POLITECNICO DI TORINO

SEDE DI TORINO

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea Magistrale

End-to-end training of Logic Tensor Networks for object detection

Relatori
prof. Fabrizio Lamberti
prof. Lia Morra

Studente
Filomeno Davide Miro
matricola: 256870

April 2021

Abstract

Neural-symbolic computing aims at integrating two fundamental paradigms in artificial intelligence: machine learning (that is, the ability to learn from examples) and symbolic knowledge representation and reasoning (that is, the ability to reason from what has been learned). The objective of this thesis is the development of a novel neural-symbolic architecture for object detection in natural images. Specifically, this architecture is based on a neuro-symbolic model called Logic Tensor Networks (LTNs). The base concept behind LTNs is the grounding of a first order logic (FOL) that allows to represent symbolic knowledge as operations between tensors in a neural network; LTNs can be trained through stochastic gradient descent to maximize the satisfiability of the FOL. Not only are LTNs able to understand logical relationships between two or more objects inside the image, they also allow to encode prior knowledge that can improve the performance in the presence of scarce or noisy datasets. LTNs have been applied before to computer vision tasks. Usually, the image is processed by a pre-trained Convolutional Neural Network (CNN) or object detector to extract semantic high-level features, which are then passed to the LTN for further refinement. To the best of our knowledge, none of the existing works addresses the issues of how to jointly train the LTN and the CNN. This thesis seeks to address this gap by designing and implementing a neuro-symbolic object detector which combines a two-stage object detector (Faster R-CNN) with the LTN. This involved a radical change in the classification process. The multi-class classifier head is substituted by a set of binary predicates, grounded in an LTN module, each predicting the presence of a specific class. This results in a high data imbalance as each training batch contains at most a few positive examples. To overcome this drawback a new function is introduced for the aggregation of logical terms, denoted Focal Log Sum, which is inspired by the Focal loss. Faster-LTN achieves a mAP of 0.72 and 0.71 on the PASCAL VOC and Pascal Part datasets, which is competitive with the results achieved by Faster R-CNN. Extensive experiments were conducted to establish the impact of different aggregation functions, training strategies, and losses. Finally, experiments are conducted to highlight how to integrate Faster-LTN with prior knowledge, in the form of logical predicates, and the impact on object recognition accuracy.

Acknowledgements

A mio padre e mia madre che mi hanno sempre sostenuto in questi anni, in ogni modo possibile.

A mia sorella Nicoletta che è e sarà sempre la mia più fedele consigliera.

Ai miei nonni paterni Filomeno, di cui porto fieramente il nome, e Nicoletta che mi proteggono ogni giorno da lassù.

Ai miei nonni materni Giuseppe e Leonarda che mi hanno sempre incoraggiato.

A tutta la mia famiglia.

A tutti i miei amici, con cui ho condiviso ansie e preoccupazioni ma anche gioia e soddisfazioni.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Artificial Intelligence has accomplished extraordinary progress in the last few years. In addititions to being object of interest for Reseach and Industry, with thousands of papers published and millions of dollars invested each years, it has also become a topic of conversation for ordinary people,fascinated but at the same time worried about its developments. Deep Learning [19] is the sub-field of AI that more than other has contributed to the success of the entire sector. Deep Learning applications in the fields of computer vision, natural language processing, machine translation, speech recognition and image understanding have revolutionized the state of the art, allowing to obtain results impossible to achieve with the traditional methods of computer science. However this powerful tool is characterized by some disadvantages that limit the possibilities of use. Deep neural networks, in order to learn the necessary knowledge for their tasks, they need to observe a lot of labelled data. In many cases, especially when dealing with a specific domain problem, getting enough annotated data is problematic. Furthermore the training process of a deep learning model takes a lot of time and computational resources. Another limitation of deep neural networks is given by their complexity, which makes it impossible for a human being to understand their decision process. This aspect limits the applications based on Deep Learning in fields , like Medicine, where it is essential to have a transparent decision process in predictions.

Symbolic AI One possible solution to many of these problems comes from Symbolic AI, an approach to Artificial Intelligence that has been popular in decades past. If in the Deep Learning (and in general in Machine Learning) the model is able to retrieve knowledge that it needs from data, in Symbolic AI knowledge is entered by the programmer into the model in the form of logical statements. The training process does not requires large computational resources, long execution times and the use of large amounts of data, but it needs human effort. The decision process of Symbolic AI models is understandable by humans because it consists on reasoning over available knowledge expressed in high-level concepts.

Integrate Symbolic AI in Deep Neural Networks The idea is to intergate in a deep neural model elements get inspired by Symbolic AI in order to have the advandages of the two paradigms. This new field of research is called **Neurosymbolic AI**. The importance of this new kind of approaches is confirmed by some of the main AI researchers. In [11] some speeches held in scientific conferences are mentioned. The AAAI-2020 fireside conversation with Economics Nobel Laureate Daniel Kahneman, including the 2018 Turing Award winners and DL pioneers Geoffrey Hinton, Yoshua Bengio and Yann LeCun ,and the 2019 Montreal AI Debate between Yoshua Bengio and Gary Marcus, have pointed to new perspectives and concerns on the future of AI. It has been argued eloquently that if the aim is to build an AI that is semantically sound, explainable and trustworthy,it needs to include a sound reasoning layer in combination with deep learning.

This work wants to make a contribution to neurosymbolic integration in the field of computer vision. More precisely, we want to introduce a new model for object detection resulting from the interaction of Faster RCNN [34] with a neural-symbolic paradigm called Logic Tensor Networks (LTNs) [38]. This new model is called **Faster-LTN**. The object detection task has the purpose of identifying the instances of a class of objects in an image. It is a very important task because it is the basis of other specific tasks such as object segmentation, image annotations, face detection and recognition, video segmentation. Furthermore, there are many practical applications based on object detection, for example those related to self-driving cars,robotics and manufacturing industry [39].

LTNs provide a complete grounding for a First Order Logic language. First-order logic statements are therefore mapped onto differentiable real-valued constraints using a many-valued logic interpretation in the interval [0,1].In the literature there are several applications of LTNs. The results of this work start from the LTNs applications in object detection presented in [8].

Experiments and results Experiments were conducted with PASCAL VOC [9] and PASCAL-Part datasets. Faster-LTN has demostated to be competitive with Faster R-CNN. The new model is able to achieve better performance than the Faster RCNN, especially using external symbolic knowledge as a constraint of training. Experiments with PASCAL Parts dataset has been demostrated that performances go from a mean Average Precision(mAP) score of 0.71 for Faster R-CNN to a mAP of 0.73 for Faster-LTN with embedded symbolic knowledge. After this brief introduction of the work we are going to introduce the content of the chapters into which the thesis has been divided.

Chapter 2 provides an overview of the main works related to Neurosymbolic AI. It is more focused on explaining the main concepts and introducing the main applications in the Computer Vision task related to Neurosymbolic AI. Chaper 3 explains the implementation details of the LTNs. It also introduces a new implementation of LTNs using the keras library. Chapter 4 presents the integration

Faster RCNN-LTN and finally Chapter 5 introduces all the experiments in which are implemented logical constrains in the Faster RCNN-LTNs model.

Chapter 2

State of the Art

2.1 Introduction

This chapter illustrates some of the main works concerning neurosymbolic AI by introducing some theoretical aspects which recur in subsequent chapters.

In Section 2.2 we explain what Symbolic AI and Connectionist AI are, what are their strengths and weaknesses and why the former has given way to the latter as the main research paradigm in the field of artificial intelligence.In Section 2.3 is explained the neurosymbolic paradigm.

Starting from some works in the literature, the main concepts related to neurosymbolic AI are illustrated.

In Subsection 2.3.1 some common architectural patterns in neurosymbolic models are visually represented. An example of how Boolean logic can be implemented in a neural network, a fundamental aspect for neurosymbolic AI, is presented in Subsection 2.3.2. In Subsection 2.4 some concepts of fuzzy logic are defined which are fundamental for subsequent developments.

In Section 2.5 the LTN is presented, a neurosymbolic model which is the starting point for this thesis work.

Finally, Section 2.6 defines the state of the art of neurosymbolic models applied to the various tasks of computer vision.

2.2 Symbolic AI vs Connectionist AI

Symbolic artificial intelligence also known as Good, Old-Fashioned AI was the predominant approach for artificial intelligence until 1980s.

This paradigm is based on the interpretation and manipulation of symbols that are high-level representations of objects or entities. These representations can be manipulated by computers. The combination of symbols that express interrelation

is called reasoning. An example of system that implements symbolic AI is expert system that is a network of rules coded as if-Then statement.

The main advantages of this paradigm is the transparency of the decision-making process of the model and the fact that it does not need huge amount of data to acquire knowledge.

This kind of approach has been left in favor of data driven approaches like deep learning because they performs better in complex task like computer vision and natural language processing in which is difficult or impossible to encode the necessary knowledge in a system based on symbols and rules to treat them.

The successful aspects of the so called connectionist AI are that it is able to learn rules given the data and learn a features representation of the raw input data suitable for the given task.

Despite the huge success of deep learning and connectionist AI at the expense of Symbolic AI these approaches have some issues related to the huge amount of data and computation resources that they need to learn rules and the interpretability of models for humans.

2.3 Neurosymbolic AI

Neurosymbolic computing is a recent field in artificial intelligence that study approaches that combine logic reasoning with neural network.

It is important to understand how researchers have thought to combine these two paradigms.

The work [6] provide a good explaination of key concepts regarding neuro symbolic systems.

Grounding In some works logic knowledge is embedded in a neural system and a way to do that is by a **grounding**. A grounding is a step in which logic knowledge is encoded in a tensor space in order to have a representation manageable by neural networks.A straightforward example of grounding is in [8]. In this case grounding is learned from examples by training a neural network in order to optimize the truth-values of the formulas in the background knowledge .

Fuzzy Logic The determinism that characterize logic may seem irreconcilable with the randomness of neural networks. A possible solution of this issue is **fuzzy logic**, a new kind of logic in which truth values can be in the continuous intervall between 0 and 1. This logic defines its logical operation AND,OR and NOT.

In [8] fuzzy logic is used by grounding to translate boolean predicates. Another way to conciliate boolean logic with neural network is given by neural logic network [37] which represent a first simple way of combining neural networks with Boolean logic.

Another interesting NeSy model is [29]. This system, instead of integrating reasoning capabilities into a neural network, integrates some neural componets in the probabilistic programming language ProLog [5]. This probabilistic language coinsist of a set of ground probabilistic facts $\mathcal{F}$,each associated with a probability p and a set of rules. The probabilities of each fact are independendent from each others. Given a query q it is possible to compute the probability $P(q)$ by considering the given facts and rules.The neural networks are intregrated as neural annotated disjunction (nAD). An annotated disjuntion is an expression of the form : $p_1 :: h_1; ...; p_n :: h_n : -b_1, ...b_m$ where p_i are probabilities and h_i and b_i are atoms. Whenever all b_i hold, h_j will be true with probability p_j. A nAD take the form: $nn(m_q, t, u) :: q(t, u_1); ...; q(t, u_n) : b_1, ..., b_m$, where b_i are the atoms, t is the input of the neural network and $u_i...u_n$ are the possible outputs of the neural network.The deepProbLog program , given a set of pairs (p, q) $\mathcal{Q}$,will be able to output the desired success probability p for the query q.

Another similar work is [41] propose an architecture that integrate neural networks with an answer set program given by a symbolic reasoning engine like CLINGO.

Subsymbols We have seen how a predicate of symbols can be represent by a tensor by using a groungding, but how represent symbols itself as real value tensors? In [8] symbols are substitute by **sub-symbols** that in this case are vectors of features extracted by an object detector.As explained in the cited paper,"reasoning about equality corresponds closely to reasoning about similarity in embedding space." Logical predicates explain relationships between symbols and the whole form knowledge.

Training end to end neuro-symbolic elements A challenge of neurosymbolic AI is to train the neuro elements of the model end to end with symbolic ones and the recent work [4] provides an interesting example of how a neural network can be linked to a model called Knowledge Enhancer that inject knowledge into models for multi-label classification. Starting from the predictions $\mathbf{y}$ this module modifies the classifier predictions in order to increase the satisfaction of a set of clauses $\mathcal{K}$.The semantic of clauses is given by using the concept of t-conorm.

The objective of KE is to maximize the t-conorm of each clause.This model proposes the concept of t-conorm boost function that is a fuction $\delta : [0,1]^n \rightarrow [0,1]^n$ that proposes the changes to be applied on the inputs to increase the value of t - conorm applied on them : $\perp (\mathbf{t} + \delta(\mathbf{t})) \geq \perp (\mathbf{t})$. Another interesting feature of this model is that each clause is weighted by a parameter w_c that is learned by the model.

2.3.1 Neurosymbolic architectures

After this brief explanation of some of the main concepts of neuro-symbolic artificial intelligence,it is important to understand how this idea was actually implemented in working models. The work [40] offers an overview of the main architectural templates of neural networks combined with symbolic elements.

Learning with symbolic input and output In this pattern a neural network accepts as input a symbolic structure and output another symbolic structure.

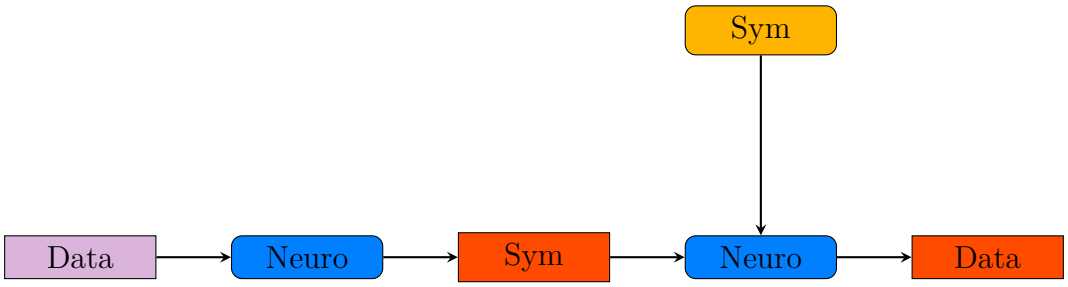
Learning on data with symbolic output In the template below a neural network accepts as input some data (images,vectors of numers) and has a symbolic output.

Ontology learning Starting from data (e.g text)this kind of module are able to extract ontology useful for subequent reasoning.

Perceptual abstraction It uses data to extract symbols useful for the training of the final neural network.

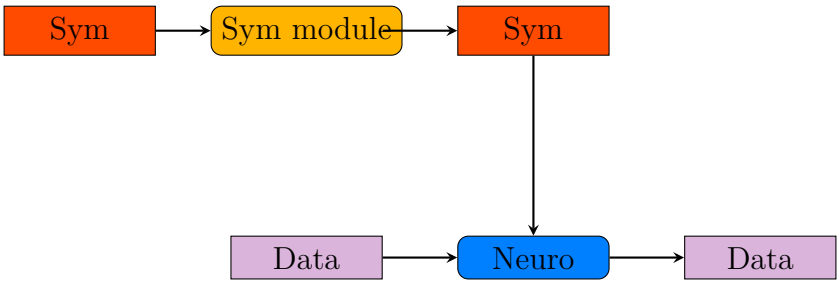
Explaination The system output retrospective symbolic explanations for predictions.

Abstraction The first neural stage refines the input data with the help of a symbolic ontology and the abstracted data are then feed to a classifier.

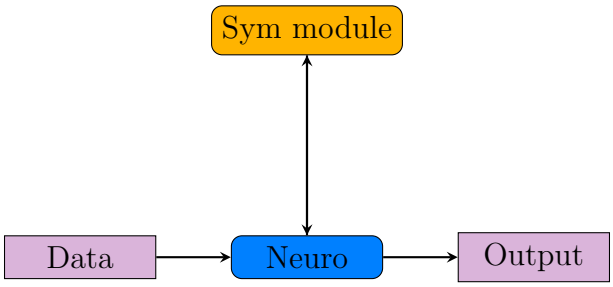


Learning with symbolic information as a prior The knowledge encoded in symbols is used as a prior for neural network learning. An example of this kind of neuro-symbolic integration is [8] that uses symbolic knowledge as a constraint in training the neural network.

Also the architecture of Faster-LTN can be traced back to this kind of integration. Neural component is given by Faster-RCNN and Symbolic module is given by the part of the network inspired by LTNs.



Meta-reasoning over learning systems In this pattern the symbolic module is used to control the behaviour of a learning agent. It should be able to set hyperparamiters of the neural module.



Merged systems Learning system in which symbolic and neural parts are closey interconnected.An exaple is [36]

In the next paragraphs are proposed insights about two models mentioned above: **Logic Tensor Networks** and **Neural Logic Network**.

2.3.2 Neural Logic Network

The idea of [37] is to approximate the behavior of an elementary logical operator like AND,OR and NOT with a simple feed forward neural network that is fitted to approximate logic operations . Input variables (True or False) are learned vector representation and logical predicates are rendered combining boolean variables with vector representations and operators with trained modules. The output of the last module is compared to the T vector using a similarity module that produce a probability score about how close to the true is the result.The AND and OR networks are feed forward neural network with one hidden layer:

$$AND(w_i, w_j) = \mathbf{H}_{a2f}(\mathbf{H}_{a1f}(w_i|w_j) + b_a) \tag{2.1}$$

where $\mathbf{H}_{a1f} \in \mathbb{R}^{d\times 2d}$,$\mathbf{H}_{a2f} \in \mathbb{R}^{d\times d}$,$b_a \in \mathbb{R}^d$. but obviously with only one vector as input. The similarity module is based on cosine similarity between two vectors w_i and w_j :

$$Sim(w_i, w_j) = \sigma\left(\alpha \frac{w_i, w_j}{\|w_i\|\|w_j\|}\right) \tag{2.2}$$

The resulting loss function:

$$L = L_c + \lambda_l \sum_i r_i + \lambda_l \sum_{w\in\mathbf{W}} \|\mathbf{w^2}\|_{\mathbf{F}} \tag{2.3}$$

in which the first term L_c is a cross-entropy loss function,the $\lambda_l \sum_i r_i$ is sum of all logic regularizer, predefined terms added to the loss to ensure that solutions that respect the logical properties of the operators are selected in the training. The last term is a regularizer to avoid overfitting. The training data are some boolean variable v_i randomly generated and combined in logical preidicates.

2.4 Fuzzy Logic

In this section some concepts related to fuzzy logics are explained, such as t-norms, t-conorms and aggregation functions, which are frequently used in neurosymbolic models such as LTNs. Further details can be found in [16].

2.4.1 Fuzzy Negation

A **Fuzzy Negation** is a function $N : [0,1] \to [0,1]$ so that $N(0) = 1$ and $N(1) = 0$.It is the function used to compute the negation of a truth value. N is **strict** if it is strictly decreasing and continuous, and **strong** if for all $a \in [0,1], N(N(a)) = a$. The strict and strong classic negation is:

$$N_c(a) = 1 - a \tag{2.4}$$

2.4.2 Triangular Norms

A $\mathsf{t-norm}$ (Triangular Norm) is a function $T : [0,1]^2 \to [0,1]$ that is commutative and associative. Other properties that it has are:

1. **Monotonicity**.$\forall a \in [0,1], T(a,.)$ *is increasing.*
2. **Neutrality**.$\forall a \in [0,1], T(1,a) = a$.

It defines the semantics of the conjunction or AND in Boolean logic. In 2.1 are illustated the main t-norms.

Table 2.1. List of main triangular norms.

Name	T-Norm
Godel	$T_G(a,b) = min(a,b)$
Product	$T_P = ab$
Lukasiewicz	$T_{LK}(a,b) = max(a+b-1,0)$
Drastic product	$T_D(a,b) = \begin{cases} \min(a,b), & if\ a == 1\ or\ b == 1 \\ 0, & \text{otherwise} \end{cases}$
Nilpotent minimum	$T_{nM}(a,b) = \begin{cases} 0 & \text{if a + b} \leq 1 \\ \min(a,b) & otherwise \end{cases}$
Yager	$T_Y(a,b) = max\left(1 - ((1-a)^p + (1-b)^p)^{\frac{1}{p}}, 0\right), p \geq 1$

2.4.3 Triangular Conorms

A $\mathsf{t-conorm}$ (Triangular Conorm) is a function $S : [0,1]^2 \to [0,1]$ that is commutative and associative. Other properties that it has are:

1. **Monotonicity**. $\forall a \in [0,1], S(a,.)$ *is increasing.*
2. **Neutrality**. $\forall a \in [0,1], S(0,a) = a$.

It defines the semantics of the disjunction or OR in Boolean logic. T- conorms are obtained from t-norms by appling the **De Morgan's laws** :

$$p \vee q = \neg(\neg p \wedge \neg q) \tag{2.5}$$

Given T a t-norm and N_C the classical negation,T's N_C-dual S is obtained using:

$$S(a,b) = 1 - T(1-a, 1-b) \tag{2.6}$$

In 2.2 are illustated the main t-conorms.

Table 2.2. List of main triangular conorms.

Name	T-Conorm
Godel	$S_G(a,b) = max(a,b)$
Product	$S_P = a + b - ab$
Lukasiewicz	$S_{LK}(a,b) = min(a+b,1)$
Drastic sum	$S_D(a,b) = \begin{cases} \max(a,b), & if\ a == 0\ or\ b == 0 \\ 0, & \text{otherwise} \end{cases}$
Nilpotent minimum	$S_{nM}(a,b) = \begin{cases} 1 & \text{if a + b} \geq 1 \\ \max(a,b) & otherwise \end{cases}$
Yager	$S_Y(a,b) = min\left((a^p + b^p)^{\frac{1}{p}}, 1\right), p \geq 1$

2.4.4 Aggregation operators

An **aggregation operator** is a function $A : [0,1]^n \rightarrow [0,1]$ that is symmetric and increasing with respect to each argument, and for which $A(0,...,0) = 0, A(1,...,1) = 1$. The aggregator is used to compute quantifiers like $\forall$ and $\exists$. Table 2.3 illustrates the list of main aggregation functions used in past works.

Table 2.3. List of main aggregation operators.

Name	Aggregation operator
Minimum	$A_m(x_1,...,x_n) = min(x_1,...,x_n)$
Product	$A_P(x_1,...,x_n) = \prod_{i=1}^{n} x_i$
Maximum	$A_M(x_1,...,x_n) = max(x_1,...,x_n)$
Mean	$A_{mean}(x_1,...,x_n) = \left(\frac{1}{n}\sum_{i=1}^{n} x_i^p\right)^{\frac{1}{p}}$ [1]

[01] For $p = 1,2,-1$ we have Aritmetic,Geometric and Harmonic means

2.5 Logic Tensor Network

LTNs [8] propose a grounding of a First Order logic Language. Given a first-order logic language $\mathcal{L}$,its signature is composed of three sets: a set $\mathcal{C}$ of constants,a set $\mathcal{F}$ of function and a set $\mathcal{P}$ of predicates. This model define a n-grounding , $n \in \mathbb{N}$, $\mathcal{G}$ for a FOL $\mathcal{L}$.It is a function defined on the signature of L satisfying the following conditions:

$$1.\mathcal{G}(c) \in \mathbb{R}^n \textit{ for every constant symbol } c \in \mathcal{C} \tag{2.7}$$
$$2.\mathcal{G}(f) \in \mathbb{R}^{n \cdot \alpha(f)} \to \mathbb{R}^n \textit{ for every function } f \in \mathcal{F} \tag{2.8}$$
$$3.\mathcal{G}(P) \in \mathbb{R}^{n \cdot \alpha(P)} \to [0,1] \textit{ for every predicate } p \in \mathcal{P} \tag{2.9}$$

The semantics of closed terms and atomic formulas is defined as follow:

$$\mathcal{G}(f(t_1, ..., t_m)) = \mathcal{G}(f)(\mathcal{G}(t_1), ..., \mathcal{G}(t_m)) \tag{2.10}$$
$$\mathcal{G}(\mathcal{P}(t_1, ...t_m)) = \mathcal{G}(P)(\mathcal{G}(t_1), ..., \mathcal{G}(t_m)) \tag{2.11}$$

The semantic for connectivities is defined according to any fuzzy logic definition (Lukasiewicz, product or Godel). The groundings of main logical operators, using Lukasiewicz logic are:

$$\mathcal{G}(\neg\phi) = 1 - \mathcal{G}(\phi) \tag{2.12}$$
$$\mathcal{G}(\phi \vee \psi) = min(1, \mathcal{G}(\phi) + \mathcal{G}(\psi)) \tag{2.13}$$
$$\mathcal{G}(\phi \wedge \psi) = max(0, \mathcal{G}(\phi) + \mathcal{G}(\psi) - 1) \tag{2.14}$$
$$\mathcal{G}(\phi \to \psi) = min(1,1 - \mathcal{G}(\phi) + \mathcal{G}(\psi)) \tag{2.15}$$
$$\mathcal{G}(\forall x \phi(x)) = \lim_{T \to term(\mathcal{L})} mean_p(\mathcal{G}(\phi(x)) | t \in T) \tag{2.16}$$

Combining the explained rules, the model is able to represent numerically a symbolic predicate like $\forall \mathsf{Dog}(x) \to \exists y((\mathsf{partOf}(x, y) \vee \mathsf{Tail}(y)))$. Considering $\mathcal{K}$, the set of all logical clauses $cl \in \mathcal{K}$ that define the knowledge base, the learning strategy is to find a grounding $\mathcal{G}^*$ such as to maximize the truth values of the clauses cl. Learnable groundings for both functions and predicates are defined.

The grounding of a m-ary function symbol is:

$$\mathcal{G}(f)(v) = M_f \mathbf{v} + N_f \tag{2.17}$$

where $\mathbf{v} = \left\langle \mathbf{v}_1^T, ..., \mathbf{v}_m^T \right\rangle^T$ is the mn-ary vector containing each $\mathbf{v}_i$. $M_f \in \mathbb{R}^{n \times mn}$ and $N_f \in \mathbb{R}^n$ are tensors of parameters that must be fitted.

The grounding of a predicate takes the form of:

$$\mathcal{G}(\mathcal{P})(\mathbf{v}) = \sigma\left(u_P^T \tanh\left(\mathbf{v_T} W_P^{[1:k]} \mathbf{v} + V_P \mathbf{v} + b_p\right)\right) \tag{2.18}$$

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.

After defining the groundings of each element of $\mathcal{L}$'s signature, we define the learning problem posed by the LTNs.

A partial grounding $\hat{\mathcal{G}}$ is a grounding that is defined on a subset of the signature of $\mathcal{L}$.

A grouded theory GT is a pair $\left\langle \mathcal{K}, \hat{\mathcal{G}} \right\rangle$ with a set of closed formulas $\mathcal{K}$ and a partial grounding $\hat{\mathcal{G}}$.

A grounding $\mathcal{G}$ is a completion of $\hat{\mathcal{G}}$ if $\mathcal{G}$ is a grounding for $\mathcal{L}$ and coincides with $\hat{\mathcal{G}}$ on the symbols where $\hat{\mathcal{G}}$ is defined.

A grounding G satisfies a GT $\left\langle \mathcal{K}, \hat{\mathcal{G}} \right\rangle$ if $\mathcal{G}$ completes $\hat{\mathcal{G}}$ and $\mathcal{G}(\phi) = 1 \; \forall \; \phi \in \mathcal{K}$.

When a GT is not satisfiable, we are interested in the best possible satisfaction that we can reach with a grounding. This is defined as the best satisfiability problem: given a GT $\left\langle \mathcal{K}, \hat{\mathcal{G}} \right\rangle$ we want to find a $\mathcal{G}^*$ that maximize the truth-value of all clauses $cl \in \mathcal{K}$:

$$\mathcal{G}^* = argmax_{\hat{\mathcal{G}} \subseteq \mathcal{G} \in \mathbb{G}} \mathcal{G}\left(\bigwedge_{cl \in \mathcal{K}} cl\right) \tag{2.19}$$

A GT is defined given the labelled data and the knowledge available and, given them, we want to learn $\mathcal{G}^*$ in order to maximize the satisfability of GT.

For example in [8] is defined a GT for Semantic Image Interpretation (SII). Considering the signature of $\sigma_{SII} = \langle \mathcal{C}, \mathcal{F}, \mathcal{P} \rangle$, $\mathcal{C} = \cup_{p \in Pics} b(p)$ is the set of all bouding boxes in all the images, $\mathcal{F} = \emptyset$ and $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2\}$ where $\mathcal{P}_1 = \{\mathsf{Dog}, \mathsf{Cat}, \mathsf{Person}, ...\}$ is the set of all unary predicates for classification and $\mathcal{P}_2 = \{\mathsf{partOf}\}$ is the set of all binary predicates,in this case only partOf.

Training set for LTNs can be defined as $\mathcal{T} = \left\langle \mathcal{K}, \hat{\mathcal{G}} \right\rangle$. $\mathcal{K}$ contains a set of literals $C_i(b)$ or $\neg C_i(b)$ for every bounding box labelled or not labelled with class C_i and $\mathsf{partOf}(\mathsf{b}, \mathsf{b}')$ or $\neg\mathsf{partOf}(\mathsf{b}, \mathsf{b}')$ for every pair $\langle b, b' \rangle$ for which the prdicate partOf is true or false. This is common in machine learning where classifiers (i.e. the grounding of predicates) are inductively learned from positive examples (such as $\mathsf{Cat}(b, b)$) and negative examples ($\neg\mathsf{Cat}(b, b)$).

The novelty is given by the fact that we can define logical formulas capable of encoding external knowledge in addition to that provided by class labels. For example, we can define more complex formulas like $\forall \mathsf{Dog}(x) \rightarrow \exists y((\mathsf{partOf}(x, y) \vee \mathsf{Tail}(y)))$ if we want to establish that "if y is part of x and x is a Dog then y is a Tail ". Loss function must be minimized in order to maximize the truth value of GT given $\mathcal{G}$.

2.6 Neuro symbolic AI and Computer Vision

The impressive ability of neural networks to find patterns in data by extracting suitable features from data alone has made them attractive in computer vision, especially with the invention of convolutional neural networks. CNNs propose solutions to the problems of image classification, object detection and segmentation that overcome traditional approaches. Moreover, in these tasks it can be useful to exploit knowledge and symbolic reasoning to have more accurate predictions, less computational effort, and in cases where there is a lack of labeled data. This paragraph is an overview of main computer vision works that get inspired by this idea.

2.6.1 Object detection and image classification

The paper [22] proposes a layer, to be inserted in a CNN architecture that uses symbolic knowledge encoded in a graph, to influence feature extraction. This layer learns a representation that combines low-level knowledge extracted by the previous convolution with high-level knowledge coming from nodes and edges of a symbolic graph. This representation is added to the input feature map and output tensor is forwarded to a next convolutional layer.

The work [30] it is another work that takes advantage of symbolic knowledge encoded in a graph. This model, given the detections of a Faster R-CNN, selects some nodes of the symbolic graph. A function, to be learned for each node evaluates its importance for the final classification. At each step the nodes close to the previous ones are evaluated.

It is possible to use LTNs [8] to improve the performance of the classification of an object detector.Class membership is a unary predicate of a FOL and, as explained before, LTNs define a grounding for a predicate of a FOL.

2.6.2 Visual Relationship Detection

The task of visual relationship detection aims to find into the image attributes (<subject, attribute >) and relationship between objects (<subject, predicate, object >). For example if an image depicts a man riding a white horse,a VRD system should be able to detect <horse, white >and <man, ride, horse >. The datasets that are most used in this task are [17] and [27].

The work [27] it's one of the most important works about this task. It gives an idea of how relationships between concepts can be exploited for VRD.Beside a visual appearance module that extract visual features from images, a language module projects annotated relationships in an embedding space in which semantically similar relationships are optimized to be close together. For example, it can be exploited the fact that the relationship <man,on,horse >is similar to <man,ride,horse

>. This paper also illustrates some challenges for VRD problem. The fact that make visual reasoning tasks so challenging is the lack of examples compared to huge amount of possible relationships:for n entities the number of possible predicates between them tend to $n \times n$. It might be useful to support data-driven learning through the use of symbolic knowledge expressed through logical predicates.

This idea is encouraged by the fact that large knowledge bases are available, such as WordNet [10],FrameNet [2] and ConceptNet [25].

In [43] starting from the bounding boxes of the objects in the image,extracted using an object detector like [35],using a first bidirectional-LSTM each bounding box is labelled taking care of the other objects in the image. This reflects the idea of [18]. Next a second bidirectional-LSTM predicts edges between objects.

The [21] instead exploits deep reinforcement learning to extract attributes and predicates from images. These models exploit the idea of representing the semantic content of an image as a graph and to train them in a supervised way are available some datasets like [17] and [27] in which images are annotated with graph representation,also called scene graph.

Also the Logic Tensor Networks [8] can be used in a VRD context. The work [7] applies LTNs on VRD dataset [27]. A relationship is a binary predicate that accepts two bounding boxes as input and return a score between 0 and 1 that measure how probable

2.6.3 Visual Question Answering

The task of visual question answering (VQA), starting from an image and a question,the system can output an answer in a natural language.

An example of how symbolic reasoning will be exploited for this task is [15]. This work is interesting because it introduces a concepts vocabulary, a set of semantic embedding concepts that describes objects,attributes and relations.This set of embeddings is used both in scene graph building and in the extraction of reasoning instructions from question expressed in natural language.

In [28] the interesting point is that starting from an image and a related question ,a fact detection module is able to generate a set of relative relations of the form ⟨subject, predicate, object⟩ and the semantic attention module selects relevant relations for the answere.

In [20] is proposed a visual question answering module in which external knowledge, coming from ConceptNet [25], is represented like a graph in which each node is an entity (person,horse,ecc) and each edge represents a predicate (ride,play,ecc). Using a learned module this system is able to query some informations from this big knowledge base.

In [42] we have a model composed of three main modules. A scene parser receives as input an image and give as output a structural scene representation. This structural scene representation collects some informations about the objects

detected in the image, for example shape, color and spatial information.A question parser, given a question in natural language,outputs a program that is executed by a program executor.

Chapter 3

Logic Tensor Networks implementation

3.1 Introduction

In this chapter, we are illustated all the implementative details about Logic Tensor Networks and presents a new Keras implementation of LTNs. The reference code,written by using Tensorflow library, is that relating to [8]. Section 3.2 provide an analysis of the main sections of the code that implements what was theoretically illustrated in 2.5. Before moving on to the Faster-LTN, it was decided to propose a Keras implementation of the LTNs. Section 3.3 explains all details about it. The two implementations are trained and tested and all the results are shown in Section 3.4.

3.2 Original LTNs implementation

The authors of [8] have made available the code and a dataset of bounding boxes previously extracted by Fast R-CNN [12] (executed over PASCAL-Part 2007 [9] dataset). Each data sample is a vector of real numbers containing the 4 coordinates that localize the object in the image and a probability score for each considered class.

Input The class **Domain** define the input for LTNs predicates. For each class are defined a batch of positive examples and a batch of negative examples.The number of samples is the same for all batches and is an hyperparameter of the model (**batch size**). In the work [8] the partOF predicate is also presented. This is a binary predicate, it accepts two bounding boxes as input and returns the probability that a bounding box is part of another.In this case each input example

is the concatenations of two single bounding boxes plus two joint features: the containment ratios between two bbxes. Given $Area(b)$ the area of a bounding box b , the containment ratios between two bounding boxes b_1 and b_2 are :

$$\frac{Area\,(b_1) \cap Area\,(b_2)}{Area\,(b_1)} \qquad \frac{Area\,(b_1) \cap Area\,(b_2)}{Area\,(b_2)} \tag{3.1}$$

The sample batches are replaced with other examples after a number of iterations specified in the **frequency batch generation** hyperparameter.

Predicate The class **Predicate** implements Equation 2.18. Each predicate takes the batch of positive examples and the batch of negative examples as input and evaluates the probability score for each example. In the code has been implemented a Predicate object for each class that we want to predict plus a predicate for the partOF.

Literal The **Literal** class does the negation of predictions on negative examples. The semantics of logical negation can be rendered through different fuzzy logics (Łukasiewicz,Product and Goedel).

Clause The class **Clause** performs a first aggregation of literals. This class defines logical clause for K as explained in 2.5. If the number of literals passed is greater than one, in this class it is possible to logically combine them in order to define **logical constraints**. Examples of logical constraints state, for instance, that the part-of relation is asymmetric ($\forall xy(\mathsf{partOF}(x,y) \rightarrow \neg\mathsf{partOF}(y,x))$) or that every whole object cannot be part of another object ($\forall xy(\mathsf{Person}(x) \rightarrow \neg\mathsf{partOF}(y,x))$). The semantics of logical connectives can be expressed using one of the many definitions of fuzzy logic (Lukasiewicz, product or Godel). Given a Literal (or a logical combination of literals into more complex logical clauses), this class aggregates all examples by using a specified **clause aggregator**. The possible choices for this hyperparameter are minimum,mean, harmonic mean and product. We can define a clause for each logical formula we want to introduce in the training. All clauses that we have defined compose the knowledge $\mathcal{K}$ that must be satisfied by the grouding $\mathbb{G}^*$. At the end, for each clause is computed a probability score that is the result of the aggregation of all scores associated with each example in the batch.

Loss The final loss is computed by aggregating all the clauses. The possible choices for **aggregator** are minimun,mean,harmonic mean and weighted mean. The objective of the training is to maximize the truth of the knowledge encoded in each istance of the class **clause** and so minimize the negative value of the aggregation of all clauses. As a regularization term, the square norm of all parameters to be trained is added to the loss.The contribution of this term to the loss can be weighted by the **smooth factor** parameter.

3.3 Keras implementation of LTNs

The first step is to replace the work [8] in order to integrate them in an object detection architecture. The starting point are the code and the data made avariable by the authors of [8]. In

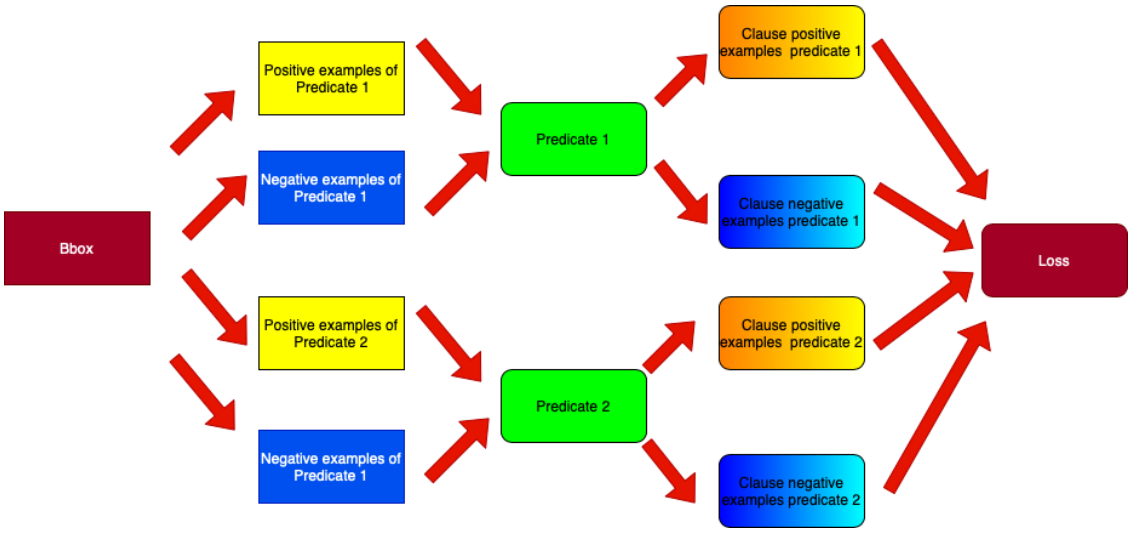


Figure 3.1. The architecture of LTNs.

Figure 3.1 depicts the architecture of Keras LTNs implementation at training time.

The logic inside the Keras implementation is similar to the original one. The Keras library is a wrapper for the Tensorflow library. The advantage obtained using Keras is given by a greater readability of the code obtained by using high-level functions. Furthermore, this implementation facilitates the final integration work, since the code used for the Faster RCNN uses Keras. The logic of **Predicate**, **Literal** and **Clause** is implemented through Keras Layer classes, a class made available by the library that allows to define the internal logic of a basic building blocks of neural networks.

3.4 Experiments

To be sure that Keras implementation obtains comparable results with the original one, both versions have been trained and tested using the same data provided by the authors. In Table 3.1 are shown the hyperparameters setted to train the two models. Different configurations of learning rate and smooth factor have been tried and in Table 3.1 have been inserted only those that allow to obtain the best performance.

Table 3.1. The parameters used for the training of the two models.

Hyperparameters	LTNs original implementation	LTNs Keras implementation
Learning rate	1e-3	1e-3
Optimizer	RSMProp	RSMProp
Decay	0.9	0.0
Smooth factor (Lambda)	1e-8	1e-4
t-norm	Łukasiewicz	Łukasiewicz
aggregator	harmonic mean	harmonic mean
clause aggregator	harmonic mean	harmonic mean
k	6	6
batch size	250	250
frequency batch generation	100	100

Figure 3.2 and Figure 3.3 show the trend of LTNs armonic mean loss in the training of the two models.

Figure 3.4 and Figure 3.5 illustrates the Precision-Recall curves of the two models. These graphs are computed by using the evaluation code provided by the authors and the test split of data. We can see that the performances reached by the two models are comparable,(near to the official ones) and this certifies that the Keras implementation reflects the state of the art of LTNs.

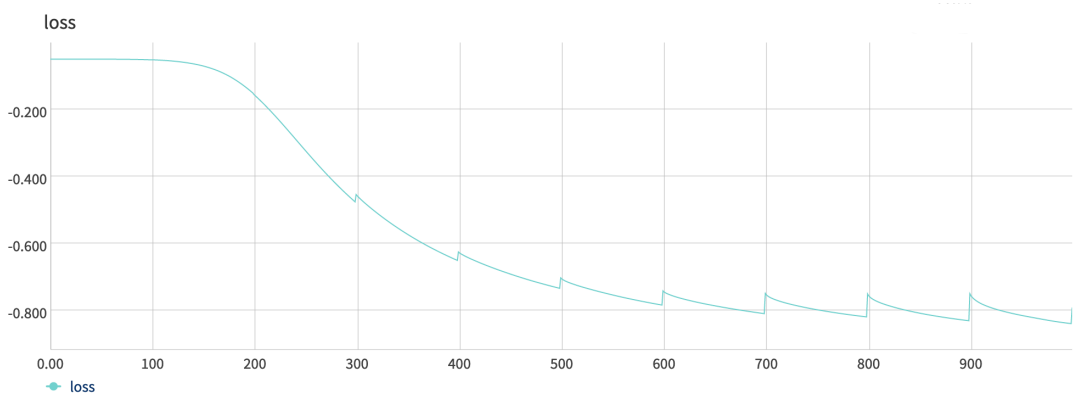


Figure 3.2. The loss of official implementation of LTNs.

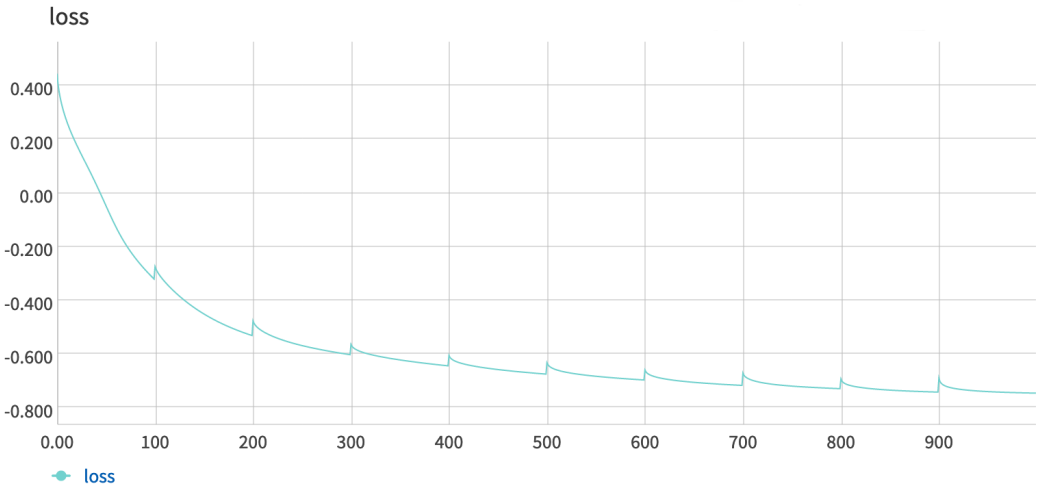


Figure 3.3. The loss of Keras implementation of LTNs.

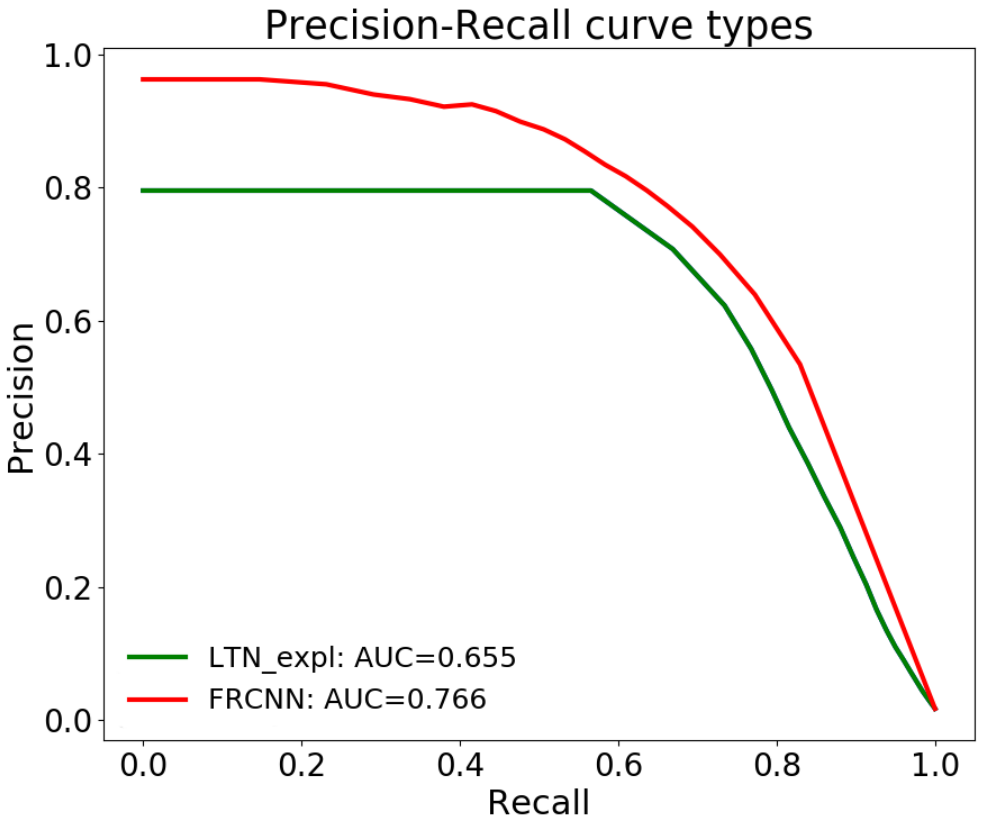


Figure 3.4. The Precision-Recall curve of official version of LTNs.

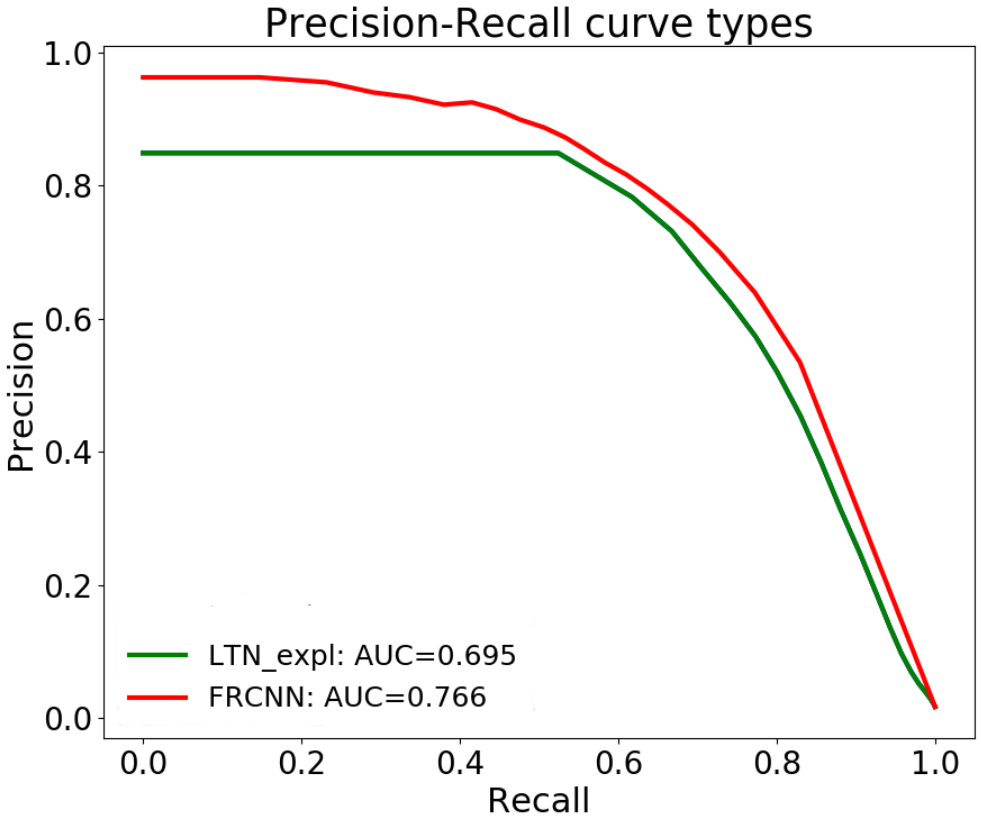


Figure 3.5. The Precision-Recall curve of implemented version of LTNs.

Chapter 4

Faster-LTN

4.1 Introduction

After defining the Keras LTNs implementation, the next step is to integrate the LTN module inside the pipeline of an object detector to try to improve its predictions. The dataset used by the authors of [8] collects a set of bounding boxes extracted by a Fast R-CNN [12]. The LTNs and the object detector are trained and tested separately.This chapter illustrates the main steps that led to the definition of Faster-LTN for joint training. Instead of the obsolete Fast RCNN [12], the more recent Faster RCNN [34] was taken as a starting point. The design of that has required the solution of many problems:

- **Architecture**: It is fundamental to understand how to insert the LTN module in the object detector architecture to maximize the results. At the point of the network where the LTN will be inserted, it will have to be modified in order to better accommodate the new layers. In disjoint training, a trained object detector extracts the features for the LTN. In this case, the two models will have to be trained together. This adds complexity to the Faster RCNN. The number of trainable parameters increases from 24,087,976 of Faster RCNN to 25,406,296 of Faster-LTN. Furthermore, in [8] the background class is not implemented, which is fundamental for Faster-LTN because allow to discriminate input elements that do not belong to any class.

- **Batch learning**: Vanilla LTNs at training time accepts as input a fixed batch of data and iterates over it for many iterations. Faster RCNN ,at each iteration, extracts a batch of positive or negative examples from an input image. For this reason, the number of objects associated with a class at each iteration cannot be fixed. The objects predicted in an iteration depend on the current image that the model receives from input.

In Section 4.2 there is an introduction of the object detection module adopted

for the following experiments [34] and a brief overwiew of main object detection modules in literature. Section 4.3 introduces the key characteristics of the proposed Faster-LTN. Section 4.4 exhibits the datasets used to train and test the FRCNN-LTN model and the results obtained.

4.2 Faster R-CNN

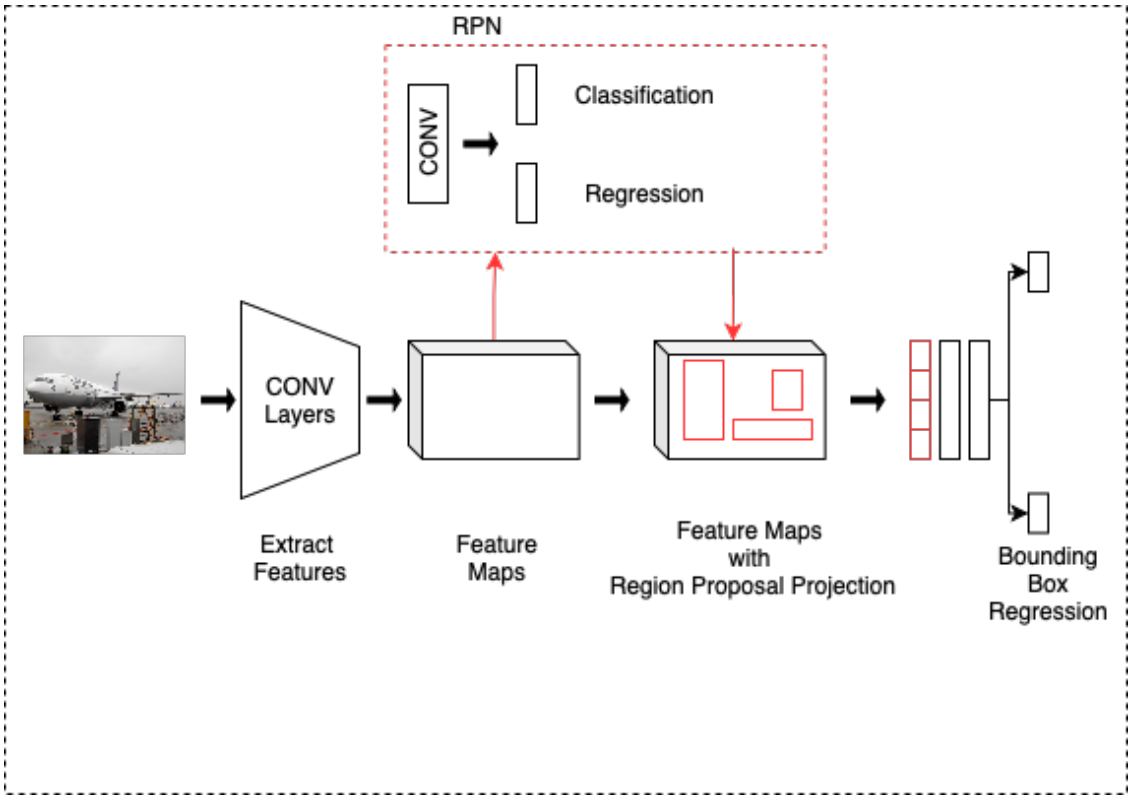


Figure 4.1. The architecture of Faster R-CNN.

Faster R-CNN has represented the object detection state of Art in the past years.It was the first deep learning model that was able to achieve this task without using region proposal extraction algorithms like selective search.The figure 4.2 is taken from the paper [34] and depicts the architecture of the original Faster RCNN. Its architecture is composed of a CNN network as a backbone that is trained end to end with 2 learnable modules : the RPN module and the detector.

Backbone In this work, Faster R-CNN uses a ResNet50 network [14] as backbone. ResNet, short for Residual Network, proposes a way to build deeper network architectures avoiding the vanishing gradient problem.

It is easy to think that a deeper network allows for better performance. It has been observed that this is true as long as the net is not too deep. If the network is too deep, during the back-propagation towards the initial layers, it tends to assume values close to zero (the chain rule multiplies error gradient values lower than one and then, when the gradient error comes to the first layers, its value goes to zero).

It is possible to have the opposite problem , the gradient value tends indiscriminately towards high values as it approaches the initial layers. This phenomenon is called gradient exploding. A ResNet network is composed of many stacked basic blocks, called **residual blocks**. A **residual block** outputs $H(x) = F(x) + x$, where x is the input of the block, $F(x)$ is the function fitted by the trainable layers inside the block.

The input of the block is added to the output, this operation is called skip connections. 16 residual blocks are stacked in ResNet50 and each residual block contains 3 different convolutional layers.

Region Proposals Network The feature map extracted by the backbone is the input of the RPN module. This module is responsible to extract and filter region proposals.

To generate proposals RPN is slided over the feature map.The network takes as input a $n \times n$ spatial window. Each sliding window is mapped to a lower-dimensional feature. This feature is fed into two sibling fully- connected layers: a box-regression layer and a box-classification layer. At each sliding window location, we simultaneously predict k different region proposals. The k proposals are relative to k reference boxes called **anchors**. These anchors are computed by considering different combination of scale and aspect ratios. For example, if we consider 3 different scales [128,256,512] and 3 different aspect ratios $1:1, 1:2, 2:1$ the number of possible anchors k is equal to 9. If the feature map has width W and height H the number of proposals is equal to WHk. They need to be filtered.

The model performs a binary classification over the extracted proposals in order to distinguish between positive and negative proposals. Positive proposals are those that reach an Interception over Union overlap with any ground-truth box greater than a given threshold.Negative proposals are those for which the IoU score is lower than 0.3 for all ground-truth boxes. Proposals that are not positive and no negative do not participate to the training. The loss relative to this module is:

$$L\left(\{p_i\}, \{t_i\}\right) = \frac{1}{N_{cls}} \sum_i L_{cls}\left(p_i, p_i^*\right) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}\left(t_i, t_i^*\right) \tag{4.1}$$

The ground-truth label p_i^* is 1 if the anchor is positive, 0 otherwise. p_i is the predicted probability of anchor i is being an object. t_i is a vector representing the parametrized 4 coordinates of the predicted bounding box. t_i^* represents the parametrized 4 coordinates of the ground truth. The classification loss L_{cls} is log

loss over two classes (object vs. not object). The regression loss is:

$$L_{reg}\left(t_i, t_i^*\right) = R\left(t_i - t_i^*\right) \tag{4.2}$$

where R is the robust loss function(smooth L_1) defined in [12].

Detector module This model accepts as input the coordinates of the extracted and filtered region proposals plus the feature map from the backbone. The first problem to solve is to map each proposal to an equal size vector representation. This is the aim of RoI pooling. RoI pooling for every region of interest from the input, it takes a section of the input feature map that corresponds to it and divides that section into a fixed number of regions k. Finally, it applies max pooling to all obtained regions. In this way each proposal is associated to a feature vector of k elements. After some fully connected layers the module ends with two sibling output layers. The first output a discrete probability $p = (p_0, ..., p_K)$, over $K categories + background class$. This probability is computed by using the softmax function over the output of a fully connected layer. The second sibling layer outputs bounding-box regression offsets, $t^k = t_x^k, t_y^k, t_w^k, t_h^k$ for each of the K object classes, indexed by k. The multitask loss takes the form of Equation 4.1. The classification loss is the log loss for the true class. The regression loss is the robust loss function(smooth L_1).

4.2.1 Other object detection works

Faster RCNN and all the models of the region proposal family (RCNN [13],Fast RCNN [12] and Faster RCNN [34]) are called **two stages detectors**. The detection happens in two stages: First, the model proposes a set of regions of interests by using selective search or region proposal network. The extracted proposals are filtered and only the selected proposals are processed by the classifier.

Other models are called **one stage detectors**. These approaches skip the region proposal stage and runs detection directly over a dense sampling of possible locations. One stage detectors are simpler and faster than two stages ones but may potentially lose in performance.

YOLO (You Only Look Once) [33] applies a CNN to the full input image.Then divides the image into $S \times S$ regions. For each regions predict bounding boxes and probability. Improvements of original YOLO are given by YOLOv2 / YOLO9000 [31] , YOLOv3 [32] and YOLOv4 [3].

Single Shot Detector [26] is one of the first works that uses a convolutional neural network's pyramidal feature hierarchy for efficient detection of objects of various sizes. On top of the backbone network, SSD adds some convolutional layers of decreasing size. Earlier large layers are suitable to detect small size objects, instead later small layers are more suitable to detect biger objects. The detection is performed at each pyramid layer.

An improvement of pyramid concept is introduced by Feature Pyramid Networks [23]. FPN is a new backbone for object detection. FPN performs a set of convolutional layers for feature extraction. The outputs of the past convolutional layers are reconstructed by upsampling the last feature map. In this way we have reconstructed layers that are semantically stronger. In order to predict the locations better, lateral connections between the reconstructed layers and the corresponding feature maps are added.

The RetinaNet [24] exploits FPN as backbone. Another improvement is given by this work is the definition of focal loss function.

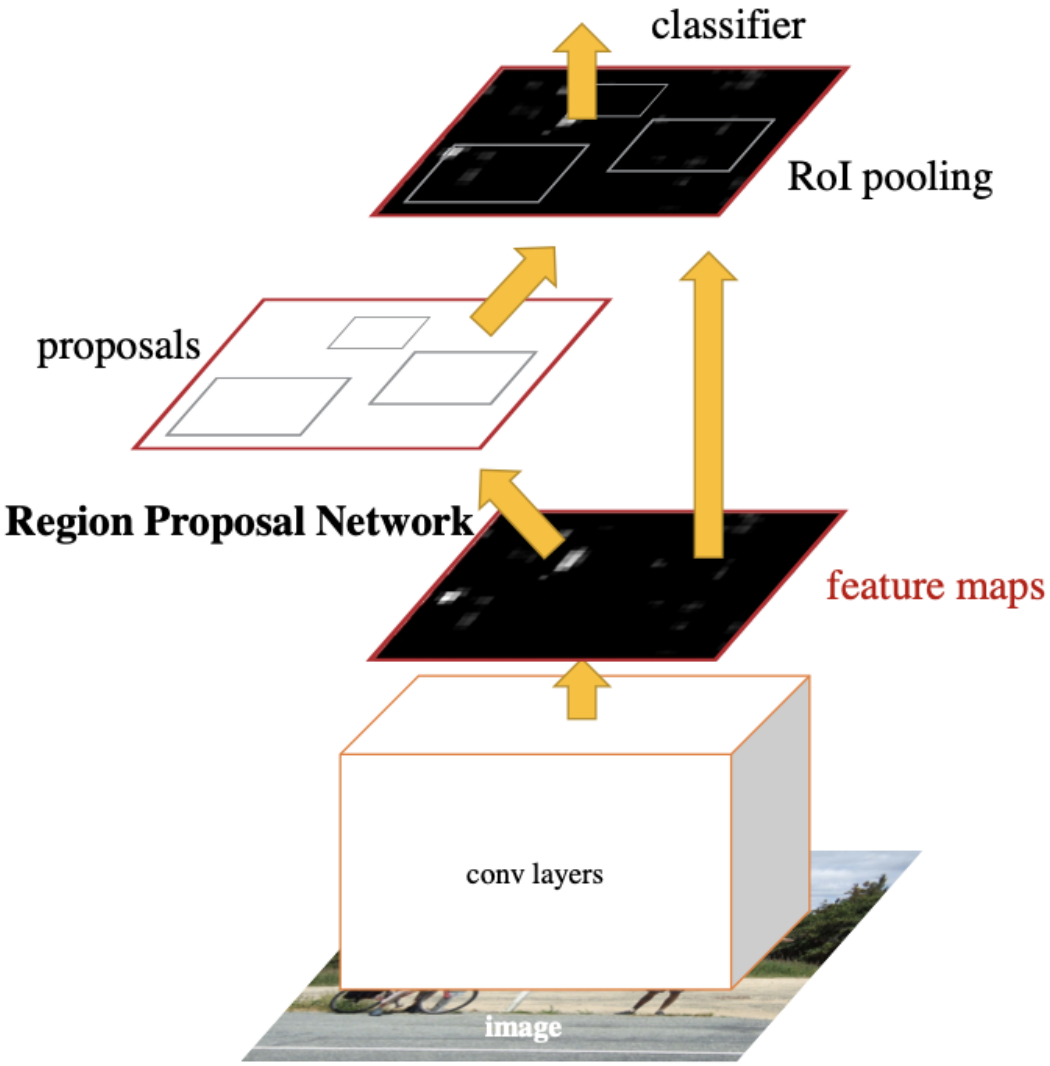


Figure 4.2. The architecture of Faster R-CNN. Picture taken from the original paper of Faster RCNN [34].

4.3 Integration of LTN in Faster RCNN

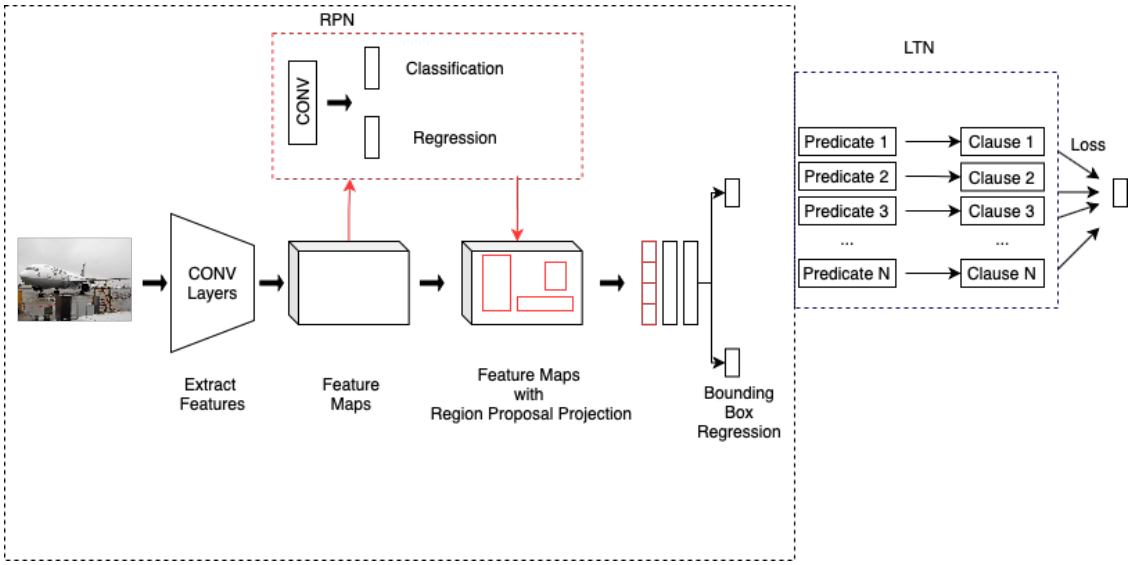


Figure 4.3. The architecture of Faster - LTN module.

In [8] and [7] LTN receives as input a set of labeled bounding boxes extracted from images by an object detector like [34]. LTN is used to improve the classification of the object detector. This aspect was the starting point for the end to end integration of the two models.

In the specific case of the Faster R-CNN, the LTN training module was inserted at the top of the detector module of the object detector. As was explained in Section 4.2 the detector module of Faster RCNN ends with two sibling layers. The first is for classification and compute the probability scores of each of $K+1$ classes. To do that, this layer is used as activation the **Softmax** function. The second layer compute for each class a tuple of four parameters used to adjust the prediction of the bounding box.

The idea is to use as input of LTNs Predicate layers the output of Faster RCNN classification layer. The softmax activation function is removed to avoid the gradient vanishing problem and so, the FRCNN classification loss is substitute by the LTNs loss.

Predicate This connects the network of the Faster RCNN with the LTN predicates. For the classification alone, a predicate must be defined for each class to predict. Predicate layer is implemented following 2.18.

In the original faster RCNN, the softmax function guarantees a strong mutual exclusion between classes and with the integration the model loses this peculiarity.

However, in addition to classification Predicates it is also possible to insert binary or more Predicates, such as the partOF, which increases the capabilities of the model. Each predicate outputs a score between 0 and 1 for each example in the batch.

Literal Starting from these probability scores we need to define the literals.In logic, a literal is an atomic formula or its negation. All false statements are negated. The semantics of the logic NOT operator is given by the logic of Łukasiewicz:

$$F_{\neg}(x) = 1 - x \tag{4.3}$$

The implementation adds a set of vectors y_j, one for each j-th class, whose elements are equal to 1 if the i-th example belongs to the j-th class, 0 otherwise. The formula of literal is:

$$Literal_j(x_i) = \begin{cases} x_i & if\ y_{j,i} = 1 \\ 1 - x_i & otherwise \end{cases} \tag{4.4}$$

Clause and aggregation of literals This class defines all logical clauses that define the knowledge K that we want to satisfy. Different from [8] K is defined at each iteration, considering the examples extracted from the current input image. The logical formulas that can be defined in this class can be related to a single predicate (and its negation), or to the relationships between predicates logically expressed using logical connectives.

The semantics of the AND and OR operators is given by the Łukasiewicz t-norm and t-conorm. This aspect allows the object detector to receive a series of information on the classes to be identified and on the relationships between them in the form of logical propositions. It is necessary to aggregate all the probability scores of all clauses / literals formulated to be able to calculate the final loss,that is a second level of aggregation over all aggregated scores. Figure 4.3 provides a visual representation of Faster-LTN. This is the general architecture of the Faster-LTN integration in the training phase.

Class imbalance problem Given that there are K predicate layes, one for each class, which, as explained so far, autonomously provide their predictions, we pass from a multiclass classification through the softmax function, as done by the original Faster -RCNN, to K binary classifications.

This has brought out a problem of unbalancing of the classes. In a batch, on average only one or a few examples are associated with the same class, the rest are examples associated with other classes or backgrounds, therefore negative examples.

The harmonic mean as an aggregation function, as done by the official implementation, because it does not weigh adequately the contribute of positive examples in estimating the error. If we have many currently classified negative examples and a few misclassified positive examples, the harmonic mean will give an optimum value.

The work [16] compares the main t-norms, t-conorms and aggregation functions used in neurosymbolic architectures, highlighting their advantages and disadvantages related above all to their derivability. A particularly interesting aggregation function is the log-product:

$$A_{log\ sum} = \sum_{i=0}^{N} log\,(x_i) \tag{4.5}$$

where x_i is the literal of i-th example in the batch of size N. The partial derivative of this function is equal to $\frac{\partial A_{log\ sum}}{\partial x_i} = \frac{1}{x_i}$. For very low literals the derivative is very high and it is nonvanishing almost everywhere. Considering the literal formula, this aggregation function is nothing more than the cross entropy loss.

To solve the class imbalance problem, we define a new aggregation function called **focal log sum**

$$A_{focal\ log\ sum} = -\sum_{i=0}^{N}(1 - x_i)^{\gamma} log\,(x_i) \tag{4.6}$$

This function is no more than the **focal loss** that was presented in [24] applied to logical aggregation of terms. In RetinaNet [24] for each pyramid layer, more anchor boxes can be extracted and only a small part of these can be assigned to an object in the ground truth while the vast majority are background classes.

The usage of this function instead of cross-entropy can allow to mitigate the class imbalance problem generated by the architecture. It is therefore very useful in situations where the model is able to identify many true negatives but has difficulty in recognizing the true positives given their small number. This is just what happens in LTN predicate training. Table 4.1 a comparison between the focal loss with $\gamma = 2$ and the cross entropy loss is proposed. A correctly classified example ($pt = 0.9$) is evaluated 2 orders of magnitude less in the focal loss.

The final loss is the sum of all the aggregations of all literals of all clauses.

$$Loss = -\sum_{j=0}^{K}\sum_{i=0}^{N}(1 - x_{i,j})^{\gamma} log\,(x_{i,j}) \tag{4.7}$$

where $x_{i,j}$ is the literal of proposal i-th in the class j-th.

Table 4.1. Comparison of Focal Loss with gamma = 2 and Cross-Entropy loss.

pt	FL	CE
0.1	-1.865	-2.303
0.9	-0.001	-0.105
0.5	-0.693	-0.173

Inference phase In the inference phase, the definition and aggregation of literals and the definition of any logical constraints are removed. The classification vector is given by the concatenation of the scores calculated by each predicate for each input proposal in the batch. Figure 4.4 shows the FRCNN-LTN integration at the inference time.

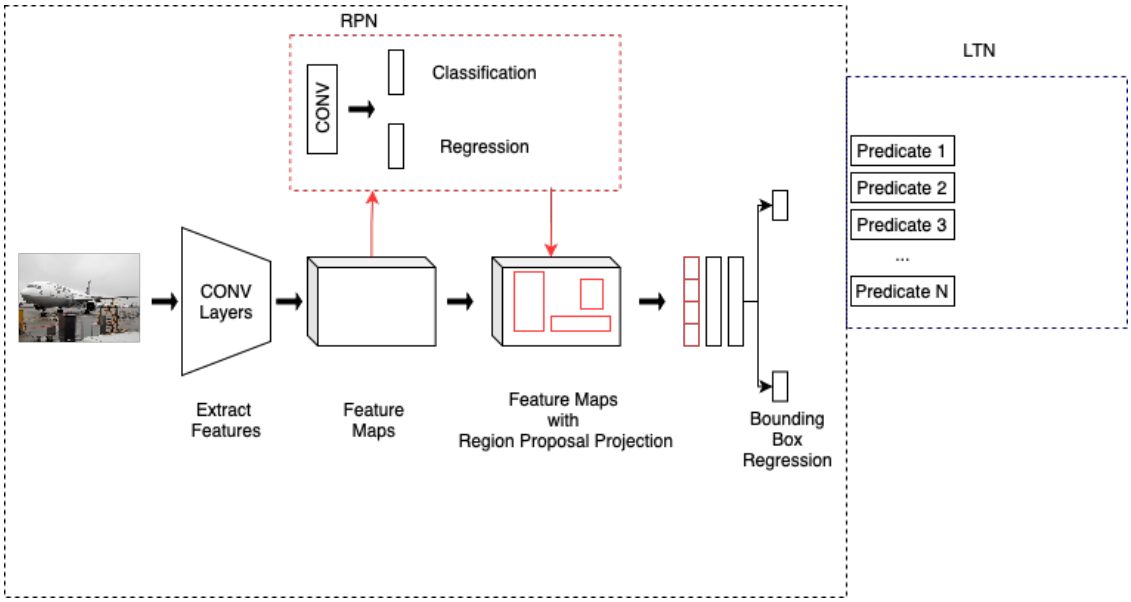


Figure 4.4. The architecture of Faster-LTN module at inference time.

4.4 Experiments

4.4.1 Introduction

In this section are shown the results of some experiments performed using Pascal VOC and its PASCAL-Part extension as datasets. All experiments try to measure the performance of Faster RCNN-LTN in classification with respect to the Faster RCNN.Therefore, only unary classification predicates are considered and no logical constraints are inserted.

4.4.2 PASCAL VOC

The dataset **PASCAL VOC** is shown in this section. It has been used in the following experiments. The Pattern Analysis,Statistical modelling and Computational Learning (PASCAL) is a Network of Excellence funded by the European Union. It also promotes the use of machine learning in many relevant application domains such as machine vision, speech, haptics, brain-computer interface, user-modelling for computer-human interaction, multimodal integration, natural language processing, information retrieval and textual information access. From 2005 to 2012, It carried out the PASCAL Visual Object Classes project. The objectives of this project are as reported by the official website:

- Provides standardised image data sets for object class recognition.
- Provides a common set of tools for accessing data sets and annotations.
- Enables evaluation and comparison of different methods.
- Run challenges evaluating performance on object class recognition.

The proposed challenges are inherent to different tasks related to computer vision. Different versions of the dataset have been published from 2005 to 2012. The differences are due to the different challenges proposed. For these experiments was chosen the **PASCAL VOC 2010** version. It collects 10103 images containing 23374 annotated objects,divided into 20 classes:

- **Person**: person.
- **Animal**: bird, cat, cow, dog, horse, sheep
- **Vehicle**: aeroplane, bicycle, boat, bus, car, motorbike, train
- **Indoor**: bottle, chair, dining table, potted plant, sofa, tv/monitor.

The split proposed by the authors of the dataset was used to train and test the model: 4998 images for training and 5105 for testing.

4.4.3 Pascal Parts

This dataset is a set of additional annotations for PASCAL VOC 2010. These new classes are related to the components of the objects tagged by the PASCAL VOC dataset. Table 4.2 shows for each original class of PASCAL VOC, all its components are recognized and labeled by the dataset.

Table 4.2. The PASCAL VOC objects and all their labelled parts.

Objects	Parts
Aereoplane	body, engine, wing, stern, tail, wheel.
Bycycle	wheel, handlebar, headlight, saddle.
Bird	beak, head, eye, foot, leg, wing, neck, eye, foot, leg, wing, tail, torso.
Boat	
Bottle	body, cap.
Bus	license plate, side, door, license plate, headlight, mirror, wheel, window.
Car	license plate, side, door, license plate, headlight, mirror, wheel, window.
Cat	head, leg, paw, ear, eye, leg, neck, nose, tail, torso.
Chair	
Cow	head, leg, ear, eye, horn, muzzle, neck, tail, torso.
Diningtable	
Dog	head, leg, ear, eye, horn, muzzle, neck, tail, torso.
Horse	head, leg, ear, eye, horn, muzzle, neck, tail, torso.
Motorbike	wheel, handlebar, headlight, saddle.
Person	hair, head, ear, eye, eyebrow, foot, hand, leg, arm, mouth, neck, nose, torso.
Pottedplant	plant, pot
Sheep	head, leg, ear, eye, horn, muzzle, neck, tail, torso.
Sofa	
Train	coach, head, headlight.
Tvmonitor	screen.

4.4.4 Experiments on PASCAL VOC

First we want to compare the performance obtained from the original Faster R-CNN with the integrated Faster-RCNN-LTN model. In these first experiments no logical constraint is implemented. Table 4.3 shows the parameters used for the first 5 experiments.

Weights the classes Starting from the model defined in the previous chapter, different configurations and combinations of these have been tested. A term α has been added to the focal focal log sum function, defined in Equation 4.6. Equation 4.6 becomes:

$$A_{focal\ log\ sum} = -\sum_{i=0}^{N} \alpha(1-x_i)^{\gamma} log\,(x_i) \tag{4.8}$$

The α parameter is used to weight the contribution of the predictions associated with each predicate in the calculation of the final loss. It therefore depends on the number of examples associated with a given predicate in the dataset. Given $p\,(\mathcal{C}) = \frac{examples\ \in\ \mathcal{C}}{Total\ examples}$ we can defines :

$$\%pos\mathcal{C}\ = \frac{batch\ size}{2} p\,(\mathcal{C}) \tag{4.9}$$

$$\%neg\mathcal{C}\ = \frac{batch\ size}{2} + \frac{batch\ size}{2}\,(1-p\,(\mathcal{C})) \tag{4.10}$$

This values try to estimate the number of examples labeled as Predicate c that are present in the batch. The parameters *alpha* for positive and negative examples for Predicate $pos\mathcal{C}$ are computed by using the formulas:

$$\alpha_{pos\mathcal{C}} = \frac{1-\beta}{1-\beta^{\%pos\mathcal{C}}} \tag{4.11}$$

$$\alpha_{neg\mathcal{C}} = \frac{1-\beta}{1-\beta^{\%neg\mathcal{C}}} \tag{4.12}$$

In this way, in addition to overweighting the positive examples of all classes compared to the negative ones, the contribution of the examples of the most represented predicates in the dataset is underweighted.

Predicate background The Predicate background is not implemented in the original version of the LTN. In Faster-RCNN it is fundamental because the model performs a multiclass classification by using the softmax function and this requires that each example can be associated with a class. Most of the examples extracted from the RPN can not be associated with objects in the ground truth and therefore are labeled as background.

Faster-LTN ,as explained before, performs n independent binary classifications and so, the class background is optional. However, if you do not use a background predicate, you force the model to classify each extracted object. This leads to a considerable increase of false positives and an underestimation of the Average Precision. Table 4.3 shows parameters that have been set to train the models. Table 4.4 depicts the Average Precisions obtained by the trained model for each class.

Figure 4.5 illustrates the Precision-Recall curves for all models.

Finally, Figure 4.6 and Figure 4.7 show the output of Faster-RCNN and Faster-LTN with bg on an image of the test set of PASCAL VOC dataset.

Table 4.3. The parameters that have been set to train the models on the PASCAL VOC dataset.

Hyperparameters	FRCNN	Faster-LTN	Faster-LTN alpha	Faster-LTN bg	Faster-LTN bg alpha
Learning rate classifier	1e-5	1e-5	1e-5	1e-5	1e-5
Learning rate rpn	1e-5	1e-5	1e-5	1e-5	1e-5
Optimizer classifier	Adam	Adam	Adam	Adam	Adam
Optimizer rpn	Adam	Adam	Adam	Adam	Adam
Anchor scales	128,256,512	128,256,512	128,256,512	128,256,512	128,256,512
Lambda rpn classifier	1	1	1	1	1
Lambda rpn regression	1	1	1	1	1
Lambda classifierltn	1	1	1	1	1
Lambda regression	1	1	1	1	1
rpn min overlap 0.3	0.3	0.3	0.3	0.3	0.3
rpn max overlap 0.7	0.7	0.7	0.7	0.7	0.7
classifier min overlap	0.1	0.1	0.1	0.1	0.1
classifier max overlap	0.5	0.5	0.5	0.5	0.5
batch size	32	32	32	32	32
epoch lenght	1000	1000	1000	1000	1000
num epochs	80	300	300	300	300
num classes	21	20	20	21	21
t-norm		Łukasiewicz	Łukasiewicz	Łukasiewicz	Łukasiewicz
clause aggregator		focal log sum	focal log sum	focal log sum	focal log sum
k		6	6	6	6
gamma		2	2	2	2
beta			0.9999		0.9999

Table 4.4. Results obtained on PASCAL VOC dataset.

Class	FRCNN	Faster-LTN	Faster-LTN alpha	Faster-LTN bg	Faster-LTN bg & alpha
aeroplane AP	0.82	0.71	0.70	**0.85**	0.75
bicycle AP	**0.78**	0.62	0.64	0.76	0.76
bird AP	**0.79**	0.62	0.55	0.71	0.69
boat AP	0.55	0.36	0.31	**0.57**	0.45
bottle AP	**0.64**	0.36	0.35	0.54	0.39
bus AP	0.71	0.69	0.74	**0.82**	0.74
car AP	**0.85**	0.52	0.53	0.76	0.61
cat AP	0.90	0.84	0.85	**0.92**	0.83
chair AP	0.47	0.30	0.30	**0.56**	0.36
cow AP	**0.69**	0.52	0.55	0.64	0.64
diningtable AP	0.58	0.46	0.38	**0.69**	0.48
dog AP	**0.87**	0.81	0.80	0.86	0.83
horse AP	0.79	0.74	0.71	**0.83**	0.74
motorbike AP	**0.86**	0.73	0.72	0.80	0.75
person AP	**0.84**	0.70	0.67	0.82	0.75
pottedplant AP	**0.44**	0.29	0.25	0.39	0.32
sheep AP	0.73	0.59	0.60	**0.79**	0.68
sofa AP	0.52	0.48	0.45	**0.55**	0.49
train AP	0.76	0.72	0.71	**0.81**	0.75
tvmonitor AP	0.66	0.53	0.51	**0.71**	0.56
mAP	0.71	0.58	0.57	**0.72**	0.63

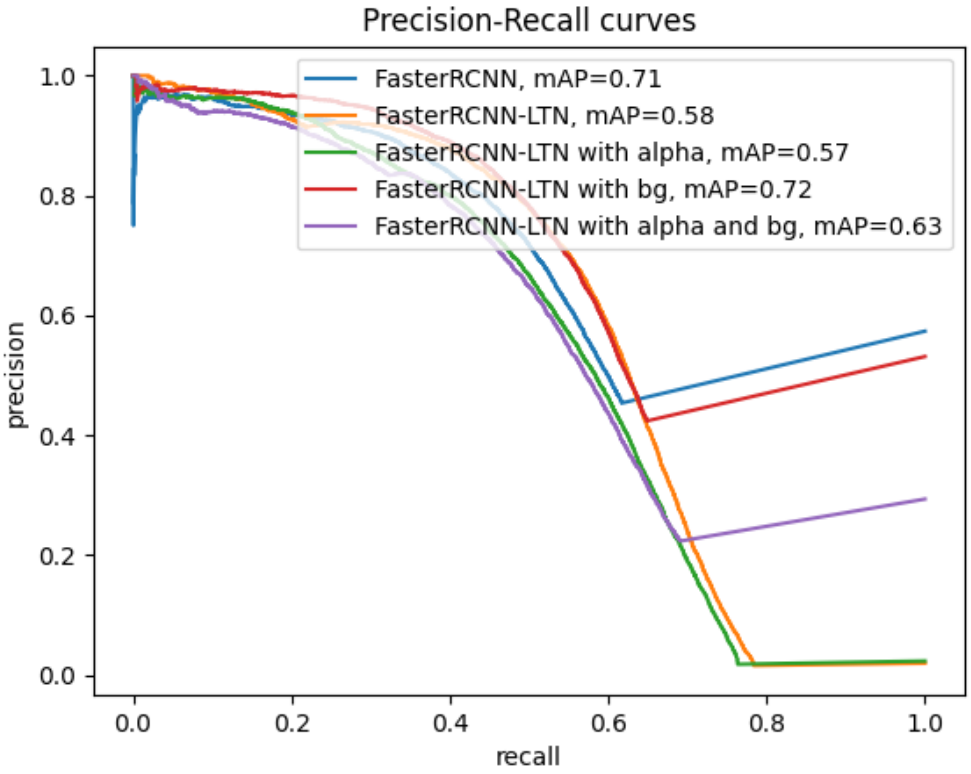


Figure 4.5. The Precision-Recall curves of the 5 trained model configurations.

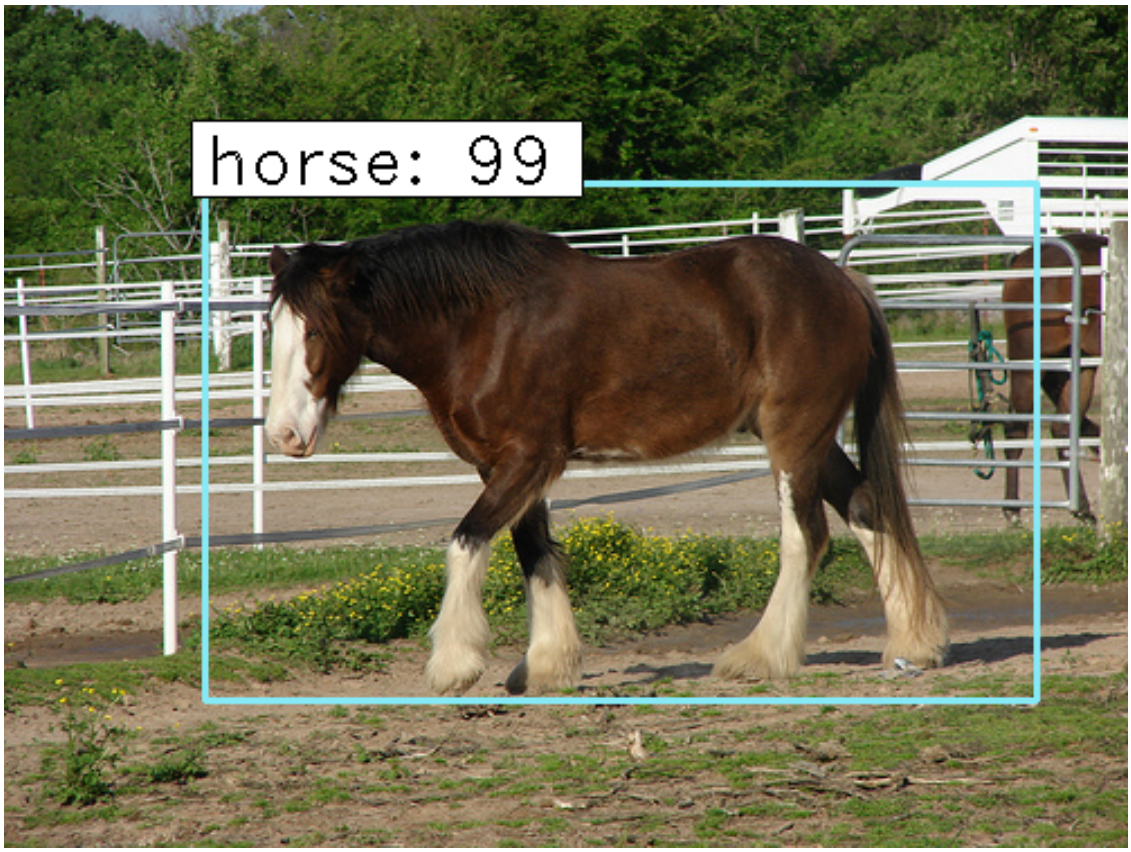


Figure 4.6. Prediction of Faster-RCNN, PASCAL VOC dataset.

4.4.5 Experiments on PASCAL-Part dataset

Two models were trained on this dataset. The original Faster RCNN and the Faster-LTN in the configuration without alpha but with the bg class.

Table 4.5 exhibits all the parameters that have been setted to train the model with PASCAL-Part dataset.

Table 4.6 illustrates the Average Precision of each trained model with respect to each class of the dataset.

On the contrary, Figure 4.8 shows the Precision-Recall curves obtained from the two models trained with PASCAL-Part, taking into account all the predictions on all the classes.

Figure 4.9 and Figure 4.10 show the output of Faster-RCNN and Faster-LTN on an image of the test set of PASCAL-Parts dataset.

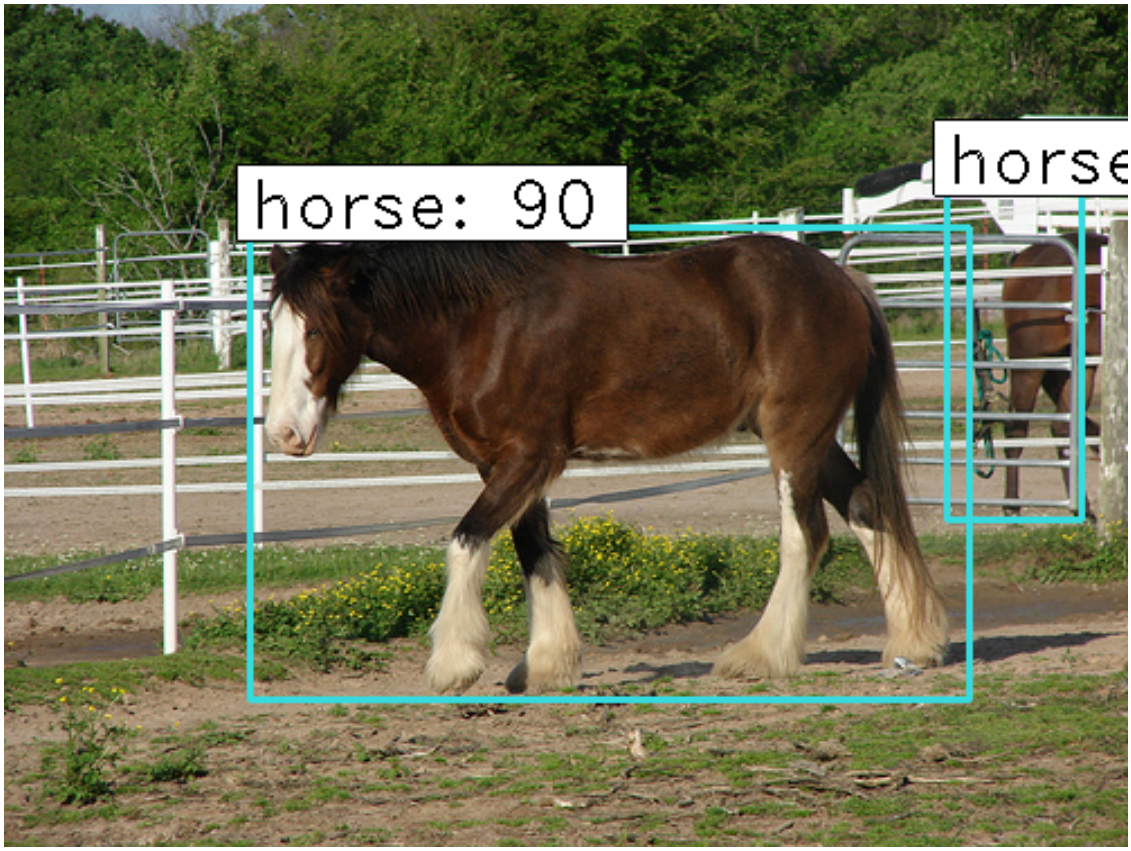


Figure 4.7. Predictions of Faster-LTN with bg, PASCAL VOC dataset.

Table 4.5. The parameters that have been set to train the models on the PASCAL-Part dataset.

Hyperparameters	FRCNN	Faster-LTN
Learning rate classifier	1e-5	1e-5
Learning rate RPN	1e-5	1e-5
Optimizer classifier	Adam	Adam
Optimizer rpn	Adam	Adam
Anchor scales	128,256,512	128,256,512
Lambda rpn classifier	1	1
Lambda rpn regression	1	1
Lambda classifierltn	1	1
Lambda regression	1	1
rpn min overlap 0.3	0.3	0.3
rpn max overlap 0.7	0.7	0.7
classifier min overlap	0.1	0.1
classifier max overlap	0.5	0.5
batch size	32	32
epoch lenght	1000	1000
num epochs	80	300
num classes	21	21
t-norm		Łukasiewicz
clause aggregator		focal logsum
k		6
gamma		2
beta		

Table 4.6. Results obtained on PASCAL-Part dataset.

Class	FRCNN	Faster-LTN
Aeroplane AP	0.86	**0.88**
Animal Wing AP	0.50	**0.67**
Arm AP	**0.74**	0.65
Artifact Wing AP	**0.70**	0.53
Beak AP	**0.71**	0.61
Bicycle AP	**0.80**	0.77
Bird AP	**0.82**	0.77
Boat AP	0.59	**0.67**
Body AP	**0.76**	0.70
Bodywork AP	0.73	**0.93**
Bottle AP	**0.61**	0.57
Bus AP	0.82	**0.90**
Cap AP	**0.89**	0.83
Car AP	0.74	**0.83**
Cat AP	**0.90**	0.88
Chain Wheel AP	**0.78**	0.75
Chair AP	**0.57**	0.51
Coach AP	0.49	**0.60**
Cow AP	0.74	**0.76**
Diningtable AP	**0.66**	0.51
Dog AP	0.81	**0.84**
Door AP	0.58	**0.74**
Ear AP	**0.80**	0.72
Ebrow AP	**0.79**	0.51
Engine AP	0.46	**0.58**
Eye AP	**0.77**	0.65
Foot AP	**0.78**	0.77
Hair AP	0.85	**0.87**
Hand AP	**0.81**	0.70
Handlebar AP	0.50	**0.67**
Head AP	**0.88**	0.87
Headlight AP	**0.70**	0.50
Hoof AP	0.72	**0.80**
Horn AP	**0.74**	0.72
Horse AP	0.77	**0.87**
Leg AP	**0.60**	0.59
License plate AP	**0.73**	0.53
Locomotive AP	0.71	**0.82**
Mirror AP	**0.82**	0.75
Motorbike AP	0.79	**0.83**
Mouth AP	**0.90**	0.68
Muzzle AP	0.81	**0.83**
Neck AP	0.68	**0.70**
Nose AP	**0.81**	0.73
Person AP	0.79	**0.83**
Plant AP	0.41	**0.47**
Pot AP	0.49	**0.70**
Pottedplant AP	0.59	**0.68**
Saddle AP	**0.84**	0.72
Screen AP	0.52	**0.68**
Sheep AP	0.80	**0.84**
Sofa AP	0.67	**0.76**
Stern AP	**0.83**	0.69
Tail AP	**0.67**	0.48
Torso AP	0.80	**0.78**
Train AP	0.68	**0.79**
Tvmonitor AP	0.62	**0.76**
Wheel AP	**0.74**	**0.74**
Window AP	0.57	**0.59**
mAP	**0.72**	0.71

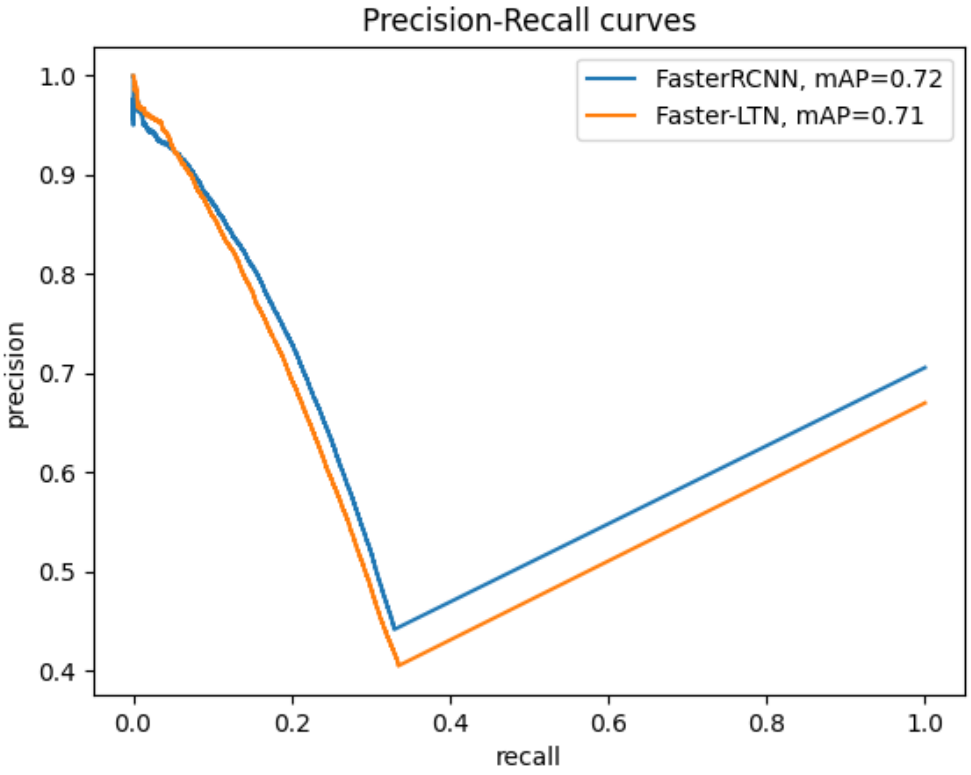


Figure 4.8. The Precision-Recall curves of the 2 trained model configurations.

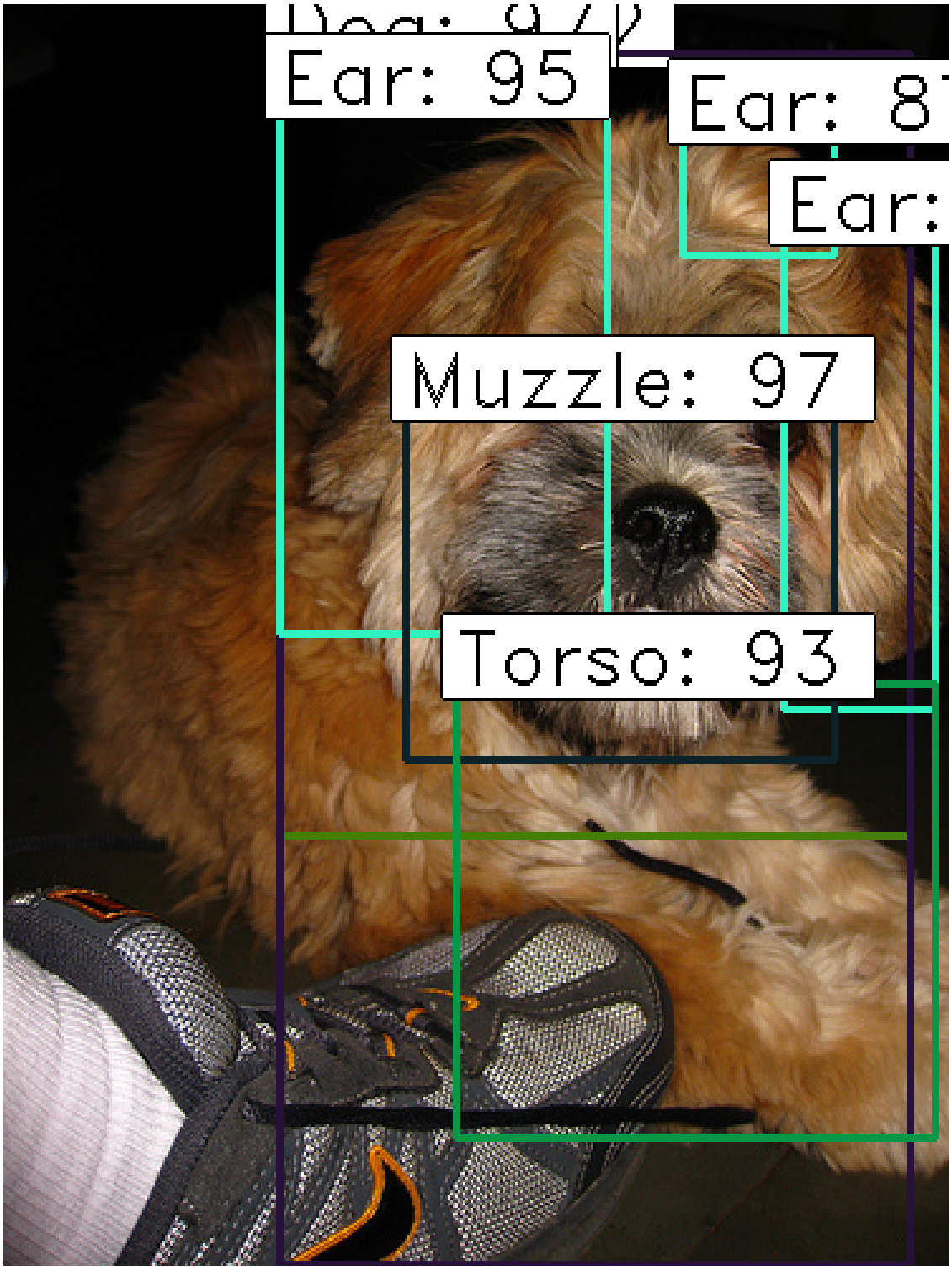


Figure 4.9. Predictions of Faster-RCNN, PASCAL-Part dataset.

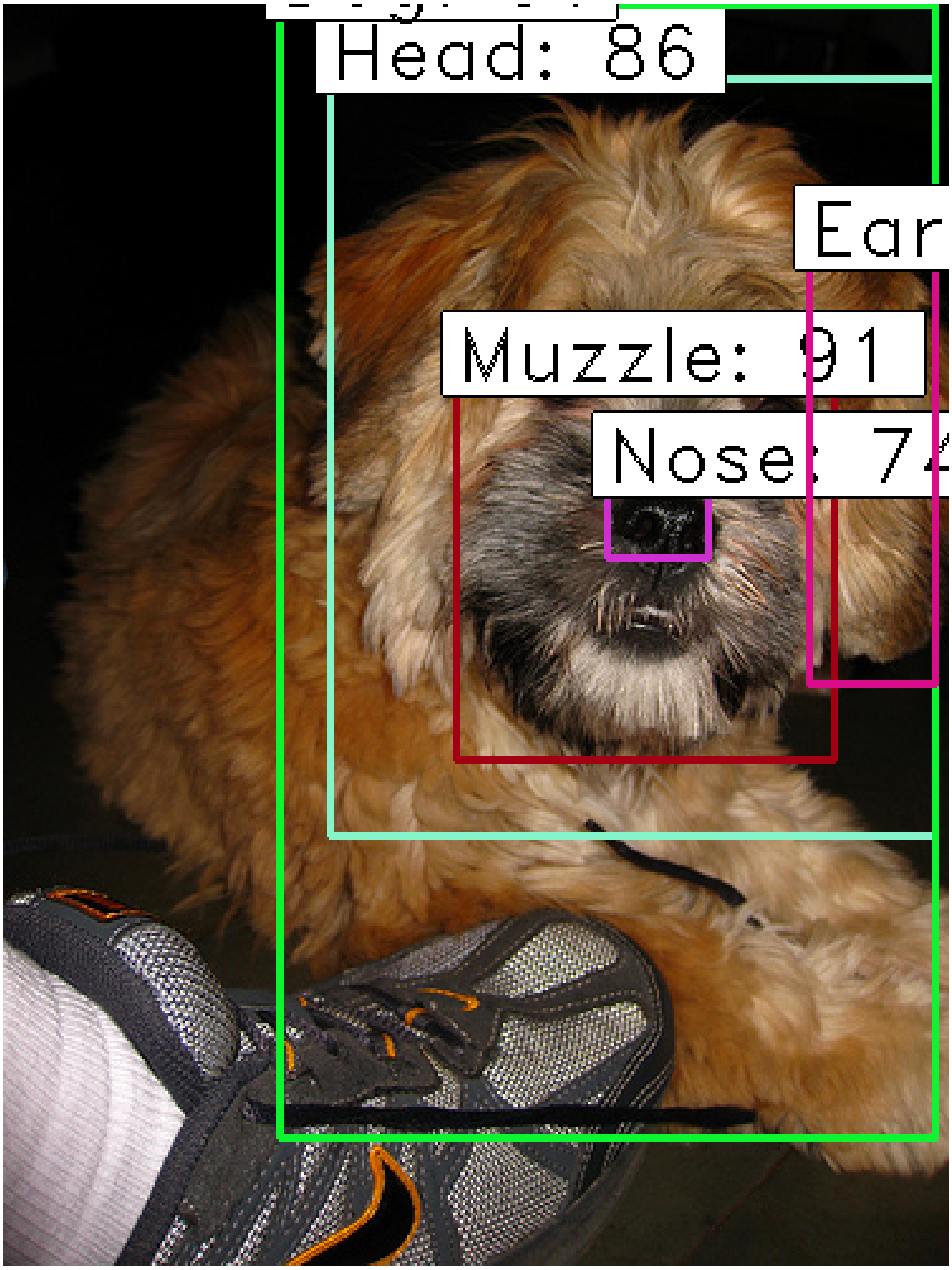


Figure 4.10. Predictions of Faster-LTN, PASCAL-Part dataset.

Chapter 5

Logical constraints for the Faster-LTN

5.1 Introduction

In this chapter are introduced some experiments performed by Faster-LTN that exploits some logical constrains at the training phase. As explained in 2.5 Logic Tensor Networks provide a grounding of a First Order logic Language(FOL). With LTNs it is possible to encode simple FOL formulas (e.g the object x is a Cat),but even more complex formulas like $\forall x\left(Cat\left(x\right)\rightarrow\neg.Dog\left(x\right)\right)$.

Given the predictions of the Predicate levels, Faster-LTN allows you to define literals and combine them logically using logical operators whose semantics are given by fuzzy logic. In this way, new clauses are obtained which contribute to the loss, therefore their truth value must be maximized.

By adding these constraints, it is hoped to provide the model with additional knowledge about the problem, in this case classification and partOf, for better performance. All axioms that were used for this work are the same introduced by [8].

5.2 Logic constrains for classification

Faster-LTN replaces multiclassification in the final layer with softmax function as backbone with n independent binary classifiers, one for each class. In this new architecture, we do not have the mutual exclusion given by the act of softmax function.

It was thought to insert in the training clauses that define mutual exclusion between classes logically. For example, if a proposal is classified as Cat, this involves that the proposal can not be at same time a Dog. In logical terms:

$$\forall x \left(Cat\left(x\right) \rightarrow \neg Dog\left(x\right) \right) \tag{5.1}$$

This expression is equivalent to:

$$\forall x \left(\neg Cat\left(x\right) \vee \neg Dog\left(x\right) \right) \tag{5.2}$$

This last formula is the one used in the code to build this kind of axiom. The semantic of disjunction is given by Lukasiewicz t-conorm:

$$S_{LK}\left(a,b\right) = min(a+b,1) \tag{5.3}$$

Given the fact that:

$$Cat\left(x\right) \rightarrow \neg Dog\left(x\right) \equiv \neg \left(\neg Dog\left(x\right) \right) \rightarrow \neg Cat\left(x\right) \equiv Dog\left(x\right) \rightarrow \neg Cat\left(x\right) \tag{5.4}$$

Are added to the model new $\frac{N\left(\dot{N}-1\right)}{2}$ clauses to define the logical implication between each possible pair of classes. Another kind of logical axiom is given by:

$$\forall x \left(Cat(x) \vee Dog(x) \vee ...Person(x) \right) \tag{5.5}$$

We want to constrain the model to assign to a given proposal x an high probability score for at least one class. At each proposal must be assigned to a class. The defined model has been trained by using PASCAL-Parts dataset. As in the experiments presented on the previous pages, for training and test sets are taken the train and validation splits proposed proposed in the annotations of the dataset.

5.3 PartOF predicate

After introducing the logical constraints in the training of the Faster-LTN it was introduced a new kind of Predicate : the partOF Predicate. This layer, given the features relative to two objects as input, must be able to understand if the first object is part of the last one. For example, if in the image are detected a Cat and a Tail, this layer will output the probability score that the Tail is part of the Cat. In Faster-LTN this layer accepts as input the logits of two object proposals ,extracted by the previous layers of the network, concatenate with the coordinates of the associated Region of Interest and two joint features given by the containment ratios between the two Regions of Interest:

$$\frac{Area_1 \cap Area_2}{Area_1} \quad and \quad \frac{Area_1 \cap Area_2}{Area_2} \tag{5.6}$$

5.3.1 Mereological constraints

The definition of partOF Predicate allows to introduce in the training a new kind of constrains more related to the characteristics of each class. We can define Mereological constrains about classes taken from external base knowledge like WORDNET [10]. In this case, axioms are considered that explain the part-whole and whole-part relationships between the various classes of the PASCAL-Part dataset. An example of whole-parts constrains can be:

$$\forall x, y \left(Cat(x) \wedge partOf(y, x) \rightarrow Tail\left(y\right) \vee Head\left(y\right) ... \vee Eye\left(y\right)\right) \tag{5.7}$$

On the contrary an example of part-wholes constrains is:

$$\forall x, y \left(Tail(y) \wedge partOf(y, x) \rightarrow Cat\left(x\right) \vee Dog\left(x\right) ... \vee Horse\left(x\right)\right) \tag{5.8}$$

An overview of mereological relationships between the classes of PASCAL parts dataset is given by Table 4.2. The formula ,used practically in the code, equivalent to 5.8 and with only OR operators is equal to:

$$\forall x, y \left(\neg Cat(y) \vee \neg partOf(y, x) \vee Tail\left(x\right) \vee Torso\left(x\right) ... \vee \text{ł}Head\left(x\right)\right) \tag{5.9}$$

This formula is obtained by exploiting the equivalence:

$$A \rightarrow B \equiv \neg A \rightarrow B \tag{5.10}$$

In this way we have:

$$\forall x, y \left(\neg \left(Tail(y) \wedge partOf(y, x) \right) \vee Cat\left(x\right) \vee Dog\left(x\right) ... \vee Horse\left(x\right) \right) \tag{5.11}$$

To remove the AND operator can be applied the **De Morgan's Law**:

$$\neg \left(A \wedge B \right) = \neg A \vee \neg B \tag{5.12}$$

Table 5.1. The parameters that have been set to train the models on the PASCAL-Part dataset.

Hyperparameters	Faster-LTN knowledge	Faster-LTN knowledge and partOF
Learning rate classifier	1e-5	1e-5
Learning rate RPN	1e-5	1e-5
Optimizer classifier	Adam	Adam
Optimizer RPN	Adam	Adam
Anchor scales	128,256,512	128,256,512
Lambda rpn classifier	1	1
Lambda rpn regression	1	1
Lambda classifierltn	1	1
Lambda regression	1	1
rpn min overlap 0.3	0.3	0.3
rpn max overlap 0.7	0.7	0.7
classifier min overlap	0.1	0.1
classifier max overlap	0.5	0.5
batch size	32	32
epoch lenght	1000	1000
num epochs	300	300
num classes	60	60
t-norm		Łukasiewicz
clause aggregator		focal logsum
k		6
gamma		2
beta		

5.4 Experiments

In this section are illustrated the results obtained by Faster-LTN with logical constrains only relative to classification (see Section 5.2) and Faster-LTN with partOf and logical constrains (see Section 5.3). In this last training are defined all the axioms relative to classification plus ones relative to mereological relations between classes.

Table 5.1 shows the parameters used for the training.

Table 5.2 instead shows the Average Precision scores of the Faster-LTN knowledge together with APs of the other trained models.

Finally Figure 5.1 depicts the Precision-Recall curves of the 4 trained models on PASCAL-Part.

Figure 5.2 shows the output of Faster-LTN with knowledge on an image of the test set of PASCAL-Part dataset.

The Average Precision measured on partOf predicate is equal to **0.12**.

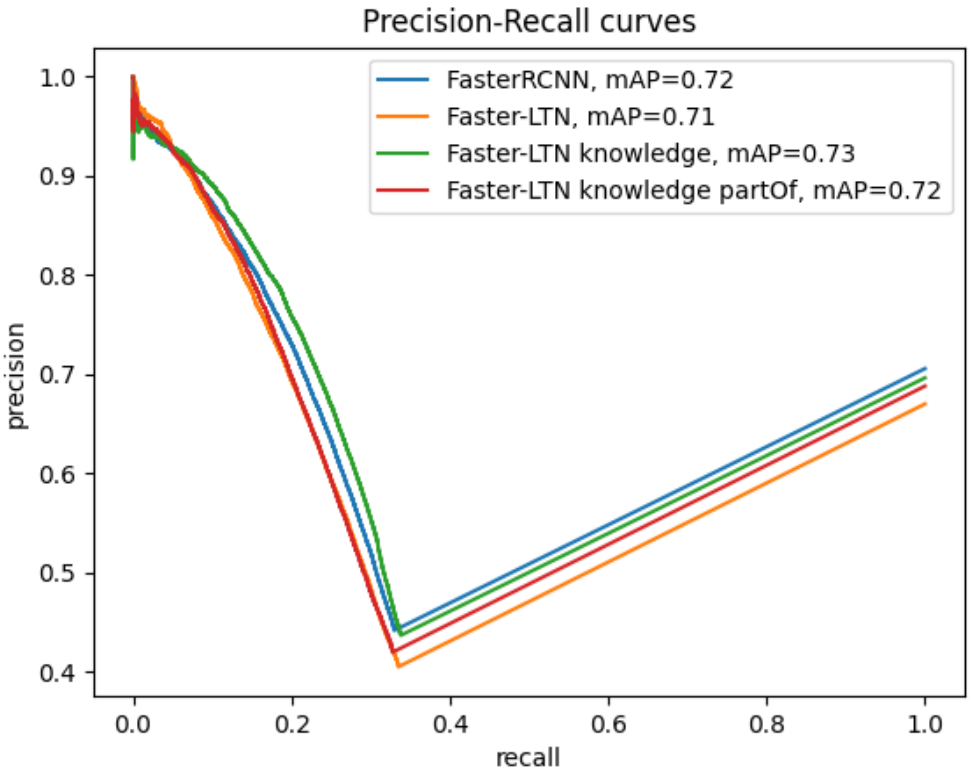


Figure 5.1. The Precision-Recall curves model configurations.

Table 5.2. Results obtained on PASCAL-Part dataset.

Class	FRCNN	Faster-LTN	Faster-LTN knowledge	Faster-LTN knowledge partOf
Aeroplane AP	0.86	**0.88**	0.87	0.87
Animal Wing AP	0.50	**0.67**	0.65	0.66
Arm AP	0.74	0.65	0.72	**0.75**
Artifact Wing AP	**0.70**	0.53	0.51	0.47
Beak AP	0.71	0.61	**0.85**	0.67
Bicycle AP	0.80	0.77	0.78	**0.84**
Bird AP	0.82	0.77	0.85	**0.86**
Boat AP	0.59	**0.67**	0.58	0.54
Body AP	0.76	0.70	**0.83**	0.70
Bodywork AP	0.73	**0.93**	0.79	0.81
Bottle AP	0.61	0.57	**0.69**	0.66
Bus AP	0.82	**0.90**	0.86	0.79
Cap AP	**0.89**	0.83	0.66	0.84
Car AP	0.74	**0.83**	0.82	0.82
Cat AP	**0.90**	0.88	0.87	0.88
Chain Wheel AP	**0.78**	0.75	0.66	0.73
Chair AP	0.57	0.51	0.49	**0.65**
Coach AP	0.49	0.60	**0.62**	0.58
Cow AP	0.74	0.76	**0.82**	0.77
Diningtable AP	0.66	0.51	**0.68**	0.59
Dog AP	0.81	**0.84**	0.79	**0.84**
Door AP	0.58	0.74	**0.77**	0.72
Ear AP	**0.80**	0.72	0.73	0.71
Ebrow AP	**0.79**	0.51	0.61	0.52
Engine AP	0.46	**0.58**	0.51	0.55
Eye AP	0.77	0.65	**0.85**	0.68
Foot AP	0.78	0.77	**0.84**	0.82
Hair AP	0.85	**0.87**	0.84	0.81
Hand AP	**0.81**	0.70	0.79	0.79
Handlebar AP	0.50	0.67	0.60	**0.68**
Head AP	0.88	0.87	**0.89**	0.86
Headlight AP	**0.70**	0.50	0.65	0.60
Hoof AP	0.72	0.80	**0.90**	0.67
Horn AP	0.74	0.72	0.78	**0.83**
Horse AP	0.77	0.87	**0.86**	0.75
Leg AP	0.60	0.59	0.58	**0.62**
License plate AP	**0.73**	0.53	0.63	0.59
Locomotive AP	0.71	**0.82**	0.78	0.77
Mirror AP	0.82	0.75	0.79	**0.87**
Motorbike AP	0.79	0.83	0.81	**0.84**
Mouth AP	**0.90**	0.68	0.80	0.72
Muzzle AP	0.81	**0.83**	0.82	0.80
Neck AP	0.68	**0.70**	0.69	0.64
Nose AP	**0.81**	0.73	0.73	0.70
Person AP	0.79	**0.83**	0.82	0.81
Plant AP	0.41	**0.47**	0.44	**0.47**
Pot AP	0.49	**0.70**	0.62	0.68
Pottedplant AP	0.59	**0.68**	0.53	0.64
Saddle AP	**0.84**	0.72	0.79	0.78
Screen AP	0.52	**0.68**	0.65	0.64
Sheep AP	0.80	**0.84**	0.83	0.81
Sofa AP	0.67	**0.76**	0.62	0.51
Stern AP	**0.83**	0.69	0.78	0.75
Tail AP	**0.67**	0.48	0.59	0.56
Torso AP	0.80	0.78	**0.82**	0.79
Train AP	0.68	0.79	**0.83**	0.76
Tvmonitor AP	0.62	0.76	0.69	**0.78**
Wheel AP	**0.74**	**0.74**	0.70	**0.74**
Window AP	0.57	0.59	**0.62**	0.61
mAP	0.72	0.71	**0.73**	0.72

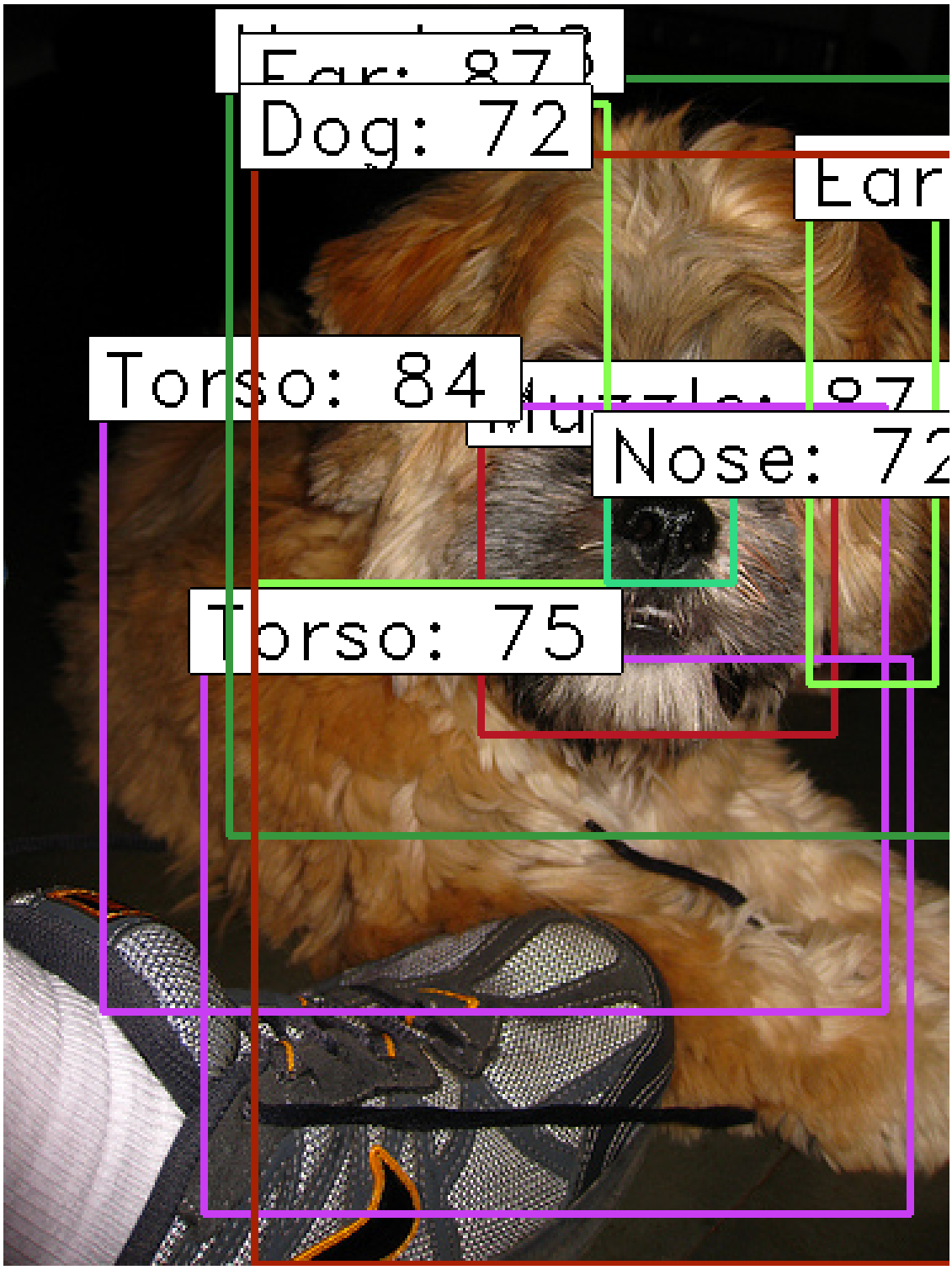


Figure 5.2. Predictions of Faster-LTN with knowledge.

Chapter 6

Conclusion

The results obtained from the Faster-LTN model show that this is competitive with the traditional Faster-RCNN model. It is able to obtain good performance on two different datasets. With some configurations it is also able to surpass the starting model.

The addition of logical knowledge significantly improves performance.

A fundamental aspect of this work is the introduction of the focal log sum function as an aggregation function of logical literals. Thanks to this innovation, it was possible to overcome the architectural problems due to integration.

The number of epochs required to train a Faster-LTN model is higher than Faster RCNN and Faster-LTN has an higher number of parameters to optimize.

However, Faster-LTN can be an interesting staring point for new models capable to solve more complex tasks. This integration strategy can be used on other object detectors. The problems to be solved are similar. Especially those related to the unbalancing classes, brilliantly solved by the focal log sum.

External knowledge can be useful in the situation in which is encountered a lack of informations in training data. In [8] has been tried LTNs with a noisy training set. The training with logical constrains has demostrated that the model is able to reach good performance also in this situation.

Moreover, starting from this idea, Faster-LTN can be applied to the Few-Shot learning.

Few-Shot learning is a topic of Machine Learning in which we want to predict something based on a few training examples. The extreme case of Few-Shot learning is called One-Shot learning, in which we want to predict something that the model has never seen before. We can compensate the lack of labelled data of a class with some logical clauses that express some logical properties of that class.

Furthermore, the fact that the model is able to handle external symbolic knowledge and to identify relationships between objects can be the starting point for defining models capable of solving complex tasks such as Visual Question Answering or Visual Relationship Detection.

Bibliography

[1] Martin Abadi et al. "TensorFlow: A system for large-scale machine learning". In: 2016, pp. 265–283.

[2] Collin F Baker, Charles J Fillmore, and John B Lowe. "The berkeley framenet project". In: *Proceedings of the 17th international conference on Computational linguistics-Volume 1*. Association for Computational Linguistics. 1998, pp. 86–90.

[3] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. "Yolov4: Optimal speed and accuracy of object detection". In: *arXiv preprint arXiv:2004.10934* (2020).

[4] Alessandro Daniele and Luciano Serafini. "Neural Networks Enhancement through Prior Logical Knowledge". In: *arXiv preprint arXiv:2009.06087* (2020).

[5] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. "ProbLog: A Probabilistic Prolog and Its Application in Link Discovery." In: *IJCAI*. Vol. 7. Hyderabad. 2007, pp. 2462–2467.

[6] Luc De Raedt et al. "From Statistical Relational to Neuro-Symbolic Artificial Intelligence". In: *arXiv preprint arXiv:2003.08316* (2020).

[7] Ivan Donadello and Luciano Serafini. "Compensating Supervision Incompleteness with Prior Knowledge in Semantic Image Interpretation". In: *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2019, pp. 1–8.

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

[9] Mark Everingham et al. "The pascal visual object classes (voc) challenge". In: *International journal of computer vision* 88.2 (2010), pp. 303–338.

[10] Christiane Fellbaum. "WordNet". In: *The encyclopedia of applied linguistics* (2012).

[11] Artur d'Avila Garcez and Luis C Lamb. "Neurosymbolic AI: The 3rd Wave". In: *arXiv preprint arXiv:2012.05876* (2020).

[12] Ross Girshick. "Fast r-cnn". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1440–1448.

[13] Ross Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.

[14] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[15] Drew Hudson and Christopher D Manning. "Learning by abstraction: The neural state machine". In: *Advances in Neural Information Processing Systems*. 2019, pp. 5901–5914.

[16] Emile van Krieken, Erman Acar, and Frank van Harmelen. "Analyzing differentiable fuzzy logic operators". In: *arXiv preprint arXiv:2002.06100* (2020).

[17] Ranjay Krishna et al. "Visual genome: Connecting language and vision using crowdsourced dense image annotations". In: *International Journal of Computer Vision* 123.1 (2017), pp. 32–73.

[18] Luis Lamb et al. "Graph Neural Networks Meet Neural-Symbolic Computing: A Survey and Perspective". In: *arXiv preprint arXiv:2003.00330* (2020).

[19] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), pp. 436–444.

[20] Guohao Li, Hang Su, and Wenwu Zhu. "Incorporating external knowledge to answer open-domain visual questions with dynamic memory networks". In: *arXiv preprint arXiv:1712.00733* (2017).

[21] Xiaodan Liang, Lisa Lee, and Eric P Xing. "Deep variation-structured reinforcement learning for visual relationship and attribute detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 848–857.

[22] Xiaodan Liang et al. "Symbolic graph reasoning meets convolutions". In: *Advances in Neural Information Processing Systems*. 2018, pp. 1853–1863.

[23] Tsung-Yi Lin et al. "Feature pyramid networks for object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 2117–2125.

[24] Tsung-Yi Lin et al. "Focal loss for dense object detection". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2980–2988.

[25] Hugo Liu and Push Singh. "ConceptNet—a practical commonsense reasoning tool-kit". In: *BT technology journal* 22.4 (2004), pp. 211–226.

[26] Wei Liu et al. "Ssd: Single shot multibox detector". In: *European conference on computer vision*. Springer. 2016, pp. 21–37.

[27] Cewu Lu et al. “Visual relationship detection with language priors”. In: *European conference on computer vision*. Springer. 2016, pp. 852–869.

[28] Pan Lu et al. “R-VQA: learning visual relation facts with semantic attention for visual question answering”. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2018, pp. 1880–1889.

[29] Robin Manhaeve et al. “Deepproblog: Neural probabilistic logic programming”. In: *Advances in Neural Information Processing Systems*. 2018, pp. 3749–3759.

[30] Kenneth Marino, Ruslan Salakhutdinov, and Abhinav Gupta. “The more you know: Using knowledge graphs for image classification”. In: *arXiv preprint arXiv:1612.04844* (2016).

[31] Joseph Redmon and Ali Farhadi. “YOLO9000: better, faster, stronger”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 7263–7271.

[32] Joseph Redmon and Ali Farhadi. “Yolov3: An incremental improvement”. In: *arXiv preprint arXiv:1804.02767* (2018).

[33] Joseph Redmon et al. “You only look once: Unified, real-time object detection”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[34] Shaoqing Ren et al. “Faster r-cnn: Towards real-time object detection with region proposal networks”. In: *Advances in neural information processing systems*. 2015, pp. 91–99.

[35] Shaoqing Ren et al. “Faster r-cnn: Towards real-time object detection with region proposal networks”. In: *Advances in neural information processing systems*. 2015, pp. 91–99.

[36] Matthew Richardson and Pedro Domingos. “Markov logic networks”. In: *Machine learning* 62.1-2 (2006), pp. 107–136.

[37] Adam Santoro et al. “A simple neural network module for relational reasoning”. In: *Advances in neural information processing systems*. 2017, pp. 4967–4976.

[38] Luciano Serafini and Artur d’Avila Garcez. “Logic tensor networks: Deep learning and logical reasoning from data and knowledge”. In: *arXiv preprint arXiv:1606.04422* (2016).

[39] Abdul Vahab et al. “Applications of Object Detection System”. In: *International Research Journal of Engineering and Technology (IRJET)* 6.4 (2019), pp. 4186–4192.

[40] Frank Van Harmelen and Annette ten Teije. "A boxology of design patterns for hybrid learning and reasoning systems". In: *arXiv preprint arXiv:1905.12389* (2019).

[41] Zhun Yang. "Extending Answer Set Programs with Neural Networks". In: *arXiv preprint arXiv:2009.10256* (2020).

[42] Kexin Yi et al. "Neural-symbolic vqa: Disentangling reasoning from vision and language understanding". In: *Advances in neural information processing systems*. 2018, pp. 1031–1042.

[43] Rowan Zellers et al. "Neural motifs: Scene graph parsing with global context". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 5831–5840.