Faster-LTN: a neuro-symbolic, end-to-end object detection architecture

Faster-LTN: a neuro-symbolic, end-to-end object detection architecture

Francesco Manigrasso[1][0000−0002−4151−8880], Filomeno Davide Miro[1], Lia Morra[1][0000−0003−2122−7178], and Fabrizio Lamberti[1][0000−0001−7703−1372]

Politecnico di Torino, Dipartimento di Automatica e Informatica, Torino, Italy
francesco.manigrasso@polito.it, filomenodavide.miro@studenti.polito.it, {lia.morra, fabrizio.lamberti}@polito.it

Abstract. The detection of semantic relationships between objects represented in an image is one of the fundamental challenges in image interpretation. Neural-Symbolic techniques, such as Logic Tensor Networks (LTNs), allow the combination of semantic knowledge representation and reasoning with the ability to efficiently learn from examples typical of neural networks. We here propose Faster-LTN, an object detector composed of a convolutional backbone and an LTN. To the best of our knowledge, this is the first attempt to combine both frameworks in an end-to-end training setting. This architecture is trained by optimizing a grounded theory which combines labelled examples with prior knowledge, in the form of logical axioms. Experimental comparisons show competitive performance with respect to the traditional Faster R-CNN architecture.

Keywords: Object detection · NeuroSymbolic AI · Convolutional Neural Network · Logic Tensor Networks.

1 Introduction

A long-standing problem in Semantic Image Interpretation (SII) and related tasks is how to combine learning from data with existing background knowledge in the form of relational knowledge or logical axioms [1]. Neural-Symbolic (NeSy) integration, which aims at integrating symbolic knowledge representation and learning with machine learning techniques [2], can provide an elegant and principled solution to augment state-of-the-art deep neural networks with these novel capabilities, increasing their performance, robustness and explainability.

The present work leverages the Logic Tensor Network (LTN) paradigm that was proposed by Serafini, Donadello and d'Avila Garcez [3,4]. In very simple terms, LTNs operate by interpreting (or *grounding*) a First-Order Logic (FOL) as functions on real vectors, which parameters can be trained via stochastic gradient descents to maximize the satisfiability of a given theory. LTNs have been successfully applied to the tasks of part-of relationship detection [3] and visual relationship detection [5]. Previous works have shown how LTNs can compensate

the lack of supervision (e.g., in few-shot learning scenarios) by relying on logical axioms derived from pre-existing knowledge bases.

To close the semantic gap between the symbolic (concept) and subsymbolic (pixel) levels, LTNs for SII rely on convolutional neural networks (CNNs) to extract semantic features which form the basis for grounding object instances in a real vector. Previous works [5,3] relied on pre-trained CNNs, which however suffer from all the limitations traditionally associated with deep learning, namely, the need for a large-scale annotated dataset for training, and lack of interpretability. To fully reap the benefits of NeSy techniques in SII, end-to-end architectures in which the LTN is jointly trained with the feature extraction CNN are needed.

In this work, we propose Faster-LTN, an object detector which unifies the Faster R-CNN object detector with a LTN-based classification head. Differently from previous works [3,5], both modules are jointly trained in an end-to-end fashion. The logical constraints imposed by the LTN can thus shape the training of the convolutional layers, that are no longer purely data-driven. To achieve this objective, we propose several modifications to the original LTN formulation to increase the architecture scalability and deal with data imbalance. Experimental results on the PASCAL VOC and PASCAL PART datasets show that Faster-LTN converges to competitive performance with respect to purely neural architectures, thus proving the feasibility of this approach. The Faster-LTN was implemented in Keras and is available at https://gitlab.com/grains2/Faster-LTN.

The rest of the paper is organized as follows. In Section 2, related work is presented. In Section 3, different variations of the Faster-LTN architecture are presented, after a brief introduction to the theory behind LTNs. Section 4 presents the experimental setting and results. Finally, conclusions are drawn.

2 Related work

A natural image is comprised of scenes, objects and parts, all interconnected by a complex network of spatial and semantic relationships. Thus, developing semantic image interpretation (SII) components requires to recognize a hierarchy of components, and entails both robust visual perception and the ability to encode and (reason about) visual relationships. Several techniques have been proposed to augment Convolutional Neural Networks (CNNs) with relationship representation and reasoning capabilities, including Relational Network [6], Graph Neural Networks [7] and Neural-Symbolic (NeSy) techniques [8,3,5]. For a more general introduction to NeSy techniques, the reader is referred to recent surveys [9,10].

Many recent approaches extract features from CNNs to a subsequent symbolic or neuro-symbolic module [11,3,5,12]. Yuke Zhu et al. [11] use a Markov Logic Network (MLN) to process text information with associated visual features; a knowledge base is used to represent relations between objects using visual, physical, and categorical attributes. Kenneth Marino et al. [13] incorporate a Graph Search Neural Network (GSNN) into a classification network. Donatello et al. [3] and Cewu Lu et al. [12] have demonstrated the use of visual features to train LTNs for visual relationship detection, in form of *subject-verb-object*

triplets or *part of* relationships. These works demonstrate how NeSy techniques enable the definition of logical axioms that serve as high-level inductive biases, driving the network to find the optimal solution that is compatible with said inductive biases. However, since in the above-mentioned cases the feature extraction and the classification networks are trained separately, the CNN cannot leverage these additional inductive biases during training.

There are, however, some practical hurdles associated with the training of NeSy architectures. Scalability, when dealing with large amounts of data, is a known issue associated with symbolic AI [14]. For this reason, many NeSy architectures rely on a conventional object detector to provide an initial list of candidate objects [3], thus disregarding the effect of the background and simplifying (i.e., reducing) the scale of the problem. In this work, we compare several strategies that are effectively capable of training a LTN-based object detector from scratch, taking into account the effect of the background and the resulting data imbalance.

Another aspect related to scalability is the choice of aggregation function and fuzzy logic operators. Emilie van Krieken et al. [14] and Samy Badreddine [4] found substantial differences between differential fuzzy logic operators in terms of computational efficiency, scalability, gradients, and ability to handle exceptions, which are important characteristics in a learning setting. Their analysis lays the groundwork for the present FasterLTN architecture, which incorporates and extends the log-product aggregator analyzed in [14].

3 The Faster-LTN architecture

This section describes the Faster-LTN architecture and training procedure in detail. An overview of the overall architecture is presented in Figure 1. We first summarize the Faster R-CNN overall architecture (Section 3.1). Then, we introduce the main concepts behind LTNs (Section 3.2) and their application to object detection (Section 3.3), referring the reader to [3,4] for additional details. Finally, the joint training procedure of Faster-LTN is explained in Section 3.4, highlighting the main changes introduced to make end-to-end training feasible.

Architecture

3.1 Faster R-CNN

Faster R-CNN is a two-stage object detector composed of a Region Proposal Network (RPN) and a classification network with a shared backbone [15]. For each anchor, the RPN generates a binary classification label (Background vs. foreground), while a regression layer computes the bounding box coordinates. Regions of Interest (ROIs) selected by the RPN are fed to an ROI Pooling layer, which extracts and resizes each proposal bounding box's features from the shared backbone. Feature maps of equal size are passed to the classifier. The classifier comprises two convolutional heads, a classification layer (with softmax

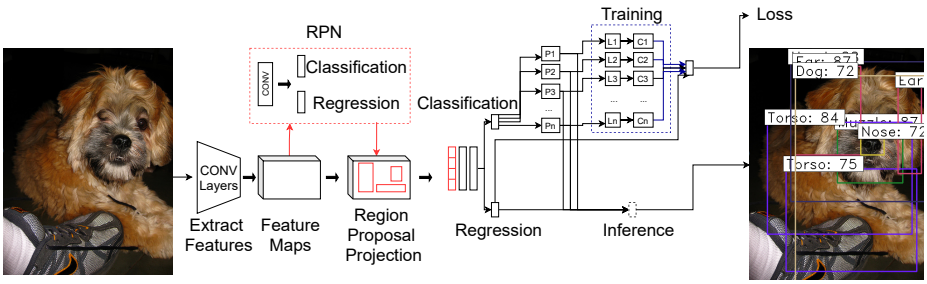

Fig. 1: Faster-LTN architecture. The first part of the architecture, up to the RPN, is the same as in the Faster R-CNN network [15]. The feature maps associated to the RPN proposals are extracted by the backbone, concatenated and passed to the LTN, which includes a collection of predicates P_i, each corresponding to a specific class. At training time, a batch of labelled examples in the training dataset are used to define a partial theory $\mathcal{T}_{expl}$. Each positive or negative example corresponds to a positive or negative literal (L) for the corresponding predicates. The truth value of the aggregated clauses (C) is maximized to find the optimal grounding $\mathcal{G}^*$. At inference time, the truth value of the predicates P_i is computed.

activation) that computes the final object classification and a regression layer (with linear activation) that computes the bounding box.

Training of the RPN and classifier heads is performed jointly in an alternating fashion. At each forward pass (corresponding to one image), the RPN is trained and updated; then, the RPN output is kept fixed, and the detector head is updated. A fixed number of positive (object) and negative (Background) examples are selected at each step to train the classifier head.

The loss is as a combination of regression and classification loss:

$$L(\{p_i\},\{b_i\}) = \frac{1}{n_c}\sum_i L_{cls}(p_i, p_i') + \lambda \frac{1}{n_r}\sum_i p_i * L_{reg}(b_i, b_i') \tag{1}$$

In the Faster-LTN, we keep the RPN module intact and substitute the classifier head with an LTN.

3.2 Logic Tensor Network

Grounding In the LTN framework, it is possible to encode a FOL language $\mathcal{L}$ by defining its interpretation domain as a subset of $\mathbb{R}^n$. In the LTN formalism, this process is called *grounding*.

Given the vector space $\mathbb{R}^n$, a grounding $\mathcal{G}$ for $\mathcal{L}$ has the following properties:

1. $\mathcal{G}(c) \in \mathbb{R}^n$, for every $c \in \mathcal{C}$;
2. $\mathcal{G}(P) \in \mathbb{R}^{n*k} \to [0,1]$, for every $p \in \mathcal{P}$

The grounding of a set of **closed terms** $t_1, .., t_m$ of $\mathcal{L}$ in an atomic formula is defined as:

$$\mathcal{G}\left(\mathcal{P}\left(t_1, ...t_m\right)\right) = \mathcal{G}\left(P\right)\left(\mathcal{G}\left(t_1\right), ..., \mathcal{G}\left(t_m\right)\right) \tag{2}$$

Formulas can be connected with fuzzy logic *operators* such as conjunctions ($\wedge$), disjunctions ($\vee$), and implications ($\implies$), including logical quantifiers ($\forall$ and $\exists$). Several real-valued, differentiable implementations are available in the fuzzy logic domain [14]. Our implementation, as in [3], is based on the Łukasiewicz [16] formulation:

$$\mathcal{G}\left(\neg\phi\right) = 1 - \mathcal{G}\left(\phi\right) \tag{3}$$

$$\mathcal{G}\left(\phi \vee \psi\right) = min(1, \mathcal{G}\left(\phi\right) + \mathcal{G}\left(\psi\right)) \tag{4}$$

Predicate symbols are interpreted as functions that map real vectors to the interval $[0, 1]$, which can be interpreted as the predicate's degree of truth. A typical example is the *is-a* predicate, which quantifies the existence of a given object. For instance, if $b = \mathcal{G}(x)$ is the grounding of a dog bounding box, than $\mathcal{G}(\mathsf{Dog})(\mathsf{v}) \simeq 1$. A logical constraint expressed in FOL allows to define its properties, i.e., $\forall x \left(\mathsf{Dog}(x) \rightarrow \mathsf{hasMuzzle}\left(x\right)\right)$.

In LTNs, predicates are typically defined as the generalization of the neural tensor network:

$$\mathcal{G}\left(\mathcal{P}\right)\left(\mathbf{v}\right) = \sigma\left(u_P^T \tanh\left(\mathbf{v_T} W_P^{[1:k]} \mathbf{v} + V_P \mathbf{v} + b_p\right)\right) \tag{5}$$

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. With this formulation, the truth value of a clause can be determined by a neural network which first computes the grounding of the *literals* (i.e., atomic objects) contained in the clause, and then combines them using fuzzy logical operators, as defined by Eqs. 3-4.

Grounded theory A Grounded Theory (GT) $\mathcal{T}$ is defined by a pair $\langle \mathcal{K}, \hat{\mathcal{G}} \rangle$, where the knowledge base $\mathcal{K}$ is a set of closed formulas, and $\hat{\mathcal{G}}$ is a partial grounding. $\mathcal{K}$ is constructed from labelled examples, as well as logical axioms, as defined in Section 3.3. In practice, a partial grounding is optimized since, qualitatively, our set $\mathcal{K}$ represents a limited and finite set of examples. A grounding G **satisfies** a GT $\langle \mathcal{K}, \hat{\mathcal{G}} \rangle$ if $\mathcal{G}$ completes $\hat{\mathcal{G}}$ and $\mathcal{G}\left(\phi\right) = 1 \; \forall \; \phi \in \mathcal{K}$.

Best satisfability problem Given a grounding $\hat{\mathcal{G}}_\theta$, where θ is the set of parameters of all predicates, the learning problem in LTNs is framed as a *best satisfability problem* which consists in determining the values of Θ^* that maximize the truth values of the conjunction of all clauses $\phi \in \mathcal{K}$:

$$\Theta^* = argmax_\Theta \hat{\mathcal{G}}_\theta \left(\bigwedge_{\phi \in \mathcal{K}} \phi\right) - \lambda ||\Theta||_2^2 \tag{6}$$

where $\lambda||\Theta||_2^2$ is a regularization term. In practical problems, it is unlikely that a grounded theory can be satisfiable in the classical sense. Hence, we opt instead to find the grounding which achieves the best possible satisfaction, while accounting for the inevitable exception to the rule. Such exceptions can easily arise in the visual domain not only to account to allow the occasional deviation from the norm, but also to account for properties that are not visible. For instance, a cat has (usually) a tail, but a few cats may be tail-less; more frequently, the tail will be occluded or cut from the image.

3.3 LTN for object detection

A grounded theory for object detection Let us consider a set of bounding boxes $b \in \mathcal{B}$ with known class $c \in \mathcal{C}$. An object with bounding box b_n is grounded by the vector:

$$\mathbf{v_{b_n}} =< \mathbf{z}_{b_n}, b_n > \tag{7}$$

Where $\mathbf{z}_{b_n} = f(I, b_n)$ is an embedding feature vector, calculated by a convolutional neural network f, given an image I and the bounding box coordinates b_n predicted by the RPN layer. This is slightly different from previous works [3], where the grounding of a bounding box was defined by the probability vector predicted by a pre-trained Faster R-CNN, and allows to effectively connect the convolutional layers and the LTN.

We set the embedding $f(I, b_n)$ to the output of the last fully connected layer of the classifier head, without softmax activation. Other choices are possible, e.g., by sum pooling the output of an earlier convolutional layer.

The *is-a* predicate for class $c \in \mathcal{C}$ is grounded by a tensor network, defined as in Eq. 5, which implements a one-vs-all classifier. It must be noticed that, differently from [3], the *is-a* predicate takes as input only the embedding features $\mathbf{z}_{b_n}$, excluding the bounding box coordinates. This allows to retain one of the basic properties of object detectors, i.e., invariance to translation.

The *part-of* predicate is defined over pairs of bounding boxes [3]. A pair of two generic bounding boxes b_m and b_l is grounded by the vector:

$$\mathbf{v_{b_{m,l}}} =< \mathbf{z}_{b_m}, b_m, \mathbf{z}_{b_l}, b_l, ir_{m,l} > \tag{8}$$

where $ir_{m,l}$ is the *containment ratio* defined as:

$$ir_{m,l} = \frac{Area\left(b_m \cap b_l\right)}{Area\left(b_m\right)} \tag{9}$$

The grounding $\mathcal{G}\left(\mathsf{part-of}\right)\left(\mathbf{v_{b_{m,l}}}\right)$ is a neural tensor network as in Eq. 5.

Defining a theory from labelled examples Let us now consider how a GT is constructed to solve the best satisfiability problem defined in Eq. 6 for object detection. As in [3], two grounded theories $\mathcal{T}_{expl}$ and $\mathcal{T}_{prior}$ are defined. The former, $\mathcal{T}_{expl}$, aggregates all the clauses derived from the labelled training

set, essentially replicating the classical learning-by-example setting. The theory $\mathcal{T}_{prior}$, on the contrary, introduces *logical* and *mereological* constraints that represent prior knowledge or, in a more general sense, desirable properties of the final solution.

In this work, two types of constraints are defined. First, we enforce *mutual exclusion* through the clause:

$$\forall x(P_1(x) \implies (\neg P_2(x) \wedge ... \wedge \neg P_n(x))) \tag{10}$$

Eq. 10 is translated into $K(K-1))/2$ clauses, corresponding to all unordered class pairs over K classes, e.g., $\mathsf{Cat}(x) \implies \neg\mathsf{Person}(x)$.

Secondly, we impose *mereological constraints* on the grounding of *part-of* and *is-a* predicates derived from an existing ontology (e.g., Wordnet) which includes *meronimy* (i.e., *part-whole*) relationships. Axioms are included to specify that a *part* cannot include another *part*, that a *whole* object cannot include another *whole* object, and that each *whole* is generally associated with a set of given *parts*. An example of such axioms is as follows:

$$\forall x, y\,(\mathsf{Cat}(x) \wedge \mathsf{partOf}(y, x) \rightarrow \mathsf{Tail}\,(y) \vee \mathsf{Head}\,(y) ... \vee \mathsf{Eye}\,(y)) \tag{11}$$

to indicate that if an object y is classified as part of x and x is a cat, than y can be only an object that we know is a part of the whole cat. Mereological constraints were enforced exploiting the KB developed in [3], to which the reader is referred for further information.

3.4 Faster-LTN

The overall architecture, illustrated in Figure 1, is an end-to-end system connecting a convolutional object detector with an LTN. Specifically, the classifier head is modified, by removing the softmax activation, and feeding the output to the LTN. At training time, a GT is constructed as defined in Section 3.4. The LTN is implemented by defining three additional layers: *Predicate*, *Literal* and *Clause* layers. For each class c, the corresponding literal computes the truth value of all positive (i.e., belonging to class c) and negative (i.e., not belonging to class c) examples. The Clause layer aggregates all literals for a given class, using the selected aggregation function. Additionally, it is possible to define clauses (e.g., for *part-of* predicates) that take as input multiple literals. For the sake of simplicity, in Figure 1 only $\mathcal{T}_{expl}$ is shown. The final loss of the LTN is given by summing L_{LTN} with the regression loss, as for the RPN layer.

Training In order to deal with memory constraints, a partial $\mathcal{T}_{expl}$ needs to be rebuilt with every batch of examples. In the original implementation [3], the LTN was trained on the predictions of a pre-trained object detector, allowing for a relatively large batch size. In our setting, the LTN is trained on all proposals extracted by the RPN, and a separate batch is constructed for each image, taking into account background as well as foreground examples. It is worth noticing

that one-vs-all classification amplifies the data imbalance between positive and negative examples for each class, even when the training batch consists of an equal number of objects and background proposals.

Aggregation function The chosen aggregator function is the log-product, which was shown in [14] to scale well with the number of inputs, and which formulation is equivalent to the cross-entropy loss. However, in our case, this choice does not weight adequately the contribution of positive examples, given the high level of class imbalance. Hence, inspired by [17], we introduce the focal log-product aggregation defined as:

$$L_{LTN} = -\sum_{j=0}^{K}\sum_{i=0}^{N} \alpha_c (1 - x_{i,j})^{\gamma} log\left(x_{i,j}\right) \tag{12}$$

where α_c is a class-dependent weight factor, γ enhances the contribution of literals with low truth value (i.e., misclassified examples), x_i is the literal of the i-th ROI in the j-th class, K is the number of classes and N is the batch size.

To set the value of α_c, we simply observe that for each training batch and each class c, the number of negative examples is given by the number of background examples (which is fixed during training), plus the positive examples that belong to other classes. Hence, we set $\alpha_c = \frac{1-\beta}{1-\beta^{pos_c}}$ and $\alpha_c = \frac{1-\beta}{1-\beta^{neg_c}}$, for positive and negative examples respectively. Let $p(c)$ be the fraction of bounding boxes in the training set belonging to class c. Then, for a given batch the percentage of positive and negative examples becomes $pos_c = \frac{N}{2} p\left(c\right)$ and $neg_c = \frac{N}{2} + \frac{N}{2}\left(1 - p\left(c\right)\right)$, respectively.

4 Experiments

4.1 Dataset

Experiments were performed on the PASCAL VOC 2010 [18] and PASCAL PART [19] benchmarks. For the latter, we selected 20 classes for whole objects and 39 classes for parts. All experiments are conducted on the trainval partition with 80:20 split. For PASCAL PART (10K images), we further experiment reducing the training set by 50% by random selection: the number of images is thus roughly 8K for PASCAL PART and 4K for PASCAL PART REDUCED.

4.2 Experimental setup

Faster R-CNN The architecture of the Faster R-CNN follows quite closely the original implementation [15]. The backbone architecture was ResNet50 pre-trained on ImageNet; the anchor scales were set to 128^2,256^2, and 512^2, with aspect ratios of 1:1, 1:2,and 2:1. The number of RPN proposals is set to 300. For training the classifier head, 128 bounding boxes were randomly selected, with a ratio of 32:96 positive and negative examples, for the PASCAL VOC dataset; for

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

Table 1: Results of the Faster R-CNN (FR-CNN), Faster R-CNN with focal loss (FR-CNN FL), and Faster-LTN (F-LTN) on PASCAL VOC.

PASCAL PART, 32 bounding boxes with 16:16 ratio. The network was trained for 100 epochs with the Adam optimizer; the learning rate was set to 10^{-5} for the first 60 epochs, and then reduced to 10^{-6}. Regularization techniques included data augmentation (horizontal flip) and weight decay (with rate 5×10^{-4}).

Faster-LTN The architecture of Faster-LTN was the same as Faster R-CNN, except for the classifier head in which the LTN was embedded.

Each predicate is defined by Eq. 5, with $k = 6$ kernels. Łukasiewicz's t–norm was chosen to encode the literals' disjunction, and the focal log-product, with $\gamma = 2$, was selected as the aggregation function. $\mathcal{T}_{prior}$ included mutual exclusion constraints for PASCAL VOC, and mutual exclusion and mereological constraints for PASCAL PART experiments. In the latter case, the LTN was expanded to include *part-of* predicates, but for the sake of comparison with Faster R-CNN, only the object detection performance was evaluated.

On the PASCAL VOC dataset, different experiments were performed with variations of the focal log-product aggregation function: with and without class weights α, and with and without adding an additional predicate *bg* to represent the background class. The experiments are denoted as Faster-LTN, Faster-LTN α, Faster-LTN *bg*, and Faster-LTN *bg*+α. Experiments on PASCAL-PART were performed with the Faster-LTN *bg* configuration. All networks were trained for 150 epochs using the Adam optimizer, with weight decay (decay rate 5×10^{-4}), random horizontal flip and L2 regularization (λ is set to 5×10^{-4}.). The learning rate was set to 10^{-5} for the first 60 epochs, and then reduced to 10^{-6}.

All experiments were performed on the HPC@Polito cluster, equipped with V100 NVIDIA GPU. The performance metric was the mean Average Precision (MAP) implemented as in the PASCAL VOC challenge 2010 [20].

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

Table 2: Comparison of Faster R-CNN and Faster-LTN (including mereological constraints) on the PASCAL PART dataset.

4.3 Results

Experiments on Pascal VOC, summarized in Table 1, show that Faster LTN achieved competitive and even superior results compared to the original Faster R-CNN architecture, with the mAP increasing from 62.6 to 73.8. In this version of the LTN, the only axiomatic constraint was the one imposing mutual exclusivity (see Eq. 11). We observed comparable performance when including the background as an additional class (mAP from 73.8 to 73.4); on the other hand, weighting positive and negative samples according to their frequency did not improve results (mAP from 73.8 to 72.1).

Qualitatively, we observed that Faster LTN was able to detect more objects than Faster R-CNN. Given that log-product aggregation is mathematically equivalent to the cross-entropy loss, and the backbone is the same, this difference can be attributed to the different classification setting (K one-vs-all classifiers instead of a single multi-class classifier) or the use of the focal loss [17]. However, when changing the loss of the Faster R-CNN classifier head to the focal loss, performance dropped from 62.6 to 59.2. Hence, we attribute Faster-LTN performance to the greater flexibility offered by a more complex classifier head, with higher number of parameters. In fairness, Faster LTN took a few more epochs to reach convergence.

In the PASCAL PART experiments, shown in Table 2, additional mereological axioms were included in $\mathcal{T}_{prior}$. This allowed to increase performance from 35.1 to 41.2; when reducing the training set size by half, the performance gap was maintained (28.5 to 32.8). The comparable quality of the learned features is further supported by the t-SNE embeddings of the extracted features, which are shown in Figure 2.

5 Conclusion and future works

The availability of large scale, high quality, labelled datasets is one of the major hurdles in the application of deep learning. A tighter integration between perception and reasoning, which is enabled by emerging Neural-Symbolic techniques, allows to complement learning by examples with the integration of axiomatic background knowledge. In this paper, we introduced the Faster-LTN architecture, an end-to-end object detector composed by a convolutional backbone and RPN (based on the Faster R-CNN architecture) and a LTN module. The detector is trained end-to-end by maximizing the satisfiability of a grounded theory combining clauses derived from labelled examples with axiomatic constraints.

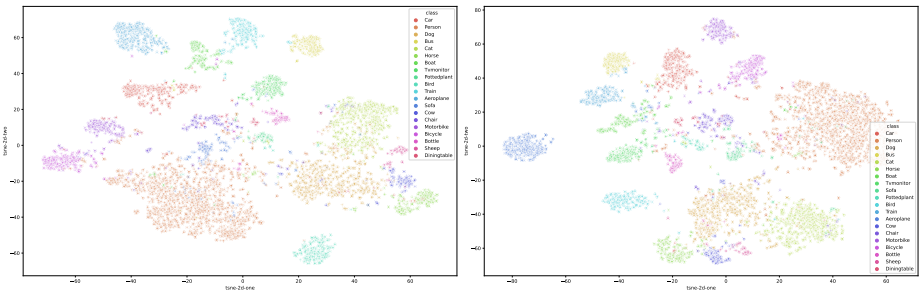

Fig. 2: Comparison of the t-SNE embeddings of the features extracted for the *whole* objects classes in the test test. Features extracted from Faster R-CNN (left) and Faster-LTN with axiomatic constraints (right).

Our goal was to establish the feasibility of this approach, and indeed the results, albeit preliminary, prove that Faster-LTN is competitive or can even outperform the baseline Faster R-CNN. However, the scalability of this approach to larger training sets and other object detector (e.g., single-stage detectors) should be further investigated. Through the Faster-LTN model, available at https://gitlab.com/grains2/Faster-LTN, we aim to provide a baseline architecture on which new experiments and applications can be built. Future work will investigate how high-level symbolic constraints can shape the learning process, increasing robustness in the presence of noise and dataset bias.

Acknolewdgement

The authors wish to thank Ivan Donadello for the helpful discussions. Computational resources were in part provided by HPC@POLITO, a project of Academic Computing at Politecnico di Torino (http://www.hpc.polito.it).

References

1. Aditya, S., Yang, Y., Baral, C.: Integrating knowledge and reasoning in image understanding. In: Proceedings of the 28th International Joint Conference on Artificial Intelligence, IJCAI 2019. pp. 6252–6259. International Joint Conferences on Artificial Intelligence (2019)
2. Raedt, L.d., Dumančić, S., Manhaeve, R., Marra, G.: From statistical relational to neuro-symbolic artificial intelligence. In: Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20. pp. 4943–4950 (2020)
3. Donadello, I., Serafini, L., Garcez, A.D.: Logic tensor networks for semantic image interpretation. In: Proceedings of the 26th International Joint Conference on Artificial Intelligence. p. 1596–1602. AAAI Press (2017)
4. Badreddine, S., Garcez, A.d., Serafini, L., Spranger, M.: Logic tensor networks. ArXiv **abs/2012.13635** (2020)

5. Donadello, I., Serafini, L.: Compensating supervision incompleteness with prior knowledge in semantic image interpretation. In: 2019 International Joint Conference on Neural Networks (IJCNN). pp. 1–8 (2019)
6. Shanahan, M., Nikiforou, K., Creswell, A., Kaplanis, C., Barrett, D., Garnelo, M.: An explicitly relational neural network architecture. In: Proceedings of the 37th International Conference on Machine Learning. vol. 119, pp. 8593–8603. PMLR (2020)
7. Lamb, L.C., Garcez, A.d., Gori, M., Prates, M.O., Avelar, P.H., Vardi, M.Y.: Graph neural networks meet neural-symbolic computing: A survey and perspective. In: Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20. pp. 4877–4884 (2020)
8. Yi, K., Wu, J., Gan, C., Torralba, A., Kohli, P., Tenenbaum, J.B.: Neural-symbolic vqa: Disentangling reasoning from vision and language understanding. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems. p. 1039–1050. Curran Associates Inc. (2018)
9. Besold, T.R., Garcez, A., Bader, S., Bowman, H., Domingos, P.M., Hitzler, P., Kühnberger, K.U., Lamb, L., Lowd, D., Lima, P., Penning, L., Pinkas, G., Poon, H., Zaverucha, G.: Neural-symbolic learning and reasoning: A survey and interpretation. ArXiv **abs/1711.03902** (2017)
10. Garcez, A., Gori, M., Lamb, L., Serafini, L., Spranger, M., Tran, S.: Neural-symbolic computing: An effective methodology for principled integration of machine learning and reasoning. FLAP **6**, 611–632 (2019)
11. Zhu, Y., Fathi, A., Fei-Fei, L.: Reasoning about object affordances in a knowledge base representation. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) European conference on computer vision – ECCV 2014. pp. 408–424 (2014)
12. Lu, C., Krishna, R., Bernstein, M., Fei-Fei, L.: Visual relationship detection with language priors. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) European conference on computer vision – ECCV 2016. pp. 852–869. Cham (2016)
13. Marino, K., Salakhutdinov, R., Gupta, A.: The more you know: Using knowledge graphs for image classification. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 20–28 (2017)
14. van Krieken, E., Acar, E., Harmelen, F.V.: Analyzing differentiable fuzzy logic operators. ArXiv **abs/2002.06100** (2020)
15. Ren, S., He, K., Girshick, R., Sun, J.: Faster r-cnn: Towards real-time object detection with region proposal networks. IEEE Transactions on Pattern Analysis and Machine Intelligence **39**(6), 1137–1149 (2017)
16. Dutta, S., Basu, S., Chakraborty, M.K.: Many-valued logics, fuzzy logics and graded consequence: A comparative appraisal. In: Logic and Its Applications. pp. 197–209 (2013)
17. Lin, T.Y., Goyal, P., Girshick, R., He, K., Dollár, P.: Focal loss for dense object detection. In: 2017 IEEE International Conference on Computer Vision (ICCV). pp. 2999–3007 (2017)
18. Everingham, M., Van Gool, L., Williams, C.K.I., Winn, J., Zisserman, A.: The PASCAL Visual Object Classes Challenge 2010 (VOC2010) Results
19. Chen, X., Mottaghi, R., Liu, X., Fidler, S., Urtasun, R., Yuille, A.: Detect what you can: Detecting and representing objects using holistic models and body parts. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1971–1978 (2014)
20. Cartucho, J., Ventura, R., Veloso, M.: Robust object recognition through symbiotic deep learning in mobile robots. In: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 2336–2341 (2018)