPS, while the y-axis displays the accuracy achieved by the models. Vertical lines denote the FLOPS used in each experiment and are listed in the legend. Models generated by the Symbolic DNN-Tuner are marked with an $\times$, and those from Autokeras are represented by circles.

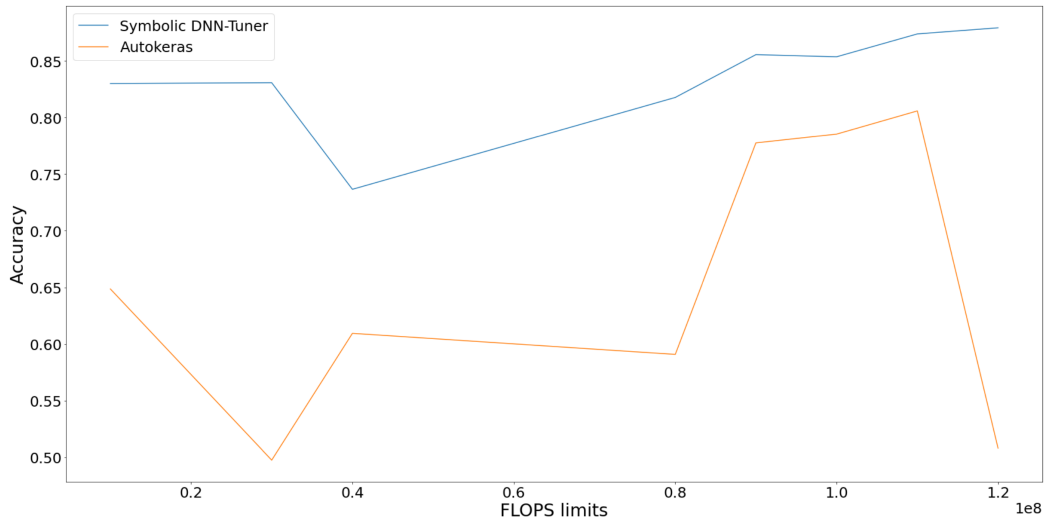


Fig. IV.8 A comparison of the accuracy between AutoKeras (orange line) and Symbolic DNN-Tuner (blue line) models at different FLOPS thresholds. The x-axis displays the FLOPS thresholds, while the y-axis shows the accuracy achieved by the top-performing models.

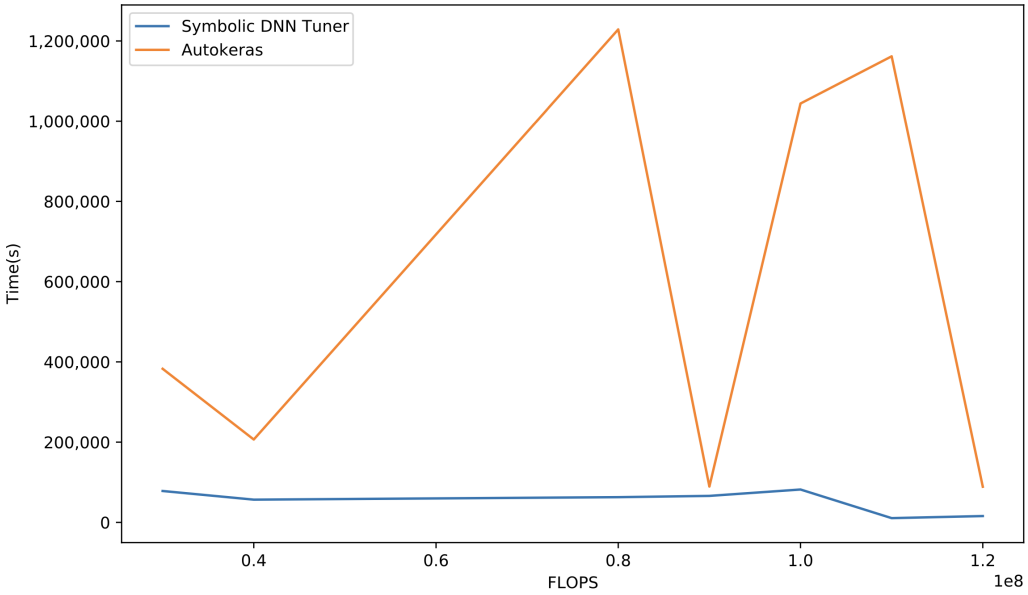


Fig. IV.9 The running time of AutoKeras (orange line) and Symbolic DNN-Tuner (blue line) at different FLOPS thresholds. The x-axis represents the FLOPS thresholds, while the y-axis indicates the time taken by each framework to complete 30 iterations under the specified FLOPS constraint.

16 Cross Entropy Overlap Distance in Anomaly Detection

In contemporary industrial production, the identification of anomalies with computer vision plays a critical role [139]. However, images captured in industrial environments often contain elements that are not part of the primary area of interest [140]. Examples include objects moving on conveyor belts or items consisting of multiple components, not all of which are relevant for defect detection.

In certain scenarios, the precise location and shape of the object are known, allowing traditional preprocessing techniques, such as applying masks, to be effective. However, this is not always feasible. To address this limitation, we have developed the method based on CNN introduced in Section 11.2. In this section, we present a case study using this method.

16.1 Problem Description

To encourage the neural network to concentrate on particular parts of the image rather than the entire image, we defined a loss function called CEOD. This loss function is given in Equation (III.7). For this specific application, the Overlap Distance term is multiplied by $y_i \in [0,1]$, the label of the example. In anomaly detection, images without defects typically lack a specific area in the heatmap with a concentration of high-intensity pixels, resulting in a uniform and low-intensity heatmap. Therefore, we use the following equation:

$$\text{CEOD} = -\sum y_i \log(p(y_i)) + \omega y_i \left(-\ln \left(\frac{|A_d \cap A_{gt}|}{\min(|A_d|, |A_{gt}|)} \right) \right) \tag{IV.5}$$

In our experiments, after different tests, the optimal value for ω was found to be 0.001.

The problem is due to the heterogeneity of data in industrial settings, where the area of interest may differ from the actual content of the image. It is not always feasible to mask the non-relevant regions, which can result in the inclusion of irrelevant information being processed by the model.

Fig. IV.10 illustrates two examples of this issue. On the left, intrinsic elements of the product are present, such as a screw hole, which may or may not be present and may appear in various positions within the image, which the model might incorrectly classify as defects. On the right, a curved surface is shown; the curvature may create shadowed areas that could lead the network to erroneous conclusions [141].

To address this problem, it's crucial for the network to concentrate on specific areas of the image, prioritizing the defects within the ROI over those outside it. Although this issue resembles a segmentation task, the field of industrial anomaly detection presents a challenge: defects are not predefined, so there is no labeled data available to train a segmentation network. Therefore, we have opted for a binary CNN that determines the presence or absence of faults.

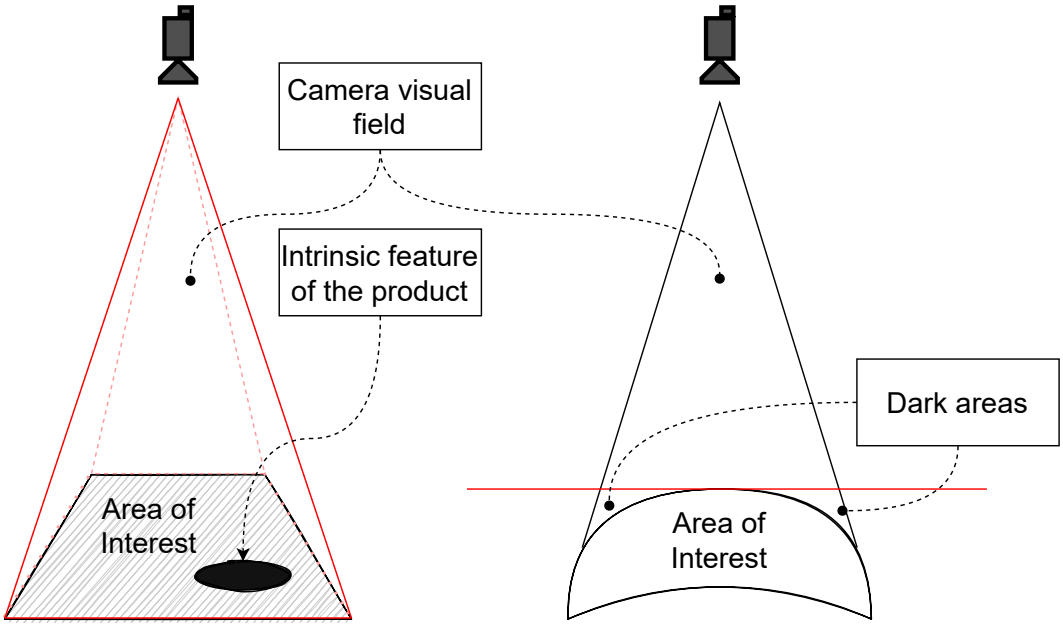


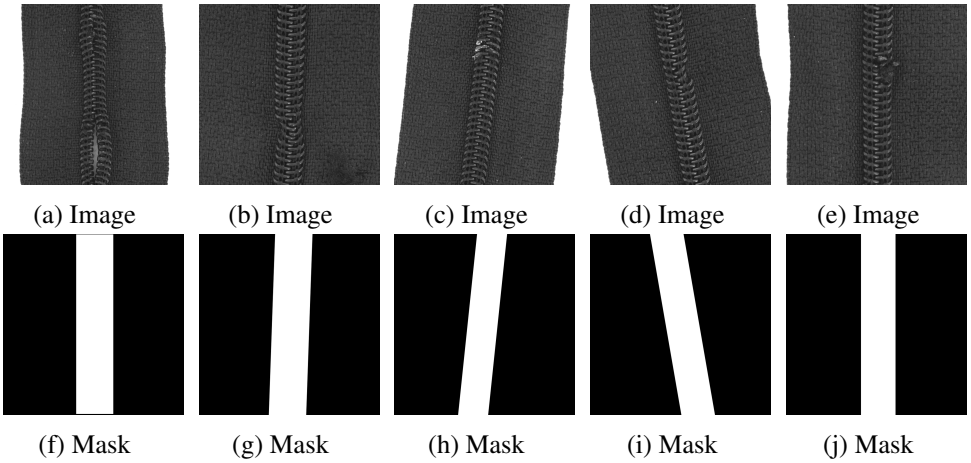
Fig. IV.10 Examples of possible problems with industrial data settings.

16.2 Related work

In ML, anomaly detection has long been a topic of significant interest, particularly within Industry 4.0, where defect identification is one of the primary tasks. Numerous surveys on anomaly detection exist in the literature [142, 143]. Our work primarily focuses on the use of CNNs to identify structural defects in production lines. However, most existing studies in the literature focus on detecting anomalies across the entire image. Weimer et al. [144] discuss the challenges of manually defining specific feature representations for each new industrial problem addressed by CNNs. In [145], the authors employ Triplet loss [146] with CNNs to identify defects in an industrial context.

Other studies [147–149] use generative methods, such as GANs [41] and VAEs [150]. These approaches learn the distribution of a particular class and use the difference between the reconstructed image and the original to identify anomalies. These methods have proven effective in reconstructing simple anomalies.

In [151], the authors demonstrate the importance of using CAMs to verify whether the network focuses on the regions of interest. In [152], the authors introduce the use of Grad-CAM to build a self-supervised system aimed at removing noise from images and creating a more robust anomaly detection system. Venkataramanan



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

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

Fig. IV.11 Example of the augmented zip dataset.

et al. [153] use activation maps to guide the training of autoencoders, forcing the network to focus only on normal areas while ignoring abnormal ones, thereby improving anomaly detection. Song et al. [154] propose an Anomaly Segmentation Network (AnoSeg) that generates an anomaly map by segmenting abnormal regions.

16.3 Dataset

The experiments were conducted on two distinct datasets. The first dataset is a subset of MVTec AD [155, 156], specifically focusing on the zipper images. Images in this sub-dataset were augmented to change their shape and proportions. Figure IV.11 provides examples of images from this dataset. Each image is accompanied by a corresponding binary mask. The dataset comprises a total of 400 images, with 216 being non-defective and 184 defective. The dataset was divided into training, validation, and testing sets with proportions 70%, 20%, and 10% respectively. This dataset was selected for its relevance to the problem at hand. To ensure alignment with the specific problem under investigation, only images displaying defects on the zipper were considered. Consequently, all images exhibiting defects solely on the surrounding fabric were reclassified as non-defective.

The second dataset used in these experiments was provided by an Italian company. Due to a Non-Disclosure Agreement (NDA) associated with the project, detailed information about this dataset cannot be disclosed. However, it is important to note that this dataset represents a real-world industrial application. The dataset comprises

5064 images, with 2818 non-defective and 2246 defective images. The training and validation sets were allocated 80% and 20% of the data, respectively. The test set consists of 893 images, of which 467 are non-defective and 426 are defective. The images depict the surface of a product developed by the company, characterized by its round shape and smooth, reflective surface. A light pattern was applied to these products using a specialized illuminator to accentuate surface defects. However, due to the reflective nature of the surface, the application of the light pattern induces random scattering effects, visually resembling defects. These effects are infrequent and highly variable.

In the field of anomaly detection, data augmentation is commonly employed to address class imbalances. However, this approach can introduce challenges, such as overfitting, and necessitates increasingly sophisticated augmentation strategies, a topic of significant interest in the literature [157–159]. Given these considerations, we opted not to implement data augmentation. As detailed in the following section, this decision did not hinder the achievement of state-of-the-art results.

16.4 Experiments

We now detail the datasets, the general experimental setup, and the results obtained. All experiments were conducted on the Marconi100 cluster provided by Cineca, where each node is equipped with 2 IBM POWER9 AC922 CPUs and 4 NVIDIA Volta V100 GPUs with 16GB of RAM, interconnected via NVLink 2.0. We experimented with two different CNNs: EfficientNet-B0 [160], pre-trained on ImageNet [161], and a custom CNN comprising 8 convolutional blocks (each convolutional block consists of a convolutional layer followed by a batch normalization layer) with a max-pooling layer after every two blocks. The custom network contains 294,994 trainable parameters, requires 1.33 GFLOPS, and has an inference time of 10.8 milliseconds per image. The custom CNN was trained from scratch.

Using the MVTec AD dataset, we trained both networks with and without the CEOD contribution. We then compared the performance of the new CEOD loss function against the standard classification loss, evaluating the results through confusion matrices, accuracy, ROC AUC, and loss.

In each experiment, as previously mentioned, EfficientNet-B0 was pre-trained on ImageNet. We evaluated both Transfer Learning (TL) and Fine-Tuning (FT)

approaches. The results indicate that FT yields significantly better outcomes than TL. This can be attributed to the fact that the network was originally trained on ImageNet using a standard loss function (categorical CE). Consequently, retraining only the final dense layer may not sufficiently enhance the contribution of the new loss function. In our experiments, FT was conducted by unfreezing the last 20 convolutional layers. Various other network configurations will be explored in future work to further assess the impact of architectural choices on performance. The custom CNN, on the other hand, was trained from scratch using the newly proposed loss function. All experiments were performed with ω set to 0.001, a batch size of 32, and the Adamax optimizer with a learning rate of 0.002.

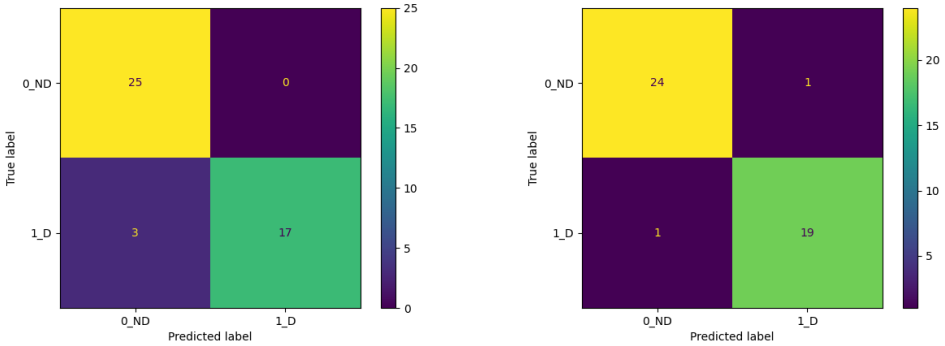
16.5 Results

Table IV.6 presents the results obtained from training EfficientNet-B0 and the custom CNN on the MVTec AD dataset. Both networks were trained using the standard categorical CE loss as well as the proposed CEOD loss. The results show that the application of the new OD method enables the networks to achieve improved metrics during the validation phases. Figure IV.12 displays the confusion matrix of EfficientNet-B0 on the test set. The figure reveals that the application of OD to the loss function (i.e., using the CEOD loss) enhances the network's ability to identify defects, albeit with a slight worsening of the identification of non-defective samples. The network trained with CEOD achieved an accuracy of 95.5% and an AUC-ROC of 0.95. In contrast, EfficientNet-B0 trained with traditional CE attained an accuracy of 93.3% and an AUC-ROC of 0.925.

Figure IV.13 displays the confusion matrix of the custom CNN on the test set. The custom CNN trained with CEOD achieved an accuracy of 73.3% and an AUC-ROC of 0.74 on the same test set, whereas the custom CNN trained with standard CE reached an accuracy of 48.8% and an AUC-ROC of 0.53. Figure IV.14 provides examples of heatmaps generated by a network trained using the standard approach compared to one trained with CEOD. The time required to perform 1,000 training epochs with EfficientNet-B0 using standard CE was 2 hours, 28 minutes, and 24 seconds. In contrast, training EfficientNet-B0 with CEOD took 2 hours, 36 minutes, and 29 seconds for the same number of epochs. The custom CNN required 4 minutes and 30 seconds to complete 100 training epochs with standard loss, compared to 4 minutes and 28 seconds when trained with CEOD.

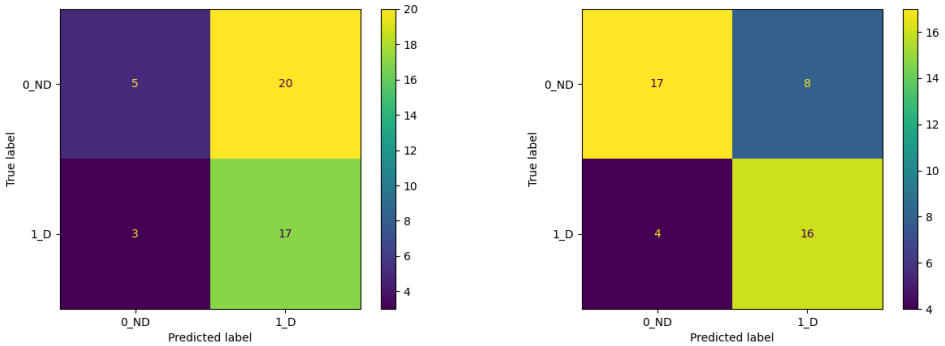
CNN	Exp.	Training			Validation		
		Accuracy	Loss	OD	Accuracy	Loss	OD
EfficientNet-B0	CE	1.000	0.005	-	0.992	**0.022**	-
	CEOD	0.990	0.005	0.0005	**1.000**	0.010	0.00032
CustomNet	CE	0.999	0.008	-	0.910	0.350	-
	CEOD	0.998	0.010	0.0002	0.960	0.090	0.00030

Table IV.6 MVTec AD Dataset: The acronyms *CE* and *Exp.* stand for *Cross-Entropy (CE) loss* and *experiment*, respectively. Bold values highlight the best results, showing the highest accuracy and the lowest loss for the dataset.



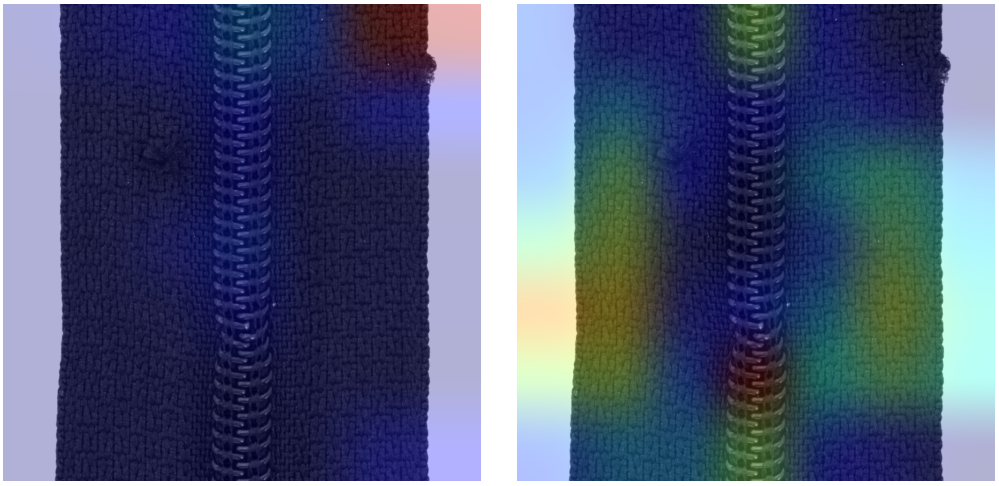
(a) CE (b) CEOD

Fig. IV.12 Confusion matrices generated using EfficientNet-B0 on the MVTec AD dataset. The labels *0_ND* and *1_D* correspond to the classes without defects and with defects, respectively.



(a) CE (b) CEOD

Fig. IV.13 Confusion matrices generated using custom CNN on the MVTec AD dataset. The labels *0_ND* and *1_D* correspond to the classes without defects and with defects, respectively.



(a) CE (b) CEOD

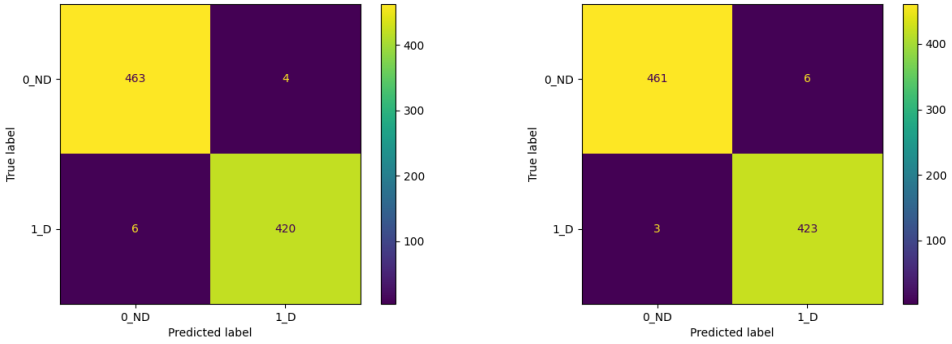
Fig. IV.14 (a): A heatmap generated using standard Cross-Entropy (CE) loss, where the defect on the zipper is not highlighted, in contrast to the external defect in the upper right corner. (b): A heatmap produced by the network trained with Cross-Entropy with Outlier Detection (CEOD). Here, the defect on the zipper is clearly highlighted, while external defects are ignored by the network.

In Table IV.7 shows the results of EfficientNet-B0 and the custom CNN on the industrial real-case dataset. The results show that the network trained with CEOD exhibits superior accuracy during the validation phase. Additionally, there is a noticeable improvement in the performance of the network trained with FT compared to that trained using only TL. The slight increase in loss observed with CEOD can likely be attributed to the higher complexity of the industrial problem compared to the MVTec AD benchmark. The industrial dataset poses more significant challenges as it is a real-world use case, where the addition of the OD component to the CE loss may result in a small incrase of the loss. This phenomenon does not occur with the MVTec AD dataset because its simpler nature allows the network trained with CEOD to outperform the one trained with standard loss, negating any negative impact of the added OD component.

Despite the strong results achieved by the network trained with standard classification loss, the CEOD approach further improves performance. Figures IV.15 and IV.16 show the confusion matrices of EfficientNet-B0 and the custom CNN respectively on the test set of the industrial dataset. These figures demonstrate that networks trained with CEOD achieve better results in defect identification, albeit with a slight deterioration in the identification of non-defective samples. Specifically,

CNN	Exp.	Training			validation		
		Accuracy	Loss	OD	Accuracy	Loss	OD
EfficientNet-B0 - TL	CE	0.956	0.11000	-	0.976	0.102	-
	CEOD	0.957	0.10000	0.000550	0.977	0.110	0.00036
EfficientNet-B0 - FT	CE	0.990	0.00020	-	0.995	0.017	-
	CEOD	1.000	0.00070	0.000380	**0.998**	0.045	0.00032
CustomNet - FT	CE	1.000	0.00001	-	0.9800	0.053	-
	CEOD	1.000	0.00002	0.000017	0.9965	**0.013**	0.00002

Table IV.7 Industrial Dataset: Results for EfficientNet-B0 are shown for Transfer Learning (TL) and fine-tuning (FT). Bold values indicate the best performance, representing the highest accuracy and lowest loss for the dataset.

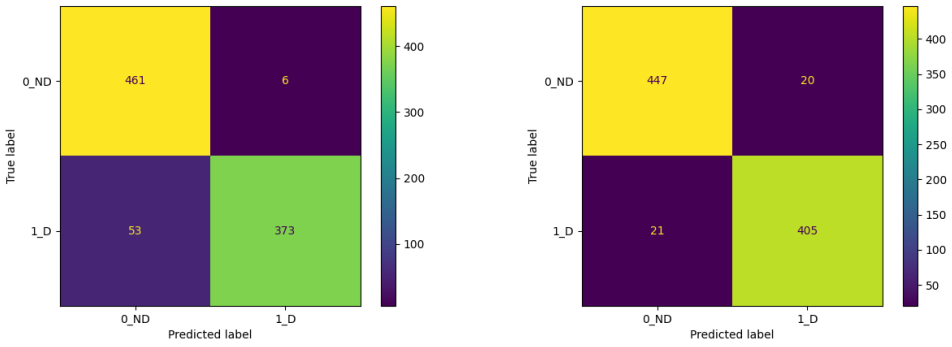


(a) CE (b) CEOD

Fig. IV.15 Confusion matrices on the test set of the industrial dataset obtained using EfficientNet-B0. The labels *0_ND* and *1_D* correspond to the classes without defects and with defects, respectively.

EfficientNet-B0 trained with CEOD achieves 98.9% accuracy and an AUCROC of 0.99, compared to 98.8% accuracy and 0.98 AUCROC when trained with standard CE on the test set. Similarly, the custom CNN trained with CEOD attains 95.4% accuracy and 0.95 AUCROC, outperforming the same network trained with standard loss, which achieves 93.3% accuracy and 0.93 AUCROC on the test set.

Regarding the computational cost, the time required to complete 360 training epochs with EfficientNet-B0 using standard CE was 9 hours, 25 minutes, and 31 seconds, whereas the same number of training epochs with CEOD took 8 hours, 12 minutes, and 47 seconds. For the custom CNN, 80 training epochs with standard loss required 1 hour, 44 minutes, and 25 seconds, compared to 1 hour, 44 minutes, and 5 seconds with CEOD.



(a) CE (b) CEOD

Fig. IV.16 Confusion matrices on the test set of the industrial dataset obtained using the custom CNN. The labels *0_ND* and *1_D* correspond to the classes without defects and with defects, respectively.

17 Network Intrusion Detection System

In today's digital world, cybersecurity solutions are crucial for safeguarding critical network assets. The rapid proliferation of IoT devices has significantly increased the impact of cyber threats, rendering traditional network security measures often inadequate. As a result, NIDS have become essential tools in defending against these evolving threats. NIDS, whether based on predefined *signatures* or anomaly detection, play a vital role in protecting the digital infrastructure [162].

The integration of AI into NIDS has led to the development of more advanced techniques for detecting cyber threats [163]. AI driven NIDS are capable of adapting to evolving threats, processing large volumes of data, and identifying complex patterns. However, these systems also present challenges. AI models often lack transparency, require vast amounts of training data, and are prone to overfitting [164].

As cyber threats continue to grow in sophistication, NIDS face unprecedented challenges. These systems are typically trained and tested on known attack types, limiting their effectiveness against new, previously unseen attacks. With the increasing number of IoT devices and the emergence of more sophisticated threats, detecting unknown attacks, or zero-day attacks, has become one of the most pressing challenges for NIDS.

NeSy AI is at the forefront of a new wave of AI research, offering promising advancements in cybersecurity. By combining neural pattern recognition with symbolic

reasoning, NeSy AI offers dynamic, robust, and transparent systems for combating cyber threats.

In the realm of cybersecurity, robustness refers to the ability of NIDS to maintain effective performance in the face of various challenges, including evolving and sophisticated cyber threats. A robust NIDS can accurately detect both known and unknown attacks, adapting to changes in attack patterns without being easily evaded or overwhelmed by adversarial tactics. This includes resilience against attempts to bypass detection mechanisms, such as evasion techniques [165] or adversarial attacks [166], and the ability to function effectively even when dealing with high traffic volumes and diverse network environments. As technology advances, NIDS must be equipped to counter increasingly sophisticated and novel threats.

17.1 Dataset

In the domain of NIDS, the selection of the right data is critical for enhancing detection. This discussion explores two key aspects of data usage in AI driven NIDS: the effectiveness of network flow data versus packet-level data, and the comparison between real-world data and synthetic data. Each of these data types offers distinct advantages and presents unique challenges that can significantly impact the effectiveness and performance of NIDS.

In AI-enhanced NIDS, the use of two data types, network flow information and packet-level data, is essential. Network flow data aggregates network traffic into flows, offering a broad overview of communication patterns, which is particularly useful for detecting potential intrusions, such as unusual traffic patterns or spikes. This type of data is especially effective for monitoring large networks due to its lower resource demands [167]. Network flow information usually comes in the form of tabular data. Conversely, packet-level data enables detailed inspection of individual packets, including their headers and payloads. This allows for the identification of specific attack vectors, protocol anomalies, and complex threats that involve payload manipulation or encryption, providing a more nuanced and thorough analysis of network traffic [168]. Both data types are critical for comprehensive intrusion detection, each offering unique insights into network security.

A recent study [169] highlights the limitations of traditional ML and DL approaches in capturing context-driven similarities between network flow data and

packet-level data within NIDS. These conventional models typically focus on analyzing either network flow or packet-level data separately. The research suggests that using Graph Representation Learning (GRL) can significantly enhance NIDS performance by integrating these two types of data. GRL techniques encode the relationships between network flow and packet-level data into a graph structure, better capturing the complex and dynamic nature of network traffic and associated threats. The study proposes that graph-based data generated through GRL could be incorporated into the symbolic reasoning frameworks of NeSy AI, enriching them with deep domain knowledge. This integration is expected to improve the predictive accuracy of NeSy AI and provide detailed, granular interpretability by leveraging packet-level data insights.

In this context, we argue that graph-based data produced through GRL methods could be effectively integrated into NeSy AI's symbolic reasoning systems, thereby enhancing its domain knowledge. Additionally, a NeSy AI-powered NIDS could use both packet-level data and network flow insights, enabling the detection of real-time threats through the analysis of both payload and encrypted traffic. This fusion of packet-level and network flow data within a NeSy AI-powered NIDS could significantly improve the system's effectiveness, resilience, and interpretability.

In the field of NIDS, the use of both real-world and synthetic data is vital, with each offering distinct benefits and facing specific challenges. Real-world datasets, such as KDDCUP'99 [170], NSL-KDD [171], CICIDS-2017 [172], UNSW-NB15 [173], and ACI-IOT-2023 [174], are often generated from simulated real-world environments. These datasets are intended for the evaluation and benchmarking of NIDS, providing labeled network traffic that includes a variety of attack types, thereby enabling comprehensive testing across multiple scenarios. However, the use of real-world data is often constrained by privacy concerns, data sensitivity, and other issues, which has led to a growing reliance on synthetic data.

Synthetic data, created using methods such as GAN, Markov models, and Bayesian networks, effectively mimics real-world traffic patterns and attack behaviors. This type of data is particularly valuable for simulating rare or complex attack scenarios, which strengthens the robustness of NIDS [175]. Furthermore, tools like Deep PackGen [176] allow for the creation of adversarial network packets, providing a rigorous test of NIDS capabilities under realistic conditions. While synthetic data offers significant advantages, including the absence of privacy issues

and the ability to be widely shared, thus fostering collaboration and improving reproducibility, it may not always capture the variability and authenticity of real-world data. This disparity can impact the performance of NIDS in actual network environments. Additionally, generating high-quality synthetic data demands substantial computational resources and specialized expertise, which can be a significant barrier for researchers with limited access to such resources [175]. Consequently, both real-world and synthetic data are essential, each playing a crucial role in the development and accuracy of NIDS algorithms within cybersecurity.

17.2 A Neuro-Symbolic AI Network Intrusion Detection System

In this section, we demonstrate how the method outlined in Section 11.3 enhances the detection capabilities of a NIDS using NeSy AI both for known and unknown attacks. DNNs are employed to learn complex patterns within network data, providing a comprehensive understanding of cyberattack characteristics. By integrating symbolic logic into DNNs, we can guide the model's training process, applying penalties when the DNN fails to accurately differentiate between malicious and benign traffic. This approach enhances the model's adaptability to new attacks, addressing the limitations of traditional signature-based NIDS. Our experiments with a large-scale dataset, including novel attack scenarios, demonstrate that our NeSyAI-enhanced NIDS achieves superior attack detection accuracy compared to conventional DNN methods. Not only our system achieve high accuracy in detecting known attacks, but it also excels in identifying unknown threats, surpassing traditional NIDS in this regard. This research contributes to cybersecurity by introducing a novel approach to detecting both known and unknown network intrusions through the combination of DNNs and symbolic logic.

17.2.1 Problem Description

The problem at hand involves OSR in NIDS within the context of a multi-class network tasked with distinguishing between different types of attacks, while also performing binary classification to differentiate between benign and malicious activities. The challenge lies in enhancing the network's ability to identify and accurately classify not only known attack types but also to effectively discern benign activities from any form of attack, including previously unseen (zero-day) attacks. To address

this, we propose the introduction of a regularization constraint that compels the network to simultaneously learn to classify specific attack types and to robustly separate benign activities from malicious ones. This dual focus is expected to improve the network's overall detection capability, particularly in identifying zero-day attacks, which are typically more challenging due to their novel nature.

17.2.2 Methodology

Our approach is a NeSy AI framework, specifically the Hybrid Logic Tensor Network (Hybrid-LTN) illustrated in Sec. 11.3.

The Hybrid-LTN is designed to leverage the strengths of both neural networks and symbolic logic. While DNNs are highly effective at learning complex patterns from network data, they often lack interpretability and struggle with generalizing to novel attack types. On the other hand, symbolic logic provides a structured and interpretable way to encode knowledge but may not be flexible enough to handle the nuances of real-world network traffic. By integrating these two paradigms, Hybrid-LTN aims to improve the detection of both known and unknown attacks.

In this methodology, we incorporate symbolic logic into the training process of a DNN. This is achieved by applying penalties when the model fails to differentiate between malicious and benign network traffic, guiding the DNN towards better adaptability to new attacks. The Hybrid-LTN utilizes a Logic Tensor Network (LTN) with a hybrid loss function, which integrates the real logic of the LTN with the standard cross-entropy loss. This approach enables the model to maintain high accuracy in recognizing known attacks while also enhancing its capability to identify novel attack scenarios.

Through this neuro-symbolic integration, Hybrid-LTN creates a more robust and transparent intrusion detection system. The methodology section further details the dataset used, the mathematical formulations of our approach, and the experiments conducted to validate the effectiveness of Hybrid-LTN in cybersecurity contexts.

Below, we detail our approach by defining the domains, variables, constants, and predicates utilized in the system, as well as the axioms and loss function that govern the learning process. Additionally, we introduce the functions $\mathbf{D}$ and $\mathbf{D}_{in}$, which are used to define the domains of variables or constants and the domains of function or predicate arguments, respectively.

Domains:

items denote the examples from the dataset. These examples are composed of elements comprising 1,500 features, which represent the payload of a packet.

labels denote the class labels.

Variables:

x_b, x_a indicate positive examples of benign and attack class.

x is used to denote all the examples.

$\mathbf{D}(x_b) = \mathbf{D}(x_a) = \mathbf{D}(x)$

Constants:

l_b, l_a, where $l_b = 0$, $l_a \neq 0$ and 0 is the benign class.

$\mathbf{D}(l_b) = \mathbf{D}(l_a) = labels$.

Predicates:

$P(x,l)$ denotes the fact that item x is labelled as l

$\mathbf{D}_{in}(P) = (items, labels)$.

Axioms: We use axioms which are applicable only to the categories of benign and non-benign (attack), rather than being universally relevant across all categories:

$$\forall x_b P(x_b, l_b) \wedge \forall x_a \neg P(x_a, l_b) \tag{IV.6}$$

This axiom expresses two conditions. The first part, $\forall x_b P(x_b, l_b)$, states that for every instance x_b classified as benign, the predicate $P(x_b, l_b)$ holds true, meaning that all benign examples are correctly recognized as benign. The second part, $\forall x_a \neg P(x_a, l_b)$, indicates that for every instance x_a not classified as benign (i.e., an attack), the predicate $P(x_a, l_b)$ does not hold, ensuring that no attack examples are incorrectly classified as benign. Together, this axiom ensures that benign instances are identified as benign, and attack instances are not mistakenly identified as benign.

Grounding:

$\mathcal{G}(items) = \mathbb{R}^{1500}$; the examples from the data set are describe using 1500 features, one for each payload byte.

$\mathcal{G}(labels) = \{0,1\}^{15}$; we use one-hot encoding to represent classes, benign or 14 different types of attacks.

$\mathcal{G}(x_n) \in \mathbb{R}^{m_n \times 1500}$, is a sequence of m_n examples of class n.

$\mathcal{G}(P|\theta) : x, l \mapsto l^{\top} \cdot \text{softmax}\left(\text{MLP}_{\theta}(x)\right)$

Learning: The logical operators and connectives are approximated using the product fuzzy logic as in [50] with $p = 2$.

Our contribution lies in integrating CE loss [87] with the ***SAT loss*** to improve the network's ability to differentiate not only between benign and attack scenarios but also among specific attack categories. By prioritizing the distinction between benign and non-benign events during training, the network gains the capacity to detect previously unknown attacks.

17.2.3 Experiments

For each experiment, we trained a benchmark DNN using the same architecture and train-validation-test dataset as the Hybrid-LTN model. Specifically, we constructed a One-Dimensional Convolutional Neural Network (1D CNN) as illustrated in Figure IV.17. Each convolutional layer employs a ReLU activation function. The first dense layer also uses a ReLU activation function, while the output layer employs a softmax activation function. To contextualize the results, it is essential to distinguish between the training and testing phases.

In these experiments, we utilized the CIC-IDS2017 dataset, developed by the Canadian Institute for Cybersecurity in 2017 [177]. This dataset is divided into two parts: packet-based data in Packet Capture (PCAP) format and flow-based data in CSV format, the latter of which was derived by extracting 80 features from the PCAP files using CICFlowMeter.

CICFlowMeter is a tool designed for the generation and analysis of network traffic flows. It calculates over 80 statistical characteristics of network traffic, such as duration, packet count, byte count, and packet length. The tool supports bidirectional flows, where the first packet determines the forward and reverse directions. It allows

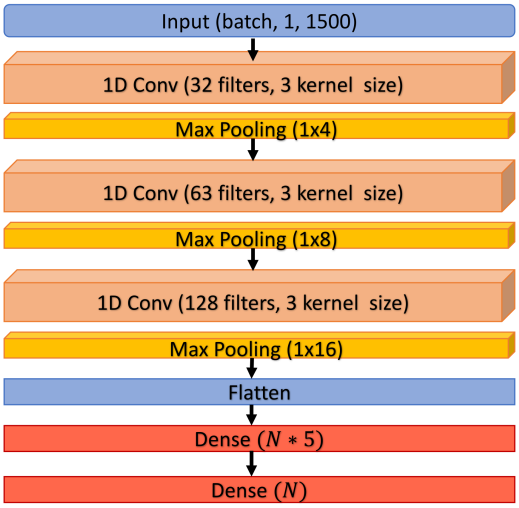


Fig. IV.17 1D CNN Model Architecture, where N represents the number of classes.

for the calculation of characteristics separately for each direction. CICFlowMeter provides the ability to select features from a predefined list, add new features, and control the flow timeout duration. The output is a CSV file that includes a row for each flow and six labeled columns: flow ID, source and destination IP addresses, source and destination ports, and protocol [178].

Both packet-based and bidirectional flow-based data were captured during simulated network traffic, encompassing both the latest attack types and benign traffic. For the purposes of our study, we utilized only the packet-based data. The dataset was collected over a five-day period of simulated traffic acquisition.

The packet-based data in CIC-IDS2017 is unlabeled, necessitating the use of the Payload-Byte tool [168] to extract and label network traffic PCAP files using the metadata provided within the dataset. This tool matches packets with flow-based labeled data instances by leveraging the features described in the PCAPs. Given the variability in packet size, Payload-Byte utilizes a maximum payload length of 1,500 bytes, with each byte converted into an integer feature ranging from 0 to 255. Following data labeling, any duplicate instances and those lacking payload data were removed.

To achieve a more balanced dataset, we under-sampled the benign samples. We excluded the examples for five attack classes from the training set, which collectively represented less than 1% of the total samples, reserving them exclusively for the test

phase as *zero-day* attacks (i.e., OOD inputs). Then, we ensured an equal number of examples for each remaining attack class by selecting a number of instances equivalent to the least represented class, FTP-Patator, resulting in 31,843 examples per class. Furthermore, by using sampling we have arbitrarily reduced the total number of benign examples from 362,108 to 200,000 to decrease complexity.

The final dataset consists of 454,744 examples, comprising 200,000 benign samples and 254,744 attack samples. Table IV.8 provides a summary of the statistics for the under-sampled dataset.

Classes	**Resampled DS**
BENIGN	**200,000**
ATTACKS	**254,744**
DoS Hulk	31,843
DDoS	31,843
DoS GoldenEye	31,843
DoS slowloris	31,843
Infiltration	31,843
DoS Slowhttptest	31,843
SSH-Patator	31,843
FTP-Patator	31,843
Zero-day attacks	**31,966**
Heartbleed	13,486
Brute Force (Web Attack)	11,754
XSS (Web Attack)	3,341
Bot	2,543
PortScan	830
Sql Injection (Web Attack)	12

Table IV.8 Number of samples for each class after under-sampling.

We divided the new dataset into 80%, 10%, 10% parts for training, validation and testing in a stratified manner. *Zero-day* attacks are used only in the test phase in order to measure the robustness of our approach.

17.2.4 Results

In this section, we compare the results of the benchmark models and the *Hybrid-LTN* models after 30 or 50 training epochs.

In our evaluation, we compared the performance of the proposed Hybrid-LTN model against two baseline LTNs. Both baseline LTNs utilize the same 1D CNN

architecture but differ in the types of classification constraints applied. The first baseline model applies a multi-class classification constraint using satisfaction loss, which aims to classify network traffic into multiple categories, including different types of attacks and benign traffic. The second baseline LTN model uses a binary classification constraint with satisfaction loss, focusing on distinguishing between attack and benign traffic only.

Additionally, we trained a standalone 1D CNN using only cross-entropy loss as a benchmark to evaluate the improvements gained by incorporating symbolic reasoning into the LTN models. By comparing these baselines with the Hybrid-LTN, which integrates both neural learning and symbolic reasoning with a hybrid loss function, we aim to highlight the advancements in detection capabilities, particularly in identifying known and zero-day attacks within network intrusion detection systems.

It is important to highlight that our comparative analysis was based on a straightforward approach, utilizing a simple 1D CNN as the underlying network architecture. This intentional simplification aligns with the primary objective of our study, which is not to achieve peak accuracy but to illustrate the potential for enhanced robustness and improved detection of novel attacks through the integration of logic.

By employing a methodologically simple approach, we ensure that any observed improvements can be directly attributed to the incorporation of logic, thereby providing a clear demonstration of our innovative contributions to enhancing system resilience against evolving cyber threats.

Table IV.9 presents the results. For each experiment, we evaluated the accuracy of both the *Hybrid-LTN* and benchmark models in the following ways:

- Multi-class accuracy on the test set, which includes only the 9 known classes (benign and 8 attack classes).
- Binary accuracy on the test set, focused on the 9 known classes (benign vs. attacks).
- Multi-class accuracy on a test set that includes all 15 classes, both known and unknown (benign and 14 attack classes).
- Binary accuracy on the test set comprising all 15 classes, including both known and unknown (benign vs. attacks).

- Binary accuracy on the test set containing only the 6 unknown attacks (benign vs. attacks).

The F1-score was calculated only for binary classification tasks.

The bold values in Table IV.9 highlight the performance of each model specifically on the "Binary 6 unknown classes" task. This represents the detection of previously unseen or zero-day attacks, which is particularly challenging. Figure IV.18

Accuracy				
30 epochs				
Test Set	***Hybrid-LTN***	**1D CNN**	**LTN multi-class**	**LTN binary**
Multi-class 9 known classes	81.04%	80.88%	80.13%	-
Binary 9 known classes	99.53%	99.44%	99.06%	99.45%
Multi-class 15 classes	67.48%	67.35%	66.73%	-
Binary 15 classes	92.20%	90.68%	91.69%	90.44%
Binary 6 unknown classes	**55.70%**	**47.13%**	**54.98%**	**45.56%**
50 epochs				
Test Set	***Hybrid-LTN***	**1D CNN**	**LTN multi-class**	**LTN binary**
Multi-class 9 known classes	81.08%	80.99%	80.41%	-
Binary 9 known classes	99.57%	99.42%	99.46%	99.42%
Multi-class 15 classes	67.52%	67.45%	66.96%	-
Binary 15 classes	93.03%	90.88%	92.19%	89.40%
Binary 6 unknown classes	**60.47%**	**48.34%**	**56.06%**	**39.40%**
F1-Score				
30 epochs				
Test Set	***Hybrid-LTN***	**1D CNN**	**LTN multi-class**	**LTN binary**
Binary 9 known classes	99.58%	99.50%	99.16%	99.51%
Binary 15 classes	93.47%	92.12%	93.00%	91.88%
Binary 6 unknown classes	**71.55%**	**64.07%**	**70.95%**	**62.60%**
50 epochs				
Test Set	***Hybrid-LTN***	**1D CNN**	**LTN multi-class**	**LTN binary**
Binary 9 known classes	99.62%	99.49%	99.52%	99.49%
Binary 15 classes	94.20%	90.88%	93.47%	90.93%
Binary 6 unknown classes	**75.37%**	**65.18%**	**71.80%**	**56.61%**

Table IV.9 Accuracy and F1-Score from 30 and 50 epochs of training with the *Adamax* optimizer for both the *Hybrid-LTN* and benchmark models. Bold values indicate the results for the detection of unknown attacks, where a higher score denotes a better performance in distinguishing these novel threats from benign traffic.

presents the confusion matrices for both the benchmark models and our NeSy AI-based NIDS. The matrices depict performance across the 9 known classes (8 attack classes and 1 benign class) and the unknown attacks for each system tested: the

vanilla 1D CNN, basic multiclass LTN (M-LTN), basic binary LTN (B-LTN), and our *Hybrid-LTN* model.

Figure IV.19 shows the training and validation loss curves over the epochs for both our system and the baseline models. Given the unbalanced nature of the dataset, even after resampling efforts, we chose to display both accuracy and F1-score, as the latter provides a more nuanced evaluation of system performance, especially when dealing with imbalanced classes.

Notably, while there is a slight difference in multi-class accuracy between the two networks on the known classes, our method demonstrates a significant improvement in binary accuracy for unknown attacks, outperforming the reference network by several percentage points.

When comparing *Hybrid-LTN* with the baseline models, it is evident that the *Hybrid-LTN* model excels in detecting unknown attacks. This advantage can be attributed to the integration of logic during the training phase, which significantly enhances the model's capacity to generalize. The detection of unknown threats is crucial, and the model's ability to discern logical patterns and relationships among different classes contributes to its superior performance.

The higher binary accuracy for unknown attacks highlights the *Hybrid-LTN* model's effectiveness in distinguishing between benign traffic and previously unseen threats. This outcome is likely due to the model's capability to learn complex logic patterns indicative of anomalous activity. The increased robustness observed in the *Hybrid-LTN* model is particularly relevant in real-world scenarios, where identifying previously unknown threats is critical for maintaining system security.

Our *Hybrid-LTN* approach demonstrates clear advantages over a Binary LTN-based system, underscoring its relevance and innovation in tackling the complexity of cyber threats. Unlike the straightforward application of Binary LTN, the integration of logic as a regularization factor within *Hybrid-LTN* leads to superior robustness. While logic alone enhances resilience compared to a traditional 1D CNN, the fusion of logic with NNs results in even greater improvements. This nuanced integration is key to strengthening intrusion detection systems against evolving threats.

These results underscore the tangible benefits that NeSy AI-based NIDS can bring to cybersecurity. The detection of previously unseen threats remains one of the most pressing challenges in the field, and our system demonstrates how incorporating

logic during the training phase can enhance robustness. Indeed, while both the *Hybrid-LTN* and 1D CNN models perform similarly in detecting known attacks, our NeSy AI-based NIDS significantly outperforms the purely NN in detecting unknown attacks. When evaluating only unseen attacks, *Hybrid-LTN* achieves an accuracy improvement of 8-12% and an F1-score improvement of 7-1% compared to traditional DL techniques.

CMs serve as a valuable tool for analyzing the performance of intrusion detection models. As shown in Figure IV.18, logic-based approaches, such as LTNs, exhibit fewer false negatives than traditional NNs like the 1D CNN. Among the LTN approaches, *Hybrid-LTN* maintains a significantly lower number of false negatives, which is crucial in cybersecurity, where accurately identifying attacks is vital to avoid serious consequences. Moreover, *Hybrid-LTN* demonstrates a lower number of false positives compared to other LTN models, indicating its ability to minimize the misclassification of benign traffic as attacks, thereby achieving higher overall accuracy. The *Hybrid-LTN* model's exceptional performance in minimizing false negatives for unknown attacks further confirms its superior capability in detecting even previously unseen threats.

Analyzing the confusion matrices provides a detailed understanding of each model's performance, with logic-based approaches, particularly *Hybrid-LTN*, emerging as the most effective in reducing false negatives and accurately handling false positives. These results suggest enhanced reliability and accuracy in identifying attacks, including unknown threats.

Monitoring the difference between training and validation accuracy during the learning process can reveal critical insights into the robustness of the *Hybrid-LTN* model. A smaller gap between these two metrics suggests reduced overfitting, which translates into significant benefits for cybersecurity.

Figure IV.19 illustrates that, for LTN models, the difference between training and validation accuracy is smaller compared to the purely neural approach. This is a key indicator, as a large discrepancy could suggest that the model is overly focused on specific data, leading to poor generalization.

A smaller difference between training and validation performance implies lower overfitting, which is essential to ensure the model's ability to generalize to new scenarios, thereby enhancing its robustness.

The hybrid nature of the model, which combines logic with DL techniques, contributes to reducing overfitting. The inclusion of logic introduces a semantic understanding component that helps the model more accurately capture the underlying patterns in the training data without succumbing to overspecialization.

These findings pave the way for the development of NeSy AI-based systems that can improve interpretability and robustness in cybersecurity contexts. By mitigating overfitting, these models become more reliable and better equipped to adapt to evolving threats without compromising accuracy.

17.3 Open Set Recogniction in Network Intrusion Detection System

OSR poses a significant challenge in distinguishing between known and unknown categories, especially when labeling is costly or incomplete. This problem is particularly critical in applications like NIDS, where OSR is crucial for detecting novel, previously unseen attacks.

In OSR, one of the key challenges is effectively distinguishing between known and unknown classes of network intrusions. A crucial component of our methodology is the use of Deep Embedding for Clustering, which incorporates an autoencoder within its architecture. The autoencoder in DEC is employed to learn a compact latent representation of the input data. By doing so, it helps to enhance the clustering process, allowing the model to better separate known classes from novel, previously unseen attacks. This latent space representation, learned through the autoencoder, is instrumental in the subsequent steps of clustering and classification, aiding in the identification of zero-day attacks and improving the overall robustness of the Network Intrusion Detection System. The detailed functioning of the autoencoder within DEC is explained more in details in the background section.

In the following sections, we demonstrate the application of TEX-DEC, described in Section 12, within this domain.

17.3.1 Problem Description

In the context of NIDS, OSR presents a significant challenge. The goal of an OSR-based NIDS is to accurately classify network traffic into known and previously

encountered attack types while also identifying and flagging novel, unseen threats (often referred to as zero-day attacks). Unlike traditional classification tasks, where the objective is to assign each instance to a predefined class, OSR requires the system to handle cases where an input does not belong to any of the known classes. This necessitates the ability to distinguish between known categories of network behavior (both benign and malicious) and to detect new, previously unclassified attack patterns.

17.3.2 Experiments

The system is applied to the task of OSR in network intrusion detection using the CIC-IDS2017 dataset [177]. As the field of NIDS is continually evolving, with the detection of new types of attacks being crucial, as mentioned above, we segmented the dataset into three categories: *Known*, *Novelty 1*, and *Novelty 2*.

The dataset contains 14 different types of attack and 1 benign class. The dataset was split as shown in Table IV.10. Three attacks were chosen as novel as in [179], where the authors considered each type of attack in turn as previously unknown, and identified these three attacks as those that showed the greatest performance degradation in terms of detection. Therefore, we also chose to use the same attacks as novelties.

We use also the UNSW-NB15 dataset [173] as a test set. This is a NIDS dataset developed to identify normal and attack network traffic. The raw network packets were generated by the Australian Centre for Information Security (ACCS). This dataset was preprocessed in the same manner as the previous one. The Payload-Byte tool [168] was applied to it. This dataset is used as a covariate to test the robustness of our approach, even with datasets from different network configurations. The UNSW dataset is used only during the testing phase and is labeled as *Novelty 2*.

17.3.3 Results

We evaluated our approach on various datasets. The DEC model was trained with different loss configurations, as detailed in Section 17.3.4. Additionally, we performed a grid search on XGBoost hyperparameters, including the number of components and maximum depth.

Class	**# sample**	**Subset**
Benign	3.328.591	Known
DoS Hulk	2.219.061	Known
DoS Slowhttptest	9.778	Known
Heartbleed	41.283	Known
Brute Force (Web Attack)	28.920	Known
Sql Injection (Web Attack)	45	Known
XSS (Web Attack)	6.767	Known
Bot	5.143	Known
PortScan	946	Known
DoS GoldenEye	34.293	Novelty 1
DoS slowloris	20.877	Novelty 1
DDoS	618.544	Novelty 1
SSH-Patator	181.147	Novelty 1 or 2
FTP-Patator	110.636	Novelty 1 or 2
Infiltration	41.725	Novelty 1 or 2

Table IV.10 The dataset split into *Known, Novelty 1 & 2*

For the CIC-IDS2017 dataset, we compared our results with those reported by [179] and [180], focusing on the AUROC for the *Novelty 2* classes. Additionally, we compared the AUROC scores for the UNSW-NB15 dataset with [179]. In this comparison, we encountered a covariate shift rather than a semantic shift, as the classes were consistent across datasets but exhibited different distributions. The results, presented in Table IV.11, demonstrate a significant improvement of performance relative to previous approaches. For instance, our method achieved an AUROC of 0.9843 for detecting the previously unknown Infiltration attack, surpassing the performance of existing methods. Similarly, our approach yielded an AUROC score of 0.9939 for the previously unknown SSH-Patator attack, slightly outperforming [179] (0.9921) and significantly exceeding [180] (0.6787). Although our AUROC of 0.9950 for the previously unknown FTP-Patator attack is slightly lower than [179] (0.9957), it largely outperforms [180] (0.7955). Furthermore, when evaluating the UNSW-NB15 dataset, which is derived from a different distribution with respect to CIC-IDS2017, our AUROC of 0.9939 surpassed the result reported by [179] (0.9583). This result highlights the robustness and generalizability of our methodology in detecting novel attacks across different datasets. While the initial three attacks were novel within the same dataset, our method demonstrates considerable adaptability and detection capability even with the entirely different

UNSW-NB15 dataset. This ability to generalize instills confidence in the efficacy and utility of our approach across a range of real-world scenarios.

	AUROC		
Novelty Type	Matejek et al. [179]	Zavrak et al. [180]	TEX-DEC
FTP-Patator	**0.9957**	0.7955	0.9950
Infiltration	0.9742	0.8965	**0.9843**
SSH-Patator	0.9921	0.6787	**0.9939**
UNSW-NB15	0.9583	-	**0.9939**

Table IV.11 The AUROC results of the methods for specified previously unknown attacks

17.3.4 Ablation Study

Our approach uses a loss function composed of three components: KL divergence (L_{kld}), contrastive loss ($L_{contrastive}$), and classification loss (L_{CE}). In this section, we explore different configurations of this loss function to identify the most effective combination. For this analysis, we consistently partitioned the dataset and trained a single autoencoder across all configurations to ensure a fair comparison, providing a shared starting point. The loss function is defined in Equation III.14 that we repeat here for readability.

$$\mathscr{L} = \alpha \cdot \mathscr{L}_{kld} + \beta \cdot \mathscr{L}_{contrastive} + \omega \cdot L_{\text{CE}}$$

The weights α, β, and ω were adjusted by evaluating all possible combinations of 0 and 1, effectively *disabling* specific components of the loss function.

The results are summarized in Table IV.12, where each cell contains the AUROC score for a given configuration, along with the deviation from the optimal setup (highlighted in bold). These experiments were conducted using the CICIDS2017 dataset, focusing on different attack types such as SSH-Patator, Infiltration, FTP-Patator, and UNSW-NB15 to evaluate the system's capability in novelty detection.

The results reveal clear trends across the various configurations. Notably, configurations where all loss components are active ($\alpha = \beta = \omega = 1$) consistently achieve AUROC values near optimal, highlighting the synergistic effect of each component in enhancing overall model performance.

α	β	ω	**SSH-Patator**	**Infiltration**	**FTP-Patator**	**UNSW-NB15**	**Average**
1	1	1	99.50	98.43	99.39	**99.39**	**99.28**
1	0	0	99.66	84.54	99.03	98.88	95.53
0	1	0	99.41	98.29	99.52	99.28	99.13
0	0	1	99.57	**98.61**	**99.56**	99.35	99.27
1	1	0	**99.69**	84.52	99.03	98.85	95.52
1	0	1	99.68	84.40	99.05	98.88	95.51
0	1	1	99.51	98.38	99.36	**99.39**	99.16

Table IV.12 The table displays the AUROC for the different test *Novelty 2*: SSH-Patator, Infiltration, FTP-Patator, and UNSW-NB15.

In contrast, configurations where only the KL divergence component is active while one or more components are set to zero ($\alpha = 1$ and β or $\omega = 0$) exhibit lower performance, particularly in the case of *Infiltration*. This underscores the critical importance of each component in enabling effective novelty detection.

Intermediate configurations, where specific loss components are selectively enabled, provide insights into their individual contributions. They demonstrate how the inclusion of classification and contrastive loss leads to improvements in novelty detection. A systematic examination of these configurations offers valuable understanding of the interaction between the components and their combined influence on performance. These findings are crucial for guiding the refinement and optimization of novelty detection systems, contributing to advancements in the field.

17.4 Related Work

Recent results in ML have significantly advanced the field of OSR. OSR methods are crucial in scenarios where the set of possible classes is open-ended or evolving, and these methods are typically categorized into discriminative and generative approaches [181]. Discriminative models, as discussed by Scheirer et al. [182], Hassen and Chan [183], and Bendale et al. [184], use probability-based or learning-based techniques to differentiate between known and unknown classes. In contrast, generative models, such as those developed by Neal et al. [185] and Ge et al. [186], employ generative techniques to identify OSR samples.

Scheirer et al. [182] proposed the Compact Abating Probability (CAP) model, which adjusts the probability of class membership as samples deviate from the

training data and approach open space, demonstrating effectiveness in OSR contexts. Building on this, Bendale et al. [184] introduced OpenMax, which integrates Extreme Value Theory (EVT) to construct a CAP model for each class, improving robustness by rejecting unknown inputs through thresholding. Although OpenMax is not primarily designed to handle adversarial inputs, it shows greater resilience compared to conventional softmax models.

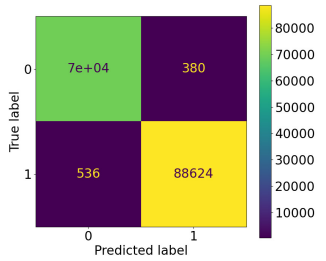
Hassen and Chan [183] investigated intermediate representations to create spatial distinctions where samples from the same class cluster together, while samples from different classes are distinctly separated. This method allows for the identification of unknown examples based on Euclidean distance and predefined thresholds. Neal et al. [185] employed GAN to produce synthetic examples that mimic the training set but do not belong to any known category, using these samples to train OSR models. Ge et al. [186] developed Generative OpenMax (G-OpenMax), extending the capabilities of OpenMax to enhance the detection of unknown samples.

In the domain of NIDS, which primarily utilize known datasets for attack classification [187–189], there are significant challenges in detecting previously unknown attacks. Traditional anomaly-based methods, which rely on deviations from normative behavior and typically require network flow data, often need additional information [190–192, 180].

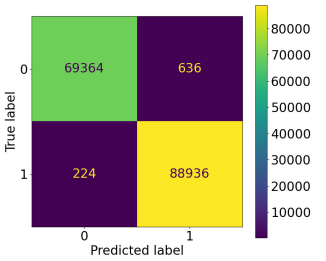
Technological advancements are increasingly undermining the effectiveness of traditional network security measures, particularly those that rely on signature detection. While these systems perform well in identifying known threats, they struggle significantly with novel, unknown attacks. AI is being explored as a potential solution to these limitations, as it can process large volumes of data in real-time, facilitating rapid threat detection and response. However, AI also introduces challenges, such as the need for extensive annotated datasets for training and the opacity of certain DL models. Additionally, AI-based systems are vulnerable to adversarial attacks [162].

In the realm of cybersecurity, NeSy AI is emerging as a promising approach. Several NeSy AI-based NIDS have been developed, combining symbolic reasoning with NN techniques [193–195]. These models leverage feedback from system operators to automatically fine-tune and enhance their accuracy. For instance, in [195], a hybrid system that integrates k-means clustering and RF with Graph Neural Network (GNN) and Long Short-Term Memory (LSTM) networks demonstrated an improved true positive rate in detecting anomalies within network traffic. The authors used a GNN

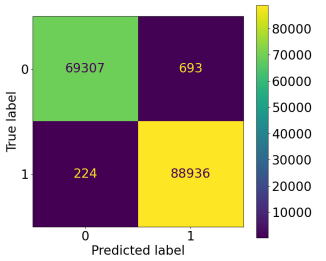
to learn typical network behavior and identify deviations. This system was further combined with GNNExplainer [196] and an ontology to reduce false positives and improve explainability.



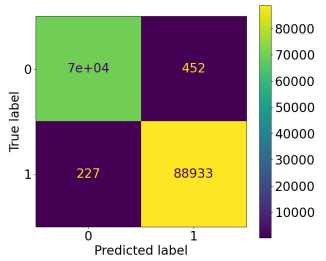
(a) 1D CNN - 9 known classes



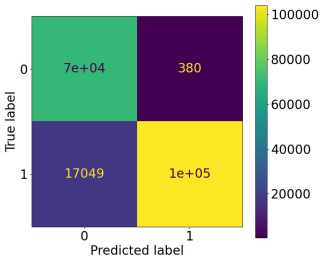
(b) M-LTN - 9 known classes



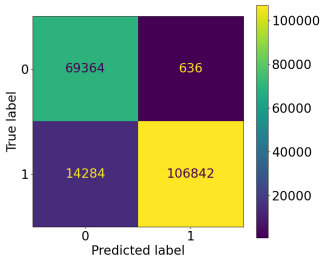
(c) B-LTN - 9 known classes



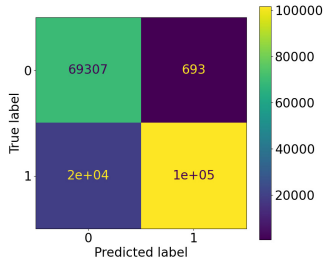
(d) *Hybrid-LTN* - 9 known classes



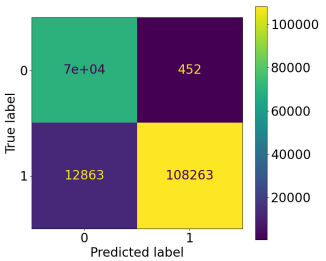
(e) 1D CNN - all 15 classes



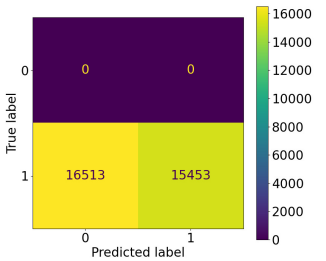
(f) M-LTN - all 15 classes



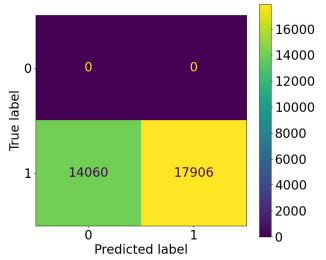
(g) B-LTN - all 15 classes



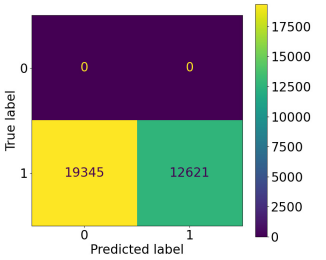
(h) *Hybrid-LTN* - all 15 classes



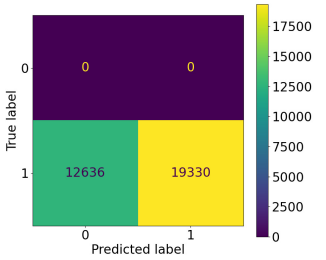
(i) 1D CNN - 6 unknown classes



(j) M-LTN - 6 unknown classes

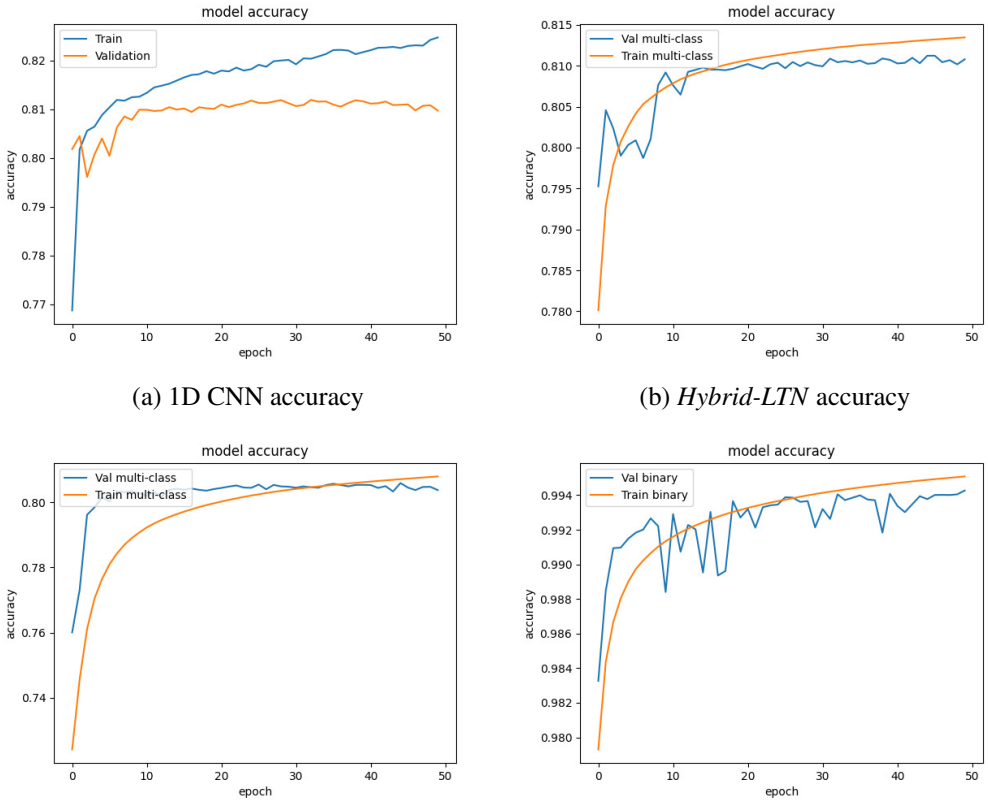


(k) B-LTN - 6 unknown classes



(l) *Hybrid-LTN* - 6 unknown

Fig. IV.18 Confusion matrix of the experiment with 50 epochs.



(a) 1D CNN accuracy

(b) *Hybrid-LTN* accuracy

(c) M-LTN accuracy

(d) B-LTN accuracy

Fig. IV.19 Validation and training accuracy of the experiment with 50 epochs.

Chapter V

Conclusions and Future Work

18 Conclusions

This thesis has explored the integration of neuro-symbolic systems as an innovative approach to improving transparency, interpretability, and effectiveness in artificial intelligence systems. The primary goal was to develop and apply neuro-symbolic models in real-world contexts, particularly in critical domains such as healthcare and cybersecurity, where the ability to justify decisions is essential.

The results obtained demonstrate that the integration of symbolic techniques with deep learning can lead to AI systems that not only achieve high performance in terms of accuracy but also provide interpretable insights into the decisions made. This was particularly effective in the medical diagnosis cases, where the combination of probabilistic logic and neural networks improved the model's ability to deliver understandable predictions. For instance, the use of probabilistic logic programming enabled more reliable forecasts in contexts such as COVID-19 patient condition prediction, offering easily comprehensible explanations for medical experts.

Another significant achievement was in the field of network intrusion detection. In this context, neuro-symbolic models not only identified known attacks but also detected previously unknown ones, thanks to their ability to recognize anomalies relative to expected behavior. The integration of deep learning techniques with symbolic reasoning provided a solid foundation for detecting emerging threats, enhancing overall network security.

However, it is important to acknowledge certain limitations and weaknesses that emerged during the research. First, the computational complexity of neuro-symbolic models can be high, especially when dealing with large datasets or particularly complex scenarios. While the systems developed showed good scalability in controlled contexts, applying such models in production environments would require further optimizations to reduce computational time and resource requirements. Additionally, the integration of deep learning and symbolic techniques is not always straightforward, particularly when balancing learning capacity with interpretability, which can complicate the design of models that are both powerful and explainable.

Another challenge concerns the applicability of neuro-symbolic models to domains beyond those explored in this thesis. While the results are promising for medical diagnosis and anomaly detection in cybersecurity, the effectiveness of these systems in other fields, such as robotics or autonomous control, remains to be demonstrated. The adaptability of the models to new or unexpected scenarios may require more flexibility in the learning and reasoning methods employed.

Finally, the importance of standardization in the neuro-symbolic field cannot be overstated. One of the contributions of this thesis was the development of benchmarks for evaluating the performance of neuro-symbolic systems, but the field still suffers from a lack of shared standards for objective comparison between different approaches. The introduction of these benchmarks represents a step forward, but further work is needed to develop more comprehensive datasets and evaluation metrics that will support the uniform progress of research in this domain.

Despite these limitations, this thesis has demonstrated that neuro-symbolic integration offers a promising path to address many of the current challenges in AI. The models developed have shown clear improvements in interpretability compared to purely neural methods, making them more suitable for contexts where human understanding of decisions is crucial. Moreover, the ability to combine existing symbolic knowledge with data-driven learning has enhanced the models' effectiveness in contexts where data is scarce or highly structured.

This thesis represents a significant contribution towards the development of more transparent and reliable AI systems, laying the groundwork for future developments that could revolutionize how AI is employed in critical sectors.

18.1 Future Work

The research presented in this thesis opens up many directions for future development, both in terms of advancing the theoretical foundations of neuro-symbolic systems and expanding their practical applications. Despite the promising results obtained, several areas require further investigation to fully realize the potential of neuro-symbolic integration in artificial intelligence.

A key area for future work is the reduction of the computational cost in neuro-symbolic models. While the integration of symbolic reasoning with deep learning has proven to enhance interpretability, it also introduces complexity, particularly in terms of computational cost. As the scale of datasets and the complexity of tasks increase, it becomes essential to design more efficient algorithms capable of handling large-scale applications without compromising the benefits of transparency and explainability. This includes improving the scalability of the models to process larger datasets and more complex scenarios while maintaining manageable resource requirements. Future research could explore methods to streamline the learning process, such as the development of more efficient optimization algorithms or the application of hardware acceleration techniques like parallel computing and specialized architectures (e.g., GPUs and TPUs) tailored to the specific needs of neuro-symbolic AI.

Another promising avenue for future research is the exploration of neuro-symbolic models in domains beyond those examined in this thesis. While the models developed here have demonstrated success in areas like medical diagnosis and network intrusion detection, the applicability of these systems to other domains remains largely unexplored. Fields such as robotics, autonomous systems, and AI-driven decision-making in social contexts could benefit from the integration of symbolic reasoning with deep learning. These domains often involve complex decision-making processes where interpretability is crucial, yet the learning systems must also be adaptable to dynamic, real-time environments. Testing the adaptability of neuro-symbolic models in these new contexts will be essential for assessing their flexibility and effectiveness in real-world applications. In particular, the ability of these models to generalize across tasks, to handle unexpected situations, and to leverage symbolic knowledge to enhance decision-making in uncertain environments are areas ripe for further exploration.

Furthermore, future work should continue to focus on improving the benchmarks used to evaluate neuro-symbolic systems. One of the challenges encountered during this research was the lack of standardized benchmarks for comparing the performance of different neuro-symbolic models. The development of new datasets and evaluation metrics is crucial for ensuring that future research in this field progresses in a consistent and comparable manner. These benchmarks should not only measure the accuracy and efficiency of the models but also assess their explainability, which is a core advantage of neuro-symbolic integration. By refining the criteria for interpretability and creating more comprehensive datasets, future research can better capture the nuances of how these models function in practice, ensuring that they are evaluated in a way that reflects both their computational power and their capacity to provide clear, understandable outputs.

There is significant potential in enhancing the integration between symbolic reasoning and learning within neuro-symbolic frameworks. As current models tend to treat the symbolic and neural components as somewhat separate entities, future research could explore more tightly coupled architectures that allow for a more fluid exchange of information between these two components. This could involve developing models where symbolic rules dynamically influence neural network learning, allowing the system to adjust its decision-making process in response to symbolic constraints or new knowledge. Similarly, research could investigate how neural networks can be used to refine and improve symbolic reasoning, potentially allowing for symbolic rules to be learned or adjusted based on the data-driven insights gained through deep learning.

In conclusion, while this thesis represents a step forward in the development of neuro-symbolic AI, it also highlights several areas where further research is necessary. Optimizing computational efficiency, expanding the applicability of these models to new domains, refining benchmarks, and enhancing the interaction between symbolic and neural components are all critical directions for future work. By addressing these challenges, future research can help push the boundaries of what neuro-symbolic systems can achieve, paving the way for more interpretable, efficient, and adaptable AI solutions across a wide range of applications.

References

[1] J Matthew Helm, Andrew M Swiergosz, Heather S Haeberle, Jaret M Karnuta, Jonathan L Schaffer, Viktor E Krebs, Andrew I Spitzer, and Prem N Ramkumar. Machine learning and artificial intelligence: definitions, applications, and future directions. *Current reviews in musculoskeletal medicine*, 13:69–76, 2020.

[2] Bikram Pratim Bhuyan, Amar Ramdane-Cherif, Ravi Tomar, and TP Singh. Neuro-symbolic artificial intelligence: a survey. *Neural Computing and Applications*, pages 1–36, 2024.

[3] Lalith Kumar Shiyam Sundar, Otto Muzik, Irène Buvat, Luc Bidaut, and Thomas Beyer. Potentials and caveats of ai in hybrid imaging. *Methods*, 188:4–19, 2021.

[4] Tian Kang, Ali Turfah, Jaehyun Kim, Adler Perotte, and Chunhua Weng. A neuro-symbolic method for understanding free-text medical evidence. *Journal of the American Medical Informatics Association*, 28(8):1703–1711, 2021.

[5] Muhammad Jaleed Khan and Edward Curry. Neuro-symbolic visual reasoning for multimedia event processing: Overview, prospects and challenges. In *CIKM (Workshops)*, 2020.

[6] Richard Evans, Matko Bošnjak, Lars Buesing, Kevin Ellis, David Pfau, Pushmeet Kohli, and Marek Sergot. Making sense of raw input. *Artificial Intelligence*, 299:103521, 2021.

[7] Pavan Kapanipathi, Ibrahim Abdelaziz, Srinivas Ravishankar, Salim Roukos, Alexander Gray, Ramon Astudillo, Maria Chang, Cristina Cornelio, Saswati Dana, Achille Fokoue, et al. Leveraging abstract meaning representation for knowledge base question answering. *arXiv preprint arXiv:2012.01707*, 2020.

[8] Paolo Coraggio and Massimo De Gregorio. A neurosymbolic hybrid approach for landmark recognition and robot localization. In *International Symposium on Brain, Vision, and Artificial Intelligence*, pages 566–575. Springer, 2007.

[9] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1:81–106, 1986.

[10] L. Breiman, J. Friedman, C.J. Stone, and R.A. Olshen. *Classification and Regression Trees*. Taylor & Francis, 1984.

[11] Kevin P. Murphy. *Probabilistic Machine Learning: An introduction*. MIT Press, 2022.

[12] R. A. Fisher. Iris. UCI Machine Learning Repository, 1988. DOI: https://doi.org/10.24432/C56C76.

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

[14] Leo Breiman. Bagging predictors. *Machine learning*, 24:123–140, 1996.

[15] Leo Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

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

[17] Yoav Freund, Robert E Schapire, et al. Experiments with a new boosting algorithm. In *icml*, volume 96, pages 148–156. Citeseer, 1996.

[18] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[19] Abiodun M Ikotun, Absalom E Ezugwu, Laith Abualigah, Belal Abuhaija, and Jia Heming. K-means clustering algorithms: A comprehensive review, variants analysis, and advances in the era of big data. *Information Sciences*, 622:178–210, 2023.

[20] James C Bezdek, Robert Ehrlich, and William Full. Fcm: The fuzzy c-means clustering algorithm. *Computers & geosciences*, 10(2-3):191–203, 1984.

[21] Robin Sibson. Slink: an optimally efficient algorithm for the single-link cluster method. *The computer journal*, 16(1):30–34, 1973.

[22] Takio Kurita. An efficient agglomerative clustering algorithm for region growing. In *Proc. of MVA'94 IAPR Workshop on Machine Vision Applications*, pages 210–213, 1994.

[23] Taisuke Sato. A statistical learning method for logic programs with distribution semantics. 1995.

[24] Fabrizio Riguzzi. *Foundations of Probabilistic Logic Programming: Languages, semantics, inference and learning*. River Publishers, Denmark, 2022.

[25] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. Problog: A probabilistic prolog and its application in link discovery. In *IJCAI 2007, the 20th international joint conference on artificial intelligence*, pages 2462–2467. IJCAI-INT JOINT CONF ARTIF INTELL, 2007.

[26] 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.

[27] Chris Mungall. Experiences using logic programming in bioinformatics. In *International Conference on Logic Programming*, pages 1–21. Springer, 2009.

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

[29] Bernd Gutmann, Ingo Thon, and Luc De Raedt. Learning the parameters of probabilistic logic programs from interpretations. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2011, Athens, Greece, September 5-9, 2011. Proceedings, Part I 11*, pages 581–596. Springer, 2011.

[30] Arnaud Nguembang Fadja, Fabrizio Riguzzi, and Evelina Lamma. Learning hierarchical probabilistic logic programs. *Machine Learning*, 110(7):1637–1693, 2021.

[31] Ian Sinclair. Chapter 10 - gating and logic circuits. In Ian Sinclair, editor, *Electronics Simplified (Third Edition)*, pages 185–201. Newnes, Oxford, third edition edition, 2011.

[32] Elena Bellodi and Fabrizio Riguzzi. Structure learning of probabilistic logic programs by searching the clause space. *Theory and Practice of Logic Programming*, 15(2):169–212, 2015.

[33] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Upper Saddle River, NJ, USA, 3rd edition, 2010.

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

[35] Sonali B Maind, Priyanka Wankar, et al. Research paper on basic of artificial neural network. *International Journal on Recent and Innovation Trends in Computing and Communication*, 2(1):96–100, 2014.

[36] Marvin Minsky and Seymour Papert. An introduction to computational geometry. *Cambridge tiass., HIT*, 479(480):104, 1969.

[37] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[38] Alex Sherstinsky. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404:132306, 2020.

[39] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[40] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units. *arXiv preprint arXiv:1511.07289*, 3, 2015.

[41] 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.

[42] Junyuan Xie, Ross Girshick, and Ali Farhadi. Unsupervised deep embedding for clustering analysis. In *International conference on machine learning*, pages 478–487. PMLR, 2016.

[43] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[44] 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.

[45] Md Kamruzzaman Sarker, Lu Zhou, Aaron Eberhart, and Pascal Hitzler. Neuro-symbolic artificial intelligence. *AI Commun.*, 34:197–209, 2021.

[46] Rabinandan Kishor. Neuro-symbolic ai: Bringing a new era of machine learning. *International Journal of Research Publication and Reviews*, 2022.

[47] Antoine Bordes, Jason Weston, Ronan Collobert, and Yoshua Bengio. Learning structured embeddings of knowledge bases. In *Proceedings of the AAAI conference on artificial intelligence*, volume 25, pages 301–306, 2011.

[48] Ilya Sutskever and Geoffrey E Hinton. Using matrices to model symbolic relationship. *Advances in neural information processing systems*, 21, 2008.

[49] Adam Santoro, David Raposo, David G Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. A simple neural network module for relational reasoning. *Advances in neural information processing systems*, 30, 2017.

[50] Luciano Serafini and Artur d'Avila Garcez. Logic tensor networks: Deep learning and logical reasoning from data and knowledge. In *International Joint Conference on Neural Networks (IJCNN)*, pages 2235–2242. IEEE, 2016.

[51] Samy Badreddine, Artur d'Avila Garcez, Luciano Serafini, and Michael Spranger. Logic tensor networks. *Artificial Intelligence*, 303:103649, 2022.

[52] 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.

[53] 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.

[54] Robin Manhaeve, Sebastijan Dumančić, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Neural probabilistic logic programming in deepproblog. *Artificial Intelligence*, 298:103504, 2021.

[55] Thomas Winters, Giuseppe Marra, Robin Manhaeve, and Luc De Raedt. Deepstochlog: Neural stochastic logic programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 10090–10100, 2022.

[56] Arnaud Nguembang Fadja, Michele Fraccaroli, Alice Bizzarri, Giulia Mazzuchelli, and Evelina Lamma. Neural-symbolic ensemble learning for early-stage prediction of critical state of covid-19 patients. *Medical & Biological Engineering & Computing*, 60(12):3461–3474, 2022.

[57] Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B. Tenenbaum, and Jiajun Wu. The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. 2019.

[58] Zachary Susskind, Bryce Arden, Lizy K. John, Patrick Stockton, and Eugene B. John. Neuro-symbolic ai: An emerging class of ai workloads and their characterization. 9 2021.

[59] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C. Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. 12 2016.

[60] Kexin Yi, Chuang Gan, Yunzhu Li, Pushmeet Kohli, Jiajun Wu, Antonio Torralba, and Joshua B. Tenenbaum. CLEVRER: collision events for video representation and reasoning. *CoRR*, abs/1910.01442, 2019.

[61] Yunzhu Li, Jiajun Wu, Jun-Yan Zhu, Joshua B. Tenenbaum, Antonio Torralba, and Russ Tedrake. Propagation networks for model-based control under partial observation. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 1205–1211, 2019.

[62] Daiki Kimura, Masaki Ono, Subhajit Chaudhury, Ryosuke Kohita, Akifumi Wachi, Don Joven Agravante, Michiaki Tatsubori, Asim Munawar, and Alexander G. Gray. Neuro-symbolic reinforcement learning with first-order logic. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3525–3536, 2021.

[63] Greg Anderson, Abhinav Verma, Isil Dillig, and Swarat Chaudhuri. Neurosymbolic reinforcement learning with formally verified exploration. 2020.

[64] Guy Hadash, Einat Kermany, Boaz Carmeli, Ofer Lavi, George Kour, and Alon Jacovi. Estimate and replace: A novel approach to integrating deep neural networks with existing applications. *arXiv preprint arXiv:1804.09028*, 2018.

[65] Jacob Devlin, Jonathan Uesato, Surya Bhupatiraju, Rishabh Singh, and Abdelrahman. Robustfill: Neural program learning under noisy i/o. pages 990–998, 2017.

[66] Chenxi Yang and Swarat Chaudhuri. Safe neurosymbolic learning with differentiable symbolic execution. 2022.

[67] Jennifer J Sun, Megan Tjandrasuwita, Atharva Sehgal, Armando Solar-Lezama, Swarat Chaudhuri, Yisong Yue, and Omar Costilla-Reyes. Neurosymbolic programming for science. *arXiv preprint arXiv:2210.05050*, 2022.

[68] Dongran Yu, Bo Yang, Dayou Liu, Hui Wang, and Shirui Pan. A survey on neural-symbolic learning systems. *Neural Networks*, 2023.

[69] Marta Garnelo, Kai Arulkumaran, and Murray Shanahan. Towards deep symbolic reinforcement learning. *arXiv preprint arXiv:1609.05518*, 2016.

[70] Artur d'Avila Garcez, Aimore Resende Riquetti Dutra, and Eduardo Alonso. Towards symbolic reinforcement learning with common sense. *arXiv preprint arXiv:1804.08597*, 2018.

[71] Michele Fraccaroli, Giulia Mazzuchelli, and Alice Bizzarri. Machine learning techniques for extracting relevant features from clinical data for covid-19 mortality prediction. In *2021 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–7. IEEE, 2021.

[72] AN Fadja, M Fraccaroli, and A Bizzarri. Neural-symbolic system for predicting covid-19 positivity. *Clin Case Rep Int. 2022; 6*, 1429.

[73] Alice Bizzarri, Michele Fraccaroli, Evelina Lamma, and Fabrizio Riguzzi. Integration between constrained optimization and deep networks: a survey. *Frontiers in Artificial Intelligence*, 7:1414707, 2024.

[74] Alice Bizzarri, Brian Jalaian, Fabrizio Riguzzi, and Nathaniel D Bastian. A neuro-symbolic artificial intelligence network intrusion detection system. In *2024 33rd International Conference on Computer Communications and Networks (ICCCN)*, pages 1–9. IEEE, 2024.

[75] Alice Bizzarri, Chung-En Yu, Brian Jalaian, Fabrizio Riguzzi, and Nathaniel D. Bastian. Neuro-symbolic integration for open set recognition in network intrusion detection. In *AIxIA 2024 - International Conference of the Italian Association for Artificial Intelligence*. Springer, 2024. To appear.

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

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

[78] Arnaud NGUEMBANG FADJA, Fabrizio Riguzzi, Evelina Lamma, et al. Learning the parameters of deep probabilistic logic programs. In *CEUR WORKSHOP PROCEEDINGS*, volume 2219, pages 9–14. CEUR-WS, 2018.

[79] Arnaud Nguembang Fadja, Fabrizio Riguzzi, and Evelina Lamma. Expectation maximization in deep probabilistic logic programming. In *AI* IA 2018–Advances in Artificial Intelligence: XVIIth International Conference of the Italian Association for Artificial Intelligence, Trento, Italy, November 20–23, 2018, Proceedings 17*, pages 293–306. Springer, 2018.

[80] Michele Fraccaroli, Alice Bizzarri, Paolo Casellati, and Evelina Lamma. Exploiting cnn's visual explanations to drive anomaly detection. *Applied Intelligence*, 54(1):414–427, 2024.

[81] 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.

[82] 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.

[83] MK Vijaymeena and K Kavitha. A survey on similarity measures in text mining. *Machine Learning and Applications: An International Journal*, 3(2):19–28, 2016.

[84] 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.

[85] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, pages 234–241. Springer, 2015.

[86] 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.

[87] 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.

[88] Sara Sangalli, Ertunc Erdil, Andeas Hötker, Olivio Donati, and Ender Konukoglu. Constrained optimization to train neural networks on critical and under-represented classes. *Advances in Neural Information Processing Systems*, 34, 2021.

[89] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[90] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[91] Jitendra Parmar, Satyendra Chouhan, Vaskar Raychoudhury, and Santosh Rathore. Open-world machine learning: applications, challenges, and opportunities. *ACM Computing Surveys*, 55(10):1–37, 2023.

[92] Fei Zhu, Shijie Ma, Zhen Cheng, Xu-Yao Zhang, Zhaoxiang Zhang, and Cheng-Lin Liu. Open-world machine learning: A review and new outlooks. *arXiv preprint arXiv:2403.01759*, 2024.

[93] Jingkang Yang, Kaiyang Zhou, Yixuan Li, and Ziwei Liu. Generalized out-of-distribution detection: A survey. *arXiv preprint arXiv:2110.11334*, 2021.

[94] Robin Manhaeve, Francesco Giannini, Mehdi Ali, Damiano Azzolini, Alice Bizzarri, Andrea Borghesi, Samuele Bortolotti, Luc De Raedt, Devendra Dhami, Michelangelo Diligenti, et al. Benchmarking in neuro-symbolic ai. In *Proceedings of The 4th International Joint Conference on Learning & Reasoning*, 2024.

[95] Wolfgang Stammer, Patrick Schramowski, and Kristian Kersting. Right for the right concept: Revising neuro-symbolic concepts by interacting with their explanations. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3619–3629, 2021.

[96] Heimo Müller and Andreas Holzinger. Kandinsky patterns. *Artificial intelligence*, 300:103546, 2021.

[97] Emanuele Marconato, Gianpaolo Bontempo, Elisa Ficarra, Simone Calderara, Andrea Passerini, and Stefano Teso. Neuro-symbolic continual learning: knowledge, reasoning shortcuts and concept rehearsal. In *Proceedings of the 40th International Conference on Machine Learning*, pages 23915–23936, 2023.

[98] 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*, 10:1–4, 2020.

[99] 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.

[100] 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.

[101] Leonardo Rundo, Roberto Pirrone, Salvatore Vitabile, Evis Sala, and Orazio Gambino. Recent advances of hci in decision-making tasks for optimized clinical workflows and precision medicine. *Journal of biomedical informatics*, 108:103479, 2020.

[102] 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.

[103] 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.

[104] Maureen Van Eijnatten, Roelof van Dijk, Johannes Dobbe, Geert Streekstra, Juha Koivisto, and Jan Wolff. Ct image segmentation methods for bone used in medical additive manufacturing. *Medical engineering & physics*, 51:6–16, 2018.

[105] Leonardo Rundo, Lucian Beer, Stephan Ursprung, Paula Martin-Gonzalez, Florian Markowetz, James D Brenton, Mireia Crispin-Ortuzar, Evis Sala, and Ramona Woitek. Tissue-specific and interpretable sub-segmentation of whole tumour burden on ct images by unsupervised fuzzy clustering. *Computers in biology and medicine*, 120:103751, 2020.

[106] Keh-Shih Chuang, Hong-Long Tzeng, Sharon Chen, Jay Wu, and Tzong-Jer Chen. Fuzzy c-means clustering with spatial information for image segmentation. *computerized medical imaging and graphics*, 30(1):9–15, 2006.

[107] 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.

[108] 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.

[109] 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.

[110] 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):18716, 2020.

[111] 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*, 15:1435–1443, 2020.

[112] Augusto Di Castelnuovo, Marialaura Bonaccio, Simona Costanzo, Alessandro Gialluisi, Andrea Antinori, Nausicaa Berselli, Lorenzo Blandi, Raffaele Bruno, Roberto Cauda, Giovanni Guaraldi, 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. *Nutrition, Metabolism and Cardiovascular Diseases*, 30(11):1899–1913, 2020.

[113] 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.

[114] Nesreen Alsharman and Ibrahim Jawarneh. Googlenet cnn neural network towards chest ct-coronavirus medical image classification. *J. Comput. Sci*, 16(5):620–625, 2020.

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

[116] 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.

[117] 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*, 34(1):1295–1310, 2022.

[118] 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(7):1213, 2021.

[119] 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.

[120] 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.

[121] 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.

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

[123] Haifeng Jin, Qingquan Song, and Xia Hu. Auto-Keras: An efficient neural architecture search system. In *25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1946–1956, 2019.

[124] 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.

[125] Michele Fraccaroli, Evelina Lamma, and Fabrizio Riguzzi. Symbolic DNN-tuner. *Machine Learning*, pages 1–26, 2021.

[126] Edgar Liberis, Łukasz Dudziak, and Nicholas D Lane. μNAS: Constrained neural architecture search for microcontrollers. In *the 1st Workshop on Machine Learning and Systems*, pages 70–79, 2021.

[127] Jin-Dong Dong, An-Chieh Cheng, Da-Cheng Juan, Wei Wei, and Min Sun. DPP-Net: Device-aware progressive search for pareto-optimal neural architectures. In *the European Conference on Computer Vision (ECCV)*, pages 517–531, 2018.

[128] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. MNASNET: Platform-aware neural architecture search for mobile. In *the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2820–2828, 2019.

[129] Edgar Liberis and Nicholas D Lane. Neural networks on microcontrollers: saving memory at inference via operator reordering. *arXiv preprint arXiv:1910.05110*, 2019.

[130] Biswajit Paria, Kirthevasan Kandasamy, and Barnabás Póczos. A flexible framework for multi-objective bayesian optimization using random scalarizations. In *Uncertainty in Artificial Intelligence*, pages 766–776. PMLR, 2020.

[131] Igor Fedorov, Ryan P Adams, Matthew Mattina, and Paul Whatmough. Sparse: Sparse architecture search for CNNs on resource-constrained microcontrollers. *Advances in Neural Information Processing Systems*, 32, 2019.

[132] Ashish Kumar, Saurabh Goyal, and Manik Varma. Resource-efficient machine learning in 2 kb ram for the internet of things. In *International conference on machine learning*, pages 1935–1944. PMLR, 2017.

[133] Chirag Gupta, Arun Sai Suggala, Ankit Goyal, Harsha Vardhan Simhadri, Bhargavi Paranjape, Ashish Kumar, Saurabh Goyal, Raghavendra Udupa, Manik Varma, and Prateek Jain. Protonn: Compressed and accurate knn for resource-scarce devices. In *International conference on machine learning*, pages 1331–1340. PMLR, 2017.

[134] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Efficient multi-objective neural architecture search via lamarckian evolution. *arXiv preprint arXiv:1804.09081*, 2018.

[135] Yanqi Zhou, Siavash Ebrahimi, Sercan Ö Arık, Haonan Yu, Hairong Liu, and Greg Diamos. Resource-efficient neural architect. *arXiv preprint arXiv:1806.07912*, 2018.

[136] Yundong Zhang, Naveen Suda, Liangzhen Lai, and Vikas Chandra. Hello edge: Keyword spotting on microcontrollers. *arXiv preprint arXiv:1711.07128*, 2017.

[137] Ji Lin, Wei-Ming Chen, Yujun Lin, Chuang Gan, Song Han, et al. Mcunet: Tiny deep learning on iot devices. *Advances in Neural Information Processing Systems*, 33:11711–11722, 2020.

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

[139] 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*, 113:35–58, 2021.

[140] 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.

[141] 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.

[142] Markos Markou and Sameer Singh. Novelty detection: a review—part 1: statistical approaches. *Signal processing*, 83(12):2481–2497, 2003.

[143] Victoria Hodge and Jim Austin. A survey of outlier detection methodologies. *Artificial intelligence review*, 22:85–126, 2004.

[144] Daniel Weimer, Bernd Scholz-Reiter, and Moshe Shpitalni. Design of deep convolutional neural network architectures for automated feature extraction in industrial inspection. *CIRP annals*, 65(1):417–420, 2016.

[145] Benjamin Staar, Michael Lütjen, and Michael Freitag. Anomaly detection with convolutional neural networks for industrial surface inspection. *Procedia CIRP*, 79:484–489, 2019.

[146] Elad Hoffer and Nir Ailon. Deep metric learning using triplet network. In *Similarity-based pattern recognition: third international workshop, SIMBAD 2015, Copenhagen, Denmark, October 12-14, 2015. Proceedings 3*, pages 84–92. Springer, 2015.

[147] Samet Akcay, Amir Atapour-Abarghouei, and Toby P Breckon. Ganomaly: Semi-supervised anomaly detection via adversarial training. In *Computer Vision–ACCV 2018: 14th Asian Conference on Computer Vision, Perth, Australia, December 2–6, 2018, Revised Selected Papers, Part III 14*, pages 622–637. Springer, 2019.

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

[149] Niccolò Ferrari, Michele Fraccaroli, and Evelina Lamma. Grd-net: Generative-reconstructive-discriminative anomaly detection with region of interest attention module. *International Journal of Intelligent Systems*, 2023(1):7773481, 2023.

[150] DP Kingma. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[151] 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.

[152] 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.

[153] 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.

[154] 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.

[155] 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.

[156] 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.

[157] Hessam Bagherinezhad, Maxwell Horton, Mohammad Rastegari, and Ali Farhadi. Label refinery: Improving imagenet classification through label progression. *arXiv preprint arXiv:1805.02641*, 2018.

[158] Suorong Yang, Weikang Xiao, Mengcheng Zhang, Suhan Guo, Jian Zhao, and Furao Shen. Image data augmentation for deep learning: A survey. *arXiv preprint arXiv:2204.08610*, 2022.

[159] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48, 2019.

[160] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

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

[162] Anna L. Buczak and Erhan Guven. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys & Tutorials*, 18(2):1153–1176, 2016.

[163] Anna Drewek-Ossowicka, Mariusz Pietrołaj, and Jacek Rumiński. A survey of neural networks usage for intrusion detection systems. *Journal of Ambient Intelligence and Humanized Computing*, 12:497–514, 2021.

[164] Robin Sommer and Vern Paxson. Outside the closed world: On using machine learning for network intrusion detection. *In 2010 IEEE Symposium on Security and Privacy*, pages 305–316, 2010.

[165] S Sibi Chakkaravarthy, D Sangeetha, and V Vaidehi. A survey on malware analysis and mitigation techniques. *Computer Science Review*, 32:1–23, 2019.

[166] Joana C Costa, Tiago Roxo, Hugo Proença, and Pedro RM Inácio. How deep learning sees the world: A survey on adversarial attacks & defenses. *IEEE Access*, 2024.

[167] Muhammad Fahad Umer, Muhammad Sher, and Yaxin Bi. Flow-based intrusion detection: Techniques and challenges. *Computers & Security*, 2017.

[168] Yasir Ali Farrukh, Irfan Khan, Syed Wali, David Bierbrauer, John V. Pavlik, and Nathaniel D. Bastian. Payload-byte: A tool for extracting and labeling packet capture files of modern network intrusion detection datasets. 2022.

[169] Augustine Premkumar, Madeleine Schneider, Carlton Spivey, John Pavlik, and Nathaniel D. Bastian. Graph representation learning for context-aware network intrusion detection. *Proc. SPIE 12538, Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications V*, 2023. 125380H (12 June 2023).

[170] KDD Cup 1999 Task Committee. KDD Cup 1999 Data. https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html, 1999.

[171] Ghulam Mohi-ud din. Nsl-kdd, 2018.

[172] Iman Sharafaldin, Arash Habibi Lashkari, Ali A Ghorbani, et al. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp*, 1:108–116, 2018.

[173] Nour Moustafa and Jill Slay. Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In *2015 military communications and information systems conference (MilCIS)*, pages 1–6. IEEE, 2015.

[174] Nathaniel Bastian, David Bierbrauer, Morgan McKenzie, and Emily Nack. Aci iot network traffic dataset 2023, 2023.

[175] Marc Chale and Nathaniel D. Bastian. Generating realistic cyber data for training and evaluating machine learning classifiers for network intrusion detection systems. *Expert Systems with Applications*, 207:117936, 2022.

[176] Soumyadeep Hore, Jalal Ghadermazi, Diwas Paudel, Ankit Shah, Tapas K. Das, and Nathaniel D. Bastian. Deep packgen: A deep reinforcement learning framework for adversarial network packet generation, 2023.

[177] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp*, 1:108–116, 2018.

[178] Gerard Draper-Gil., Arash Habibi Lashkari., Mohammad Saiful Islam Mamun., and Ali A. Ghorbani. Characterization of encrypted and vpn traffic using time-related features. In *Proceedings of the 2nd International Conference on Information Systems Security and Privacy - ICISSP*, pages 407–414. INSTICC, SciTePress, 2016.

[179] Brian Matejek, Ashish Gehani, Nathaniel D Bastian, Daniel Clouse, Bradford Kline, and Susmit Jha. Safeguarding network intrusion detection models from zero-day attacks and concept drift.

[180] Sultan Zavrak and Murat Iskefiyeli. Anomaly-based intrusion detection from network flow features using variational autoencoder. *IEEE Access*, 8:108346–108358, 2020.

[181] Chuanxing Geng, Sheng-jun Huang, and Songcan Chen. Recent advances in open set recognition: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 43(10):3614–3631, 2020.

[182] Walter J Scheirer, Lalit P Jain, and Terrance E Boult. Probability models for open set recognition. *IEEE transactions on pattern analysis and machine intelligence*, 36(11):2317–2324, 2014.

[183] Mehadi Hassen and Philip K Chan. Learning a neural-network-based representation for open set recognition. In *Proceedings of the 2020 SIAM International Conference on Data Mining*, pages 154–162. SIAM, 2020.

[184] Abhijit Bendale and Terrance E Boult. Towards open set deep networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1563–1572, 2016.

[185] Lawrence Neal, Matthew Olson, Xiaoli Fern, Weng-Keen Wong, and Fuxin Li. Open set learning with counterfactual images. In *Proceedings of the European conference on computer vision (ECCV)*, pages 613–628, 2018.

[186] ZongYuan Ge, Sergey Demyanov, Zetao Chen, and Rahil Garnavi. Generative openmax for multi-class open set classification. *arXiv preprint arXiv:1707.07418*, 2017.

[187] Adnan Shahid Khan, Zeeshan Ahmad, Johari Abdullah, and Farhan Ahmad. A spectrogram image-based network anomaly detection system using deep convolutional neural network. *IEEE access*, 9:87079–87093, 2021.

[188] Giuseppina Andresini, Annalisa Appice, Nicola Di Mauro, Corrado Loglisci, and Donato Malerba. Multi-channel deep feature learning for intrusion detection. *IEEE Access*, 8:53346–53359, 2020.

[189] Muhammad Asam, Saddam Hussain Khan, Altaf Akbar, Sameena Bibi, Tauseef Jamal, Asifullah Khan, Usman Ghafoor, and Muhammad Raheel Bhutta. Iot malware detection architecture using a novel channel boosted and squeezed cnn. *Scientific Reports*, 12(1):15498, 2022.

[190] Ulya Sabeel, Shahram Shah Heydari, Khalid Elgazzar, and Khalil El-Khatib. Building an intrusion detection system to detect atypical cyberattack flows. *IEEE Access*, 9:94352–94370, 2021.

[191] Ren-Hung Hwang, Min-Chun Peng, Chien-Wei Huang, Po-Ching Lin, and Van-Linh Nguyen. An unsupervised deep learning model for early network traffic anomaly detection. *IEEE Access*, 8:30387–30399, 2020.

[192] Nada Abdalgawad, A Sajun, Y Kaddoura, Imran A Zualkernan, and F Aloul. Generative deep learning to detect cyberattacks for the iot-23 dataset. *IEEE Access*, 10:6430–6441, 2021.

[193] Siva S. Sivatha Sindhu, S. Geetha, M. Marikannan, and A. Kannan. A neuro-genetic based short-term forecasting framework for network intrusion prediction system. *International Journal of Automation and Computing*, 6(4):406–414, oct 21 2009.

[194] Darian Onchis, Codruta Istin, and Eduard Hogea. A neuro-symbolic classifier with optimized satisfiability for monitoring security alerts in network traffic. *Applied Sciences*, 12(22):11502, nov 12 2022.

[195] Chao Liu, Zhaojun Gu, and Jialiang Wang. A hybrid intrusion detection system based on scalable k-means+ random forest and deep learning. *Ieee Access*, 9:75729–75740, 2021.

[196] Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. Gnnexplainer: Generating explanations for graph neural networks. *Advances in neural information processing systems*, 32, 2019.

Pubblications

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

- International Journals:
  - Nguembang Fadja, A., Fraccaroli, M., Bizzarri, A., Mazzuchelli, G., & Lamma, E. (2022). *Neural-Symbolic Ensemble Learning for Early-Stage Prediction of Critical State of COVID-19 Patients*. *Medical & Biological Engineering & Computing*, 60(12), 3461-3474.
  - Nguembang Fadja, A., Fraccaroli, M., & Bizzarri, A. (2022). *Neural-Symbolic System for Predicting COVID-19 Positivity*. *Clin Case Rep Int.*, 6.
  - Bizzarri, A., Fraccaroli, M., Lamma, E., & Riguzzi, F. (2024). *Integration Between Constrained Optimization and Deep Networks: A Survey*. *Frontiers in Artificial Intelligence*, 7, 1414707.
  - Fraccaroli, M., Bizzarri, A., Casellati, P., & Lamma, E. (2024). *Exploiting CNN's Visual Explanations to Drive Anomaly Detection*. *Applied Intelligence*, 54(1), 414-427.
- International Conferences:
  - Fraccaroli, M., Mazzuchelli, G. & Bizzarri, A. *Machine Learning Techniques for Extracting Relevant Features from Clinical Data for COVID-19 Mortality Prediction*. *2021 IEEE Symposium on Computers and Communications (ISCC)*, pages 1-7, IEEE, 2021.
  - Bellodi, E., Bertozzi, D., Bizzarri, A., Favalli, M., Fraccaroli, M., & Zese, R. (2023). *Efficient Resource-Aware Neural Architecture Search with a Neuro-Symbolic Approach*. *IEEE 16th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*.
  - Gentili, E., Bizzarri, A., Azzolini, D., Zese, R., & Riguzzi, F. (2023). *Regularization in Probabilistic Inductive Logic Programming. In International Conference on Inductive Logic Programming*, pp. 16-29, Cham: Springer Nature Switzerland.

  - Azzolini, D., Bizzarri, A., Fraccaroli, M., Bertasi, F., & Lamma, E. (2023). *A Machine Learning Pipeline to Analyse Multispectral and Hyperspectral Images*. In 2023 International Conference on Computational Science and Computational Intelligence (CSCI), pp. 1306-1311, IEEE.
  - Bizzarri, A., Jalaian, B., Riguzzi, F., & Bastian, N. D. (2024). *A Neuro-Symbolic Artificial Intelligence Network Intrusion Detection System. 33rd International Conference on Computer Communications and Networks (ICCCN)*, IEEE.
  - Bizzarri, A., Yu, C., Jalaian, B., Riguzzi, F., & Bastian, N. D. (2024). *A Neuro-Symbolic Integration for Open Set Recogniction in Network Intrusion Detection System. AIxIA 2024 - International Conference of the Italian Association for Artificial Intelligence*, Springer, 2024. To appear.

- Pre-Print
  - Bizzarri, A., Yu, C. E., Jalaian, B., Riguzzi, F., & Bastian, N. D. (2024). *A Synergistic Approach In Network Intrusion Detection By Neurosymbolic AI.* arXiv preprint arXiv:2406.00938.
  - Ferrari, N., Zanarini, N., Fraccaroli, M., Bizzarri, A., & Lamma, E. *Integration of Deep Generative Anomaly Detection Algorithm in High-Speed Industrial Line*. Available at SSRN 4858664.