The Workshops of the Thirty-Second
AAAI Conference on Artificial Intelligence

Image Classification Using Deep Learning and Prior Knowledge

Soumali Roychowdhury
Computer Science
IMT School of Advanced Studies
Lucca, Italy

Michelangelo Diligenti, Marco Gori
Information Engineering and Mathematical Sciences
University of Siena
Siena, Italy

Abstract

Deep learning has been very successful on image classification tasks in the past few years, because it allows to develop end-to-end solutions, taking as input the raw images in form of a grid of pixels and returning the class assignments. Semantic Based Regularization is used in this paper as a general and novel way to integrate prior knowledge into deep learning. Semantic Based Regularization takes as input the prior knowledge, expressed as a collection of first-order logic clauses (FOL), where each task to be learned corresponds to a predicate in the knowledge base. Then, it translates the knowledge into a set of constraints which can be either integrated into the learning process or used in a collective classification step during the test phase. The integration of the domain knowledge during the train or test phase is realized via the same backpropagation schema that runs over the expression trees of the grounded FOL clauses. The methodology can be applied on top of any learner and the experimental results on CIFAR-10 show how the integration of the prior knowledge boosts the accuracy of many different deep architectures.

Introduction

The advances on Deep Convolutional Neural Networks (CNNs) (Bengio 2009), [(LeCun 2015), (Hinton 2012) started a new line of research that has been outperforming other architectures on the most popular benchmarks on image classification, starting from AlexNet (Krizhevsky, Sutskever, and Hinton 2012) and later improved architectures like VGG (Szegedy et al. 2016a), Network In Network (Lin, Chen, and Yan 2013), Inception (Szegedy et al. 2015), (Szegedy et al. 2016b). Some of these architectures rely on the success of Residual networks (ResNet) (Kaiming et al. 2015), which have shown how to further increase the depth of the networks with no loss of accuracy.

The classification of images using deep CNNs relies on the collection of a large number of supervised training examples to avoid over-fitting and poor classification accuracy. Unfortunately, the collection of supervised data is complex and highly labour intensive.

An alternative way to effectively train complex networks with limited data is to enforce some kind of structure in the network architecture: this structure encodes any available domain prior knowledge in the network without relying on massive amount of data to extrapolate it. For example, explicitly inserting important feature crosses without relying on the network to implicitly develop them has shown some gains in classification tasks (Ruoxi et al. 2017; Cheng et al. 2016). Beside simple tricks at the feature representation level, deep learning is generally used off-the-shelf without relying on any general and flexible way for incorporating prior knowledge like relationships among the data instances, or enforcing consistency among the predicted classes in a multi-label environment.

In the field of Statistical Relational Learning, injecting symbolic knowledge into sub-symbolic learning via neural networks has been previously explored. Markov Logic Networks (Richardson and Domingos 2006; Wang and Domingos 2008) and Probabilistic Soft Logic (Broecheler, Mihalkova, and Getoor 2010) have received a lot of attention: these attempts do not provide a very tight integration between the logic and the deep learner layers, which are only stacked as input to the probabilistic logic layer. More recently, a direct integration of FOL logic rules into the training of neural networks has been proposed in (Hu et al. 2016; Serafini, Donadello, and d'Avila Garcez 2017). However, the proposed solution is still limited in terms of the kind of knowledge that can be integrated. Furthermore, the knowledge can be injected only at the training time but it can not be enforced during classification.

This paper is based on the Semantic Based Regularization (SBR) (Diligenti et al. 2012), (Diligenti, Gori, and Saccà 2015) framework, which integrates the ability to learn from examples and logic rules. The learning task in SBR is formulated as a multi objective optimization problem where a set of constraints must be satisfied along with the traditional regularization term. The constraints translate First-order Logic formulas which expresses the relationships on the patterns (i.e the relationships among the attributes of data instances or supervised examples) as well as general prior knowledge from the environment. SBR has the unique ability to enforce the constraints even during the test time, which makes it flexible to perform collective classification (Jensen, Neville, and Gallagher 2004a; Getoor and Taskar 2007). This paper proposes a backpropagation schema, running across the expression trees of the

FOL formulas of the knowledge base, which can be elegantly applied during training or during the test phase with no modification.

In the experimental section, the expressive power of the FOL knowledge is used inject consistency constraints into an image classification task. In particular, this paper uses the hierarchical structure of the classes of the CIFAR-10 dataset to enforce the consistency of the predictions obtained from various deep CNN architectures. The integration of logic knowledge improves the results on the CIFAR-10 dataset for a wide range of underlying deep learning architectures with a negligible increase in the computational cost.

The main contributions of the paper are the following:

- the definition of a general backpropagation schema that allows to inject any prior knowledge expressed via FOL into deep learning.
- Improving classification accuracy for some state-of-the-art models using the proposed collective classification mechanism.

Learning with Constraints

Semantic Based Regularization (Diligenti et al. 2012; Diligenti, Gori, and Saccà 2015) learns a set of T functions $\boldsymbol{f} = \{f_1, \ldots, f_T\}$ which are correlated by a set of H functionals $\Phi_h(\boldsymbol{f}), 0 \leq \Phi_h(\boldsymbol{f}) \leq 1,\ h = 1, \ldots, H$ describing the prior knowledge about how the functions should behave. These functionals can express some property of the functions like correlations, bounds and so on, therefore limiting the parameter space where good solutions can be found.

Let's assume that the j-th function is associated to a sample $\mathcal{X}_j$ of patterns input to the function, Each pattern is then represented via a vector of real-valued features. The set of examples $\mathcal{E}_j \subset \mathcal{X}_j$ is also provided to express the desired function outputs over a subset of the available patterns.

Generally speaking, different functions can share the same sample of input patterns (e.g. $\mathcal{X}_j = \mathcal{X}_i\ i \neq j$). The framework does not impose any limit to the arity of the predicates, which can also express relationships, even if the following description will be limited to unary predicates to keep the notation simple. When dealing with n-ary relations, the pattern representations associated to these functions can be obtained as the Cartesian product of a set of finite domains: $\mathcal{X}_j = \mathcal{X}_{j1} \times \mathcal{X}_{j2} \times \ldots$.

The vector of values obtained by applying the function f_k to the set of patterns $\mathcal{X}_k$ is indicated as $f_k(\mathcal{X}_k)$, while $\boldsymbol{f}(\mathcal{X}) = f_1(\mathcal{X}_1) \cup f_2(\mathcal{X}_2) \cup \ldots$ collects the groundings for all functions.

Using the classical constrained optimization approach, the functionals can be integrated into learning by transforming them into constraints enforced by penalizing their violation on the sample of data together with another term which forces the fitting of the supervised data for each function and a regularization term:

$$\begin{aligned} C_e[\boldsymbol{f}(\mathcal{X})] &= \sum_{k=1}^{T} \Big(\overbrace{||f_k||^2}^{Reg} + \\ &+ \overbrace{\lambda_l \sum_{x \in \mathcal{E}_k} L(f_k(x), y_k(x))}^{Labeled} \Big) + \\ &+ \overbrace{\sum_{h=1}^{H} \lambda_h L_c\left(\Phi_h\big(\boldsymbol{f}(\mathcal{X})\big)\right)}^{Logic} \qquad (1) \end{aligned}$$

where $\mathcal{E}_k$ is the set of labelled data available for the k-th function, $L(\cdot,\cdot)$ is a loss function, $y_k(x)$ is the target output value for the x pattern for task k, λ_l is the weight for the labelled portion of the cost function, $L_c(\cdot)$ is the loss function used for the constraint part and λ_h is the weight for the h-th constraint.

In the following section we will discuss how to define the functionals/constraints and optimize the cost function.

Constraints and Logic

Let us assume that a First Order Logic (FOL) knowledge base (KB) is given, whose predicates are either fully known a priori (*given*) for all possible groundings or are approximated via functions that are being learned. The KB is converted into a form suitable for optimization by a variation of fuzzy generalizations of FOL, which have been first proposed by Novak (Novák 1987). Fuzzy FOL uses tnorms to compute the degree of satisfaction of the rule for a given grounding of the variables. A degree of satisfaction of the FOL formula is obtained by iteratively grounding the variables and the aggregating the values using the average and maximum operations over the obtained values for the universal and existential quantifiers, respectively.

Grounded Expressions Any grounded FOL rule is an expression in propositional logic. Therefore, we start studying the conversion of a propositional logic expression into a continuous and differentiable constraint. A *t-norm* is a function $t : [0, 1] \times [0, 1] \to [0, 1]$, which is continuous, commutative, associative, monotone, and featuring a neutral element 1 (i.e. $t(a, 1) = a$). A *t-norm fuzzy logic* is defined by its t-norm $t(a_1, a_2)$ that models the logical AND. The other logic operations can be derived from the tnorm, therefore allowing to build a continuous function for any propositional logic expression.

One expression tree is built for each considered grounded FOL rule, where the basic logic operations ($\neg, \wedge, \vee, \Rightarrow$) are replaced by a unit computing the logic operation specific to the used tnorm. Once the output values of the grounded operands are computed, the expression tree recursively computes the output values of all the nodes. The value obtained on the root node is the result of the evaluation of the grounded expression.

Table 1 details the operations computed by the units in the forward step given inputs for different selections of the

Table 1: The operations performed by the single units of an expression tree depending on the left and right inputs x, y and the tnorm used.

op \ tnorm	Product	Minimum	Lukasiewicz
$x \wedge y$	$x \cdot y$	$\min(x, y)$	$\max(0, 1-x-y)$
$x \vee y$	$x+y-x \cdot y$	$\max(x, y)$	$\min(1, x+y)$
$\neg x$	$1-x$	$1-x$	$1-x$
$x \Rightarrow y$	$\min(1, \frac{y}{x})$	$x<y?1:y$	$x<y?1:y-x$

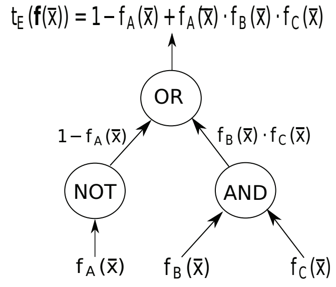


Figure 1: Forward propagation of the values in the expression tree of a FOL formula for the grounding $x = \bar{x}$ when using the product tnorm. The output of the root node returns a value in $[0, 1]$ corresponding to the evaluation t_E of the rule for the given grounding.

tnorm. For example consider the rule $\forall x[\neg A(x)] \vee [B(x) \wedge C(x)]$. For any given grounding, the expression tree returns the output value: $t_E(\boldsymbol{f}(x)) = 1 - f_A(x) + f_A(x) \cdot f_B(x) \cdot f_C(x)$. Figure 1 shows the expression tree and the computation that is performed for the previous FOL rule grounded with $x = \bar{x}$.

Quantifiers We focus on FOL formulas in the Prenex Normal Form form (quantifiers are placed at the beginning of the expression) as any FOL formula can be equivalently written in this form. The degree of truth of a formula containing an expression E with a universally quantified variable x_i is the average of the tnorm generalization $t_E(\cdot)$, when grounding x_i over $\mathcal{X}_i$: $\Phi(\boldsymbol{f}(\mathcal{X})) = \frac{1}{|\mathcal{X}_i|} \sum_{x_i \in \mathcal{X}_i} t_E\big(\boldsymbol{f}([x, \mathcal{X}/\mathcal{X}_i])\big)$ In the previous example, this would yield the following functional:

$$\begin{aligned}\Phi(\boldsymbol{f}(\mathcal{X})) &= \sum_{x \in \mathcal{X}} t_E\big(\boldsymbol{f}(x)\big) = \\ &= \sum_{x \in \mathcal{X}} 1 - f_A(x) + f_A(x) \cdot f_B(x) \cdot f_C(x)\end{aligned}$$

For the existential quantifier, the truth degree is instead defined as the *maximum* of the tnorm expression over the domain of the quantified variable. When multiple universally or existentially quantified variables are present, the conversion is recursively performed from the outer to the inner variables. When only universal quantifiers are present in a formula, the aggregation reduces to the overall average over each grounding $\boldsymbol{x}$:

$$\Phi(\boldsymbol{f}(\mathcal{X})) = \frac{1}{|\mathcal{X}|} \sum_{\boldsymbol{x} \in \mathcal{X}} t_E\big(\boldsymbol{f}(\boldsymbol{x})\big)$$

Backpropagation with Logic Constraints

Equation 1 can be optimized via gradient descent, where the derivative of the cost function with respect to the j-th weight of the i-th function w_{ij} is:

$$\begin{aligned}\frac{\partial C_e}{\partial w_{ij}} &= \sum_{x \in \mathcal{E}_i} \frac{\partial L(f_i(x), y_k(x))}{\partial f_i} \cdot \frac{\partial f_i}{\partial w_{ij}} + \\ &+ \sum_k \frac{\partial C_e}{\partial L_c} \cdot \frac{\partial L_c}{\partial \Phi_k} \cdot \frac{\partial \Phi_k}{\partial f_i} \cdot \frac{\partial f_i}{\partial w_{ij}} . \qquad (2)\end{aligned}$$

where the regularization term has been omitted to keep the notation simple. Assuming that the $f_i(\cdot)$ are implemented by a neural network, the first term corresponds to the classical labelled error which can be minimized via backpropagation.

Since any tnorm guarantees that Φ_k is in $[0, 1]$, choosing $L_c(\cdot) = L^1(1, \cdot) = 1 - \cdot$ yields $\frac{\partial C_e}{\partial L_c} = -1$. Finally, $\frac{\partial f_i}{\partial w_{ij}}$ can also be computed via standard back-propagation step using the underlying neural network.

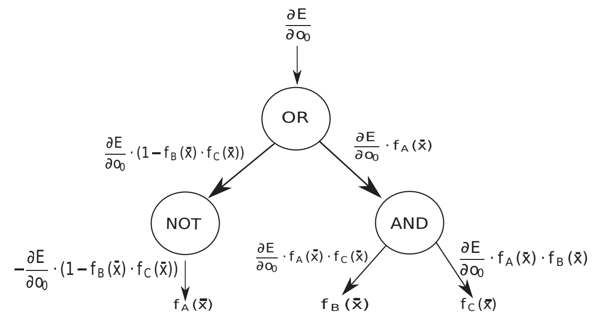
This section shows how to efficiently compute the $\frac{\partial \Phi_k}{\partial f_i}$. To kept notation simple, we will discuss only rules with universally quantified variables but the same ideas can be trivially extended to existentially quantified variables. A universally quantified rule expresses the fact that all the groundings should respect the grounded FOL formula. Backpropagation is performed over the expression trees built for the selected tnorm and each single grounding, yielding:

$$\frac{\partial \Phi_k}{\partial f_i} = \frac{1}{|\mathcal{X}|} \sum_{\boldsymbol{x} \in \mathcal{X}} \frac{\partial t_E\left(\boldsymbol{f}(\boldsymbol{x})\right)}{\partial f_i}$$

It can be noticed that $\frac{\partial t_E(\boldsymbol{f}(\boldsymbol{x}))}{\partial f_i}$ can be computed by backpropagation over the expression tree built for the propositional expression E in the FOL formula. In particular, indicating as o_n the output of node n of the expression tree, and assuming that the root node is node 0 for which its output $o_0 = t_E\left(\boldsymbol{f}(\boldsymbol{x})\right)$, then the derivative is recursively propagated backward using the chain rule over the expression tree:

$$\frac{\partial t_E(\boldsymbol{f}(\boldsymbol{x}))}{\partial o_n} = \frac{\partial t_E(\boldsymbol{f}(\boldsymbol{x}))}{\partial o_{p(n)}} \cdot \frac{\partial o_{p(n)}}{\partial o_n}$$

where $p(n)$ indicates the parent of node n in the tree and $\frac{\partial t_E(\boldsymbol{f}(\boldsymbol{x}))}{\partial o_0} = 1$. For any node n, the derivative $\frac{\partial o_{p(n)}}{\partial o_n}$



∂E/∂o_0
OR
∂E/∂o_0 · (1 − f_B(x̄) · f_C(x̄))
∂E/∂o_0 · f_A(x̄)
NOT
AND
−∂E/∂o_0 · (1 − f_B(x̄) · f_C(x̄))
∂E/∂o_0 · f_A(x̄) · f_C(x̄)
∂E/∂o_0 · f_A(x̄) · f_B(x̄)
f_A(x̄)
f_B(x̄)
f_C(x̄)

Figure 2: The backpropagation of the values over the expression tree for the grounding $x = \bar{x}$ of the FOL rule $\forall x[\neg f_A(x)] \vee [f_B(x) \wedge f_C(x)]$ using the product tnorm. The backpropagated derivative reaches a leaf node and it is passed down to trigger a further backpropagation pass over the network which implements the function in the leaf node.

is determined by the used tnorm and the logic operation $tnorm(n)$ computed by the unit, such that

$$\frac{\partial o_{p(n)}}{\partial o_n} = tnorm(p(n))'_{\{l,r\}}$$

where the subscripts $\{l, r\}$ indicate whether node n is the left or right child of $p(n)$. This is needed because the inputs to the logic operations may be not symmetric. The negation is an unary operation and requires no subscripts. Table 2 details how the gradients are back-propagated depending on the specific operation and considered tnorm.

This establishes an efficient gradient computation schema over the expression tree, where the error of the considered FOL constraint is back-propagated from the root to the leaves. Figure 2 shows the backpropagation of the error for the example used across this paper $\forall x[\neg f_A(x)] \vee [f_B(x) \wedge f_C(x)]$. At the bottom of the expression tree, the backpropagated error reaches a leaf node, and it triggers a further backpropagation pass over the network implementing the function stored in that node.

Collective Classification

Collective classification (CC) (Sen et al. 2008) is one of the most important tasks studied by Statistical Relational Learning (Jensen, Neville, and Gallagher 2004b), (Neville and Jensen 2003), (Jensen, Neville, and Gallagher 2004a), (Taskar, Segal, and Koller 2001). Collective classification assumes that the test patterns are not independent, therefore their classification should not be independently carried out like assumed in standard machine learning tasks. CC performs inference over a set of instances that are connected by one or more kind of relationships. In this paper, collective classifications is used to refine the neural network outputs to be consistent with the available FOL knowledge.

In particular, let $f_k(\mathcal{X}'_k)$ indicate the vector of values obtained by evaluating the function f_k over the data points of the test set $\mathcal{X}'_k$. The set of vectors is indicated as: $\boldsymbol{f}(\mathcal{X}') = f_1(\mathcal{X}'_1) \cup \ldots \cup f_T(\mathcal{X}'_T)$.

Collective classification searches for the values $\bar{\boldsymbol{f}}(\mathcal{X}'_k) = \bar{f}_1(\mathcal{X}'_1) \cup \ldots \cup \bar{f}_T(\mathcal{X}'_T)$ respecting the FOL formulas on the

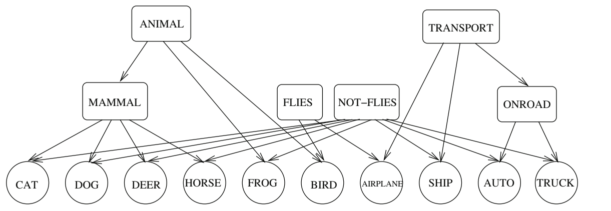


Figure 3: The taxonomy showing the hyperonymy relations among the predicates of the classes in the CIFAR-10 dataset. The final target classes in the benchmark are the leaf classes in the tree.

test data, while being close to the prior values established by the networks:

$$\begin{aligned} C_{cc}[\bar{\boldsymbol{f}}(\mathcal{X}'), \boldsymbol{f}(\mathcal{X}')] \quad &= \quad \frac{1}{2}\sum_{k=1}^{T} |\bar{f}_k(\mathcal{X}'_k) - f_k(\mathcal{X}'_k)|^2 \ + \\ &+ \quad \sum_h L_c\left(\Phi_h\big(\bar{\boldsymbol{f}}(\mathcal{X}')\big)\right) \end{aligned} \tag{3}$$

The learning schema proposed in the previous section natively generalizes to the collective classification case. The gradient computation over the expression tree is completely preserved, the only difference is that the gradient is propagated only up the function outputs during collective classification, whereas it is back propagated through the network in case of training with constraints. In particular, no backpropagation down to the model weights is performed: $\frac{\partial f_i}{\partial w_{ij}}$ is dropped from equation 2 during collective classification.

Experimental Results

The experimental analysis is based on the CIFAR-10 dataset (Krizhevsky and Hinton 2009)[1], which is composed of 60000 images, of which 50000 and 10000 ones form the training and test datasets, respectively. Each image is stored as RGB with a size of 32×32. Each image is assigned to one of 10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Using WordNet[2] and additional common knowledge, these classes have been recursively mapped into their hyperonymy (e.g. more general concept) obtaining the taxonomy shown in Figure 3. In all our experiments we trained the models using the training set and obtained the classification outputs from the test set.

CIFAR-10 has been used as one of the main dataset to measure the improvements of deep architectures on the image classification task. In particular, we compared against the following CNN deep architectures:

- Network In Network (NIN): this model was proposed in (Lin, Chen, and Yan 2013) and it is based on a CNN architecture where the linear filters of the convolutional layer have been replaced by a small neural network to abstract the data within the receptive field.

[1]https://www.cs.toronto.edu/~kriz/cifar.html
[2]https://wordnet.princeton.edu

Table 2: The derivatives used in the back-propagation step depending on the Tnorm used and the operation implemented by the unit.

tnorm / $tnorm'_{op}$	Product	Minimum	Lukasiewicz
$(x \wedge y)'_l = \frac{\partial(x \wedge y)}{\partial x}$	y	$y < x?1:0$	$x+y<1?-1:0$
$(x \wedge y)'_r = \frac{\partial(x \wedge y)}{\partial y}$	x	$x < y?1:0$	$x+y<1?-1:0$
$(x \vee y)'_l = \frac{\partial(x \vee y)}{\partial x}$	$1-y$	$y > x?1:0$	$x+y<1?1:0$
$(x \vee y)'_r = \frac{\partial(x \vee y)}{\partial y}$	$1-x$	$x > y?1:0$	$x+y<1?1:0$
$(\neg x)' = \frac{\partial(\neg x)}{\partial x}$	-1	-1	-1
$(x \Rightarrow y)'_l = \frac{\partial(x \Rightarrow y)}{\partial x}$	$x < y?0:-\frac{y}{2 \cdot x^2}$	0	$y > x?y:1$
$(x \Rightarrow y)'_r = \frac{(\partial x \Rightarrow y)}{\partial y}$	$x < y?0:-\frac{1}{x}$	$x < y?0:1$	$x > y?x:-1$

Table 3: Prior Knowledge injected into the classification process for the CIFAR-10 dataset.

$\forall x$ ANIMAL(x) $\vee$ TRANSPORT(x)
$\forall x$ TRANSPORT(x) $\Rightarrow$ AIRPLANE(x) $\vee$ SHIP(x) $\vee$ ONROAD(x)
$\forall x$ ONROAD(x) $\Rightarrow$ AUTOMOBILE(x) $\vee$ TRUCK(x)
$\forall x$ AIRPLANE(x) $\Rightarrow$ TRANSPORT(x)
$\forall x$ SHIP(x) $\Rightarrow$ TRANSPORT(x)
$\forall x$ ONROAD(x) $\Rightarrow$ TRANSPORT(x)
$\forall x$ AUTOMOBILE(x) $\Rightarrow$ ONROAD(x)
$\forall x$ TRUCK(x) $\Rightarrow$ ONROAD(x)
$\forall x$ ANIMAL(x) $\Rightarrow$ BIRD(x) $\vee$ FROG(x) $\vee$ MAMMAL(x)
$\forall x$ BIRD(x) $\Rightarrow$ ANIMAL(x)
$\forall x$ FROG(x) $\Rightarrow$ ANIMAL(x)
$\forall x$ MAMMAL(x) $\Rightarrow$ CAT(x) $\vee$ DEER(x) $\vee$ DOG(x) $\vee$ HORSE(x)
$\forall x$ CAT(x) $\Rightarrow$ MAMMAL(x)
$\forall x$ DOG(x) $\Rightarrow$ MAMMAL(x)
$\forall x$ DEER(x) $\Rightarrow$ MAMMAL(x)
$\forall x$ HORSE(x) $\Rightarrow$ MAMMAL(x)(x)
$\forall x$ OTHERLEAVES(x) $\Rightarrow$ BIRD(x) $\vee$ FROG(x) $\vee$ SHIP(x) AIRPLANE(x)
$\forall x$ BIRD(x) $\Rightarrow$ OTHERLEAVES(x)
$\forall x$ FROG(x) $\Rightarrow$ OTHERLEAVES(x)
$\forall x$ SHIP(x) $\Rightarrow$ OTHERLEAVES(x)
$\forall x$ AIRPLANE(x) $\Rightarrow$ OTHERLEAVES(x)
$\forall x$ FLIES(x) $\Rightarrow$ BIRD(x) $\vee$ AIRPLANE(x)
$\forall x$ NOTFLIES(x) $\Rightarrow$ CAT(x) $\vee$ DOG(x) $\vee$ HORSE(x) $\vee$ DEER(x) $\vee$ TRUCK(x)
$\vee$ SHIP(x) $\vee$ AUTOMOBILE(x) $\vee$ FROG(x)
$\forall x$ CAT(x) $\Rightarrow$ NOTFLIES(x)
$\forall x$ DOG(x) $\Rightarrow$ NOTFLIES(x)
$\forall x$ HORSE(x) $\Rightarrow$ NOTFLIES(x)
$\forall x$ DEER(x) $\Rightarrow$ NOTFLIES(x)
$\forall x$ TRUCK(x) $\Rightarrow$ NOTFLIES(x)
$\forall x$ SHIP(x) $\Rightarrow$ NOTFLIES(x)
$\forall x$ AUTOMOBILE(x) $\Rightarrow$ NOTFLIES(x)
$\forall x$ FROG(x) $\Rightarrow$ NOTFLIES(x)
$\forall x$ BIRD(x) $\Rightarrow$ FLIES(x)
$\forall x$ AIRPLANE(x) $\Rightarrow$ FLIES(x)

- Resnet and PreResnet: residual networks (Kaiming et al. 2015) feature the higher level layers to compute and learn residual functions (e.g. deltas) with respect of the previous layer output. Pre-activated Resnets (He et al. 2016) further improve the resnet behaviour by analysing the propagation formulations behind the residual building blocks.
- Pyramidal Resnet Models (Han, Kim, and Kim 2016) where the feature map dimension resembles a pyramid in Pyramidal Resnets.

Training Procedure. The benefits of injection of prior knowledge into deep learning have been tested against the baseline results obtained from the different CNN models. The used knowledge base has 7 extra predicates added to the predicates for the output classes as shown in Figure 3, and each network was modified to predict the output for all classes. This was done by adding additional output layers using a softmax activation, one layer predicts $Animal$ or $Transport$, one layer Fly or $NotFly$, one layer for $Onroad$, $Mammal$ or $Others$ as a bin class collecting all other categories. In order to exactly reproduce the baseline results, the baseline models have been trained from scratch using their publicly available implementations without no change in their architecture and training procedure. Once the baseline model is trained, its weights are frozen and only the additional output layers for the new 7 predicates are trained in a second step.

The models are trained in mini batches of size 128 and the training time required by each network architecture is recorded in Table 5 for each of the models.

The testset classification error rates over the 10 final classes for each of the deep CNN models is shown in the Table 5. The classification outputs initialize a collective classification step where the proposed methodology adjusts the class assignments in order to respect the constraints. We experimented with different T-Norms to convert the knowledge: Lukaseiwicz, Weak-Lukaseiwicz (Giannini et al. 2017), Minimum and Product T-Norms. Collective classification is performed using gradient descent for 400 iterations and an initial learning rate equal to 0.01.

Performance Evaluation. Table 4 reports the error obtained when running collective classification on top of the results obtained with the single architectures using different tnorms. The Weak Lukaseiwicz tnorm is the best performer in almost all configurations, this could be partially explained by the theoretical results shown in (Giannini et al. 2017).

Table 5 reports a comparison of the results obtained with the single CNN models and the proposed approach using collective classification with the Weak Lukaseiwicz tnorm. In particular, the classification error rates obtained from the baseline models on CIFAR-10 are compared against the error rates obtained after collective classification. Collective classification brings a remarkable improvement over the individual classification in several cases. Another advantage of the proposed methodology is the negligible cost of the collective classification step compared to initial cost of training the network. Indeed, the collective classification training can be completed in less than 6 minutes for the entire

Table 4: Comparison of the Error rate for the 10 final classes on CIFAR-10 dataset using collective classification in combination with different Tnorms.

CNN Model	# of Layers	CC Error Weak Lukaseiwicz TNorm	CC Error Minimum TNorm	CC Error Product TNorm
NIN	50	**8.71**	8.78	8.81
Resnet	20	**8.01**	8.24	8.75
Resnet	32	**7.09**	7.29	7.50
Resnet	44	**6.58**	6.8	7.10
Resnet	56	**6.39**	6.61	7.00
Resnet	110	5.96	6.0	6.60
Resnet	164	**5.76**	5.90	5.90
Resnet	1202	**5.17**	5.42	6.88
Pre-Activated Resnet	110	**6.15**	6.31	6.35
Pre-Activated Resnet	164	**5.20**	5.42	5.46
Pre-Activated Resnet	1202	**6.05**	6.30	6.24
AdditivePyramid Net, α = 84	110	**4.15**	4.27	4.27
AdditivePyramid Net, α = 270	164	3.44	3.46	3.48
AdditivePyramid Net, α = 200	272	3.30	3.30	3.30

Table 5: Error rate for the 10 final classes on CIFAR-10 dataset using collective classification over the outputs from different deep CNNs and comparing between usage and non-usage of prior knowledge.

CNN Model	# of Layers	# of Parameters (%)	Train Time (Hours)	Error	Error after CC (%)
Human Performance	-	-	$>$ 100	6	-
NIN	50	1.01M	1.39	8.81	8.71
Resnet	20	0.27 M	0.27	8.75	**8.01**
Resnet	32	0.46 M	0.48	7.51	**7.09**
Resnet	44	0.66 M	0.67	7.17	**6.58**
Resnet	56	0.85 M	0.83	6.97	**6.39**
Resnet	110	1.7 M	1.38	6.61	**5.96**
Resnet	164	1.7M	1.39	5.93	**5.76**
Resnet	1202	19.4 M	20.83	7.93	**5.17**
Pre-Activated Resnet	110	1.7 M	1.39	6.37	**6.15**
Pre-Activated Resnet	164	1.7 M	1.39	5.46	**5.20**
Pre-Activated Resnet	1202	19.4 M	20.83	6.85	**6.05**
AdditivePyramid Net, α = 84	110	3.8 M	3.56	4.27	**4.15**
AdditivePyramid Net, α = 270	164	27.0 M	29.78	3.48	3.44
AdditivePyramid Net, α = 200	272	26.0 M	30	3.31	3.30

test set of CIFAR-10 on the same machine used to train the CNN architectures. The results for all ResNet architectures show remarkable improvements like the Resnet-1202 gets an error reduction larger than 2.7% (from 7.93 to 5.17) and PreResnet-1202 error is reduced from 6.85 to 6.05. The improvement for for the best PyramidNetwork configuration is not statistically significant, probably because of the lower initial error. The improvements given by the methodology and the low additional cost seem to suggest that the injection of prior knowledge could allow to use a simpler model (greatly reducing the training time) and then letting the collective classification step to improve the results at the level which could be achievable using a larger and slower-to-train model.

Conclusions and Future Work

This paper presents a novel framework to inject prior knowledge into image classification problems like CIFAR-10. The methodology can be applied on top of any deep learning architecture used to process the raw images. The full expressive power of FOL is available, and the framework does not add any limitation in the kind of knowledge that can be integrated. The presented results show remarkable improvements with respect to several tested architectures. This shows that it is possible to train simpler models and then cover some of the gap with more complex ones by instantiating a lightweight collective inference on top of the obtained class assignments. As future work, we plan to extend these results to CIFAR-100 and larger datasets, where the complexity of the benchmark allows to formulate extensive domain knowledge.

References

Bengio, Y. 2009. Learning deep architectures for ai. *Found. Trends Mach. Learn.* 2(1):1–127.

Broecheler, M.; Mihalkova, L.; and Getoor, L. 2010. Probabilistic similarity logic. In *Proceedings of the Twenty-Sixth*

Conference on Uncertainty in Artificial Intelligence (UAI), 73–82.

Cheng, H.-T.; Koc, L.; Harmsen, J.; Shaked, T.; Chandra, T.; Aradhye, H.; Anderson, G.; Corrado, G.; Chai, W.; Ispir, M.; et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, 7–10. ACM.

Diligenti, M.; Gori, M.; Maggini, M.; and Rigutini, L. 2012. Bridging logic and kernel machines. *Machine learning* 86(1):57–88.

Diligenti, M.; Gori, M.; and Saccà, C. 2015. Semantic-based regularization for learning and inference. *Artificial Intelligence*.

Getoor, L., and Taskar, B. 2007. *Introduction to statistical relational learning*.

Giannini, F.; Diligenti, M.; Gori, M.; and Maggini, M. 2017. Learning lukasiewicz logic fragments by quadratic programming. In *Proceedings of the European COnference on Machine Learning (ECML)*.

Han, D.; Kim, J.; and Kim, J. 2016. Deep pyramidal residual networks. *arXiv preprint arXiv:1610.02915*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, 630–645. Springer.

Hinton, G. E. 2012. *A Practical Guide to Training Restricted Boltzmann Machines*. Berlin, Heidelberg: Springer Berlin Heidelberg. 599–619.

Hu, Z.; Ma, X.; Liu, Z.; Hovy, E. H.; and Xing, E. P. 2016. Harnessing deep neural networks with logic rules. *CoRR* abs/1603.06318.

Jensen, D.; Neville, J.; and Gallagher, B. 2004a. Why collective inference improves relational classification. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, 593–598. ACM.

Jensen, D.; Neville, J.; and Gallagher, B. 2004b. Why collective inference improves relational classification. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 593–598.

Kaiming, H.; Xiangyu, Z.; Shaoqing, R.; ; and Jian, S. 2015. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*.

Krizhevsky, A., and Hinton, G. 2009. Learning multiple layers of features from tiny images.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In Pereira, F.; Burges, C. J. C.; Bottou, L.; and Weinberger, K. Q., eds., *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc. 1097–1105.

LeCun, Y. 2015. Deep learning. *Nature* 521.

Lin, M.; Chen, Q.; and Yan, S. 2013. Network in network. *arXiv preprint arXiv:1312.4400*.

Neville, J., and Jensen, D. 2003. Collective classification with relational dependency networks. In *Proceedings of the Second International Workshop on Multi-Relational Data Mining*, 77–91.

Novák, V. 1987. First-order fuzzy logic. *Studia Logica* 46(1):87–109.

Richardson, M., and Domingos, P. 2006. Markov logic networks. *Mach. Learn.* 62(1-2):107–136.

Ruoxi, W.; Bin, F.; Gang, F.; and Mingliang, W. 2017. Deep & cross network for ad click predictions. In *Proceedings of the ADKDD workshop*.

Sen, P.; Namata, G.; Bilgic, M.; Getoor, L.; Galligher, B.; and Eliassi-Rad, T. 2008. Collective classification in network data. *AI magazine* 29(3):93.

Serafini, L.; Donadello, I.; and d'Avila Garcez, A. S. 2017. Learning and reasoning in logic tensor networks: theory and application to semantic image interpretation. In *Proceedings of the Symposium on Applied Computing, SAC 2017, Marrakech, Morocco, April 3-7, 2017*, 125–130.

Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; and Rabinovich, A. 2015. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1–9.

Szegedy, C.; Vanhoucke, V.; Ioffe, S.; Shlens, J.; and Wojna, Z. 2016a. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2818–2826.

Szegedy, C.; Vanhoucke, V.; Ioffe, S.; Shlens, J.; and Wojna, Z. 2016b. Rethinking the inception architecture for computer vision. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Taskar, B.; Segal, E.; and Koller, D. 2001. Probabilistic classification and clustering in relational data. In *IJCAI International Joint Conference on Artificial Intelligence*, 870–876.

Wang, J., and Domingos, P. 2008. Hybrid markov logic networks. In *Proceedings of the 23-rd AAAI Conference on Artificial Intelligence*, 1106–1111.