Doctoral Dissertation
Doctoral Program in Computer Engineering (36^{th}cycle)

Robust machine learning models for high dimensional data interpretation

By

Manigrasso Francesco

******

Supervisor:
Prof. Fabrizio Lamberti

Referees:
Prof. Annette ten Teije, Vrije Universitat (Referee)
Prof. Luciano Serafini, Fondazione Bruno Kessler (Referee)
Prof. Francesco Setti, Università di Verona
Prof. Luca Bortolussi, Università di Trieste
Prof. Paolo Garza, Politecnico di Torino

Politecnico di Torino
2024

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.

Manigrasso Francesco
2024

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

My deepest and most sincere gratitude goes to my family, whose love, encouragement, and unwavering support have been the foundation of this achievement. Without them, this journey would not have been possible.

To my father: thank you for instilling in me a passion for knowledge and teaching me the importance of determination and integrity. You believed in me when doubt overshadowed certainty, and when no one—including myself—could see the potential for this path. Much like a coach stepping aside to let the team savor the glory of victory, I know you would humbly let the spotlight shine elsewhere today. Yet, if I have reached this point, if I have become the person I am, it is also because of the words you shared with me as a child, words that grew into the pillars I clung to when there were no lights to guide my way. Your unwavering dedication to work and family has been an inspiration, pushing me to strive for my best in every aspect of life. This doctoral dissertation stands as a testament to your belief in me, your endless love, and your sacrifices. From the depths of my heart, thank you.

To all those who have played a role in this story, my heartfelt thanks. These few words cannot fully convey the esteem and gratitude I hold for each of you, but I will do my best to express it. First and foremost, to Salvatore, my lifelong friend: you have been a cornerstone of my journey and, perhaps, of my entire life. Despite the miles that separate us, you remain the person I can always count on without hesitation. The moments we have shared have enriched my life in immeasurable ways. I am profoundly grateful to you for your steadfast friendship and for being an extraordinary companion along the way. To Luciana, my dear friend and a key figure in this chapter of my life: despite all the playful banter, you are one of the strongest people I know. Your presence and encouragement have been a guiding light in moments of doubt and difficulty. Thank you for your strength, your kindness, and for always being there when it mattered most. To Valeria, Enrico, Giampaolo, *Franco, Paolo, Antonio, Francesco G., and Roberto: each of you has left an indelible mark on this journey. Your support, insights, and encouragement have profoundly shaped my personal and professional growth. You have broadened my horizons and enriched my perspective on life. I carry immense gratitude for each of you, and I will always cherish the role you have played in this achievement.*

Lastly, I would like to extend my heartfelt thanks to my Turin family. Each of you has contributed to this journey in your own unique way, supporting me and shaping the person I am today. Without your presence and guidance, this milestone would not have been possible. A heartfelt thank you goes to Carla, who remembers even the earring I was wearing the first day I set foot in this city. You are a special person and a friend I know I can always count on. Thank you for being there and for your support. Finally, to the newest additions to my life: Andrea, Goki, and Giorgina. I could list the adventures we've shared, but if I did, they might revoke my doctoral degree and hand me back a middle school diploma instead! Thank you for bringing joy, laughter, and a touch of chaos to this journey. Your presence has made this path all the more memorable.

To all of you, thank you from the bottom of my heart. This milestone is as much yours as it is mine.

Acknowledgements

I wish to express my deepest gratitude to the distinguished professors who have played a pivotal role in my academic journey and the completion of this doctoral dissertation.

First and foremost, I am profoundly thankful to Professor Fabrizio Lamberti for his expert guidance, unwavering support, and dedication to my success. His vast knowledge and enthusiasm for research have significantly enhanced the quality of this work.

I also extend my heartfelt thanks to Professor Lia Morra for her invaluable contributions, wise counsel, and steadfast support throughout the research process. Her experience and critical insights have enriched this dissertation and were instrumental in its successful completion.

I am especially grateful to Professor Peter Bloem, who was a key figure during my tenure at Vrije Universiteit Amsterdam. His continuous support made my time abroad both memorable and impactful, serving as a source of knowledge and inspiration in an unfamiliar and challenging environment.

Furthermore, I extend my appreciation to all the other professors and lecturers from whom I have had the privilege to learn throughout my academic career. Your lectures, support, and encouragement have greatly contributed to my education and professional growth. Without your invaluable input and guidance, reaching this significant milestone would not have been possible. I deeply appreciate your commitment and dedication to educating and inspiring students.

Abstract

Deep neural networks (DNNs) are highly effective in modeling complex data, like image pixels, by employing both linear and non-linear feature representations. Their success is mainly due to the vast amounts of data available and the optimized supervised learning algorithms. Nonetheless, DNNs lack essential qualities such as robustness, control, and transparency, which are crucial for practical applications. To overcome this limitation, researchers are increasingly exploring the integration of knowledge representation and relational statistical reasoning strategies. While statistical models have their drawbacks, symbolic knowledge representation offers significant benefits in terms of transparency, inference and reasoning capabilities, high-level concept representation, integration of prior knowledge, and modeling of the external world.

Leveraging current advancements in neural-symbolic integration architectures, this Ph.D. research seeks to explore methods to bridge the gap between symbolic knowledge representation in multidimensional data analysis and deep representation learning. The emphasis is on neurosymbolic integration, which enhances DNN learning features by incorporating a priori information. This method allows symbolic representation techniques, such as fuzzy logic, to be encoded as tensors in a neural network.

To improve performance benchmarks, reduce the data required for model training, and advance knowledge acquisition, this doctoral dissertation explores the development of architectures for addressing image-based problems (e.g., object detection and classification) using image representation features for logical reasoning. It aligns the learning process with various inductive biases through practical implementations of groundings, utilizing diverse datasets to make these theories applicable to real-world scenarios.

Contents

List of Figures

List of Tables

Chapter 1

Neural-symbolic architectures for high-dimensional data interpretation

Over the past few decades, Artificial Intelligence (AI) has made significant progress and gained popularity in both academic and business contexts, thanks to improvements in hardware performance, increased resource availability, and optimization approaches. Deep Learning (DL) researchers have achieved remarkable results, enabling the application of these models in numerous sectors such as bioinformatics, finance, healthcare, and autonomous driving [3–6].

However, growing concerns about the reliability, security, and accountability of these systems highlight the need to integrate different methods of representation and reasoning within data-driven systems, with the goal of alleviating these critical issues. Knowledge representation and reasoning technique can provide a way to process and leverage structured representations, essential for building robust and transparent systems. Combining such techniques with data-driven machine learning models into hybrid techniques can potentially lead to increased system reliability and security. Using formal logic and knowledge representation, such hybrid systems could yield clearer and more interpretable results, fostering user trust. Currently, the primary goal of the research community is to approximate the human mind's ability to assess and process data. Through the incorporation of advanced methods of representation and reasoning, progress is being made toward achieving this objective, leading to the development of systems that are both powerful and trustworthy, as well as understandable [7].

AI has historically been divided into two competing paradigms, leading to two divergent interpretations: symbolic approaches and sub-symbolic (or connectionist) approaches.

Symbolic approaches attempt to describe the universe through a system of symbols, which may be thought of as the fundamental components of human intelligence manipulated by cognitive and reasoning processes through a system of rules and logical processes. Before the advent of deep learning, these systems were commonly employed and offered several advantages, such as minimal data needs, declarative language for knowledge representation, and the capability to explicitly manipulate internal conceptual representations. These systems are inherently readable and interpretable. However, they face bottlenecks such as the labor-intensive process of knowledge acquisition based on human annotations, perform less effectively in real-world scenarios, and are particularly intolerant to ambiguous data or noise [8].

Sub-symbolic or connectionist approaches include machine learning subfields such as deep learning, which define systems capable of learning representations at various levels of complexity. Nonlinear representations allow for the conversion of input data to more abstract levels and simple compositions [9], enabling mastery of incredibly complex functions with a wide range of applications. For example, deep learning can be applied to tasks involving images, audio, text, speech recognition [10], object classification [11], image recognition [12], topic classification [13], sentiment analysis [14], or language translation activities [15]. By learning through weight adjustments, deep learning models minimize an error-derived cost function from a training dataset using gradient backpropagation techniques [16]. These structures pose challenges such as the difficulty of translating model knowledge into human-understandable concepts [17, 18], requiring large amounts of data for high accuracy, and being significantly affected by the quantity and quality of available annotations [19–21].

The two paradigms have complementary strengths and weaknesses. Deep learning models excel at handling large amounts of unstructured data and are resilient to noise and inconsistencies, but they act as “black boxes”, making it difficult to interpret their decision-making processes. In contrast, symbolic systems offer clear, human-readable representations of knowledge and reasoning, but they struggle with the unpredictability and variability of real-world data. By combining these approaches, researchers hope to leverage their strengths to create AI systems that are both powerful and interpretable [22, 23].

The integration of symbolic reasoning with neural networks—referred to as Neuro-Symbolic (NeSy) systems—has thus gained significant attention. The NeSy field aims to bridge the gap between symbolic reasoning, which abstracts human cognitive processes, and neural networks, which mirror the physiological mechanisms of the brain [22]. This hybrid approach is viewed as a potential solution to some of the long-standing challenges in AI, such as improving interpretability, generalization, and robustness in complex real-world environments such as autonomous systems and natural language understanding. A key advantage of NeSy methodologies is their ability to reformulate the learning process, potentially reducing the reliance on extensive labeled datasets. By leveraging symbolic knowledge as a form of higher-level supervision, NeSy systems can narrow the "supervision gap"—the difference between the large amounts of annotated data typically required by neural networks and the more limited data needed when symbolic guidance is employed. This reduction in data demands is particularly beneficial in fields where annotated data is scarce or costly to obtain.

Several approaches have been proposed to bridge this supervision gap and enhance the interaction between neural and symbolic components, including:

1. Augmenting neural network learning with symbolic systems, where logical constraints are imposed to improve both interpretability and performance [24–26];

2. Developing bidirectional architectures, where the outputs of neural networks feed into symbolic components and vice versa, allowing for a more interpretable and transparent reasoning process [27–29].

At the same time, although outside of the scope of this dissertation, the computational efficiency of symbolic reasoning can be improved by replacing symbolic reasoning techniques with approximate algorithms based on neural networks to reduce the search space; neural networks are also capable of abstracting and extracting symbolic representations from data, thereby facilitating reasoning [30–32]. Achieving an optimal balance and proper combination of sub-symbolic and symbolic systems, which utilize distinct problem-solving strategies and data representations, is essential for addressing specific problem requirements, yet integrating these systems into a unified network presents considerable challenges.

1.1 Main research challenges

The increasing focus on developing NeSy architectures in computer vision, which integrate neural networks with symbolic reasoning, is driven by key advantages such as enhanced interpretability, improved robustness to noisy data, and reduced dependence on large labeled datasets through the incorporation of external symbolic knowledge, offering the potential to tackle critical challenges in modern AI by combining data-driven learning with structured rule-based reasoning.

This doctoral dissertation focuses on NeSy architectures in computer vision tasks, emphasizing the development of suitable knowledge bases to represent external knowledge and logical constraints, the end-to-end training of the entire architecture, and the creation of appropriate grounding for the symbolic component of the architecture. These aspects are evaluated and compared to conventional DL architectures, demonstrating advantages in classic computer vision tasks while maintaining low overhead in architectural complexity and training effort.

In particular, the dissertation focuses on the integration of a specific class of NeSy techniques, denoted as Logic Tensor Networks (LTNs), in computer vision tasks and benchmarks. LTNs have the ability to integrate background knowledge in the learning task as differentiable logical axioms. They employ Real Logic, a differentiable extension of first-order logic, interpreting predicates as continuous functions within $[0, 1]$. This approach addresses uncertainty and soft constraints, solving AI tasks such as clustering, classification, relational learning, query answering, semi-supervised learning, regression, and embedding within a unified framework that seamlessly integrates data learning and reasoning, as demonstrated in [33].

This dissertation aims to demonstrate how this NeSy technique can improve machine learning models and facilitate the interpretation of multidimensional data, especially images [34]. The research is divided into two core sections, each addressing a distinct but related research question. The first explores the embedding of logical constraints in object detection frameworks to improve accuracy and interpretability. The second investigates advances in NeSy architectures for Zero-Shot

Learning (ZSL), a field where leveraging external knowledge can significantly enhance model performance in the absence of labeled examples.

1.1.1 Logical constraints to improve object detection frameworks

The architecture developed in this doctoral disertation, named Faster-LTN [35] and discussed inChapter 4, merges a NeSy framework with the Faster R-CNN model to improve object detection. This hybrid system integrates a convolutional backbone with an LTN, creating an end-to-end trainable system. Faster-LTN aims to combine the semantic knowledge representation and reasoning capabilities of LTNs with the feature extraction and learning efficiency of convolutional neural networks (CNNs).

Specifically, the Faster-LTN architecture includes a Region Proposal Network (RPN) and an LTN-based classification head, which are jointly trained to maximize the satisfiability of a grounded theory that integrates labeled examples with logical axioms. In the LTN framework, grounding refers to the process of interpreting First-Order Logic (FOL) language as functions on real-valued vectors. The grounding function $\mathcal{G}$ maps terms and formulas in the knowledge base $\mathcal{K}$ to real values, with predicates interpreted as differentiable operations on real tensors. In Faster-LTN, this framework is used to encode object detection tasks, defining predicates such as `isOfClass` and `isOfPart` to represent object classes and their relationships. The model enforces mutual exclusion and mereological constraints derived from ontologies such as WordNet, ensuring logical consistency in object relationships.

Faster-LTN incorporates modifications aimed at enhancing scalability and addressing class imbalance by employing a focal log-product aggregation function, which increases the influence of misclassified instances while adjusting for class disparities. Experimental evaluations on the PASCAL VOC and PASCAL PART datasets reveal that Faster-LTN achieves competitive performance when compared to the conventional Faster R-CNN architecture; specifically, it enhances the mean Average Precision (mAP) from 62.6 to 73.8 on the PASCAL VOC dataset, effectively improving object detection and providing a more adaptable classification head, albeit necessitating additional training epochs for convergence. Consequently, Faster-LTN signifies a notable advancement in the integration of symbolic reasoning with deep learning for object detection, showcasing the potential to incorporate high-level logical constraints and background knowledge to develop more robust AI systems.

The implementation is available at https://gitlab.com/grains2/Faster-LTN.

1.1.2 Neuro-Symbolic techniques for Zero-Shot Learning

This dissertation also presents novel architectures PROTOtypical Logic Tensor Networks (Proto-LTN) [36], Fuzzy Logic Visual Network (FLVN) [37], and Fuzzy Logic Prototypical Network (FLPN), which offer a cohesive and systematic investigation of the integration of symbolic reasoning within deep learning architectures designed for classification tasks under the ZSL setting. This research addresses key scientific questions concerning the limitations of conventional ZSL approaches, focusing on how symbolic knowledge can be effectively incorporated into neural network architectures to improve both performance and generalization.

The initial investigation utilizing Proto-LTN centered on grounding abstract class concepts within continuous embedding spaces by integrating NeSy architectures with prototypical networks. Subsequently, FLVN introduced a novel method for enhancing the integration of visual and semantic information, employing high-level inductive biases and logical constraints to improve the robustness and interpretability of ZSL models while mitigating premature overfitting to seen classes. The most recent advancement, FLPN, builds upon previous approaches by incorporating attention mechanisms and supporting both CNN and transformer backbones, thereby enhancing model performance and generalization to unseen classes through the creation of a unified embedding space for visual features and class attributes, utilizing logical axioms and refining prototype matching. Collectively, the Proto-LTN, FLVN, and FLPN architectures represent significant advancements in the integration of symbolic reasoning with deep learning for visual feature matching, not only improving ZSL performance but also providing an extensible framework applicable to a variety of NeSy tasks.

PROTOtypical Logic Tensor Network (Proto-LTN)

The architecture of the PROTOtypical Logic Tensor Network (Proto-LTN) [36] integrates NeSy principles with prototypical networks to enhance ZSL. This framework formulates the learning task by grounding abstract concepts as parameterized class prototypes in a high-dimensional embedding space, while minimizing the number of parameters needed to establish the knowledge base.

Proto-LTN consists of a CNN that extracts visual features and an LTN that formulates the learning objective as maximizing the satisfiability of a knowledge base of logical axioms. This integration allows the model to represent class prototypes as abstract concepts and assign new instances to the nearest class prototype in an Euclidean distance.

In the context of ZSL, where the goal is to recognize objects from unseen classes, Proto-LTN leverages logical axioms to encode relationships between classes and their attributes, providing a robust framework for handling both seen and unseen classes during training. The architecture optimizes the embedding space to ensure that class prototypes and instances of the same class are close, while those of different classes are distant.

Experimental results on Generalized Zero-Shot Learning (GZSL) benchmarks, such as AWA2, CUB, aPY, and SUN, demonstrate that Proto-LTN achieves competitive performance compared to traditional embedding-based approaches. The architecture's ability to integrate prior knowledge and logical constraints, including information on unseen classes, proves advantageous.

The implementation is available at https://github.com/FrancescoManigrass/Proto-LTN.git.

The Fuzzy Logic Visual Network (FLVN)

The Fuzzy Logic Visual Network (FLVN) architecture [37] integrates a NeSy framework with LTNs to enhance ZSL by learning a visual-semantic embedding space that incorporates background knowledge through logical axioms grounded in differentiable operations. This approach leverages class hierarchies and inductive biases, such as handling exceptions in class attributes and enforcing similarity between images of the same class, to prevent overfitting to seen classes.

FLVN includes a CNN for feature extraction and an LTN that maximizes the satisfiability of a knowledge base composed of First-Order Logic (FOL) axioms. The architecture introduces novel predicates, such as `isOfClass`, $\texttt{isOfClass}_{\texttt{masked}}$, and `isOfMacro`, to align visual features with semantic attributes, thus incorporating symbolic knowledge into the learning process.

FLVN's knowledge base includes both positive and negative examples, considering class hierarchies and macro-classes. This comprehensive approach enables better generalization to unseen classes by grounding visual features in class attributes.

The implementation is available at https://gitlab.com/grains2/flvn.

Fuzzy Logic Prototypical Network (FLPN)

The Fuzzy Logic Prototypical Network (FLPN) architecture combines NeSy principles with Prototypical Networks to enhance ZSL by harnessing symbolic knowledge representation along with the learning capabilities of deep neural networks. FLPN formulates the classification task as prototype matching, where both class-level and attribute-level prototypes are extracted through an attention mechanism. This mechanism is specialized for CNN- and transformer-based backbones, allowing the model to handle exceptions in class attributes and enforce similarity between images of the same class.

The architecture utilizes a CNN or Visual Transformer (ViT) backbone for feature extraction, while the prototype network generates class and attribute prototypes in a unified embedding space. FLPN incorporates prior knowledge about class hierarchies and inductive biases, improving its generalization to unseen classes.

Experimental results on GZSL benchmarks, such as AWA2, show that FLPN achieves state-of-the-art performance, improving accuracy with less computational overhead compared to recent ZSL methods.

1.2 Structure of the Document

This doctoral dissertation is organized into several key sections. In Chapter 2, an overview of LTNs is presented. This section introduces the NeSy framework used to integrate logical information into deep learning models, establishing the foundational concepts necessary to understand how LTNs facilitate the incorporation of symbolic knowledge into neural architectures.

Following this, Chapter 3 provides a review of the state of the art, offering a brief survey of the relevant literature. This section situates the subsequent chapters within the broader research landscape, contextualizing advancements in the field, and identifying the gaps that this dissertation aims to address.

In Chapter 4, the application of NeSy networks for object detection tasks is explored, with a particular focus on the Faster-LTN architecture. This chapter compares traditional deep learning methods to NeSy networks, assessing the benefits of integrating a logical knowledge base in terms of performance enhancement and data efficiency.

Subsequently, Chapter 5, Chapter 6, and Chapter 7 examine three distinct NeSy architectures developed for ZSL. Each chapter introduces a unique knowledge base tailored to model the feature space and investigates the semantic relationships encoded within the classes. These sections provide a detailed analysis of the strengths and limitations of each approach, supported by qualitative and quantitative evaluations of the experimental results.

Finally, the dissertation concludes with Chapter 8, summarizing the key findings and contributions of the research. This last chapter outlines potential directions for future work, reflecting on insights gained throughout the study and proposing new avenues for advancing NeSy integration in machine learning.

Chapter 2

Learning with neuro-symbolic architectures

Integrating data-driven learning with preexisting background information, such as relational knowledge or logical axioms, remains a significant challenge in Semantic Image Interpretation (SII) and related tasks [38]. The purpose of NeSy integration is to merge symbolic knowledge representation and learning with machine learning techniques [39]. This strategy enables the incorporation of these new capabilities into state-of-the-art deep neural networks, potentially improving their robustness, explainability, and performance.

This section presents the LTN paradigm proposed by Serafini and d'Avila Garcez [40, 41]. LTNs work by interpreting, or grounding, a FOL as functions on real vectors. The satisfiability of a given theory can be maximized by training its parameters via stochastic gradient descent. Examples of tasks successfully tackled by LTNs include part-of-relationship detection [41] and visual relationship detection [42]. Previous studies have demonstrated how LTNs can use logical axioms drawn from pre-existing knowledge bases to make up for the lack of supervision, e.g., in Few-Shot Learning (FSL) settings.

2.1 An overview of logic tensor networks

LTNs are built on a foundation called Real Logic [43], which extends traditional first-order logic (FOL) by introducing a continuous-valued interpretation of truth. In a standard FOL framework, a logical language $\mathcal{L}$ consists of *terms* (which represent objects) and *formulas* (which represent assertions about these objects), constructed using symbols, variables, constants, and predicate functions. For example, consider the constants *Alice* and *Bob*. A formula like $\texttt{friend\_of}(Alice, Bob) \wedge \texttt{friend\_of}(Bob, Alice)$ can be used to assert that Alice and Bob are mutual friends. In classical logic, this formula would evaluate to either true or false. However, in Real Logic, the predicate `friend_of` can return a continuous truth value within [0,1], expressing, for instance, that Alice and Bob are friends to a certain degree (e.g., 0.8 or 0.9), rather than in an absolute sense.

Moreover, Real Logic allows for the formulation of general rules over entire domains. Suppose that we define two variables, x and y, ranging over a set `People`. We can then express a symmetric relationship between these variables using the formula $\forall x, y(\texttt{friend\_of}(x, y) \implies \texttt{friend\_of}(y, x))$, which states that if x is a friend of y, then y is a friend of x. In Real Logic, this rule holds with varying degrees of truth, capturing the uncertainty or fuzziness inherent in social relationships.

The continuous nature of Real Logic enables LTNs to integrate symbolic reasoning with gradient-based learning, making it possible to apply logic-based reasoning while simultaneously learning from data. This flexibility supports a range of AI tasks, such as classification, relational learning, and query answering, all within a unified framework.

The term *grounding* is employed to define an interpretation of FOL in a real domain $\mathbb{R}^n$. It enables the interpretation of these formulas, whether true or false, in features with real value, associating the degree of truth with a scalar value in the range $(0,1),(1,0)$. The grounding of terms, in a more general sense, can be composed of an embedding tensor containing features extracted from an object through a neural network. For example, the term *Alice* can be a tensor containing information extracted from a person's facial image or other types of data such as years, age, hair color, etc.

Real logic allows associating fuzzy truth values with propositions. LTNs, by grounding a knowledge base in the form of logical axioms using real logic, allow the creation of a unified learning process applicable to neural networks. The grounding function, denoted as $\mathcal{G}$, maps terms and functions to real values. Defining a complete knowledge base $\mathcal{K}$, it is possible to find the value $\mathcal{G}_\theta$ as a trainable function with parameters θ. This can be interpreted as an optimization problem for LTNs.

2.2 Learning a grounding function

Before defining terms and formulas, it is necessary to define the set of chosen objects, i.e., the domain of membership. Referring to the example given in the previous section, *Alice* and *Bob* were defined as belonging to the domain of *People*, but alternatively it could be possible to define the domain of membership as *Italians* or *Dutch*. A grounding function $\mathcal{G}$ expresses the interpretation in the language $\mathcal{L}$; this function must be defined for each sentence, as well as for the individual terms or formulas present in the knowledge base $\mathcal{K}$.

In the context of the initial example, each term could be associated with a tensor extracted through a neural network. The grounding of *Alice* and *Bob* could be defined as $\mathcal{G}(Alice) \in \mathbb{R}^4$ and $\mathcal{G}(Bob) \in \mathbb{R}^4$, while that of *Italians* and *Dutch*, to which *Alice* and *Bob* respectively belong in the domain, could be $\mathcal{G}(Italians) \in \mathbb{R}^4$ and $\mathcal{G}(Dutch) \in \mathbb{R}^4$ or $\mathcal{G}(People) \in \mathbb{R}^4$.

To explain how LTNs can be combined with a neural network for an end-to-end trainable system, it is possible to consider an example of binary classification into two classes A and B, with training examples defined in a space $\mathbb{R}^5$. It is possible to define the domain of examples and the variables p_A and p_B, which represent examples for classes A and B, respectively, as $\mathcal{D}(p_A) = \mathcal{D}(p_B) = \mathcal{D}(x) = \text{examples}$.

Moreover, it is possible to define the labels for the classes A and B as l_A and l_B, respectively, and the classification predicate as $\texttt{isOfClass}(x,l)$. The predicate $\mathcal{G}_\theta(\texttt{isOfClass}) : x,l \implies [0,1]$, yields the likelihood that the example x belongs to the class l. This makes it possible to learn a grounding function $\mathcal{G}_\theta$ depending on the parameters θ. Following the groundings of terms:

$$\mathcal{G}_\theta(examples) = \mathbb{R}^5, \quad (2.1)$$
$$\mathcal{G}_\theta(labels_A) = [0,1], \quad (2.2)$$
$$\mathcal{G}_\theta(labels_B) = [1,0], \quad (2.3)$$
$$\mathcal{G}_\theta(x_A) \in \mathbb{R}^{n_A \times 5}, \quad (2.4)$$
$$\mathcal{G}_\theta(x_B) \in \mathbb{R}^{n_B \times 5}, \quad (2.5)$$
$$\mathcal{G}_\theta(x) = \text{concat}(\mathcal{G}_\theta(x_A), \mathcal{G}_\theta(x_B)), \quad (2.6)$$
$$\mathcal{G}_\theta(l_A) = [1,0]^T, \quad (2.7)$$
$$\mathcal{G}_\theta(l_B) = [0,1]^T, \quad (2.8)$$
$$\mathcal{G}_\theta(isOfClass) : (x,l) \mapsto l^T \cdot \text{softmax}(\text{MLP}_\theta(x)), \quad (2.9)$$

where MLP is a possible example of a trainable classifier based on the parameters θ that returns a scalar value in the range $[0,1]$.

In conventional LTNs [35, 44, 45], predicates are typically defined as the generalization of the neural tensor network:

$$\mathcal{G}(\mathcal{P})(\mathbf{v}) = \sigma\left(u_P^T \tanh\left(\mathbf{v_T} W_P^{[1:k]} \mathbf{v} + V_P \mathbf{v} + b_p\right)\right) \quad (2.10)$$

where σ is the sigmoid function, $W[1:k] \in \mathbb{R}^{k \times mn \times mn}$, $V_p \in \mathbb{R}^{k \times mn}$, $u_p \in \mathbb{R}^k$, and $b_p \in \mathbb{R}$ are learnable tensors of parameters. In multi-class scenarios, the sigmoid function can be replaced with a softmax layer to ensure mutual exclusivity [40]. Nonetheless, the LTN specification is independent of the specific implementation of the predicates [40].

In this way, the problem reduces to an optimization problem, i.e., finding the optimal parameters θ^* to satisfy the knowledge base $\mathcal{K}$. A useful knowledge base for the proposed classification example can be composed as follows:

$$\phi_1 = \forall x_A \forall l_A (\texttt{Diag}(x_A, l_A) : l_A == 1, \texttt{isOfClass}(x_A, l_A)), \quad (2.11)$$
$$\phi_2 = \forall x_B \forall l_B (\texttt{Diag}(x_B, l_B) : l_B == 1, \texttt{isOfClass}(x_B, l_B)). \quad (2.12)$$

where x_A and l_A represent labeled examples from set A, and similarly, x_B and l_B denote labeled examples from set B.

Diagonal quantification (`Diag`) refers to a form of quantification where the quantifier applies to a formula in such a way that the variable being quantified over appears in both the domain of the quantification and the quantified formula itself. This creates a self-referential or "diagonal" structure in the logical expression.

Guarded quantifiers ($l_A == 1, l_B == 1$) are a type of quantifier used in logical formulas where the variables being quantified are restricted by a "guard" condition. This guard is typically an atomic formula or a conjunction of atomic formulas that restricts the range of variables. Guarded quantifiers help maintain desirable properties like decidability and tractability in logical systems.

Finding the solution to the optimization problem ideally involves perfectly satisfying the knowledge base $\mathcal{K} = \phi_1, \phi_2$ based on the parameter θ:

$$\mathcal{G}_\theta(\phi_1) = \mathcal{G}_\theta(\forall x_A \forall l_A (\texttt{Diag}(x_A, l_A) : l_A == 1, \texttt{isOfClass}(x_A, l_A) = 1)), \quad (2.13)$$

$$\mathcal{G}_\theta(\phi_2) = \mathcal{G}_\theta(\forall x_B \forall l_B (\texttt{Diag}(x_B, l_B) : l_B == 1, \texttt{isOfClass}(x_B, l_B) = 1)). \quad (2.14)$$

The described system is ideal in theory and could be impossible in the case, e.g., of noisy labels. Therefore, θ^* is defined as a reasonable satisfaction of the knowledge base, i.e., the optimization problem is to find:

$$\theta^* \text{ s.t. } \mathcal{G}_{\theta^*}(\phi) \approx 1 \quad \forall \phi \in \mathcal{K}. \quad (2.15)$$

To obtain a single value to optimize, an aggregation functions that allow obtaining a probability value of the overall knowledge base *SatAgg* : $[0,1]^* \implies [0,1]$ is defined, solving the following equation:

$$\max_{\theta \in \theta} \text{SatAgg}_{\phi \in \mathcal{K}} \mathcal{G}_\theta(\phi). \quad (2.16)$$

The *best satisfiability* problem, the optimization problem underlying LTNs, consists of determining the values of θ^* that maximize the truth values of the conjunction of all formulas $\phi \in \mathcal{K}$:

$$\theta^* = argmax_\theta \left(\hat{\mathcal{G}}_\theta \left(\bigwedge_{\phi \in \mathcal{K}} \phi \right) - \lambda ||\theta||_2^2 \right) \quad (2.17)$$

where $\lambda||\theta||_2^2$ is a convenient regularization term.

2.3 Fuzzy logic operators

To facilitate the understanding of the following sections, essential background details are provided. Relevant terminology is defined by drawing on established works in the field, specifically those of Ying et al. [46], van Krieken et al. [47], and Calvo et al [48].

2.3.1 Propositional connectives

Consider $p_1, ..., p_n$ to be n propositional variables, all propositional formulas generated from $p_1, ..., p_n$ are defined as follows:

- $p_1, ..., p_n$ are propositional formulas;
- if ϕ, μ are propositional formulas, then $\neg\phi, \phi \wedge \mu$, and $\phi \vee \mu$ are also propositional formulas.

Therefore, $P(p_1, \ldots, p_n)$ is a composition of operators $\{\neg, \wedge, \vee\}$ generated with $p_1, \ldots, p_n$. The set of all propositional formulae formed from $p_1, \ldots, p_n$ is designated as $PL(p_1, \ldots, p_n)$. Specifically, it may be observed that, in classical propositional logic, propositional variables are interpreted in $\{0, 1\}$; in fuzzy logic, they are interpreted in the entire interval $[0, 1]$.

Fuzzy negation. A mapping of a function N from $[0, 1]$ to itself that satisfies $N(N(a)) = a$ for every $a \in [0, 1]$, where $a \leq b$, i.e., $N(b) \leq N(a)$. This mapping is known as a negation operator, the negation operator in fuzzy logic is a truth function of negation where for each $a \in [0, 1]$, the negation operator is represented as $N(a) = 1 - a$, other definitions of negation are defined in [47].

Triangular norm and conorms. A binary operator in the range $[0, 1]$ that is commutative, associative, non-decreasing, and has 1 (respectively, 0) as its identity element is called a *T-norm* T (and, conversely, *T-conorm* C). In fuzzy logic, the terms *T-norm* and *T-conorm* denote the truth functions for conjunction and disjunction, respectively. Table 2.1 and Table 2.2 show the main operations that were used for computing the *T-norm* and *T-conorm* functions, respectively, along with their corresponding properties.

In particular, a *T-norm* must be:

1. *Monotonic*: For every $a \in [0,1]$, $T(a,\cdot)$ is increasing, i.e., if $0 \leq b_1 \leq b_2 \leq 1$, then $T(a,b_1) \leq T(a,b_1) \leq T(a,b_2)$.
2. *Neutral*: For every $a \in [0,1]$, $T(1,a) = a$.
3. *Left-continuous*: A left-continuous *T-norm* is left-continuous in both arguments.
4. *Idempotent*: An idempotent *T-norm* has the property that for every $a \in [0,1]$, $T(a,a) = a$.
5. *Strictly monotonic*: A strictly monotonic *T-norm* has the property that for every $a \in (0,1]$, $T(a,\cdot)$ is strictly increasing.

Name	T-norm	Properties
Gödel (minimum)	$T_G(a,b) = \min(a,b)$	idempotent, continuous
Product	$T_P(a,b) = a \cdot b$	strict
Łukasiewicz	$T_{LK}(a,b) = \max(a+b-1,0)$	continuous
Drastic product	$T_D(a,b) = \begin{cases} \min(a,b), & \text{if } a = 1 \text{ or } b = 1 \\ 0, & \text{otherwise} \end{cases}$	
Nilpotent minimum	$T_{NM}(a,b) = \begin{cases} 0, & \text{if } a+b \leq 1 \\ \min(a,b), & \text{otherwise} \end{cases}$	left-continuous
Yager	$T_Y(a,b) = \max\left(1-((1-a)^p+(1-b)^p)^{\frac{1}{p}},0\right), p \geq 1$	continuous

Table 2.1 T-norms

A *T-conorm* must satisfy the properties of being commutative, associative, and:

1. *Monotonic*: For every $a \in [0,1]$, the function $T(a,\cdot)$ is non-decreasing. Specifically, if $0 \leq b_1 \leq b_2 \leq 1$, then $T(a,b_1) \leq T(a,b_2)$ holds.
2. *Neutral*: For every $a \in [0,1]$, the condition $T(0,a) = a$ must be satisfied.

Name	T-conorm	Properties
Gödel (maximum)	$S_G(a,b) = \max(a,b)$	idempotent, continuous
Product (probabilistic sum)	$S_P(a,b) = a + b - a \cdot b$	strict
Łukasiewicz	$S_{LK}(a,b) = \min(a+b,1)$	continuous
Drastic sum	$S_D(a,b) = \begin{cases} \max(a,b), & \text{if } a = 0 \text{ or } b = 0 \\ 1, & \text{otherwise} \end{cases}$	
Nilpotent maximum	$S_{NM}(a,b) = \begin{cases} 1, & \text{if } a+b \geq 1 \\ \max(a,b), & \text{otherwise} \end{cases}$	right-continuous
Yager	$S_Y(a,b) = \min\left((a^p + b^p)^{\frac{1}{p}}, 1\right), p \geq 1$	continuous

Table 2.2 T-conorms

The *T-conorm* is still based on the De Morgan laws of classical logic, which assert that $p \vee q = \neg(\neg p \wedge \neg q)$. Specifically, if T is a *T-norm*, then the function S can be expressed as $S(a,b) = 1 - T(1-a, 1-b)$. However, for the case where the neutral element of the T-conorm is 0, this relation still holds, but the neutral element reflects that the disjunction operation has 0 as its identity.

Implications. A fuzzy implication is a function $I : [0,1]^2 \to [0,1]$ with the following properties: For any $a, c \in [0,1]$, the function $I(\cdot, c)$ is decreasing with respect to its first argument, and $I(a, \cdot)$ is increasing with respect to its second argument. Additionally, the fuzzy implication must satisfy the conditions: $I(0,0) = 1$, $I(1,1) = 1$, and $I(1,0) = 0$. From these conditions, it can be deduced that $I(0,1) = 1$.

S-Implications. In classical logic, the implication is defined as follows: $p \implies q = \neg p \vee q$. The properties of *left-neutrality*, *the exchange principle*, and *N-right-contrapositive symmetry* are satisfied by an implication. The implication is also characterized as a strong implication and meets the property of *N-contrapositive symmetry* if N is a strong negation. Furthermore, it fulfills *N-left-contrapositive symmetry* if it is also strong. The attributes and potential methods of implementing the implications are explicitly shown in Table 2.3.

Name	T-conorm	S-implication	Properties
Kleene-Dienes	S_G	$I_{KD}(a,c) = \max(1-a,c)$	All but IP
Reichenbach	S_P	$I_{RC}(a,c) = 1-a+a\cdot c$	All but IP
Lukasiewicz	S_{LK}	$I_{LK}(a,c) = \min(1-a+c,1)$	All
Dubouis-Prade	S_D	$I_{Dp}(a,c) = \begin{cases} c, & \text{if } a=1 \\ 1-a, & \text{if } c=0 \\ 1, & \text{otherwise} \end{cases}$	All
Fodor	S_{Nm}	$I_{FD}(a,c) = \begin{cases} 1, & \text{if } a \leq c \\ \max(1-a,c), & \text{otherwise} \end{cases}$	

Table 2.3 Properties and implementations of various S-implications. The table outlines different S-implications, their corresponding T-conorms, and the specific logical properties that they satisfy.

2.3.2 Aggregation operators

The functions that combine several inputs into one are called *aggregation operators*, and are used to compute quantifiers such as $\forall, \exists$.

An aggregation operator is a function $A : \cup_{n\in\mathbb{N}}[0,1]^n \to [0,1]$ that is nondecreasing with respect to each argument, and for which $A(0,\dots,0) = 0$ and $A(1,\dots,1) = 1$. Table 2.4 shows some common aggregation operators that have been proposed in the literature. Different aggregation operators exhibit distinct properties concerning differentiability, which is crucial for gradient-based optimization methods commonly employed in training neuro-symbolic models. For instance, as highlighted by van Krieken et al. [47], some fuzzy implications, such as the Reichenbach and Łukasiewicz implications, have derivatives that align well with gradient descent, making them suitable for use in differentiable learning setting.

The $\forall$ quantifier is often interpreted as the conjunction over all arguments p. Therefore, it is possible to extend a T-norm T from 2-dimensional inputs to n-dimensional inputs as they are commutative and associative:

$$A_T() = 0 \tag{2.18}$$

$$A_T(p_1, x_2, \dots, p_n) = T\left(p_1, A_T(x_2, \dots, p_n)\right) \tag{2.19}$$

These operators are a straightforward choice for modeling the $\forall$ quantifier, as they can be seen as a series of conjunctions. All operators constructed in this way are symmetric aggregation operators for which the output value is the same for every ordering of their arguments and have both commutative and associative properties.

Formally, given a fuzzy predicate $PL(p)$, the fuzzy universal quantifier is defined using a T-norm T (which generalizes the AND operation):

$$\forall p\, PL(p) = \inf_{x \in X} PL(p) \tag{2.20}$$

where inf (infimum) represents the greatest lower bound of $PL(p)$ over the domain P.

On the other hand, the aggregator $\exists$ is understood to represent a disjunction of its parameters. The existential quantifier in fuzzy logic can be interpreted as the degree to which there exists at least one element in the domain for which the property is true. Formally, given a fuzzy predicate $PL(p)$, the fuzzy existential quantifier is defined using a T-conorm S (which generalizes the OR operation):

$$\exists p\, PL(p) = \sup_{p \in P} PL(p)$$

where sup (supremum) represents the least upper bound of $PL(p)$ over the domain P.

The aggregation of truth values for fuzzy quantifiers is achieved through the use of T-norms and T-conorms. For the universal quantifier ($\forall$), the infimum of the truth values of the fuzzy predicate over all elements in the domain is used. For example, for a fuzzy predicate $PL(p)$ over a domain P, $\forall p\, PL(p)$ is the minimum of the values $PL(p)$ over P.

Similarly, for the existential quantifier ($\exists$), the supremum of the truth values of the fuzzy predicate over all elements in the domain is used. For example, for a fuzzy predicate $PL(p)$ over a domain P, $\exists p\, PL(p)$ is the maximum of the values $PL(p)$ over P.

Two different aggregation functions are proposed by Badreddine et al. [40]: the generalized mean A_{pM} approximates the existential quantifier $\exists$, whereas the generalized mean with respect to the error A_{pME} approximates the universal quantifier $\forall$ [40, 49]. Considering n truth values in $[0,1]$, such as $a_1,\ldots,a_n$:

Name	Generalizes	Aggregation operator
Minimum	T_G	$A_{T_G}(p_1,\ldots,p_n) = \min(p_1,\ldots,p_n)$
Product	T_P	$A_{T_P}(p_1,\ldots,p_n) = \prod_{i=1}^{n} x_i$
Lukasiewicz	T_{LK}	$A_{T_{LK}}(p_1,\ldots,p_n) = \max(\sum_{i=1}^{n} x_i - (n-1), 0)$
Maximum	S_G	$E_{S_G}(p_1,\ldots,p_n) = \max(p_1,\ldots,p_n)$
Probabilistic sum	S_G	$E_{S_P}(p_1,\ldots,p_n) = 1 - \prod_{i=1}^{n}(1-x_i)$
Bounded sum	S_{LK}	$E_{S_{LK}}(p_1,\ldots,p_n) = \min(\sum_{i=1}^{n} x_i, 1)$

Table 2.4 Common aggregation operators

$$\exists : A_{pM}(a_1,\ldots,a_n) = \left(\frac{1}{n}\sum_{i=1}^{n} a_i^{p_\exists}\right)^{\frac{1}{p_\exists}} \qquad p_\exists \geqslant 1 \tag{2.21}$$

$$\forall : A_{pME}(a_1,\ldots,a_n) = 1 - \left(\frac{1}{n}\sum_{i=1}^{n} (1-a_i)^{p_\forall}\right)^{\frac{1}{p_\forall}} \qquad p_\forall \geqslant 1 \tag{2.22}$$

However, there are alternatives to these common operators. For example, in this thesis, the log-product aggregator used in Faster-LTN [35] is defined as:

$$\forall : A_{log_{product}}(a_1,\ldots,a_n) = \sum_{i=0}^{N} \alpha_c (1-a_i)^{\gamma} log(a_i) \tag{2.23}$$

where a_i is the literal of the i-th class and α_c is a class-dependent coefficient. Additionally, γ increases the contribution of literals with low truth value (e.g., misclassified examples).

By carefully selecting appropriate aggregation operators with desirable differentiability characteristics, it is possible to enhance the training efficiency and performance of neuro-symbolic systems, leveraging the strengths of both symbolic logic and neural networks. This approach ensures that the learning process remains stable and effective, even when integrating complex background knowledge [47].

Chapter 3

Related work

NeSy architectures, an emerging paradigm that synergizes the strengths of neural networks and symbolic reasoning, have shown remarkable potential in various domains of artificial intelligence. By integrating the data-driven learning capabilities of neural networks with the rule-based manipulation of symbolic systems, neurosymbolic architectures aim to enhance generalization and reasoning in AI systems. This chapter explores the integration of neuro-symbolic techniques for object detection and the principles of zero-shot learning (ZSL), as these tasks were selected due to their critical importance in advancing real-world applications and their unique challenges in the field of computer vision. We will highlight the challenges and solutions in training these models.

Object Detection is a computer vision task that involves identifying and localizing objects within an image. Traditional methods often rely on CNNs to extract features from images, which are then processed to classify and delineate objects. Recent advancements have incorporated symbolic reasoning to improve the robustness of object detection systems.

Zero-Shot Learning is a machine learning approach where a model is trained to recognize objects from unseen categories using semantic information that relates seen and unseen classes. This allows the model to generalize to new categories without explicit training examples, leveraging attributes or descriptions to bridge the gap between known and unknown classes.

3.1 Neuro-symbolic approaches for object detection

A natural image, composed of scenes, objects, and parts interconnected by a complex web of spatial and semantic relationships, necessitates the development of semantic image interpretation components that recognize a hierarchy of elements while incorporating enduring visual perception and the ability to encode and reason about visual relationships. Various methods have been proposed to enhance CNNs with capabilities for relationship representation and reasoning, including Relational Networks [50], Graph Neural Networks [51], and NeSy techniques [41, 42, 52]. For a broader introduction to NeSy techniques it is possible to consult recent surveys [53, 23].

Several recent methods employ CNNs to extract features, which are then processed by a symbolic or neuro-symbolic module [41, 42, 54, 55]. For example, Yuke Zhu et al. [54] utilize a Markov Logic Network (MLN) to manage textual data along with corresponding visual features, using a knowledge base to represent the relations between objects based on their visual, physical, and categorical properties. Kenneth Marino et al. [56] integrate a Graph Search Neural Network (GSNN) with a classification network. Donatello et al. [41] and Cewu Lu et al. [55] apply visual features in training LTNs for detecting visual relationships in the form of subject-verb-object triplets or part of relations. These studies illustrate how NeSy techniques facilitate the formulation of logical axioms that act as high-level inductive biases, guiding the network towards an optimal solution in harmony with these inductive biases. However, in the aforementioned studies, feature extraction and classification networks are trained independently, which prevents the CNNs from using these supplementary inductive biases during training.

One of the significant challenges in object detection is dealing with data imbalance, where the number of background examples far exceeds the number of foreground objects. This imbalance can cause models to be biased towards predicting the majority class (background), leading to suboptimal performance in detecting objects. Traditional methods to address data imbalance include hard example mining and re-sampling techniques, which focus on re-balancing the training dataset. Recent advancements have introduced various loss functions designed to address the issue of data imbalance in object detection. Focal loss reduces the loss of well-classified examples, focusing on hard ones to prevent easy negatives from dominating the detector, effective in models like RetinaNet [57]. Balanced cross entropy uses a weighting factor to balance positive and negative examples, addressing class imbalance by their frequencies [58]. Class-balanced loss adjusts the contribution based on the number of samples per class, giving more weight to underrepresented classes and addressing long-tailed distributions [59].

However, other practical obstacles in training NeSy architectures include the well-known issue of scalability when managing large datasets, as highlighted in symbolic AI literature [47]. Consequently, many NeSy architectures use a traditional object detector to generate an initial set of candidate objects [41], thereby ignoring the background effect and simplifying the problem's scale. Another factor related to scalability is the selection of aggregation functions and fuzzy logic operators. Emile van Krieken et al. [47] and Samy Badreddine et al. [40] identified significant differences among differential fuzzy logic operators regarding computational efficiency, scalability, gradients, and their capability to handle exceptions, all of which are important in a learning context. Their analysis provides a foundation for the Faster-LTN (Chapter 4) architecture, which incorporates and improves the log-product aggregator discussed in [47].

3.2 Introduction to Zero-Shot Learning

ZSL is a machine learning approach where a model is trained to identify and categorize objects from new, unseen categories by using additional information like semantic attributes, which connect unseen categories with the seen categories from training data. GZSL broadens the ZSL approach to identify and categorize objects from both seen and unseen categories. Unlike ZSL, GZSL demands that the model adeptly manage a more complex scenario by differentiating between seen and unseen classes within the same space. Several methods have been devised for ZSL, including those founded on embeddings, end-to-end embeddings, generative models, and transformer-based models.

Embedding-based. These techniques combine semantic attributes and visual data by mapping them into a unified space. Some methods map images into the attribute space via an embedding function, considering semantic attributes as the shared space [60, 61]. On the other hand, other techniques use the image embedding space as the common platform to solve the hubness issue [62, 63]. Additionally, there are approaches [64–66] that create a distinct shared space that is separate from both the image and attribute domains. To prevent overfitting to known categories, these techniques often utilize pseudo-labeling methods or operate in a transductive manner [65], using unlabeled images from new categories during training. In Chapter 5, Proto-LTN will be discussed, an architecture that builds upon this type of approach, leveraging the embedding-based method to enhance performance and address specific challenges in ZSL.

End-to-end embeddings. These approaches, based on embeddings, have been employed to identify the crucial regions of an image for the purpose of class categorization and to improve the embedding space during training [60, 61, 67, 68]. Previous research has utilized attributes to create class-level representations through regularization methods [60, 65] or contrastive techniques [61] to prevent overfitting of seen classes. FLPN and FLVN belong to this category of work. Chapter 6 will introduce FLVN, which translates images into attribute space, learning the representation of classes at the attribute level. This capability has been empirically shown to boost classification accuracy. Building on past studies [69, 70], Chapter 7 will present FLPN, an architecture that integrates attribute and class representations to leverage the reasoning skills of a NeSy network, enhancing performance through the use of semantic descriptions and an attention mechanism combined with a prototypical network. Specifically, by using semantic descriptions of classes, combined effectively with an attention mechanism and a prototypical network, the framework learns representations of classes and attributes efficiently, improving the model's ability to generalize. In addition, it introduces a symbolic prior that groups visually and semantically similar features.

Generative methods. These methods employ auxiliary models, such as Generative Adversarial Networks (GANs), to produce synthetic examples that represent unseen classes by learning the conditional probability distribution for each class [71–74]. Recently, feature generation models have been integrated with embedding-based models within a contrastive framework [75]. Generative approaches necessitate prior knowledge of unseen classes to create training data.

Transformer-based models have been found to be particularly effective in achieving outstanding results across a variety of natural language processing tasks [76]. These architectures demonstrate enhanced robustness compared to CNNs [77]. Du et al. [69] suggested that it is possible to use global features for embedding spaces and to apply ViT embedding patches as an attention mechanism to capture attribute-level embedding features. Applying this approach in FLPN, advantages in a NeSy context are highlighted.

3.2.1 Prototypical networks for Zero-Shot Learning

Prototypical networks are increasingly popular in ZSL due to their ability to classify both seen and unseen classes effectively. These networks represent each class with a prototype in an embedding space, which is usually generated by averaging the embeddings of the class's training samples. To improve ZSL performance, various studies suggest incorporating semantic similarities between classes. The authors of [62] discuss utilizing deep neural networks to transform images into embedding spaces. In [36] and [37], it is emphasized the role of semantic similarities in enhancing classification accuracy. One of the challenges in applying prototypical networks in ZSL is the **hubness problem**, where certain points in the feature space (hubs) connect too closely with numerous other points, negatively impacting class prototypes and hindering the model's ability to generalize, as noted in [62]. Proto-LTN addresses the hubness problem by using a shared embedding space for images, which helps in distributing connections more evenly. Additionally, in FLPN, utilizing prototypical networks with shared weights allows for the representation of both individual attributes and classes.

Chapter 4

Faster-LTN: neuro-symbolic object detection architecture

Work described in this chapter was originally presented in [35].

This chapter presents Faster-LTN, an object detector that combines Faster R-CNN with an LTN, in contrast to other efforts that relied on pre-trained CNNs [42]. Notably, both the backbone and LTN undergo joint end-to-end training. The convolutional layers are trained with the help of LTN-specific logical constraints, which reduces the reliance on labeled data. Experiments conducted on the PASCAL VOC and PASCAL PART datasets confirm the feasibility of Faster-LTN by demonstrating performance comparable to neural frameworks. Keras's implementation of Faster-LTN is available at https://gitlab.com/grains2/Faster-LTN. This chapter provides a detailed description of the Faster-LTN architecture and its training process (Section 4.1), of the experiments conducted (Section 4.2) and finally the results (Section 4.3) obtained.

4.1 Architecture

The architectures presented integrate a traditional Faster R-CNN object detection model with an LTN. Faster R-CNN provides region proposals and feature extraction, while the LTN applies logical constraints and relations to enhance object classification and bounding-box predictions, creating a robust, logic-guided object detection framework.

4.1.1 Faster R-CNN

A Region Proposal Network (RPN) and a classification network with a shared backbone constitute Faster R-CNN, a two-phase object detector [1]. A regression layer calculates the bounding box coordinates, while the RPN produces a binary classification label (background vs. foreground) for each anchor. The ROI Pooling layer receives the Regions of Interest (ROIs) chosen by the RPN and uses them to extract and resize the features of each proposal bounding box from the shared backbone. The classifier receives feature maps of equal size. The classifier consists of two convolutional heads: a regression layer with linear activation that computes the bounding box and a classification layer with softmax activation that computes the final object categorization.

The training process for the RPN and classifier heads is conducted alternately in an end-to-end manner. Initially, the RPN is trained and updated with each forward pass, during which all region proposals are extracted from the image. Following this, the detector head is updated while the RPN weights are kept constant. For the training of the classifier head, a specified number of positive (object) and negative (background) instances are selected at each stage.

The loss function encompasses both regression and classification losses:

$$\mathcal{L}(p_i, b_i) = \frac{1}{n_c} \sum_i \mathcal{L}_{cls}(p_i, p_i') + \lambda \frac{1}{n_r} \sum_i p_{i^*} \mathcal{L}_{reg}(b_i, b_i') \tag{4.1}$$

where the first part of the equation $\frac{1}{n_c} \sum_i \mathcal{L}_{cls}(p_i, p_i')$ represents the classification loss and the second part, $\lambda \frac{1}{n_r} \sum_i p_{i^*} \mathcal{L}_{reg}(b_i, b_i')$ denotes the regression loss. In the Faster-LTN, the RPN module remains unchanged, while the classifier head is replaced with an LTN.

4.1.2 Logic tensor network for object detection

A grounded theory for the detection of objects

Assuming a collection of bounding boxes $b \in \mathcal{B}$, where $c \in \mathcal{C}$ represents a recognized class, the vector grounding an item with bounding box b_n is:

$$\mathbf{v_{b_n}} = < \mathbf{z}_{bn}, b_n > \tag{4.2}$$

where $\mathbf{z}_{b_n} = f(I, b_n)$ is an embedding feature vector, calculated by a convolutional neural network f, given an image I and the bounding box coordinates b_n predicted by the RPN layer. This makes it possible to successfully connect the convolutional layers and the LTN, which is a departure from previous work [41], where the grounding of a bounding box was specified by the probability vector predicted by a pre-trained Faster R-CNN.

The `isA` predicate for class $c \in \mathcal{C}$ is defined as a real vector, defined as in Equation 2.10 implementing a one-vs-all classifier.

Notably, unlike [41], the `isA` predicate takes only the real vectors $\mathbf{z}_{bn}$ as input, excluding the bounding box coordinates. This preserves a fundamental property of convolutional object detectors such as equivariance to shifting. The grounding $f(I, b_n)$ is defined as the result of the final fully connected layer of the classifier head. Pairs of bounding boxes define the predicate `partOf` [41]. The vectors b_m and b_l, representing two generic bounding boxes, are grounded by:

$$\mathbf{v_{b_{m,l}}} = < \mathbf{z}_{bm}, b_m, \mathbf{z}_{bl}, b_l, ir_{m,l} > \tag{4.3}$$

where $ir_{m,l}$ is the *inclusion ratio* from Donadelloet al. [45] which express the overlapping between two bounding boxes, defined as:

$$ir_{m,l} = \frac{Area\,(b_m \cap b_l)}{Area\,(b_m)} \tag{4.4}$$

The grounding $\mathcal{G}\,(\texttt{partOf})\,(\mathbf{v_{b_{m,l}}})$ is a neural tensor network defined as in Equation 2.10.

Constructing a theory from labeled examples

Herewith it is explored how a grounded theory is formulated to address the best satisfiability problem outlined in Equation 2.17 for object detection. Following the approach in [41], different grounded theories, $\mathcal{T}_{expl}$ and $\mathcal{T}_{prior}$, are introduced.

In essence, the former, $\mathcal{T}_{expl}$, replicates the traditional learning-by-example setting by aggregating all the axioms obtained from the labeled training set. In contrast, *logical* and *mereological* constraints are introduced by the theory $\mathcal{T}_{prior}$, which stands for previous knowledge or, more broadly, desirable qualities of the ultimate solution.

In this chapter, two types of constraints are defined. The first type, *mutual exclusion* , is captured by the following formula:

$$\forall x(P_1(x) \implies (\neg P_2(x) \wedge \ldots \wedge \neg P_n(x))) \tag{4.5}$$

To encode that an item belonging to class c_i cannot belong to class c_j, Equation 4.5 is translated into $K(K-1))/2$ clauses, corresponding to all unordered class pairs over K classes, e.g., $\texttt{Cat}(x) \implies \neg\texttt{Person}(x)$.

Based on an existing ontology (e.g., Wordnet) that contains *meronimy* (i.e., *part-whole*) relationships, *mereological constraints* are imposed on the `partOf` and `isA` predicates. Specific axioms are introduced to indicate that each *whole* is typically associated with a set of given *parts*, that a *whole* object cannot include another *whole* object, and that a *part* cannot include another *part*. Examples of such axioms can be seen in autorefeq:parts1 and Equation 4.7:

$$\forall x, y\,(\texttt{Cat}(x) \wedge \texttt{partOf}(y, x) \rightarrow \texttt{Tail}\,(y) \vee \texttt{Head}\,(y) \ldots \vee \texttt{Eye}\,(y)) \tag{4.6}$$

to indicate that an object y that is part of cat x should be classified as a head, tail, or eye.

Conversely, the following axiom indicate that a certain part (for instance, a tail) must belong to a consistent object (in this case, an animal):

$$\forall x, y\,(\texttt{Tail}(x) \wedge \texttt{partOf}(y, x) \rightarrow \texttt{Cat}\,(y) \vee \texttt{Dog}\,(y) \ldots \vee \texttt{Horse}\,(y)) \tag{4.7}$$

Mereological limitations were imposed using the knowledge base created in [41] shown in Table 4.1.

Objects	Parts
Airplane	body, engine, wing, tail, wheel
Bicycle	wheel, handlebar, headlight, saddle
Bird	beak, head, eye, foot, leg, wing, neck, tail, torso
Boat	hull, deck, mast, sail, rudder
Bottle	body, cap
Bus	license plate, side, door, headlight, mirror, wheel, window
Car	license plate, side, door, headlight, mirror, wheel, window
Cat	head, leg, paw, ear, eye, neck, nose, tail, torso
Chair	backrest, seat, legs, armrests
Cow	head, leg, ear, eye, horn, muzzle, neck, tail, torso
Dining table	tabletop, legs
Dog	head, leg, ear, eye, neck, nose, tail, torso
Horse	head, leg, ear, eye, mane, neck, tail, torso
Motorbike	wheel, handlebar, headlight, saddle
Person	hair, head, ear, eye, eyebrow, foot, hand, leg, arm, mouth, neck, nose, torso
Potted plant	plant, pot
Sheep	head, leg, ear, eye, horn, muzzle, neck, tail, torso
Sofa	cushions, armrests, backrest
Train	coach, locomotive, headlight
TV monitor	screen, stand

Table 4.1 Objects and their parts present in the PASCAL PART dataset

4.1.3 Faster-LTN

Figure 4.1 presents an overview of the complete architecture. The whole structure is an end-to-end system that connects an LTN and a Faster R-CNN. In particular, the output of the traditional Faster R-CNN architecture is sent to the LTN, and the softmax activation is removed from the classifier head. During training, a partial grounded theory is built for each batch according to Table 4.1. The three additional *Predicate*, *Literal*, and *Clause* layers are defined to implement the LTN.

A *predicate* in logic represents a property or relation among objects, mapping them to a truth value in [0, 1] in fuzzy logic. Formally, a predicate PL is defined as $PL : p_1 \times p_2 \times \cdots \times p_n \to [0,1]$, with p_i being domains of objects. A *literal* is an atomic formula or its negation, reflecting the fuzzy complement in fuzzy logic. If p is a predicate, p and $\neg p$ are literals, with $\neg p$ as the complement of p. In fuzzy

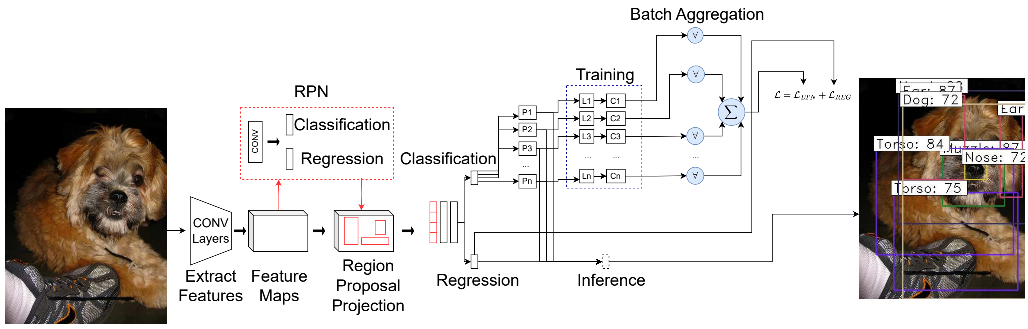


Fig. 4.1 Faster-LTN architecture. It shares the first part of its architecture, up to the RPN, with the Faster R-CNN network [1]. The backbone extracts, concatenates, and inputs the feature maps linked to the RPN proposals to the LTN, which consists of a set of predicates P_i, each of which corresponds to a distinct class. At training time, a partial theory $\mathcal{T}_{expl}$ is defined using a batch of labelled examples from the training dataset. A positive or negative literal (L) for the appropriate predicates relates to each positive or negative example. The optimal grounding $\mathcal{G}^*$ is found by maximizing the truth value of the aggregated clauses (C). The truth value of the predicates P_i is calculated at inference time.

logic, a *clause* is a fuzzy disjunction of fuzzy literals, which can be represented as $C = L_1 \vee L_2 \vee \cdots \vee L_m$ where L_i are literals. The matching literal calculates the truth value of all positive (i.e., class c) and negative (i.e., non-class c) examples for each class c. The clause layer uses the chosen aggregation function to aggregate all literals for a specific class. Furthermore, clauses that accept multiple literals as input can be defined (for example, for `partOf` predicates). Figure 4.1 only displays $\mathcal{T}_{expl}$ for simplicity. The LTN's complete loss can be calculated by adding the regression loss to $\mathcal{L}_{LTN}$, much like the loss of the RPN layer.

Training

In order to address memory limitations, for every batch of instances, a partial $\mathcal{T}_{expl}$ must be rebuilt. The LTN was trained using the predictions of a pre-trained object detector in previous works [41], enabling a reasonably large batch size. In an end-to-end training setup, where the memory required to store feature maps and related gradients is much higher, a batch is created starting from a single image, and including both background and foreground instances. The LTN is thus trained on all proposals that are retrieved by the RPN. It is important to note that even in training batches with an equal number of objects and background proposals, the one-vs-all classification exacerbates the data imbalance between positive and negative cases for each class.

Aggregation function

The log-product is the selected aggregator function because it scales well with the number of inputs, as demonstrated in [47], and because its formulation is equivalent to the cross-entropy loss. Motivated by [78], the focal log-product aggregator is introduced and detailed as follows:

$$\mathcal{L}_{LTN} = -\sum_{j=0}^{K}\sum_{i=0}^{N}\alpha_c(1-x_{i,j})^{\gamma}log\left(x_{i,j}\right) \tag{4.8}$$

where $x_{i,j}$ is the literal of the i-th ROI in the j-th class, K is the number of classes, and N is the batch size. Additionally, γ increases the contribution of literals with low truth value (i.e., misclassified cases).

The focal loss was introduced to address the issue of class imbalance during training, particularly in the context of object detection. The traditional cross-entropy loss tends to be overwhelmed by the numerous easy, well-classified examples, which can dominate the gradient and hinder the learning process for harder, misclassified examples. The focal loss modifies the standard cross-entropy loss by adding a modulating factor $(1-p_t)^{\gamma}$, where p_t is the predicted probability of the true class. This factor reduces the loss contribution from well-classified examples, thereby focusing more on hard examples. The parameter γ controls the rate at which easy examples are down-weighted. When $\gamma = 0$, the focal loss is equivalent to the cross-entropy loss. As γ increases, the effect of the modulating factor also increases, allowing the model to focus more on difficult, misclassified examples. This approach has been shown to improve the performance of object detectors, making them more accurate and robust in the presence of class imbalance [57].

The number of background examples plus the positive examples that belong to other classes determine the number of negative examples for each training batch and each class c. This observation is all that is needed to determine the value of α_c. For positive and negative instances, the coefficients are set as $\alpha_c = \frac{1-\beta}{1-\beta^{pos_c}}$ and $\alpha_c = \frac{1-\beta}{1-\beta^{neg_c}}$, respectively. Let $p(c)$ represent the proportion of the training set bounding boxes that correspond to class c. Subsequently, $pos_c = \frac{N}{2}p(c)$ and

$neg_c = \frac{N}{2} + \frac{N}{2}(1 - p(c))$ represent the proportion of positive and negative examples, respectively, for a given batch.

The overall loss function for the Faster-LTN model is defined as the sum of the regression loss and the LTN loss, i.e., $\mathcal{L} = \mathcal{L}_{reg} + \mathcal{L}_{LTN}$. This approach allows the integration of logical constraints during training, thereby enhancing the model's ability to learn more complex relationships between classes.

4.2 Experimental setup

Dataset The PASCAL PART [79] and PASCAL VOC 2010 [80] benchmarks were used for the experiments. 39 classes for pieces and 20 classes for entire objects for the latter are chosen. Every experiment is run with an 80:20 split trainval partition. Reducing the training set by 50% is tested by randomly selecting from the PASCAL PART (10K images); as a result, there are approximately 8K images for PASCAL PART and 4K images for PASCAL PART REDUCED.

The **Faster R-CNN** architecture closely resembles that of the original implementation [1]. With aspect ratios of 1:1, 1:2, and 2:1, the anchor scales were set at 128^2, 256^2, and 512^2. The backbone architecture was ResNet50 pretrained on ImageNet. There will be 300 RPN proposals overall. 128 bounding boxes with a 32:96 positive and negative sample ratio for the PASCAL VOC dataset and 32 bounding boxes with a 16:16 ratio for the PASCAL PART were selected at random to train the classifier head. Using the Adam optimizer, the network was trained for 100 epochs. The learning rate was initially set at 10^{-5} for the first 60 epochs and later lowered to 10^{-6}. Data augmentation (horizontal flip) and weight decay (at a rate of 5×10^{-4}) were used to regularize the training process.

The **Faster-LTN** model shared the same architecture as Faster R-CNN, except the LTN's embedded classifier head. Equation 2.10 defines each predicate with $k = 6$ kernels. The disjunction of the literals was encoded using Łukasiewicz's T-norm, the focal log-product with $\gamma = 2$ were chosen as the aggregation with $\beta = 0.999$, both values obtained through experimental validation. For PASCAL VOC, $\mathcal{T}_{\text{prior}}$ contained mutual exclusion constraints; for PASCAL PART experiments, it contained mereological and mutual exclusion constraints. In the latter scenario, the

LTN was extended to include `partOf` predicates; nevertheless, only object detection performance was assessed to compare with Faster R-CNN.

Various experiments with modifications of the focal log-product aggregation function were conducted on the PASCAL VOC dataset: with and without the addition of an additional predicate bg to represent the background class, and with and without class weights α. Faster-LTN, Faster-LTN α, Faster-LTN bg, and Faster-LTN $bg+\alpha$ identify the different experimental settings. In PASCAL PART experiments, the Faster-LTN bg configuration was used. Using the Adam optimizer for 150 epochs, all networks were trained with random horizontal flip, L2 regularization (λ set to 5×10^{-4}) and weight decay (decay rate 5×10^{-4}). For the first 60 epochs, the learning rate was set to 10^{-5}; after that, it was lowered to 10^{-6}.

Every experiment was run on the HPC@Polito cluster equipped with V100 NVIDIA GPUs. The mean Average Precision (MAP), as used in the PASCAL VOC challenge 2010 [81], served as the performance metric.

4.3 Results

The results of the PASCAL VOC experiments, which are presented in Table 4.2, demonstrate that Faster-LTN outperformed the standard Faster R-CNN architecture, achieving an increase in mAP from 62.6 to 73.8. Mutual exclusivity is enforced by including an axiomatic constraint in the LTN knowledge base (Equation 4.5). From a quantitative viewpoint, Faster-LTN outperformed Faster R-CNN in terms of average precision. This difference can be attributed to the use of the focal loss or the different classification setting (K one-vs-all classifiers instead of a single multi-class classifier), given that the log-product aggregation is mathematically equivalent to the cross-entropy loss and the backbone is the same [78]. However, the performance decreased from 62.6 to 59.2 when the focal loss was substituted for the Faster R-CNN classifier head's loss. As a consequence, the improved performance of Faster-LTN can be credited to the added flexibility offered by a more complex classifier head with additional parameters.

For the PASCAL PART experiments, new mereological axioms were inserted in $\mathcal{T}_{prior}$, as indicated in Table 4.3. Performance was able to increase from 35.1 to 41.2 as a result; the performance difference persisted from 28.5 to 32.8 when the size of

Class	FR-CNN	FR-CNN FL	F-LTN	F-LTN α	F-LTN bg	F-LTN bg+α
aeroplane	66.5	56.9	87.1	85.1	87.8	85.2
bicycle	69.9	64.1	75.6	77.3	77.8	77.4
bird	70.8	68.4	84.9	87.8	87.2	87.1
boat	41.3	35.8	59.7	70.3	62.2	67.1
bottle	51.0	44.1	48.2	45.8	43.7	47.0
bus	75.8	71.3	79.1	79.0	79.8	78.6
car	59.0	53.1	60.0	58.7	62.9	60.1
cat	92.4	90.0	93.5	92.4	94.1	94.8
chair	32.1	32.7	53.4	42.8	53.4	42.9
cow	64.6	60.7	67.1	66.3	60.1	72.6
diningtable	57.2	51.1	74.2	77.0	71.3	77.1
dog	85.3	83.3	93.6	92.3	92.5	92.0
horse	61.1	62.3	82.2	80.4	85.4	85.0
motorbike	62.0	65.3	86.7	81.0	85.6	85.0
person	70.7	68.7	72.6	49.5	74.1	53.3
pottedplant	29.0	25.4	53.1	49.2	48.8	51.8
sheep	62.2	62.1	71.2	71.4	74.7	69.1
sofa	59.9	51.9	79.2	82.0	86.4	80.1
train	73.3	73.2	75.4	77.2	79.6	81.6
tvmonitor	68.7	63.3	78.5	76.6	77.1	76.6
mAP	**62.6**	**59.2**	**73.8**	**72.1**	**73.3**	**73.25**

Table 4.2 Results of the Faster R-CNN (FR-CNN), Faster R-CNN with focal loss (FR-CNN FL), and Faster-LTN (F-LTN) on PASCAL VOC.

the training set was reduced in half. t-Distributed Stochastic Neighbor Embedding (t-SNE) visualizations of the extracted features, which are shown in Figure 4.2, provide additional evidence for the equivalent quality of the learned features.

Unlike the original work [42], Faster-LTN enables the sharing of knowledge between the logic module and the convolutional architecture. In previous work, fixed features that had been extracted from a convolutional model were used to train LTN. Although the classifier's performance improved, it was still constrained by the input features, which did not benefit from the logical constraints that LTNs brought about. By enabling end-to-end training in Faster-LTN, it is possible to increase the quality of the features obtained at the input of the logic module by overriding this restriction. The final aggregator chosen and the `isA` predicate's input represent an additional difference. Instead of the classification scores as in [42], the predicate in this setting receives the features extracted from the RPN as input. In this instance, the final aggregator is a log-product aggregator, while in the prior study, it was created from a mean aggregator.

The current architecture's complexity increases with the complexity of the underlying knowledge base. Future research may explore new grounding techniques to streamline this complexity with efficient representations and relations. Upcoming studies will use measured predicates to quantify the distance between features and prototypes, allowing insights into simplifying the architecture while enhancing functionality. These advancements will be detailed in subsequent sections of this dissertation, supported by recent research such as [82].

Dataset	Metric	FR-CNN	F-LTN $\mathcal{T}_{prior}$
PASCAL PART	mAP	35.1	41.2
PASCAL PART REDUCED	mAP	28.5	32.8

Table 4.3 Comparison of Faster R-CNN and Faster-LTN (including mereological constraints) on the PASCAL PART dataset.

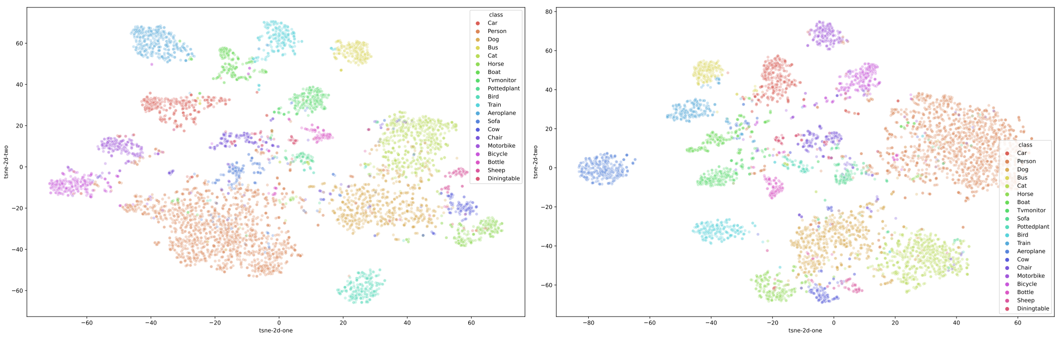


Fig. 4.2 Comparison of the t-SNE embeddings of the extracted features for the whole *testing set* object classes. Features taken from Faster-LTN with axiomatic restrictions (right) and conventional Faster R-CNN (left).

4.4 Conclusion

One of the main obstacles to deep learning applications is the lack of high-quality large-scale labeled datasets. Emerging NeSy approaches allow a deeper integration between perception and reasoning, complementing learning by example with the incorporation of axiomatic background information.

The Faster-LTN architecture, an end-to-end object detector comprising an RPN (based on the Faster R-CNN architecture) and an LTN module, was presented. This detector is trained in an end-to-end manner by optimizing the satisfiability of a grounded theory that combines sentences formed from labeled examples with axiomatic constraints.

The objective was to determine the efficacy of this strategy and the results indicate that Faster-LTN can perform at the same level or better than the Faster R-CNN baseline. More research is needed to evaluate whether this method can be scaled to larger training sets and other object detectors, such as single-stage detectors.

The goal is to provide a baseline architecture for further experiments and applications using the Faster-LTN model, which is accessible at https://gitlab.com/grains2/Faster-LTN.

Chapter 5

Proto-LTN a neuro symbolic architecture for zero-shot learning

Work described in this chapter was originally presented in [36].

This chapter delineates Proto-LTN, a framework that integrates a NeSy paradigm with Prototypical Networks (PNs) to enhance ZSL performance [36]. Proto-LTN employs a CNN to extract salient visual features and an LTN to optimize the satisfiability of a knowledge base expressed in first order logic statements. This integration facilitates the representation of class prototypes as abstract concepts, enabling the assignment of novel instances based on the Euclidean distance. Empirical evaluations on GZSL benchmarks demonstrate Proto-LTN's superior performance relative to conventional embedding-based methodologies. The details of the implementation are accessible at https://github.com/FrancescoManigrass/Proto-LTN.git. This chapter is organized as follows:Section 5.1 provides an overview of PNs in the context of FSL and GZSL, whileSection 5.2 describes the development of Proto-LTN for FSL and GZSL applications,Section 5.3 details the experimental setup, andSection 5.4 presents the results of the conducted experiments.

5.1 Prototypical networks

PNs offer a unified approach to handle scenarios where only a few examples are available for each class (FSL) or when recognizing unseen classes (ZSL and GZSL).

These networks create prototypes for each class based on available examples and use these prototypes to classify new instances.

Prototypical networks: the FSL setting

In an N-way-K-shot FSL scenario, a classifier must identify the correct class among N options, with K examples available per class for observation [83–85]. Labeled examples are called *support* examples, while unlabeled ones are called *query* examples. The fundamental assumption is that there exists an embedding space where elements of different classes are well-separated, which can be mathematically represented by an embedding function f_θ with parameter θ to be inferred, serving as a mapping:

$$f_\theta : \mathbb{R}^D \to \mathbb{R}^M. \tag{5.1}$$

In Equation 5.1, D represents the dimension of the input space, while M denotes the dimension of the embedding space. Therefore, for a given example x, $f_\theta(x)$ corresponds to its embedding. In FSL, a *prototype* for class n is obtained as the mean embedding of the K support examples of class n at train time:

$$p_n = \frac{1}{K} \sum_{(x,y)\in S} f_\theta(x) \tag{5.2}$$

Class prototypes must reside in the embedding space since they capture the average characteristics common to the elements of the class they symbolize. During training, θ is adjusted to minimize the distance between each prototype and its corresponding class elements while maximizing the distance between different prototypes. Ultimately, classification during testing involves assigning each query sample to the closest prototype.

During the testing phase, a support set containing N_S labeled examples $S = \{(x_1^S, y_1^S), ..., (x_{N_S}^S, y_{N_S}^S)\}$ is assumed to be available, where each $x_i^S \in \mathbb{R}^D$ represents the feature vector of an example, and $y_i^S \in C \subset \mathbb{N}$ is the corresponding label. In an N-way-K-shot scenario, exactly K support examples are provided for each of the N classes. In addition, a query set $Q = \{x_1^Q, ..., x_{N_Q}^Q\}$ consisting of N_Q unlabeled examples is given, and the goal is to accurately classify these examples into their respective classes. The elements of the query set Q originate from the same domain as those of the support set S.

During training, it might be impossible to predict which classes will appear in the testing phase. This means that a support set S cannot be predetermined. To address this, a training set $T = \{(x_1^T, y_1^T), ..., (x_{N_T}^T, y_{N_T}^T)\}$ is selected to best represent the prior knowledge about the testing scenario. The labels $y_i^T \in C_T \subset \mathbb{N}$ and $|C_T| = N_T$ classes may coincide with or exceed the number of testing classes ($N_T \geq N$). In other words, there may be overlap between C and C_T, but this cannot be known beforehand. Consequently, *artificial* support and query sets $S \subset T$ and $\tilde{Q} \subset T$ are generated to simulate the testing scenario and guide the model's learning process.

Prototypical networks: the GZSL setting

In ZSL, labeled examples for all classes are not available. Instead, it is assumed that N abstract vectors, represented as $\{a^{(1)}, a^{(2)}, ..., a^{(N)}\}$ with $a^{(n)} \in \mathbb{R}^A$, capture the characteristics of all N classes. Similarly to FSL, during training, a set $T = \{(x_1^T, y_1^T), ..., (x_{N_T}^T, y_{N_T}^T)\}$ of labeled examples from classes $y_i^T \in C_T \subset \mathbb{N}$ is utilized, where ideally $|C_T| = N_T \geq N = |C|$. The training process remains the same in the ZSL scenario, but class prototypes are defined differently:

- the embedding of a query instance x^Q is still derived as $f_\theta(x^Q)$, where $f_\theta : \mathbb{R}^D \to \mathbb{R}^M$;
- the prototype for class $n \in C$ is obtained as $p_n = g_\theta(a^{(n)})$ using a distinct embedding function $g_\theta : \mathbb{R}^A \to \mathbb{R}^M$, which translates the semantic attribute space into the shared embedding space.

5.2 Architecture

The complete architecture of Proto-LTN, adapted for the ZSL context, is depicted in Figure 5.1. The image embeddings are derived from a CNN, and the attribute vectors are projected into the embedding space using an embedding function. This section elaborates on the grounding definitions for constants, variables, functions, and predicates. Subsequently, the knowledge base $\mathcal{K}$, which encapsulates the learning problem, is defined.

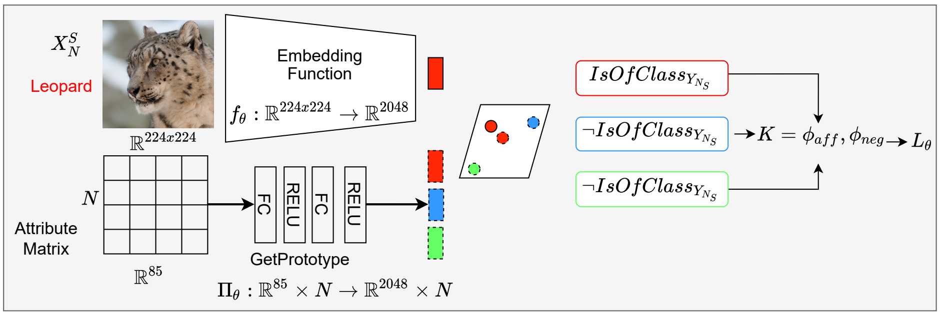


Fig. 5.1 Architecture of Proto-LTN for ZSL classification. An attribute encoder and a convolutional feature extractor constitute the architecture. Semantic and visual information is mapped in an embedding space shared by the two branches. Using affirmative and negative formulas encoded in the knowledge base $\mathcal{K}$, the `isOfClass` predicate seeks to minimise the distance between instances (solid line circles) and class prototypes (dashed line circles). The loss function maximises the satisfiability of all formulas in $\mathcal{K}$ (truth value) at train time.

5.2.1 Grounding terms

A batch of training samples in the form of query sets $\tilde{Q}$ and fake support $\tilde{S}$ is chosen within a single training episode. The definition of groundings for variables and their domain D (not learnable) is as follows:

$$
\begin{aligned}
\mathcal{G}(q) &= \langle x_1^{\tilde{Q}}, \ldots, x_{N_{\tilde{Q}}}^{\tilde{Q}} \rangle\,, && (5.3)\\
\mathcal{G}(q_l) &= \langle y_1^{\tilde{Q}}, \ldots, y_{N_{\tilde{Q}}}^{\tilde{Q}} \rangle\,, && (5.4)\\
\mathcal{G}(q_e) &= \mathcal{G}(\texttt{getEmbedding}(q)) && (5.5)\\
&= \langle f_\theta(x_1^{\tilde{Q}}), \ldots, f_\theta(x_{N_{\tilde{Q}}}^{\tilde{Q}}) \rangle\,, && (5.6)\\
\mathcal{G}(s) &= \langle x_1^{\tilde{S}}, \ldots, x_{N_S}^{\tilde{S}} \rangle\,, && (5.7)\\
\mathcal{G}(s_l) &= \langle y_1^{\tilde{S}}, \ldots, y_{N_S}^{\tilde{S}} \rangle\,, && (5.8)\\
\mathcal{G}(p), \mathcal{G}(p_l) &= \mathcal{G}(\texttt{getPrototypes}(s, s_l)) && (5.9)\\
&= \Pi_\theta(\mathcal{G}(s, s_l)) && (5.10)\\
&= \Pi_\theta(\langle (x_1^{\tilde{S}}, y_1^{\tilde{S}}), \ldots, (x_{N_S}^{\tilde{S}}, y_{N_S}^{\tilde{S}}) \rangle) && (5.11)
\end{aligned}
$$

where the labels are q_l ($D(q_l) =$ `labels`), the query examples are q ($D(q) =$ `features`), and the embeddings are q_e ($D(q_e) =$ `embeddings`). In contrast, the samples in the support set ($D(s) = \texttt{features}$) are denoted by s, and their labels are s_l. Finally, $D(p) = \texttt{embeddings}$ and $D(p_l) = \texttt{labels}$ represent the prototypes and their labels, respectively, p and p_l.

5.2.2 Grounding functions and predicates

The foundation of Proto-LTNs consists of the predicate `isOfClass` and two functions `embeddingFunction` $= f_\theta$ and `getPrototype` $= \Pi_\theta$, respectively.

The `getPrototypes` function accepts a support set of labeled examples as input and generates labeled prototypes. The input and output dimensions are specified as $D_{\text{in}}(\texttt{getPrototypes}) = \texttt{features} \times \texttt{labels}$ and $D_{\text{out}}(\texttt{getPrototypes}) = \texttt{embeddings} \times \texttt{labels}$, respectively. Equation 5.2 defines the support set of a class, which is what each prototype is dependent on. Thus, a new grounding for *generalized* LTN functions is suggested.

To get the reason behind the necessity for a generalized function, it is worth recalling that LTN variables are grounded onto the set of their instantiations. It can be assumed that s is a variable associated to support points, or:

$$\mathcal{G}(s) = \langle x_1^{\tilde{S}}, \ldots, x_{N_S}^{\tilde{S}} \rangle \,. \tag{5.12}$$

If h is an LTN function that is compatible with variable s, or $D_{\text{in}}(f) = D(s) = \mathbb{R}^D$, the grounding for $h(s)$ is

$$\mathcal{G}(h(s)) = \langle \mathcal{G}(h)(x_1^{\tilde{S}}), \ldots, \mathcal{G}(h)(x_{N_S}^{\tilde{S}}) \rangle \,. \tag{5.13}$$

This means that $\mathcal{G}(h)$ only takes as input a single element of $\mathbb{R}^D$. Unfortunately, a conventional LTN function such as h cannot help with prototypes, as their definition for a class $n \in \tilde{C}$, given in Equation 5.2, is:

$$p_n = \frac{1}{K} \sum_{\substack{(x^{\tilde{S}}, y^{\tilde{S}}) \in \tilde{S} \\ \text{s.t. } y^{\tilde{S}} = n}} f_\theta(x^{\tilde{S}}) = p_n(x_1^{\tilde{S}}, \ldots, x_{N_S}^{\tilde{S}}). \tag{5.14}$$

In real terms, each prototype depends on every support point within the same class. The embedding function $f_\theta : \mathbb{R}^D \to \mathbb{R}^M$ remains the same as in the FSL framework, whereas

$$\Pi_\theta : \bigcup_{l=1}^{\infty} l^l_{m=1} \mathbb{R}^D \times \mathbb{N} \to \bigcup_{l=1}^{\infty} {}^l_{m=1} \mathbb{R}^M \times \mathbb{N} \tag{5.15}$$

is designed for computational simplicity (e.g., implementable via a neural network) and is capable of generalizing to an N-way-K-shot scenario, even when N_S and $\tilde{N}$ are not fixed. Specifically, Π_θ operates as follows:

1. Take as input:
   (a) a support set $\tilde{S} = \{(x_1^{\tilde{S}}, y_1^{\tilde{S}}), ..., (x_{N_S}^{\tilde{S}}, y_{N_S}^{\tilde{S}})\} \in (\mathbb{R}^D \times \mathbb{N})^{N_S}$ of labelled examples, with $x_i^{\tilde{S}} \in \mathbb{R}^D$ and $y_i^{\tilde{S}} \in \mathbb{N}$;
   (b) the parameter θ or, for the sake of clarity, the embedding function $f_\theta : \mathbb{R}^D \to \mathbb{R}^M$.
2. Extract the classes contained in $\tilde{S}$ by applying:
   $$p^{(labels)} = \text{Unique}(y^{\tilde{S}}), \tag{5.16}$$
   where a vector's unique elements are retrieved via the "Unique" function. Since this variable will be linked to prototype labels named it $p^{(labels)}$. The number of items in $p^{(labels)}$ is defined as $\tilde{N}$.
3. Let $L \in \{0,1\}^{\tilde{N} \times N_S}$ be defined as a sparse "labels" matrix whose i, j-th entry is equal to 1 if support item i is of class $p_j^{(labels)}$, and 0 otherwise. Calculate the tensor $p \in \mathbb{R}^{\tilde{N} \times M}$ of the prototypes as
   $$p = \text{Diag}(L \mathbb{1}_{N_S})^{-1} L f_\theta(x^{\tilde{S}}), \tag{5.17}$$
   where
   $$\mathbb{1}_{N_S} = [1, 1, ..., 1]^T \in \mathbb{R}^{N_S} \tag{5.18}$$

is a vector of N_S ones, and

$$f_\theta(x^{\tilde{S}}) = [f_\theta(x_1^{\tilde{S}}), f_\theta(x_2^{\tilde{S}}), ..., f_\theta(x_{N_S}^{\tilde{S}})]^T \in \mathbb{R}^{N_S \times M} \tag{5.19}$$

is the way that f_θ is applied piece-by-piece to the elements in $x^{\tilde{S}}$, while "Diag" calculates the Diagonal matrix that is connected to a vector. Equation 5.2 accomplishes the same task as this expression, but is more general because it takes into account unbalanced support sets. For balanced support sets, where $\text{Diag}(L\mathbb{1}_{N_S})^{-1} = \frac{1}{K}I$, where I is the identity matrix and θ corresponds to the learnable parameters of the system. This is equivalent to a perfect N-way-K-shot scenario.

4. Return p and $p^{(labels)}$.

The `isOfClass` predicate for class $n \in C$ is grounded as:

$$\mathcal{G}(\texttt{isOfClass}) = e^{-\alpha d(\cdot,\cdot)^2}, \tag{5.20}$$

where α is a hyperparameter and d is a measure of distance. $\mathcal{G}(\texttt{isOfClass}) : \mathbb{R}^M \times \mathbb{R}^M \rightarrow [0,1]$; $\mathcal{G}(\texttt{isOfClass})$ takes the value of 1 when the distance from the class prototype $d(\cdot,\cdot)$ is 0. In the devised formulation the squared Euclidean distance is used, as in DEM [62].

5.2.3 Knowledge base

Every training episode updates $\mathcal{K}$, which is the formulation of the given learning task based on the current support set. Each query item in $\mathcal{K} = \{\phi_{\text{aff}}, \phi_{\text{neg}}\}$ is a positive example for its class and a negative example for all other classes, according to two aggregations of formulas:

$$\phi_{\text{aff}} = \forall\text{Diag}(q_e, q_l)(\forall\text{Diag}(p, p_l) : q_l = p_l(\texttt{isOfClass}(q_e, p))), \tag{5.21}$$

$$\phi_{\text{neg}} = \forall\text{Diag}(q_e, q_l)\,(\forall\text{Diag}(p, p_l) : q_l \neq p_l(\neg\texttt{isOfClass}(q_e, p))). \tag{5.22}$$

The formal definitions of both diagonal quantification and guarded quantifiers are described in Chapter 2.

Proto-LTN is trained by maximizing the satisfiability

$$\mathcal{L}^{\text{ep}} = 1 - \left(\bigwedge_{\phi \in \mathcal{K}} \phi \right) = -\mathcal{G}(\phi_{\text{aff}}) - w_{\text{n}}\, \mathcal{G}(\phi_{\text{neg}}), \tag{5.23}$$

with w_{n} representing the assumption that negations are less impactful than affirmations for classification. The value of w_{n} was set to 0, focusing solely on ϕ_{aff}, and defer examination of this hyper-parameter to subsequent research.

By introducing an aggregation function [40, 49], one obtains:

$$\mathcal{L}^{\text{ep}} = \left(-\log\left(\mathcal{G}(\phi_{\text{aff}})\right)^{\frac{1}{p_{\text{agg}}}} + w_{\text{n}}(1 - \mathcal{G}(\phi_{\text{n}}))^{\frac{1}{p_{\text{agg}}}} \right)^{p_{\text{agg}}} \tag{5.24}$$

where $\mathcal{G}(\phi_{\text{aff}})$ is implemented through the generalized product p-mean operator and $\mathcal{G}(\phi_{\text{neg}})$ with the generalized mean operator A_{pM}:

$$\begin{aligned} A_{pPR}(\tau_1, ..., \tau_n) &= \left(\prod_{i=1}^{n} \tau_i \right)^{\frac{1}{p_\forall}}, \\ A_{pM}(\tau_1, ..., \tau_n) &= \left(\frac{1}{n} \sum_{i=1}^{n} \tau_i^p \right)^{\frac{1}{p_\forall}}. \end{aligned} \tag{5.25}$$

It should be noted that the selection of p_{agg} does not have to match that of $p_\forall$ for quantification purposes, and both hyper-parameters require experimental tuning. When optimizing a positive metric, a common approach involves optimizing its logarithm: the product of similarities becomes more favourable when A_{pPR} is employed as the aggregation operator for $\forall$. Regrettably, this does not yield an equally attractive expression for ϕ_{neg}.

If a squared Euclidean distance is used as similarity measure and the negation weight w_{n} is set to 0, one obtains the same formulation of the loss function of DEM

[62], up to a scaling constant:

$$\mathcal{L}^{\text{ep}} = -\log\left(e^{-\frac{\alpha}{p_\forall}\left(\sum_{n\in\tilde{C}}\sum_{\substack{(x^{\tilde{Q}},y^{\tilde{Q}})\in\tilde{Q}\\ \text{s.t. } y^{\tilde{Q}}\neq n}} d(f_\theta(x^{\tilde{Q}}),p_n)^2\right)}\right) = \frac{\alpha}{p_\forall}\Big(\sum_{n\in\tilde{C}}\sum_{\substack{(x^{\tilde{Q}},y^{\tilde{Q}})\in\tilde{Q}\\ \text{s.t. } y^{\tilde{Q}}\neq n}} d(f_\theta(x^{\tilde{Q}}),p_n)^2\Big). \tag{5.26}$$

5.2.4 Proto-LTN: the GZSL scenario

The primary distinction between the GZSL and FSL settings is in the definition and computation of prototypes. For the GZSL example, no generalized LTN functions are required. In Algorithm 1, computations are provided for a training epoch.

The prototypes and elements of the support set correspond one to one because each class n has only one semantic vector $a^{(n)}$ provided. The semantic embedding function $g_\theta : \mathbb{R}^A \to \mathbb{R}^D$ represents the latter, yielding the common embedding space as the feature space. Simply put, `getPrototypes` reduces to a standard LTN function, with $\mathcal{G}(\texttt{getPrototypes}) = g_\theta$. On the other hand, given the query map, `getEmbedding` is unchanged.

5.3 Experimental setup

In the benchmarks Awa2 (Animals with Attributes) [2], CUB [86], aPY (Attribute Pascal and Yahoo) [87], and SUN (Scene Understanding) [88], experiments were carried out in both ZSL and GZSL settings. The image encodings, features and splits were obtained from the original reference [2] for all datasets.

The overall architecture consists of two separate components: the semantic encoder and the visual image encoder. The embedding function f_θ converts an image I into a vector $\mathbf{x} \in \mathbb{R}^M$, with $M = 2048$, by means of a ResNet101 [89] embedding model, which has been pretrained on ImageNet [90] and remains fixed. This setup is used in all datasets for each experiment. A function g_θ, which comprises two fully connected layers (FC) with ReLU activation function and is initialized by a truncated

Algorithm 1 Proto-LTN GZSL - Epoch

function TRAIN
  Input : q Training Images
  Input : q_l Training label
  Input : a Semantic attribute set
  Input : a_l Semantic attribute label
  for i *in* $N_{TrainingSteps}$ **do**
    q_{e_i} : `getEmbedding`(q)
    a_i, a_{l_i} : `getAttributes`(a)
    p_i, p_{l_i}: `getPrototypes`(a_i, a_{l_i})
    $\phi_{\text{aff}} = \forall \text{Diag}(q_{e_i}, q_{l_i}) \, (\forall \text{Diag}(p_i, p_{l_i}) : q_{l_i} = p_{l_i} \, (\texttt{isOfClass}(q_{e_i}, p_i)))$
    $\phi_{\text{n}} = \forall \text{Diag}(q_i, q_{l_i}) \, (\forall \text{Diag}(p_i, p_{l_i}) : q_{l_i} \neq p_{l_i} \, (\neg \texttt{isOfClass}(q_{e_i}, p_i)))$
    $\mathcal{L}^{\text{ep}} = \left(-\log \left((\mathcal{G}(\phi_{\text{aff}}))^{\frac{1}{p_{\text{agg}}}} \right) + w_{\text{n}} \, (1 - \mathcal{G}(\phi_{\text{n}}))^{\frac{1}{p_{\text{agg}}}} \right)^{p_{\text{agg}}}$
    $computeGradient(\mathcal{L}^{\text{ep}})$
    $updateGradient$
  end for
end function

function TEST
  Input : q Test Images
  Input : a Semantic attribute set
  q_e : `getEmbedding`(q)
  a,a_l : `getAttributes`(a)
  p, p_l: `getPrototypes`(a,a_l)
  for i *in* len(q_e) **do**
    for j *in* len(p) **do**
      $prediction_i$: `isOfClass`(q_{e_i}, p_j)
    end for
  end for
end function

Method	Awa2				CUB				APY				SUN			
	T1	U	S	H	T1	U	S	H	T1	U	S	H	T1	U	S	H
SYNC (2016) [92]	46.6	10.0	90.5	18.0	55.6	11.5	**70.9**	19.8	-	-	-	-	56.3	7.9	43.3	13.4
Relation Net (2017) [93]	64.2	30.0	93.4	45.3	55.6	38.1	61.1	47	-	-	-	-	-	-	-	-
PrEN† (2019) [94]	74.1	32.4	88.6	47.4	66.4	35.2	55.8	43.1	-	-	-	-	**62.9**	**35.4**	27.2	**30.8**
VSE (2019) [63]	**84.4**	**45.6**	**88.7**	**60.2**	**71.9**	**39.5**	68.9	**50.2**	**65.4**	**43.6**	**78.7**	**56.2**	-	-	-	-
DEM (2017) [62]	67.1	30.5	86.4	45.1	51.7	19.6	57.9	29.2	35.0	11.1	75.1	19.4	61.9	20.5	34.3	25.6
Proto-LTN	67.6	32.0	83.7	46.2	48.8	20.8	54.3	30.0	35.0	17.1	66.2	27.21	60.4	20.4	**36.8**	26.2
	±1.1	±1.3	±0.3	±1.3	±1.2	±2.6	±1.1	±3.0	±3.1	±2.0	±5.1	±2.9	±2.5	±1.0	±4.4	±1.9
	(70.8)	(34.8)	(84.3)	(49.1)	(50.3)	(23.4)	(55.7)	(33.0)	(38.6)	(19.4)	(70.7)	(30.0)	(62.1)	(22.15)	(39.9)	(28.0)

Table 5.1 For Proto-LTNm mean $\pm$ standard deviation and maximum (in parentheses) performance values are shown. $\mathrm{TOP1}^{\mathrm{ZSL}}$ (T1), $\mathrm{TOP1}^{\mathrm{GZSL\_UNSEEN}}$ (U), $\mathrm{TOP1}^{\mathrm{GZSL\_SEEN}}$ (S), and $\mathrm{H}^{\mathrm{GZSL}}$ (H) are always obtained on the proposed split (PS) of Awa2, CUB, aPY, and SUN classes, as described in [2]. † assumes a transductive ZSL setting. Best performance values are reported in bold.

normal distribution function, encodes semantic vectors in the embedding space. The hyper-parameters $p_{agg} = 1$ and $p_{\forall} = 2$ are based on preliminary tests on Awa2 [2].

Based on the LTN package, the framework was implemented in Tensorflow [40, 91]. The workstation used for the experiments had an RTX2080 TI GPU and an Intel® CoreTM i7-10700K CPU. Using a batch size of 64 and the Adam optimizer, all networks were trained for 30 epochs. For every dataset, the hyper-parameters (learning rate, α, and regularisation term, λ) underwent separate optimisation. Performance metrics for GZSL were the standard ones, as specified in [2]. Each experiment was run three times to calculate the mean and standard deviation.

5.4 Results

Table 5.1 presents the Proto-LTN results as well as those for similar embedding-based techniques. Class prototypes are marked to show the embedding space in Figure 5.2.

Because most embedding-based techniques, including DEM [62] and Relation Net [93], rely on identical assumptions and use the same input as the present implementation of Proto-LTN, the experimental performance is competitive, as expected on the basis of this analytical study. As demonstrated insubsection 5.2.3, the Proto-LTN loss is comparable to the DEM loss under some circumstances, up to a scaling constant, but with distinct regularisation terms. For all experimental benchmarks, DEM is outperformed on unseen classes, suggesting that the suggested formulation is a solid foundation for a novel, NeSy solution to the GZSL challenge.

VSE performs better than Proto-LTN because it uses a different approach to generate visual feature embeddings, aligning the embedding space with part-feature concepts given by a semantic oracle. The latter comprises concepts outside the given semantic vector $\{a^{(1)}, a^{(2)}, \ldots, a^{(N)}\}$ because it depends on an external knowledge base. This is particularly useful for noisy and visually non-informative benchmarks such as aPY [63]. This is not a feature of Proto-LTNs; rather, it is a limitation of this specific formulation that will be addressed in the following chapters.

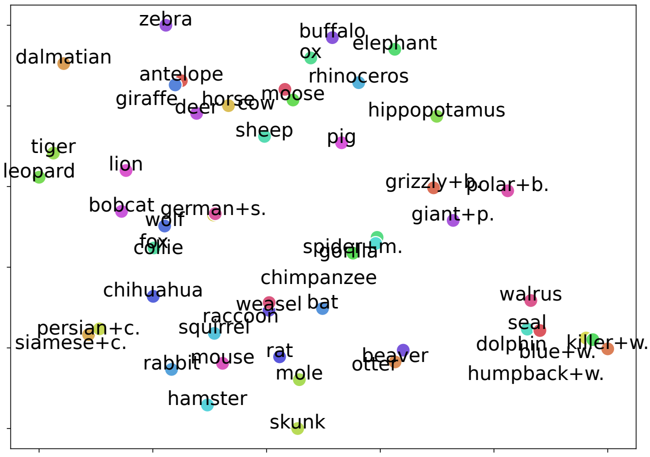


Fig. 5.2 Class prototypes visualised using t-SNE for the Awa2 dataset

5.5 Conclusion

A novel NeSy architecture, Proto-LTN, was introduced, which extends the classical formulation of LTNs by incorporating embedding-based techniques. Following the strategy of PNs, the focus is exclusively on learning embedding functions (such as f_θ and g_θ), where class prototypes are derived ex-post based on a support set. These methods have demonstrated robustness to noise, which is essential in few-shot learning (FSL), and offer a framework for embedding both examples (images) and class prototypes within the same metric space. This characteristic in the LTN context enables different levels of abstraction, allowing for statements about individual examples or entire classes, as prototypes can serve as parameterized class labels. The feasibility of this approach has been validated in generalized zero-shot learning (GZSL), with plans to extend it by introducing the concept of extracted attributes and macroclasses.

Although the experimental results are promising, the strength of this formulation lies in its generality, with the full potential of Proto-LTN yet to be realized. Future research can pursue two complementary directions. First, alternative formulations of the `isOfClass` relationship could be explored by varying the distance metric and/or the prototype encoding. Mapping class prototypes back to the input space, as suggested in [95], could enhance explainability.

Second, as done in Chapter 6, knowledge $\mathcal{K}$ can be extended to take advantage of prior information from external knowledge bases, to improve generalization to unseen classes. Experiments should include both inductive and transductive settings. The assumption that information about attributes and relationships of unseen classes is available at training or test time (e.g., from WordNet) is less restrictive than assuming that actual examples, albeit unlabeled, are available.

Chapter 6

Fuzzy logic visual network (FLVN)

Work described in this chapter was originally presented in [37].

This chapter introduces FLVN, a more advanced framework that further integrates the LTN paradigm with CNNs to improve ZSL [37]. Specifically, FLVN uses a CNN as a feature extractor to convert images into the attribute space, and the LTN knowledge base is extended to introduce the concept of macroclasses. Additionally, for each class, the model considers cases where an image may not contain all visible attributes. The organization of this chapter is as follows. InSection 6.1, the development of FLVN is thoroughly examined.Section 6.2 outlines the experimental setup. The results obtained from these experiments are discussed inSection 6.3. Finally,Section 6.4 concludes the chapter with a summary of the findings.

6.1 Architecture

The FLVN architecture is built on two fundamental components: a convolutional feature extractor and an LTN. The convolutional feature extractor is responsible for processing the raw input data, converting it into a set of meaningful features, while the LTN leverages these features to reason over structured knowledge using formal logic. These two components work together to facilitate alignment between visual data and semantic attributes, enabling the model to perform tasks such as ZSL. The overall architecture is depicted inFigure 6.1, which illustrates how these components interact to transform visual information into a space where logical reasoning can be applied.

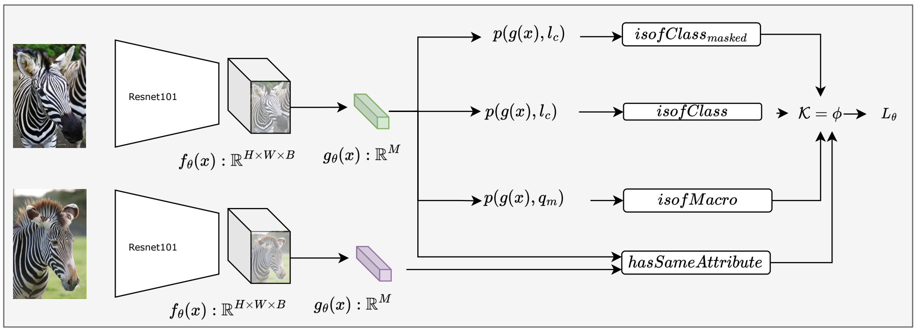


Fig. 6.1 A convolutional feature extractor and an attribute encoder are combined in the FLVN architecture, which is intended for ZSL classification, to efficiently transfer visual information to the attribute space. Image features and class attributes are aligned by utilising predicates like `isOfClass`, $\texttt{isOfClass}_{\texttt{masked}}$, and `isOfMacro`, while the `hasSameAttribute` predicate measures the similarity between two images. Since they are all included in the formulae of a knowledge base called $\mathcal{K}$, these predicates are all essential to the construction of the architecture. The goal of the design of the loss function during training is to maximize the satisfiability of each formula, or the truth value, within $\mathcal{K}$.

6.1.1 Feature extraction

The CNN that performs feature extraction (embedding) translates the input x into a feature space $f_\theta(x) \in \mathbb{R}^{H \times W \times B}$, where H, W, and B represent the height, width and number of channels of the feature map, respectively. The global discriminative characteristics $g_\theta(x) \in \mathbb{R}^{B \times 1}$ are derived by mean pooling over H and W. A linear projection is used to translate these global characteristics into a semantic space denoted by $V \in \mathbb{R}^{B \times M}$, where B is the dimension of the vector space of features and M is the length of the attribute vector [60].

6.1.2 Logic tensor network

The learning objective in the LTN framework is to maximize the satisfiability of a knowledge base $\mathcal{K}$. For each training batch, the knowledge base is updated by incorporating axioms that represent labeled examples (ϕ_{class}) as well as prior knowledge ($\phi_{macro}, \phi_{s-attr}, \phi_{d-attr}, \phi_{e-attr}, \phi_{masked}$). Subsequently, the overall satisfiability is evaluated by aggregating these axioms, enabling the model to effectively align its predictions with the available information.

This section first defines the variables, predicates, and domain that form the FOL language, followed by the definition of the knowledge base $\mathcal{K}$.

Groundings

The variables and their domains are grounded as follows:

$$\mathcal{G}(l) \in \mathbb{N}^C, \mathcal{G}(q) \in \mathbb{N}^Q \tag{6.1}$$

$$\mathcal{G}(a) = \mathcal{G}(a^{\text{mask}}) \in \mathbb{R}^{M\times C} \tag{6.2}$$

$$\mathcal{G}(a^{\text{macro}}) \in \mathbb{R}^{M\times Q} \tag{6.3}$$

$$\mathcal{G}(x) = g_\theta(f_\theta(\mathcal{G}(\texttt{images}))) \in \mathbb{R}^M \tag{6.4}$$

Each class and macroclass are defined by a set of non-binary semantic attributes, denoted as a and a^{macro}, respectively. Labels are organized hierarchically into classes and macroclasses, with macroclasses serving as high-level categories that group multiple classes based on shared visual properties. To efficiently manage this variability and address cases where images of a specific class may be missing certain attributes, the attribute mask $a_{mask} \in \mathbb{M}^{C\times A}$ is introduced. This mask mirrors a but with k elements randomly set to 0. The variables l and q denote the class labels for the sets of classes C and Q, respectively, as well as the macroclass labels. The final representation $\mathcal{G}(x)$ is achieved by embedding images into the attribute space using the functions f_θ and g_θ.

In the FOL language, four primary predicates are defined: a picture x is classified into class l using $\texttt{isOfClass}(x,l)$ and $\texttt{isOfClass}_{\texttt{masked}}(x,l)$. Likewise, $\texttt{isOfMacro}(x,q)$ signifies that an image x is a member of macroclass q, and $\texttt{hasSameAttribute}(x_1,x_2)$ denotes that two images share identical attributes.

The similarity between the respective class attribute vectors and the input image serves as the foundation for the $\mathcal{G}(\texttt{isOfClass})$ predicate. Firstly, the scaled product of the global features mapped in the attribute space with the semantic vectors to compute the similarity between the picture x and a class l_c is used:

$$p(x,l) = \frac{\exp\left(x^T V a_l\right)}{\sum_{s=1}^{S} \exp\left(x^T V a_s\right)} \tag{6.5}$$

the semantic attribute vector connected to class l is represented by a_y. In order to derive a prediction score for an example x, the dot product of the one-hot encoding l_c^T for class $c \in C$ and the output of p (Equation 7.10) are computed as follows:

$$\mathcal{G}(\texttt{isOfClass}) : x, l_c \to {l_c}^T p(\mathcal{G}(x), l_c) \tag{6.6}$$

$\mathcal{G}(\texttt{isOfClass}_{\texttt{masked}})$ and $\mathcal{G}(\texttt{isOfMacro})$ are defined as follows. A trainable attribute vector a_m^{macro} for macro-class q_m is built to compute $\mathcal{G}(\texttt{isOfMacro})$, since the attributes for macro-classes are unknown. For class l_c, $\mathcal{G}(\texttt{isOfClass}_{\texttt{masked}})$ employs the masked attribute vector a_c^{masked}, in which the remaining attributes of a_c are preserved. Lastly, grounding of `hasSameAttribute` is as follows:

$$\mathcal{G}(\texttt{hasSameAttribute}) : x_1, x_2 \to \texttt{sigmoid}(\alpha \texttt{d}(\mathcal{G}(x_1), \mathcal{G}(x_2))) \tag{6.7}$$

where $\texttt{d}$ is the cosine similarity, α a scale factor, and $\mathcal{G}(x_1)$, $\mathcal{G}(x_2)$ correspond to the embeddings of the two images.

Learning from labeled examples

Incorporating labelled examples is achieved by introducing an axiom ϕ_{class}, which stipulates that all facts concerning labelled instances must be true, meaning that all labelled samples must be correctly classified:

$$\phi_{\text{class}} = \forall \texttt{Diag}(x, l_c)(\texttt{isOfClass}(x, l_c)) \tag{6.8}$$

To account for the class hierarchy, an axiomatic statement ϕ_{macro} is introduced to indicate that "if an image contains a zebra", then "the image belongs to the family of ungulates":

$$\phi_{\text{macro}} = \forall \texttt{Diag}(x, l_c, q_m)(\texttt{isOfClass}(x, l_c) \implies \texttt{isOfMacro}(x, q_m)) \tag{6.9}$$

Acquiring stronger feature representations

The following axiom encodes the assumption that features extracted from two images of the same class should possess the same attributes:

$$\phi_{\text{s-attr}} = \forall \texttt{Diag}(x_1, l_{c_1}) \left(\forall \texttt{Diag}(x_2, l_{c_2}) : c_1 = c_2 \ \ \texttt{hasSameAttribute}(x_1, x_2) \right) \tag{6.10}$$

Likewise, images from different classes should possess different attributes:

$$\phi_{\text{d-attr}} = \forall \texttt{Diag}(x_1, l_{c_1}) \left(\forall \texttt{Diag}(x_2, l_{c_2}) : c_1 != c_2 \ \ \neg\texttt{hasSameAttribute}(x_1, x_2) \right) \tag{6.11}$$

To further emphasize the similarity between visual attributes and semantic vectors, the following axiom enforces the similarity between image embeddings and attribute vectors of the same class:

$$\phi_{\text{e-attr}} = \forall \texttt{Diag}(x, l_c) \left(\forall \texttt{Diag}(a, l_a) : c == a \ \ \texttt{hasSameAttribute}(x, a) \right) \tag{6.12}$$

Acquiring knowledge through disagreement

In traditional ZSL benchmarks like Awa2, a crisp or fuzzy matrix is used to correlate properties with class labels. However, this association does not guarantee that all instances of a class have exactly the same features since some of them may be obscured or expressed by a subset of different training samples. The existential assertion ϕ_{masked} denotes the possibility that some class attributes might not be present for all samples. The $\texttt{isOfClass}_{\texttt{masked}}$ predicate that eliminates randomly chosen attributes to simulate image-level attributes that are not available is defined as:

$$\phi_{\text{masked}} = \forall l^{\text{seen}} (\exists x, \texttt{isOfClass}_{\texttt{masked}}(x, l^{\text{seen}})) \tag{6.13}$$

where l^{seen} denotes the list of seen classes.

Logic connectives and aggregators grounded

Each training phase updates the collection of formulas that makes up the knowledge base $\mathcal{K}$. The Reichenbach implication $\rightarrow$: $I_R(a,b) = 1 - a + ab$ and the standard negation $\neg : N_S(a) = 1 - a$ are defined to adopt the symmetric configuration from [40] given two truth values a and b in $[0,1]$.

The generalized mean A_{pM}, discussed in Equation 6.14, was used to approximate the existential quantifier $\exists$, whereas the generalized mean w.r.t. the error A_{pME} was used to approximate the universal quantifier $\forall$ [40, 49].

$$\forall : A_{pME}\left(a_1, \ldots, a_n\right) = 1 - \left(\frac{1}{n}\sum_{i=1}^{n}\left(1 - a_i\right)^{p_\forall}\right)^{\frac{1}{p_\forall}} \qquad p_\forall \geqslant 1 \tag{6.14}$$

The measure A_{pME} indicates the average deviation of truth values a_i from the true value of 1.

Querying the knowledge base

At inference time, the class with the highest score is selected as the predicted class:

$$\hat{y} = \underset{\tilde{y} \in Y^U}{argmax}(g(x)^{\mathrm{T}} V a_{\tilde{y}}) \tag{6.15}$$

FVLN was assessed in GZSL and ZSL configurations. While the model is evaluated on both seen and unseen classes in the GZSL setting, only unseen photos are assumed to be present at test time in the ZSL setting. This setting creates a bias in favour of seen classes. In order to address this, the Calibrated Stacking technique is used, which was introduced in [96, 97], to reduce the classification score of the observed classes. Thus, $\hat{y}$ is used to calculate the class score:

$$\hat{y} = \underset{\tilde{y} \in Y^U \cup Y^S}{arg\,max}\left(g(x)^{\mathrm{T}} V a_{\tilde{y}} - \gamma \mathbb{I}\left[\tilde{y} \in \mathcal{Y}^S\right]\right) \tag{6.16}$$

where γ is a calibration coefficient tuned on a validation set, $\mathcal{Y}^S$ are the labels of seen classes, and $\mathbb{I} = 1$ if $\tilde{y}$ is from a seen class and zero otherwise.

Building the training batch

For a positive input image x_i, a collection of positive examples x^+ and K negative examples $x_1^-,...,x_K^-$ is chosen, in accordance with the methodology in [61]. Negative examples are chosen at random from the remaining classes, and positive examples are chosen from the same category as x_i.

6.2 Experimental setup

This section focuses on the datasets used, the list of hyperparameters selected, and the knowledge base used for each dataset.

Dataset. The Awa2 [2], CUB [86], and SUN [88] benchmarks were used for the experiments. The standards outlined in earlier research [2] served as the basis for the GZSL evaluation metrics. Constructing a semantic hierarchy, the classes from the Awa2 and CUB datasets are organized into a total of 9 and 49 macroclasses, respectively, according to a similar methodology described in [98].

Knowledge base. Classes from the Awa2 and CUB datasets are mapped to match synsets in WordNet, using the methodology of Sikka et al. to construct a class hierarchy [98]. Each dataset's macroclasses were defined by first identifying the synset root whose subtree included classes from the chosen dataset, and then classifying its immediate offspring as classes using WordNet. As classes in the SUN dataset [88] did not have a semantic structure, the knowledge base did not have axioms pertaining to macroclasses.

In the predicate $\texttt{isOfClass}_{\texttt{masked}}$, $k = 15$ characteristics were omitted at random for each trial. For the Awa2 dataset, the α parameter in Equation 6.7 was set to 0.01, and for the CUB and SUN, to 1. The parameters of the aggregation function (specified in Equation 2.21 and Equation 6.14) are initially set to $p_\exists = 2$ and $p_\forall = 2$ in order to account for the presence of outliers in the knowledge base. For the Awa2 and CUB datasets, both parameters were increased by 2 every 4 epochs, where for SUN, they were increased at specified epochs (2, 4, 24, and 32), until $p_\exists = 6$ and $p_\forall = 6$, in accordance with the schedule recommended in [40].

Hyperparameter selection. Based on an ImageNet pre-trained ResNet101 model, the embedding function f_θ transforms the 224×224 image into a vector

$\mathbf{x} \in \mathbb{R}^{H \times W \times B}$, where $B = 2048$ and H and W represent the height and width of the extracted features, respectively. These features are then transformed into the attribute space using the function g_θ, which translates them into a dimensionally consistent with the dataset.

First, the head is trained while the backbone is kept frozen, and then the entire network is fine-tuned, to reduce overfitting. For the pre-training stage of Awa2, CUB, and SUN, the Adam optimizer is employed with a learning rate of $1e-4$ and 5e-4, respectively. Subsequently, in the fine-tuning phase, the learning rate was reduced to $1e-6\alpha$ for CUB and SUN and $1e-7\alpha$ for Awa2, where $\alpha = 0.8^{epoch//10}$ and *epoch* is the current epoch out of a total of 300 epochs. L2 norm regularization coefficient is set to 5e-4 for Awa2, 5e-6 for CUB, and 1e-3 for SUN.

For the Awa2, CUB and SUN data sets, the ratio of positive to negative samples in the training batches was 12 to 12. The scaling factor γ is set to 0.4 for SUN and 0.7 for Awa2 and CUB at inference time to modify the scores obtained for the viewed classes. For data augmentation, random flip with 0.5 chance and random crop are used in each experiment. Using the LTNtorch library [99], the architecture is developed in PyTorch and trained on a single GPU, a Nvidia 2080 Ti. For the purpose of determining the mean and standard deviation, each experiment was conducted three times.

6.3 Results

The experimental results, which can be seen in Table 6.1, demonstrate how the suggested FLVN architecture outperforms alternative embedding-based techniques, specifically CC-ZSL [61] and APN [60]. FVLN achieves strong performance on two out of the three benchmark datasets, generally outperforming other models in terms of harmonic mean (H) and demonstrating a stronger capacity to recognise both seen and unseen classes. Specifically, FLVN increases H and the accuracy of the unseen class by 1. 3% and 0. 89%, respectively, in Awa2; moreover, it increases H and the accuracy of the seen class by 12% and 3%, respectively, in CUB.

The results presented here imply that the suggested design can distinguish between seen and unseen data more consistently than Proto-LTN, resulting in fewer classification errors. On SUN, FLVN achieves performance close to the state of

	Awa2				CUB				SUN			
Model	T1	U	S	H	T1	U	S	H	T1	U	S	H
Proto-LTN [36]	67.6	32.0	83.7	46.2	48.8	20.8	54.3	30.0	60.4	20.4	**36.8**	26.2
DEM [62]	67.1	30.5	**86.4**	45.1	51.7	19.6	57.9	29.2	61.9	20.5	34.3	25.6
VSE [63]	84.4	45.6	88.7	60.2	71.9	39.5	68.9	50.2	-	-	-	-
TCN [66]	71.2	61.2	65.8	63.4	59.5	52.6	52.0	52.3	61.5	31.2	37.3	34.0
CSNL [98] †	61.0	-	-	0.0	32.5	-	-	0.7	-	-	-	-
AREN [68]	67.9	54.7	79.1	64.7	71.8	63.2	69.0	66.0	60.6	40.3	32.3	35.9
APN [60]	68.4	56.5	78.0	65.5	72.0	65.3	69.3	67.2	61.6	41.9	34.0	37.6
AMGML [67]	71.7	56.0	74.6	64.0	70.0	58.2	55.7	56.9	59.7	42.0	35.1	38.3
CC-ZSL [61]	68.8	62.2	83.1	71.1	74.3	66.1	73.2	69.5	62.4	44.4	36.9	40.3
Cycle-CLSWGAN [74]	-	-	-	-	58.4	45.7	61.0	52.3	60.0	**49.4**	33.6	40.0
LisGan [73]	-	-	-	-	58.8	46.5	57.9	51.6	60.0	42.9	37.8	40.2
E-PGN [71]	73.4	52.6	83.5	64.6	72.4	52.0	61.1	56.2	-	-	-	-
TGMZ [72]	**78.4**	64.1	77.3	70.1	66.1	60.3	56.8	58.5	-	-	-	-
CEGZSL [75]	70.4	63.1	78.6	70.0	77.5	63.9	66.8	65.3	63.3	48.8	38.6	43.1
DFCA-GZSL [100]	74.7	66.5	81.5	73.3	**80.0**	**70.9**	63.1	66.8	62.6	48.9	**38.8**	**43.3**
FLVN †	69.8	65.8	82.3	73.1	71.2	62.6	83.1	71.5	61.7	48.4	32.7	39.0
	±0.8	±0.9	±0.3	±0.6	±0.2	±0.5	±0.3	±0.2	±0.18	±0.57	±0.2	±0.1
	(71.0)	(67.1)	(82.8)	(**74.1**)	(71.4)	(63.2)	(83.4)	(**71.7**)	(61.9)	(48.9)	(32.9)	(39.1)

Table 6.1 Performance on the test sets for Awa2, CUB, and SUN. The mean ± standard deviation and maximum (in parenthesis) values for TOP1$_{zsl}$ (T1), TOP1$_{gzsl}$ unseen (U), TOP1$_{gzsl}$seen (S), and H$_{gzsl}$(H) over three runs are displayed for FLVN. A description of the metrics can be found in [2]. The models in the table are divided into three sections: generative models, attention-based models, and embedding-based models. Bold face indicates the highest performing values. † indicates techniques that make use of external information.

the art; nevertheless, it should be noted that since axioms on macro classes are not presented in this scenario, FLVN is unable to take advantage of outside semantic information.

In terms of performance, the suggested approach has a reasonably simple design compared to the closest methods. In contrast to APN, FVLN does not need extra regularization terms or weights for each attribute to build the loss function. Compared to the teacher-student framework [61], which achieves closer performance, FLVN only needs one backbone replica as opposed to two in CC-ZSL. Compared to generative techniques, FLVN does not need to create more training samples or make predictions about classes that have not been observed yet during training.

Qualitative observations from these experiments suggest that the axioms incorporated into the knowledge base can have a significantly varied impact on overall performance. Chapter 7 provides further experiments to evaluate this phenomenon in detail.

As can be seen from the distribution of the feature space (Figure 5.2) previously published in earlier work (Proto-LTN [36]), similar classes seem to naturally cluster in feature space during training. However, the `isOfClass`$_{\texttt{masked}}$ predicate provided significant advantages, accounting for errors in the semantic annotation of classes or qualities (e.g., “All zebras are agile”) that exist at the class level but are not readily apparent or deducible from a single image. Lastly, axioms based on the `hasSameAttribute` predicate increased performance, especially on datasets that require more effort to extract fine-grained image-level characteristics, such as CUB (fine-grained bird recognition) and SUN (scene recognition).

6.4 Conclusion

Building on principles from the recent ZSL literature [60] and incorporating them within a NeSy framework [36], a novel NeSy architecture named FLVN is introduced for ZSL and GZSL tasks. FLVN incorporates axioms that combine prior class-specific knowledge (e.g., class hierarchies) with high-level inductive biases to handle exceptions within the dataset (e.g., “there exists a zebra that is not agile”) and establish relationships between images (e.g., “if two images belong to the same class, they must be similar”). These axioms act as semantic priors and compensate for the lack of annotations, thereby providing a solid NeSy foundation for GZSL tasks. FLVN does not require multiple backbones and maintains roughly the same parameter count as standard embedding-based methods. The proposed approach can also be incorporated into other architectures, for example, by changing the grounding of the predicates. Experimental results demonstrate that FLVN achieves performance comparable to or exceeding that of the current literature on common GZSL benchmarks. Different formulations of the `isOfClass` predicate or the introduction of a `hasSameAttribute` predicate to predict image-level attributes could incorporate attention to discriminative regions within the image: these aspects are expanded upon inChapter 7.

Chapter 7

Fuzzy logic prototypical network (FLPN)

In this chapter, FLPN is introduced. As in previous architectures presented in Chapter 5 and Chapter 6, it is designed to integrate a series of class prototypes in the image space and trained them via the LTN framework. The learning process occurs within the embedding space, where class prototypes are derived from a set of attributes to develop abstractions for classes that are unseen during training. Although the problem formulation resembles that of previous approaches such as Proto-LTN, this model incorporates a more sophisticated knowledge base, building on FLVN [37], within a single system trained in an end-to-end manner.

FLVN also enables learning not only class prototypes but also prototypical representations of attributes within a unified space. A novel grounding for representing concepts at both the class and macroclass levels is demonstrated, allowing for the representation of semantically related concepts within a single space. Furthermore, by introducing attribute prototypes, the research reports an attention mechanism capable of identifying which attributes best characterize a given sample at the image level, thereby addressing the issue of insufficient attribute-level annotations for each sample.

Finally, the application of two distinct backbones to tackle the designated task is examined, highlighting the advantages and disadvantages by comparing the developed architecture with other state-of-the-art methods.

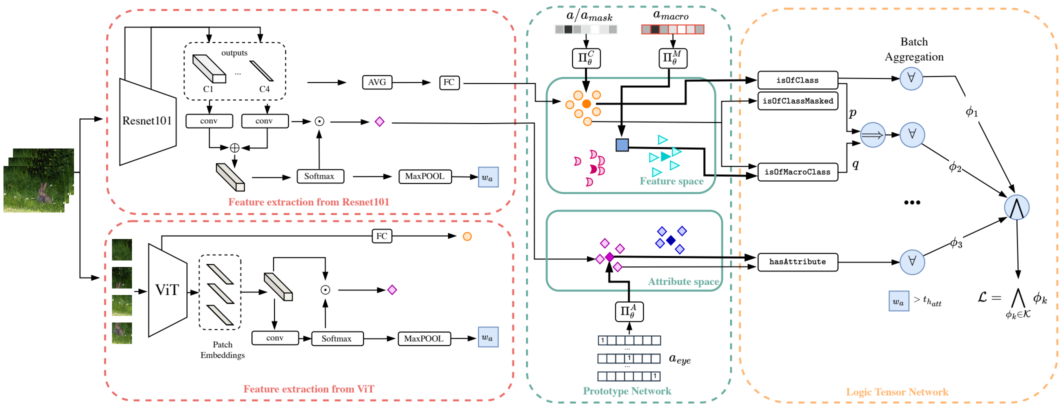


Fig. 7.1 The design of FLPN consists of two main components: the feature extractor and the prototype network. The feature extractor can be either a CNN, such as ResNet-101 utilized in the trials, or a visual transformer. The prototype network (Π^C_θ, Π^M_θ, Π^A_θ) integrates input images with class (a), macroclass (a_{macro}), and attribute (a_{eye}) labels into a unified embedding space, thereby grounding the symbols within this space according to the LTN terminology. Various predicates, such as `isOfClass`, $\texttt{isOfClass}_{\text{masked}}$, and `isOfMacro`, are established as class membership functions based on this embedding space. Moreover, the `hasAttribute` predicate identifies specific attributes within images. These predicates form the foundation of the knowledge base, denoted as $\mathcal{K}$, of the LTN module. The training objective (loss function) is designed to improve the satisfiability or truth value of this $\mathcal{K}$. The symbol $\oplus$ signifies element addition, whereas $\odot$ denotes element multiplication.

The structure of this chapter is detailed as follows: Section 7.1 provides an in-depth analysis of FLVN development. Section 7.2 describes the experimental setup used in the study. The results of these experiments are reviewed in Section 7.3. Lastly, Section 7.4 wraps up the chapter with a summary of the key findings.

7.1 Architecture

The structure of the FLPN architecture is illustrated in Figure 7.1, highlighting its main components. The backbone, detailed in subsection 7.1.2, is responsible for the extraction of image features. The prototype network, described in subsection 7.1.3, generates prototypes for various classes, macroclasses, and attributes. The architecture was assessed using two distinct backbones: a CNN and a transformer.

Furthermore, subsection 7.1.4 outlines the integration of prototypes and images into a unified embedding space, which constitutes the core of the knowledge base $\mathcal{K}$, continuously refined during the training phase [101].

7.1.1 The ZSL and GZSL settings

In alignment with previous approaches [36, 62, 69], an embedding space is defined to map the input images and the attribute matrix into a shared space:

- a transformation function, as described in subsection 7.1.2, translates images from their initial domain into a common *embedding space* $\mathbb{R}^{A\times M}$.
- a collection of functions, referenced in subsection 7.1.3, projects attribute vectors into *prototypes* within the shared embedding space $\mathbb{R}^{A\times M}$.

7.1.2 Image feature extraction

As mentioned, two distinct techniques were developed for feature extraction, employing a CNN (ResNet-101) or transformer (ViT). These architectures extract detailed image information, producing feature embeddings at the class, macroclass, and attribute levels to minimize the distances to the representative prototypes.

Global features representation

Following the methodology outlined by [36], a specialized function has been devised to map images to their corresponding class embedding spaces, including macroclasses that share the same embedding space:

$$f_\theta : \mathbb{R}^{Ch\times W\times H} \rightarrow \mathbb{R}^{M} \tag{7.1}$$

Here, $D = Ch \times W \times H$ represents the input image domain, with Ch referring to the number of channels, W indicating the width, and H denoting the height. The feature space has a dimensionality of M, and θ denotes the model trainable parameters.

CNN-based. The final layer of the ResNet-101 backbone is utilized to extract global features directly.

ViT-based. The images are divided into square patches that are then processed by the transformer encoder to derive feature tensors. In contrast to CNNs, the class-level feature is encapsulated by integrating a learnable classification token [CLS], as described in [69].

Attribute features representation

An additional embedding function is utilized to convert each image into appropriate attribute embeddings through an attention mechanism:

$$g_\theta : \mathbb{R}^{Ch \times W \times H} \to \mathbb{R}^{A \times M} \tag{7.2}$$

where $D = Ch \times W \times H$ signifies the input image dimensions, M represents the dimensionality of the shared feature space, A indicates the number of attributes, and θ is the set of trainable parameters.

Drawing inspiration from [69], g_θ integrates an attention mechanism atop the shared feature extraction module employed by f_θ, thus sharing the backbone parameters between the two functions.

After each stage, four convolutional layers are incorporated to generate feature tensors. These tensors are combined to form an attention map, which is normalized using the Softmax function. Within this attention map, each feature map functions as a soft mask to highlight probable attribute sites. Attribute-specific features are extracted by performing bilinear pooling between the attention map and the image feature tensor, involving pixel-wise multiplication and average pooling to identify attributes, as detailed in [69].

ViT-based module. An attention mechanism-based method for attribute localization, analogous to the CNN-based framework, is developed. The features derived from the backbone are passed through a convolutional layer to produce the attended feature tensor. Max-pooling is employed to create a soft mask for each attribute localization. Ultimately, each attribute feature is bi-linearly combined with the image feature to extract the attribute-level feature.

Attribute features weight

The concept of Attribute Weight (W_a), represented as a vector ($W_a \in \mathbb{R}^A$), is introduced as a data-driven gating mechanism to compute image-level attention. In the contexts of GZSL and ZSL, attributes are generally defined at the class level without individual annotations for each image. The purpose of the vector W_a is to identify class-associated attributes that are not visually detectable by applying a threshold th_{att} on the activation. Although W_a may seem theoretically redundant, it notably reduces the computational burden required to calculate $\mathcal{K}$. Further details on this efficiency enhancement will be elaborated later in the chapter.

7.1.3 Prototypes for semantic mapping

Within the shared embedding space of dimension M, each entity, whether a class, macroclass, or attribute, is represented by a prototype embedding. These prototypes act as reference points for classifying input images through comparison with their corresponding embeddings.

Prototype functions

Various functions, collectively represented as Π_θ (where θ indicates the parameters), are used to learn prototypes for categories, macrocategories, and attributes.

- **Class Prototypes**: The attribute matrix a (dimensions $C \times A$) is mapped to Class Prototypes in the shared space M using the function denoted as Π_θ^C. Here, C denotes the number of classes, and A denotes the number of attributes.
- **MacroClass Prototypes**: Analogously to class prototypes, the attribute matrix a_{macro} (dimensions $C_{\text{macro}} \times A$, where C_{macro} signifies the number of macroclasses) is transformed into MacroClass Prototypes within the shared space M using a function symbolized as Π_θ^M. These prototypes consolidate the C classes into a reduced set of C_{macro} macroclasses based on attribute similarities. A trainable matrix $a_{macro} \in \mathbb{R}^{C_{macro} \times A}$ is utilized to identify the attributes relevant to each macroclass that are absent from the ground truth.

- **Attribute Prototypes**: Individual attributes are projected in a common space using a mapping function Π_θ^M that takes the identity matrix as input, representing the one-hot encoded attribute labels.

It should be noted that the initial layers of the network modules implementing all three prototype functions (Π_θ^C, Π_θ^M, and Π_θ^A) share the same weights.

7.1.4 Logic Tensor Network

Within the framework of the LTN, the learning process is redefined to identify the weights that enhance the satisfiability of a knowledge base, denoted as $\mathcal{K}$. For each training batch, $\mathcal{K}$ is assembled from axioms derived from both labeled training instances (ϕ_{class}) and existing domain knowledge ($\phi_{macro}, \phi_{masked}, \phi_{attr}, \phi_5, \phi_6$). Further elaboration on these components will be provided in later sections of this chapter.

Variables

The following variables are defined:

$$\mathcal{G}(x) = \mathbb{R}^{Ch \times H \times W} \tag{7.3}$$
$$\mathcal{G}(l) = \mathbb{N}^{C} \tag{7.4}$$
$$\mathcal{G}(l_m) = \mathbb{N}^{C_{macro}} \tag{7.5}$$
$$\mathcal{G}(l_a) = \mathbb{N}^{C \times A} \tag{7.6}$$
$$\mathcal{G}(w_a) \in \mathbb{R}^{A} \tag{7.7}$$
$$\mathcal{G}(a) = \mathcal{G}(a_{\text{mask}}) = \mathbb{R}^{C \times A} \tag{7.8}$$
$$\mathcal{G}(a_{\text{macro}}) = \mathbb{R}^{C_{macro} \times A} \tag{7.9}$$

where x denotes an RGB image with dimensions $Ch \times H \times W$, l refers to class labels, l_m represents macroclass labels, l_a signifies attribute labels, C indicates the total number of classes, C_{macro} represents the total count of macroclasses, A represents the entire count of attributes, and M denotes the dimensions of the embedding space.

Predicates and functions

This section discusses the predicates used for image classification within the FLPN framework. In addition, it explains the feature extraction functions outlined in subsection 7.1.2, where f and g represent the functions that generate class and attribute embeddings. In addition, class-specific semantic prototypes are obtained through the functions Π^C_θ, Π^M_θ, and Π^A_θ, as detailed in subsection 7.1.3.

Four primary predicates are identified. The $\texttt{isOfClass}(x,l)$ predicate evaluates the probability that an image x belongs to class l. This probability is determined by calculating the cosine similarity between the global features of the image, extracted by f, and the prototype class represented by $\Pi^C_\theta(a_l)$, where a_l indicates the attribute vector for class l. This probability is then calculated using the Softmax function, as shown in Equation 7.10.

$$p(x,l) = \frac{\exp\left(\delta \cos(f_\theta(x^T), \Pi^C_\theta(a_l))\right)}{\sum_{s=1}^{S} \exp\left(\delta \cos(f_\theta(x^T), \Pi^C_\theta(a_s))\right)} \tag{7.10}$$

The score indicating whether a data point x belongs to class c (denoting the truth value of the predicate $\texttt{isOfClass}$ for input x and class c) is calculated by finding the dot product of the transposed one-hot encoded class label l_c^T and the result of the function p as shown in Equation 7.10:

$$\mathcal{G}(\texttt{isOfClass})(x,l_c) = l_c^T p(x,l_c) \tag{7.11}$$

A masked variant, denoted as $\texttt{isOfClass}_{\text{masked}}$, is introduced. The grounding function $\mathcal{G}(\texttt{isOfClass})(x,l_c)$ is similar to that of the predicate $\texttt{isOfClass}$ but utilizes a masked attribute vector, a_c^{masked}. Following the methodology outlined in FLVN, this predicate is designed to mitigate potential discrepancies between image-level and class-level attributes. To effectively manage this variability and represent images that may not present certain class attributes, the masked attribute vector $a_{\text{mask}} \in \mathbb{R}^{C\times A}$ is used. This vector originates in the same domain as the attribute vector a_c, with k elements randomly set to 0. Unlike FLVN, FLPN removes only the attributes in a_c with scores below the class average, as these attributes are observed less frequently in the class.

The function $\texttt{isOfMacro}(x, l_m)$ calculates the probability that an image x belongs to the macroclass m. It is determined based on the distance between the embedding of the image x and the prototype embedding representing the macroclass l_m:

$$\mathcal{G}(\texttt{isOfMacro}) : x, l_m \rightarrow \mathrm{e}^{-\alpha+\alpha\cdot\mathrm{d}(f_\theta(x),\Pi_\theta^M(a_m))} \tag{7.12}$$

where x is an input image, l_m is the macroclass label, a_m is the attribute matrix, and $\mathrm{d}(\cdot,\cdot)$ is a distance metric. In the studies conducted, $\mathrm{d}(\cdot,\cdot)$ is defined as the cosine distance, which differs from the methods used in PROTO-LTN [36] and DEM [62]. $\mathcal{G}(\texttt{isOfMacro})$ reaches a value of 1 when the distance to the macroclass prototype is zero.

The `hasAttribute` predicate determines the probability that an image x possesses the attribute l_a. It is calculated as a function of the cosine distance between an embedding and an attribute prototype:

$$\mathcal{G}(\texttt{hasAttribute}) : x, l_a \rightarrow \mathrm{e}^{-\alpha+\alpha\cdot\mathrm{d}(g_\theta(x),\Pi_\theta^A(a_{eye}))} \tag{7.13}$$

Experimentally, improved results were observed when using different distance metrics for the predicates `isOfClass`, `isOfMacro`, and `hasAttribute`. It is hypothesized that the improved efficacy of `isOfClass` grounding (Equation 7.11) arises from fostering class exclusivity. In contrast, the grounding of `isOfMacro` and `hasAttribute` encourages the overlap of class attributes, thereby improving performance.

7.1.5 Knowledge base axioms

Following the introduction of variables and predicates in the FOL language, this section defines the knowledge base $\mathcal{K}$ utilized for FLPN training.

Learning from labeled examples

The formula ϕ_{class} encourages the predictions to align with the provided data labels (e.g., "This image is a zebra").

$$\phi_{class} = \forall \mathrm{Diag}(x, l_c)(\mathtt{isOfClass}(x, l_c))^1 \tag{7.14}$$

To incorporate prior knowledge regarding class hierarchies, the axiom ϕ_{macro} is introduced to assert that any image classified under a specific class should also be classified under the corresponding macroclass (e.g., "zebras are part of the ungulate class"):

$$\begin{aligned}\phi_{macro} = \forall \mathrm{Diag}(x, l_c, l^c_{macro})\ (&\mathtt{isOfClass}(x, l_c) \\ &\implies \mathtt{isOfMacro}(x, l^c_{macro}))\end{aligned} \tag{7.15}$$

where l^c_{macro} denotes the macroclass associated with class c.

Learning with refutation

The formula ϕ_{masked} encapsulates the idea that class attributes are not universally present in every instance (e.g., "there exists a zebra that is not agile"). To address this discrepancy, a method is proposed that randomly obscures k attributes per class during the computation of class prototypes. This involves averaging the nonzero elements of the attribute vector a_c for each class and then masking the k attributes that fall below this average. Alternative approaches, such as arbitrary selection regardless of the attribute matrix a, were tested, but empirically found to be less effective. The axiom is framed using the $\mathtt{isOfClass}_{\mathrm{masked}}$ predicate:

$$\phi_{masked} = \forall l^{\mathrm{seen}}(\exists x\ \mathtt{isOfClass}_{\mathrm{masked}}(x, l^{\mathrm{seen}})) \tag{7.16}$$

where l^{seen} represents the list of observed classes.

Learning with attention

An additional axiom is integrated into the knowledge base, promoting proximity of images with similar attributes in the embedding space, independent of their respective classes:

[1] Diagonal Quantification quantifies over pairs of instances, such as images and their labels. A more detailed definition is available in [40].

$$\phi_{attr} = \forall \text{Diag}(x, w_a, l_a^c)\Big(w_a[l_a^c] > th_{att}\ \texttt{hasAttribute}(x, l_a^c)\Big) \tag{7.17}$$

In this context, x denotes an input image, l_a^c signifies class-related attributes, and w_a represents the attribute weight vector. For increased computational efficiency, attention is focused on attributes l_a that are relevant to the class (as indicated by the attribute matrix a) and likely present in the image x, considering only those attributes l with weights w_a^l above a threshold th_{att}. Practically, since the knowledge base $\mathcal{K}$ is refreshed with each training batch, the prevailing attention values are used to enforce this criterion.

Learning better feature representation

Classes are generally characterized by a collection of attributes. To group similar image representations within the same class, the axiom ϕ_5 is introduced.

$$\begin{aligned}\phi_{simil} = \forall l^{\text{seen}}, \forall x, \forall y : \cos(f_\theta(x), f_\theta(y)) > th_{sg} :\\ \texttt{isOfClass}(x, l^{\text{seen}}) \Leftrightarrow \texttt{isOfClass}(y, l^{\text{seen}})\end{aligned} \tag{7.18}$$

Equation 7.18 encapsulates the notion that if two images, x and y, share the same class label (l^{seen}), they should be classified together based on their embeddings. To maintain a manageable size for the knowledge base $\mathcal{K}$, this axiom is applied only to pair of images (x, y) with a cosine similarity cos between their embeddings that exceeds a specified threshold th_{sg}.

In contrast, axiom ϕ_6 is introduced to ensure that images with different embeddings are not classified into the same class. This is expressed by the axiom ϕ_6:

$$\begin{aligned}\phi_{dissimil} = \forall l^{\text{seen}}, \forall x, \forall y : \cos(f_\theta(x), f_\theta(y)) < th_{sl} :\\ \neg(\texttt{isOfClass}(x, l^{\text{seen}}) \wedge \texttt{isOfClass}(y, l^{\text{seen}}))\end{aligned} \tag{7.19}$$

In this context, $f_\theta(x)$ and $f_\theta(y)$ denote visual embeddings of different images. Similarly to the criteria defined in the axiom ϕ_{simil} (Equation 7.18), only image pairs with cosine similarity cos below a threshold th_{sl} are considered. These axioms (ϕ_{simil} and $\phi_{dissimil}$) promote a distribution of image embeddings in which similar representations are grouped within the same classes while maintaining distinct separations between different classes. This strategy prevents excessive reliance on prior knowledge from the attribute matrix.

Grounding logical connectives and aggregators

Different operator semantics were investigated, identified here as LTN [40]. LTN operates within the probability space. Table 7.1 presents a comparative summary of the main differences between logical connectives and aggregation operators implemented in the conventional LTN framework.

Connective	**LTN**
$\neg$	$1-a$
$\wedge$	ab
$\vee$	$a+b-ab$
$\rightarrow$	$1-a+ab$
$\forall$	$1-\left(\frac{1}{n}\sum_{i=1}^{n}(1-a_i)^p\right)^{\frac{1}{p}}$
$\exists$	$\left(\frac{1}{n}\sum_{i=1}^{n}a_i^p\right)^{\frac{1}{p}}$

Table 7.1 Comparison of connectives and aggregators in LTN ($p \geq 1$)

Querying the knowledge base

During the inference phase, the predicted class is identified as the one with the highest score:

$$\hat{y} = \underset{\tilde{y} \in Y^U}{argmax}\left(\delta \cos(x^{\mathrm{T}}, C f_\theta(a_{\tilde{y}}))\right) \tag{7.20}$$

To mitigate this bias between seen and unseen images, the calibrated stacking method was used (as in FLVNEquation 6.16).

7.2 Experimental setup

This section details the datasets used in the experiments, the corresponding knowledge bases used for each dataset, and the specified list of hyperparameters.

Dataset. The experiments utilized the Awa2 dataset, with GZSL evaluation metrics aligned with standards established in previous studies. Following the methodology described in [98], a semantic hierarchy was constructed by organizing the classes from the Awa2 dataset into 9 macro-classes.

Hyperparameter selection. The embedding function f_θ utilizes an ImageNet pre-trained ResNet101 model to transform a 224×224 image into a vector $\mathbf{x} \in \mathbb{R}^{H \times W \times B}$, where $B = 2048$, and H and W denote the height and width of the extracted features. The function g_θ then projects these features in the attribute space, maintaining dimensional consistency with the dataset.

For the ViT backbone, the head is initially trained with the frozen backbone (warmup), followed by fine-tuning of the entire network to mitigate overfitting. During the pre-training phase, the Adam optimizer is used with a learning rate of $1e-4$. In the fine-tuning phase, the learning rate progressively decreases to $1e-7\alpha$, where $\alpha = 0.8^{epoch//10}$, over a total of 300 epochs.

For the architecture with the ResNet-101 backbone, an initial learning rate of $1e-3$ is applied, with the model being trained end-to-end using the SGD optimizer. The learning rate is subsequently reduced to $1e-7\alpha$, where $\alpha = 0.8^{epoch//10}$, similar to ViT training, for a total of 5 epochs. In both scenarios, regularization is enhanced by doubling the L2 norm with a scaling factor of $5e-4$.

When training the ResNet-101 backbone, batches of 64 samples are randomly selected, while for ViT, training batches comprise an equal ratio of 12 positive and 12 negative Awa2 instances. The scaling factor γ is set to 0.7 during inference to adjust the scores for the seen classes. For data augmentation, random flipping with a probability of 0.5 and random cropping are employed in each experiment. Using the LTNtorch library [99], the architectures are implemented in PyTorch and trained on a single Nvidia 3090 Ti GPU. Each experiment is carried out three times to report the mean and standard deviation.

7.3 Results

This section presents a comparative analysis of the architecture developed in the zero-shot learning challenge against state-of-the-art models. Specifically, the models are compared with multiple methods: **Embedding based methods**(PROTO-LTN [36], DEM [62], VSE [63], TCN [66], CSNL [98]), **Attention-based techniques** (AREN [68], APN [60], AMGML [67], CC-ZSL [61], CoAR-ZSL [69], FLVN [37]), **Generative techniques** (Cycle-CLSWGAN [74], LisGan [73], E-PGN [71], TGMZ [72], CEGZSL [75], DFCA-GZSL [100]) and lastly **Tranformer-based architectures** (COAR [69]).

The experimental results are reported in Table 7.2, which shows how FLPN is competitive with the state-of-the-art. Specifically, the devised architecture achieves better results in terms of H both for CNN-based and transformer-based architectures, demonstrating better results on classes not observed in both a ZSL and GZSL settings, as well as on classes observed in a GZSL setting.

Compared to the state of the art, the *CNN-based approach* performs better on unseen classes. FLPN outperforms FLVN [37] by 4.0 and 3.1 points, respectively, obtaining a value of 73.8 for unseen classification in ZSL (T1) and an H value of 75.3. The best performing method in ZSL [72] (T1 = 78.4), does not perform as well in the GSZL scenario (77.3 accuracy on seen classes and 64.1 on unseen ones). This demonstrates how the introduced axioms provide additional information that allows the architecture to represent features that identify unseen classes, without losing much information about the seen classes. Unlike generative models, and like FLVN and Proto-LTN, the approach does not require the generation of synthetic images. Moreover, FLPN converged relatively fast in roughly 5 training epochs, in contrast to other comparable works in terms of backbone [61, 69] and/or neurosymbolic approaches [36, 37].

With a *transformer-based* backbone, FLPN still achieves competitive results with respect to the state-of-the-art. In particular, the H value for the Awa2 dataset increases by nearly 2.9 percentage points, from 82.0 to 84.9, over the closest comparator.

Model	T1	U	S	H
Embedding-based				
PROTO-LTN [36]	67.6	32.0	83.7	46.2
DEM [62]	67.1	30.5	**86.4**	45.1
VSE [63]	84.4	45.6	88.7	60.2
TCN [66]	71.2	61.2	65.8	63.4
CSNL [98] †	61.0	-	-	0.0
Attention-based				
AREN [68]	67.9	54.7	79.1	64.7
APN [60]	68.4	56.5	78.0	65.5
AMGML [67]	71.7	56.0	74.6	64.0
CC-ZSL [61]	68.8	62.2	<u>83.1</u>	71.1
CoAR-ZSL(Resnet-101) [69]	74.1	<u>68.1</u>	79.1	73.2
FLVN (Resnet-101) †	69.8	65.8	82.3	73.1
Generative-based				
Cycle-CLSWGAN [74]	-	-	-	-
LisGan [73]	-	-	-	-
E-PGN [71]	73.4	52.6	83.5	64.6
TGMZ [72]	**78.4**	64.1	77.3	70.1
CEGZSL [75]	70.4	63.1	78.6	70.0
DFCA-GZSL [100]	<u>74.7</u>	66.5	81.5	<u>73.3</u>
FLPN (ViT) †	73.8	**71.2**	79.9	**75.3**
	±1.0	±0.9	±1.1	±0.7
	(74.8)	(72.1)	(81.0)	(76.0)
Transformer-based				
CoAR-ZSL(ViT) [69]	78.7	76.5	88.2	82.0
FLPN (ViT) †	**80.2**	**79.7**	**88.2**	**83.7**
	±2.0	±1.9	±0.8	±1.0
	(82.2)	(81.6)	(89.9)	(84.7)

Table 7.2 Comparing Awa2 for T, U, S, and H for GZSL e ZSL to the state of the art at the moment. The methods are broken down into four categories: embedding-based models, attention mechanism-based models, transformer-based models and generative models. The best results are in bold, while the second-best results are underlined.

7.3.1 Ablation studies

Ablation studies in Table 7.3 were conducted to clarify the contribution of each axiom.

	KB						Awa2			
Model	ϕ_{class}	ϕ_{macro}	ϕ_{masked}	ϕ_{attr}	ϕ_{simil}	$\phi_{dissimil}$	T1	U	S	H
FLPN(Resnet-101)	✓	-	✓	-	-	-	66.3	62.2	84.7	71.7
FLPN(Resnet-101)	✓	-	✓	✓	-	-	69.8	66.4	80.2	72.7
FLPN(Resnet-101)	✓	✓	✓	✓	-	-	70.9	67.7	79.3	73.1
FLPN(Resnet-101)	✓	✓	✓	✓	✓	✓	72.9	71.0	78.4	74.5
FLPN(ViT)	✓	-	✓	-	-	-	79.8	78.9	89.6	83.9
FLPN(ViT)	✓	-	✓	✓	-	-	80.9	80.3	89.2	84.5
FLPN(ViT)	✓	✓	✓	✓	-	-	83.1	82.6	85.2	83.9
FLPN(ViT)	✓	✓	✓	✓	✓	✓	83.0	82.4	87.6	84.9

Table 7.3 Ablation study at different level of knowledge base $\mathcal{K}$ based on different backbones

Role of ϕ_{class} and ϕ_{masked}. The foundational architecture incorporates axioms aimed at learning from annotated instances, while handling anomalies by intermittently masking attributes (axiom ϕ_{masked}(Equation 7.16)). Consequently, it emphasizes observed classes (S=84.7 in Table 7.3). Each class possesses intrinsic traits that persistently manifest; for example, a zebra will invariably have stripes, even in a cropped image. Initial experimental results indicate that the axiom ϕ_{masked} proves particularly effective when the k elements are selected from attributes with values below the class average.

Role of ϕ_{attr}. Based on the matrix of attributes present for a certain class and an attribute weight w_a, which denotes the presence of the attribute in the particular image, a similarity between the attributes present in the image is used using ϕ_{attr} (Equation 7.13). The knowledge base implements constraints that allow the network to focus on different parts of the image. Table 7.3 shows an improvement in the classification of unseen classes of 3.5 and 4.2 points, respectively, in both the ZSL and GZSL (T1 and U) settings.

Role of ϕ_{macro}. After introducing the axiom ϕ_{macro}, the harmonic mean slightly improves from 72.7 to 73.1, indicating that including semantically relevant restrictions with macroclasses has a relatively minor impact on the results. In Figure 5.2, PROTO-LTN (Chapter Chapter 5) shows that the model inherently learns to cluster prototypes within the feature space, even in the absence of explicit hierarchical constraints. This is evident from the t-SNE plot presented in Figure 7.10. Despite similar overall results (as demonstrated by H), there is an improvement in the performance on unseen classes, both in the ZSL (T1) and GZSL (U) settings, and with both backbones. Aggregating classes in a smaller feature space that benefits unseen samples may be responsible for the phenomenon.

Role of ϕ_{simil} and $\phi_{dissimil}$. These two axioms deal with the relationship between images without taking into account the class to which they belong. The classification of unseen classes (T) improves by about 2-3 percentage points in all settings when using a CNN backbone.

7.3.2 Class feature visualization using t-SNE

t-SNE visualizations are used in Figure 7.10 to illustrate how the classes are divided in the feature space. In particular, the results of the experiments conducted using two different knowledge bases are compared: the one composed by axioms $\phi 1$ and ϕ_{masked}, and the one with all the axioms presented in this study. Specifically, the embedding space in a GZSL scenario using test images is examined, taking into account the seen and unseen classes independently. Only 500 samples per class are included, each of which is represented by a distinct color, for a better visualization.

Reducing the number of overlapping examples also improves classification, as previously demonstrated in Table 7.2. Figure 7.3 and Figure 7.7 demonstrate that the introduction of more specific axioms for the classification of classes seen in a GZSL setting allows, for both ResNet-101 and even more so for ViT, to obtain a more separate representation of the classes.

In Figure 7.10, with regard to the classification of classes not seen in a GZSL context, it is observed that there is initially some overlap between some of the architectures, which is more pronounced than the classes seen. Figure 7.5 and Figure 7.9 illustrate how the additional axioms of the knowledge base have reduced this initial overlap for several of the observed classes.

7.3.3 Visualization of attribute-level attention maps

Table 7.2 shows that with the `hasAttribute` predicate it is possible to achieve better overall classification results. In this subsection, attention is shown to help localize specific attributes for both the CNN- and transformer-based architectures shown

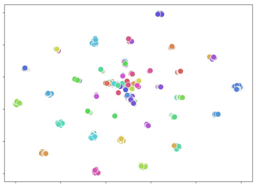


Fig. 7.2 FLPN with ResNet-101 backbone trained with a KB composed by ϕ_{class} and ϕ_{masked}: predictions of seen classes in GZSL setting

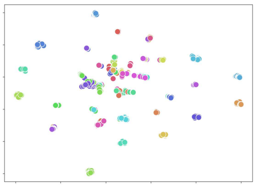


Fig. 7.3 FLPN with ResNet-101 backbone trained with a KB composed by all axioms: predictions of seen classes in GZSL setting

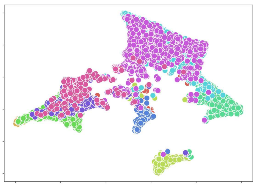


Fig. 7.4 FLPN with ResNet-101 backbone trained with a KB composed by ϕ_{class} and ϕ_{masked}: predictions of unseen classes in GZSL setting

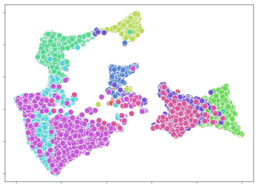


Fig. 7.5 FLPN with ResNet-101 backbone trained with a KB composed by all axioms: predictions of unseen classes in GZSL setting

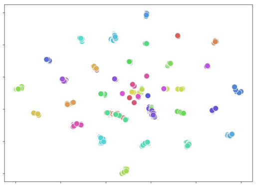


Fig. 7.6 FLPN with ViT backbone trained with a KB composed by ϕ_{class} and ϕ_{masked}: predictions of seen classes in GZSL setting

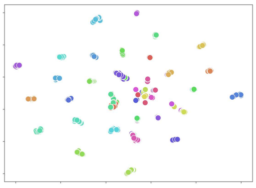


Fig. 7.7 FLPN with ViT backbone trained with a KB composed by all axioms: predictions of seen classes in GZSL setting

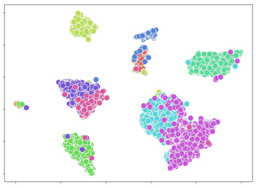


Fig. 7.8 FLPN with ViT backbone trained with a KB composed by ϕ_{class} and ϕ_{masked}: predictions of unseen classes in GZSL setting

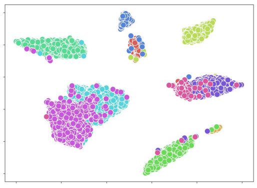
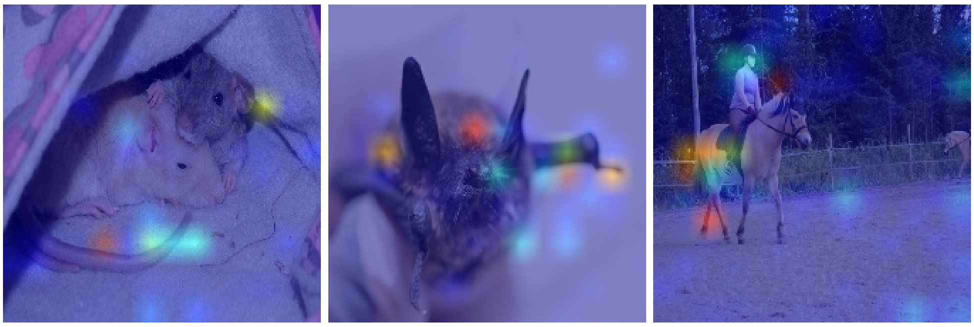


Fig. 7.9 FLPN with ViT backbone trained with a KB composed by all axioms: predictions of unseen classes in the GZSL setting

Fig. 7.10 The class-level feature distribution for FLPN with the ResNet-101 (first row) and ViT (second row) backbones is represented using t-SNE. The likelihood scores generated by the model for every class were utilized to produce this representation.

in Figure 7.12 and Figure 7.11. It is evident from both examples that the architecture accurately pinpoints the regions of the picture that possess a certain property. Qualitatively, the CNN-based architecture is less accurate than the transformer-based architecture. As noted in a previous work [69], there is significant noise in both scenarios, since each sample lacks an annotation at the attribute level.



(a) has tail (b) can fly (c) has longleg

(d) has longneck (e) has flippers (f) has flippers

Fig. 7.11 FLPN with ViT-backbone attributes attention on Awa2

7.3.4 Architecture comparison

The proposed architecture represents an advancement and enhancement of the earlier models, Proto-LTN [36] and FLVN [37], both of which are founded on similar NeSy principles. Table 7.4 outlines the distinctions among the three architectures in terms of the knowledge base $\mathcal{K}$ utilized for model training, the *embedding space* employed to measure similarity between image embeddings and class prototypes, the *grounding* of the `isOfClass` predicate, the construction of the *Batch* for the training set, the *Aggregator* that merges all axioms to determine final satisfiability, and the *backbone*.

Model	KB	Common Space	End-to-End Training	`isOfClass`	Batch	Aggregator	Backbone
PROTO-LTN [36]	$\{\phi_{class}\}$	Image Space	-	Euclidean distance	random sampler	logProduct	ResNet-101
FLVN [37]	$\{\phi_{class}, \phi_{macro}, \phi_{masked}, \phi_{simil}, \phi_{dissimil}\}$	Attribute Space	✓	dot product	Shots=12, Ways=8	Mean-Error	ResNet-101
FLPN(Resnet-101)	$\{\phi_{class}, \phi_{macro}, \phi_{masked}, \phi_{attr}, \phi_{simil}, \phi_{dissimil}\}$	Attribute Space	✓	dot product	random sampler	Mean-Error	ResNet-101
FLPN(ViT)	$\{\phi_{class}, \phi_{macro}, \phi_{masked}, \phi_{attr}, \phi_{simil}, \phi_{dissimil}\}$	Attribute Space	✓	dot product	Shots=12, Ways=8	Mean-Error	ViT

Table 7.4 Comparison of the three NeSy architectures, with commonalities and differences highlighted.

Proto-LTN vs FLPN While Proto-LTN [36] is substantially different from FLPN, it is nonetheless built on a similar assumption. Although there are some

(a) is quadrapedal (b) has tail (c) can fly

(d) has tail (e) has longleg (f) has flippers

Fig. 7.12 FLPN with ResNet-101-backbone attributes attention on Awa2

differences in the final aggregator, the adoption of a new knowledge base and the way the `isOfClass` predicate is implemented are the key distinctions. Proto-LTN employs Euclidean distance to increase the similarity between images and prototypes that belong to the same class in a case where the backbone is frozen and training is restricted to the prototype network. By applying a multiplication between image and prototype features using a softmax activation function, FLPN, in contrast, uses a different similarity metric, that encourages mutual exclusiViTy between different classes. Since only positive examples are used for classes during training, encouraging mutual exclusivity compensates for the lack of negative examples and allows the network to be trained end-to-end. Experimentally, Proto-LTN tends to overfit to seen classes when the backbone is not kept frozen.

FLVN vs FLPN. If the embedding space and ViT backbone are ignored, FLVN and FLPN may be regarded as similar designs. However, because of the way FLVN is constructed, it cannot compute prototype vectors for attributes such as AP_θ). Although FLVN does not explicitly compute prototype vectors for classes, it does provide a multiplication of the image features (appropriately converted into the attribute space) with the semantic matrix of the attributes. Therefore, the architecture may include the introduction of a prototype network for the attributes but without sharing the same structure of the classes (it is worth recalling that in FLPN, the prototype networks share the initial layers). Another significant distinction is the amount of time needed for training; FLVN needed about 300 epochs to train, while FLPN only needed about 5.

Adopting a network based on ViT obviously prevents comparisons with earlier architectures because it is a more performing network with many more parameters and has been trained on a larger amount of data, but makes it possible to demonstrate how this architecture can be used in various contexts and with various backbones.

7.4 Conclusion

Proto-LTN, FLVN, and FLPN are three new NeSy designs that are introduced for ZSL and GZSL tasks, based on ideas from the current ZSL literature, and embedded within a NeSy framework. In order to handle exceptions within the dataset (e.g., "there exists a zebra that is not agile") and establish relationships between images (e.g., "if two images belong to the same class, they must be similar"), FLVN and FLPN incorporate axioms that combine prior class-specific knowledge (e.g., class hierarchies) with high-level inductive biases. These axioms serve as semantic priors that compensate up for the absence of annotations, giving GZSL tasks a strong NeSy base.

These architectures maintain approximately the same number of parameters as typical embedding-based techniques. The suggested methods can also be used to different architectures by altering the predicates' grounding, for example. Empirical findings demonstrate that it attains performance comparable to or better than that of recent research on widely used GZSL benchmarks.

There are many ways in which this method might be expanded upon. To improve explainability, one may pay more attention to discriminative areas within the image by introducing additional predicates to anticipate image-level features or rephrasing the `isOfClass` predicate. Without necessitating extra labels in the dataset, axioms that only take into account how similar two photos are to one another (without taking into account the images' class membership) may yield better results for classes that have not yet been seen. In addition, the knowledge base might be expanded to take into account interactions of a different kind, one that goes beyond class hierarchies and emerges from new sources, like language models. Lastly, a transductive ZSL scenario might be used to investigate the suggested approach.

Chapter 8

Concluding remarks

This doctoral dissertation summarizes the research conducted during the doctoral path, which centered on developing NeSy methods to enhance the robustness of machine learning models when handling high-dimensional data, such as images. The proposed methodologies are capable of learning from both the distributed data and from prior knowledge related to the underlying concepts present in the images. By making deep learning models more reliable and interpretable, these designs can be utilized in various business domains, offering significant benefits. Incorporating logical constraints can compensate for the knowledge gaps that cannot be statistically derived from the input data, thereby addressing, at least partially, well-known deep learning challenges such as noise, scarcity of labeled data, sensitivity to biases, and fairness issues.

This doctoral dissertation has been structured to emphasize how LTNs were applied to increasingly complex architectures in computer vision tasks, offering improved performance and demonstrating their potential in tackling sophisticated challenges. In order to demonstrate how NeSy architectures, such as LTNs [43], can be applied in a complementary manner to CNNs and transformers, effectively sharing knowledge between the convolutional model and the LTN knowledge base, two tasks were analyzed in depth in this dissertation: object detection and classificatio in a ZSL setting. The goal is to improve the generalization capabilities of the overall architecture by leveraging knowledge expressed in symbolic form to integrate semantically connected concepts. The final goal is to create an architecture that incorporates symbolic knowledge in the learning process to improve the quality of the features learnt by the convolutional model.

The integration of LTNs in an object detection task is examined in Chapter 4. Specifically, the LTN replaced the model classification layers of a Faster-CNN [1] object detection network. Building on previous works by Donadello et al. [41], it is demonstrated that it is possible to train the entire architecture in an end-to-end fashion, whereas in previous work the LTN was trained separately. This is a crucial step to allow the CNN backbone to benefit from the NeSy loss, rather than employing the LTN to merely refine the network's prediction. A knowledge base with the labeled samples can be embedded in the training loss by grounding the `isOfClass` predicate through a trainable layer. The experimental results show that Faster-LTN outperforms the original Faster-CNN architecture. By applying extra constraints that limit the classification space of the image's objects, mereological constraints provide additional information that further improves the classification of the detected ROIs. An additional analysis is provided to demonstrate the approach's potential by assessing the architecture's performance while training on reduced datasets.

The second part of this doctoral dissertation analyzes in depth a ZSL setting. If every class symbol in the dataset is associated with a vectorized representation, it is possible to create class prototypes that live in the same embedding space as the images by suitably converting them using a neural network. In Proto-LTN, in which the CNN network is frozen and only the LTN is trained, the `isOfClass` predicate is grounded based on the Euclidean distance between class prototypes and image embeddings.

FLVN (Chapter 6) improves Proto-LTN by allowing more effective information exchange between the CNN and the LTN, supported by a more comprehensive knowledge base. In Proto-LTN, the positive axioms are designed to minimize the distance between image embeddings and their corresponding class prototypes using the Euclidean distance metric. This setup, combined with a frozen backbone, prevents the collapse of prototypes by maintaining a stable representation space, thus ensuring that the learned prototypes retain meaningful separability across classes. Furthermore, FLVN shows that the model can be trained end-to-end if the predicate `isOfClass` is grounded as a multiplication between image features and class attributes in a common space, with the inclusion of a softmax activation function.

Additionally, the knowledge base incorporates prior knowledge regarding the classified objects: classes and macroclasses are arranged hierarchically ("if an image belongs to a class, it must also belong to its macroclass"); images that belong to a class but do not possess all of its properties, with at least one exception for each class; and images' relationships ("two images that have similar features belong to the same class"). The architecture is compared with embedding-based models (like Proto-LTN), architectures that introduce attention mechanisms and are trained end-to-end, and generative models (which learn a relationship between seen classes and attributes with the goal of generating examples for unseen classes). The results indicate that it is possible to achieve improvements over the state-of-the-art results without introducing additional layers to the backbone architecture. This demonstrates that the enhancements in performance is achieved through an improved training process, rather than by increasing the complexity of the model.

In the end, FLPN (Chapter 7) combines the best aspects of previous architectures; it uses the prototype space as an embedding space, similar to Proto-LTN, but it also allows for end-to-end training and includes a sophisticated knowledge base, similar to FLVN. With the help of this attention mechanism, the final design can incorporate a knowledge base that considers both the global properties of the image and the attributes that are specific to a given image or class. The study assesses the information collected by the architecture as the axioms incorporated in the knowledge base change, taking into consideration an alternative and more sophisticated backbone to extract the features. The results obtained achieve state-of-the-art performance in terms of performance and explainability.

All studies have limitations that could be addressed in future work. Faster-LTN should be validated on larger and more complex datasets, which will likely require alternative implementations of the final aggregator or alternative implementations of the `isOfClass` predicate. Additionally, the introduction of partially labeled datasets could be evaluated (for example, providing a description of the parts for some classes, in lieu of labeled examples), further reducing the lack of supervision.

One of the crucial points of these systems concerns their scalability. As the complexity of the datasets increases, the number of rules used to describe the knowledge base may also grow, making it difficult to optimize the overall satisfiability. The way in which various predicates are grounded can also add complexities in terms of resources required to train the model.

For the ZSL tasks, the next objective could be to investigate the generalizability of the approach with different models and datasets to improve the learning of seen and unseen classes. Improving individual attribute learning could open the way to new knowledge bases capable of better describing each individual class, for example "if an image has black and white stripes and is similar to a horse then it represents a zebra". The lack of attribute-level supervision for the individual image is another obstacle to overcome.

In general, NeSy architectures allow the introduction of logical constraints in the loss function, unlike the standard training cycle of deep learning networks. However, the creation of the knowledge base and its rules represents one of the most significant barriers to adoption. Firstly, constructing the knowledge base requires considerable effort, necessitating domain experts capable of annotating the correct relationships between extracted concepts. Secondly, it is essential to develop a knowledge base that accounts for exceptions to general rules, which are often expressed in principle but may not always be consistent with complex real-world data. In addition, the quality of the knowledge base, the time required for annotation, and the feasibility of training on very large knowledge bases are critical challenges. To address these issues, methods could be evaluated to automate or semi-automate the process by introducing more general knowledge bases that can be adapted to more specific domains or by employing large language models to directly generate FOL statements. Therefore, addressing these challenges is crucial not only for the effective deployment of NeSy architectures but also for enhancing their applicability in diverse real-world scenarios, ultimately leading to more robust and interpretable AI systems.

References

[1] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. IEEE transactions on pattern analysis and machine intelligence, 39(6), 2016.

[2] Yongqin Xian, Christoph H Lampert, Bernt Schiele, and Zeynep Akata. Zero-shot learning—a comprehensive evaluation of the good, the bad and the ugly. IEEE transactions on pattern analysis and machine intelligence, 41(9), 2018.

[3] Haiying Wang, Estelle Pujos-Guillot, Blandine Comte, Joao Luis de Miranda, Vojtech Spiwok, Ivan Chorbev, Filippo Castiglione, Paolo Tieri, Steven Watterson, Roisin McAllister, et al. Deep learning in systems medicine. Briefings in bioinformatics, 22(2), 2021.

[4] Akshara Pramod, Harsh Sankar Naicker, and Amit Kumar Tyagi. Machine learning and deep learning: open issues and future research directions for the next 10 years. Computational analysis and deep learning for medical care: Principles, methods, and applications, 2021.

[5] Travers Ching, Daniel S Himmelstein, Brett K Beaulieu-Jones, Alexandr A Kalinin, Brian T Do, Gregory P Way, Enrico Ferrero, Paul-Michael Agapow, Michael Zietz, Michael M Hoffman, et al. Opportunities and obstacles for deep learning in biology and medicine. Journal of The Royal Society Interface, 15(141), 2018.

[6] Sorin Grigorescu, Bogdan Trasnea, Tiberiu Cocias, and Gigel Macesanu. A survey of deep learning techniques for autonomous driving. Journal of Field Robotics, 37(3), 2020.

[7] Leslie G. Valiant. Knowledge infusion: in pursuit of robustness in artificial intelligence. In Proceedings of the Foundations of Software Technology and Theoretical Computer Science, 2008.

[8] Wenguan Wang, Yi Yang, and Fei Wu. Towards Data-and Knowledge-Driven Artificial Intelligence: A Survey on Neuro-Symbolic Computing. arXiv e-prints, October 2022.

[9] Yann LeCun, Yoshua Bengio, and Geoffrey E. Hinton. Deep learning. Nature, 521, 2015.

[10] Ali Bou Nassif, Ismail Shahin, Imtinan B. Attili, Mohammad Azzeh, and Khaled F. Shaalan. Speech recognition using deep neural networks: a systematic review. IEEE Access, 7, 2019.

[11] Hao Wu, Qi Liu, and Xiaodong Liu. A review on deep learning approaches to image classification and object segmentation. Computers, Materials & Continua, 2019.

[12] Azhar Toilybaikyzy Tursynova, B. S. Omarov, O. A. Postolache, and M. Zh. Sakypbekova. Convolutional deep learning neural network for stroke image recognition: Review. Journal of Mathematics, Mechanics and Computer Science, 112(4), Dec. 2021.

[13] Kohei Watanabe and Yuan Zhou. Theory-driven analysis of large corpora: semisupervised topic classification of the un speeches. Social Science Computer Review, 40, 2020.

[14] Sristy Lalaika Vemula and Nisha Rathee. Bio-inspired feature selection techniques for sentiment analysis – review. 2023 Third International Conference on Artificial Intelligence and Smart Energy (ICAIS), 2023.

[15] Blanka Frydrychova Klimova, Marcel Pikhart, Alice Delorme Benites, Caroline Lehr, and Christina Sanchez-Stockhammer. Neural machine translation in foreign language teaching and learning: a systematic review. Education and Information Technologies, 28, 2022.

[16] Shun-ichi Amari. Backpropagation and stochastic gradient descent method. Neurocomputing, 5, 1993.

[17] Giuseppe Futia and Antonio Vetrò. On the integration of knowledge graphs into deep learning models for a more comprehensible ai—three challenges for future research. Information, 11(2), 2020.

[18] Tirtharaj Dash, Sharad Chitlangia, Aditya Ahuja, and Ashwin Srinivasan. A review of some techniques for inclusion of domain-knowledge into deep neural networks. Scientific Reports, 12(1), 2022.

[19] Nathan Drenkow, Numair Sani, Ilya Shpitser, and Mathias Unberath. A Systematic Review of Robustness in Deep Learning for Computer Vision: Mind the gap? arXiv e-prints, December 2021.

[20] James F Mullen Jr, Franklin R Tanner, and Phil A Sallee. Comparing the effects of annotation type on machine learning detection performance. In Proceedings of the conference on computer vision and pattern recognition workshops, 2019.

[21] Khaled Alhazmi, Walaa Alsumari, Indrek Seppo, Lara Podkuiko, and Martin Simon. Effects of annotation quality on model performance. In Proceedings of the 2021 International Conference on Artificial Intelligence in Information and Communication (ICAIIC). IEEE, 2021.

[22] Pascal Hitzler, Aaron Eberhart, Monireh Ebrahimi, Md Kamruzzaman Sarker, and Lu Zhou. neuro-symbolic approaches in artificial intelligence. National Science Review, 9(6), 03 2022.

[23] Artur d'Avila Garcez, Sebastian Bader, Howard Bowman, Luis C Lamb, Leo de Penning, BV Illuminoo, Hoifung Poon, and COPPE Gerson Zaverucha. Neural-symbolic learning and reasoning: a survey and interpretation. Neuro-Symbolic Artificial Intelligence: The State of the Art, 342(1), 2022.

[24] Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard Hovy, and Eric Xing. Harnessing Deep Neural Networks with Logic Rules. arXiv e-prints, March 2016.

[25] Michelangelo Diligenti, Marco Gori, and Claudio Sacca. Semantic-based regularization for learning and inference. Artificial Intelligence, 244, 2017.

[26] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Broeck. A semantic loss function for deep learning with symbolic knowledge. In Proceedings of the International conference on machine learning. PMLR, 2018.

[27] 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.

[28] Zhi-Hua Zhou. Abductive learning: towards bridging machine learning and logical reasoning. Science China Information Sciences, 62, 2019.

[29] Le-Wen Cai, Wang-Zhou Dai, Yu-Xuan Huang, Yu-Feng Li, Stephen H Muggleton, and Yuan Jiang. Abductive learning with ground knowledge base. In Proceedings of the IJCAI, 2021.

[30] Meng Qu and Jian Tang. Probabilistic Logic Neural Networks for Reasoning. arXiv e-prints, June 2019.

[31] Yuyu Zhang, Xinshi Chen, Yuan Yang, Arun Ramamurthy, Bo Li, Yuan Qi, and Le Song. Efficient probabilistic logic reasoning with graph neural networks. ArXiv, abs/2001.11850, 2020.

[32] Giuseppe Marra, Michelangelo Diligenti, Francesco Giannini, Marco Gori, and Marco Maggini. Relational neural machines. In Proceedings of the European Conference on Artificial Intelligence, 2020.

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

[34] Xiaochun Luo, Heng Li, and SangHyun Lee. Bridging the gap: neuro-symbolic computing for advanced ai applications in construction. Frontiers of Engineering Management, 10(4), 2023.

[35] Francesco Manigrasso, Filomeno Davide Miro, Lia Morra, and Fabrizio Lamberti. Faster-LTN: a neuro-symbolic, end-to-end object detection architecture. In Artificial Neural Networks and Machine Learning–ICANN 2021: 30th International Conference on Artificial Neural Networks, Bratislava, Slovakia, September 14–17, 2021, Proceedings, Part II 30. Springer, 2021.

[36] Simone Martone, Francesco Manigrasso, Fabrizio Lamberti, and Lia Morra. PROTOtypical logic tensor networks (PROTO-LTN) for zero shot learning. In 2022 26th International Conference on Pattern Recognition (ICPR). IEEE, 2022.

[37] Francesco Manigrasso, Lia Morra, and Fabrizio Lamberti. Fuzzy Logic Visual Network (FLVN): A neuro-symbolic approach for visual features matching. In International Conference on Image Analysis and Processing. Springer, 2023.

[38] Somak Aditya, Yezhou Yang, and Chitta Baral. Integrating knowledge and reasoning in image understanding. In Proceedings of the 28th International Joint Conference on Artificial Intelligence, IJCAI 2019. International Joint Conferences on Artificial Intelligence, 2019.

[39] Luc de Raedt, Sebastijan Dumančić, Robin Manhaeve, and Giuseppe Marra. From statistical relational to neuro-symbolic artificial intelligence. In Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, 2020.

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

[41] Ivan Donadello, Luciano Serafini, and Artur D'Avila Garcez. Logic tensor networks for semantic image interpretation. In Proceedings of the 26th International Joint Conference on Artificial Intelligence. AAAI Press, 2017.

[42] Ivan Donadello and Luciano Serafini. Compensating supervision incompleteness with prior knowledge in semantic image interpretation. In Proceedings of the International Joint Conference on Neural Networks, 2019.

[43] Luciano Serafini and Artur S. d'Avila Garcez. Learning and reasoning with logic tensor networks. In Proceedings of the International Conference of the Italian Association for Artificial Intelligence, 2016.

[44] Merrie Bergmann. An introduction to many-valued and fuzzy logic: semantics, algebras, and derivation systems. Cambridge University Press, 2008.

[45] Ivan Donadello, Luciano Serafini, and Artur D'Avila Garcez. Logic tensor networks for semantic image interpretation. In 26th International Joint Conference on Artificial Intelligence, 2017.

[46] Mingsheng Ying. Implication operators in fuzzy logic. IEEE Trans. Fuzzy Syst., 10, 2002.

[47] Emile van Krieken, Erman Acar, and Frank van Harmelen. Analyzing differentiable fuzzy logic operators. Artificial Intelligence, 302:103602, 2022.

[48] Tomasa Calvo, Anna Kolesárová, Magda Komorníková, and Radko Mesiar. Aggregation operators: properties, classes and construction methods. Aggregation operators: new trends and applications, pages 3–104, 2002.

[49] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. Journal of Machine Learning Research, (11), 2008.

[50] Murray Shanahan, Kyriacos Nikiforou, Antonia Creswell, Christos Kaplanis, David Barrett, and Marta Garnelo. An explicitly relational neural network architecture. In Proceedings of the 37th International Conference on Machine Learning, volume 119. PMLR, 2020.

[51] Luís C. Lamb, Artur d'Avila Garcez, Marco Gori, Marcelo O.R. Prates, Pedro H.C. Avelar, and Moshe Y. Vardi. Graph neural networks meet neural-symbolic computing: a survey and perspective. In Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, 2020.

[52] Kexin Yi, Jiajun Wu, Chuang GAN, Antonio Torralba, Pushmeet Kohli, and Joshua B Tenenbaum. Neural-symbolic VQA: disentangling reasoning from vision and language understanding. In 32nd International Conference on Neural Information Processing Systems, 2018.

[53] A. Garcez, M. Gori, L. Lamb, L. Serafini, Michael Spranger, and S. Tran. Neural-symbolic computing: an effective methodology for principled integration of machine learning and reasoning. FLAP, 6, 2019.

[54] Yuke Zhu, Alireza Fathi, and Li Fei-Fei. Reasoning about object affordances in a knowledge base representation. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, Proceedings of the European conference on computer vision – ECCV 2014, 2014.

[55] Cewu Lu, Ranjay Krishna, Michael Bernstein, and Li Fei-Fei. Visual relationship detection with language priors. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, Proceedings of the European conference on computer vision – ECCV 2016, Cham, 2016.

[56] Kenneth Marino, Ruslan Salakhutdinov, and Abhinav Gupta. The more you know: using knowledge graphs for image classification. In Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition, 2017.

[57] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In Proceedings of the IEEE international conference on computer vision, 2017.

[58] Saining Xie and Zhuowen Tu. Holistically-nested edge detection. In Proceedings of the IEEE international conference on computer vision, 2015.

[59] Yin Cui, Menglin Jia, Tsung-Yi Lin, Yuting Song, and Serge Belongie. Class-balanced loss based on effective number of samples. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019.

[60] Wenjia Xu, Yongqin Xian, Jiuniu Wang, Bernt Schiele, and Zeynep Akata. Attribute prototype network for zero-shot learning. Advances in Neural Information Processing Systems, 33, 2020.

[61] De Cheng, Gerong Wang, Nannan Wang, Dingwen Zhang, Qiang Zhang, and Xinbo Gao. Discriminative and robust attribute alignment for zero-shot learning. IEEE Transactions on Circuits and Systems for Video Technology, 2023.

[62] Li Zhang, Tao Xiang, and Shaogang Gong. Learning a deep embedding model for zero-shot learning. In Proceedings of the IEEE conference on computer vision and pattern recognition, 2017.

[63] Pengkai Zhu, Hanxiao Wang, and Venkatesh Saligrama. Generalized zero-shot recognition based on visually semantic embedding. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2019.

[64] Long Chen, Hanwang Zhang, Jun Xiao, Wei Liu, and Shih-Fu Chang. Zero-shot visual recognition using semantics-preserving adversarial embedding networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1043–1052, 2018.

[65] Lei Zhang, Peng Wang, Lingqiao Liu, Chunhua Shen, Wei Wei, Yanning Zhang, and Anton Van Den Hengel. Towards effective deep embedding for zero-shot learning. IEEE Transactions on Circuits and Systems for Video Technology, 30(9), 2020.

[66] Huajie Jiang, Ruiping Wang, Shiguang Shan, and Xilin Chen. Transferable contrastive network for generalized zero-shot learning. In Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019.

[67] Yun Li, Zhe Liu, Lina Yao, Xianzhi Wang, and Can Wang. Attribute-modulated generative meta learning for zero-shot classification. ArXiv, 2021.

[68] Guo-Sen Xie, Li Liu, Xiaobo Jin, Fan Zhu, Zheng Zhang, Jie Qin, Yazhou Yao, and Ling Shao. Attentive region embedding network for zero-shot learning. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2019.

[69] Yu Du, Miaojing Shi, Fangyun Wei, and Guoqi Li. Boosting zero-shot learning via contrastive optimization of attribute representations. IEEE Transactions on Neural Networks and Learning Systems, 2023.

[70] Yang Liu, Lei Zhou, Xiao Bai, Yifei Huang, Lin Gu, Jun Zhou, and Tatsuya Harada. Goal-oriented gaze estimation for zero-shot learning. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2021.

[71] Yunlong Yu, Zhong Ji, Jungong Han, and Zhongfei Zhang. Episode-based prototype generating network for zero-shot learning. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2020.

[72] Zhe Liu, Yun Li, Lina Yao, Xianzhi Wang, and Guodong Long. Task aligned generative meta-learning for zero-shot learning. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 35, 2021.

[73] Jingjing Li, Mengmeng Jing, Ke Lu, Zhengming Ding, Lei Zhu, and Zi Huang. Leveraging the invariant side of generative zero-shot learning. 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019.

[74] Rafael Felix, Ian Reid, Gustavo Carneiro, et al. Multi-modal cycle-consistent generalized zero-shot learning. In Proceedings of the European conference on computer vision (ECCV), 2018.

[75] Zongyan Han, Zhenyong Fu, Shuo Chen, and Jian Yang. Contrastive embedding for generalized zero-shot learning. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2021.

[76] A Vaswani. Attention is all you need. Advances in Neural Information Processing Systems, 2017.

[77] José Maurício, Inês Domingues, and Jorge Bernardino. Comparing vision transformers and convolutional neural networks for image classification: A literature review. Applied Sciences, 13(9), 2023.

[78] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In Proceedings of the 2017 IEEE International Conference on Computer Vision, 2017.

[79] Xianjie Chen, Roozbeh Mottaghi, Xiaobai Liu, Sanja Fidler, Raquel Urtasun, and Alan Yuille. Detect what you can: detecting and representing objects using holistic models and body parts. In Proceedings of the IEEE conference on computer vision and pattern recognition, 2014.

[80] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John M. Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. International Journal of Computer Vision, 88:303–338, 2010.

[81] João Cartucho, Rodrigo Ventura, and Manuela Veloso. Robust object recognition through symbiotic deep learning in mobile robots. In Proceedings of the 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018.

[82] Luciano Serafini, A.S. d'Avila Garcez, Samy Badreddine, Ivan Donadello, Michael Spranger, and Federico Bianchi. Logic tensor networks: theory and applications. In Proceedings of the Neuro-Symbolic Artificial Intelligence, 2021.

[83] E. G. Miller, N. E. Matsakis, and P. A. Viola. Learning from one example through shared densities on transforms. In IEEE Conference on Computer Vision and Pattern Recognition, 2000.

[84] B. Lake, R. Salakhutdinov, J. Gross, and J. Tenenbaum. One shot learning of simple visual concepts. Cognitive Science, 2011.

[85] Gregory Koch, Richard Zemel, Ruslan Salakhutdinov, et al. Siamese neural networks for one-shot image recognition. In ICML deep learning workshop. Lille, 2015.

[86] Catherine Wah, Steve Branson, Pietro Perona, and Serge Belongie. Multiclass recognition and part localization with humans in the loop. In 2011 International Conference on Computer Vision, 2011.

[87] Ali Farhadi, Ian Endres, Derek Hoiem, and David Forsyth. Describing objects by their attributes. In IEEE Conference on Computer Vision and Pattern Recognition, 2009.

[88] Jianxiong Xiao, James Hays, Krista A Ehinger, Aude Oliva, and Antonio Torralba. SUN database: Large-scale scene recognition from abbey to zoo. In 2010 IEEE computer society conference on computer vision and pattern recognition. IEEE, 2010.

[89] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In IEEE Conference on Computer Vision and Pattern Recognition, 2016.

[90] 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, 2009.

[91] Samy Badreddine, Artur Garcez, Luciano Serafini, and Michael Spranger. gts: logic tensor network library. https://github.com/logictensornetworks/logictensornetworks, 2021.

[92] Soravit Changpinyo, Wei-Lun Chao, Boqing Gong, and Fei Sha. Synthesized classifiers for zero-shot learning. In 2016 IEEE Conference on Computer Vision and Pattern Recognition, 2016.

[93] F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. S. Torr, and T. M. Hospedales. Learning to compare: Relation network for few-shot learning. In IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2018.

[94] Meng Ye and Yuhong Guo. Progressive ensemble networks for zero-shot recognition. In IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019.

[95] Chaofan Chen, Oscar Li, Daniel Tao, Alina Barnett, Cynthia Rudin, and Jonathan K Su. This looks like that: Deep learning for interpretable image recognition. Advances in Neural Information Processing Systems, 2019.

[96] Shiming Chen, Ziming Hong, Guosen Xie, Wenhan Wang, Qinmu Peng, Kai Wang, et al. MSDN: Mutually semantic distillation network for zero-shot learning. 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2022.

[97] Wei Wang and Qingzhong Li. Generalized zero-shot activity recognition with embedding-based method. ACM Transactions on Sensor Networks, 19(3), 2023.

[98] Karan Sikka, Jihua Huang, Andrew Silberfarb, Prateeth Nayak, Luke Rohrer, Pritish Sahu, John Byrnes, Ajay Divakaran, and Richard Rohwer. Zero-shot learning with knowledge enhanced visual semantic embeddings. arXiv preprint arXiv:2011.10889, 2020.

[99] Tommaso Carraro. LTNtorch: PyTorch implementation of Logic Tensor Networks. https://doi.org/10.5281/zenodo.6394282, mar 2022.

[100] Hongzu Su, Jingjing Li, Ke Lu, Lei Zhu, and Heng Tao Shen. Dual-aligned feature confusion alleviation for generalized zero-shot learning. IEEE Transactions on Circuits and Systems for Video Technology, 33(8), 2023.

[101] Samy Badreddine, Luciano Serafini, and Michael Spranger. logLTN: Differentiable fuzzy logic in the logarithm space. ArXiv, abs/2306.14546, 2023.