UCLouvain

École polytechnique de Louvain

Combining Deep Learning and Logical Constraints on ROAD-R

Author: **Charles Packo**
Supervisor: **Siegfried Nijssen**
Readers: **Lucile Dierckx, Sébastien Jodogne**
Academic year 2023–2024
Master [120] in Computer Science and Engineering

Abstract

Deep neural networks (DNNs) have recently become increasingly popular due to their impressive performance in various tasks. However, their “black box" nature can lead to unexpected behaviors, contradicting task-specific background knowledge. To address this, researchers introduced ROAD-R, a dataset containing videos for road event detection in the context of autonomous driving, coupled with a set of requirements expressed as logical constraints. It was shown that exploiting these constraints enables the improvement of deep learning models' performance while ensuring compliance with the requirements. This master's thesis then has two objectives. Firstly, we investigate the feasibility of improving a pre-trained model's performance by fine-tuning it using the set of requirements, without the need for a complete re-training in order to save time in the process. Second, we want to improve post-processing methods that enforce the constraints to the predictions. Subsequent experiments involved training baseline models, fine-tuning them, and applying our post-processings to them using constrained loss and output techniques from the ROAD-R paper. Our results showed that while fine-tuning pre-trained models using the set of constraints did not provide significant improvements, our proposed post-processing methods, based on the model's confidence in its predictions, consistently outperformed the ones presented in the ROAD-R paper. Nonetheless, a more in-depth study of fine-tuning will have to be carried out in future work, as well as other uncovered aspects.

Keywords Deep learning, convolutional neural networks, logic, requirements, logical constraints, object detection, ROAD-R.

Acknowledgements

The completion of this master's thesis was made possible thanks to the help of several people, whom I would like to thank here.

First of all, I would like to express my most sincere gratitude to Prof. Siegfried Nijssen for supervising my work. His support, guidance, advice and expertise helped me enormously. I also greatly appreciated his availability throughout the year.

Many thanks also to Lucile Dierckx for her availability, advice and expertise, and for reviewing my work.

I would also like to thank Prof. Sébastien Jodogne for accepting to read my work and be part of my jury.

Of course, I would also like to thank my family and friends for always supporting me, from the beginning of my academic journey to the present day. Special thanks to Aurélien Baudart for voluntarily offering to proofread my work.

Finally, computational resources have been provided by the Consortium des Équipements de Calcul Intensif (CÉCI), funded by the Fonds de la Recherche Scientifique de Belgique (F.R.S.-FNRS) under Grant No. 2.5020.11 and by the Walloon Region.

Preliminary Note

Two AI-based tools were used to help write this work: ChatGPT by OpenAI[1] and Grammarly by Grammarly Inc[2]. Both tools were only used to correct spelling and grammar mistakes and improve readability. They were not used to generate any explanation and are therefore not responsible for the content of this work.

[1] https://chatgpt.com/
[2] https://app.grammarly.com/

Contents

Chapter 1

Introduction

In recent years, deep neural networks (DNNs) have become more and more popular for all kinds of tasks, from simple ones like classification or regression of tabular data to more complex ones like text and speech recognition and understanding, or even like object detection. Their popularity is explained by the fact that they are very powerful at extracting features in big amounts of data [21], leading them to produce impressive results, never achieved before.

However, DNNs are often described as "black box" models, meaning that while they can produce accurate predictions, it is very difficult to understand how they actually arrive at their conclusions, unlike simpler models like rule-based ones for which you can follow their logic step by step. Indeed, the way a DNN processes the data is hidden in many layers of complex calculations between thousands, or even millions of weights. This can lead to unexpected behaviors, where DNNs contradict known requirements that express background knowledge, which in turn can lead to serious consequences, particularly in applications where safety is a critical matter, such as autonomous driving.

To address this problem, researchers introduced the ROad event Awareness Dataset with logical Requirements, abbreviated as ROAD-R. This dataset is composed of videos for autonomous driving, on which we can perform object detection and multi-label classification, and is equipped with a set of requirements expressed as logical constraints [13]. In the associated paper, the researchers showed that it is possible to exploit them to improve the performance of models and to guarantee that their predictions are compliant with the requirements themselves. They then hosted the (now closed) ROAD-R 2023 Challenge[1], in which participants were asked to develop the best possible models exploiting the requirements.

[1] `https://sites.google.com/view/road-r/home`

In the context of this challenge, but without taking part in it, the objectives of this work are twofold. Firstly, we want to see if it is possible to improve the performance of an existing trained model by using the requirements, but without completely re-training the model in order to save time. Indeed, training such models is very time-consuming, and if we see that it is possible to improve them in a short amount of time thanks to a set of requirements, this could be transferred to other applications where models sometimes take weeks or even months to be trained. Secondly, we will try to improve the process of making the predictions of a model compliant with the requirements, which has already been introduced in the ROAD-R paper.

This master's thesis is structured as follows. Chapter 2 contains the theoretical background needed to understand this work, introducing and explaining the concepts of logic and deep learning. Chapter 3 then describes the ROAD-R Challenge framework in detail. It first gives a more formal definition of the multi-label classification problem with requirements. It also describes ROAD-R in more detail and explains how to perform road-events detection on it. Next, it explains how requirements are exploited to improve base models, and it ends by presenting some state-of-the-art methods to achieve that, which participants to the challenge used. After that, chapter 4 presents our proposed contributions to achieve the objectives set out above, and chapter 5 presents the experiments we carried out as well as their results, which are of course analyzed and discussed. Finally, chapter 6 concludes this work by summarizing the results obtained, the limitations and difficulties encountered, and by providing leads for future work.

Chapter 2

Theoretical Background

We will start by explaining the concepts needed to understand this work. The two main topics covered in the context of the ROAD-R challenge are logic and deep learning. Of course, as these are vast topics, we will stay on the surface and only focus on what is needed for this work.

2.1 Logic

In general, logic is the study of reasoning, through sentences representing knowledge. It is used in several domains, like mathematics or computer science for example. There are multiple types of logic, each defined by a syntax, expressing the allowed structures of sentences, and by semantics, expressing the meaning of the sentences. Here, we will describe propositional logic and fuzzy logic.

2.1.1 Propositional Logic

Propositional logic is the simplest type of logic, where sentences are propositions that may either be *true* or *false*. We will first describe its syntax and semantics before discussing the satisfiability of sentences and their conjunctive normal form. This section was inspired by [37].

Syntax

There are three types of symbols in propositional logic: first, the logical constants, *true* and *false*, the two only values that a proposition can take; second, propositional symbols, representing propositions and usually written with a capital letter or a word beginning with a capital letter; third, parentheses and logical connectives, acting as operators between propositional symbols. The five most used logical connectives are the negation ($\neg$, also called "not"), the conjunction ($\wedge$, also called "and"),

the disjunction ($\vee$, also called "or"), the implication ($\Rightarrow$), and the equivalence ($\Leftrightarrow$). Sentences are then built using logical connectives and parentheses between propositional symbols and logical constants. For example, $\neg A \vee (B \wedge C)$ is a valid sentence, where A, B, and C can be any proposition, like "the water is hot".

Semantics

Table 2.1 below shows the truth table of the five most common logical connectives, that allows to determine the truth value (*true* or *false*) of a sentence, given the truth values of its propositional symbols.

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
$true$	$true$	$false$	$true$	$true$	$true$	$true$
$true$	$false$	$false$	$false$	$true$	$false$	$false$
$false$	$true$	$true$	$false$	$true$	$true$	$false$
$false$	$false$	$true$	$false$	$false$	$true$	$true$

Table 2.1: truth table for the most common logical connectives.

There is also an operator precedence: $(), \neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$ (from highest to lowest precedence). Using the same example as above, $\neg A \vee (B \wedge C)$ is *true* if and only if A is *false*, or B and C are *true*.

Moreover, we say that two sentences are logically equivalent if they have the same truth value for each combination of truth values of their propositional symbols. We denote logical equivalence between sentences by the symbol $\equiv$. Even though the two concepts are intrinsically related, logical equivalence between sentences is different from the equivalence connective ($\Leftrightarrow$), as it acts between sentences and not between propositional symbols. Here are some known logical equivalences:

- Commutativity of $\wedge$ (idem for $\vee$): $P \wedge Q \equiv Q \wedge P$.
- Associativity of $\wedge$ (idem for $\vee$): $P \wedge Q \wedge R \equiv (P \wedge Q) \wedge R \equiv P \wedge (Q \wedge R)$.
- De Morgan's law (idem by inverting $\wedge$ and $\vee$): $\neg(P \wedge Q) \equiv \neg P \vee \neg Q$.
- Implication elimination: $P \Rightarrow Q \equiv \neg P \vee Q$.

Still using the same example, we can write: $\neg A \vee (B \wedge C) \equiv A \Rightarrow (B \wedge C)$.

Satisfiability

A sentence is satisfiable if it is true for (or satisfied by) at least one assignment of truth values for its propositional symbols. For example, $\neg A \vee (B \wedge C)$ is satisfiable, as it is satisfied by the assignment $\{A = \mathit{false}, B = \mathit{true}, C = \mathit{false}\}$. A simple example of an unsatisfiable sentence is $A \wedge \neg A$, as it will always be *false* no matter the value of A.

The problem of verifying the satisfiability of a sentence is called the SAT problem. It can be solved by enumerating all the possible assignments, but it is very costly as the number of possible assignments is exponential. Actually, the SAT problem is NP-complete, meaning that we don't know any algorithm able to solve all instances of the problem in polynomial time. However, there exist SAT solvers that are efficient and usable for relatively large instances, like MiniSat [10] and Chaff [32].

Conjunctive Normal Form

Before explaining what the conjunctive normal (CNF) form is, we have to introduce literals and clauses. A literal is simply a propositional symbol or its negation, and a clause is a disjunction of literals. Then, a sentence is in CNF if it is expressed as a conjunction of clauses. Thanks to some standard equivalence rules, every sentence can be converted into an equivalent CNF, which is the form used by most SAT solvers. For example, by distributing $\vee$ over $\wedge$, $\neg A \vee (B \wedge C)$ is equivalent to the $(\neg A \vee B) \wedge (\neg A \vee C)$ CNF.

2.1.2 Fuzzy Logic

In fuzzy logic, the truth values are no longer limited to *true* or *false* but are real numbers. Consequently, logical connectives in fuzzy logic are real functions on real numbers. Working with real values instead of binary values allows to introduce the notion of vagueness, i.e. a proposition may be *true* or *false* to a certain degree. There are several forms of fuzzy logic, but here we will only deal with the t-norm-based one, where the truth values are comprised in the real unit interval $[0, 1]$, with 0 corresponding to *false* and 1 corresponding to *true*. The syntax of such logic is essentially the same as for propositional logic, with propositional symbols that can be linked with connectives and parentheses. The semantics, however, are different and use triangular norms (t-norms) as a base for connectives. This section was inspired by [28].

T-norm

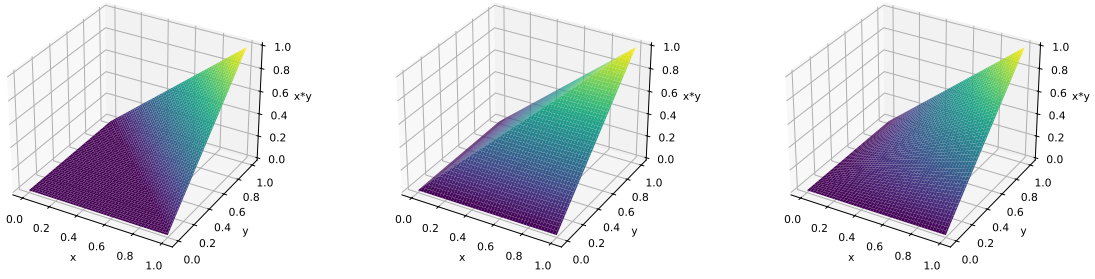
A t-norm is a function $* : [0, 1]^2 \to [0, 1]$ that is:

- commutative: $x * y = y * x \quad \forall x, y \in [0, 1]$,
- associative: $(x * y) * z = x * (y * z) \quad \forall x, y, z \in [0, 1]$,
- monotonic: $x \leq y \Rightarrow x * z \leq y * z \quad \forall x, y, z \in [0, 1]$,

and that admits 1 as neutral element, i.e. $1 * x = x * 1 = x$, $\forall x \in [0, 1]$. From this last property and the monotonicity one, we can easily derive that 0 is an absorbing element, i.e. $0 * x = x * 0 = 0$, $\forall x \in [0, 1]$. Hence, with all these properties, we can see a t-norm as the fuzzy logic equivalent of the conjunction in propositional logic. There are many t-norms, that can be continuous or not, but we will only describe the three fundamental ones, that are continuous:

- Łukasiewicz t-norm: $x *_{\text{Ł}} y \triangleq \max(0, x + y - 1) \quad \forall x, y \in [0, 1]$.
- Gödel t-norm: $x *_G y \triangleq \min(x, y) \quad \forall x, y \in [0, 1]$.
- Product t-norm: $x *_\Pi y \triangleq x \cdot y \quad \forall x, y \in [0, 1]$.

These functions can be visualized in figure 2.1 below. Although they have different shapes, we can see that their values are high when both x and y are high, and low when either x or y is low, which is the same behavior of a conjunction in propositional logic.



(a) Łukasiewicz t-norm. (b) Gödel t-norm. (c) Product t-norm.

Figure 2.1: The three fundamental t-norms.

T-conorm

A t-conorm is a function $\circ : [0, 1]^2 \to [0, 1]$ that also has the properties of commutativity, associativity, and monotonicity, but that differs from the t-norm by admitting 0 as neutral element instead of 1, i.e. $0 \circ x = x \circ 0 = x$, $\forall x \in [0, 1]$. Combining this with the monotonicity property, we can now derive that 1 is an absorbing element, i.e. $1 \circ x = x \circ 1 = 1$, $\forall x \in [0, 1]$. Hence, we can see a t-conorm

as the fuzzy logic equivalent of the disjunction in propositional logic.

In the latter, we have the following equivalence:

$$A \vee B \Leftrightarrow \neg(\neg A \wedge \neg B)$$

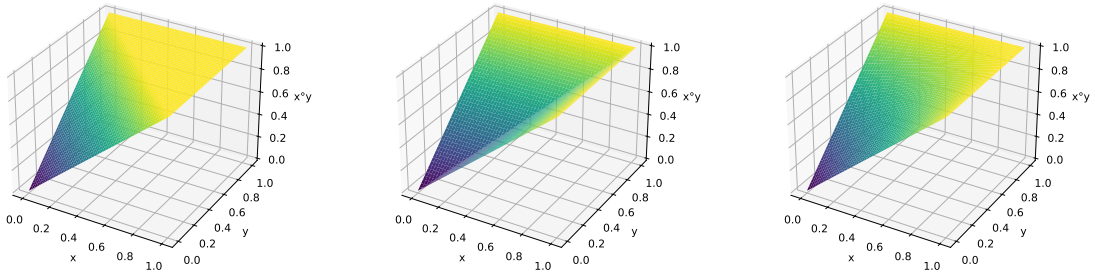
and in fuzzy logic, we can naturally define the negation as $\neg x \triangleq 1 - x$, $\forall x \in [0, 1]$. Combining these two formulas, we can now derive a t-conorm from its corresponding t-norm:

$$x \circ y = 1 - ((1 - x) * (1 - y)) .$$

Applying this last formula to the three fundamental t-norms, we finally obtain their corresponding fundamental t-conorms:

- Bounded sum: $x \circ_{\text{Ł}} y \triangleq \min(1, x + y) \quad \forall x, y \in [0, 1]$.
- Maximum: $x \circ_{G} y \triangleq \max(x, y) \quad \forall x, y \in [0, 1]$.
- Probabilistic sum: $x \circ_{\Pi} y \triangleq x + y - x \cdot y \quad \forall x, y \in [0, 1]$.

These functions can be visualized in figure 2.2 below. Although they once again have different shapes, we can see that their values are high when either x or y is high, and low when both x and y are low, which is the same behavior as a disjunction in propositional logic.



(a) Bounded sum. (b) Maximum. (c) Probabilistic sum.

Figure 2.2: The three fundamental t-conorms.

2.2 Deep Learning

Deep Learning (DL) is a subset of Machine Learning (ML), which is itself a field of Artificial Intelligence (AI). While the latter refers to the simulation of human intelligence processed by machines in general, ML focuses on algorithms that

learn and make predictions from data in order to perform tasks for which they have not been explicitly programmed. Then, DL is a subset of ML using artificial neural networks with many layers (see section 2.2.4), which is why it is called "deep".

There are many types of ML (and thus DL) paradigms, we will however focus on supervised learning, which is the one of interest for this work. In this type of ML, a model is trained on labeled data, i.e. each training example has a label and the goal for the model is to learn a mapping from data to labels so that it can predict the label of new, unseen data. There are two kinds of tasks: classification tasks where the labels are discrete and represent classes, and regression tasks where the labels are continuous values. Two other well-known ML paradigms are unsupervised learning where the goal is to train a model able to extract knowledge from unlabeled data, and reinforcement learning where an agent learns to make decisions in an environment to maximize its reward, by performing certain actions and receiving feedback.

While classical supervised ML algorithms use "handcrafted" features of the data to learn and make predictions, DL algorithms are able to extract multiple levels of features by themselves to form a hierarchical and hidden representation of the data. This makes DL particularly efficient in many tasks, like object recognition, outperforming many other ML algorithms. [33]

In the following sections, we will first introduce the general training process of a supervised ML algorithm and its performance evaluation, before presenting different types of neural networks that will be useful to understand for this work.

2.2.1 General ML Training Process

Mathematically speaking, in supervised learning, we estimate a function $f : X \to Y$, where X is the space of input data, and Y is the space of output labels [16]. This function is unknown and often very complex, which makes it nearly impossible to find exactly. However, thanks to a finite training set of input data $(\mathbf{X}, \mathbf{y}) = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$ for which the labels are known, it is possible to produce an estimate $\hat{f} : X \to Y$ of f by minimizing a function measuring the discrepancy between the model's predicted labels on training data and their true labels. Such a function is called a loss function, and we can use the gradient descent algorithm to update the model's parameters so that the loss function decreases towards its minimum.

Loss Function

As said just before, a loss function $l : Y^2 \to \mathbb{R}$ measures the discrepancy between the model's predicted labels $\hat{y}_i = \hat{f}(\mathbf{x}_i)$ on training data and their true labels $y_i = f(\mathbf{x}_i)$. Actually, it is not the loss function itself that we want to minimize. Let θ be the set of parameters of the model $\hat{f}_\theta$, we then want to minimize the expected loss $L(\theta)$ instead, that can be estimated by:

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n} l(\hat{f}_\theta(\mathbf{x}_i), y_i) = \frac{1}{n}\sum_{i=1}^{n} L_i(\theta) \tag{2.1}$$

where $L_i(\theta)$ is the loss component associated with the i^{th} data point [27]. For practical reasons, we will also call $L(\theta)$ a loss function, without loss of meaning.

Those functions have several desirable properties, including convexity and differentiability. A loss function is convex if it admits a unique global minimum, and is differentiable if its derivative with respect to the model parameters exists and is continuous. Differentiability allows the use of the gradient descent algorithm (see next section) while convexity makes the latter more efficient [42]. Moreover, most loss functions are non-negative, ensuring that they give a meaningful measure of the discrepancy between the predicted labels and the true labels. Indeed, a high loss indicates significant errors, while a low loss indicates close alignment with the actual values, 0 being the perfect match.

A well-known example of such a loss function for a regression task is the Mean Squared Error (MSE), simply measuring the square of the difference between the predicted value and the ground truth:

$$l_{MSE}(\hat{y}_i, y_i) = (\hat{y}_i - y_i)^2.$$

It is indeed convex, differentiable, and non-negative. Then, a well-known example of such loss function for a classification task is the Binary Cross-Entropy (BCE), measuring the dissimilarity between the predicted probability of a class and its true class label:

$$l_{BCE}(\hat{y}_i, y_i) = -(y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)).$$

Note that BCE loss is used for binary classification tasks, so $y_i \in \{0, 1\}$ corresponds to a class, and $\hat{y}_i \in [0, 1]$ corresponds to a probability. It is also convex, differentiable, and non-negative.

Gradient Descent Algorithm

The gradient descent algorithm allows to iteratively update the parameters θ of a model so that the loss function decreases until there is convergence to a minimum. It works by moving step by step the parameters in the opposite direction of the gradient of the loss with respect to the parameters, indicating the direction of the steepest ascent of the loss function. This naturally makes the loss decrease, if the steps are not too large, until reaching a minimum. The gradient descent algorithm is thus the following:

1. Pick some intial parameters θ_0 and a sequence of stepsizes $\{\eta_0, \eta_1, \dots, \eta_k, \dots\}$.
2. Repeat until convergence: $\theta_{k+1} \leftarrow \theta_k - \eta_k \nabla_\theta L(\theta_k)$.

Convergence is reached when the values of the parameters "no longer change", i.e. when their differences in values between two successive iterations are below a certain threshold. We can also set a maximum number of iterations at which the algorithm automatically stops. The stepsizes η_k are also called learning rates because they determine the "speed" at which the parameters are moving. In theory, they must satisfy some conditions to guarantee convergence. However, most of the time in practice, we use a constant learning rate and it gives good results: $\eta_0 = \eta_1 = \cdots = \eta_k = \cdots = \eta$. We must be careful while tuning this parameter: if it is too small, the convergence will be too slow, but if it is too high, there could be no convergence at all. Figure 2.3 below illustrates the gradient descent algorithm for the hypothetic loss function $L(\theta) = (\theta - 3)^2$, with θ a one-dimensional parameter. We used $\theta_0 = 0$ as the initial value, $\eta = 0.1$ as the learning rate, and we performed the algorithm for 25 iterations. We can see that the parameter effectively converges towards the value that minimizes the loss function.

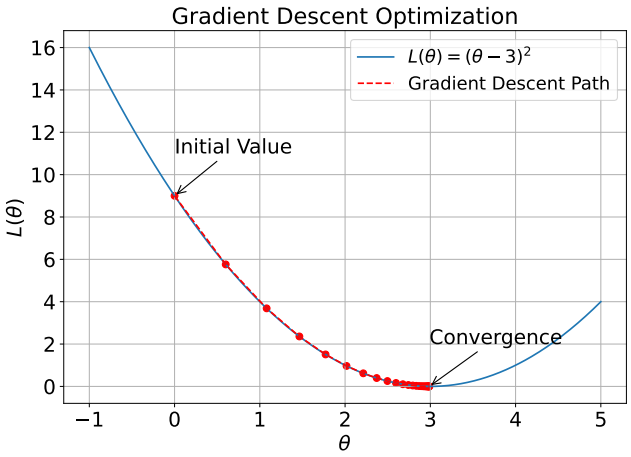


Figure 2.3: Illustration of the gradient descent algorithm.

The issue with this algorithm is that it requires to compute the gradient of the loss function at each iteration. As its computational cost is proportional to the number of data points, it can become very expensive for some models, like deep neural networks which have a lot of parameters. To overcome this issue, we use the stochastic gradient descent algorithm (SGD) in which we estimate the gradient by computing it over a mini-batch of training data points.

2.2.2 Performance Evaluation

When a model is trained, we naturally have to assess its performance. As the goal of a model is to generalize well on new unseen data, we evaluate its performance on a set of labeled data that was not part of the training. It is called the test set. There are different metrics, depending on whether it is a classification or regression model.

Regression

A common choice for regression tasks is the Root Mean Squared Error (RMSE):

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}.$$

It is simply the square root of the average of the squared differences between the predictions of the test data and their true labels. There are also many other metrics for regression, which often have in common that they are based on the differences between the true and predicted labels.

Classification

For a binary classification task with classes 1 (positive) and 0 (negative), we have to introduce the confusion matrix categorizing the predictions on the test set among 4 types:

- True positives (TP): the true and predicted labels are positive.
- False positives (FP): the true label is negative but the predicted label is positive.
- False negatives (FN): the true label is positive but the predicted label is negative.
- True negatives (TN): the true and predicted labels are negative.

The representation of the confusion matrix in figure 2.4 below summarizes well these 4 categories.

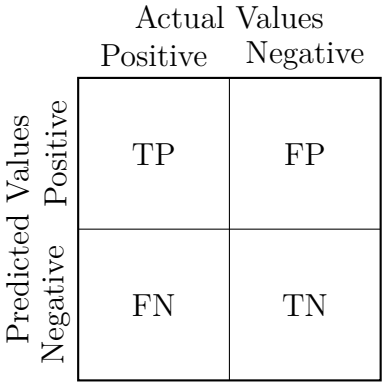


Figure 2.4: Confusion matrix.

From this confusion matrix, we can compute common performance metrics such as accuracy, precision, and recall:

- Accuracy: it is the ratio of correct predictions to the total number of predictions:
$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}.$$
It is quite simple and intuitive but does not assess well when the class distribution is imbalanced, leading to naive classifiers having a good accuracy. Therefore, we would rather use precision and recall, described just below.

- Precision: it is the ratio of correct positive predictions to the total number of positive predictions:
$$Precision = \frac{TP}{TP + FP}.$$
A high precision means that there are few false positives, meaning that positive predictions are likely to be correct.

- Recall: it is the ratio of correct positive predictions to the total number of predictions whose ground-truth values are positive:
$$Recall = \frac{TP}{TP + FN}.$$
A high recall means that there are few false negatives, meaning that ground-truth positive data are likely to be predicted as positive.

Note that those measures depend on the probability threshold at which the positive class is predicted. However, there exists a metric that summarizes the trade-off between precision and recall, independently of a threshold. It is called the Area Under the Precision-Recall Curve (AUC-PR). To obtain it, we first have to determine the precision-recall (PR) curve. To do so, we need to compute precision and recall for different prediction thresholds from 0 to 1, and then plot a curve with recall and precision on the x- and y-axis respectively. After that, the AUC-PR is simply the area under the obtained curve. Figure 2.5 below illustrates that process.

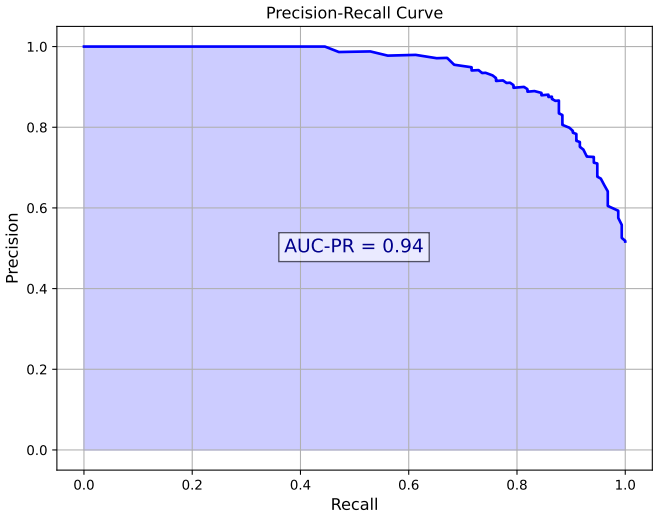


Figure 2.5: Example of AUC-PR.

Other Considerations

Other important considerations in the performance evaluation of models are overfitting and underfitting, as well as cross-validation. However, we will not go into too much detail about these notions, as we will not really need them to understand the rest of the work.

Overfitting happens when a model fits too well the training data, leading to poor performance on new unseen data. It often happens when the model is too complex for the task. Conversely, underfitting happens when a model is too simple for the task, leading to poor performance during the training and test phases. We can limit them by making a good choice of hyperparameters, i.e. parameters that are not the weights to optimize. There also exist other techniques to limit overfitting, like regularization where a weight-dependent term is added to the loss function.

Cross-validation is a technique used to assess the performance and generalizability of a model. The training set is divided into several folds, and the model is then trained on some folds and validated on others, using some performance metric. This process is repeated several times, with different validation folds each time. The final performance result is the average of the obtained performances on each validation phase. A common technique is the k-fold cross-validation, where the training set is divided into k folds, using each time $k-1$ folds for training and 1 fold for validation. The process is thus repeated k times here to use all folds as a validation fold once. Such techniques can help in hyperparameter tuning while avoiding overfitting.

Now that we have introduced the general training process of ML algorithms as well as performance evaluation for ML models, let us dive into neural networks, beginning with the perceptron.

2.2.3 Perceptron

The perceptron, also called neuron because of its biological neuron's inspiration, is the simplest form of a neural network and thus constitutes a building block of many modern neural networks.

Structure

Its general structure is represented in figure 2.6 below.

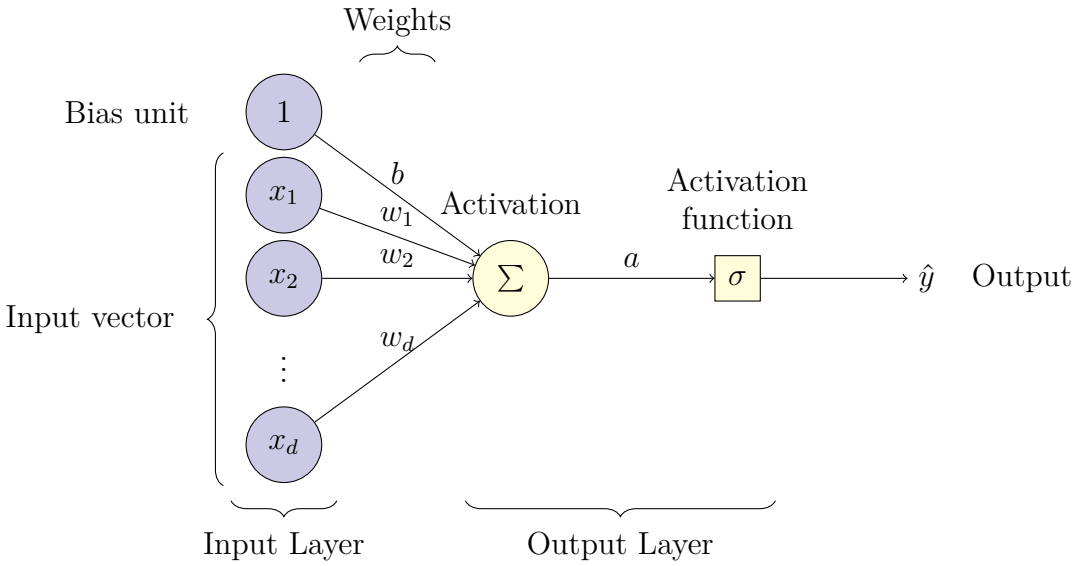


Figure 2.6: Perceptron.

It receives a d-dimensional input vector $\mathbf{x} = [x_1, x_2, \ldots, x_d]$ and processes it to produce an output signal $\hat{y}$. This processing consists of a weighted sum of each feature x_i of $\mathbf{x}$, to which we also add a bias term represented by the special weight b. This bias allows the weighted sum to be shifted vertically. The result is called the activation value a, as it then passes through an activation function σ to give the final output $\hat{y}$. The whole process can be described by the following equation [6]:

$$\hat{y} = \sigma(a) = \sigma(\sum_{j=1}^{d} w_j x_j + b).$$

Here, the weights $\mathbf{w} = [w_1, \ldots, w_d]$ and the bias b are the parameters of the model. The activation function σ determines the output $\hat{y}$ of the perceptron. Initially, a Heaviside step function was used as the perceptron was used as a classifier:

$$\sigma(a) = H(a) = \begin{cases} 1 & \text{if } a \geq 0 \\ 0 & \text{if } a < 0. \end{cases}$$

However, other continuous activation functions can be used to perform a regression. We can also perform a regression that outputs values in $[0, 1]$ representing probabilities, and then use a threshold for classification.

Training

After choosing an appropriate loss function L, a learning rate η, and initial weights $\mathbf{w}_0$, we can write the update rule of the weights for the perceptron:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} L(\mathbf{w}, b)$$

$$b \leftarrow b - \eta \frac{\partial L(\mathbf{w}, b)}{\partial b}$$

or element-wise for the classical weights:

$$w_j \leftarrow w_j - \eta \frac{\partial L(\mathbf{w}, b)}{\partial w_j}.$$

Then, by using the chain rule and equation 2.1, the element-wise update rule becomes:

$$w_j \leftarrow w_j - \eta \cdot \frac{1}{n} \sum_{i=1}^{n} \frac{\partial l}{\partial \hat{y}}(\hat{y}_i, y_i) \cdot \frac{\partial \hat{y}}{\partial a}(a_i) \cdot \frac{\partial a}{\partial w_j}(\mathbf{x}_i, \mathbf{w}, b)$$

$$b \leftarrow b - \eta \cdot \frac{1}{n} \sum_{i=1}^{n} \frac{\partial l}{\partial \hat{y}}(\hat{y}_i, y_i) \cdot \frac{\partial \hat{y}}{\partial a}(a_i) \cdot \frac{\partial a}{\partial b}(\mathbf{x}_i, \mathbf{w}, b)$$

where we have $\frac{\partial \hat{y}}{\partial a}(a_i) = \sigma'(a_i)$, $\frac{\partial a}{\partial w_j}(\mathbf{x}_i, \mathbf{w}, b) = [\mathbf{x}_i]_j = x_{ij}$, and $\frac{\partial a}{\partial b}(\mathbf{x}_i, \mathbf{w}, b) = 1$.

Limitation

The layer containing the bias unit and the input vector is called the input layer, while the one computing the weighted sum and passing it in the activation function is called the output layer. As only one layer performs the computations (the output layer), the perceptron can be seen as a single-layer neural network. Due to this structure, the perceptron can perfectly solve linearly separable problems, i.e. problems where the two classes can be separated by a hyperplane. However, this is not the case for problems that are not linearly separable. Fortunately, this limitation can be overcome by using multiple layers of several perceptrons, forming a Multilayer Perceptron (MLP).

2.2.4 Multilayer Perceptron

The MLP is the generalization of the perceptron, obtained by stacking up one or multiple layers of interconnected neurons, and with possibly multiple outputs.

Structure

Its general structure is represented on figure 2.7 below:

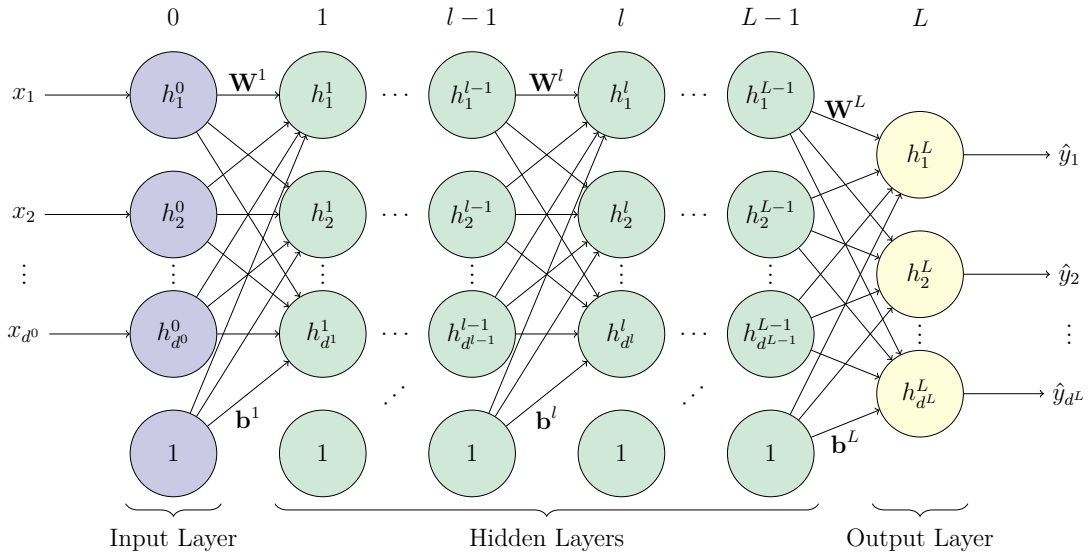


Figure 2.7: Multilayer perceptron.

It also receives a d^0-dimensional input vector $\mathbf{x} = [x_1, x_2, \ldots, x_{d^0}]$, but this time it will be processed through L layers to produce the outputs $\hat{\mathbf{y}} = [\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_{d^L}]$. Each neuron of each layer works like the perceptron: it performs a weighted sum

of the outputs of the neurons on the previous layer, to which we add a bias term, forming an activation value that passes through an activation function. This output is then used by the neurons on the next layer. $\mathbf{W}^l$ is the matrix of weights between neurons of layer $l-1$ and neurons of layer l, with $[\mathbf{W}^l]_{ij} = w_{ij}^l$ the weight that goes from i^{th} neuron of layer $l-1$ to the j^{th} neuron of layer l. Besides, $\mathbf{b}^l$ is the bias vector of layer l, with $[\mathbf{b}^l]_j = b_j^l$ the bias term of the j^{th} neuron of layer l. The whole process can then be described by the following equations, in matrix form:

$$\mathbf{h}^0 = \mathbf{x}$$

$$\mathbf{a}^l = \mathbf{W}^l \mathbf{h}^{l-1} + \mathbf{b}^l$$

$$\mathbf{h}^l = \sigma^l(\mathbf{a}^l)$$

$$\hat{\mathbf{y}} = \mathbf{h}^L$$

with $\mathbf{a}^l$ the activation of layer l, and $\mathbf{h}^l$ the hidden unit of layer l, i.e. the result of the activation function σ^l of layer l on its activation (acting element-wise). There are multiple common choices for the activation and loss functions, that will be discussed just after.

Training

As the weight matrixes and bias vectors are the parameters of the model, we can write the set of parameters of an MLP as $\theta = [\mathbf{W}^1, \mathbf{b}^1, \mathbf{W}^2, \mathbf{b}^2, \ldots, \mathbf{W}^L, \mathbf{b}^L]$. Then, after choosing a learning rate η and initial parameters θ_0, we can use the general gradient descent update rule of parameters:

$$\theta \leftarrow \theta - \eta \nabla_\theta L(\theta).$$

However, it can become costly to compute for MLPs with many layers of many neurons, resulting in a large amount of parameters. Consequently, it is often SGD that is used for MLP instead of classical gradient descent. Even with SGD, computing the gradient remains costly for the biggest models. So, to address this issue, we use the backpropagation algorithm.

In a nutshell, we can notice that there are a lot of redundant computations when using the chain rule for the partial derivatives of the loss function with respect to the weights and biases. Indeed, the partial derivatives associated with a layer need the partial derivatives associated with the next layer. Therefore, after a forward pass of the data through the network to compute the outputs, we proceed to a backward pass where we compute all the partial derivatives from end to beginning, so that a layer can use the values of the partial derivatives of the next layer to compute efficiently its own partial derivatives. All the computed partial derivatives

are stored along the way so that they can be reused to compute all the new weights at the end thanks to the update rule.

There are also many other ways to improve the training of an MLP, like using custom parameter initializations, regularization, normalization, etc. There also exists SGD variants using multiple optimizations like adaptative stepsizes, scaling, etc.

Activation Functions and Universal Approximation Theorem

The MLP also has an input and an output layer. The hidden layers between these two are what makes it a multilayer neural network. An MLP is called deep when it has many hidden layers. There is still one important point to be made about activation functions: they must be non-linear. Indeed, there exists a theorem called the "Universal approximation theorem" that can be described as: "Multilayer perceptrons, with at least one hidden layer and a variety of non-linear activation functions (sigmoid, tanh, ReLU, ...) can approximate any continuous function $f : X \to Y$ from a finite-dimensional space X to another one Y, with any desired precision given enough hidden units" [16]. In contrast to the single-layer perceptron that is limited to linearly separable problems, an MLP can thus in theory solve any continuous non-linear problem. It could not be possible if the activation functions were linear, since successively applying linear transformations on linear weighted sums would result in linear outputs.

Here are some common choices of non-linear activation functions:

- Sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$.
- Rectified Linear Unit (ReLU): $\sigma(x) = \max(0, x)$.
- Leaky ReLU (with parameter $0 < \alpha < 1$): $\sigma(x) = \max(\alpha x, x)$.
- Hyperbolic tangent: $\sigma(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$.
- Hard hyperbolic tangent: $\sigma(x) = \text{htanh}(x) = \max(\min(x, 1), -1)$.

And they are represented in figure 2.8 below:

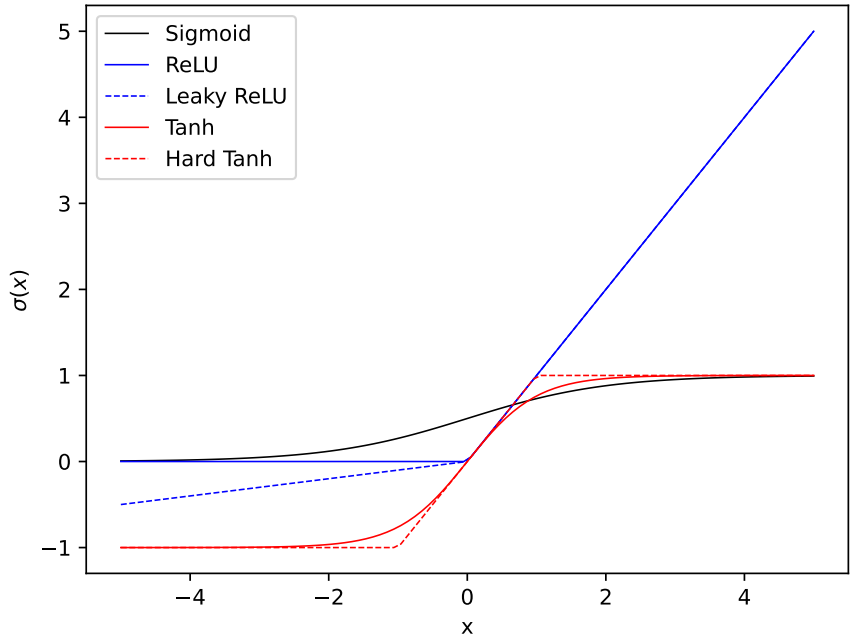


Figure 2.8: Common choices of non-linear activation functions.

One of the most popular choices for hidden layers is the ReLU activation function, because its derivative is equal to 1 whenever the neuron is active, i.e. its activation is greater than 0. This limits the vanishing gradient phenomenon, that the sigmoid and hyperbolic tangent activation functions suffer from because of their derivatives being significant only near 0. Also, a popular choice to perform a multi-class classification with $d^L = C$ classes is to use the softmax activation function on the output layer, combined with the categorical cross-entropy loss (CCE) [2]. The softmax function does the following:

$$\hat{y}_k = \hat{h}_k^L = \text{softmax}_k(\mathbf{a}^L) = \frac{e^{a_k^L}}{\sum_{j=1}^{C} e^{a_j^L}}.$$

So, we have $\hat{y}_k \in [0,1]\ \forall k \in 1,2,\ldots,C$, and $\sum_{k=1}^{C} \hat{y}_k = 1$. We notice that each output $\hat{y}_k$ is thus a probability, corresponding to the probability of the input belonging to class $k \in 1,2,\ldots,C$. The corresponding CCE loss, for the i^{th} datapoint, is then the following:

$$l_{CCE}(\hat{y}_i, y_i) = -\sum_{j=1}^{C} y_{ij} \log(\hat{y}_{ij})$$

where y_{ij} equals 1 if the i^{th} datapoint is of class j, and $\hat{y}_{ij}$ is the output probability of the i^{th} datapoint being of class j. CCE loss is thus the generalization of the

BCE loss with more than 2 classes.

Now that we have explained the basis of deep neural networks with the MLP, let us introduce new types of deep neural networks that will be used in the context of our work: convolutional neural networks (CNN) and feature pyramid networks (FPN). As they use the same principles as for MLPs, we will not go deep into the details but rather describe them on the surface and discuss their benefits.

2.2.5 Convolutional Neural Networks

Unlike MLPs where the order and structure of inputs do not matter, CNNs are a type of neural network specialized in processing data with a grid-like topology. For example, an image can be seen as a 2-dimensional grid of pixels, each pixel having a certain value. They are therefore widely used in computer vision tasks like object recognition, showing good performance. They take their name from the fact that they use a particular mathematical operation: convolution. According to [14], "convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers". Let us describe the structure of a CNN and how it works, before explaining why such networks are so efficient in computer vision.

Convolutional Layer

The convolution used in ML applications differs a bit from the one used in mathematical or signal processing applications but keeps the same principle of applying a sliding filter to a function. Let us describe a convolutional layer for 2-dimensional data. A 2-dimensional convolutional layer takes as input a 2-dimensional grid of values and applies a sliding 2-dimensional filter to it to produce another 2-dimensional grid of values as output. Let I be the input grid, K be the filter, also called kernel, and S be the output grid, also called feature map. Then, the value at position (i, j) of the output feature map S is computed by:

$$S(i,j) = (I * K)(i,j) = \sum_{m=1}^{M} \sum_{n=1}^{N} I(i+m, j+n) K(m,n)$$

where M and N are the dimensions of the kernel, and $*$ denotes the 2-dimensional convolution operator. In reality, the operation performed here is called the cross-correlation, but it is commonly called convolution in the context of ML. We can find a simple example of this operation in figure 2.9 below, with a 3x4 input grid and a 2x2 kernel, producing a 2x3 feature map.

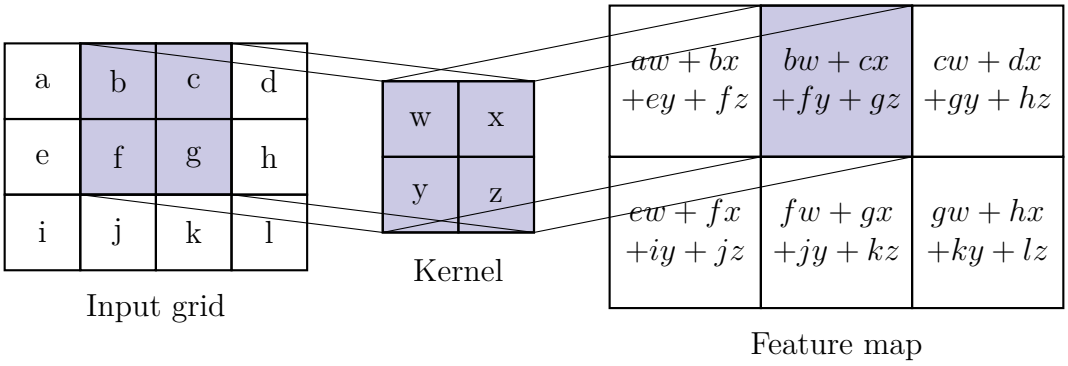


Figure 2.9: Simple example of a 2-dimensional convolutional layer.

This is the base case of a convolution in a CNN, but in reality there are multiple hyperparameters that define a convolutional layer. To get an overview, three important of them are the kernel size defining the dimensions of the kernel, the stride defining the number of units between each shift of the kernel, and the padding allowing to add pixels on the borders of the input. Moreover, as for MLP, a bias term can be added to each computation.

Another important consideration is that values on a grid can be vectors. For example, a pixel in a color image is defined by a vector of three RGB values, representing its intensity in red, green, and blue respectively. In this case, the initial vector grid is separated into several scalar grids; one for each component. Each scalar grid is called a channel. In a convolution, the input and the output can have channels. In that case, each pair of input and output channels has its own kernel. Still in the 2-dimensional case, let I_k be the k^{th} channel of the input, and S_l be the l^{th} channel of the output. Then, K_{kl} is the kernel associated with these channels, and the value at position (i, j) of the l^{th} output channel S_l is computed by:

$$S_l(i,j) = \sum_k (I_k * K_{kl})(i,j) = \sum_k \sum_m \sum_n I_k(i+m, j+n) K_{kl}(m,n).$$

Figure 2.10 below shows such a multi-channel convolution with 3 input channels and 4 output channels.

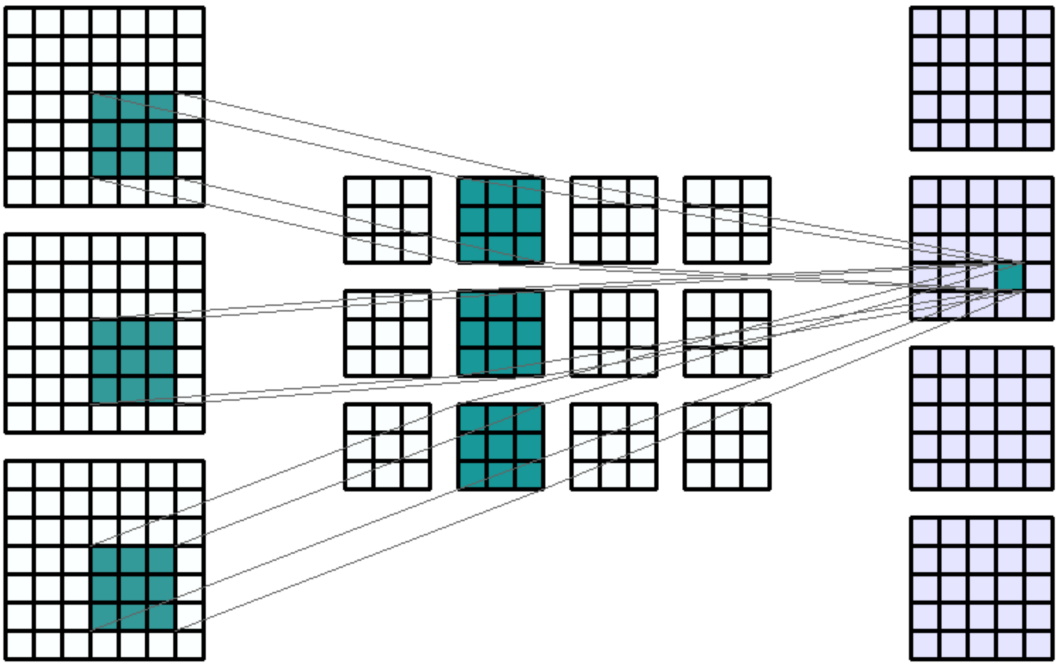


Figure 2.10: Multi-channel convolution (image taken from [43]).

So far, we only covered the 2-dimensional case. However, all this can be extended to the 3-dimensional case, with 3-dimensional grids and kernels. For example, a color video can be seen as the concatenation of 2-dimensional grids of RGB pixels, representing successive color images separated by a timestep. This concatenation thus forms a 3-dimensional grid of RGB pixels, that can be separated in three channels of 3-dimensional grids.

CNN Architecture

In a typical CNN, convolutional layers are combined with other types of layers. First, as convolution is a linear operation, non-linear activation function layers are used after convolutional layers to introduce non-linearities in the network, which is essential as explained in section 2.2.4. Then, we also use what we call pooling layers. This type of layer allows to summarize the information contained in neighborhoods of the grids. A popular choice is to use max pooling layers, outputting the maximum value of a neighborhood, as shown in figure 2.8 below with a 2x2 max pooling layer. There exist other types of pooling layers, like the average pooling layers which output the average value of a neighborhood.

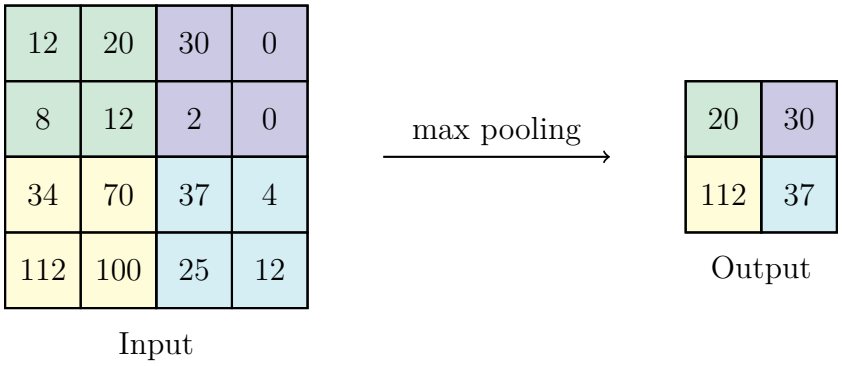


Figure 2.11: Simple example of a 2-dimensional max pooling layer.

As will be developed after, convolutional layers act as feature extractors. So, to perform tasks like classification, they are often combined with fully connected layers (the same layers as in MLPs). To do so, feature maps of the last convolutional layer pass through a flatten layer to flatten all values of all feature maps in a single 1-dimensional vector. Then, it works like an MLP until classification. Figure 2.12 below shows a representation of such a network.

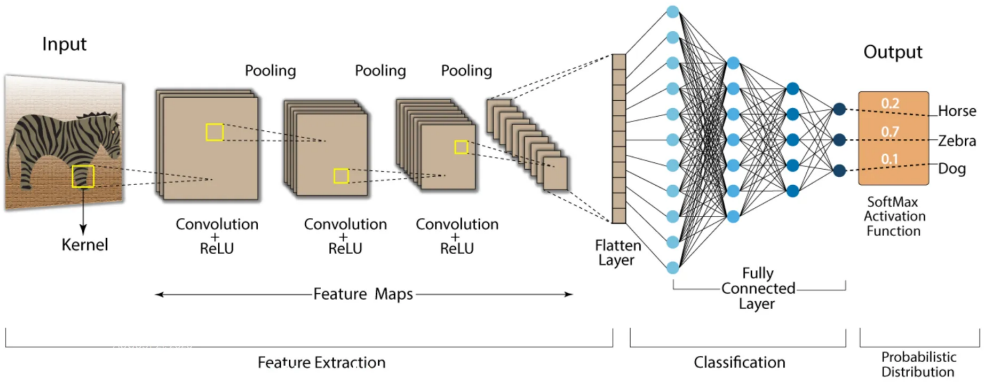


Figure 2.12: A basic CNN architecture (image taken from [38]).

Training

The parameters to train in convolutional layers are the weights of the kernels, as well as biases, if any. In general, SGD is used with an adapted version of the backpropagation algorithm for CNNs. Also, to stabilize and speed up the training of CNNs, batch normalization layers are often used after convolutional layers [19]. Such a layer normalizes its input with respect to its mean and standard deviation in the mini-batch and is also trainable.

Benefits

CNNs offer many advantages over conventional MLPs. First, convolutional layers induce sparse interactions and parameter sharing. Sparse interactions (or sparse connectivity) means that each output unit interacts only with a (small) part of the input units. Indeed, if the kernel is of size k, each output unit of a feature map interacts with k input units. Parameter sharing refers to using the same parameter in more than one computation. In a convolutional layer, the same kernel is used at every position of the input to produce the output, sharing its k parameters with each output unit.

These two properties result in a great gain of performance compared to a fully connected layer, as it reduces the number of computations and parameters. Therefore, it leads to better computational and memory efficiency. Let us consider an input layer of size m and an output layer of size n. For a fully connected layer, this would result in mn computations and mn parameters, whereas for a convolutional layer with a kernel of size $k \ll m, n$, it would result in only kn computations and k parameters, which is considerably better. This reduced number of parameters and computations also makes a CNN less prone to overfitting.

Also, the form of parameter sharing in a convolutional layer induces a property called equivariance to translation. It means that if we translate a part of the input in a certain way, its representation will be translated the same way in the output. Therefore, we can see kernels as filters that will be applied to each location of the input to detect specific patterns or features anywhere in the input.

Moreover, since each unit interacts with multiple previous units which themselves interact with multiple previous units, even though direct interactions in a CNN are sparse, deeper units may indirectly interact with a larger portion of the input. Complex interactions between many variables can thus be efficiently captured by CNNs. Besides, they are capable of learning hierarchical representations of the input. For example in the 2-dimensional case, early layers tend to learn detecting low-level features such as edges and textures, while deeper layers tend to learn detecting higher-level features such as shapes and objects [46]. That is why we say that CNNs are powerful feature extractors.

Finally, in addition to reducing computational costs as they reduce the dimensions of feature maps, pooling layers help to introduce a sort of local invariance to small translations. That means that a small translation of the input should not change most of the pooled values. It can be useful if we care more about the presence of a feature than about its location, and can make the model more robust to small

translations of the input.

Now that we have introduced CNNs as powerful feature extractors, let us introduce even better feature extractors specialized in object detection: feature pyramid networks (FPNs).

2.2.6 Feature Pyramid Networks

As a reminder, object detection includes their localization and classification. Whereas classical CNNs perform well at recognizing single objects, detecting multiple objects on the same image is often challenging, especially if they have different scales. In particular, they often struggle to detect small objects. To address this issue, FPNs were created, taking advantage of CNNs' inherent capability of learning hierarchical representations of the input [24]. Let us describe their structure as well as their benefits.

Structure

An FPN is composed of two stages:

- **Bottom-up pathway:** it is the standard feedforward computation of a backbone CNN that generates a feature hierarchy with layers of increasing semantic value but decreasing spatial resolution, as each successive feature map is downscaled with a scaling factor of 2 using stride.
- **Top-down pathway and lateral connections:** the top-down pathway upsamples higher-level feature maps, with higher semantic but lower spatial value (resolution), by a factor of 2 using a simple nearest neighbor upsampling. Then, lateral connections merge these upsampled feature maps with the ones from the bottom-up pathway, which have lower semantic but higher spatial value. The feature maps from the bottom-up pathway undergo a 1x1 convolution before being added to the ones of the top-down pathway, in order to adjust channel dimensions. Finally, each merged map undergoes a 3x3 convolution in order to reduce the aliasing effect due to upsampling.

For greater clarity, figure 2.13 below shows these two pathways.

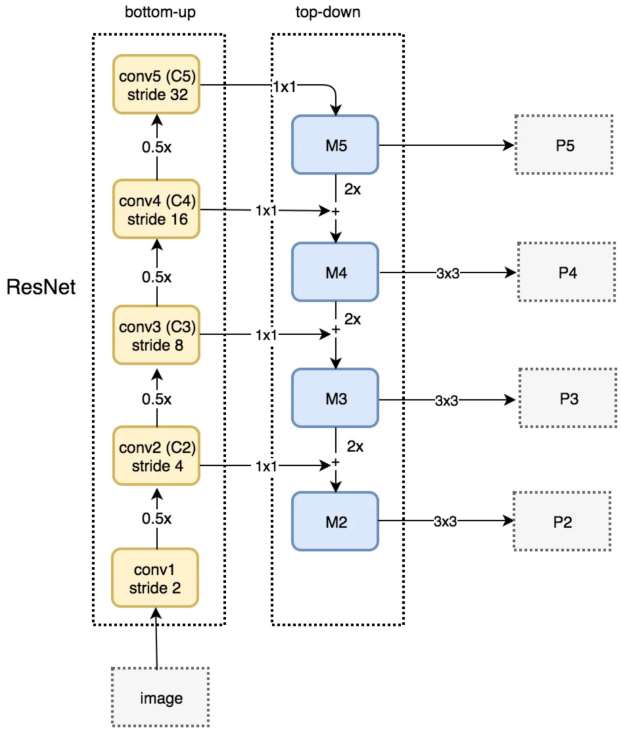


Figure 2.13: FPN architecture (image taken from [18]).

Benefits

The key of FPNs is that the constructed feature pyramid combines both high-level, semantically strong features from deeper layers and low-level, spatially strong features from lower layers. So, it will increase a detector's capacity to predict object locations. Moreover, by combining different feature maps of different scales and resolutions, FPNs enable better handling of objects of different sizes, especially small ones. All that without much computational overhead compared to conventional CNNs.

Note that FPNs are only feature extractors. In order to detect objects, a detector must be added, that will use the produced feature maps at each level of the feature pyramid to make its predictions, as depicted in figure 2.14 below.

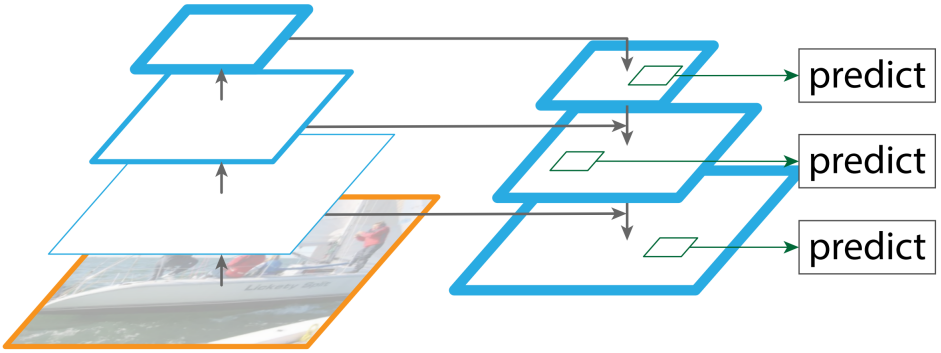


Figure 2.14: A detector uses feature maps of all levels to make its predictions (image taken from [24]).

Chapter 3

ROAD-R Challenge Framework

Now that we have explained the theoretical background needed to understand this work, let us dive into the ROAD-R dataset and its associated challenge. The goal here is to increase the performance of state-of-the-art models for road event detection by making them able to learn from requirements expressed as logical constraints, representing background knowledge about the problem, and by making their predictions compliant with the requirements. In this chapter, we will first formally define the problem, then describe the ROAD-R dataset, containing road events videos and equipped with with a set of requirements expressed as logical constraints. We will continue by explaining how the authors of the ROAD-R paper [13] performed the baseline road event detection, and then how they took advantage of the requirements to increase the performance of their models. We will finish by briefly presenting some other state-of-the-art methods to achieve the same goal. Naturally, a lot of information in this chapter comes from the ROAD-R paper.

3.1 Formal Definition of the Problem

Road event detection takes place in 2 stages: first predict bounding boxes, giving the locations of the events on the images, and then predict their associated sets of labels for classification. Focussing on the second stage, we can formulate it as a multi-label classification (MC) problem with requirements.

An MC problem is a problem $\mathcal{P} = (\mathcal{C}, \mathcal{X}, \mathcal{Y})$, where $\mathcal{C}$ is the set of all possible labels, denoted by $A_1, A_2, \ldots$, $\mathcal{X}$ is a set of bounding boxes $x \in \mathbb{R}^4$ (a bounding box can be entirely described by 4 coordinates), and $\mathcal{Y}$ is the associated set of ground truth values $y \subseteq \mathcal{C}$ for each bounding box. Let a model m be a function $m : \mathcal{C} \times \mathcal{X} \to [0, 1] : (A, x) \to m(A, x)$. A bounding box x is predicted by a model m to have label A if $m(A, x) > \theta$, with $\theta \in [0, 1]$ a user-defined

threshold. Then, the prediction p of model m for bounding box x is the set $p = \{A \mid A \in \mathcal{C}, m(A, x) > \theta\} \cup \{\neg A \mid A \in \mathcal{C}, m(A, x) \leq \theta\}$. Therefore, $\forall A \in \mathcal{C}$, either $A \in p$ or $\neg A \in p$. A bounding box is then labeled only with the positive labels in its prediction.

An MC problem with propositional logic requirements is a problem $(\mathcal{P}, \Pi)$, where $\mathcal{P} = (\mathcal{C}, \mathcal{X}, \mathcal{Y})$ is an MC problem and Π is a set of propositional logic constraints r on $\mathcal{C}$. A prediction p is admissible if it satisfies each constraint $r \in \Pi$. A model m satisfies Π on a bounding box $x \in \mathcal{X}$ if its prediction for x is admissible. Therefore, if m satisfies Π on all the bounding boxes, the set of all possible predictions is restricted to only the admissible ones with respect to Π.

Let us now introduce the ROAD-R dataset, which will be used to train a model capable of performing road event detection while taking advantage of logical requirements, giving rise to an MC problem with requirements.

3.2 ROAD-R Dataset

ROAD-R, standing for ROad event Awareness Dataset with logical Requirements, "is the first publicly available dataset for autonomous driving with requirements expressed as logical constraints" according to the paper. It extends the ROAD [41] video dataset by adding the set of requirements to it.

3.2.1 Videos

The ROAD-R dataset is made up of 22 videos, each about 8 minutes long, where we have the point of view of a vehicle driving on the road, in the manner of an autonomous vehicle. These videos are annotated with road events, i.e. sequences of frame-wise labeled bounding boxes linked in time whose labels correspond to subsets of a set $\mathcal{C}$ of 41 labels, divided into three main categories: agents, actions, and locations. To be processed, each video is converted to images, at a rate of 12 images per second, and with a width and height of 1280 and 960 pixels respectively. Figure 3.1 below shows an example of such an annotated image. We structured the labels so that a line corresponds to a label category, in the same order as presented just before. Note that there can be multiple labels of the same category for the same bounding box and that a bounding box does not necessarily have a label in every category.

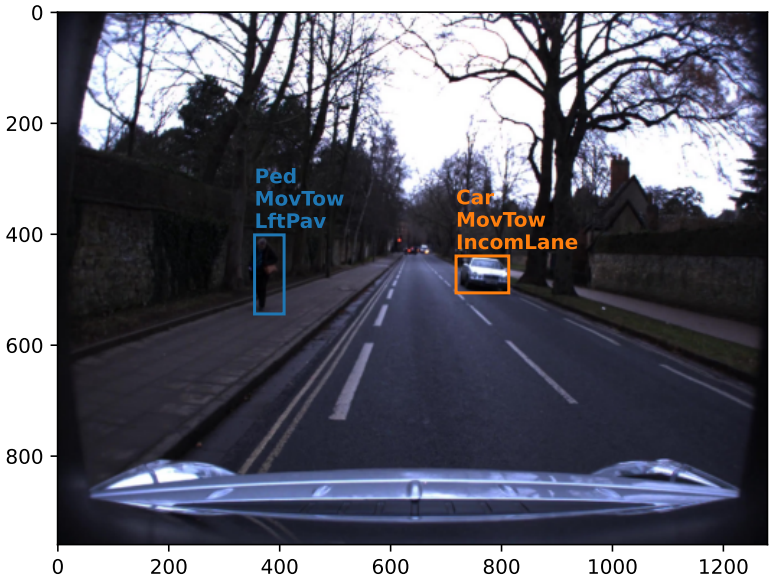


Figure 3.1: An example of an annotated image from the ROAD-R dataset.

3.2.2 Requirements

As said before, the video dataset is completed by a set of requirements Π expressed as logical constraints $r \in \Pi$. In total, there are $|\Pi| = 243$ constraints limiting the number of admissible combinations of labels. Together, all these constraints form a big logic formula in CNF, where each clause corresponds to a constraint. A SAT solver was used to verify that the whole formula was satisfiable, and it was also verified that all ground-truth annotations respected the constraints. There are different types of constraints in this set:

- First, the more common ones are constraints with only two negative labels, which express mutual exclusion between the two labels. Indeed, they have the general form $\neg L_1 \vee \neg L_2$, which is equivalent to $\neg(L_1 \wedge L_2)$ expressing that we cannot have L_1 and L_2 being *true* at the same time, or which is even equivalent to $L_1 \Rightarrow \neg L_2$ (or $L_2 \Rightarrow \neg L_1$) expressing that if L_1 is *true*, L_2 must be *false* (or conversely). Note that all combinations of two agent labels are used in these constraints, so an admissible prediction has at most one agent label.

- Then, there are constraints with one negative label and with the remaining labels being positive. They have the form $\neg L_1 \vee L_2 \vee \cdots \vee L_n$, which is

equivalent to $L_1 \Rightarrow (L_2 \vee \cdots \vee L_n)$ expressing that if L_1 is *true*, at least one label among $L_2, \ldots, L_n$ must be *true*.

- Finally, the least common ones are constraints with only positive labels, expressing that at least one of their labels must be *true*. There are exactly two of these: one stating that a bounding box must have at least one agent label and the other stating that it must have at least one location label except for traffic lights. Consequently, by combining these constraints with those of mutual exclusion, we can notice that an admissible prediction must have exactly one agent label, and at least one location label except for traffic lights.

An example of constraint $r \in \Pi$ is "a traffic light cannot be red and green at the same time", which is thus expressed as $\neg\text{Red} \vee \neg\text{Green}$, and can also be noted as $r = \{\neg\text{Red}, \neg\text{Green}\}$. From now on, we will always use this set notation for the constraints. Figure 3.2 below shows an example of a violation of this constraint.

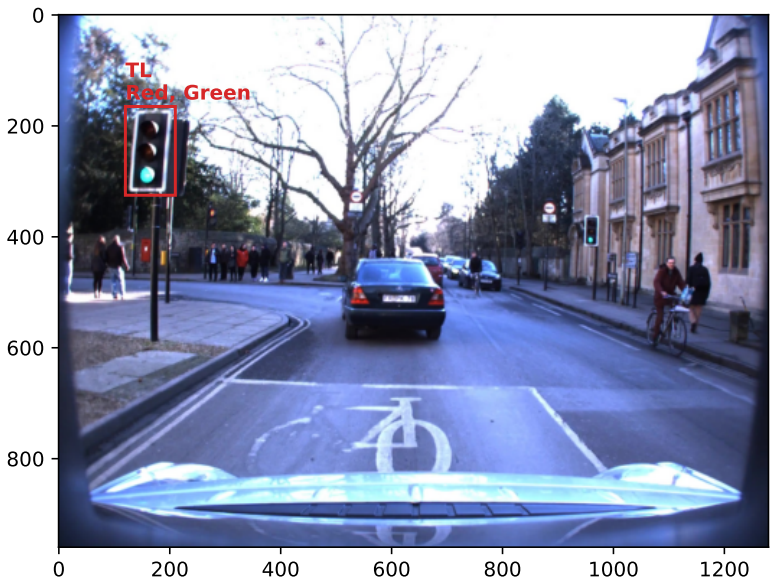


Figure 3.2: Example of violation of the constraint $\{\neg\text{Red}, \neg\text{Green}\}$.

Now that we have described the problem and the dataset that will be used, we can explain how to perform road event detection and some related concepts.

3.3 road event detection

Now that the problem has been formulated and that we know what the ROAD-R dataset looks like, we will explain how the authors of the ROAD-R paper performed and evaluated their baseline predictions, before taking advantage of the requirements. But first, let us explain some notions about object detection in general.

3.3.1 General Notions of Object Detection

Three important notions that are very useful to understand how object detection works in general are the intersection over union metric, the concept of anchor boxes, and the non-maximum suppression algorithm.

Intersection over Union

Intersection over Union (IoU), also called Jaccard index, is a metric used to measure the similarity between two shapes (or volumes) $A, B \subseteq \mathbb{R}^n$ [36]. It is computed by:

$$IoU = \frac{|A \cap B|}{|A \cup B|}.$$

For two bounding boxes $A, B \subseteq \mathbb{R}^2$, this amounts to compute the ratio between their area of intersection and their area of union, as shown in figure 3.3 below.

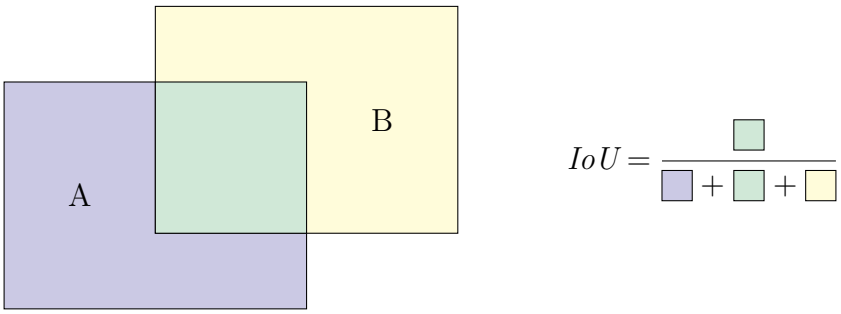


Figure 3.3: IoU between two bounding boxes A and B.

We always have $IoU \in [0, 1]$, with 0 obtained when the bounding boxes do not intersect and 1 obtained when they perfectly overlap.

Anchor Boxes

The concept of anchor boxes, first introduced by [35], is very useful in object detection, especially to predict the locations of bounding boxes. They correspond

to a set of predefined boxes of various sizes (scales) and shapes (aspect ratios), that are centered at each location of a feature map. Figure 3.4 below shows a set of three anchor boxes with different scales and aspect ratios, centered at each location of a 4x4 feature map obtained from a 16x16 image by a CNN. Note that although the feature map has a lower resolution than the image (i.e. has fewer pixels), the anchor boxes are scaled according to the image (and not according to the feature map).

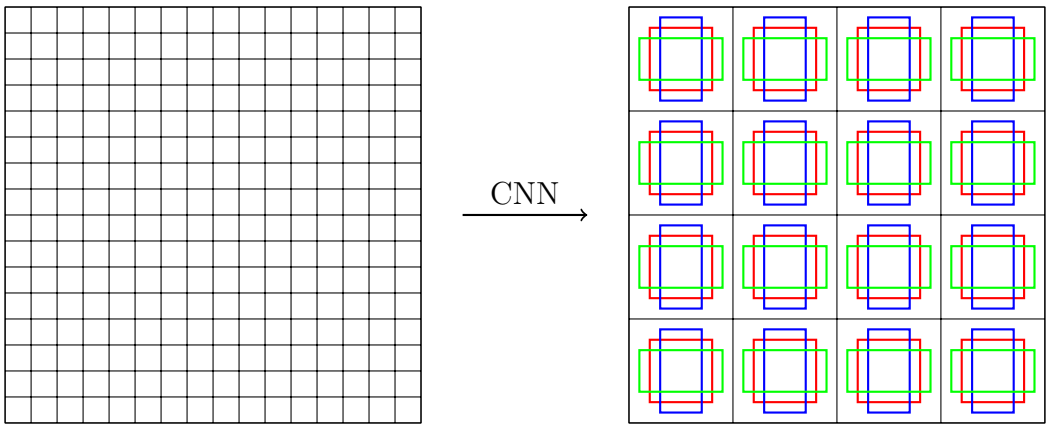


Figure 3.4: Set of 3 anchor boxes centered at each location of a feature map.

They act as reference points for the object detector to predict bounding boxes. Therefore, it is important to choose their sizes and shapes so that they could match with the possible objects to predict. Then, for each anchor box, the model predicts 4 offsets, refining the base location of the anchor box to a more precise one for the prediction, and a classification score for each possible class in $\mathcal{C}$. Hence, for a feature map of size $W \times H$ with k anchor boxes per location, a model outputs $W \times H \times k$ predictions (one per anchor box), and thus $W \times H \times k \times (|\mathcal{C}| + 4)$ values ($|\mathcal{C}| + 4$ per anchor box).

During training, we assign each anchor box to a ground-truth bounding box, or not, based on two IoU thresholds t_H and t_L with $t_H > t_L$. If an anchor box has an IoU with a ground-truth bounding box higher than t_H, then it is assigned to the ground-truth bounding box with which it has the highest IoU. In this case, its ground-truth labels are the ones of the ground-truth bounding box, and its ground-truth offsets are the ones between it and the ground-truth bounding box. If an anchor box has an IoU with every ground-truth bounding box lower than t_L, then it is considered as background, which can be seen as a label only useful for

training, and its ground-truth offsets are omitted. Finally, if an anchor box has an IoU with a ground-truth bounding box between t_L and t_H, then it is ignored as it is ambiguous. After that, the discrepancy between the predicted labels and offsets of each anchor box and their ground-truth labels and offsets is measured thanks to an appropriate loss function.

Non-Maximum Suppression

As a consequence of the anchor boxes method, a typical object detector outputs many detections for a same ground-truth object, with overlapping but slightly offset bounding boxes. Naturally, many of these detections are superfluous, and we only want to keep the best one. A first step is then to discard all detections with a very small confidence on their predicted labels. It does not suppress all superfluous detections, but it makes a good first selection.

Now, let us consider the case of binary classification, where the detector only detects one class of objects with a confidence between 0 and 1. Then, the goal of non-maximum suppression (NMS) is to only keep the most confident bounding box for a same ground-truth object. The principle of NMS is to consider that two bounding boxes that largely overlap correspond to the same ground-truth object. We can measure this overlap with the IoU metric presented just before, and if two bounding boxes have an IoU that exceeds a given threshold, then we only keep the most confident bounding box. Figure 3.5 below illustrates this with a simple example of three overlapping boxes with confidences of 0.6, 0.8, and 0.7. After NMS, only the box with a confidence of 0.8 is kept.

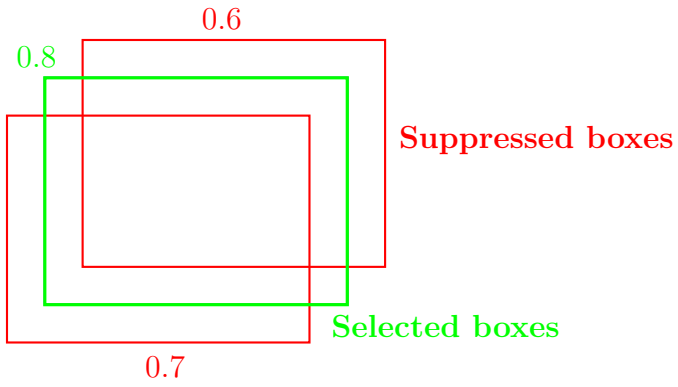


Figure 3.5: Example of NMS.

[3] gives a simple algorithm to perform NMS given a set of bounding boxes along with their confidences, and an IoU threshold. In brief, first select the bounding

box having the highest confidence and suppress all other bounding boxes for which the IoU with the selected box exceeds the threshold. Then, select the bounding box with the second highest confidence among the remaining ones and do the same. Repeat by selecting the third-best remaining bounding box, then the fourth-best,... until all bounding boxes have been processed.

We can extend NMS to the case where a same bounding box can have multiple labels, either by summarizing each of its classification scores into one global score (for example by taking the mean, or the maximum), or by performing NMS for each class separately and then merging the results.

We can now introduce the model that the authors of the ROAD-R paper used to perform road event detection: 3D-RetinaNet.

3.3.2 3D-RetinaNet

3D-RetinaNet [41] is the 3-dimensional version of RetinaNet [25]. It is a state-of-the-art model for object detection, that is built around the FPN concept. We will describe its structure as well as its training and inference (detection) steps.

Structure

Figure 3.6 below shows the structure of the 3D-RetinaNet architecture.

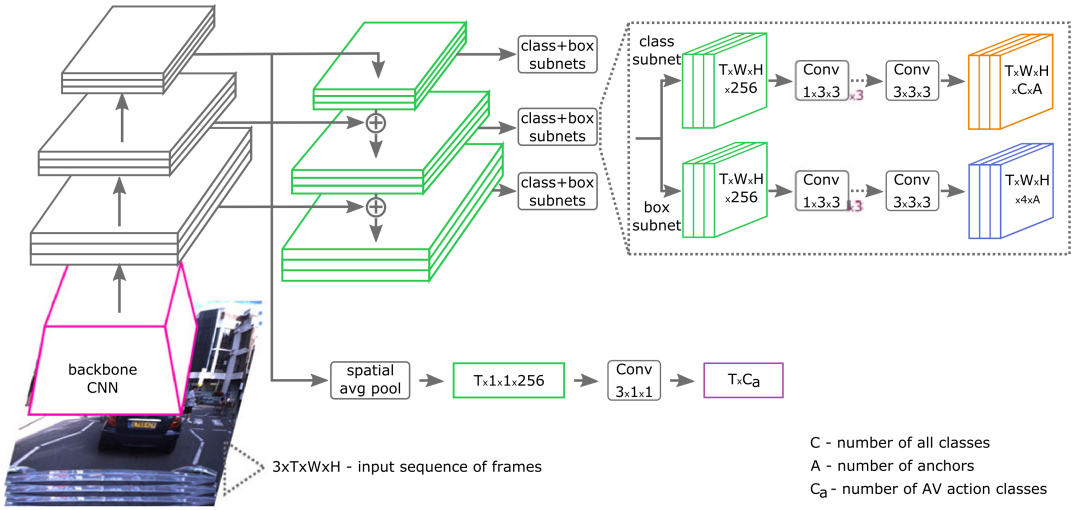


Figure 3.6: 3D-RetinaNet architecture (image taken from [41]).

It is composed of a 3-dimensional FPN with a 3-dimensional backbone CNN, thus

outputting a feature pyramid of 3-dimensional feature maps. The input of the backbone CNN is a sequence of T successive RGB frames of size $W \times H$ (representing a color video), resulting in 3 input channels of 3-dimensional feature maps of size $T \times W \times H$. At the end, each level of the feature pyramid has 256 channels of 3-dimensional feature maps, whose spatial resolution decreases the higher the level is. Note that in figure 3.6, the dimensions of $T \times W \times H$ were kept for these feature maps, but that in reality their W and H are smaller than those of the input image for higher levels in the feature pyramid, as their spatial resolution decreases. So, we will denote the width and height of the feature map at level l by W_l and H_l respectively. The authors of the ROAD-R paper considered 6 state-of-the-art 3-dimensional backbone CNNs based on a ResNet-50 architecture [15]: 2D-ConvNet (C2D) [44], Inflated 3D-ConvNet (I3D) [4], Recurrent Convolutional Network (RCN) [40], Random Connectivity Long Short-Term Memory (RCLSTM) [17], Random Connectivity Gated Recurrent Unit (RCGRU) [17], and SlowFast [11]. We will not detail them here, but these models have been shown to perform well for object detection in videos.

Then, at each level of the feature pyramid, two sub-networks are attached: one to perform object classification and the other to perform object regression. They have the same structure: they consist of successive convolutional layers with 256 kernels of size $1 \times 3 \times 3$ followed by ReLU activation functions, and of a last convolutional layer with kernels of size $3 \times 3 \times 3$ to produce the output feature maps. This is where anchor boxes come into play. At each level, we will use a set of A anchor boxes per location on the feature map, with scales and aspect ratios varying across the levels. Therefore, the last convolutional layer of the object classification sub-network will have $|\mathcal{C}| \times A$ kernels of size $3 \times 3 \times 3$ and will be followed by a sigmoid activation function (not softmax as a bounding box can have multiple labels here) to produce classification scores between 0 and 1 for each class in $\mathcal{C}$ and for each anchor box, on each location of the feature map. The same goes for the last convolutional layer of the box regression sub-network that will have $4 \times A$ kernels of size $3 \times 3 \times 3$ to produce 4 offsets for each anchor box, on each location of the feature map. This layer does not have any activation function. Consequently, the classification and regression sub-networks at level l will output $T \times W_l \times H_l \times |\mathcal{C}| \times A$ and $T \times W_l \times H_l \times 4 \times A$ values respectively. Note that these two sub-networks do not share parameters between them, but that they share their parameters between levels.

Training

As a reminder, when we introduced the anchor boxes method we said that, for training, some anchor boxes were assigned to ground-truth boxes while some other

ones were assigned to background. Well, during training, we will consider the background as a label. Indeed, it allows to penalize background anchor boxes for which other labels than background were predicted. We will assign the 0^{th} label to the background class; there are thus $|\mathcal{C}| + 1$ labels during training.

The loss function that they used for 3D-RetinaNet is a combination of the focal loss for classification [25] and the smooth L1 loss for box regression [12]. Let $y \in \{0, 1\}^{|\mathcal{C}|+1}$ be the ground-truth labels of an anchor box and $\hat{y} \in [0, 1]^{|\mathcal{C}|+1}$ be its classification scores. Let also $o \in \mathbb{R}^4$ be its ground-truth offsets and $\hat{o} \in \mathbb{R}^4$ be its predicted offsets. Then, the loss term associated with this anchor box is computed by:

$$l(\hat{y}, y, \hat{o}, o) = l_{FL}(\hat{y}, y) + \lambda(1 - y_0)l_{SL1}(\hat{o}, o)$$

where l_{FL} is the focal loss term, and l_{SL1} is the smooth L1 loss term. Parameter λ is a weight factor balancing these two loss terms, and factor $(1 - y_0)$ allows to ignore the smooth L1 loss term for background anchor boxes, for which $y_0 = 1$, as they are not assigned to any ground-truth bounding box. Focal loss is a variant of the standard cross entropy loss presented before, that address class imbalance. Its loss term is computed by:

$$l_{FL}(\hat{y}, y) = -\sum_{j=0}^{|\mathcal{C}|} y_j \alpha (1 - \hat{y}_j)^\gamma \log(\hat{y}_j) + (1 - y_j)(1 - \alpha)\hat{y}_j^\gamma \log(1 - \hat{y}_j)$$

where $\alpha \in [0, 1]$ and $\gamma \geq 0$ are tunable parameters. We see that the background label is taken into account in the focal loss as the sum starts at $j = 0$. Smooth L1 loss is a combination of L1 loss, which uses absolute differences, and L2 loss, which uses squared differences. It is less sensitive to outliers than L2 loss, and in some cases prevents exploding gradients [9]. For an anchor box that is not assigned to background, its loss term is computed by:

$$l_{SL1}(\hat{o}, o) = \sum_{d=1}^{4} \text{smooth}_{L1}(\hat{o}_d - o_d)$$

with

$$\text{smooth}_{L1}(x) = \begin{cases} \dfrac{0.5x^2}{\beta} & \text{if } |x| < \beta \\ |x| - 0.5\beta & \text{otherwise} \end{cases}$$

where β is the threshold from which we switch from an L2 loss to an L1 loss.

Finally, the parameters of the model are trained using SGD with backpropagation, using some SGD optimizers to speed up the process.

Inference

Inference is the process after which detections are produced on input images. It is separated into three steps. The first one is the production of raw detections, the second one is their filtering, and the third one is their thresholding. Raw detections are produced simply by forwarding a sequence of images to the network, thus producing a detection per anchor box. The problem is that there are too many anchor boxes compared to the number of objects to detect, as explained before. Therefore, we have to filter the raw detections in order to only keep the best one for each potential object. This is done by first deleting the raw detections with very low classification scores, and then by applying the NMS algorithm presented before. Finally, we only keep the filtered detections that have classification scores above the threshold θ as final detections.

3.3.3 Evaluation

A common evaluation metric for object detection is the mean average precision (mAP). It combines IoU with AUC-PR to provide a measure taking both location and classification into account. To compute mAP, we first need to redefine the confusion matrix introduced in section 2.2.2, and then explain what is average precision (AP) for a given class. This section is inspired by [34] and [39].

Confusion Matrix for Object Detection

In the object detection context, we have to introduce IoU in the confusion matrix. Indeed, the AP of a model for a given class is computed with respect to an IoU threshold, that will impact the number of detections in each category of the confusion matrix, allowing to take the locations of the detections into account. For a single class object detection task, given an IoU threshold τ and a classification threshold θ, we define the following detection categories:

- True positives (TP): predicted classification probability greater than θ and IoU between the predicted and the ground-truth bounding boxes greater than τ.
- False positives (FP): predicted classification probability greater than θ and IoU between the predicted and the ground-truth bounding boxes lower than τ.
- False negatives (FN): ground-truth detections that the model could not correctly detect.

Note that we did not define the true negatives (TN), since they will not be required to compute the AP for a given class, as we will see now.

Average Precision

The AP at a given IoU threshold τ for a given class is a particular case of the AUC-PR, explained in section 2.2.2, where we simply use the confusion matrix adapted for object detection just above instead of the classical confusion matrix for classification, to compute precision and recall at different classification thresholds and obtain the PR curve.

Besides, there exists a way to construct a PR curve without computing the entire confusion matrix, for all detections, at each classification threshold. Indeed, we will first sort all detections, across all images, by descending classification scores. Then, we will iterate through the detections in this order and each time consider their classification score as the current classification threshold. By doing so, we can simply classify the current detection as TP if the IoU between its predicted bounding box and a ground-truth bounding box is greater than τ, or as FP if it is not the case. We can then compute the corresponding cumulative precision and recall, that will correspond to a point on the PR curve. In particular, for the i^{th} most confident detection, with corresponding classification threshold θ_i, we can compute:

$$Recall_i = \frac{TP_i}{\#(\text{Ground-truth detections})}$$

$$Precision_i = \frac{TP_i}{TP_i + FP_i}$$

where TP_i and FP_i are the numbers of TP and FP respectively among the i most confident detections, thus at classification threshold θ_i, incremented through the iterations. So, $(Recall_i, Precision_i)$ is the i^{th} point of the PR curve, and the whole PR curve is plotted when all the detections have been processed.

The PR curve obtained with this method has the typical characteristics of PR curves. Indeed, on its left side, precision is typically higher and recall lower, as there are typically more TP than FP for high classification thresholds, but still not many compared to the number of ground-truth detections. On its right side, recall increases but precision might drop as the number of TP increases but more FP are included for low classification thresholds.

Finally, the AP at a given IoU threshold τ for a given class is the area under the obtained PR curve, that can be computed by a simple numerical integration scheme like the trapezoidal rule. [34] also explains a way of computing the AP more accurately by interpolating points on the PR curve, but a simple numerical integration is generally sufficiently accurate and easier to implement.

Mean Average Precision

We just explained how to compute the AP at a given IoU threshold τ for a single class object detection task. However, in the ROAD-R Challenge, we face a multi-labeled object detection task, where a single bounding box can have multiple labels among a set of classes. Hence, we can compute the AP at IoU threshold τ for each class separately. To compute the AP for a single class only, we can consider only its classification score for each detection and ignore the scores for the other classes. Then, we can compute the mAP at IoU threshold τ by taking the mean over all the single-class AP at IoU threshold τ:

$$mAP@\tau = \frac{1}{|\mathcal{C}|} \sum_{i=1}^{|\mathcal{C}|} AP_i@\tau$$

where $AP_i@\tau$ is the AP at IoU threshold τ for class i.

3.4 Learning with Requirements

We explained the baseline model used to perform road event detection, as well as the method to evaluate such a model. We will now explain how to take advantage of the set of requirements to improve the model. The authors of the ROAD-R paper presented two ways of doing so. The first one is to modify the loss function of the model so that it learns from the requirements during training. The second one is to add a post-processing step to the detections, enforcing the requirements on them. They called them constrained loss (CL) and constrained output (CO) models respectively. They also combined these methods to build models with constrained loss and constrained output (CLCO).

3.4.1 Constrained Loss

Integrating the requirements in the loss function allows the model to take them into account directly during training. Indeed, as they represent prior knowledge with respect to the detections, it should normally increase the model's performance. To integrate the requirements into the loss function, we add a regularization term representing the degree of satisfaction of the constraints in Π for the classification scores $\hat{y}$ of a detection:

$$l_{\Pi}(\hat{y}) = \alpha \sum_{i=1}^{|\Pi|} (1 - t_i(\hat{y}))$$

where t_i represents the fuzzy logic relaxation of the i^{th} constraint in Π, using t-norms and t-conorms as logic connectors between the scores as introduced in section 2.1.2.

If the scores reflect compliance with the i^{th} constraint, then $t_i(\hat{y})$ will be close to 1 and will therefore have little impact on loss. On the contrary, if the scores reflect non-compliance with the i^{th} constraint, then $t_i(\hat{y})$ will be close to 0 and will therefore have a greater impact on the loss. The hyperparameter α allows to control the weight and thus the importance of this regularization term in the total loss function.

There are several choices of t-norm that we can use. The authors of the ROAD-R paper chose to use the three fundamental t-norms (along with their corresponding t-conorms) presented in section 2.1.2: Łukasiewicz, Gödel, and Product t-norms.

3.4.2 Constrained Output

This method is a post-processing step, that is applied after the generation of the detections. It aims to correct predictions that do not respect the constraints so that they become compliant with the requirements, while trying to improve the performance of the model or at least not reduce it. This problem can be formulated as a weighted partial maximum satisfiability (PMaxSAT) problem, that we will introduce now.

Weighted PMaxSAT Problem

The weighted PMaxSAT problem is a generalization of the SAT problem [1]. Let us consider a logic formula in CNF, where each of its clauses corresponds to a constraint. Whereas the SAT problem consists of finding an assignment of the propositional variables that satisfies all the constraints, in the weighted PMaxSAT problem we assume that not all constraints can always be satisfied. So, we divide them into two groups: hard and soft constraints. Hard constraints must be satisfied, while soft constraints may be satisfied or not. We also add a weight to each soft constraint, representing a penalty if it is not satisfied. Then, the weighted PMaxSAT problem consists in finding the assignment that satisfies all the hard constraints, while minimizing the sum of the weights of the falsified soft constraints.

As it is a generalization of the SAT problem which is NP-complete, the weighted PMaxSAT problem is naturally also NP-complete. Therefore, no algorithm solves this problem in polynomial time. However, as for the SAT problem, there exists efficient solvers for it. For example, the authors of the ROAD-R paper used the publicly available solver MaxHS [8] [7].

Application of the Weighted PMaxSAT Problem to Post-processing

To model the post-processing step as a weighted PMaxSAT problem, we have to define what will be the hard constraints, and what will be the soft constraints. Let p be a prediction of the labels of a bounding box. As we want p to be compliant with all requirements, each constraint $r \in \Pi$ will naturally be a hard constraint. Then, as stated in section 3.1, $\forall A \in \mathcal{C}$, either $A \in p$ or $\neg A \in p$. It means that the prediction p contains all possible labels either in positive or negative form, depending on whether they were predicted or not by the model. So, each positive or negative label in p will be a soft constraint, forming a unit clause containing a single literal. If soft constraints are falsified in the solution of the problem, simply flip the corresponding labels to make the corrected prediction compliant with the requirements. Indeed, the flipped labels will now result to *true* with the same assignment. Let us illustrate this with the simple example of traffic lights.
Let $p = \{\text{Red}, \text{Green}\}$ (meaning Red = *true* and Green = *true*) be a prediction saying that a traffic light is red and green at the same time. Obviously, it cannot be true, as stated by the requirement $r = \{\neg\text{Red}, \neg\text{Green}\}$ (equivalent to $\neg\text{Red} \vee \neg\text{Green}$) already presented before. Therefore, we want to correct p so that it becomes compliant with r. By formulating this as a weighted PMaxSAT problem, r will be a hard constraint, while "Red" and "Green" will be soft constraints, giving the following CNF formula:

$$(\neg\text{Red} \vee \neg\text{Green}) \wedge \text{Red} \wedge \text{Green}.$$

We can notice that the solution of this weighted PMaxSAT problem will have an unsatisfied soft constraint, as the only way for the hard constraint to be satisfied is to have at least "Red" or "Green" to be *false*. Let us assume that the soft constraint "Red" has a greater weight than the soft constraint "Green". Then, the solution of this weighted PMaxSAT problem will be $\{\text{Red} = \textit{true}, \text{Green} = \textit{false}\}$. Indeed, with this assignment, the hard constraint is satisfied, and the weight of the falsified soft constraint is minimal. Finally, since the soft constraint "Green" is not satisfied in the solution, we flip the label "Green" to "$\neg$Green", and we obtain the corrected prediction $p' = \{\text{Red}, \neg\text{Green}\}$ which is now compliant with r.

To flip the i^{th} label, we change its output classification score $\hat{y}_i$ as follows:

$$f(\hat{y}_i) = \begin{cases} \theta + \epsilon & \text{if } \hat{y}_i < \theta \\ \theta - \epsilon & \text{otherwise} \end{cases}$$

with ϵ a small value, e.g. 10^{-3}, and θ the classification threshold, so that it reflects the fact that a flipped label is close to the limit of being predicted or not. Indeed, we expect that a label that was flipped from negative to positive has a lower

classification score than the labels that were initially predicted positive, and vice versa for the labels that were flipped from positive to negative. Now, there is a last point to discuss: the choice of the weights of the soft constraints.

Weights of the Soft Constraints

As the soft constraints of our applied weighted PMaxSAT problem correspond to positive and negative labels that will be flipped if they are falsified in the solution of the problem, the weights associated with them will correspond to the penalties of flipping them in order to correct the prediction. Let w_i be the weight associated with the i^{th} soft constraint, and thus the i^{th} label. The authors of the ROAD-R paper considered three choices of these weights:

- Minimal Distance (MD): all weights are unitary:

$$w_i = 1 \quad \forall i.$$

  This simple choice implies that the solution will give as few labels to flip as possible. However, it does not take any confidence or performance factors into account, and therefore often leads to poor results.

- Average Precision (AP): the weights are the average precisions of the labels:

$$w_i = AP_i.$$

  It allows to take into account the performance of the model for each label. In this way, labels with higher average precisions will have fewer chances to be flipped, as they are more likely to be correctly predicted compared to labels with lower average precisions.

- Average Precision and Output (AP×O): average precisions are now multiplied by the output scores of the model:

$$w_i = \begin{cases} AP_i \cdot \hat{y}_i & \text{if } \hat{y}_i \geq \theta \\ AP_i \cdot (1 - \hat{y}_i) & \text{otherwise.} \end{cases}$$

  This allows to combine both the model's global performance and its confidence in the current prediction, for each label.

3.4.3 Constrained Loss and Output

CL and CO models can be combined to produce CLCO models. These models thus use both a constrained loss, allowing to learn from the requirements during

training, and a constrained output, enforcing the requirements to the predictions. The predictions of such a model are thus compliant with the requirements, and we could expect a better performance since the requirements, representing prior knowledge, are taken into account during training. Different CLCO models can be considered by combining different t-norms for the constrained loss with different weights of the soft constraints for the constrained output.

3.5 State of the Art

There exist other methods to tackle the ROAD-R Challenge. We will here present some techniques used by the Challenge winners.

YQQL Team separated their model training into two stages. In the first stage, after a first preprocessing of the data, they used the YOLOv8 model by Ultralytics [20], a state-of-the-art object detection model showing good performance, to detect the bounding boxes and the types of agents only. In the second stage, after a second preprocessing of the data, they extracted regional image sequences of agents and predicted their action(s) and location(s) using the AIM ViT-L model [45], specialized in video action recognition. Also, they took the requirements into account by adding a post-processing step that downweights the classification scores of actions violating constraints. More information can be found on their GitHub [26].

MWIT Team also used YOLOv8, but they modified it to support multiple labels per bounding box by using n-hot vectors as ground-truth values instead of one-hot vectors, allowing multiple labels to have high confidence scores at the same time. They also added the constrained loss discussed before to it. Moreover, they designed a custom NMS algorithm specifically for the ROAD-R dataset, which focuses on the agent labels only. They also showed how to take advantage of unlabelled data with two models that together extend YOLOv8, refining its output scores thanks to the constrained loss applied to unlabelled data. See their GitHub [30] and their associated paper [31] for more information.

Lastly, PCIE-LR Team used an ensemble technique that combines the outputs of 6 models from the baseline (used by the ROAD-R paper authors) and one model from MMdetection [5][29] by OpenMMLab, a good-performing state-of-the-art object detection toolbox based on PyTorch. They modified some code of MMDetection to allow it to deal with multi-label data and pre-processed the data to convert it into a supported format. With MMDetection, they trained a Faster-RCNN [35] first only for the agent labels, and then they added the action and location labels to the training of the agent-only model. For further information, see their GitHub [23].

Chapter 4

Proposed Contributions

Now that we have explained the ROAD-R Challenge's aim of improving a deep learning model using a set of requirements expressed as logical constraints, as well as the concepts and methods used in this context, we will detail the objectives of our work and the different contributions we will make to help us achieve them.

4.1 Objectives

In addition to what has already been done in the ROAD-R Challenge paper [13], our objectives are twofold.

Firstly, training such a big model for object detection on such a big dataset is quite computationally expensive and can thus take much time. Therefore, we will explore whether it is possible to easily integrate a set of logical constraints into an existing model to improve it without having to re-train it entirely, with the aim of saving time in the process. This could be achieved by fine-tuning, using the requirements, already trained models that did not use them.

Secondly, the authors of the ROAD-R paper showed that, although the post-processing step makes the predictions compliant with the requirements, the gain in performance is usually limited, or even negative for some configurations. Therefore, we will explore new ways of performing the post-processing with the aim of making it more performant.

We will later analyze the performances obtained using different metrics, notably the mAP score, as well as the number of predictions that do not respect the constraints and need to be corrected. We will also discuss the time taken, as well as trade-offs between these metrics.

4.2 Fine-tuning

As said before, fine-tuning consists of the integration of the set of logical requirements into an already trained model that did not use them in its training. To achieve that, we can simply resume the training of such a model but with the constrained loss regularization term added to the total loss, as explained in section 3.4.1. As the purpose of this method is to save time by not re-training a model entirely, the number of fine-tuning iterations should be low compared to the number of iterations for which the model was initially trained. This would allow to integrate the additional knowledge contained in the requirements for a few iterations so that the model could adapt to them for a short time. The value of the hyperparameter α can be varied to produce fine-tuned models with different importances of the constrained loss regularization term, which thus take the requirements into account during the fine-tuning at different degrees.

4.3 Post-processing

We saw that the post-processing to make the predictions compliant with the requirements was based on the solving of a weighted PMaxSAT problem whose hard constraints were the requirements, and soft constraints were the predicted labels. As the predictions are corrected by flipping the labels of the falsified soft constraints in the solution of the problem, the key to having a performant post-processing relies on the choice of the weights of these soft constraints. The authors of the ROAD-R paper already presented some of them (see section 3.4.2), and we will explore new post-processing methods by proposing new weight choices for the soft constraints, essentially based on the prediction scores.

4.3.1 Pred-based Post-processing

In the ROAD-R paper, they proposed a post-processing that uses the labels' APs and confidence scores for the weights of the soft constraints, as well as one that only uses the labels' APs for the weights. However, they did not try to only use the confidence scores. So, for our first post-processing, we propose to set the weights of the soft constraints to the confidences of the model in predicting the labels or not:

$$w_i = \begin{cases} \hat{y}_i & \text{if } \hat{y}_i \geq \theta \\ 1 - \hat{y}_i & \text{otherwise.} \end{cases}$$

Of course, in the case where $\hat{y}_i < \theta$, the i^{th} label is not predicted, and so $1 - \hat{y}_i$ corresponds to the probability that the i^{th} label is not predicted. Hence, a label with low output probability is confident in being unpredicted. With this post-processing,

more confident predictions according to the model have a higher weight and have thus fewer chances to be flipped. In other words, this post-processing tends to flip the labels that are the closest to the classification threshold θ. We will call this post-processing *pred-based*. Unlike the AP and AP×O post-processing described in section 3.4.2, which are based on a global score reflecting the model's performance for each label, this *pred-based* post-processing is based solely on a local score for each label, specific to each prediction, reflecting the model's confidence in the current prediction. It can be beneficial in cases where the APs of some labels are so low that, after the post-processing, they become predicted almost by default in raw predictions that have missing labels.

Let us analyze such a case. Figure 4.1 below shows a ground truth annotation of an image along with the raw detections of a model on this image (with a confidence threshold $\theta = 0.5$). We can see that the model did not detect the truck, probably because of the light conditions, but that is not what we are interested in here. Let us look at the pedestrian detection instead; we can notice that it detected only its action of moving away. However, we saw in section 3.2.2 that there are two particular requirements stating that every bounding box must have an agent label and that every agent except traffic lights must have a location label, which is not the case here. Therefore, the post-processing step will correct this prediction so that it has an agent label and a (or several) location label(s).

Figure 4.1: Example of ground truth annotations on an image (left) and raw detections from a model on the same image (right).

We thus applied the AP and AP×O post-processings on that non-compliant prediction. We decided not to apply the MD post-processing, as it would have been essentially a random selection of an agent and a location label from the combina-

tions of labels that respect the constraints. Figure 4.2 below shows the results of these post-processings. We can notice that they both produced the same result: emergency vehicle and parking as agent and location labels respectively. These labels are obviously wrong compared to the ground truth, which says that it is a pedestrian on the left pavement. To understand these results, let us have a look at the class APs of these labels with the used model: $AP_{EmVeh} = 0\%$ and $AP_{parking} = 0.03\%$. These class APs are by far the two worst of all. They are so low that if there is a missing agent or location label, these labels will automatically be predicted after these post-processings as long as they are compliant with the other labels of the prediction. Indeed, for the emergency vehicle agent label, it is not even a penalty to flip it after solving the weighted PMaxSAT problem, as it has a zero AP. Moreover, even with the AP×O post-processing, no confidence score can be big enough to compensate for such low APs at threshold $\theta = 0.5$.

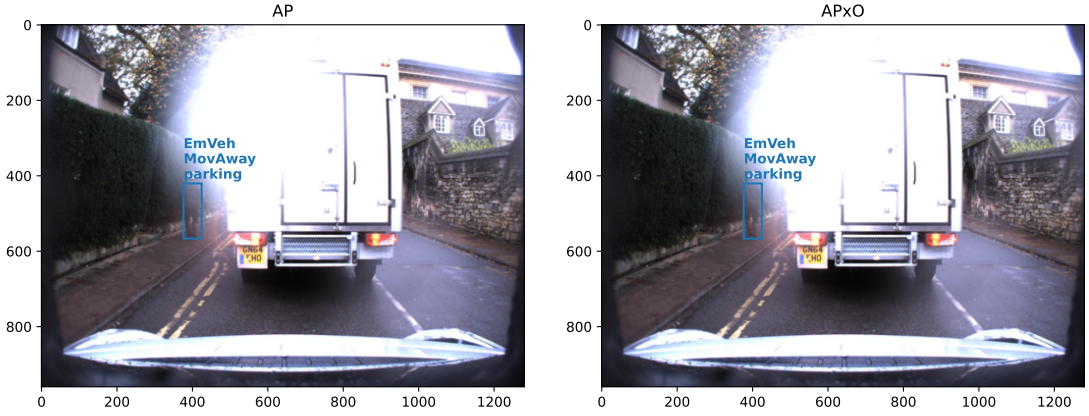


Figure 4.2: AP and AP×O post-processings applied on the raw prediction of the previous example.

Let us now see how the proposed *pred-based* post-processing deals with this example. As shown in figure 4.3 below, the post-processed prediction now aligns with the ground truth. We can explain this result by analyzing the confidence scores produced by the model. Indeed, we find that $\hat{y}_{Ped} = 0.48$, which is the highest agent label score but still under the threshold $\theta = 0.5$, thus unpredicted. Therefore, it is also the closest score to the threshold, and we have $w_{Ped} = 1 - \hat{y}_{Ped} = 0.52$ which is the lowest weight for the agent labels soft constraints. Similarly, we have $\hat{y}_{LftPav} = 0.25$ as the highest location label score, and thus $w_{LftPav} = 0.75$ as the lowest weight for the location labels soft constraints. Since they have the lowest weights in their respective label categories, the *pred-based* post-processing will flip the pedestrian and left pavement labels to be positive on this prediction, which is

what we want.

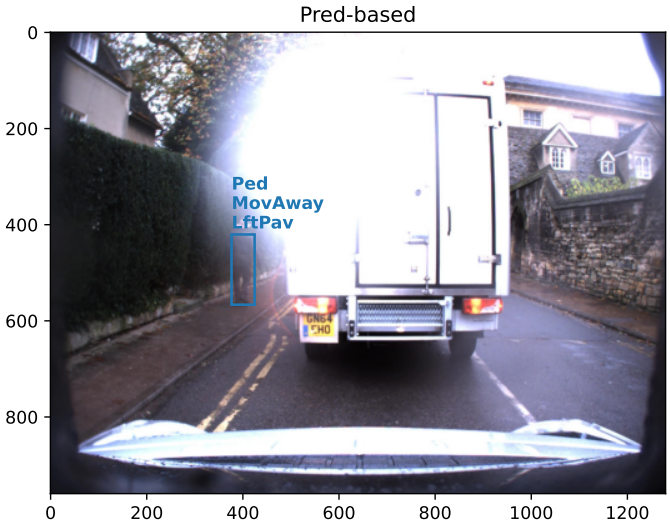


Figure 4.3: *pred-based* post-processing applied on the raw prediction of the previous example.

Let us now introduce another choice of weights for the post-processing that also relies on the confidence scores of the model, but with an extra twist.

4.3.2 Log-pred-based Post-processing

It often happens that instead of working with probabilities, we work with their logarithms. So, for our second post-processing, we will use the same idea as for the *pred-based* post-processing, except that we now use the logarithm of the probabilities. However, we must be careful with the fact that the MaxHS solver only works with positive weights. So, to have positive weights and still keep an increasing function, we will take the negation of the inverse of the logarithm of the probabilities:

$$w_i = \begin{cases} \dfrac{-1}{\log(\hat{y}_i)} & \text{if } \hat{y}_i > \theta \\ \dfrac{-1}{\log(1-\hat{y}_i)} & \text{otherwise} \end{cases}$$

where $\log(\cdot)$ is the natural logarithm function (base e). We will call this post-processing *log-pred-based*. Figure 4.4 below shows this function for confidence scores

ranging from 0 to 1. Note that the function is undefined on $\hat{y}_i = 0$ and $\hat{y}_i = 1$, with $\lim_{\hat{y}_i \to 0} \frac{-1}{\log(\hat{y}_i)} = 0$ and $\lim_{\hat{y}_i \to 1} \frac{-1}{\log(\hat{y}_i)} = +\infty$. Nevertheless, these two values are never output by any model since the output activation functions are sigmoids, whose values range from 0 to 1 excluded.

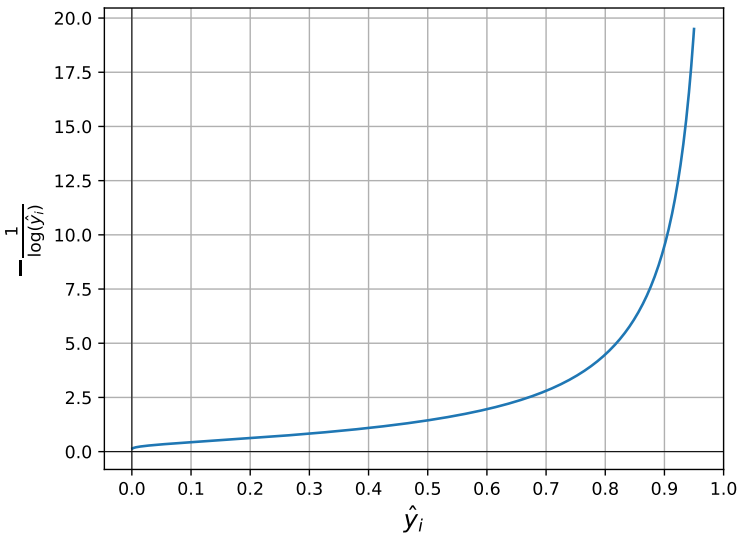


Figure 4.4: Plot of the function used for the *log-pred-based* post-processing, between 0 and 1.

The plot shows that the function is convex almost everywhere between 0 and 1. Therefore, compared to the *pred-based* post-processing, this method gives even more weight to more confident predictions than to less confident ones. The phenomenon is all the more significant as the confidence scores approach 1, where the function tends to positive infinity. This can be beneficial in some cases, for example when a label has a so high confidence score that we do not want to flip it, but prefer to flip several other labels with lower confidence scores instead.

Let us analyze such a case. Figure 4.5 below shows a new example of a ground truth annotation of an image along with the raw detections of a model on the same image (still with a confidence threshold $\theta = 0.5$). We can see that the model detected the car correctly, but that it has trouble with the cyclists. Indeed, the location labels are missing for both cyclists, and the cyclist on the left has wrong agent and action labels. Moreover, two other constraints are violated for the cyclist on the left: a cyclist cannot be a pedestrian, and a cyclist cannot push an object. We will now see how the different post-processings will correct these predictions.

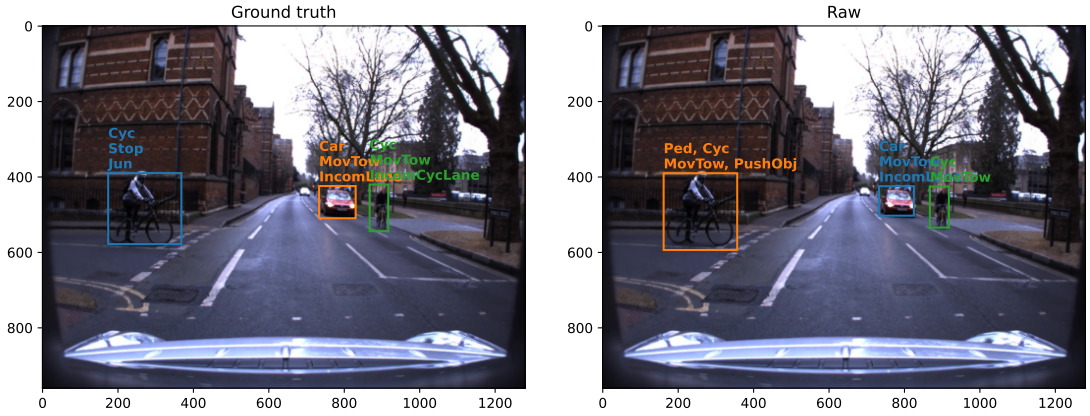


Figure 4.5: New example of ground truth annotations on an image (left) and raw detections from a model on the same image (right).

Again, we first applied the AP and AP×O post-processings on these detections, and we reported their results in figure 4.6 below. We can observe the same phenomenon as explained before: on both cyclists, both post-processings set the missing location label to a parking, which is wrong. This is still due to the fact that it is the location label with the lowest AP. However, for the cyclist on the left, the two post-processings produced different results for the agent and action labels. Before analyzing these results, let us take a look at the different possible combinations that a post-processing could produce in this case. Since a cyclist cannot be a pedestrian, one of these two labels will be flipped to negative. If the pedestrian label is flipped, then the constraint that a cyclist cannot push an object is still not respected. As a result, the pushing object label would also be flipped. If the cyclist label is flipped, then the two constraints are now respected. We see that the AP post-processing led to the second case, while the AP×O post-processing led to the first case. The APs and classification scores of the pedestrian, cyclist and pushing object labels are the following: $AP_{Ped} = 69.66\%$, $\hat{y}_{Ped} = 0.56$, $AP_{Cyc} = 70.81\%$, $\hat{y}_{Cyc} = 0.79$, $AP_{PushObj} = 5.17\%$ and $\hat{y}_{PushObj} = 0.52$. For the AP post-processing, as $AP_{Ped} + AP_{PushObj} = 74.83\% > 70.81\% = AP_{Cyc}$, the cyclist label is flipped. For the AP×O post-processing, as $AP_{Ped} \cdot \hat{y}_{Ped} + AP_{PushObj} \cdot \hat{y}_{PushObj} = 41.70\% < 55.94\% = AP_{Cyc} \cdot \hat{y}_{Cyc}$, the pedestrian and pushing object labels are flipped. Overall, we see that these post-processings are not very efficient for this example, but that the AP×O one is better than the AP one here as it at least found that the cyclist on the left was indeed a cyclist.

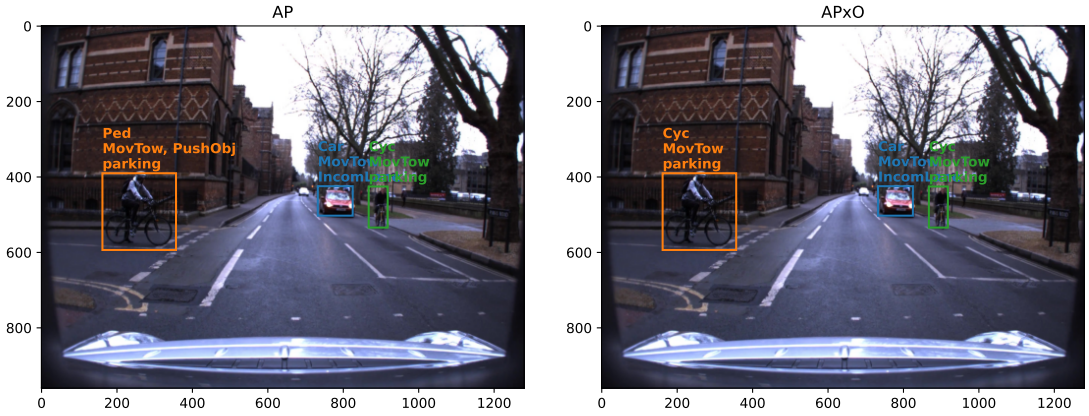


Figure 4.6: AP and AP×O post-processings applied on the raw prediction of the new example.

Let us now have a look at the *pred-based* and the new *log-pred-based* post-processings for this example. Figure 4.7 below shows their results. The first thing we can notice is that now for both cyclists, both post-processings set the missing location labels to the correct labels. This is because, as explained before, they were the unpredicted location labels with the highest confidence scores, which were thus the closest to the threshold and had the lowest weights. Thanks to that, the cyclist on the right is now correctly classified with respect to the ground truth. Then, for the cyclist on the left, we can notice that the *pred-based* post-processing chose to only flip the cyclist label to negative, while the *log-pred-based* one chose to flip the pedestrian and pushing object labels to negative. Obviously, even if the action label of moving toward remains incorrect in both cases, saying that the cyclist is indeed a cyclist is better than saying that it is a pedestrian that pushes an object. The *log-pred-based* is thus better in this case, but let us analyze why. For the *pred-based* post-processing, we have $\hat{y}_{Ped} + \hat{y}_{PushObj} = 1.08 > 0.79 = \hat{y}_{Cyc}$, so the cyclist label is flipped to negative. For the *log-pred-based* post-processing, let $f(\hat{y}_i) = \frac{-1}{\log(\hat{y}_i)}$. We then have $f(\hat{y}_{Ped}) + f(\hat{y}_{PushObj}) = 3.25 < 4.24 = f(\hat{y}_{Cyc})$, so the pedestrian and pushing object labels are flipped negative this time. Thus we see that in some cases, the *log-pred-based* post-processing prefers flipping several lower confidence labels than a high confidence label, whereas the sum of the lower confidences would basically have been higher than the high confidence as it is the case here with the *pred-based* post-processing.

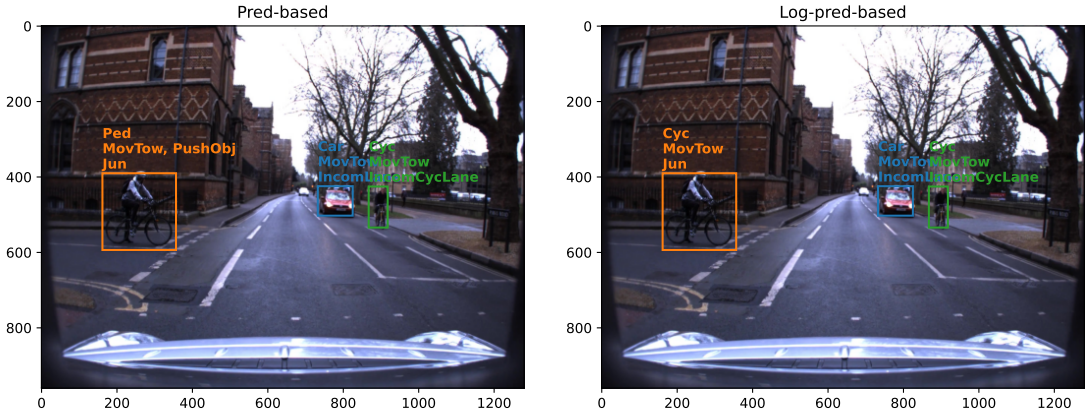


Figure 4.7: *pred-based* and *log-pred-based* post-processings applied on the raw prediction of the new example.

Chapter 5

Experiments and Analysis

We will now describe the experiments we carried out for the various contributions proposed in the previous chapter. For each experiment, we will give the parameters used and show the results obtained for each set of parameters, using different metrics like the mAP score and the number of predictions that are not compliant with the requirements. Of course, we will also analyze and discuss our results. But before presenting the results of interest, we will first present our setup and baseline.

5.1 Setup

From now on, all the models that we will train will have the same base parameters, which are basically almost the same as the ones the authors of the ROAD-R paper used. In particular, they will have a batch size equal to 4 and a sequence length equal to 8 frames. They will all be initialized with the Kinetics-400[1] pre-trained weights, and they will use an SGD optimizer [22] with step learning rate, starting from 0.0041. They will be trained for 30 epochs (without fine-tuning), during which the learning rate will drop by a factor of 10 after the 20^{th} and 25^{th} epochs. Training a 3D-RetinaNet on the ROAD-R dataset is computationally very expensive, and my own computer was not powerful enough to support the training of such models. Therefore, we used the Manneback cluster from the CISM, the UCLouvain platform for supercomputing which is part of the CÉCI consortium. In particular, we used a node with 96 CPUs, and 4 GeForce RTX3090 GPUs having 24GB of VRAM each. Even with this equipment, training a 3D-RetinaNet model on the ROAD-R dataset for one epoch takes around one hour. So, a complete training for 30 epochs takes about 30 hours, which is quite long. Therefore, we will limit ourselves only to 3D-RetinaNet models using an **RCGRU backbone CNN**, and a **Łukasiewicz t-norm** for the constrained loss, which were the configurations giving the best

[1] `https://deepmind.google/`

results in the ROAD-R paper.

Before moving on to the experiments, we have a last issue to address. As already said in section 3.2.1, there are normally 22 annotated videos in the ROAD-R dataset. The authors of the ROAD-R paper thus separated this dataset into 18 training videos, and 4 test videos. However, they did not make their annotations on the test videos publicly available. Therefore, if we want to be able to evaluate our models, we will have to split the annotated training set of 18 videos into a new smaller training set, and a new test set. Our new training set is now composed of 15 videos, while the test set is made up of the 3 remaining annotated videos. This could lead to small differences in the results obtained, compared with those obtained in the ROAD-R paper.

5.2 Baseline

First and foremost, we need baseline results with which we can compare all our future experimental results. To do so, we will re-train models by picking among those already discussed in the ROAD-R paper. We will train a baseline model that does not take the requirements into account, and we will also train baseline models that take them into account either with a constrained loss, a constrained output, or even both. We will compare their results with the ones in the paper, and then they will then serve as a basis against which to compare our new models.

5.2.1 Configurations

Our most basic baseline model will thus be a 3D-RetinaNet model with RCGRU backbone, without constrained loss or output. Then, we will train baseline CL models, using the Łukasiewicz t-norm, with different values for the α hyperparameter: $\alpha = 1.0$, $\alpha = 5.0$ and $\alpha = 10.0$. Finally, we will also develop CO and CLCO models by applying the post-processings presented in the ROAD-R paper (MD, AP, and AP×O) to all trained baseline models, with different values for the classification threshold θ ranging from 0.1 to 0.9 with steps of 0.1.

5.2.2 Results

Basic and CL models

Figure 5.1 below shows the percentage of predictions violating at least one constraint for our basic ($\alpha = 0.0$) and CL ($\alpha > 0.0$) baseline models, with respect to the classification threshold θ. We computed these values only for bounding boxes that had at least one predicted label, i.e. at least one label score above θ, since we do

not consider the bounding boxes without any predicted label in the final outputs of the models. We can observe U-shaped curves, with minimal percentages around $\theta = 0.3$ and $\theta = 0.4$. We can explain these curve shapes by the fact that at a smaller threshold θ, more labels are predicted per bounding box, and therefore more constraints such as those of mutual exclusion are not respected. Furthermore, at a higher threshold θ, fewer labels are predicted per bounding box, and so more constraints such as those requiring each bounding box to have at least one agent label and one location label (except for traffic lights) are not respected. We can thus conclude that around $\theta = 0.3$ and $\theta = 0.4$, a trade-off is found between the non-respect of these different types of constraints.

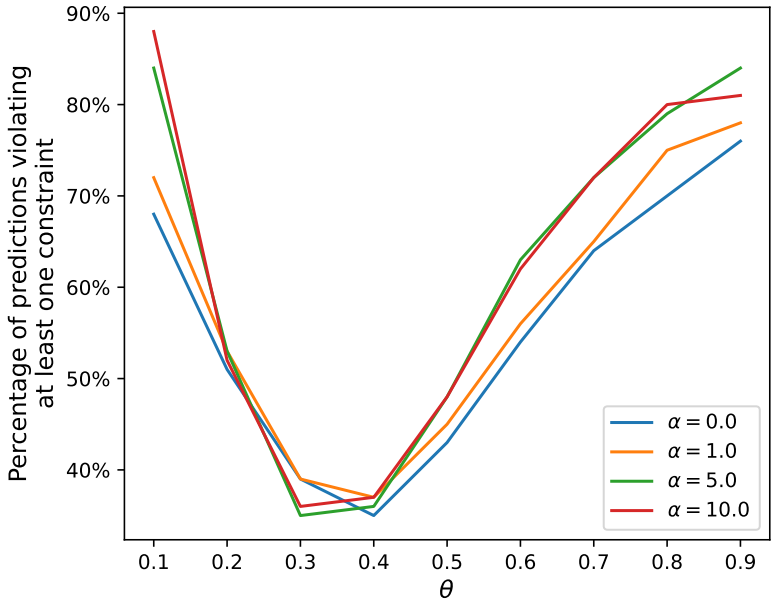


Figure 5.1: Percentage of predictions violating at least one constraint for baseline basic and CL models, with respect to the classification threshold.

A surprising fact, but that was also observed in the ROAD-R paper, is that integrating the constrained loss does not seem to reduce the percentage of predictions violating at least one constraint, compared to the basic model. Even more surprising, we can observe that the curve of the basic model is almost always under the other three (except for $\theta = 0.3$). However, since taking into account the degree of constraints non-compliance in the loss allows background knowledge associated with these constraints to be integrated into the model, we might have expected CL models to have lower percentages of predictions violating at least one constraint

than the basic model. We might also have expected that the higher the α, the lower the percentages would be, since the constrained loss regularization term, and thus the degree of respect of the constraints, would then have more importance in the loss and therefore in the training process. Nonetheless, we can see that this is not the case here. Note that our results differ from the ones presented in the ROAD-R paper in one aspect. Indeed, in the latter, we can observe curves with percentages that evolve almost linearly with θ, from about 90% at $\theta = 0.1$ to almost 100% at $\theta = 0.9$. We do not have access to their code for this metric, but a possible explanation could be that, unlike us, they used all the predictions in their computations, including the ones where all label scores were below θ and thus where nothing was predicted, of which there are many. Since there exist constraints that prohibit bounding boxes without any predicted label, this would explain the large number of predictions violating at least one constraint, as well as the linear and increasing shape of the curves, as the number of bounding boxes without any predicted label naturally increases with θ.

Table 5.1 below shows the mAP@50 scores of the basic and CL baseline models. We can see that only the CL model with $\alpha = 1.0$ increased the performance of the basic model, raising the mAP@50 score from 33.03% to 33.86%, which represents a significant improvement. However, CL models with $\alpha = 5.0$ and $\alpha = 10.0$ induced a drop in performance compared to the basic model. This may be due to the fact that, with these values of α, the constrained loss regularization term had too much weight compared to the focal loss terms, which did not give enough importance to the loss terms related to the regression of the bounding boxes and their classification, which are the two main ingredients to compute the mAP score of a model.

α	0.0	1.0	5.0	10.0
mAP@50 [%]	33.03	33.86	32.73	32.55

Table 5.1: mAP@50 scores of basic and CL baseline models.

The scores obtained here are overall slightly higher than in the ROAD-R paper but remain within the same ranges. This small difference is probably due to the fact that we did not use exactly the same training and test sets as they did. However, we can also observe in the ROAD-R paper that the different CL models can have both a positive as well as a negative impact on the performance. In our case, as in the paper, our best CL model outperforms our basic model by a good margin.

CO and CLCO models

Table 5.2 below reports the different mAP@50 scores (in [%]) of our baseline CO models, obtained by applying different post-processings on the basic model. The best score(s) with respect to θ for each type of post-processing is (are) put in bold for greater clarity. We notice that none of these post-processings improved the performance of the basic model.

		$\alpha = 0.0$		
		post-processing		
		MD	AP	AP×O
	0.1	27.75	29.86	31.46
	0.2	30.93	31.74	32.18
	0.3	32.18	32.43	32.57
	0.4	32.67	32.83	32.83
θ	0.5	**32.69**	**32.86**	**32.86**
	0.6	32.59	**32.86**	32.84
	0.7	32.37	32.73	32.71
	0.8	32.19	32.79	32.79
	0.9	31.98	32.77	32.87

Table 5.2: mAP@50 scores of baseline CO models.

For the MD post-processing, it seems normal as it is a very naive post-processing that just seeks to make a prediction compliant with the requirements by flipping as few labels as possible, without taking the context or the performance of the model into account. We also observed this decrease in performance in the ROAD-R paper. However, in the paper, the performances of the best AP and AP×O post-processings were always better than for the basic model, which is not the case here. A possible explanation of this difference is that in our case, as illustrated in the examples of section 4.3, certain labels have a very low AP compared to the other labels, which means that they have a very low weight in the weighted PMaxSAT problem. They are therefore very often flipped "by default" in predictions where agent or location labels are missing to meet the constraints, leading to wrong predictions and thus to a negative impact on the performance. We can then assume that in the ROAD-R paper, due to the fact that they used different training and test sets, the APs of these labels were surely higher than in our case, which limited this phenomenon of "flipped by default" labels, especially in the case of the AP×O post-processing which also takes the model confidence into account. Note that at least, despite a slight loss of performance, we can be sure that all predictions are compliant with the requirements.

Let us now look at table 5.3 below. This one now reports the mAP@50 scores (in [%]) of our baseline CLCO models, obtained by applying different post-processings on each of the baseline CL models. Once again, we notice that none of these post-processings improved the performance of the baseline CL models and that, in each case, the MD post-processing has the worst performance. The loss of performance of the AP and AP×O post-processings can be explained the same way as just before.

		$\alpha = 1.0$			$\alpha = 5.0$			$\alpha = 10.0$		
		post-processing			post-processing			post-processing		
		MD	AP	AP×O	MD	AP	AP×O	MD	AP	AP×O
	0.1	28.24	30.52	32.67	25.98	28.10	30.94	25.77	28.32	30.87
	0.2	31.85	33.01	33.31	30.22	31.16	31.92	30.66	31.33	31.81
	0.3	33.25	33.59	33.56	32.11	32.25	32.37	**32.35**	32.34	32.38
	0.4	**33.59**	33.76	**33.70**	**32.44**	**32.63**	**32.63**	32.32	32.44	**32.43**
θ	0.5	33.45	**33.77**	33.64	32.27	32.61	32.61	31.99	32.42	32.34
	0.6	33.30	33.73	33.64	32.01	32.61	32.61	31.74	32.45	32.38
	0.7	33.11	33.74	33.66	31.79	32.60	32.60	31.49	**32.46**	32.38
	0.8	32.72	33.71	33.61	31.55	32.60	32.62	31.01	**32.46**	32.36
	0.9	32.53	33.72	33.67	31.22	32.61	32.61	31.20	32.44	32.39

Table 5.3: mAP@50 scores of baseline CLCO models.

Again, despite slight losses of performance compared to their respective CL models, we can at least be sure that all predictions of these CLCO models are compliant with the requirements. Moreover, the CLCO model with $\alpha = 1.0$ for the constrained loss shows greater mAP@50 scores than the baseline basic model. It means that, in addition to having all its predictions compliant with the requirement, it also had a better performance than the basic model, which is a great improvement. However, it is still not the best model in terms of pure performance since the CL model with $\alpha = 1.0$ without post-processing achieves a greater mAP@50 score.

Now that we presented all our baseline experiments and analyzed their results, let us do the same with the experiments featuring our proposed contributions.

5.3 New post-processings

We will begin with the experiments using our proposed *pred-based* and *Log-pred-based* post-processings on the baseline models. The objective of these experiments is to see if our new post-processings could give better results than the ones presented in the ROAD-R paper, that form our baseline.

5.3.1 Configurations

In these experiments, we will create new CO and CLCO models by applying our proposed *pred-based* (P) and *Log-pred-based* (LP) post-processings to all trained basic ($\alpha = 0.0$) and CL ($\alpha \in \{1.0, 5.0, 10.0\}$) baseline models, with different values for the classification threshold θ ranging from 0.1 to 0.9 with steps of 0.1.

5.3.2 Results

Table 5.4 below shows the mAP@50 scores of all our new CO and CLCO models. In addition to bolding the best score(s) of each post-processing with respect to θ, we colored in green the scores that are greater than their corresponding baseline scores, without post-processing. A first thing we can notice is that, for each $\alpha \in \{0.0, 1.0, 5.0, 10.0\}$, P and LP post-processings' best scores are greater than the best scores of the other baseline post-processings presented just before. Secondly, unlike just before, we can now observe performance improvements compared with the basic and CL baseline models, as shown by the mAP@50 scores highlighted in green, for each α except for $\alpha = 1.0$. These two observations mean two things: in our case, our proposed post-processings seem better than the ones proposed in the ROAD-R paper, and for each α except 1.0, we improved the model's performance while having all predictions compliant with the requirements.

		$\alpha = 0.0$		$\alpha = 1.0$		$\alpha = 5.0$		$\alpha = 10.0$	
		post-processing		post-processing		post-processing		post-processing	
		P	LP	P	LP	P	LP	P	LP
	0.1	31.92	31.89	32.70	32.70	31.54	31.48	31.49	31.37
	0.2	32.42	32.49	33.47	33.44	32.36	32.36	32.30	32.29
	0.3	32.63	32.86	33.70	33.73	32.66	32.69	**32.56**	**32.60**
	0.4	32.78	33.03	**33.82**	**33.83**	**32.70**	**32.75**	32.47	32.56
θ	0.5	**32.88**	**33.10**	33.76	33.82	32.44	32.46	32.26	32.37
	0.6	32.74	32.97	33.41	33.46	32.30	32.40	32.06	32.16
	0.7	32.48	32.62	33.28	33.31	32.16	32.22	31.82	31.91
	0.8	32.37	32.36	32.79	32.81	32.32	32.41	31.74	31.78
	0.9	32.48	32.45	32.74	32.74	32.47	32.48	32.26	32.28

Table 5.4: mAP@50 scores of CL and CLCO models with our proposed post-processings.

Unfortunately, the baseline CL model with $\alpha = 1.0$ was precisely the most performant one, and our new post-processings did not improve it. This means that, for the moment, our best model in terms of pure performance is not a model whose

predictions are always compliant with the requirements. However, still for $\alpha = 1.0$, we can see that the difference between the best score of the LP post-processing and the score of the baseline CL model is very small, amounting to just 0.03%. So, if we absolutely want to have only predictions that respect all the constraints, it should not be too big deal to use the CLCO model with $\alpha = 1.0$ and an LP post-processing with a threshold $\theta = 0.4$, instead of the very slightly most performant CL model with $\alpha = 1.0$ which does not ensure that all constraints are respected for each prediction.

5.4 Fine-tuning

We will now present the experiments using fine-tuning. As a reminder, this consists of resuming the training of an already trained (and thus functional) model that does not take constraints into account, this time integrating the constrained loss to it for just a few iterations. The objective of the following experiments is thus to see if it is possible to increase the performance of an existing model, thanks to this fine-tuning that allows the model to adapt to the requirements for a few iterations. If so, this would allow to save great amounts of time, as we would not have to re-train a model entirely.

5.4.1 Configurations

We will fine-tune the basic baseline model, that was trained without constrained loss for 30 epochs. To achieve that, we will resume its training for 5 additional epochs, during which we will this time use a constrained loss with the Łukasiewicz t-norm. As training a model for one epoch takes around one hour, each fine-tuning takes about 5 hours, which is not much compared to the 30 hours of training for a fully trained model. Once again, we will use different values for the α hyperparameter: $\alpha = 1.0$, $\alpha = 5.0$, and $\alpha = 10.0$. Finally, we will apply all the previous post-processings to these fine-tuned models, including those in the ROAD-R paper and those proposed by us, with different values for the classification threshold θ ranging from 0.1 to 0.9 with steps of 0.1.

5.4.2 Results

Figure 5.2 below shows the percentage of predictions violating at least one constraint for our different fine-tuned models, with respect to the classification threshold θ. We also included the curve of the baseline basic model back in the graph for easier comparison. We can observe U-shaped curves similar to those observed previously, which is logical as we saw that there was not much difference between CL and

non-CL models in this regard. It therefore seems normal that partial CL models have similar behaviors and values. The same explanations and analyses as before thus still apply to these curves.

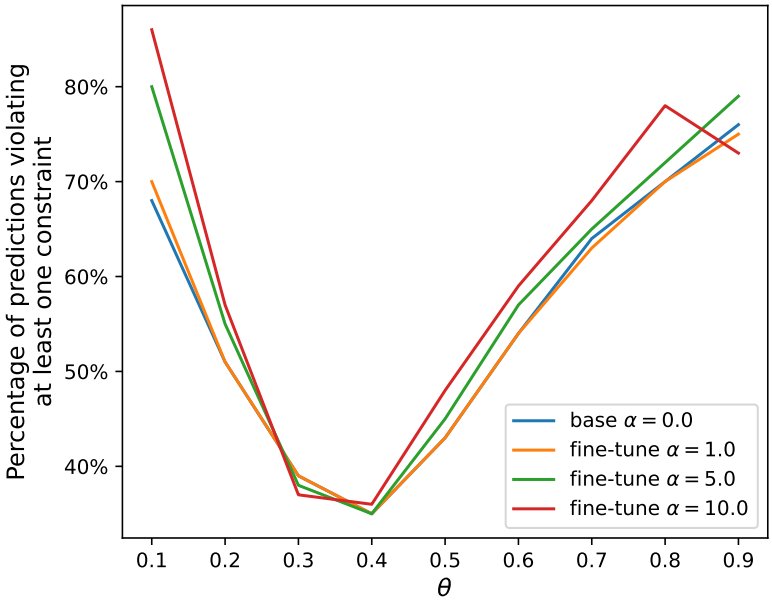


Figure 5.2: Percentage of predictions violating at least one constraint for fine-tuned models, with respect to the classification threshold.

However, we can see that the curve for the fine-tuned model with $\alpha = 1.0$ almost confuses itself with the curve for the basic model. It could suggest that this fine-tuning may not have had a great impact on the original model. This could be understandable, given that this is the smallest value of α that we use, and that the fine-tuning only lasted for 5 more epochs on the basic model. Let us now thus analyze their performances.

Table 5.5 below shows the mAP@50 scores of the fine-tuned models. Unfortunately, we can notice that all three have had a detrimental effect on the performance of the basic model, which initially had a mAP@50 score of 30.03%. However, we can observe that the larger α is, the higher the performance of the fine-tuned model. Besides, the mAP@50 score for $\alpha = 10.0$ is close to the score of the basic model. We can therefore believe that certain values of α that we did not test, possibly coupled with different numbers of additional epochs (other than 5), could potentially have produced fine-tuned models with an improved performance compared to the basic

model. Nonetheless, we certainly cannot state this with certainty, as we did not observe it in our results. We were indeed limited in computational and storage resources, which is why we were not able to go much further in these fine-tuning experiments.

α	1.0	5.0	10.0
mAP@50 [%]	32.55	32.79	32.91

Table 5.5: mAP@50 scores of fine-tuned models.

Let us now see the results of applying all post-processings to these fine-tuned models. Tables 5.6, 5.7 and 5.8 below show the mAP@50 scores of each post-processed fine-tuned model, for each value of α, each type of post-processing and each value of θ. We can observe some results similar to those obtained before. In particular, we can again see that our proposed post-processings always perform better than those presented in the ROAD-R paper, as their best scores in function of θ (still indicated in bold) are always greater than the best scores of those of the ROAD-R paper. Furthermore, we see that our LP post-processing increases the performance of fine-tuned models with $\alpha = 1.0$ and $\alpha = 5.0$. However, it does not increase the performance of the fine-tuned model with $\alpha = 10.0$.

		$\alpha = 1.0$				
		post-processing				
		MD	AP	AP×O	P	LP
	0.1	27.11	29.47	31.31	31.81	31.76
	0.2	30.43	31.49	32.09	32.28	32.29
	0.3	31.70	32.19	32.35	32.48	32.52
	0.4	**32.24**	32.46	**32.48**	**32.53**	**32.58**
θ	0.5	32.20	**32.47**	32.45	32.50	**32.58**
	0.6	32.13	32.45	32.42	32.43	32.48
	0.7	31.87	32.39	32.37	31.93	32.00
	0.8	31.71	32.39	32.38	31.83	31.91
	0.9	31.51	32.37	32.35	31.84	31.85

Table 5.6: mAP@50 scores of post-processed fine-tuned models with $\alpha = 1.0$.

		$\alpha = 5.0$				
		post-processing				
		MD	AP	AP×O	P	LP
	0.1	26.08	28.71	31.12	31.83	31.78
	0.2	30.30	31.49	32.17	32.45	32.41
	0.3	32.26	32.53	32.65	**32.78**	32.76
	0.4	**32.54**	**32.74**	**32.75**	**32.78**	**32.81**
θ	0.5	32.44	32.71	32.72	32.69	32.73
	0.6	32.25	32.68	32.68	32.34	32.43
	0.7	32.07	32.66	32.65	31.99	32.06
	0.8	31.85	32.64	32.63	31.89	31.91
	0.9	31.56	32.64	32.64	32.12	32.12

Table 5.7: mAP@50 scores of post-processed fine-tuned models with $\alpha = 5.0$.

		$\alpha = 10.0$				
		post-processing				
		MD	AP	AP×O	P	LP
	0.1	25.04	27.85	30.89	31.77	31.69
	0.2	30.08	31.35	32.07	32.51	32.46
	0.3	32.41	32.67	32.76	**32.87**	32.85
	0.4	**32.66**	**32.85**	**32.86**	**32.87**	**32.90**
θ	0.5	32.54	32.82	32.83	32.76	32.78
	0.6	32.31	32.79	32.77	32.29	32.40
	0.7	32.07	32.79	32.77	32.11	32.17
	0.8	31.66	32.77	32.76	31.98	31.99
	0.9	31.73	32.79	32.78	32.52	32.54

Table 5.8: mAP@50 scores of post-processed fine-tuned models with $\alpha = 10.0$.

Overall, whether with or without post-processing, the performance of our fine-tuned models does not reach the performance of the baseline basic model and even less the one of the baseline CL model with $\alpha = 1.0$. For the latter, it is not a very big deal since it is not the one that was fine-tuned; it was therefore not really the reference for this experiment. However, we hoped and expected an increase in performance compared to the basic model before fine-tuning since, in our opinion, the fine-tuning should have produced almost the same model but improved by the integration of additional background knowledge thanks to the consideration of the degree of compliance with the requirements in the training.

5.5 Discussion

Before discussing our results, here is a summary of them:

- First of all, our most basic baseline model (without CL and CO) obtained a mAP@50 score of 33.03%.
- Concerning our baseline CL models, we observed that they did not reduce the number of predictions violating at least one constraint, compared to the basic model. Furthermore, only the CL model with $\alpha = 1.0$ scored better than the basic model, achieving a mAP@50 score of 33.86%.
- None of our baseline CO and CLCO models improved the performance of the corresponding baseline basic and CL models without post-processing.
- For each model, our P and LP post-processings always gave better results than the baseline post-processings (MD, AP, and AP×O, presented in the ROAD-R paper). In addition, we observed performance improvements for the baseline basic model, rising to a mAP@50 score of 33.10% with the LP post-processing with $\theta = 0.5$, as well as for the baseline CL models with $\alpha = 5.0$ and $\alpha = 10.0$. However, this was not the case for the baseline CL model with $\alpha = 1.0$, for which we were still able to obtain a map@50 score of 33.83% with the LP post-processing with $\theta = 0.4$.
- Finally, none of our fine-tuned models was able to reduce the number of predictions violating at least one constraint and, whether with or without post-processing, none of them improved the performance of the basic baseline model from which they were created. By applying all post-processings to them, however, we were able to make observations similar to the previous point: our proposed post-processings obtained better scores than those presented in the ROAD-R paper, and some LP post-processings were able to improve the score compared with no post-processing.

In our case, there are therefore two possible choices for the final model, depending on the main objective sought. If we want to have the best possible model in terms of pure performance (reflected by the mAP@50 score) without having the certainty that all its predictions will be compliant with the requirements, we will choose the CL model with $\alpha = 1.0$. On the other hand, if we want to have the assurance that all the predictions always respect all the constraints, we will prefer to use the CLCO model with $\alpha = 1.0$ and the LP post-processing with $\theta = 0.4$, which has a slightly lower performance but remains the best-performing model with a constrained output.

To return to the comparison with the results obtained in the ROAD-R paper concerning post-processings, they always observed an increase in performance by applying AP and AP×O post-processings to a non-CO model, which was not the case at all here. However, our P and LP post-processings always showed better performance than the AP and AP×O post-processings, and in most cases, they were even able to increase the performance of models without post-processing, especially for the LP one (which was by the way always better than the P one). The results of our proposed post-processings were therefore conclusive here, and we thus believe that they could be reused in other deep learning applications where compliance with the requirements is critical.

Finally, regarding fine-tuning, our results were unfortunately not positive, since none of our fine-tunings improved the performance of the baseline basic model. However, due to the fact that we had limited computational and storage resources, we were not able to test many different configurations. Indeed, it could have been interesting to perform fine-tuning with other values of α, or other numbers of additional epochs. We cannot say with certainty that other new configurations would improve performance. However, if this was the case, this method would constitute a good way to improve existing deep learning models for other applications, in a short time compared to completely re-training a model (especially for bigger ones). This would involve the creation of a set of requirements expressing background knowledge specific to the application, and then resuming the training of the model with added constrained loss regarding these new requirements.

Chapter 6

Conclusion

As stated in our introduction, DNNs are nowadays widely used for various tasks, of various degrees of complexity. However, despite their impressive performance, they can act against requirements expressing background knowledge, which can be critical in certain sensitive contexts. Researchers then created ROAD-R, a video dataset for autonomous driving equipped with a set of requirements in the form of logical constraints. They showed that it was possible to exploit these requirements in order to improve a road event detection model and to guarantee compliance with themselves. This master's thesis was quite inspired by their work and its main objectives also went in this direction, but with different new approaches.

After having introduced the theoretical notions needed to understand this work, covering the topics of logic and deep learning with a focus on CNNs, we went more deeply into the understanding of ROAD-R and the methods used to perform road event detection as well as exploit the requirements in this regard. Indeed, we first explained how the 3D-RetinaNet model works, combining an FPN acting as a feature extractor with two CNNs specialized respectively in bounding box regression and classification, and how to evaluate it using the mAP metric. Then, we explained the two ways in which the authors of the ROAD-R paper integrated the set of requirements into the model.

The first one, called constrained loss (CL), consisted of adding a term to the loss function, representing the degree of violation of the constraints by the predictions, which allowed to take the requirements into account during training. This allowed to integration of background knowledge into the model, which could lead to a greater or lesser performance gain, depending on the case. The second one, called constrained output (CO), consisted of applying a post-processing step to the predictions, by solving a weighted PMaxSAT problem with the requirements as hard constraints and the predicted labels as soft constraints. This allowed to make the

predictions compliant with the requirements and could bring a small performance gain depending on the weights chosen for the soft constraints. These two methods could also be combined to form CLCO models which, in addition to producing predictions compliant with the requirements, often showed the best performance.

The positive results of these methods inspired us to set ourselves two objectives. The first was to investigate whether it is possible to improve an already trained model using the requirements, without having to completely re-train it to save time. The second objective was to improve the post-processing step that makes the predictions compliant with the requirements. Before carrying out the experiments associated with these two objectives, we trained some basic (i.e. without CL or CO), CL, CO, and CLCO models as in the ROAD-R paper, serving as a baseline against which to compare our future results. At that point, we observed some dissimilarities with the ROAD-R paper, notably in the CO and CLCO models which never improved performance in our case, unlike in the paper.

Next, in an attempt to achieve the first objective, we resumed the training of the basic model, adding the constrained loss term to the total loss for a small number of epochs compared with their initial number. We performed this fine-tuning process with different weights for the constrained loss term. However, our results were not very conclusive as they degraded the performance of the initial model.

In an attempt to achieve the second objective, we proposed two new choices of weights for the soft constraints of the weighted PMaxSAT problem, based on the model's confidence in its predictions, and we applied them to the different baseline (non-CO) models. They were called P and LP post-processings, and were applied for different classification thresholds. Despite the aforementioned dissimilarities between the results of the ROAD-R paper and our baseline results, we observed that our proposed P and LP post-processings were still always better than those of the ROAD-R paper, and that the LP one was itself always better than the P one. We also observed that they often led to a performance gain compared to the models on which they were applied, except in certain cases like for one of the baseline CL models, which was eventually the best-performing model in terms of mAP. Nonetheless, the difference with its LP-post-processed version was very small, and we can be sure that the latter always respects the constraints.

For these reasons, we consider that the second objective has been achieved as well and that the new post-processings that we have proposed could be used in other deep learning applications where the respect of certain constraints is primary. However, we were surprised by our fine-tuning results, and believe that experiments

should be carried out to a greater extent, with several parameter configurations notably in terms of the weight given to the constrained loss term and the number of additional training epochs. Unfortunately, we were not able to do this because we were limited in computational and storage resources. Nonetheless, if the results prove positive, this would be a good way of improving, in a relatively small amount of time, models of other applications that would otherwise take a long time to train entirely, by creating a set of requirements expressing application-specific background knowledge. We will thus leave this for future work.

Finally, concerning post-processing, we limited ourselves to simple functions for the weights of the soft constraints in the weighted PMaxSAT problem. We can then imagine that more complex functions, for example taking into account the different labels and their different types, could achieve better performances if they were well designed. Moreover, during our work, we noticed that some specific constraints were more often violated than others. As a result, we could imagine post-processing that adapts to such constraints to give better results. Concerning the constraints themselves, we saw that in our case they were expressed in propositional logic. However, there are other, more complete types of logic, such as first-order logic. It might therefore be interesting to see whether it would be possible to generalize the methods used here to these other, more complete types of logic, which would make it possible to model a wider variety of constraints. All these ideas could then be explored in future work.

In conclusion, this master's thesis proposed some methods to improve deep learning models thanks to requirements expressed as logical constraints. Although some challenges remain such as optimizing the fine-tuning process, it demonstrated the good performance of new promising post-processing methods ensuring compliance with the requirements, and provided valuable insights for future work in the field.

Bibliography

[1] Carlos Ansotegui, Maria Luisa Bonet, and Jordi Levy. "A New Algorithm for Weighted Partial MaxSAT". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 24.1 (July 2010), pp. 3–8. DOI: 10.1609/aaai.v24i1.7545. URL: https://ojs.aaai.org/index.php/AAAI/article/view/7545.

[2] Christopher M Bishop. *Pattern recognition and machine learning by Christopher M. Bishop*. Springer Science+ Business Media, LLC, 2006.

[3] Navaneeth Bodla et al. "Improving Object Detection With One Line of Code". In: *CoRR* abs/1704.04503 (2017). arXiv: 1704.04503. URL: http://arxiv.org/abs/1704.04503.

[4] João Carreira and Andrew Zisserman. "Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset". In: *CoRR* abs/1705.07750 (2017). arXiv: 1705.07750. URL: http://arxiv.org/abs/1705.07750.

[5] Kai Chen et al. "MMDetection: Open MMLab Detection Toolbox and Benchmark". In: *arXiv preprint arXiv:1906.07155* (2019).

[6] Lucas Costa et al. "Multilayer perceptron". In: *Introduction to Computational Intelligence* 105 (2023).

[7] Jessica Davies and Fahiem Bacchus. "Exploiting the power of MIP solvers in MAXSAT". In: *Theory and Applications of Satisfiability Testing–SAT 2013: 16th International Conference, Helsinki, Finland, July 8-12, 2013. Proceedings 16*. Springer. 2013, pp. 166–181.

[8] Jessica Davies and Fahiem Bacchus. "Solving MAXSAT by solving a sequence of simpler SAT instances". In: *International conference on principles and practice of constraint programming*. Springer. 2011, pp. 225–239.

[9] PyTorch 2.4 documentation. *SmoothL1Loss*. URL: https://pytorch.org/docs/stable/generated/torch.nn.SmoothL1Loss.html.

[10] Niklas Eén and Niklas Sörensson. "An Extensible SAT-solver". In: *Theory and Applications of Satisfiability Testing*. Ed. by Enrico Giunchiglia and Armando Tacchella. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 502–518. ISBN: 978-3-540-24605-3.

[11] Christoph Feichtenhofer et al. “SlowFast Networks for Video Recognition”. In: *CoRR* abs/1812.03982 (2018). arXiv: 1812.03982. URL: http://arxiv.org/abs/1812.03982.

[12] Ross B. Girshick. “Fast R-CNN”. In: *CoRR* abs/1504.08083 (2015). arXiv: 1504.08083. URL: http://arxiv.org/abs/1504.08083.

[13] Eleonora Giunchiglia et al. “ROAD-R: the autonomous driving dataset with logical requirements”. In: *Machine Learning* 112.9 (May 2023), pp. 3261–3291. ISSN: 1573-0565. DOI: 10.1007/s10994-023-06322-z. URL: http://dx.doi.org/10.1007/s10994-023-06322-z.

[14] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. Cambridge, MA, USA: MIT Press, 2016.

[15] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[16] Thibault Helleputte. *LINFO2262 – Machine Learning Classification and Evaluation*. Lecture slides. Course material. ICTEAM Institute, UCLouvain – Belgium. 2023.

[17] Yuxiu Hua et al. “Traffic Prediction Based on Random Connectivity in Deep Learning with Long Short-Term Memory”. In: *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*. 2018, pp. 1–6. DOI: 10.1109/VTCFall.2018.8690851.

[18] Jonathan Hui. *Understanding Feature Pyramid Networks for Object Detection (FPN)*. Apr. 2020. URL: https://jonathan-hui.medium.com/understanding-feature-pyramid-networks-for-object-detection-fpn-45b227b9106c.

[19] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

[20] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. *Ultralytics YOLO*. Version 8.0.0. Jan. 2023. URL: https://github.com/ultralytics/ultralytics.

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

[22] Yann A. LeCun et al. “Efficient BackProp”. In: *Neural Networks: Tricks of the Trade: Second Edition*. Ed. by Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 9–48. ISBN: 978-3-642-35289-8. DOI: 10.1007/978-3-642-35289-8_3. URL: https://doi.org/10.1007/978-3-642-35289-8_3.

[23] Kanokphan Lertniphonphan. *Road_event*. 2023. URL: https://github.com/KanokphanL/Road_event/tree/road_r?tab=readme-ov-file.

[24] Tsung-Yi Lin et al. “Feature Pyramid Networks for Object Detection”. In: *CoRR* abs/1612.03144 (2016). arXiv: 1612.03144. URL: http://arxiv.org/abs/1612.03144.

[25] Tsung-Yi Lin et al. “Focal Loss for Dense Object Detection”. In: *CoRR* abs/1708.02002 (2017). arXiv: 1708.02002. URL: http://arxiv.org/abs/1708.02002.

[26] Liu Linkai. *ROAD-R-NIPS2023*. 2023. URL: https://github.com/LIULINKAI/ROAD-R-NIPS2023.

[27] Estelle Massart. *LINMA2472 – Algorithms in Data Sciences*. Lecture slides. Course material. INMA Institute, UCLouvain – Belgium. 2023.

[28] George Metcalfe. “Fundamentals of Fuzzy Logics”. In: URL: https://api.semanticscholar.org/CorpusID:17549644.

[29] MMDetection Contributors. *OpenMMLab Detection Toolbox and Benchmark*. Aug. 2018. URL: https://github.com/open-mmlab/mmdetection.

[30] Sota Moriyama. *MOD-CL*. 2023. URL: https://github.com/sotam2369/MOD-CL?tab=readme-ov-file.

[31] Sota Moriyama et al. *MOD-CL: Multi-label Object Detection with Constrained Loss*. 2024. arXiv: 2403.07885 [cs.CV]. URL: https://arxiv.org/abs/2403.07885.

[32] M.W. Moskewicz et al. “Chaff: engineering an efficient SAT solver”. In: *Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)*. 2001, pp. 530–535. DOI: 10.1145/378239.379017.

[33] Pariwat Ongsulee. “Artificial intelligence, machine learning and deep learning”. In: *2017 15th international conference on ICT and knowledge engineering (ICT&KE)*. IEEE. 2017, pp. 1–6.

[34] Rafael Padilla, Sergio L. Netto, and Eduardo A. B. da Silva. “A Survey on Performance Metrics for Object-Detection Algorithms”. In: *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*. 2020, pp. 237–242. DOI: 10.1109/IWSSIP48289.2020.9145130.

[35] Shaoqing Ren et al. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. In: *CoRR* abs/1506.01497 (2015). arXiv: 1506.01497. URL: http://arxiv.org/abs/1506.01497.

[36] Hamid Rezatofighi et al. “Generalized Intersection Over Union: A Metric and a Loss for Bounding Box Regression”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019.

[37] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson, 2020. ISBN: 9780134610993. URL: http://aima.cs.berkeley.edu/.

[38] Nafiz Shahriar. *What is Convolutional Neural Network - CNN (deep learning)*. Feb. 2023. URL: https://nafizshahriar.medium.com/what-is-convolutional-neural-network-cnn-deep-learning-b3921bdd82d5.

[39] Aditya Sharma. *Mean average precision (MAP) using the Coco Evaluator*. July 2022. URL: https://pyimagesearch.com/2022/05/02/mean-average-precision-map-using-the-coco-evaluator/.

[40] Gurkirt Singh and Fabio Cuzzolin. “Recurrence to the Rescue: Towards Causal Spatiotemporal Representations”. In: *CoRR* abs/1811.07157 (2018). arXiv: 1811.07157. URL: http://arxiv.org/abs/1811.07157.

[41] Gurkirt Singh et al. “ROAD: The ROad event Awareness Dataset for autonomous Driving”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 01 (Feb. 5555), pp. 1–1. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2022.3150906.

[42] Juan Terven et al. *Loss Functions and Metrics in Deep Learning*. 2023. DOI: 10.48550/ARXIV.2307.02694. URL: https://arxiv.org/abs/2307.02694.

[43] Axel Thevenot. *CONV2D: Finally understand what happens in the forward pass*. Feb. 2022. URL: https://towardsdatascience.com/conv2d-to-finally-understand-what-happens-in-the-forward-pass-1bbaafb0b148.

[44] Xiaolong Wang et al. “Non-local Neural Networks”. In: *CoRR* abs/1711.07971 (2017). arXiv: 1711.07971. URL: http://arxiv.org/abs/1711.07971.

[45] Taojiannan Yang et al. “Aim: Adapting image models for efficient video action recognition”. In: *arXiv preprint arXiv:2302.03024* (2023).

[46] Matthew D. Zeiler and Rob Fergus. “Visualizing and Understanding Convolutional Networks”. In: *CoRR* abs/1311.2901 (2013). arXiv: 1311.2901. URL: http://arxiv.org/abs/1311.2901.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**