UNICA

UNIVERSITÀ
DEGLI STUDI
DI CAGLIARI

**Ph.D. DEGREE IN**

Electronic and Computer Engineering

Cycle XXXV

# TITLE OF THE Ph.D. THESIS

Evaluating Adversarial Robustness of Detection-based

Defenses against Adversarial Examples

Scientific Disciplinary Sector(s)

ING-INF/05


| | |
|---|---|
| Ph.D. Student: | Angelo Sotgiu |
| Supervisor | Prof. Battista Biggio |
| Co-supervisor | Prof. Fabio Roli |



Final exam. Academic Year 2021/2022

Thesis defence: February 2023 Session

# Abstract

Machine Learning algorithms provide astonishing performance in a wide range of tasks, including sensitive and critical applications. On the other hand, it has been shown that they are vulnerable to adversarial attacks, a set of techniques that violate the integrity, confidentiality, or availability of such systems. In particular, one of the most studied phenomena concerns adversarial examples, i.e., input samples that are carefully manipulated to alter the model output. In the last decade, the research community put a strong effort into this field, proposing new evasion attacks and methods to defend against them.

With this thesis, we propose different approaches that can be applied to Deep Neural Networks to detect and reject adversarial examples that present an anomalous distribution with respect to training data.

The first leverages the domain knowledge of the relationships among the considered classes integrated through a framework in which first-order logic knowledge is converted into constraints and injected into a semi-supervised learning problem. Within this setting, the classifier is able to reject samples that violate the domain knowledge constraints. This approach can be applied in both single and multi-label classification settings.

The second one is based on a Deep Neural Rejection (DNR) mechanism to detect adversarial examples, based on the idea of rejecting samples that exhibit anomalous feature representations at different network layers. To this end, we exploit RBF SVM classifiers, which provide decreasing confidence values as samples move away from the training data distribution.

Despite technical differences, this approach shares a common backbone structure with other proposed methods that we formalize in a unifying framework. As all of them require comparing input samples against an oversized number of reference prototypes, possibly at different representation layers, they suffer from the same drawback, i.e., high computational overhead and memory usage, that makes these approaches unusable in real applications. To overcome this limitation, we introduce FADER (Fast Adversarial Example Rejection), a technique for speeding up detection-based methods by employing RBF networks as detectors: by fixing the number of required prototypes, their runtime complexity can be controlled.

All proposed methods are evaluated in both black-box and white-box settings, i.e., against an attacker unaware of the defense mechanism, and against an attacker who

knows the defense and adapts the attack algorithm to bypass it, respectively. Our experimental evaluation shows that the proposed methods increase the robustness of the defended models and help detect adversarial examples effectively, especially when the attacker does not know the underlying detection system.

# Contents

# List of Figures

# List of Tables

# Symbols

| | |
|---|---|
| $c$ | number of classes |
| $d$ | number of input dimensions |
| $f$ | decision function of the classifier |
| $\mathcal{K}$ | domain knowledge (FOL formulas) |
| $\mathcal{L}$ | training data |
| $\mathcal{T}$ | testing data |
| $\mathcal{V}$ | validation data |
| $\boldsymbol{x} \in \mathcal{X} \subseteq \mathbb{R}^d$ | input sample |
| $\boldsymbol{x}^\star$ | adversarial example |
| $y \in \mathcal{Y} \subseteq \mathbb{R}^c$ | sample label |
| $\hat{y}$ | predicted label |
| $\Gamma$ | supervision attached to (some) of the data in $\mathcal{L}$ |
| $\boldsymbol{\delta}$ | perturbation |
| $\varepsilon$ | upper bound on the norm of the difference between a clean example and its perturbed instance |
| $\eta$ | step size |
| $\theta$ | rejection threshold |
| $\Pi$ | projection operator |
| $\Omega$ | attacker loss |
| `suploss` | loss function, supervised data only |
| $\wedge$, $\vee$, $\neg$, $\Rightarrow$ | logical connectives |
| $\nabla_{\boldsymbol{x}} f$ | gradient of function $f$ w.r.t. $\boldsymbol{x}$ |

# Chapter 1

# Introduction

Machine learning can be included among the most impacting technologies of the last decade. Although the first research work in the area is not recent, after several years, it has experienced exponential growth. The availability of large amounts of data and the increase of computational resources on computer systems led to the growth of application domains and diffusion of machine learning-based solutions, reporting impressive performance in many fields, like speech recognition (Hinton et al., 2012; Nassif et al., 2019; Deng and Li, 2013), computer vision (He et al., 2015; Rehman et al., 2017; Kato et al., 2017) (used, for example, by self-driving cars (Chen and Huang, 2017)), natural language processing (Vaswani et al., 2017), reinforcement learning (Silver et al., 2016) and cybersecurity tasks like spam and malware detection (Lowd and Meek, 2005b; Barreno et al., 2006; Biggio et al., 2007a,b, 2008; Newsome et al., 2006; Nelson et al., 2008; Raff et al., 2018).

Despite their impressive performances on a variety of tasks, it has been known for more than a decade that machine learning algorithms can be misled by different adversarial attacks staged either at training or at test time (Joseph et al., 2018; Biggio and Roli, 2018). The most used machine learning algorithms are designed to work under the so-called *stationarity assumption*: both the training and test data are assumed to belong to the same distribution. However, distribution drifts can happen either naturally or *adversarially*. The former can happen for several reasons, such as missing data due to sensor failures. In the latter, an adversary tampers with data purposely to cause failures during system operation (Huang et al., 2011; Biggio and Roli, 2018). After the first attacks proposed against linear classifiers in 2004 (Dalvi et al., 2004; Lowd and Meek, 2005b), Biggio et al. (2013) have been the first to show that nonlinear machine-learning algorithms, including support vector machines (SVMs) and neural networks, can be misled by gradient-based optimization attacks (Joseph et al., 2018). Nevertheless, such vulnerabilities of learning algorithms have become extremely popular only after that Szegedy et al. (2014) have demonstrated that also deep learning algorithms exhibiting superhuman performances on image classification tasks suffer from the same problems. They have shown that even only slightly manipulating the pixels of an input image can be suf-

ficient to induce deep neural networks to misclassify its content. Such attacks have then been popularized under the name of *adversarial examples*.

Since the seminal works on adversarial attacks, many defense methods have been proposed to mitigate the threat of adversarial examples. However, a large number of these defenses reported overly optimistic evaluations, whereas they have been shown to be ineffective against more sophisticated attacks, such as attacks that are aware of the defense mechanism. Therefore, the problems of defending machine learning models against adversarial examples and evaluating their robustness are still open.

According to Biggio and Roli (2018), the most promising defenses can be broadly categorized into two families. The first includes approaches based on robust optimization and game-theoretical models (Globerson and Roweis, 2006; Brückner et al., 2012; Rota Bulò et al., 2017). These approaches, which also encompass *adversarial training* (Goodfellow et al., 2015), explicitly model the interactions between the classifier and the attacker to learn robust classifiers. The underlying idea is to incorporate knowledge of potential attacks during training. This typically requires, however, generating attack samples during model training, which may be very computationally demanding for state-of-the-art DNNs. Moreover, they are usually developed against a specific class of attacks, thus they might not be robust against adversarial examples generated with different techniques (Araujo et al., 2020). The second family of defenses (complementary to the first) is based on the idea of detecting and rejecting samples that exhibit an outlying behavior with respect to unperturbed training data (Bendale and Boult, 2016; Lu et al., 2017), providing an additional class for anomalies and potential out-of-distribution attacks. Typically, they are designed to work under the so-called *manifold hypothesis*: in several domains, natural data are assumed to lie in a low-dimensional manifold embedded in a high-dimensional space (e.g., grayscale digits image domain). Remarkably, not every high-dimensional representation belongs to the natural data manifold (e.g., salt and pepper noise). Assuming adversarial examples to be out-of-manifold data, these defenses work by identifying adversarial points from their distance to the manifold, based on a distance-based rejection strategy: as far as a sample moves away from class prototypes, classifier support decreases to zero. If an input sample is not supported by any class, then it is rejected. Several remarkable instances of this approach can be found in the literature (Feinman et al., 2017; Melis et al., 2017; Papernot and Mcdaniel, 2018; Lamb et al., 2018; Metzen et al., 2017; Crecchi et al., 2019; Meng and Chen, 2017). Apart from technical differences, all these rejection-based defenses share a common backbone structure and suffer from the same drawback, i.e., high computational overhead and memory usage, as they require comparing input samples against a set of reference prototypes.

Most of the existing approaches work in the fully-supervised learning setting, whereas only a few approaches also leverage unlabeled data to improve adversarial robustness (Miyato et al., 2016; Park et al., 2018; Akcay et al., 2018; Carmon et al., 2019; Miyato et al., 2018; Schmidt et al., 2018; Najafi et al., 2019; Alayrac et al.,

2019), although the semi-supervised learning setting provides a natural scenario for real-world applications in which labeling data is costly while unlabeled samples are readily available. More importantly, the case of multi-label classification, in which each sample can belong to more classes, is only preliminary discussed in the context of adversarial learning in Song et al. (2018), while using adversarial examples to improve the accuracy on legitimate (non-adversarial) samples of some multi-label classifiers is studied in Wu et al. (2017) and Babbar and Schölkopf (2018).

## 1.1 Contributions

In this thesis, we focus on detection-based defenses against adversarial examples, providing three different techniques that can be applied to deep neural networks in order to increase their robustness by relying on an additional reject class. All the proposed methods do not need to generate adversarial examples during their training and can be either integrated during the model training (in the case of the first one) or applied to pre-trained models (in the case of the other ones), with a low additional computational overhead for the training phase. Furthermore, we evaluate them with specifically designed adaptive attacks. Indeed, an attack that is unaware of the defense mechanism may tend to craft adversarial examples in areas of the input space which are assigned to the rejection class; thus, such attacks, as well as previously-proposed ones, may rarely bypass our defenses.

#### Increasing Robustness with Domain Knowledge

In the first contribution, we exploit the use of domain knowledge to improve the robustness of multi-label classifiers and to detect adversarial examples by integrating it into the learning process through logic constraints. While the generic idea of considering domain information in adversarial attacks has been recently followed by other authors to different extents, to the best of our knowledge, we are the first to use domain knowledge expressed by First-Order Logic (FOL) and converted into polynomial constraints to improve adversarial robustness of multi-label classifiers. To properly evaluate the robustness of this approach, we also propose a novel multi-label attack that can implement both black-box and white-box adaptive attacks, being driven by the domain knowledge in the latter case. Both the defense and the attack can be applied in the single-label setting too.

#### Detecting Adversarial Examples in Inner DNN layers

In the second contribution, we propose a multi-layer adversarial examples detection mechanism for deep neural networks, which is able to reject those samples showing anomalous behavior with respect to samples from the training data distribution. Although the idea of rejecting inputs on which the classifier is not sufficiently confident or that are far from the training data distribution is not new (even though

the majority of these approaches are not tested against adversarial examples), we leverage multiple DNN layers without requiring to generate adversarial examples at test time. In addition, we design a novel adaptive attack that is able to break the defense in a white-box setting to evaluate it properly.

**Improving the Efficiency of Prototypes-based Detectors**

In the third contribution, after designing a framework that unifies detector-based defenses against adversarial examples, we present a technique to speed up such detectors. In fact, all of them share a huge computational overhead and memory usage due to the fact that they require comparing input samples with a (usually large) set of reference prototypes, making it difficult to apply those approaches in practical settings. To overcome these limitations, we propose FADER (Fast Adversarial Example Rejection), a technique that allows obtaining an end-to-end differentiable detector capable of an up to 80× prototypes reduction with respect to analyzed competitors.

## 1.2 List of Publications

This thesis is based on the following publications:

- S. Melacci, G. Ciravegna, **A. Sotgiu**, A. Demontis, B. Biggio, M. Gori, and F. Roli. Domain Knowledge Alleviates Adversarial Attacks in Multi-Label Classifiers. In *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2021 (Melacci et al., 2021);

- **A. Sotgiu**, A. Demontis, M. Melis, B. Biggio, G. Fumera, Xiaoyi Feng, and F. Roli. Deep neural rejection against adversarial examples. In *EURASIP Journal on Information Security*, 2020 (Sotgiu et al., 2020);

- F. Crecchi, M. Melis, **A. Sotgiu**, D. Bacciu, and B. Biggio. FADER: Fast Adversarial Example Rejection. In *Neurocomputing*, 2022 (Crecchi et al., 2022).

Other works carried out during the Ph.D. course, that are not included in this thesis:

- M. Pintor, D. Angioni, **A. Sotgiu**, L. Demetrio, A. Demontis, B. Biggio, and F. Roli. ImageNet-Patch: A Dataset for Benchmarking Machine Learning Robustness against Adversarial Patches. In *Pattern Recognition*, 2022 (Pintor et al., 2022a);

- M. Pintor, L. Demetrio, **A. Sotgiu**, A. Demontis, N. Carlini, B. Biggio, and F. Roli. Indicators of Attack Failure: Debugging and Improving Optimization of Adversarial Examples. In *Thirty-sixth Conference on Neural Information Processing Systems (NeurIPS 2022)*, 2022 (Pintor et al., 2022b);

- M. Pintor, L. Demetrio, **A. Sotgiu**, M. Melis, A. Demontis, and B. Biggio. secml: Secure and explainable machine learning in Python. In *SoftwareX*, 2022 (Pintor et al., 2022c);
- **A. Sotgiu**, M. Pintor, and B. Biggio. Explainability-based Debugging of Machine Learning for Vulnerability Discovery. In *The 17th International Conference on Availability, Reliability and Security (ARES 2022)*, 2022 (Sotgiu et al., 2022).

# Chapter 2

# Background

This preliminary chapter introduces some basic concepts about machine learning (Section 2.1) and two of the most deployed learning algorithms, i.e., support vector machines (Section 2.2) and neural networks (Section 2.3), on which the defense approaches presented in this thesis are based. In Section 2.4, an overview of adversarial machine learning is given, with a particular focus on evasion attacks and defenses and their evaluation methodology. Finally, Section 2.5 summarizes the limitations and open issues of the current approaches.

## 2.1 Machine Learning

Machine learning refers to an ensemble of techniques that enables computer systems to learn from data, performing specific tasks without being explicitly programmed to do so. These tasks range from *classification* and *regression* to *clustering* problems. The process starts from data, represented by a *feature vector* that contains specific discriminant characteristics of data. Algorithms learn from a set of collected data called *training data* (or *training set*).

### Supervised and Unsupervised Learning

*Supervised learning* algorithms use labeled training data, while *unsupervised learning* ones rely on unlabeled data. Classification and regression belong to supervised learning algorithms and have the purpose of predicting labels of novel data, different from training one. In classification, these labels are discrete categories (classes); in regression, they are continuous values. Clustering is a set of unsupervised learning algorithms. In these learning algorithms, data is unlabeled, and the goal is to find structure and relationships in the data. In clustering algorithms, the task is to group sets of data that have similar characteristics. Finally, semi-supervised learning refers to the case in which both labeled and unlabeled data are available. Typically, in this case, only a small fraction of data is labeled as it might require a considerable amount of resources, and unlabeled data can help to improve learning accuracy.

### Binary, Multi-class and Multi-label Classification

Classification algorithms can be categorized based on the number of classes that are modeled by the classifier. In the simplest case of *binary classification*, only two classes exist, whereas in the *multi-class* setting three or more classes are considered. For the latter, we can further distinguish between the *single-label* setting, where classes are mutual-exclusive and a sample can be assigned to one and only one class, and the *multi-label setting*, where a sample might belong to more than one class. In this work, we focus on different settings: the first contribution is mostly focused on a semi-supervised setting in both single and multi-label cases, whereas the other two contributions are based on a fully-supervised single-label setting. Unless when explicitly specified (Chapter 3 and its related experimental evaluation in Section 6.1), in the rest of the thesis, we assume that the single-label fully-supervised setting is considered.

### Learning Algorithm Overview

To be effective, classification algorithms have to be able to *generalize* what they learned during the training phase to new unseen data.

To understand how these learning algorithms work, we can analyze a simple classification problem, as the famous one showed in Duda et al. (2000), where the task is to distinguish salmons from sea basses from images of fishes. These are the two classes, and many features can be selected to represent each sample. In this specific example, the lightness and width of fishes are used as features. We can then represent each fish with its feature vector in a two-dimensional *feature space* (Figure 2.1). In this vector space, the algorithm aims to learn a *decision boundary*, separating the space into two different *decision regions* so that a new fish can be classified depending on the region to which it belongs in its feature representation.
The optimal decision boundary is chosen in order to separate training samples while preserving the ability to generalize to new (test) samples. Classification is performed with a *decision function* $f : \mathcal{X} \mapsto \mathcal{Y}$ that maps an input sample $\boldsymbol{x} \in \mathcal{X} \subseteq \mathbb{R}^d$ to a class label $y \in \mathcal{Y} \subseteq \mathbb{R}^c$, where $c$ and $d$ are the number of classes and features, respectively. The learning of the optimal decision function needs a set of *training* data $\mathcal{L} = (\mathbf{X}, \boldsymbol{y})$. A general definition of the decision function can be formulated as follow:

$$f(\boldsymbol{x}) = [f_1(\boldsymbol{x}), \ldots, f_c(\boldsymbol{x})] \ , \tag{2.1}$$

where each component $f_i$ provide a membership *score* w.r.t. the class $i$ for the input sample $\boldsymbol{x}$. The predicted class label can then be computed as:

$$\hat{y} = \arg\max_{i=0,\ldots,c} f_i(\boldsymbol{x}) \, . \tag{2.2}$$

Often, in a two-class dataset, the labels are a number $y \in \{-1, 1\}$, and the learning algorithm looks for a single-output function $f(\boldsymbol{x})$ so that $y = +1$ if $f(\boldsymbol{x}) \geq 0$, $-1$

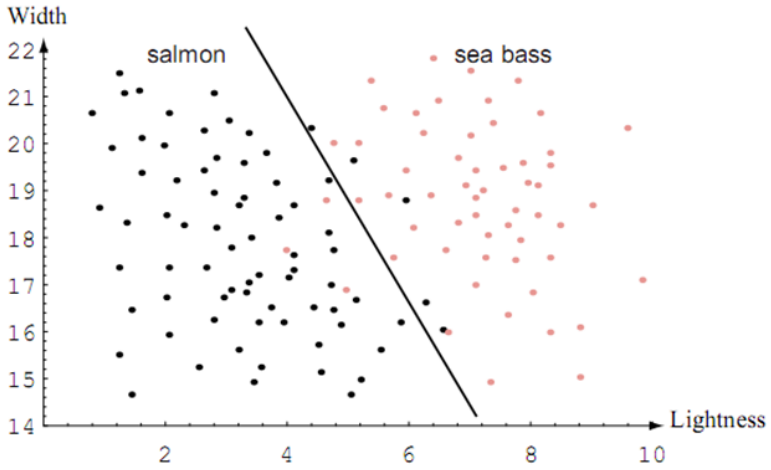


Figure 2.1: Salmon and sea bass samples represented in a two-dimensional feature space, where features are lightness and width. The dark line represents a possible decision boundary between the two classes (Duda et al., 2000).

otherwise. This function can assume several shapes. If we consider Figure 2.1, classes are separated by a straight line: this is an example of *linear model*. If we generalize to a $d$-dimensional space, we can assume that classes can be separated by a hyperplane, and the decision function is given as:

$$f(\boldsymbol{x}) = \text{sign}(\boldsymbol{w}^\top \boldsymbol{x} + b)\,, \tag{2.3}$$

where the vectors $\boldsymbol{w} \in \mathbb{R}^{\mathsf{d}}$ and $b \in \mathbb{R}$ contain the so-called *feature weights* and *bias*, respectively. These parameters define different hyperplanes (that can be obtained with the equation $f(\boldsymbol{x}) = 0$) and must be optimized to find the ones that provide the best trade-off between fitting training samples and generalizing to novel ones. Indeed, if a classifier overfits training samples, it will not be able to classify correctly new testing samples. To solve the optimization problem can thus be defined the following objective function to minimize over all the $n$ training samples:

$$\min_{\boldsymbol{w},b} R(\boldsymbol{w}) + C \sum_{i=1}^{\mathsf{n}} Q(y_i, \boldsymbol{w}^T \boldsymbol{x}_i, b)\,. \tag{2.4}$$

The term $Q(y_i, \boldsymbol{w}^T \boldsymbol{x}_i, b)$ denotes a *loss function*, that basically measures the number of errors made by the decision function $f(\boldsymbol{x})$ on the samples in $\mathcal{L}$. Minimizing this term, in fact, has the effect of fitting the algorithm to training data. To avoid overfitting on such data, the regularization term $R(\boldsymbol{w})$ is used. Basically, minimizing this term forces the algorithm to find a simpler and smoother solution to the problem. Finally, $C$ is a trade-off parameter, called *regularization parameter*, that balances the influence of $R$ and $Q$.

If more than two classes are present, we can still use the aforementioned method by leveraging different strategies. In *One-Versus-One* classifiers, a two-class classifier is

trained for each pair of classes. However, this approach is not efficient, as it requires $\frac{c(c-1)}{2}$ classifiers to be trained, with $c$ being the number of classes. In *One-Versus-All* classifiers, $c$ two-class classifiers are used. The $i$-th classifier is trained using the $i$-th class as positive and all other classes as negatives. The predicted class $\hat{y}$ (being $f_i(\boldsymbol{x})$ the decision function of the $i$-th classifier) can be computed with Equation (2.2).

## 2.2 Support Vector Machines

Among classification algorithms, before the recent interest in Deep Learning, Support Vector Machines (SVMs) were one of the mostly used. Based on a sound mathematical formulation, this algorithm ensures efficient and accurate classification in many tasks. In the simplest application scenario with two linearly separable classes, its behavior strongly resembles the example shown in Section 2.1, as it basically constructs a hyperplane separating data with a discriminant function. The algorithm looks for the *maximum-margin* hyperplane, i.e., the one with the maximum distance from the nearest training data points (called *support vectors*) of the two classes. This is the *hard-margin* case, which works well if data does not contain samples out of class distributions (also known as outliers). Otherwise, the computed solution might not be effective when classifying new data samples unless a *soft-margin* is used, allowing some training points to violate the separating hyperplane. In order to do this, the algorithm applies Equation (2.4), exploiting an $\ell_2$ regularizer on the feature weights and the so-called *hinge loss* as loss function.
Minimizing this function, called *primal form*, can be difficult. A simplified problem can be obtained using the Lagrangian dual, thus the function to be solved becomes:

$$\max_{\alpha} \quad -\frac{1}{2}\sum_{i,j=1}^{n} y_i y_j \boldsymbol{x_i}\boldsymbol{x_j} + \sum_{i=1}^{n}\alpha_i \tag{2.5}$$

$$\text{s.t.} \quad 0 \leq \alpha_i \leq C\,, \tag{2.6}$$

$$\sum_{i=1}^{n} y_i \alpha_i = 0\,. \tag{2.7}$$

This is called the *dual form*. The decision function of the classifier also changes:

$$f(\boldsymbol{x}) = \sum_{i}^{n} \alpha_i y_i (\boldsymbol{x_i}'\boldsymbol{x}) + b\,. \tag{2.8}$$

Quadratic programming or gradient descent algorithms can solve the optimization problem above. If classes are not linearly separable, these approaches do not ensure an acceptable solution. To overcome this issue, the so-called *kernel trick* is used. It consists of mapping samples into feature spaces with higher dimensions, selected in order to make them separable. Indeed, with appropriate nonlinear mapping, data can always be separated by a hyperplane. The aforementioned dual form allows

training the classifier without mapping samples in the new feature space. Mapping functions are called *Kernel* ($K$) functions, and they are applied by computing a scalar product between samples, satisfying $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \Phi(\boldsymbol{x}_i) \cdot \Phi(\boldsymbol{x}_j)$ where $\Phi(\boldsymbol{x})$ represents the mapped sample. In the algorithm, we use only $K$ for the sake of brevity, without regard for $\Phi$. Replacing all the products $\boldsymbol{x}_i \boldsymbol{x}_j$ (see Equation (2.5) and (2.8)) with $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$, the data points are easily projected into a new space, where they are separable, and all previous considerations are still valid. Many kernel functions can be used. One of the most famous is the *radial basis function* (RBF) kernel, which allows mapping the samples in an infinite-dimensional features space:

$$K(\boldsymbol{x}_1, \boldsymbol{x}_2) = exp(-\gamma ||\boldsymbol{x}_1 - \boldsymbol{x}_2||^2) \,, \tag{2.9}$$

where the parameter $\gamma$ defines the variance of the Gaussian function.

The number of support vectors characterizes the complexity of the resulting classifier, and it is independent of the dimensionality of the transformed space. Therefore, the kernel trick allows efficient deployment of SVMs classifiers in high-dimensional spaces. In the multi-class setting, either One-Versus-One or One-Versus-All scheme described in Section 2.1 can be applied.

## 2.3 Neural Networks and Deep Learning

Neural networks are inspired by the human brain, as they are composed of artificial neurons, which receive and can activate based on inputs, eventually giving an output. Their basic units were initially developed in the 1950s and 1960s and are connected into a network where each connection has a weight. Typically neural networks are organized in layers, where the first layer receives the input data, and the last layer outputs the result of the algorithm. Between these layers, a network can have one or more hidden layers. In *feed-forward* neural networks, only neurons of adjacent layers are connected. Feed-forward networks can be seen as annealed functions, one for each layer. Let us assume that, without loss of generality, the network consists of $m$ layers. This implies that the prediction function $f$ can be expressed as a composition of functions:

$$f(h_m(h_{m-1}(\dots h_1(\boldsymbol{x}; \boldsymbol{w}_1)); \boldsymbol{w}_{m-1}); \boldsymbol{w}_m) \,, \tag{2.10}$$

where $h_1$ and $h_m$ denote the mapping function learned, respectively, by the input and the output layer, and $\boldsymbol{w}_1$ and $\boldsymbol{w}_m$ are their weight parameters (learned during training). If each neuron in one layer is connected to all neurons in the next layer, the network is *fully-connected*.

The training of feed-forward neural networks basically consists in minimizing a loss function, which usually is an error function computed on the classifier output, adjusting the values of connection weights. Since this function can be very complex as the number of neurons increases, efficient learning algorithms have been developed since the 1980s, namely *backpropagation algorithms*. They exploit continuous

and differentiable activation functions of neurons, which makes the network output a continuous and differentiable function of its inputs, leveraging the *chain rule* to compute derivatives:

$$\frac{\partial f}{\partial \boldsymbol{w}} = \frac{\partial h_m}{\partial h_{m-1}} \cdots \frac{\partial h_2}{\partial h_1} \frac{\partial h_1}{\partial \boldsymbol{w}}. \tag{2.11}$$

The training loss can thus be minimized using *gradient descent* methods by forwarding training data into the network, computing the loss from the network output and its gradient with respect to the network weights, and updating them accordingly. This procedure is repeated iteratively for a certain number of training *epochs*, whereas the *learning rate* tunes the magnitude of weights adjustments. It can be applied by forwarding all training samples (*batch gradient descent*), subsets of these samples (*mini-batch gradient descent*), or one sample at a time (*stochastic gradient descent*).

Deep learning models are based on neural networks with a large number of hidden intermediate layers, called *deep neural networks* (DNNs), which are able to perform very complex tasks. This has become possible in the last decade, especially due to the increase in available data and computational power (e.g., dedicated GPUs). DNNs can be applied in both supervised and unsupervised learning settings, and their main applications are speech and audio processing, natural language processing, information retrieval, object recognition, and computer vision. In these models, multiple layers process the raw input with transformations that can also be non-linear, where each layer learns during the training a feature representation that becomes more and more abstract, in a hierarchical structure.

For computer vision applications *convolutional neural networks* (CNNs) are typically used, which have specialized layers (not fully connected) that process raw input images mapping them into increasingly abstract feature spaces, followed by fully-connected layers that perform the classification on such feature representations. Convolutional layers are used by CNNs to incorporate the relative invariance of the spatial relationship in typical image pixels with respect to the location. They use local receptive fields so that each unit processes data only on its assigned image region, with kernels that are computed during the learning and convolved to the input. Pooling layers, also called *subsampling* layers, take the output of convolutional layers and reduce the dimensions of data, usually using the average or the maximum value of input clusters. Two additional operations can be added to increase performances in CNNs. The first is a powerful regularization technique called *dropout*, which drops individual units with a fixed probability, training the network on remaining neurons and then reinserting dropped neurons in order to avoid overfitting. The second is the use of the *rectified linear unit* (ReLU) activation function, which computes $f(\boldsymbol{x}) = max(\boldsymbol{x}, 0)$, which adds sparsity, increases efficiency, and reduces the presence of vanishing gradients.

## 2.4 Adversarial Machine Learning

Since machine learning algorithms have been deployed in security-critical applications, it quickly emerged that attackers could threaten their functionality. For instance, the first works (Dalvi et al., 2004; Lowd and Meek, 2005a,b) showed that email spam filters based on linear classifiers could be evaded with a few small carefully-crafted changes to the text. These works laid the foundations for the adversarial machine learning research field, specifically focused on the study of attacks against machine learning algorithms, defense strategies to mitigate such attacks, and methodologies for their security evaluations.

### 2.4.1 Attacker Model

As highlighted in Biggio and Roli (2018), it is first necessary to model threats against learning-based systems in order to evaluate their security against the corresponding attacks by defining different attack scenarios and strategies. To this end, they propose an attacker model based on seminal work by Barreno et al. (2006) and following works, where attackers are characterized with respect to their goal, knowledge of the target system, and capability of manipulating the input data. Afterward, it is possible to define an optimization problem corresponding to the optimal attack strategy that allows the attacker to achieve its goal by manipulating the input data.

#### Attacker's Goal

It can be further detailed with the following three different aspects.

**Security Violation.** The attacker may aim to violate one or more components of the *security triangle*, i.e., *availability*, *integrity* and *confidentiality*. The first can be threatened by compromising normal system operation, while the second refers to, e.g., evading the detection without making the system unavailable. Confidentiality violation aims to steal private information from and about the system.

**Attack Specificity.** It can be *indiscriminate* or *targeted*. The goal is, in both cases, to cause the misclassification of samples, but in the second case, this is performed only with a specific set of them.

**Error Specificity.** If the attacker wants the system to misclassify only samples from a specific class, it is *specific*. Otherwise, it is *generic* if the aim is to make the system misclassify samples of any of the classes different from the true class.

#### Attacker's Knowledge

The attacker can have different knowledge of the main elements of the targeted system, namely the training data, the feature set, the learning algorithm, the objective function minimized during training, and the trained model parameters/hyper-

parameters. Depending on these assumptions, one can describe different attack scenarios.

**Perfect-knowledge (PK) White-box Attacks.** In this setting, the attacker is assumed to know everything about the targeted system, i.e., all the elements mentioned above. This scenario can be considered the worst-case, and an evaluation of the system security in this setting can provide empirical upper bounds on its performance degradation under attack.

**Limited-knowledge (LK) Gray-box Attacks.** This case includes different combinations related to partial knowledge of the system components. Typically, the attacker is assumed to know only the kind of learning algorithm and the feature representation but neither the model parameters nor the training data. However, he can still perform an attack if he is able to collect a surrogate dataset and eventually get feedback from the target classifier by observing its output, estimating its parameters, and then training a surrogate classifier. A surrogate classifier can also be used when the optimization required to perform an attack is too complex if conducted against a known target classifier. In this case, the attack is crafted against the surrogate classifier and tested against the target one, and it represents an example of the transferability of attacks between different algorithms, as shown in Biggio et al. (2013) and Papernot et al. (2017).

**Zero-knowledge (ZK) Black-box Attacks.** If the attacker has no knowledge of the system components but can query the system and get its feedback (e.g., the predicted labels or the confidence scores for the classes), he can still perform an attack (Tramèr et al., 2016; Papernot et al., 2017; Chen et al., 2017), by using or not surrogate classifiers. Actually, it's unlikely that the attacker has zero knowledge of the system: it is natural that he knows the task that the classifier is designed to perform and consequently has an idea of which potential transformations to apply to cause some feature changes, and which kind of training data was used, otherwise, neither change can be inflicted to the output of the classification function, nor any useful information can be extracted from it.

### Attacker's Capability

According to the influence that the attacker can have on the input data and on application-specific data manipulation constraints, two different characteristics can be identified.

**Attack Influence.** Here, two settings can be distinguished. In the former, the attacker can manipulate training data, altering the training phase in order to obtain a desired model behavior. This kind of attacks is commonly known as *poisoning*. In the latter, the attacker can manipulate test data to achieve his goal. In this thesis, we only focus on *evasion attacks* (widely discussed in Section 2.4.2), where input test data is manipulated to alter the model output.

| | Attacker's Goal | | |
|---|---|---|---|
| **Attacker's Capability** | **Integrity** | **Availability** | **Privacy / Confidentiality** |
| **Test data** | Evasion (a.k.a. adversarial examples) | Energy-latency attacks (a.k.a. *sponge examples*) | Model extraction / stealing and model inversion (a.k.a. hill-climbing attacks) |
| **Training data** | Poisoning (to allow subsequent intrusions) – e.g., backdoors or neural network trojans | Poisoning (to maximize classification error) | - |

Table 2.1: Categorization of attacks against machine learning, as defined in Biggio and Roli (2018).

**Data Manipulation Constraints.** Based on the specific application domain of the targeted system, the manipulation of input data might be constrained. Generally, these constraints are taken into account in the definition of the optimal attack strategy, e.g., limiting the alteration of attack samples within a space of possible modifications. Also, feature values of attack samples can be bounded (Biggio et al., 2013).

### Attack Strategy

The attacker goal, given the above-defined attack scenario, can be represented with an objective function that quantifies the effectiveness of the attack. This function changes depending on the kind of attack and is usually minimized or maximized by the attack algorithm.
In Table 2.1, the taxonomy of adversarial attacks against machine learning is shown based on the defined attacker's goal and capability. In this thesis, we will only focus on evasion attacks, considering different settings regarding the attacker's knowledge. Our evaluations are performed with untargeted and error-generic attacks, whereas they can be easily extended to the targeted and error-specific configurations.

## 2.4.2 Evasion Attacks

The already mentioned first works attempting to perform evasion attacks, mainly focused on anti-spam filters and intrusion detection systems, date back to 2004-2006 (Dalvi et al., 2004; Wittel and Wu, 2004; Lowd and Meek, 2005b; Fogla et al., 2006). It quickly turned out that in critical domains, where an adversary has an interest in fooling the system, a learning algorithm can be easily misled. Even by changing the algorithm reactively, an adaptive attacker can find new solutions to reach this goal, like an arms race.
Evasion was performed firstly on linear classifiers, obfuscating bad words and/or adding good words in emails. Afterward, Biggio et al. (2013) showed that also non-linear machine learning algorithms, including support vector machines and neural networks, can be misled by gradient-based optimization attacks (Joseph et al., 2018).

**Algorithm 1:** Generalized gradient-based attack for optimizing adversarial examples.

**Input :** $\boldsymbol{x}$, the initial sample; $y_t$, the target class label; $n$, the number of iterations; $\eta$, the step size; $\boldsymbol{\Theta}$, the target model; $\Omega$ the loss function; $\Pi$, the projection operator enforcing the constraints in Equation (2.13).

**Output:** $\boldsymbol{x}^\star$, the solution found by the algorithm

$\boldsymbol{x}_0 \leftarrow$ `initialize`$(\boldsymbol{x})$ ▷ `Initialize starting point`

$\hat{\boldsymbol{\Theta}} \leftarrow$ `approximation`$(\boldsymbol{\Theta})$ ▷ `Use surrogate model (if required)`

$\boldsymbol{\delta}_0 \leftarrow \mathbf{0}$ ▷ `Initialize` $\delta$

**for** $i \in [1, n]$ **do**

  $\boldsymbol{\delta}_i \leftarrow \boldsymbol{\delta}_{i-1} - \eta \nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x} + \boldsymbol{\delta}_{i-1}, y_t; \hat{\boldsymbol{\Theta}})$ ▷ `Compute gradient update(s)`

  $\boldsymbol{\delta}_i \leftarrow \Pi(\boldsymbol{x}, \boldsymbol{\delta}_i)$ ▷ `Project` $\boldsymbol{\delta}$ `onto the feasible domain`

**return** $\boldsymbol{x}^\star \leftarrow \boldsymbol{x} +$ `best`$(\boldsymbol{\delta}_0, ..., \boldsymbol{\delta}_n)$ ▷ `Return best solution`

Figure 2.2 shows a 2-dimensional toy example of error-specific and error-generic evasion attacks computed using gradient-based algorithms. A gradient-based attack was also presented in Šrndic and Laskov (2014), targeting a PDF malware detector based on nonlinear classifiers.

Nevertheless, such vulnerabilities of learning algorithms have become extremely popular when Szegedy et al. (2014) and subsequent works (Goodfellow et al., 2015; Nguyen et al., 2015; Moosavi-Dezfooli et al., 2016) have demonstrated that also deep learning algorithms exhibiting superhuman performances on image classification tasks suffer from the same problems. They have shown that even only slightly manipulating the pixels of an input image can be sufficient to induce deep neural networks to misclassify its content. Such attacks have then been popularized under the name of *adversarial examples*.

Since then, several attack algorithms have been proposed to craft adversarial examples (Carlini and Wagner, 2017b; Croce and Hein, 2020a). In addition to distinctions that can be made based on the categorization in Section 2.4.1, evasion attack strategies can be different. In *maximum-confidence* attacks, the objective is to cause a misclassification with the higher possible confidence, given a fixed upper bound for the perturbation that can be applied to data samples. On the other hand, *minimum-distance* attacks aim to find the smallest perturbation leading to misclassification.

In this thesis, we only consider maximum-confidence attacks, as they are required to perform black-box evaluations of proposed detection methods by generating high-confidence adversarial examples against the undefended models and evaluating the defenses with them. In addition, these attacks allow us to create adversarial examples with a controlled amount of perturbation and thus produce security evaluation curves (see Section 2.4.4) in both black-box and white-box settings. Most of them adopt a similar optimization procedure that can be summarized with the following

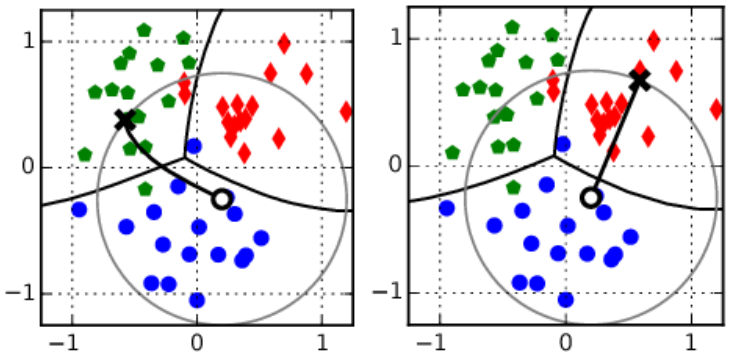


Figure 2.2: Error-specific (left) and error-generic (right) evasion attacks against a multiclass RBF-SVM (Melis et al., 2017). Decision boundaries among the three classes (blue, red and green points) are shown as black solid lines. The gray circles show a $\ell_2$ constraint on the adversarial perturbation. In the first case, the initial (blue) sample is shifted towards the green class (selected as the target one). In the second case, instead, it is shifted towards the red class, as it is the closest class to the initial sample.

general formulation:

$$\min_{\boldsymbol{\delta}} \quad \Omega(\boldsymbol{x}+\boldsymbol{\delta}, y; \boldsymbol{\Theta}), \tag{2.12}$$

$$\text{s.t.} \quad \|\boldsymbol{\delta}\|_p \leq \varepsilon, \quad \text{and} \quad \boldsymbol{x}+\boldsymbol{\delta} \in [0,1]^d, \tag{2.13}$$

where $\boldsymbol{\delta} \in \mathbb{R}^d$ is the perturbation applied to the input sample $\boldsymbol{x}$ during the optimization, and $\boldsymbol{\Theta}$ are the parameters of the model. The loss function $\Omega$ is defined such that minimizing it amounts to having the perturbed sample $\boldsymbol{x}+\boldsymbol{\delta}$ misclassified, i.e., its predicted label must be different from the true label $y$ (additionally, if the attack is targeted, the predicted label must be equal to the target label $y_t$). Typical examples include the Cross-Entropy (CE) loss[1], or the so-called *logit* loss (Carlini and Wagner, 2017b), shown in the following equation:

$$\Omega(\boldsymbol{x}+\boldsymbol{\delta}, y; \boldsymbol{\Theta}) = \begin{cases} \max_{k \neq y_t} f_k(\boldsymbol{x}+\boldsymbol{\delta}, \boldsymbol{\Theta}) - f_{y_t}(\boldsymbol{x}+\boldsymbol{\delta}, \boldsymbol{\Theta}) & \text{if targeted,} \\ f_y(\boldsymbol{x}+\boldsymbol{\delta}, \boldsymbol{\Theta}) - \max_{k \neq y} f_k(\boldsymbol{x}+\boldsymbol{\delta}, \boldsymbol{\Theta}) & \text{if untargeted,} \end{cases} \tag{2.14}$$

being $f_i(\cdot, \boldsymbol{\Theta})$ the model's prediction (*logit*) for class $i$. Let us finally discuss the constraints in Equation (2.13). While the $\ell_p$-norm constraint $\|\boldsymbol{\delta}\|_p \leq \varepsilon$ bounds the maximum perturbation size, the box constraint $\boldsymbol{x}+\boldsymbol{\delta} \in [0,1]^d$ ensures that the perturbed sample stays withing the given (normalization) bounds.

Biggio et al. (2013), Papernot et al. (2016a) and subsequent works (Demontis et al., 2019) have shown the *transferability* property of adversarial examples, i.e., adversarial samples computed against a given model, are also effective against different,

[1] If the attack is targeted, the CE is computed on the target class $y_t$, whereas if the attack is untargeted it is computed on the true class $y$ and its sign is inverted

potentially unknown, models. This enables to perform evasion attacks without having access to target model parameters and gradients and can also be useful if the target model is either non-differentiable or not sufficiently smooth (Athalye et al., 2018a).

We provide here a generalized algorithm, given as Algorithm 1, which summarizes the main steps followed by gradient-based attacks to solve Problem (2.12)-(2.13). The algorithm starts by defining an *initialization point* (line 1), which can be the input sample $\boldsymbol{x}$, a randomly-perturbed version of it, or even a sample from the target class (Brendel et al., 2019). If the target model $\boldsymbol{\Theta}$ is either non-differentiable or not sufficiently smooth, a surrogate model $\hat{\boldsymbol{\Theta}}$ can be used to approximate it, and perform a transfer attack (line 2). The attack then iteratively updates the point to find an adversarial example (line 4), computing one (or more) gradient updates in each iteration (line 5), while the perturbation $\boldsymbol{\delta}_{i+1}$ is projected onto the feasible domain (i.e., the intersection of the constraints in Equation 2.13) via a projection operator $\Pi$ (line 6). The algorithm finally returns the best perturbation across the whole *attack path*, i.e., the perturbed sample that evades the (target) model with the lowest loss (line 7).

Remarkably, other works exploit different techniques to perform these attacks by relying only on the classifier output (either the predicted label or the output scores) (Chen et al., 2017; Andriushchenko et al., 2020). Finally, different works showed the feasibility of generating adversarial examples in the real world. Sharif et al. (2016) attacked a face recognition system by printing a pair of eyeglasses frames, performing both targeted and untargeted evasion attacks, Eykholt et al. (2018) added stickers to road sign to make them misclassified, Athalye et al. (2018b) used a 3D printer to create real adversarial examples by shaping their texture, all using modified attack algorithms to produce pose-invariant perturbations.

### 2.4.3 Defenses against Evasion Attacks

Together with works on evasion attacks and adversarial examples, many defense systems were introduced to mitigate this issue. Those defenses can be categorized, accordingly to Biggio and Roli (2018), in *reactive* and *proactive*. While reactive defenses aim to counter *past* attacks (e.g., by timely detection of novel attacks, frequently retraining classifiers, or verifying the consistency of classifier decisions), proactive defenses aim to prevent *future* attacks, either by hiding information to the attacker (a.k.a. *security by obscurity*) or developing machine learning models able to provide intrinsic robustness (a.k.a. *security by design*), evaluating them in a white-box setting where the attacker has a perfect knowledge of the system. The latter approaches are the most promising and include several different techniques that we will briefly describe.

**Adversarial Training.** This approach aims to inject knowledge of adversarial attacks in the defended model during its training so that the model can learn how

to map them to their true labels. Dalvi et al. (2004) proposed for the first time an adversary-aware classifier able to increase its security by iteratively retraining itself on simulated attacks. Other approaches were introduced by Goodfellow et al. (2015) and Kurakin et al. (2017), and then extended by Madry et al. (2018). Despite improving adversarial robustness, adversarial learning is resource-demanding and time-consuming, as it requires creating adversarial examples during training. Training time is slowed down by the inner attack iterations that create the augmented samples. However, applying faster attacks still produces good results (Wong et al., 2019; Rony et al., 2019). This method is often combined with other defense mechanisms for further increasing robustness.

**Robust Regularization.** It consists in penalizing the input gradients during the classifier training, smoothing out its decision function and hence its sensitivity to input perturbations. With respect to adversarial learning, the cost of computing adversarial examples is removed, but the method still requires the computation of the input gradients to regularize them along with the objective function, increasing the computational overhead. This method was shown to be equivalent to adversarial training in Ross and Doshi-Velez (2018).

**Manifold Projections.** In this defense approach, the input test data is projected into a previously learned data manifold that models the distribution of the training samples. In this way, if an adversarial example is submitted to the model, it can be reprojected in the natural data distribution with the intent of removing the adversarial perturbation (Jalal et al., 2017; Shen et al., 2017; Samangouei et al., 2018; Song et al., 2018; Schott et al., 2018).

**Stochasticity.** This defense method tries to hinder the attacker by adding randomness to the input (Xie et al., 2018; Prakash et al., 2018), the activations (Xiao et al., 2020; Dhillon et al., 2018), or to the outputs (Park et al., 2021). These approaches follow the security-by-obscurity paradigm, and many of them are often broken by the use of smoothing techniques in the loss (Tramer et al., 2020).

**Preprocessing/Input Quantization.** In this defense technique, input samples or the activations (Buckman et al., 2018) are preprocessed (Guo et al., 2018; Munusamy Kabilan et al., 2018) and often quantized (Papernot et al., 2016b; Lu et al., 2017), but also, in this case, it has been shown that many of them are ineffective (Carlini and Wagner, 2016).

**Detectors.** This family of defenses is based on the idea of detecting and rejecting samples that exhibit an outlying behavior with respect to unperturbed training data. To do this, usually, a separate model is trained to detect the adversarial examples (Bendale and Boult, 2016; Meng and Chen, 2017; Li and Li, 2017; Melis et al., 2017; Yu et al., 2019). Most of these works focus on rejecting samples that are far from the training data in feature space. Such samples appear in regions of the feature space scarcely populated by training data, and they are called *blind-spot* evasion points. During the classifier training, these regions can be assigned to any

class, almost indifferently regarding the training loss. This occurs consequently to the assumption that training and test data come from the same distribution (Pillai et al., 2013), which underlies many machine learning algorithms. Typically the outputs of the model are the label and the probability that the sample is adversarial, which can be thresholded to reject the decision. In this case, the model does not provide the prediction unless a separate algorithm recovers the input sample.

**Certified Defenses.** These defenses provide guarantees of robustness to norm-bounded attacks by ensuring no adversarial example is present in the certified radius (Cohen et al., 2019; Zhang et al., 2020). The limitation of these defenses is that they do not scale well or are supported for very specific types of models or attacks.

### 2.4.4 Security Evaluation of Defenses against Evasion Attacks

In light of what was stated in previous sections, evaluating the robustness of machine learning models against adversarial examples is a crucial task, although not easy to perform. The security evaluation is typically performed by simulating the attacks and evaluating the *robust accuracy* of the system under attack, under a given setting, which is especially characterized by the attacker's knowledge and its strength.
For the former is usually assumed white-box access to the model where the attacker has full knowledge of the defense system in order to provide an evaluation under a worst-case scenario, whereas the black-box setting can still be useful to provide insights about the effectiveness of the defense. The latter can be fixed, but to have a complete overview of defenses' robustness and compare them *security evaluation curves* (Biggio and Roli, 2018) are more suitable. They compute the robust accuracy as a function of the attack strength by running it with different amounts of perturbation, showing how gracefully the performance decreases while the attack increases in strength, up to the point where the defense reaches zero accuracy (Figure 2.3). This is another important phenomenon to be observed since any defense against test-time evasion attacks *has to fail* when the perturbation is sufficiently large (or, even better, unbounded); in fact, in the unbounded case, the attacker can ideally replace the source sample with any other sample from another class (Athalye et al., 2018a). If accuracy under attack does not reach zero for very large perturbations, then it may be that the attack algorithm fails to find a good optimum, which corresponds to a good adversarial example. This, in turn, means that we are probably providing an optimistic evaluation of the defense. As suggested in Athalye et al. (2018a), the purpose of a security evaluation should not be to show which attacks the defense withstands but rather to show when the defense fails. If one shows that larger perturbations that may compromise the content of the input samples and their nature (i.e., its true label) are required to break the defense, then we can retain the defense mechanism to be sufficiently robust. Another relevant point is to show that such a *breakdown* point occurs at a larger perturbation than that

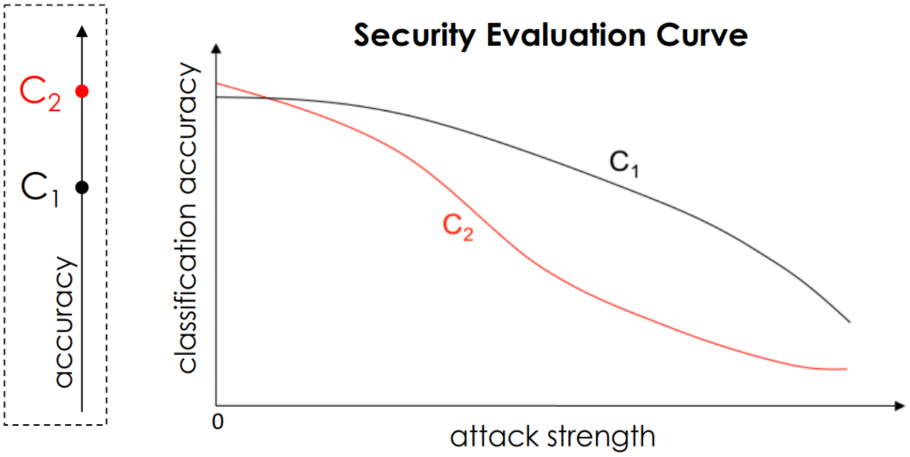


Figure 2.3: Example of security evaluation curves for two classifiers ($C_1$ and $C_2$) (Biggio and Roli, 2018). The two curves show how $C_1$ is more robust to an increasing adversarial perturbation than $C_2$, although $C_2$ report a slightly better performance in absence of perturbations.

exhibited by competing defenses, to show that the proposed defense is more robust than previously-proposed ones.

Despite the importance of this procedure, for many proposed defenses, it was not correctly performed. Some of them have been only evaluated against previous attacks rather than against ad-hoc attacks crafted specifically to break them (Papernot et al., 2016b; Lu et al., 2017; Meng and Chen, 2017; Athalye et al., 2018a; Carlini and Wagner, 2017a), providing overly optimistic results. It has indeed been shown afterward that such defenses can be easily bypassed by simple modifications to the attack algorithm (Athalye et al., 2018a; Carlini and Wagner, 2017b,a). Other defenses have been found to perform *gradient obfuscation*, i.e., they learn functions which are harder to optimize for gradient-based attacks; however, they can be easily bypassed by constructing a smoother, differentiable approximation of their function, e.g., via learning a surrogate model (Biggio et al., 2013; Biggio and Roli, 2018; Russu et al., 2016; Papernot et al., 2017; Demontis et al., 2019; Melis et al., 2018) or replacing network layers which obfuscate gradients with smoother mappings (Athalye et al., 2018a; Carlini and Wagner, 2017b,a). Finally, in other cases, some defenses were evaluated by running existing attacks with inappropriate hyperparameters (e.g., using an insufficient number of iterations). To prevent such evaluation mistakes, and help develop better defenses, evaluation guidelines and best practices have been described in recent work (Carlini et al., 2019). However, they have been mostly neglected as their application is non-trivial; in fact, 13 defenses published after the release of these guidelines were found again to be wrongly evaluated, reporting overestimated adversarial robustness values (Tramer et al., 2020). It has also been shown that, even when following such guidelines, robustness can still be overestimated (Popovic et al., 2022).

## 2.5 Limitations and Open Issues

In the previous section, we gave an overview of the state-of-the-art on evasion attacks and defenses. Despite the strong effort made by the research community, which produced a huge amount of proposed approaches, defending against adversarial examples and correctly evaluating proposed defense methods still remain an open issue.
Whereas certified defenses are the only ones providing provable robustness bounds, currently, their application is limited to specific settings.
On the other hand, among empirical defenses, the most promising results came from adversarial training. However, this technique needs to be applied during the model training, adding a relevant computational overhead and being unable to defend models without training them from scratch or applying an additional retraining phase to already trained models. Moreover, as adversarially-trained models are often developed to be robust with respect to a specific class of attacks, they might still be vulnerable to adversarial examples generated with different techniques (Araujo et al., 2020).
Complementary to these kinds of defenses are detection methods aiming to reject adversarial examples which are out-of-distribution with respect to natural samples. These approaches usually provide better computational performance with respect to adversarial training during the training phase, but they add some overhead at test time. Many of them rely on comparing input samples with a set of training data samples, whose number is thus crucial for the runtime efficiency of these detectors. In these methods, the number of selected training prototypes cannot be tuned and might almost match the training set size, increasing (i) the memory usage, as they usually need to be kept in memory, and (ii) the computational overhead, as the more prototypes are selected, the more computations are required when comparing input samples with them. A second issue we found in existing approaches is that, especially for multilayered defenses, classifiers are optimized to maximize detection separately, i.e., they are not jointly trained to perform rejection, as they are typically based on ensembling classifiers with heuristic methods, rather than optimizing the whole architecture in an end-to-end manner to better perform detection.
Another point that was not thoroughly considered when designing defenses, despite it represents a natural scenario for real-world applications, is the presence of unsupervised samples in the training dataset. The required cost to obtain them is typically very small with respect to labeled data, whereas they can contribute to increase the robustness of classifiers. Finally, the vast majority of attacks and defenses are designed for a single-label classification setting, while only few works preliminary addressed the multi-label scenario.

# Chapter 3

# Increasing Robustness with Domain Knowledge

In this chapter, we focus on multi-label classification and, in particular, in the case in which domain knowledge on the relationships among the considered classes is available. Such knowledge can be naturally expressed by First-Order Logic (FOL) clauses, and, following the learning framework of Gnecco et al. (2015) and Diligenti et al. (2017), it can be used to improve the classifier by enforcing FOL-based constraints on the unsupervised or partially labeled portions of the training set. A well-known intuition in adversarial machine learning suggests that a reliable model of the distribution of the data could be used to spot adversarial examples, being them not sampled from such distribution, but it is not a straightforward procedure (Grosse et al., 2017). We borrow such intuition, and we intersect it with the idea that semi-supervised examples can help learn decision boundaries that better follow the marginal data distribution, coherently with the available knowledge (Melacci and Belkin, 2011; Diligenti et al., 2017), and we investigate the role of such knowledge in the context of data generated in an adversarial manner.

The chapter is organized as follows. Section 3.1 introduces the notion of learning with domain knowledge, emphasizing its effects in the input space. Section 3.2 shows how domain knowledge can be used to defend against adversarial attacks, together with a knowledge-aware attack procedure. Finally, related work is discussed in Section 3.3.

A detailed experimental analysis is reported in Section 6.1, evaluating the quality of our defense mechanisms and also considering state-of-the-art attacks and existing defense schemes.

The work presented in this chapter (including the related experiments) has been published on the IEEE Transactions on Pattern Analysis and Machine Intelligence (Melacci et al., 2021).

## 3.1 Learning with Domain Knowledge

In the context of this chapter, we focus on a multi-label classification problem in which each input $x \in \mathcal{X}$ is associated with one or more of the $c$ classes. We consider the case in which additional *domain knowledge* is available for the problem at hand, represented by a set of relationships that are known to exist among (a subset of) the $c$ classes. Exploiting such knowledge when training the classifier is the main subject of this section, and it has been shown to improve the generalization skills of the model (Gnecco et al., 2015; Gori and Melacci, 2013; Diligenti et al., 2017). The introduction of domain knowledge in the learning process provides precious information only when the training data are not fully labeled, as in the classic semi-supervised framework. Some examples might be partially labeled (i.e., for each data point, a subset of the $c$ classes participates in the ground truth), or a portion of the training set might be unsupervised. Of course, if the data are fully labeled, then all the class relationships are already encoded in the supervision signal. However, in this thesis, we also consider domain knowledge as a means to define a criterion that can spot potentially adversarial examples at test time, as we will discuss in Section 3.2, and that is also feasible in fully-supervised learning problems.

**Notation.** Formally, we consider the vector function defined in Equation (2.1), where each function $f_i$ is responsible for implementing a specific task on the input domain $\mathcal{X}$.[1] In a classification problem, function $f_i$ predicts the membership degree of $x$ to the $i$-th class. Moreover, when we restrict the output of $f_i$ to $[0, 1]$, we can think of $f_i$ as the fuzzy logic predicate that models the truth degree of belonging to class $i$. In order to simplify the notation, we will frequently make no explicit distinctions between function names, predicate names, class names, or between input samples and predicate variables. Whenever we focus on the predicate-oriented interpretation of each $f_i$, First-Order Logic (FOL) becomes the natural way of describing relationships among the classes, i.e., the most effective type of domain knowledge that could be eventually available in a multi-label problem; e.g., $\forall x \in \mathcal{X},\ f_i(x) \wedge f_j(x) \Rightarrow f_k(x)$, for some $i, j, k$, meaning that the intersection between the $i$-th class and the $j$-th class is always included in the $k$-one.

**Learning from Constraints.** The framework of Learning from Constraints (Gnecco et al., 2015; Gori and Melacci, 2013; Diligenti et al., 2017) follows the idea of converting domain knowledge into constraints on the learning problem, and it studies, amongst a variety of other knowledge-oriented constraints (see, e.g., Table 2 in Gnecco et al. (2015)), the process of handling FOL formulas so that they can be both injected into the learning problem or used as a knowledge verification measure (Gori and Melacci, 2013; Diligenti et al., 2017). Such knowledge is enforced on those training examples for which either no information or only partial/incomplete labeling is available, thus casting the learning problem in the semi-supervised

[1] This notion can be trivially extended to the case in which the task functions operate in different domains.

setting. As a result, the multi-label classifier can improve its performance and make predictions on out-of-sample data that are more coherent with the domain knowledge (see, e.g., Table 4 in Gnecco et al. (2015)). In particular, FOL formulas that represent the domain knowledge of the considered problem are converted into numerical constraints using Triangular Norms (T-Norms, from Klement et al. (2013)), binary functions that generalize the conjunction operator $\wedge$. Following the previous example, $f_i(x) \wedge f_j(x) \Rightarrow f_k(x)$ is converted into a bilateral constraint $\phi(f(x)) = 1$ that, in the case of the product T-Norm, is $1 - f_i(x)f_j(x)(1 - f_k(x)) = 1$. The 1 on the right-hand side of the constraint is due to the fact that the numerical formula must hold true (i.e., 1), while the left-hand side is in $[0, 1]$. We indicate with $\hat{\phi}(f(x))$ the loss function associated to $\phi(f(x))$. In the simplest case (the one followed in this thesis) such loss is $\hat{\phi}(f(x)) = 1 - \phi(f(x))$, where the minimum value of $\hat{\phi}(f(x))$ is zero. The quantifier $\forall x \in \mathcal{X}$ is translated by enforcing the constraints on a discrete data sample $\mathcal{U} \subset \mathcal{X}$. The loss function $\varphi(f, \mathcal{U})$ associated with all the available FOL formulas $\mathcal{K}$ is obtained by aggregating the losses of all the corresponding constraints and averaging over the data in $\mathcal{U}$.
Since we usually have $\ell > 1$ formulas whose relative importance could be uneven, we get

$$\varphi(f, \mathcal{U}, \mathcal{K}, \mu) = \frac{1}{|\mathcal{U}|} \sum_{j=1}^{|\mathcal{U}|} \left( \sum_{k=1}^{\ell} \mu_k \hat{\phi}_k(f(x_j)) \right) \in [0, \zeta], \tag{3.1}$$

where $\mu$ is the vector that collects the scalar weights $\mu_k > 0$ of the FOL formulas, and $\zeta = \sum_{k=1}^{\ell} \mu_k$.
In this specific context, $f$ is implemented with a neural architecture with $c$ output units and weights collected in $W$. We distinguish between the use of Equation (3.1) as a loss function in the training stage and its use as a measure to evaluate the constraint fulfillment on out-of-sample data. In detail, the classifier is trained on the training set $\mathcal{L}$ by minimizing

$$\min_{W} \left[ \texttt{suploss}(f(\cdot, W), \mathcal{L}, \Gamma) + \lambda \cdot \varphi(f(\cdot, W), \mathcal{L}, \mathcal{K}, \mu^{\mathcal{L}}) \right], \tag{3.2}$$

where $\mu^{\mathcal{L}}$ is the importance of the FOL formulas at training time, and $\lambda > 0$ modulates the weight of the constraint loss with respect to the supervision loss `suploss`, being $\Gamma$ the supervision information attached to some of the data in $\mathcal{L}$. The optimal $\lambda$ is chosen by cross-validation, maximizing the classifier performance. When the classifier is evaluated on a test sample $\overline{x}$, the measure

$$\varphi(f(\cdot, W), \{\overline{x}\}, \mathcal{K}, \mu^{\mathcal{T}}) \in [0, \zeta^{\mathcal{T}}], \tag{3.3}$$

with weights $\mu^{\mathcal{T}}$ and $\zeta^{\mathcal{T}} = \sum_{k=1}^{\ell} \mu_k^{\mathcal{T}}$, returns a score that indicates the fulfillment of the domain knowledge on $\overline{x}$ (the lower the better). Note that $\mu^{\mathcal{L}}$ and $\mu^{\mathcal{T}}$ might not necessarily be equivalent, even if certainly related. In particular, one may differently weigh the importance of some formulas during training to better accommodate the gradient-descent procedure and avoid bad local minima.

It is important to notice that Equation (3.2) enforces domain knowledge only on the training data $\mathcal{L}$. There are no guarantees that such knowledge will be fulfilled in the whole input space $\mathcal{X}$. This suggests that optimizing Equation (3.2) yields a stronger fulfillment of knowledge $\mathcal{K}$ over the space regions where the training points are distributed (low values of $\varphi$), while $\varphi$ could return larger values when departing from the distribution of the training data. The constraint enforcement is soft so that the second term in Equation (3.2) is not necessarily zero at the end of the optimization.

## 3.2 Exploiting Domain Knowledge against Adversarial Attacks

The constraint loss of Equation (3.1) is not only useful to enforce domain knowledge into the learning problem but also (*i*) to gain some robustness with respect to adversarial attacks and (*ii*) as a tool to detect adversarial examples at no additional training cost.
The underlying idea of our defense approach is conceptually represented in Figure 3.1. At training time, domain-knowledge constraints are enforced on the unlabeled (or partially-labeled) data to learn decision boundaries that better align with the marginal distributions. At test time, the same constraints can be efficiently evaluated on the test samples to identify and reject incoherent predictions, ideally outside of the training data distribution, potentially including adversarial examples. Our approach can also be used in single-classification tasks where domain knowledge and auxiliary classes are present and can be exploited internally by the classifier to implement the rejection mechanism based on domain-knowledge constraints. We will show some concrete examples of this latter setting in our experiments (Section 6.1), reporting comparisons with state-of-the-art adversarial attacks and concurrent defenses developed for single-classification tasks.

**A Paradigmatic Example.** The example in Figure 3.2 illustrates the main principles followed in this work in a multi-label classification problem with 4 classes (cat, animal, motorbike, vehicle) for which the following domain knowledge is available, together with labeled and unlabeled training data:

$$\forall x, \text{CAT}(x) \Rightarrow \text{ANIMAL}(x)\,, \tag{3.4}$$
$$\forall x, \text{MOTORBIKE}(x) \Rightarrow \text{VEHICLE}(x)\,, \tag{3.5}$$
$$\forall x, \text{VEHICLE}(x) \Rightarrow \neg\text{ANIMAL}(x)\,, \tag{3.6}$$
$$\forall x, \text{CAT}(x) \vee \text{ANIMAL}(x) \vee \text{MOTORBIKE}(x) \vee \text{VEHICLE}(x). \tag{3.7}$$

Such knowledge is converted into numerical constraints, as described in Section 3.1, while the loss function $\varphi$ is enforced on the training data predictions during classifier training (Equation (3.2)). Figure 3.2 shows two examples of the learned classifier.

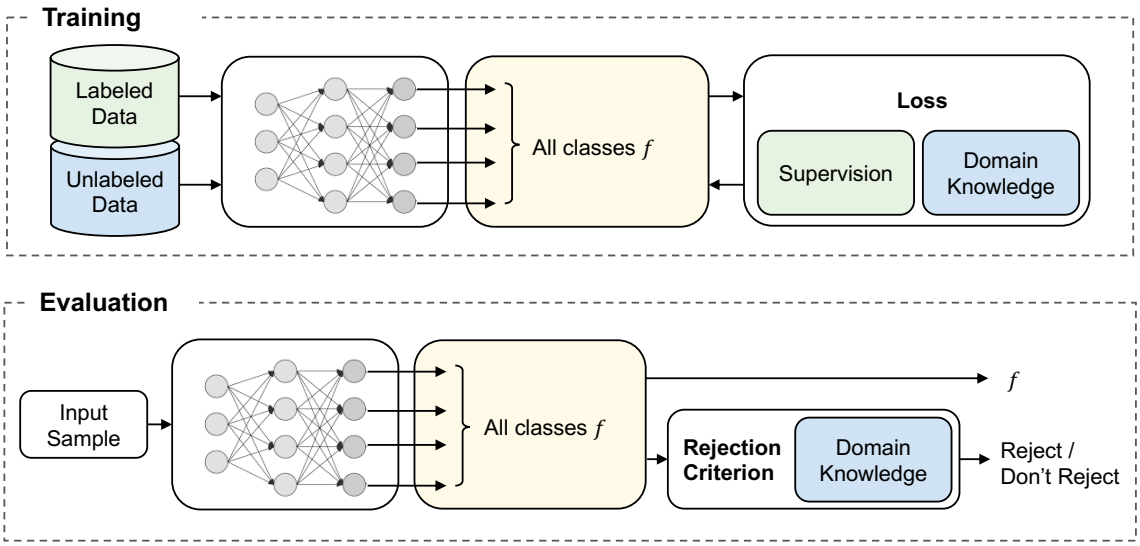


Figure 3.1: Leveraging domain knowledge to improve the robustness of multi-label classifiers. At training time, domain knowledge is used to enforce constraints on the learning process using unlabeled or partially-labeled data. At evaluation time, domain-knowledge constraints are used to detect and reject samples outside of the training data distribution.

Considering point $(i)$, in both cases, the decision boundaries are altered on the unlabeled data, enforcing the classifier to take a knowledge-coherent decision over the unlabeled training points and to better cover the marginal distribution of the data. This knowledge-driven regularity improves classifier robustness to adversarial attacks, as we will discuss in Section 6.1.

Going into further details to illustrate claim $(ii)$, in (a) we have the most likely case, in which decision boundaries are not always perfectly tight to the data distribution, and they might not be closed (ReLU networks typically return high-confidence predictions far from the training data (Hein et al., 2019)). Three different attacks are shown (purple). In attack 1, an example of a motorbike is perturbed to become an element of the cat class, but Equation (3.4) is not fulfilled anymore. In attack 2, an example of an animal is attacked to avoid being predicted as an animal. However, it falls in a region where no predictions are yielded, violating Equation (3.7). Attack number 3 consists of an adversarial attack to create a fake cat that, however, is also predicted as a vehicle, thus violating Equation (3.4) and Equation (3.6). In (b), we have an ideal and extreme case with very tight and closed decision boundaries. Some classes are well separated, it is harder to generate adversarial examples by slightly perturbing the available data, while it is easy to fall in regions for which Equation (3.7) is not fulfilled. The pictures in (c-d) show the unfeasible regions in which the constraint loss $\varphi$ is significantly larger, thus offering a natural criterion to spot adversarial examples that fall outside of the training data distribution.

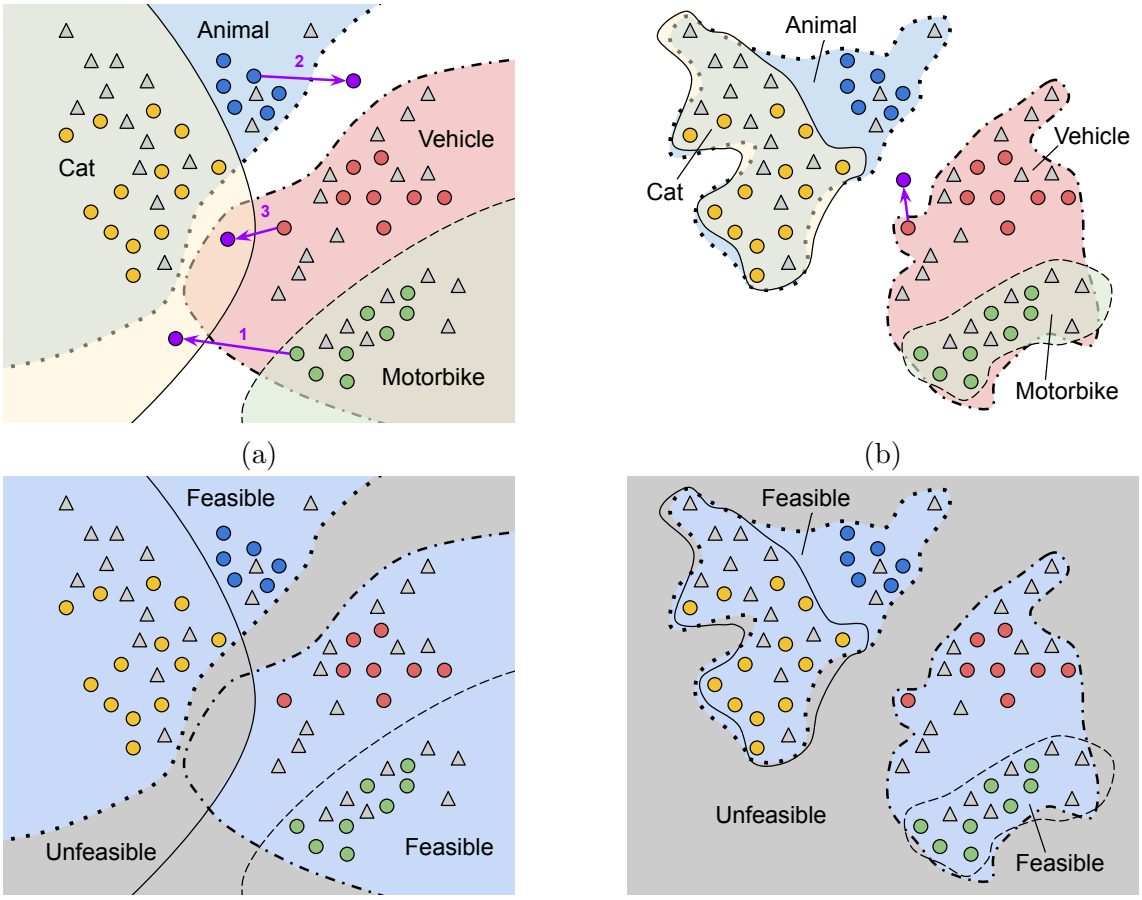


Figure 3.2: Toy example using the domain knowledge of Eqs. (3.4)- (3.7)) on 4 classes: cat (yellow), animal (blue), motorbike (green), vehicle (red). Labeled/unlabeled training data are depicted with rounded dots/gray triangles. (a,b) The decision regions for each class are shown in two sample outcomes of the training procedure: (a) open/loose decision boundaries; (b) tight/closed decision boundaries. The white area is associated with no predictions. Some adversarial examples (purple arrows/dots) are detected as they end up in regions that violate the constraints. Moreover, in (c,d), The feasible/unfeasible regions (blue/gray) that fulfill/violate the constraints for (a,b) are shown. Decision boundaries of the classes in (a,b) are also depicted in (c,d).

**Domain Knowledge-based Rejection.** Following these intuitions, and motivated by the approach of Hendrycks and Gimpel (2017b) and Hendrycks and Gimpel (2017a), we define a rejection criterion $\Psi$ as the Boolean expression

$$\Psi(\overline{x}, \theta | f(\cdot, W), \mathcal{K}, \mu^{\mathcal{T}}) = \varphi(f(\cdot, W), \{\overline{x}\}, \mathcal{K}, \mu^{\mathcal{T}}) > \theta, \tag{3.8}$$

where $\theta > 0$ is estimated by cross-validation in order to avoid rejecting (or rejecting a small number of[2]) the examples in the validation set $\mathcal{V}$. Equation (3.8) evaluates the constraint loss on the validation data $\mathcal{V}$, using the importance weights $\mu^{\mathcal{T}}$ (that we will discuss in what follows), as in Equation (3.3). The rationale behind this idea is that those samples for which the constraint loss is larger than what it is on the distribution of the data that are available when training/tuning the classifier should be rejected. The training samples are the ones over which domain knowledge was enforced during the training stage, while the validation set represents data on which knowledge was not enforced but that are sampled from the same distribution from which the training set is sampled, making them good candidates for estimating $\theta$. Notice that $\Psi$ is measured at test time on an already trained classifier, and it can be used independently on the nature of the training data (fully or partially/semi-supervised). Differently from ad-hoc detectors, which usually require training generative models, this rejection procedure comes at no additional training cost.[3]

**Pairing Effect.** The procedure is effective whenever the functions in $f$ are not too strongly paired with respect to $\mathcal{K}$, and we formalize the notion of "pairing" as follows.

**Definition 3.2.1.** *Pairing.* We consider a classification problem whose training data are distributed accordingly to the probability density $p(x)$. Given $\mathcal{K}$ and $\mu^{\mathcal{T}}$, the functions in $f$ are strongly paired whenever $\tau(\mathcal{H}, \mathcal{L}) = \|\varphi(f(\cdot, W), \mathcal{H}, \mathcal{K}, \mu^{\mathcal{T}}) - \varphi(f(\cdot, W), \mathcal{L}, \mathcal{K}, \mu^{\mathcal{T}})\| \approx 0$, being $\mathcal{H}$ a discrete set of samples uniformly distributed around the support of $p(x)$.

This notion indicates that if the constraint loss is fulfilled in similar ways over the training data distribution and space areas close to it, then there is no room for detecting those examples that should be rejected. While it is not straightforward to evaluate pairing before training the classifier, the soft constraining scheme of Equation (3.2) allows the classification functions to be paired in a less strong manner than what they would be when using hard constraints.[4] Note that a multi-label system is usually equipped with activation functions that do not structurally enforce any dependencies among the classes (e.g., differently from what happens with softmax),

[2] 10% in our experiments.

[3] Generative models on the fulfillment of the single constraints could be considered too.

[4] See Teso (2019) for a discussion on hard constraints and graphical models in an adversarial context.

so it is naturally able to respond without assigning the input to any class (white areas in Figure 3.2). This property has been recently discussed as a means for gaining robustness to adversarial examples (Shafahi et al., 2019; Bendale and Boult, 2016). The formula in Equation (3.7) is what allows our model to spot examples that might fall in this "I don't know" area. Dependencies among classes are only introduced by the constraint loss $\varphi$ in Equation (3.2) on the training data.
The choice of $\mu^{\mathcal{T}}$ is crucial in the definition of the reject function $\Psi$. On the one hand, in some problems, we might have access to the certainty degree of each FOL formula that could be used to set $\mu^{\mathcal{T}}$, otherwise, it seems natural to select an unbiased set of weights $\mu^{\mathcal{T}}$, $\mu_k = 1$, $\forall k$. On the other hand, several FOL formulas involve the implication operator $\Rightarrow$, which naturally implements if-then rules (if class $i$ then class $j$) or, equivalently, rules that are about hierarchies, since $\Rightarrow$ models an inclusion (class $i$ included in class $j$). However, whenever the premises are false, the whole formula holds true. It might be easy to trivially fulfill the associated constraints by zeroing all the predicates in the premises, eventually avoiding rejection. As a rule of thumb, it is better to select $\mu_k$'s that are larger for those constraints that favor the activation of the involved predicates.

**Single-label Classifiers.** The type of domain knowledge described so far usually involves logic formulas that encode relationships among multiple classes, thus it is naturally associated with multi-label problems. Let us focus our attention on multi-label scenarios in which there exists a subset of categories that are known to be mutually exclusive, that we will refer to as *main classes*, while the remaining categories will be referred to as *auxiliary classes*. If we restrict the original classification problem to the main classes only, we basically end up in a single-label scenario. Let us assume that the available logic formulas introduce relationships between (some of) the main classes and (some of) the auxiliary ones. As a result, in order to set up our defense mechanism (Equation (3.8)) or to learn with domain knowledge (Equation (3.2)), predictions on both the main and auxiliary classes must be available, so that the truth degree of the logic formulas can be evaluated. This consideration can be exploited to design classifiers that expose single-label predictions on the main classes, thus acting as single-label classifiers, and include predictions on the auxiliary classes that are not exposed to the user at all but that are internally used to set up our defense mechanism or to improve the quality of the whole classifier when learning in a semi-supervised context. Formally, let us assume that the components $\{f_i, i = 1, \ldots, c\}$ of the vector function $f$ are partitioned into two disjoint subsets, where the first one considers the components about the mutually-exclusive main classes and the second subset is about the auxiliary classes. We define with $f^v$ the vector function with the elements in the first subset, while $f^a$ is the vector function based on the elements of the second one, as shown in Figure 3.3. The system only exposes to the user predictions computed by means of $f^v$, while the computations of $f^a$ are hidden. Overall, the system can still exploit domain knowledge that consists of relationships between the classes associated with $f^v$ (main classes) and the ones

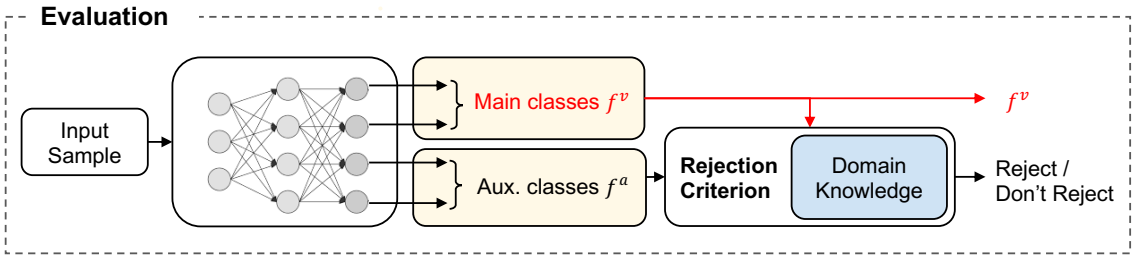


Figure 3.3: Single-label classifier on a set of mutually exclusive classes (*main classes*), computing the class activations by $f^v$ and exposing them to the user (red path). It internally computes by $f^a$ additional predictions over *auxiliary classes* that are involved in the domain knowledge (together with the main classes). Training considers *all* the classes, Figure 3.1.

associated with $f^a$ (auxiliary classes) or among the ones in $f^a$ only, thus leveraging the learning principles that were described in Section 3.1. Moreover, the system can exploit the hidden predictions and the available knowledge to implement the knowledge-based rejection mechanism that we proposed in this section, as sketched in Figure 3.3.

Due to the single-label nature of the visible portion of the classifier, existing state-of-the-art attacks, specifically designed for single-label models, can be used to fool the classifier in a black-box scenario. In Section 6.1, when the considered data are compatible with this special setting, we will exploit recent attack procedures to generate adversarial examples and evaluate the proposed knowledge-based rejection mechanism. Of course, different from what we previously stated about the real multi-label setting, we cannot consider the cost of the rejection mechanism negligible in this case since the system must learn the functions in $f^a$ in order to be able to compute the rejection criterion.

### 3.2.1 Attacking Multi-label Classifiers

As described in Section 2.4.2 and 2.4.4, the robustness against adversarial examples is typically evaluated against *black-box* and *white-box* attacks. In the black-box setting, the attacker is assumed to ignore the presence of our defense mechanisms, without having access to any additional domain knowledge and related constraints. In the white-box setting, instead, the attacker is assumed to know everything about the target model, including the defense mechanism. White-box attacks are thus expected also to exploit the available domain knowledge to try to bypass the knowledge-based defense.

The existing literature on the generation of adversarial examples is strongly focused on single-label classification problems (see Miller et al. (2020) and references therein). In such a context, the classifier is expected to take a decision that is only about one of the $c$ classes, and, in a nutshell, attacking the classifier boils down to

perturbing the input in order to make the classifier predict a wrong class. The whole procedure is subject to constraints on the amount of perturbation that the system is allowed to apply. Formally, given $x \in \mathcal{T}$, being $\mathcal{T}$ the test set, the attack generation procedures in single-label classification commonly solve the following problem,

$$\begin{aligned} x^\star = \ & \arg\min_{x'}[-\texttt{suploss}(f(x', W), \mathcal{L}, \Gamma)], \\ \text{s.t.} \quad & \|x - x'\| < \varepsilon, \end{aligned} \tag{3.9}$$

being $\|\cdot\|$ an $L_p$-norm and $\varepsilon > 0$. Each $x$ has a unique class label/index attached to it and stored in $\Gamma$, and `suploss` is usually the cross-entropy loss. Different attacks and optimization techniques for solving the problem of Equation (3.9) have been proposed (Croce and Hein, 2020a). While there are no ambiguities on the class on which we want the classifier to reduce its confidence, i.e., the ground-truth (positive) class of the given input $x$, the class that the classifier will predict in input $x^\star$ might be given or not, thus each of the remaining $c - 1$ classes could be a valid option. When moving to the multi-label setting, each $x \in \mathcal{T}$ is associated with multiple ground-truth positive classes, collected in set $P_x$, and we indicate with $N_x$ the set of ground-truth negative classes of $x$. Different from the previous case, due to the lack of mutual exclusivity of the predictions, creating an adversarial example out of $x$ is more arbitrary. For example, the optimization procedure could focus on making the classifier not able to predict any of the classes in $P_x$ or a subset of them. Similarly, the optimization could focus on making the classifier positively predict one or more classes of $N_x$.

Departing from the overwhelming majority of existing attacks for single-label classifiers, we propose a multi-label attack that focuses on the classes on which the classifier is less confident (thus easier to attack), that are selected and re-defined during the optimization procedure in function of the way the predictions of the classifier progressively change. Of course, in the *black-box* case, this attack is not considering that classes are related, and it is not taking care that, perhaps, changing the prediction on a certain class should also trigger a coherent change in other related classes. Differently, in the *white-box* setting, the previously introduced domain knowledge and, in particular, the corresponding loss of Equation (3.1) is what encodes such relationships in a differentiable way so that we can easily exploit it when crafting attacks. We first introduce the proposed multi-label attack in a *black-box* setting, in which domain knowledge is not available. To make gradient computation numerically more robust, as in Carlini and Wagner (2017b), we consider the activations (logits) of the last layer of $f$ to compute the objective function instead of using the cross-entropy loss.

Let us define $p = \arg\min_i[f_i(x),\ i \in P_x]$, and $n = \arg\max_i[f_i(x),\ i \in N_x]$, i.e., $p$ ($n$) is the index of the positive (negative) class with the smallest (largest) output score. These are essentially the indices of the classes for which $x$ is closer to the decision

boundaries. Our attack optimizes the following objective,

$$\begin{aligned} x^\star = \ & \arg\min_{x'}[\max(l_p(x'), -\kappa) - \min(l_n(x'), \kappa)] \\ \text{s.t.} \quad & \|x - x'\| < \varepsilon, \end{aligned} \tag{3.10}$$

where $l_j$ is the value of the logit of $f_j$, $\|\cdot\|$ is an $L_p$-norm ($L_2$ in our experiments), and in the case of image data with pixel intensities in $[0, 1]$ we also have $x' \in [0, 1]$. The scalar $\kappa \geq 0$ is used to threshold the values of the logits to avoid increasing/decreasing them in an unbounded way (in our experiments, we set $\kappa = 2$). Optimizing the logit values is preferable to avoid sigmoid saturation. While the definition of Equation (3.10) is limited to a pair of classes, we *dynamically* update $p$ and $n$ whenever logit $l_p$ ($l_n$) goes beyond (above) the threshold $-\kappa$ ($\kappa$), thus multiple classes are considered by the attack, compatibly with the maximum number of iterations of the optimizer. This strategy resulted in being more effective than jointly optimizing all the classes in $P_x$ and $N_x$. Moreover, the classes involved in the attack can be a subset of the whole set, as in Song et al. (2018). In a *white-box* scenario, when the attacker has the use of the domain knowledge, the information in $\mathcal{K}$ provides a comprehensive description of how the predictions of the classifier should be altered over several classes in order to be coherent with the knowledge. In such a scenario, we enhance Equation (3.10) to implement what we refer to as multi-label knowledge-driven adversarial attack (MKA), including the differentiable knowledge-driven loss $\varphi$ in the objective function,

$$\begin{aligned} x^\star = \ & \arg\min_{x'}[\max(l_p(x'), -\kappa) - \min(l_n(x'), \kappa) + \\ & \rho \cdot \varphi(f, \{x'\}, \mathcal{K}, \mu^{\mathcal{T}})], \quad \text{s.t. } \|x - x'\| < \varepsilon, \end{aligned} \tag{3.11}$$

in which we set $\rho > 0$ to enforce domain knowledge and avoid rejection. When crafting adversarial examples, MKA softly enforces the fulfillment of domain knowledge by means of the loss function $\varphi$. For *black-box* attacks, instead, we set $\rho = 0$ to recover Equation (3.10). MKA naturally extends the formulation of single-label attacks (when $P_x$ is composed of a single class) and it allows the staging of both black-box and white-box (adaptive) attacks against our approach. Equation (3.11) is minimized via projected gradient descent (1000 samples and 50 iterations in our experiments).

### 3.2.2 Impact of Domain Knowledge and Main Issues

Our approach is built around the idea of exploiting the available domain knowledge $\mathcal{K}$ on the target classification problem, both in the cases of rejection and multi-label attack. Several existing works use additional knowledge on the learning problem with different goals, being it represented by logic (d'Avila Garcez et al., 2019; Diligenti et al., 2017; Gnecco et al., 2015; Gori and Melacci, 2013), inherited by knowledge graphs or other external resources (Melacci et al., 2018; Yu and Dredze, 2014),

and encoded in multiple ways to face specific tasks (Pi et al., 2017; Morgado and Vasconcelos, 2017; Melacci et al., 2018). For instance, Semantic-based Regularization (Diligenti et al., 2017) and the theory formalized in Gnecco et al. (2015) focus on the same approach we use here to convert generic FOL knowledge. On one hand, $\mathcal{K}$ might not always be available, thus limiting the applicability of what we propose and of the other aforementioned approaches. On the other hand, $\mathcal{K}$ is about relationships among classes that, in the case of the universal quantifier, hold $\forall x$. As a result, such knowledge is more generic than specific example-level supervision. Human experts can produce FOL rules to a lesser effort than what is needed to manually label large batches of examples since $\mathcal{K}$ naturally represents the type of high-level knowledge on the target domain that a human would develop during a concrete experience on the considered task (e.g., *if A happens, then also B or C are triggered, but not D*). Moreover, we are currently working on methods to extract the type of knowledge that we consider in this chapter by means of special neural architectures, with clear connections to Explainable AI (Ciravegna et al., 2020; Barbiero et al., 2022).

When the number $\ell$ of FOL formulas in $\mathcal{K}$ is large, a larger number of penalty terms $\hat{\phi}_k$ will be considered in $\varphi$ of Equation (3.1). Of course, every approach that exploits additional knowledge usually incurs increased complexity when the knowledge base is large (d'Avila Garcez et al., 2019; Diligenti et al., 2017; Gnecco et al., 2015; Gori and Melacci, 2013). In our case, the T-Norm-based conversion does not represent an issue since it is computed only once in a pre-processing stage, and similarly, the output of the network $f(x, W)$ is computed only once in order to evaluate $\varphi$ for a certain sample $x$ and for given weights $W$, independently on the size of $\mathcal{K}$. However, the computation of $\varphi$ must be repeated at each iteration of the optimization of Equation (3.2) or Equation (3.11), and when evaluating whether an input should be rejected or not, Equation (3.8). From the practical point of view, the computational complexity scales almost linearly with $\ell$, but each $\hat{\phi}_k$ has a different structure depending on the FOL formula from which it was generated—roughly speaking, formulas involving more predicates usually yield more complex T-Norm-based polynomials. Several heuristic solutions are indeed possible to overcome these issues. For example, the knowledge base could be sub-selected in order to bind the number of rules in which each class is involved, or a stochastic optimization could be devised to sample the rules included in $\varphi$ at each iteration of the optimization process. However, we remark that in the related experimental activities none of the mentioned issues arose.

The way we convert FOL rules into polynomial constraints, described in Section 3.1, inherits the flexibility of logic in terms of knowledge representation capabilities. Of course, the concrete impact of $\mathcal{K}$ in the rejection mechanisms or in MKA depends on the specific information that is encoded by the FOL rules. For instance, suppose that $f_i(x) = 1$ for a certain $x$. The formula $f_i(x) \Rightarrow f_j(x) \vee \ldots \vee f_u(x)$ is "more likely" to be fulfilled than the formula with an analogous structure in which $\vee$'s are

replaced by $\wedge$'s. In the former, it is enough for a predicate in the conclusions to be 1, while in the latter, all the predicates of the conclusions must be jointly true. The rejection criterion or MKA is likely to be more effective in the latter case, but it cannot be strongly stated in advance since it depends on the concrete way in which $f(\cdot, W)$ is developed by the learning procedure, as discussed in Section 3.2, and, in the case of MKA, on the difficulty in optimizing Equation (3.11).

When restricting our attention to the rejection function of Equation (3.8), a key element to the success of the proposed criterion is the choice of $\theta$. In Section 3.2, we suggested using data in $\mathcal{V}$ to tune $\theta$. Although this is a valuable solution, it strongly depends on the quality of $\mathcal{V}$, similarly to what happens when tuning other hyper-parameters. More generally, a too-small $\theta$ will result in a reject-prone system that does not reject only those inputs that are strongly coherent with the domain knowledge. A too large $\theta$ would end up in not rejecting inputs, being them coherent with $\mathcal{K}$ or not. If further information on the formulas in $\mathcal{K}$ is available, such as their expected importance with respect to the considered task, one could avoid computing an averaged measure as $\varphi$ and evaluate the penalty term $\hat{\phi}_k$ of every single formula against its own reject threshold (i.e., multiple $\theta$'s), that might be selected accordingly to the importance of the formula itself (i.e., smaller $\theta$'s in more important formulas).

## 3.3 Related Work

We emphasize here the main differences between what we propose in this chapter with respect to the most strongly related approaches.

**Multi-label Adversarial Perturbations.** Most of the work in the adversarial ML area focuses on single-label classification problems. To the best of our knowledge, the first and only study on this problem is the one in Song et al. (2018), in which the authors focus on targeted multi-label adversarial perturbations defining in advance the set of classes on which the attack is targeted (being them positive or negative) and also introducing another set of classes for which the attack is expected not to change the classifier predictions. The framework described in Song et al. (2018) is only experimented in a *static/targeted* context, i.e., by selecting in advance the sets mentioned above using custom criteria to simulate the attacking scenario artificially. The multi-label attack that we propose in this work is instead *dynamic/untargeted* and without the need to define in advance what are the classes to be considered. Regarding the defenses, to our best knowledge, none of the previously proposed ones leverage multi-label classification outputs.

**Semi-supervised Learning and Adversarial Training.** In the context of adversarial machine learning, unlabeled data are usually employed to improve the robustness of the classifier by performing adversarial training. The rationale behind such a training scheme is that if the available unlabeled samples are perturbed, then the predicted class should not change. Miyato et al. (2016), Miyato et al.

(2018) and Park et al. (2018) exploit adversarial training (virtual adversarial training and adversarial dropout, respectively) to favor regularity around the supervised and unsupervised training data and improve the classifier performance. The work in Akcay et al. (2018) develops an anomaly detector using adversarial training in the semi-supervised setting. Self-supervised learning is exploited in Carmon et al. (2019) and Najafi et al. (2019) to gain stronger adversarial robustness. Stability criteria are enforced on unlabeled training data in Schmidt et al. (2018), whereas the work in Alayrac et al. (2019) specifically focuses on an unsupervised adversarial training procedure in the context of semi-supervised classification. Our model neither exploits adversarial training nor any adversary-aware training criteria aimed at gaining intrinsic regularity. We focus on the role of domain knowledge as an indirect means to increase adversarial robustness and, afterward, to detect adversarial examples. Therefore, the proposed work is not attack-dependent, and it is faster at training time as it does not require generating adversarial examples. We believe that using unlabeled data also to simulate attacks and incorporate them into the training process may further improve robustness. All the described methods could also be applied jointly with what we propose.

**Rejection-based Approaches for Adversarial Examples.** A different line of defenses, complementary to adversarial training, is based on detecting and rejecting samples sufficiently far from the training data in feature space. Our approach differs from other adversarial-example detectors (Carlini and Wagner, 2017a; Ma et al., 2018; Samangouei et al., 2018; Miller et al., 2020) as it has no additional training cost and negligible runtime cost. We are the first to show that domain knowledge can be used to reject adversarial examples and also to propose a detector that exploits unlabeled data.

**Domain-Agnostic Methods and Semantic Attacks.** Recent work in adversarial attacks considers the role of the learning domain and of additional semantic information, even if with different goals to the ones of this chapter. The way the learning domain is related to the generation of attacks was recently studied in Naseer et al. (2019), which is based on the idea of developing generative adversarial perturbations that turn out to be easily transferable from the source domain (where the attack function is modeled) to another domain. Differently, we focus on knowledge that is domain-specific and used both for defending and creating more informed attacks. The knowledge of a set of semantic attributes is used to implement the threat model of semantic adversarial attacks in Joshi et al. (2019). A generative network is considered, and the attack procedure focuses on altering the activation of such human-understandable attributes, that, in turn, yield visible changes in the input image (e.g., adding glasses to the input face). Differently, our work is built on an $L_p$-norm-bounded perturbation model that does not enforce the input image to change in a human-understandable manner. Our approach considers a more generic notion of knowledge, that includes information also on the relationships within subsets of logic predicates, and that exploits the power of FOL. Predicate activations

are modeled by neural networks and not by scalar variables as for the attributes of Joshi et al. (2019).

# Chapter 4

# Detecting Adversarial Examples in Inner DNN layers

In this chapter, we propose a defense mechanism, named *Deep Neural Rejection* (DNR), based on analyzing the representations of input samples at *different* network layers and on rejecting samples that exhibit anomalous behavior with respect to that observed from the training data at such layers. In fact, it has been shown that only relying upon the feature representation learned by the last network layer to reject adversarial examples is not sufficient (Melis et al., 2017; Bendale and Boult, 2016). In particular, it happens that adversarial examples become indistinguishable from samples of the target class at such a higher representation level even for small input perturbations. With respect to similar approaches based on analyzing different network layers (Lu et al., 2017; Crecchi et al., 2019), our defense does not require generating adversarial examples during training, and it is thus less computationally demanding. We evaluate our defense against an adaptive white-box attacker that is aware of the defense mechanism and tries to bypass it. To this end, we propose a novel gradient-based attack that accounts for the rejection mechanism and aims to craft adversarial examples that avoid it.

The chapter is structured as follows. In Section 4.1, the DNR technique is introduced and formulated. Section 4.2 presents the adaptive attack we use to perform the robustness evaluation. Finally, in Section 4.3, related works are discussed.

Experimental results are reported in Section 6.2.

The work presented in this chapter (including the related experiments) has been published on the EURASIP Journal on Information Security (Sotgiu et al., 2020).

## 4.1 Deep Neural Rejection

The underlying idea of our DNR method is to estimate the distribution of unperturbed training points at different network layers and reject anomalous samples that

may be incurred at test time, including adversarial examples. The architecture of DNR is shown in Figure 4.1.

Before delving into the details of our method, let us recall some notation. We denote the prediction function of a deep neural network with $f : \mathcal{X} \mapsto \mathcal{Y}$, where $\mathcal{X} \subseteq \mathbb{R}^d$ is the $d$-dimensional space of input samples (e.g., image pixels) and $\mathcal{Y} \subseteq \mathbb{R}^c$ is the space of the output predictions (i.e., the estimated confidence values for each class), being $c$ the number of classes. If we assume that the network consists of $m$ layers, then the prediction function $f$ can be rewritten to make this explicit as: $f(h_1(h_2(\dots h_m(\boldsymbol{x}; \boldsymbol{w}_m); \boldsymbol{w}_2); \boldsymbol{w}_1)$, where $h_1$ and $h_m$ denote the mapping function learned respectively by the output and the input layer, and $\boldsymbol{w}_1$ and $\boldsymbol{w}_m$ are their weight parameters (learned during training).

For our defense mechanism to work, one has first to select a set of network layers; for instance, in Figure 4.1, we select the outer layers $h_1$, $h_2$, and $h_3$. Let us assume that the representation of the input sample $\boldsymbol{x}$ at level $h_i$ is $\boldsymbol{z}_i$. Then, on each of these selected representations, DNR learns an SVM with the RBF kernel $g_i(\boldsymbol{z}_i)$, trying to assign input samples to their respective classes.

The confidence values on the $c$ classes provided by this classifier are then concatenated with those provided by the other base SVMs and used to train a combiner, using again an RBF SVM.[1] The combiner will output predictions $s_1, \dots, s_c$ for the $c$ classes but will reject samples if the maximum confidence score $\max_{k=1,\dots,c} s_k$ is not higher than a rejection threshold $\theta$.

This decision rule can be compactly represented as: $\arg\max_{k=0,\dots,c} s_k(\boldsymbol{x})$, where we define an additional, *constant* output $s_0(\boldsymbol{x}) = \theta$ for the rejection class. According to this rule, if $s_0(\boldsymbol{x}) = \theta$ is the highest value in the set, the sample is rejected; otherwise, it is assigned to the class exhibiting the larger confidence value.

As proposed in Melis et al. (2017), we use an RBF SVM here to ensure that the confidence values $s_1, \dots, s_c$, for each given class, decrease while $\boldsymbol{x}$ moves further away from regions of the feature space which are densely populated by training samples of that class. This property, named *compact abating probability* in open-set problems (Scheirer et al., 2014; Bendale and Boult, 2016), is a desirable property to easily implement a *distance-based rejection* mechanism as the one required in our case to detect outlying samples.

With respect to Melis et al. (2017), we train this combiner on top of other base classifiers rather than only on the representation learned by the last network layer to further improve the detection of adversarial examples. For this reason, in the following, we refer to the approach by Melis et al. (2017), rejecting samples based only on their representation at the last layer, as *Neural Rejection* (NR); and to ours, also exploiting representations from other layers, as *Deep Neural Rejection* (DNR).

[1] Validation samples should be used to train the combiner here and avoid overfitting, as suggested by stacked generalization (Wolpert, 1992).

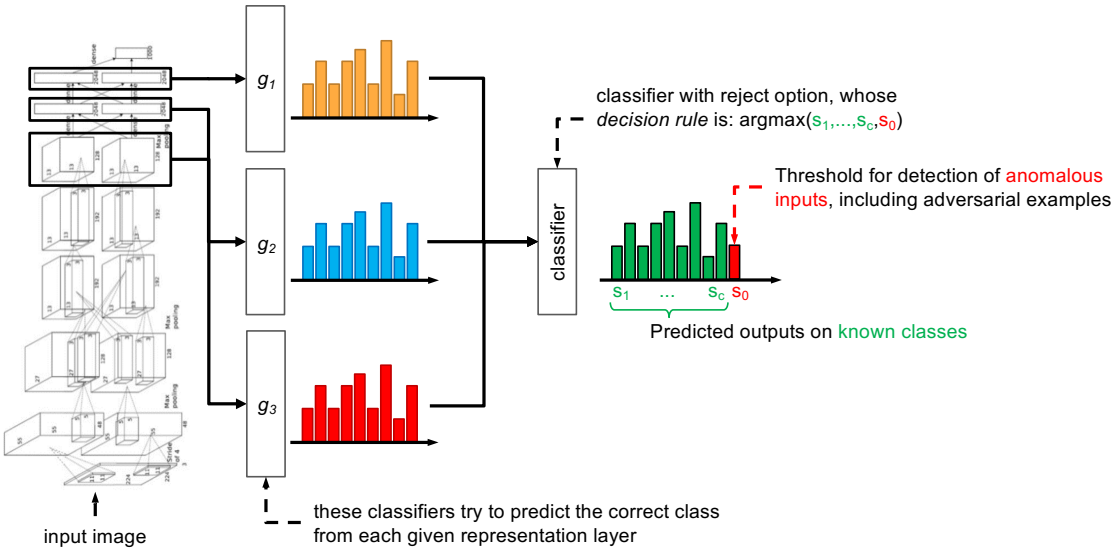


Figure 4.1: Architecture of Deep Neural Rejection (DNR). DNR considers different network layers and learns an SVM with the RBF kernel on each of their representations. The outputs of these SVMs are then combined using another RBF SVM, which will provide prediction scores $s_1, \ldots, s_c$ for each class. This classifier will reject samples if the maximum score $\max_{k=1,\ldots,c} s_k$ is not higher than the rejection threshold $\theta$. This decision rule can be equivalently represented as $\arg\max_{k=0,\ldots,c} s_k(\boldsymbol{x})$, if we consider rejection as an additional class with $s_0 = \theta$.

## 4.2 Attacking Deep Neural Rejection

A correct evaluation of proposed detection methods against adversarial examples is essential (Biggio and Roli, 2018; Athalye et al., 2018a), and it is not sufficient to evaluate such defenses against previous defense-unaware attacks that are likely to fail (Lu et al., 2017; Papernot et al., 2016b; Meng and Chen, 2017), leading to overly optimistic results in terms of classifier robustness. To perform a fair defense evaluation, attacks should take into account the defense mechanism. Under this condition, many defenses were shown not to be as effective as claimed (Carlini and Wagner, 2017a,b; Athalye et al., 2018a). When a defense exploits rejection, a defense-unaware attack may craft adversarial examples belonging to rejection regions, making it very difficult to evade such defense. To perform a fair robustness evaluation of the proposed defense method, an adaptive defense-aware attack is required.

We formulate here an adaptive white-box attack to properly evaluate the security, or adversarial robustness, of rejection-based defenses. Given a source sample $\boldsymbol{x}$, the attacker can compute a maximum-allowed $\varepsilon$-sized adversarial perturbation, obtaining the adversarial example $\boldsymbol{x}^\star$ by solving the following constrained optimization

problem:

$$\boldsymbol{x}^\star = \underset{\boldsymbol{x}':\|\boldsymbol{x}-\boldsymbol{x}'\|\leq\varepsilon}{\arg\min}\ \Omega(\boldsymbol{x}) \quad \text{where} \quad \Omega(\boldsymbol{x}) = s_y(\boldsymbol{x}') - \max_{j\notin\{0,y\}} s_j(\boldsymbol{x}')\,, \tag{4.1}$$

where $\|\boldsymbol{x}-\boldsymbol{x}'\| \leq \varepsilon$ is an $\ell_p$-norm constraint (typical norms used for crafting adversarial examples are $\ell_1$, $\ell_2$ and $\ell_\infty$, for which efficient projection algorithms exist (Duchi et al., 2008)), $y \in \mathcal{Y} = \{1,\ldots,c\}$ is the true class, and 0 is the rejection class.
Intuitively, to perform an untargeted (error-generic) evasion, the output of the true class must be minimized, and the output of one competing class (excluding the reject class) must be maximized. The resulting objective function is negative in case of successful evasion, and its absolute value increases with the increasing classification confidence of the competing class. The algorithm does not simply search for a minimum-distance adversarial example, but it maximizes the confidence of the attack. Although in this work we focus only on untargeted attacks, the proposed formulation can be easily extended to account for targeted (error-specific) evasion, as in Melis et al. (2017).

To solve the optimization problem above, we use a projected gradient descent (PGD) algorithm with a line search to optimize the step size, named PGD-LS (Algorithm 2). The initial step size $\eta_0$ is doubled ten times, computing the objective function at each step and choosing the step size which minimizes the function. The selected step size is then used to update the point $\boldsymbol{x}'$. Choosing a different $\eta$ at each step gives two main advantages:

- Speeding-up the optimization: using larger step sizes (when possible) allows us to reach the convergence with a reduced number of iteration steps.
- Escaping local minima which may hinder the optimization process, using the larger step sizes.

In fact, while attacking DNR, we found that the optimization often got stuck in local minima inside reject decision regions, where the objective function gradient reaches very small values close to zero. The magnitude of these gradients strongly depends on the value of the $\gamma$ parameter of SVM-RBF classifiers used by DNR, which is a negative exponent used in the kernel computation that controls the shape of decision regions around training samples. Larger values of $\gamma$ produce more complex decision regions and smaller gradient magnitudes, whereas smaller values of $\gamma$ conversely produce smoother decision regions. Our proposed attack exploiting an adaptive step size turns out to be more effective than standard fixed-size step attacks, as shown in the experiments of Section 6.3.3.
In Figure 4.2, we report an example of a bi-dimensional toy problem to show how our defense-aware attack works against a rejection-based defense mechanism.

**Algorithm 2:** PGD-LS: PGD-based Maximum-confidence Adversarial Examples with Exponential Line Search

**Input :** $\boldsymbol{x}_0$: the input sample; $\eta_0$: the initial step size; $\Pi$: a projection operator on the $\ell_p$-norm constraint $\|\boldsymbol{x}_0 - \boldsymbol{x}'\| \leq \varepsilon$; $t > 0$: a small positive number to ensure convergence.

**Output:** $\boldsymbol{x}'$: the adversarial example.

$\boldsymbol{x}' \leftarrow \boldsymbol{x}_0$
**repeat**
  $\boldsymbol{x} \leftarrow \boldsymbol{x}'$
  $\boldsymbol{x}''_0 \leftarrow \Pi\left(\boldsymbol{x} - \eta_0 \nabla\Omega(\boldsymbol{x})\right)$
  **for** $k = 1$ ***to*** $k < 10$ **do**
    $\eta_k \leftarrow 2^k \eta_0$
    $\boldsymbol{x}''_k \leftarrow \Pi\left(\boldsymbol{x} - \eta_k \nabla\Omega(\boldsymbol{x})\right)$
    **if** $\Omega(\boldsymbol{x}''_k) < \Omega(\boldsymbol{x}''_{k-1})$ **then**
      $\eta' \leftarrow \eta_k$
    $\boldsymbol{x}' \leftarrow \Pi\left(\boldsymbol{x} - \eta' \nabla\Omega(\boldsymbol{x})\right)$
**until** $|\Omega(\boldsymbol{x}') - \Omega(\boldsymbol{x})| \leq t$;
**return** $\boldsymbol{x}'$

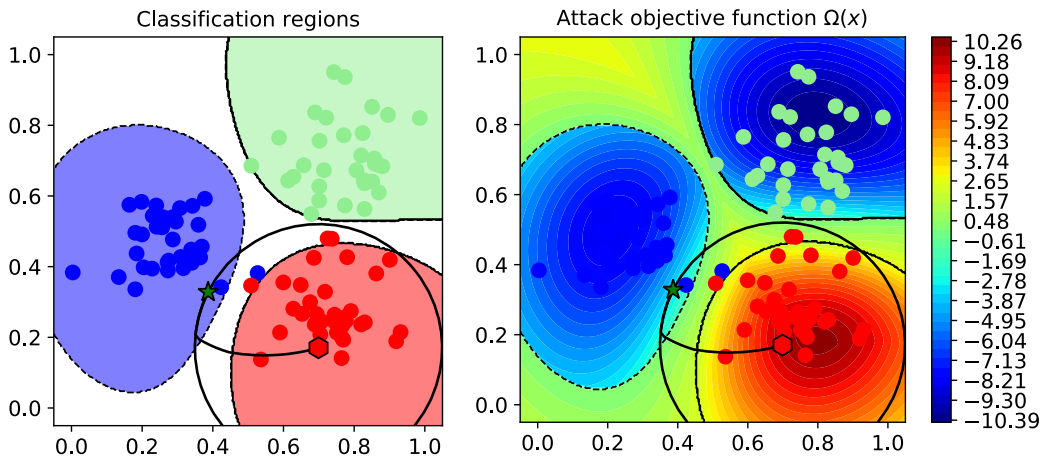


Figure 4.2: Our defense-aware attack against an RBF SVM with rejection on a 3-class bi-dimensional classification problem. The initial sample $x_0$ and the adversarial example $x^\star$ are respectively represented as a red hexagon and a green star, while the $\ell_2$-norm perturbation constraint $\|x_0 - x'\|_2 \leq \varepsilon$ is shown as a black circle. The left plot shows the decision region of each class, along with the reject region (in white). The right plot shows the values of the attack objective $\Omega(\boldsymbol{x})$ (in colors), which correctly enforces our attacks to avoid the reject region.

## 4.3 Related Work

Different approaches have been recently proposed to perform rejection of samples that are outside of the training data distribution (Thulasidasan et al., 2019; Geifman and El-Yaniv, 2019, 2017) For example, Thulasidasan et al. (2019) and Geifman and El-Yaniv (2019) have proposed novel loss functions accounting for rejection of inputs on which the classifier is not sufficiently confident. Geifman and El-Yaniv (2017) have proposed a method that allows the system designer to set the desired risk level by adding a rejection mechanism to a pre-trained neural network architecture. These approaches have, however, not been originally tested against adversarial examples, and it is thus of interest to assess their performance under attack in future work, also in comparison to our proposal.

Even if the majority of approaches implementing rejection or abstaining classifiers have not considered the problem of defending against adversarial examples, some recent work has explored this direction, too (Bendale and Boult, 2016; Melis et al., 2017). Nevertheless, with respect to the approach proposed in this work, they have only considered the output of the last network layer and performed rejection based solely on that specific feature representation. In particular, Bendale and Boult (2016) have proposed a rejection mechanism based on reducing the open-set risk in the feature space of the activation vectors extracted from the last layer of the network, while Melis et al. (2017) have applied a threshold on the output of an RBF SVM classifier. Despite these differences, the rationale of the two approaches is quite similar and resembles the older idea of distance-based rejection.

Few approaches have considered a multi-layer detection scheme similar to that envisioned in our work (Lu et al., 2017; Carrara et al., 2018; Crecchi et al., 2019; Pang et al., 2018; Papernot and Mcdaniel, 2018). However, most of these approaches require generating adversarial examples at training time, which is computationally intensive, especially for high-dimensional problems and large datasets (Lu et al., 2017; Carrara et al., 2018; Crecchi et al., 2019; Pang et al., 2018). Finding a methodology to tune the hyperparameters for generating the attack samples is also an open research challenge. Finally, even though the Deep k-Nearest Neighbors approach by Papernot and Mcdaniel (2018) does not require generating adversarial examples at training time, it requires computing the distance of each test sample against all the training points at different network layer representations, which again raises scalability issues to high-dimensional problems and large datasets.

# Chapter 5

# Improving the Efficiency of Prototypes-based Detectors

In this chapter, we start from the intuition that the vast majority of detector defenses in literature are a form of instance-based classifiers: in fact, when a new sample is fed to the classifier, it is compared with a set of prototypes to produce an output prediction. We thus provide a comprehensive review of such adversarial examples detection methods in the form of a unifying framework. Each proposed detector defense can be obtained by correctly instantiating our framework components. Subsuming each analyzed detector defense in the framework allowed us to identify common drawbacks, leading us to propose FADER, a technique for speeding up detection methods. It works by replacing the detector's distance-based classifiers with size-constrained RBF networks to reduce computational overhead at test time. The proposed solution is capable of enforcing adversarial robustness even in presence of adaptive attacks specifically designed to defeat the defense.

The chapter is structured as follows. In Section 5.1, we present our adversarial examples detector framework. Section 5.2 introduces FADER, our proposed fast detection method. Finally, Section 5.3 discusses complementary methods for addressing adversarial examples than detection strategies.

We performed an extensive experimental evaluation, which is reported in Section 6.3.

The work presented in this chapter (including the related experiments) has been published on the Neurocomputing journal (Crecchi et al., 2022).

## 5.1 A Framework for Adversarial Example Detection

The proposed detection framework, conceptually depicted in Figure 5.1, assumes an already-trained DNN classifier to be protected against adversarial examples, denoted

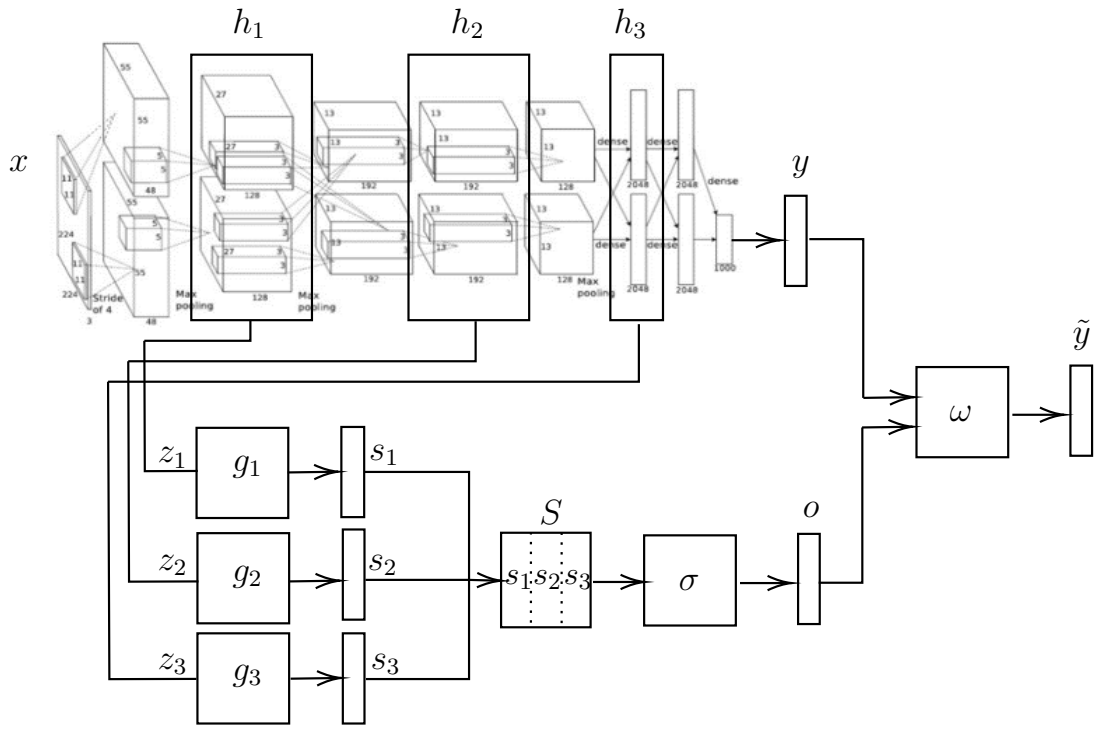


Figure 5.1: Architecture of the proposed framework for adversarial example detection. It extends a pre-trained DNN by attaching several layer detectors $g$ whose goal is to determine distribution drifts in the representation of an input $\boldsymbol{x}$ at a given layer. Multiple layer-detector predictions are combined and fed to a combiner classifier $\sigma$, which outputs the final detector prediction. The undefended network and detector outputs are combined by $\omega$ to provide the final predictions.

with $f : \mathcal{X} \to \mathcal{Y}$, being $\mathcal{X} \subseteq \mathbb{R}^d$ a $d$-dimensional input space, and $\mathcal{Y} \subseteq \mathbb{R}^c$ the output space consisting of $c$ classes.[1]

Building a detector amounts to selecting a set of network layers to inspect the internal DNN representations corresponding to each given input sample, with the goal of identifying anomalous (adversarial) patterns with respect to those exhibited by the natural samples. Let $\boldsymbol{z}_j = h_j(h_{j-1}(\ldots(h_1(\boldsymbol{x}; \boldsymbol{w}_1)); \boldsymbol{w}_{j-1}); \boldsymbol{w}_j)$ be the representation learned by the network for a given input sample $\boldsymbol{x}$ at layer $h_j$. A *layer detector* function $g_j : \mathcal{Z}_j \to \mathcal{S}_j$ is applied to $\boldsymbol{z}_j$ producing a score vector $\boldsymbol{s}_j$ of size $c$, where each element represents the probability that $\boldsymbol{x}$ belongs to a given class, according to the given representation $\boldsymbol{z}_j$.

The predictions coming from the $k$ different layer detectors are then stacked in a scoring matrix $\mathbf{S} \in \mathbb{R}^{k \cdot c}$, and combined by a *multilayer detector* $\sigma : \mathbb{R}^{k \cdot c} \to \mathcal{O}$, where $\boldsymbol{o} \in \mathcal{O}$ is the vector of predictions provided by the multilayer detector.

[1] We assume here that the classifier predicts continuous output values for each class and that the final decision is made as usual by selecting the class exhibiting the maximum support.

The multilayer detector may provide one output for each class, i.e., $\mathcal{O} = \mathcal{Y}$, or alternatively only produce a single score that measures how likely the input sample is retained anomalous.
The predictions of the multilayer detector may be finally combined with those coming from the undefended DNN to produce the final predictions for each input sample $\boldsymbol{x}$. This can be formalized as a function $\omega : \mathcal{Y} \times \mathcal{O} \to \tilde{\mathcal{Y}}$, being $\boldsymbol{y} \in \mathcal{Y}$ and $\boldsymbol{o} \in \mathcal{O}$ the DNN and the detector predictions, respectively, and $\tilde{\boldsymbol{y}} \in \tilde{\mathcal{Y}}$ a $c+1$ output vector which includes an additional rejection class reserved for the detected adversarial examples. For single-layer defenses, no combiner is clearly needed. In our framework, this corresponds to instantiating $\sigma$ as the identity function.

In the remainder of the section, we will rephrase the aforementioned adversarial-example defenses in terms of the proposed framework. As already mentioned, we are considering only rejection-based defenses against adversarial attacks. For the sake of clarity, the considered defenses are also schematized in terms of framework components in Table 5.1.

### 5.1.1 Neural Reject

Inspired by the concept of *open set* recognition, Melis et al. (2017) proposed a method called Neural Reject (NR), which attaches a support vector machine with an RBF kernel (SVM RBF) on the last (i.e., $m-1$) hidden layer of a DNN to perform rejection of samples showing an outlying behavior. In particular, the choice of the RBF kernel implies that the prediction scores provided by the SVM are proportional to the distance of the input sample to the reference prototypes (i.e., the support vectors), thus enabling the rejection of samples that fall far away from the training data in the given representation space. This single-layered defense can be expressed in our framework, instantiating $g_{m-1}$ as an SVM RBF.

### 5.1.2 Kernel Density Estimation

Feinman et al. (2017) proposed an adversarial examples detector exploiting a Kernel Density Estimator (KDE) on the embeddings obtained from the last hidden layer of the neural network to identify low-confidence input regions. As for NR, such defense can be obtained by instantiating $g_{m-1}$ as a KDE.

### 5.1.3 DNN Binary Classifier

The idea of a layer-wise detector is further developed in Metzen et al. (2017) providing a single detector subnetwork connected to an arbitrary layer of the DNN which is intended to protect. This subnetwork is trained to perform a binary classification task to distinguish genuine data from samples containing adversarial perturbations. In our framework, $g_j$ is the detector subnetwork at a given layer $j$.

### 5.1.4 Dimensionality Reduction

Multiple-layer inspection has been performed by Crecchi et al. (2019), who proposed a detection method combining non-linear dimensionality reduction techniques, i.e., $t$-SNE (Van Der Maaten and Hinton, 2008), and density estimation to detect adversarial samples. For a given layer $j$ of the network, the classifier obtained by performing density estimation on top of the embeddings produced by $t$-SNE represents $g_j$, whereas the support vector machine combiner is a realization of $\sigma$.

### 5.1.5 Deep Neural Reject

In Chapter 4, we propose to apply NR to multiple internal layer representations to form a Deep Neural Rejection (DNR) detector, empirically demonstrating improvements upon the single-layered solution. As for NR, $g_j$ is obtained through SVM RBF classifiers at layer $j$, whereas $\sigma$ is again an SVM RBF, trained via *stacked generalization* (Wolpert, 1992).

### 5.1.6 Deep k-Nearest Neighbour

Papernot and Mcdaniel (2018) proposed a detection method named Deep k-Nearest Neighbour (DkNN), which employs a k-nearest neighbor classifier on the representations of the data learned by each layer of the DNN. When a test input is fed to DkNN, it is compared to its neighboring training points according to the distance that separates them in the representations to estimate the nonconformity, i.e., the lack of support, for a prediction in the training data. If the input sample is not conformed with the training data, it is rejected as an adversarial example. This defense can be obtained by employing kNNs for $g_j$ for layer $j$ of the DNN. Statistical hypothesis tests for combiner predictions in the realm of *conformal predictions* (Saunders et al., 1999; Vovk et al., 1999; Shafer and Vovk, 2008) can be used as $\sigma$.

### 5.1.7 Generative Models

As generative models are trained to *approximate* the data-generating distribution (which is typically unknown), they are a natural candidate for manifold-based defenses against adversarial examples. Meng and Chen (2017) proposed MagNet for defending neural network classifiers against adversarial samples leveraging generative models. MagNet works in the input space and employs one or more separate detector networks in the form of a denoising autoencoder (DAE), exploiting the reconstruction error to estimate how far a test sample is from the manifold of normal samples and to *reform* it to a natural sample lying on the data manifold, which is used for classification.
Fortified Networks (Lamb et al., 2018) exploit this very same idea but on the learned hidden representation distribution: DAEs are inserted at crucial points between

| Defense | Adv. Training | $g$ | $\sigma$ |
|---|---|---|---|
| Feinman et al. (2017) | ✗ | KDE | - |
| Melis et al. (2017) | ✗ | SVM RBF | - |
| DNR (from Chapter 4) | ✗ | SVM RBF | SVM RBF |
| Papernot and Mcdaniel (2018) | ✗ | k-NN | Statistical Test |
| Lamb et al. (2018) | ✗ | DAE | - |
| Metzen et al. (2017) | ✓ | DNN | - |
| Crecchi et al. (2019) | ✓ | $t$-SNE + KDE | SVM |
| Meng and Chen (2017) | ✓ | DAE | - |

Table 5.1: Detector-based defenses against adversarial examples framed in our proposed detector framework (− for unnecessary components).

layers of the original DNN to *clean up* the adversarial sample lying away from the original data manifold, arguing that this provides stronger protection against adversarial examples than acting in the input space.
Magnet and Fortified Network defenses can be obtained in our framework by instantiating $g_j$ as a DAE for layer $j$. By having $\sigma$ as the identity function, threshold-based detection ($\omega$) on input sample reconstruction error can be used to identify outliers with respect to the expected input distribution.
However, despite the promising theoretical background, all these methods are still vulnerable (Carlini and Wagner, 2017a; Athalye et al., 2018a).

## 5.2 Fast Adversarial Example Rejection

In this section, we present our proposal for speeding up existing detection methods for adversarial examples by controlling the number of reference prototypes they make use of. Previous instance-based detectors, in fact, do not allow one to specify the number of prototypes (e.g., *support vectors* for SVM-based ones) used for identifying adversarial examples. Selecting a large number of reference prototypes, possibly at different representation layers, dramatically increases classification time, as the input sample has to be compared with each prototype at each selected representation layer to compute the corresponding prediction. Thus, controlling the number of prototypes employed by detectors is crucial for runtime efficiency. With FADER, we propose to replace existing classifiers in such detectors with size-constrained RBF networks designed for an optimal accuracy vs. speed trade-off.

RBF networks are *shallow* artificial neural networks that use radial basis functions (RBF) as activation functions. The output of the network is a linear combination of radial basis functions of the inputs and neuron parameters. Despite their architectural simplicity, they have been shown to possess structural resistance to adversarial attacks (Goodfellow et al., 2015; De Alfaro, 2018; Habib Zadeh et al., 2019; Chenou et al., 2019) thanks to their localized nature, thus they are a natural candidate for

building fast and secure detectors. The use of RBF activation functions enforces the classifier to assume a desirable *compact abating probability* property for open set recognition (Bendale and Boult, 2016; Scheirer et al., 2011). Being $s_1, \ldots, s_c$ the output scores produced by the classifier for an input sample $\boldsymbol{x}$, such property ensures that, for each given class, such scores decrease while $\boldsymbol{x}$ moves away from input regions densely populated by training samples of that class. This property allows us to easily implement a *distance-based rejection* mechanism required to detect adversarial examples, as also suggested in Melis et al. (2017) and Bendale and Boult (2016).

### 5.2.1 FADER

The central idea of FADER is to speed up instance-based adversarial example detectors by controlling the number of prototypes used for comparison while maintaining comparable performances with original solutions. To this end, we replace detector classifiers with RBF network-based ones, allowing for the joint optimization of prototypes and network parameters. Suppose to take an RBF SVM as a reference layer detector (e.g., used in NR or DNR) that we intend to speed up. Then, given $\boldsymbol{z}_j$ as the input representation for an input sample $\boldsymbol{x}$ obtained at the $j$-th layer of the DNN, the layer-detector function can be formulated as follows:

$$g_j(\boldsymbol{z}_j) = \text{sign}\left(\sum_{i=1}^{n_{\text{sv}}} \alpha_{j,i} \exp\left(-\gamma_j \left\|\boldsymbol{z}_j - \boldsymbol{z}_{j,i}\right\|^2\right) + b_j\right), \tag{5.1}$$

where $\boldsymbol{z}_{j,i}$ is the representation of the training sample $\boldsymbol{x}_i$ at layer $j$, $\alpha_{j,i}$ is its corresponding (signed) coefficient, $b_j$ is the bias, and $\gamma_j$ is the RBF kernel parameter. The coefficients $\alpha_{j,i}$ and $b$ are learned via SVM training.

One issue with kernelized SVMs is that the number of support vectors (i.e., the training samples for which $\alpha_{j,i} \neq 0$) grows linearly with the training set size (Steinwart, 2003; Chapelle, 2007; Demontis et al., 2016), and it tends to match the training set size $n$ when considering multiclass SVMs (as the union of the support vector sets learned in a disjoint manner by the different binary SVMs tends to be non-sparse). The runtime complexity of RBF SVMs is $O(q \cdot n_{\text{sv}} \cdot d)$ (Claesen et al., 2014), and it scales linearly with the number of test samples $q$, the number of support vectors $n_{\text{sv}}$, and the dimensionality $d$ of the input (at the considered layer). Accordingly, RBF SVMs tend to become too computationally demanding at runtime when trained on large training sets. In addition, they also require storing a much larger number of reference prototypes in memory, which hinders portability on low-memory embedded devices. These problems are also witnessed by the fact that a substantial amount of previous work has proposed many different SVM variants aimed at reducing or pruning the set of support vectors to speed up classification and reduce memory consumption (Claesen et al., 2014; Demontis et al., 2016).

FADER aims to replace the RBF SVM layer detectors with an RBF network to reduce the number of prototypes while maintaining the desired detector behavior, i.e., nearly the *same* decision regions of the unpruned SVMs (see Figure 5.2). The new detector decision function can be formulated as follows:

$$g_j(\boldsymbol{z}_j) = \text{sign}\left(\sum_{i=1}^{n_r} \beta_{j,i} \exp\left(-\gamma_{j,i} \left\|\boldsymbol{z}_j - \boldsymbol{r}_{j,i}\right\|^2\right) + b_j\right) . \tag{5.2}$$

Although the two definitions look alike, they substantially differ in practical terms. The reference prototypes $\boldsymbol{r}_{j,i}$ can now be optimized during RBF network training, as well as the kernel parameters $\gamma_{j,i}$ (one per prototype), to better fit the training data. This improved flexibility allows us to drastically reduce the number of reference prototypes. Moreover, jointly optimizing prototypes ($\boldsymbol{r}_{j,i}$), kernel ($\gamma_{j,i}$), and network parameters ($\beta_{j,i}, b_j$) enables a significant reduction of the number of prototypes while maintaining comparable performances with respect to over-specified solutions, as shown in our experiments (see Section 6.3). In addition, $n_r$ does not need to scale linearly with the training set size (as it is fixed a priori), thereby significantly reducing runtime complexity and memory consumption.

In terms of the proposed adversarial detector framework in Section 5.1, FADER can be represented as follows: the layer detector function $g$ can be instantiated as an RBF network. In the case of multilayered detectors, the combiner $\sigma$ can be instantiated again as an RBF network. Adversarial examples are rejected if the prediction values of the combiner do not exceed a predefined threshold, which is tuned on a validation set not to exceed a given fraction of false rejections (i.e., natural samples detected as adversarial).

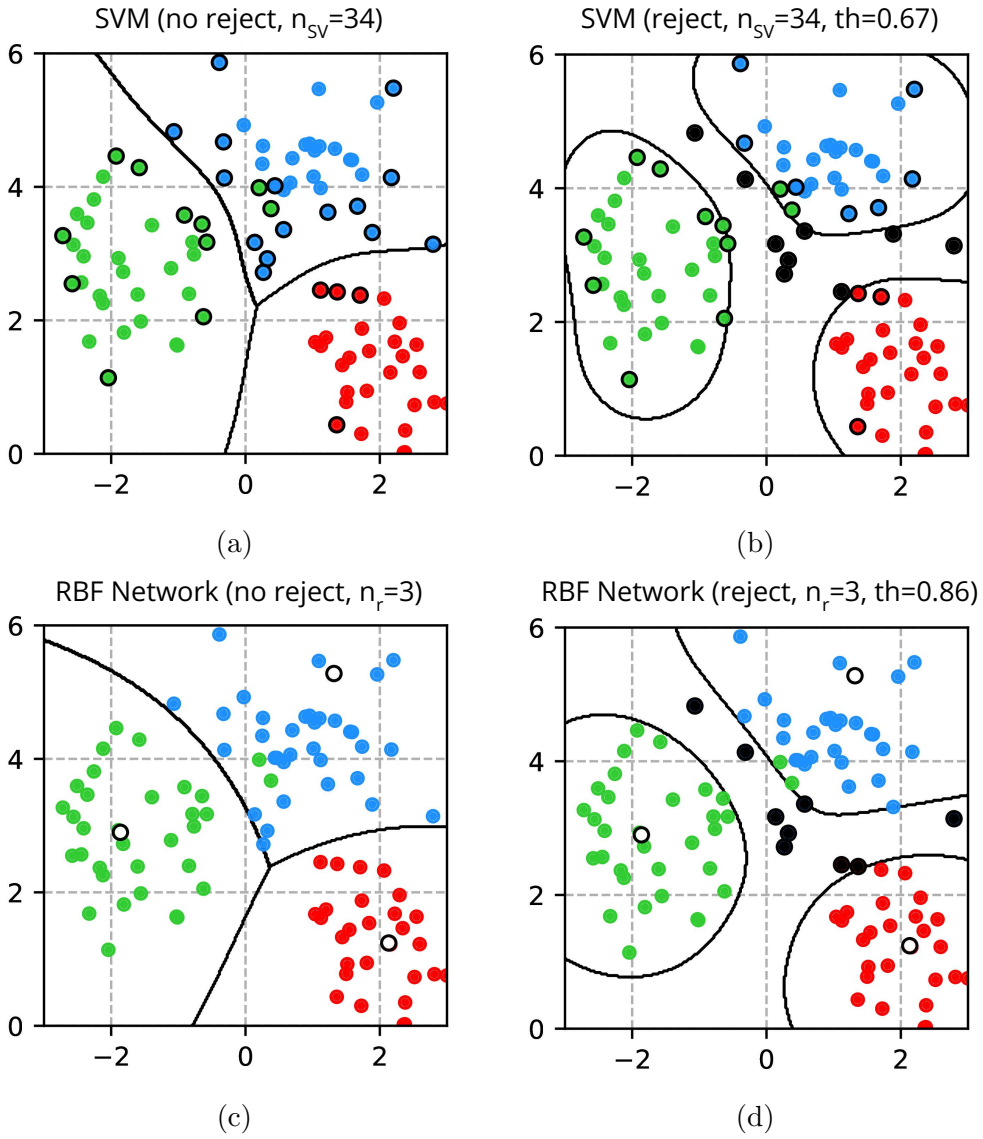


Figure 5.2: Comparison of classifiers decision regions on a two-dimensional classification example with three classes (green, blue, and red points), using multiclass SVMs with RBF kernels (SVM) and RBF Networks. a) SVM without reject option, the solution found exploits $n_{\text{sv}} = 34$ support vectors (circled in black). b) SVM with reject option using a threshold $th = 0.67$ to obtain 10% FPR, rejected samples are highlighted with black dots. c) RBF Network without rejecting option, the solution found properly separates all classes using only $n_r = 3$ bases (black circles). d) RBF Network with reject option using a threshold $th = 0.86$ to obtain 10% FPR, rejected samples are highlighted with black dots. Notably, $n_r = 3$ is the minimum number of bases to ensure each class is correctly enclosed.

## 5.3 Related Work

The problem of countering adversarial attacks is far from being new. The first adversary-aware classification algorithm against evasion attacks was proposed in 2004, which is based on simulating attacks and iteratively retraining the classifier on them (Dalvi et al., 2004). More recently, similar techniques took the name of *adversarial training* and were employed to counter adversarial examples in DNNs (Szegedy et al., 2014; Goodfellow et al., 2015) or to harden decision trees and random forests (Kantchelian et al., 2016). The scalability of these methods to large datasets and high-dimensional feature spaces is in doubt, as it may be too computationally costly to generate a sufficient number of attack samples to correctly represent their distribution.

Other variants of this approach, instead of encouraging the correct labeling of the attacks by augmenting the training set, introduce a new class in the model solely for the adversarial attacks and train the model to detect them. Bhagoji et al. (2017) propose a similar defense based on dimensionality reduction instead, which retrains the classifier on a D-dimensional version of the inputs, with $D \ll d$. This defense restricts the attacker to manipulate only the first D components, resulting in a dramatic magnitude increase of the perturbation required to produce an effective attack.

Other defenses try to detect adversarial examples by comparing the distribution of legitimate samples to the distribution of attacks. In Grosse et al. (2017), a classical statistical method is exploited, Maximum Mean Discrepancy (MMD), which allows determining if two sets of samples are drawn from the same underlying distribution. Hendrycks and Gimpel (2017b) propose the use of Principal Component Analysis (PCA) by arguing that adversarial samples place a higher (lower) weight on the later (earlier) principal components with respect to legitimate samples. Li and Li (2017), instead, apply PCA to the values after the inner convolutional layers of a CNN model and use a *cascade classifier* to detect the differences between the two distributions. Similarly, in Feinman et al. (2017), a Gaussian Mixture Model is used to analyze if the outputs of the model in the case of adversarial examples belong to a distribution different than that of legitimate samples.

Finally, recent efforts combine the advantages of adversarial training and detection mechanisms (Yin et al., 2020), reporting promising results. However, the problem remains still challenging and largely unsolved, especially when it comes to evaluating adversarial robustness against adaptive white-box attacks (Tramer et al., 2020).

# Chapter 6

# Experiments

In this chapter, we report the experimental evaluations performed on the defense approaches we proposed in the previous chapters. For all of them, we describe the experimental setting and discuss the results and findings. In particular, Section 6.1, Section 6.2, and Section 6.3 collect experiments related to Chapter 3, Chapter 4, and Chapter 5, respectively. All these experiments are performed using `secml`[1] (Pintor et al., 2022c), i.e., a Python framework that enables benchmarking of attacks and defenses for secure machine learning, on a workstation equipped with an Intel(R) Xeon(R) CPU E5-2670 v3 @ 2.30GHz with 48 cores, 126 GB of RAM and an NVIDIA Quadro M6000 graphics card with 24 GB of memory.

## 6.1 Increasing Robustness with Domain Knowledge

In this section, we report our experimental analysis for the defense described in Chapter 3, discussing the experimental setup in Section 6.1.1, and the results of standard and adversarial evaluations for multi-label classifiers in Section 6.1.2 and 6.1.3. We then show in Section 6.1.4 and 6.1.5 how our multi-label classifiers can also be adopted to mitigate the impact of adversarial examples in single-label classification tasks when auxiliary classes are exploited. This allows us to highlight that our approach exhibits competitive performances with respect to other baseline defense methods designed under the same assumptions (i.e., without assuming any specific knowledge of the attacks) and against state-of-the-art attacks that are developed for single-label classification tasks.

[1] https://github.com/pralab/secml

Table 6.1: Datasets and details on the experimental setting. "Classes" reports the total number of categories, specifying the number of main classes in parentheses. The fraction of labeled ($\%L$) samples, the level of partial labeling ($\%P$), along with the number of training ($|\mathcal{L}|$), validation ($|\mathcal{V}|$), and test ($|\mathcal{T}|$) examples are also reported.

| *Dataset* | *Classes* | $\%L$ | $\%P$ | $\lvert\mathcal{L}\rvert$ | $\lvert\mathcal{V}\rvert$ | $\lvert\mathcal{T}\rvert$ |
|---|---|---|---|---|---|---|
| ANIMALS | 33 (7) | 30% | 90% | 5808 | 1244 | 1243 |
| CIFAR-100 | 120 (100) | 30% | 0% | 40000 | 10000 | 10000 |
| PASCAL-Part | 64 (20) | 30% | 70% | 7072 | 1515 | 1515 |

Table 6.2: Values of the hyperparameter $\lambda$ selected via cross-validation in our experiments. Note that baseline models TL and FT do not exploit domain knowledge ($\lambda = 0$).

| Model | ANIMALS | CIFAR-100 | PASCAL-Part |
|---|---|---|---|
| TL+C | $10^{-2}$ | 3 | $10^{-1}$ |
| TL+CC | 1 | 10 | 1 |
| FT+C | $10^{-2}$ | 3 | $10^{-1}$ |

### 6.1.1 Experimental Settings

**Datasets.** We considered three image classification datasets, referred to as ANIMALS, CIFAR-100, and PASCAL-Part, respectively. The first one is a collection of 8287 images of animals, taken from the ImageNet database,[2] the second one is a popular benchmark composed of RGB images ($32 \times 32$) belonging to different types of classes (vehicles, flowers, people, etc.),[3] while the last dataset is composed of images in which both objects (Man, Dog, Car, Train, etc.) and object-parts (Head, Paw, Beak, etc.) are labeled.[4]
All datasets are used in a multi-label classification setting so that the ground truth of each example is composed of a set of binary class labels. In the case of ANIMALS, there are 33 categories, where the first 7 ones, also referred to as "main" classes, are about specific categories of animals (albatross, cheetah, tiger, giraffe, zebra, ostrich, penguin) while the other 25 classes are about more generic features (mammal, bird, carnivore, fly, etc.). The CIFAR-100 dataset is composed of 120 classes, out of which 100 are fine-grained ("main" classes) and 20 are superclasses. In the PASCAL-Part dataset, after having processed data as in Donadello et al. (2017), we are left with

[2]ANIMALS http://www.image-net.org/
[3]CIFAR-100 https://www.cs.toronto.edu/~kriz/cifar.html
[4]PASCAL-Part: https://www.cs.stanford.edu/~roozbeh/pascal-parts/pascal-parts.html

Table 6.3: Values of the constraint loss $\varphi$ on the test data $\mathcal{T}$.

| Model | ANIMALS | CIFAR-100 | PASCAL-Part |
|---|---|---|---|
| TL | 0.5833 $\pm$0.0316 | 1.4440 $\pm$0.0087 | 2.7286 $\pm$0.0853 |
| TL+C | 0.2134 $\pm$0.0160 | 1.0406 $\pm$0.0020 | 1.7422 $\pm$0.0516 |
| TL+CC | 0.2004 $\pm$0.0097 | 0.7267 $\pm$0.0020 | 0.7387 $\pm$0.0151 |
| FT | 0.3751 $\pm$0.0169 | 0.9603 $\pm$0.0041 | 2.4478 $\pm$0.0723 |
| FT+C | 0.0897 $\pm$0.0113 | 0.4449 $\pm$0.0068 | 0.8434 $\pm$0.0471 |

Table 6.4: Multi-label classification results in $\mathcal{T}$, for different models, averaged across different repetitions (standard deviations are $< 1\%$). The second-row block is restricted to the main classes (Accuracy or F1). See the main text for details.

| Metric | Dataset | TL | TL+C | TL+CC | FT | FT+C |
|---|---|---|---|---|---|---|
| F1 (%) | ANIMALS | 98.3 | 98.6 | 98.1 | 98.6 | 99.2 |
| | CIFAR-100 | 52.0 | 55.1 | 53.1 | 59.3 | 64.0 |
| | PASCAL-Part | 69.5 | 70.0 | 69.4 | 69.1 | 71.0 |
| AccMain (%)[1] F1Main (%)[2] | ANIMALS[1] | 98.8 | 99.2 | 99.2 | 98.5 | 99.1 |
| | CIFAR-100[1] | 53.3 | 55.6 | 52.8 | 60.5 | 61.6 |
| | PASCAL-Part[2] | 73.8 | 75.9 | 69.5 | 70.4 | 75.0 |

64 categories, out of which 20 are objects ("main" classes) and the remaining 44 are object-parts.

The use of domain knowledge holds for all the available examples. In the case of ANIMALS, it is a collection of FOL formulas that were defined in the benchmark of P.H. Winston (Winston and Horn, 1986), and they involve relationships between animal classes and animal properties, such as $\forall x$ FLY$(x) \wedge$ LAYEGGS$(x)$ $\Rightarrow$ BIRD$(x)$. In CIFAR-100, FOL formulas are about the father-son relationships between classes, while in PASCAL-Part they either list all the parts belonging to a certain object, i.e., MOTORBIKE$(x) \Rightarrow$ WHEEL$(x) \vee$ HEADLIGHT$(x) \vee$ HANDLEBAR$(x) \vee$ SADDLE$(x)$, or they list all the objects in which a part can be found, i.e., HANDLEBAR$(x) \Rightarrow$ BICYCLE$(x) \vee$ MOTORBIKE$(x)$. We also introduced a disjunction or a mutual-exclusivity constraint among the main classes and another disjunction among the other classes. See Table 6.1 and Appendix B for more details.

Each dataset was divided into training and test sets (the latter indicated with $\mathcal{T}$). The training set was divided into a learning set ($\mathcal{L}$), used to train the classifiers,

and a validation set ($\mathcal{V}$), used to tune the model parameters. We defined a semi-supervised learning scenario in which only a portion of the training set is labeled, sometimes partially (i.e., only a fraction of the binary labels of an example is known), as detailed in Table 6.1. We indicated with $\%L$ the percentage of labeled training data and with $\%P$ the percentage of binary class labels that are unknown for each labeled example.[5]

**Classifiers.** We compared two neural architectures based on the popular backbone ResNet50, trained using ImageNet data. In the first network, referred to as TL, we transferred the ResNet50 model and trained the last layer from scratch in order to predict the dataset-specific multiple classes (sigmoid activation). The second network, indicated with FT, has the same structure as TL, but we also fine-tuned the last convolutional layer. Each model is based on the product T-Norm, and it was trained for a number of epochs $e$ that we selected as follows: 1000 epochs in ANIMALS, 300 (TL) or 100 (FT) epochs in CIFAR-100, and 500 (TL) or 250 (FT) in PASCAL-Part, using mini-batches of size 64. We used the Adam optimizer, with an initial step size of $10^{-5}$, except for FT in CIFAR-100, for which we used $10^{-4}$ to speed up convergence. We selected the model at the epoch that led to the largest F1 in $\mathcal{V}$. We considered unconstrained ($\lambda = 0$) and knowledge-constrained ($\lambda > 0$) models. The latter are indicated with the +C (and +CC) suffix.

**Evaluation Metrics.** To evaluate performance, we considered the (macro) *F1* score and a metric restricted to the main classes.[6] For ANIMALS and CIFAR-100, the main classes are mutually exclusive, so we measured the accuracy in predicting the winning main class (*AccMain*), while in PASCAL-Part, we kept the F1 score (*F1Main*) as multiple main classes can be predicted on the same input.

**Hyperparameter Tuning.** In Table 6.2, we report the optimal value of $\lambda \in \{10^{-2}, 10^{-1}, 1, 3, 5, 8, 10, 10^2\}$ for the TL+C and FT+C models used in our experiments, selected via a 3-fold cross-validation procedure. In the case of TL, we also considered a strongly-constrained (+CC) model with inferior performance but higher coherence (greater $\lambda$) among the predicted categories (that might lead to a worse fitting of the supervisions).[7] Table 6.3 reports the value of the constraint loss $\varphi$ measured on the test set $\mathcal{T}$. We used $\mu^{\mathcal{L}} = \mu^{\mathcal{T}}$, setting each component $\mu_h$ to 1, with the exception of the weight of the mutual exclusivity constraint or the disjunction of the main classes, which was set to 10 to enforce the classifier to take decisions.

**Attack Optimization.** Our attack optimizes Equation (3.11) via projected gradient descent. Black-box attacks are non-adaptive and thus ignore the defense mecha-

[5] When splitting the training data into $\mathcal{L}$ and $\mathcal{V}$, we kept the same percentages of unknown binary class labels per example ($\%P$) in both the splits. Of course, in $\mathcal{V}$, there are no fully-unlabeled examples ($\%L$ is 100). Moreover, when generating partial labels, we ensured that the percentages of discarded positive (i.e., 1) and negative (i.e., 0) class labels were the same.

[6] We compared the outputs against 0.5 to obtain binary labels.

[7] FT+C has more learnable weights: constraint loss is already small.

nism. For this reason, the constraint loss term $\varphi$ in our attack is ignored by setting its multiplier $\rho = 0$ and $\kappa = \infty$. For white-box attacks on ANIMALS and PASCAL-PART, we set $\rho = 0.1$ and $\rho = 1$, respectively, while setting $\kappa = 2$. These values are chosen to appropriately scale the values of the constraint loss term $\varphi$ w.r.t. the logit difference (i.e., the first term in Equation 3.11, lower bounded by $-2\kappa$). This is required to have the sample misclassified while also fulfilling the domain-knowledge constraints.

The process is better illustrated in Figure 6.1 and 6.2, in which we respectively report the behavior of the black-box and white-box attack optimization on a single image from the ANIMALS dataset, with $\varepsilon = 1$. In particular, in each Figure, we report the source image, the (magnified) adversarial perturbation, and the resulting adversarial examples, along with some plots describing the optimization process, i.e., how the attack loss of Equation (3.11) is minimized across iterations, and how the softmax-scaled outputs on the main classes and the logarithm of the constraint loss $\varphi$ change accordingly.
In both the black-box and white-box cases, the attack loss is progressively reduced during the iterations of the optimization procedure. While the *albatross* prediction is progressively transformed into *an ostrich*, the constraint loss increases across iterations, exceeding the rejection threshold. Thus, the adversarial example is correctly detected. Similarly, the white-box attack is able to initially flip the prediction from *albatross* to *ostrich*, allowing the constraint loss to increase. However, after this initial phase, the attack correctly reduces the constraint loss after its initial bump, bringing its value below the rejection threshold. The system thus fails to detect the corresponding adversarial example. Finally, it is also worth remarking that, in both cases, the final perturbations do not substantially compromise the source image content, remaining essentially imperceptible to the human eye.

### 6.1.2 Experimental Results on Multi-label Classifiers

We discuss here the main experiments related to the evaluation of the considered multi-label classifiers.

**Standard Evaluation.** In order to assess the behaviors of the classifiers in the considered datasets and the available domain knowledge, we compared classifiers that exploit domain knowledge with the ones that do not exploit it. The results of our evaluation are reported in Table 6.4, averaged over the 3 training-test splits. For each of them, 3 runs were considered, using different initialization of the weights. The introduction of domain knowledge allows the constrained classifiers to slightly outperform the unconstrained ones.

**Adversarial Evaluation.** To evaluate adversarial robustness, we used the MKA attack procedure described in Section 3.2. and we restricted the attack to work on the already introduced main classes, being them associated with the most important categories of each problem. In ANIMALS and CIFAR-100, we assumed the attacker

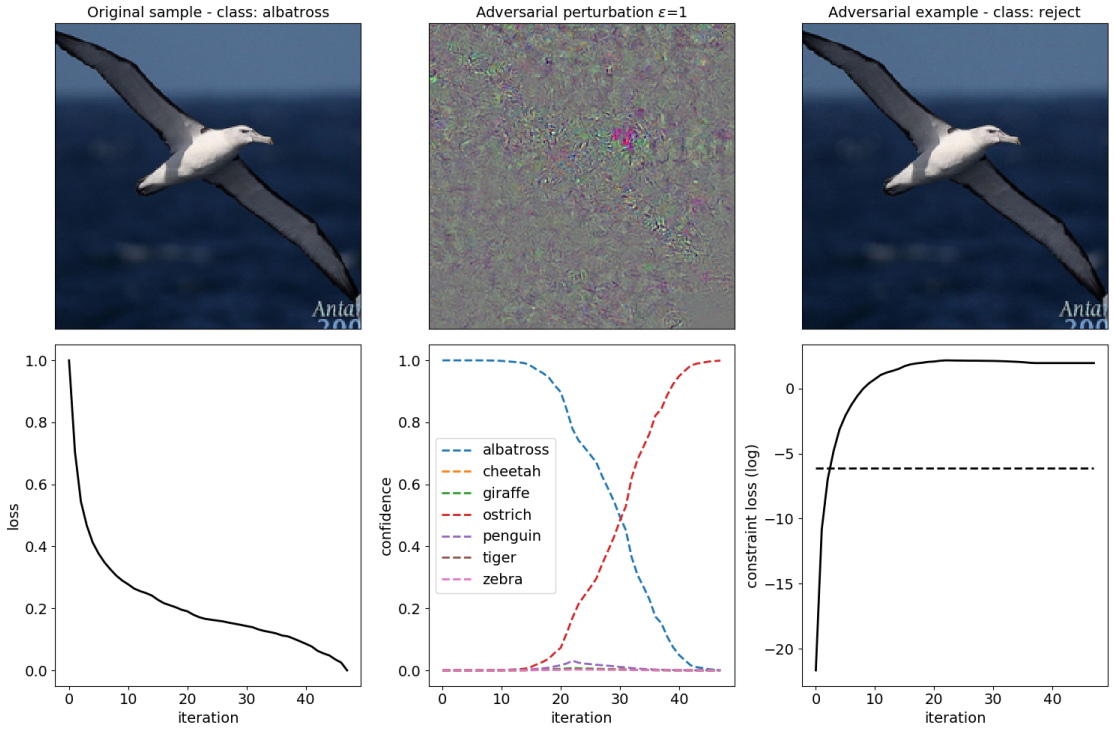


Figure 6.1: Black-box attack on the ANIMALS dataset. While the attack is able to flip the initial prediction from *albatross* to *ostrich*, the attack is eventually detected as the constraint loss remains above the rejection threshold (dashed black line).

to have access to the information on the mutual exclusivity of the main classes so that $p$ in Equation (3.11) is not required to change during the attack optimization. We also set $\kappa = \infty$ to maximize confidence of misclassifications at each given perturbation bound $\varepsilon$. All the following results are averaged after having attacked twice the model obtained after each of the 3 training runs.

In the *black-box* setting, we assumed the attacker to be also aware of the network architecture of the target classifier, and attacks were generated from a surrogate model trained on a different realization of the training set. Figure 6.3 shows the classification quality as a function of the data perturbation bound $\varepsilon$, comparing models trained with and without constraints against those implementing the detection/rejection mechanism described in Equation (3.3). When using such a mechanism, the rejected examples are marked as correctly classified if they are adversarial ($\varepsilon > 0$). Otherwise ($\varepsilon = 0$), they are marked as points belonging to an unknown class, slightly worsening the performance.

The +C/+CC models show larger accuracy/F1 than the unconstrained ones. Despite the lower results at $\varepsilon = 0$, models that are more strongly constrained (+CC) resulted in being harder to attack for increasing values of $\varepsilon$. When the knowledge-based detector is activated, the improvements with respect to models without rejec-

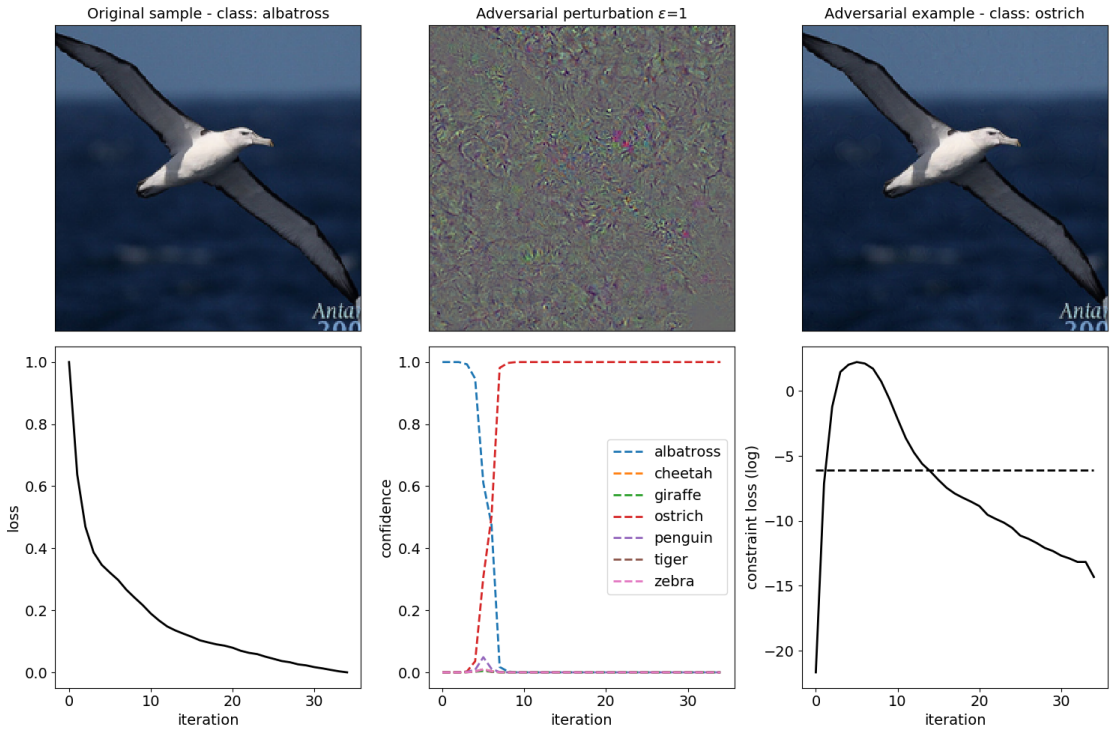


Figure 6.2: White-box attack on the ANIMALS dataset. The attack is able to flip the initial prediction from *albatross* to *ostrich*, and then starts reducing the constraint loss, which eventually falls below the rejection threshold (dashed black line). The attack sample remains thus undetected.

tion are significantly evident. No model is specifically designed to face adversarial attacks, and, of course, there are no attempts to reach state-of-the-art results.[8] However, the positive impact of exploiting domain knowledge can be observed in all the considered models and datasets and for almost all the values of $\varepsilon$, confirming that such knowledge is not only useful to improve classifier robustness but also as a means to detect adversarial examples at no additional training cost.

In general, FT models yield better results due to the larger number of optimized parameters. In ANIMALS, the rejection dynamics are providing large improvements in both TL and FT, while the impact of domain knowledge is mostly evident in the robustness of FT. In CIFAR-100, domain knowledge only consists of basic hierarchical relations, with no intersections among child classes or among father classes. By inspecting the classifier, we found that it is pretty frequent for the fooling examples to be predicted with a strongly-activated father class and a (coherent) child class, i.e., we have strongly-paired classes, accordingly to Def. 3.2.1. Differently,

[8]Recall that our rejection mechanism is completely agnostic to the attack; it neither assumes any knowledge of the attack algorithm nor is retrained on adversarial examples. Nevertheless, it can be used as a complementary defense mechanism.

the domain knowledge in the other datasets is more structured, yielding better detection quality on average, remarking the importance of the level of detail of such knowledge to counter adversarial examples. In the case of PASCAL-Part, the detection mechanism turned out to behave better in unconstrained classifiers, even if it has a positive impact also on the constrained ones. This is due to the intrinsic difficulty of making predictions on this dataset, especially when considering small object parts. The false positives have a negative effect in the training stage of the knowledge-constrained classifiers.

To provide a comprehensive, worst-case evaluation of the adversarial robustness of our approach, we also considered a *white-box* adaptive attacker that knows everything about the target model and exploits knowledge of the defense mechanism to bypass it. Of course, this attack always evades detection if the perturbation size $\varepsilon$ is sufficiently large. We evaluated multiple values of $\rho$ of Equation (3.11), selecting the one that yielded the lowest values of such objective function. In Figure 6.4, we report the outcome of this analysis for FT models, showing that, even if the accuracy drop is obviously evident for all datasets, in ANIMALS, the constrained classifiers require larger perturbations than the unconstrained ones to reduce the performance of the same quantity. Similar behavior is shown in CIFAR-100, even though only with small $\varepsilon$ values. Accordingly, fooling the proposed detection mechanism is not always as trivial as one might expect, even in this worst-case setting. The impact of the rejection mechanisms is significantly reduced, as expected, but still having a positive impact. Finally, let us point out that the performance drop caused by the white-box attack is much larger than that observed in the black-box case. However, since domain knowledge is not likely to be available to the attacker in many practical settings, it remains an open challenge to develop stronger, practical black-box attacks that are able to infer and exploit such knowledge to bypass our defense mechanism.

### 6.1.3 In-depth Analysis on Multi-label Classifiers

**Incomplete Domain Knowledge.** We investigated in more detail the relative impact of the domain information on a target problem, simulating the availability of differently sized knowledge bases, $\mathcal{K}_1$, $\mathcal{K}_2$, $\mathcal{K}_3$, $\mathcal{K}_4$, where each $\mathcal{K}_j \subseteq \mathcal{K}$. In particular, we considered the ANIMALS dataset, and we generated $\mathcal{K}_1$, $\mathcal{K}_2$, $\mathcal{K}_3$ by removing some of the FOL formulas of the original $\mathcal{K}$ that was used in the previous experiments (i.e., the one of Table B.1 in Appendix B), while $\mathcal{K}_4 = \mathcal{K}$. This means that some information that belongs to $\mathcal{K}_4$ is actually missing in the other knowledge sets. In detail, we created $\mathcal{K}_1$ by removing the rules that either include the 'mammal' or the 'bird' categories, while $\mathcal{K}_2$ is the outcome of discarding from $\mathcal{K}$ the rules including the 'mammal' category. Similarly, $\mathcal{K}_3$ is obtained by removing the rules of the 'bird' category. We repeated the experiments, using only one of the generated knowledge bases at a time and focusing on the same models

of Figure 6.3 (bottom) and Figure 6.4, that were retrained from scratch. Figure 6.5 (a,c) shows the classification quality we obtained in the black (a) and white box (c) settings, using the MKA attack and $\varepsilon = 0.5$ (almost the mid of the plots in Figures 6.3-6.4). In the black-box case, focusing on the models that include rejection (Rej), it is evident how larger knowledge bases yield better results. Interestingly, comparing the outcome of such models with the ones without rejection, we can see that our defense makes the classifier more robust to attacks even when using the smallest amount of knowledge ($\mathcal{K}_1$), confirming the versatility of what we propose. In the white-box setting, there are still changes in the accuracy when varying $\mathcal{K}_j$, but they are not so evident and they lack a clear trend. This was expected since, in this case, the attack procedure is aware of the domain knowledge. However, this result confirms the capability of MKA to craft adversarial examples that lead to knowledge-coherent predictions (to a certain extent) even when varying the level of detail of the knowledge sets.

**Rejection Threshold.** In the same experimental setting, we also explored the sensitivity of the system to the rejection threshold $\theta$, using the whole knowledge set $\mathcal{K}$. We compared different $\theta$'s that are smaller or greater than the one we selected using validation data (Section 3.2), indicated here with $\theta^\star$. In particular, we evaluated $\theta \in \{10^\kappa \theta^\star,\ \kappa \in [-5, 5], \text{integer}\}$, and we measured both the classification quality on the perturbed data, as we did so far, and the rejection rate on the clean test data, where no samples should be rejected. Figure 6.5 (b,d) reports these two measures on the y-axis and x-axis, respectively, and each marker is about a specific $\theta$, considering black (b) and white (d) box settings. The value of $\theta^\star$ has been highlighted with a red circle, and only the significant portions of the plots are shown. Too small thresholds (rightmost areas of each plot) lead to systems that frequently reject also clean examples, while too large $\theta$'s (leftmost areas) do not improve the quality of the classifiers, which are fooled by some of the data generated in an adversarial manner within the $\varepsilon$-bound. Overall, the $\theta^\star$ we selected represents a pretty appropriate trade-off between the two measures.

**Noisy Domain Knowledge.** We further extended our analysis by considering the case in which the available domain knowledge is noisy, thus including a small percentage of information that is incorrect in the ANIMALS domain. We simulated three scenarios by altering the original knowledge base $\mathcal{K}$ using three different criteria, yielding the noisy knowledge bases $\tilde{\mathcal{K}}_a$, $\tilde{\mathcal{K}}_b$, $\tilde{\mathcal{K}}_c$, respectively. The chosen criteria either modify four of the existing rules, making them not coherent with the clean knowledge, or they add four new rules that are not correct in the considered domain, as shown in detail in Table B.5, Table B.6, and Table B.7 of Appendix B, and for which we provide a brief description in the following. In the case of $\tilde{\mathcal{K}}_a$, we selected four existing implications of $\mathcal{K}$, and we altered the premises of two of them and the conclusions of the other two ones. We ensured to inject noise in the main and auxiliary classes in a balanced manner. The same balancing is also kept when generating $\tilde{\mathcal{K}}_b$, that, however, was obtained by adding four new rules to $\mathcal{K}$. Finally, $\tilde{\mathcal{K}}_c$ is the

result of augmenting each main-class-oriented conclusion of four existing implications with a disjunction involving a different, randomly selected, main class. For example, the clean rule BLACKSTRIPES($x$)$\wedge$UNGULATE($x$)$\wedge\ldots\Rightarrow$ZEBRA($x$) is altered by replacing the conclusion ZEBRA($x$) with ZEBRA($x$)$\vee$TIGER($x$). These rules are fulfilled both for configurations that make true their original/clean counterparts and for other configurations that are actually wrong in the ANIMALS domain. Focusing on the same experimental setup we defined when testing differently sized knowledge bases, we investigated the effects of using each noisy knowledge base both to learn the classifier and/or as a rejection criterion. Figure 6.6 shows the results of our experience. As expected, learning with a noisy knowledge base reduces the accuracy of the classifier since the network learns from FOL rules that are partially in contrast with some of the available labeled examples. It is the case of FT+C trained on any noisy $\tilde{\mathcal{K}}$. when compared with the case of the clean $\mathcal{K}$ (rightmost set of bars). However, when adding the rejection criterion, we still observe a significant improvement in the performance, even if slightly smaller than in the case of $\mathcal{K}$. Rejection is the outcome of evaluating the average violations of all the available rules so that the effect of the noisy portion of the knowledge base is partially compensated by the other non-noisy rules. Of course, the final outcome depends on the type of noise we injected into the knowledge base. In the considered experience, adding wrong rules ($\tilde{\mathcal{K}}_b$) led to lower accuracies than when perturbing some of the existing rules ($\tilde{\mathcal{K}}_a$). The case of $\tilde{\mathcal{K}}_c$ is the one that most evidently impacted the classifier performance, with or without rejection. On one hand, we simply introduced more error-tolerant conclusions; on the other hand, we did it by altering FOL rules whose conclusions are all about main classes, significantly compromising the way they are related. Interestingly, for all the noisy knowledge bases, adding the rejection module (Rej) to FT turned out to be better than adding it to FT+C, suggesting that a rejection criterion based on noisy knowledge could be more effective in classifiers that have not been already exposed to such noisy information during the learning stage. As expected, results collected in the white-box case show that whenever the attacker has full access to the knowledge, being it noisy or not, he can craft MKA attacks with similar outcomes in terms of performance drops. Overall, this experience confirms that what we propose can indeed be applied also in the case of partially noisy domain knowledge, still increasing the robustness of the classifier, even if to a smaller extent than in the case of clean knowledge.

### 6.1.4 Experimental Results on Single-label Classifiers

The focus of this section is on multi-class classification paired with domain knowledge. However, as anticipated in Section 3.2 and qualitatively shown in Figure 3.3, we can consider a special setting in which a single-label classifier internally includes predictors over auxiliary classes that are involved in the knowledge constraints. We experimentally evaluate this setting in the context of the ANIMALS and CIFAR-100 datasets, where the respective main classes (described in Section 6.1.1) are mutu-

ally exclusive (which is not the case of PASCAL-Part), thus well suited to simulate the setting of Figure 3.3. We compared the proposed rejection mechanisms to a concurrent defense mechanism developed under the same assumptions (i.e., without assuming any knowledge of the attack algorithm), known as Neural Rejection (NR) (Melis et al., 2017), and against the state-of-the-art attacks included in the AutoAttack framework (Croce and Hein, 2020a), developed for single-label classification tasks.

**Compared Defense and Attack Strategies.** The NR defense mechanism, proposed in Melis et al. (2017), aims to reject inputs that are far from the training data in a given representation space. The rationale is that points with low support from the training set cannot be reliably classified and should be thus rejected. To this end, the output layer of the deep network is replaced with a support vector machine trained using the RBF kernel (SVM-RBF), which enforces the prediction scores to be proportional to the distance between the input sample and the reference prototypes (i.e., the support vectors) in the representation space. Samples are rejected if the prediction scores do not exceed the rejection threshold. Similarly to our approach, this defense mechanism does not make any assumptions about the attack to be detected other than assuming an anomalous behavior with respect to the observed training data.

Table 6.5: ANIMALS dataset. Vulnerability analysis of the classifiers against MKA and state-of-art attacks—classification quality is reported, the same as Figuress 6.3-6.4 (first column). For each type of classifier (TL, FT), rows are organized into three groups, that are: models without rejection, with rejection (Rej), classifier equipped with Neural Rejection (NR). For each attack (columns—see Croce and Hein (2020a) for a description of the compared attacks), the result of the most robust classifier in the group is highlighted in bold. Models exploiting the proposed rejection (Rej) that overcome NR are marked with *, and vice-versa.

| | | White-box attacks ($\varepsilon = 0.5$) | | | | | Black-box transfer attacks ($\varepsilon = 0.5$) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Model | $\varepsilon = 0$ | MKA | APGD-CE | APGD-T | FAB-T | Square | MKA | APGD-CE | APGD-T | FAB-T | Square |
| TL | 99.0 | 25.0 | 17.5 | 14.9 | 20.0 | 96.6 | 45.3 | 29.4 | 29.1 | 83.1 | 98.4 |
| TL+C | 99.3 | **25.0** | **19.0** | 15.4 | 21.5 | 98.0 | 47.7 | 29.8 | 30.6 | 87.7 | 98.9 |
| TL+CC | 99.3 | 24.5 | 18.1 | **15.5** | **22.7** | **98.2** | **48.0** | **30.2** | **32.0** | **89.5** | **99.0** |
| TL (Rej) | 91.8 | 49.8 | 43.3 | 97.0* | 100.0* | 100.0* | 85.6 | 53.7 | 98.4 | 99.9 | 99.9 |
| TL+C (Rej) | 92.3 | 56.8* | **47.8** | 97.7* | 100.0* | 100.0* | 91.2 | **56.0** | 98.6 | 99.9 | 99.9 |
| TL+CC (Rej) | 92.7 | **57.5*** | 45.8 | **98.1*** | 100.0* | 100.0* | **93.4** | 55.4 | **98.8** | **100.0*** | **100.0*** |
| TL (NR) | 99.2 | 55.5 | 58.5* | 96.8 | 99.6 | 99.8 | 98.0* | 71.7* | 99.3* | 100.0* | 100.0* |
| FT | 98.6 | 25.6 | 21.9 | 12.9 | 20.0 | 96.3 | 51.2 | 47.2 | 75.6 | 95.7 | 98.2 |
| FT+C | 99.1 | **31.7** | **51.1** | **18.0** | **29.5** | **97.7** | **76.7** | **57.5** | **88.8** | **98.3** | **98.9** |
| FT (Rej) | 92.7 | 39.3* | 36.8 | 90.0* | 99.7* | 99.7* | 88.9 | 66.9 | 99.6* | 99.7* | 99.8* |
| FT+C (Rej) | 93.2 | **60.7*** | **66.6*** | **97.3*** | **99.8*** | **99.9*** | **98.3*** | **82.2*** | **99.9*** | **99.9*** | **99.9*** |
| FT (NR) | 98.6 | 37.3 | 38.3 | 87.3 | 97.0 | 99.6 | 91.0 | 79.2 | 98.7 | 99.2 | 99.5 |

TL: Transfer Learning
FT: Fine Tuning
+C: knowledge-constrained (optimal)
+CC: strongly knowledge-constrained
(NR): model with Neural Rejection
(Rej): model with rejection

Table 6.6: CIFAR-100 dataset. Vulnerability analysis of the classifiers against MKA and state-of-art attacks—classification quality is reported, the same as Figures 6.3-6.4 (second column). Refer to the caption of Table 6.5 for more details (see Croce and Hein (2020a) for a description of the compared attacks).

| | | White-box attacks ($\varepsilon = 0.03$) | | | | | Black-box transfer attacks ($\varepsilon = 0.03$) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Model | $\varepsilon = 0$ | MKA | APGD-CE | APGD-T | FAB-T | Square | MKA | APGD-CE | APGD-T | FAB-T | Square |
| TL | 51.0 | 21.9 | 22.2 | 21.6 | 22.3 | 51.4 | 23.1 | 23.7 | 23.9 | 39.5 | 52.7 |
| TL+C | 52.9 | **27.4** | 24.6 | 24.2 | 25.1 | **53.3** | 32.3 | 35.5 | 37.9 | **48.4** | **54.5** |
| TL+CC | 50.5 | 27.1 | **25.0** | **24.7** | **25.3** | 49.5 | **35.5** | **38.6** | **40.4** | 46.9 | 51.5 |
| TL (Rej) | 48.1 | 26.9 | 33.2* | 34.3* | 34.4 | 59.2* | 27.6 | 35.1 | 36.9 | 49.1 | 60.2* |
| TL+C (Rej) | 49.4 | **31.8*** | **35.0*** | **35.6*** | **36.2** | **60.6*** | 40.7 | 44.8 | 47.0 | **56.0*** | **61.0*** |
| TL+CC (Rej) | 46.1 | 30.8* | 34.0* | 34.7* | 35.4 | 55.7* | **45.4** | **46.3** | **47.6** | 53.5* | 57.0* |
| TL (NR) | 49.0 | 30.5 | 30.1 | 24.5 | 39.6* | 45.6 | 49.0* | 48.3* | 49.0* | 51.3 | 53.3 |
| FT | 59.4 | 29.0 | 26.4 | 26.0 | 26.7 | 57.2 | 48.4 | 49.1 | 49.7 | 55.5 | 59.5 |
| FT+C | 60.0 | **31.4** | **29.6** | **28.3** | **30.6** | **60.1** | **51.6** | **52.2** | **52.8** | **57.8** | **61.0** |
| FT (Rej) | 57.4 | 31.1 | 37.5* | **42.0*** | 41.1 | 66.1* | 55.1 | 57.2* | 58.4* | 62.7* | 66.2* |
| FT+C (Rej) | 56.7 | **37.6*** | **37.8*** | 41.1* | **44.6** | **67.0*** | **60.2*** | **59.5*** | **60.3*** | **64.4*** | **67.1*** |
| FT (NR) | 59.7 | 36.5 | 35.3 | 30.4 | 50.9* | 55.1 | 58.0 | 54.2 | 55.7 | 60.0 | 62.7 |

TL: Transfer Learning
FT: Fine Tuning
+C: knowledge-constrained (optimal)
+CC: strongly knowledge-constrained
(NR): model with Neural Rejection
(Rej): model with rejection

To compare our defense with NR, we have considered four different state-of-the-art evasion attacks: APGD-CE, APGD-T, FAB-T, and Square, implemented within the framework of AutoAttack (Croce and Hein, 2020a). APGD-CE (APGD-T) is an indiscriminate (targeted) step-free variant of the famous attack called PGD (Madry et al., 2018). Unlike PGD, the step size reduction is not scheduled a priori but instead governed by the optimization function trend. Moreover, both APGD attacks use momentum. FAB-T is the targeted version of an attack called Fast Adaptive Boundary Attack (FAB) (Croce and Hein, 2020b), which tries to find the minimum distance sample beyond the boundary of the desired class. The Square attack (Andriushchenko et al., 2020), differently from the previously mentioned ones, is a black-box attack; namely, it can query the classifier to obtain the predicted scores without exploiting any knowledge of the model architecture. By default, APGD-CE makes five random restarts, whereas the targeted versions of the attacks, i.e., APGD-T and FAB-T, run the attack repeatedly sweeping among all the available target classes.

**Adversarial Evaluation.** In our experiments, we fixed the maximum allowed perturbation $\varepsilon$ to 0.5 and 0.03 on the ANIMALS and CIFAR-100 datasets, respectively – the values in the middle of the x-axis of Figures 6.3- 6.4 – and we used the default value for all the other attacks' parameters. Table 6.5 and Table 6.6 report the results of this analysis, showing the classification quality (the same measure of Figure 6.3- 6.4) on the clean (unmodified) test set $\mathcal{T}$ ($\varepsilon = 0$) and on the attacked instances of $\mathcal{T}$ generated in the same white-box and black-box scenarios described in Section 6.1.2.
As expected, white-box attacks are more effective than black-box ones, reducing the

model accuracy in a more evident manner. Results confirm that both the domain knowledge introduced at training time (+C and +CC) or exploited to implement the proposed rejection mechanism (Rej) improve the model robustness against all the considered attacks and jointly using these strategies further improves it.
On average, the performances of the unconstrained classifiers (TL and FT) paired with the proposed knowledge-based rejection are comparable with the ones paired with NR, even though they clearly behave in different manners across the datasets/attacks. Differently, when also considering constrained models (+C and +CC), in most cases, we can find a classifier with rejection (Rej) that outperforms the unconstrained classifiers equipped with NR. On the clean samples ($\varepsilon = 0$), the knowledge-based rejection criterion resulted more aggressive than NR.
MKA and APGD-CE/T are more effective than the other attacks. On average, their performances are comparable, and they depend on both the considered model and the dataset. In CIFAR-100, MKA outperforms APGD-CE/T on the black-box transfer scenario and against the model equipped with the proposed rejection mechanism (Rej), whereas on the ANIMALS dataset, APGD-CE usually obtained better results. APGD-CE/T leverages an optimization strategy that is more advanced than the one of MKA that, differently from APGD-CE/T, is designed to be used in multi-label problems too. For example, APGD-CE/T makes several attack restarts and uses a special type of adaptive step size. In the white-box setting, attacks yield a larger reduction in the performances. However, in the case of ANIMALS, the proposed rejection mechanism is still robust to all the attacks, with the exceptions of MKA, which is knowledge aware, and of APGD-CE. The fine-tuned optimization procedure in APGD-CE allows the attack to create samples that are confidently misclassified, and they end up belonging to space regions in which the classification functions are paired (Def. 3.2.1). In CIFAR-100, the rejection mechanism still has a positive impact, even if it is less significant than in ANIMALS.

### 6.1.5 In-depth Analysis on Single-label Classifiers

We further analyzed our results, visualizing the behavior of all the compared attacks in terms of the value of the constraint loss of Equation (3.3) and of the supervision loss – the first term of Equation (3.2). Figures 6.7- 6.8 show each generated adversarial example, highlighting them with different markers/colors in function of the corresponding attack procedure on the ANIMALS and CIFAR-100 datasets, respectively, black-box (i.e., the constraint loss is measured for the purpose of determining whether to reject or not an example). Samples that are rejected are indicated with crosses, while circles represent the non-rejected ones. The dotted line is about the rejection threshold $\theta$ from Equation (3.8).
In line with what we observed in the numerical results, in the case of ANIMALS, Figure 6.7, it is evident how APGD-CE is actually able to craft attacks that strongly increase the supervision loss, still fulfilling the constraints (top-left area). Differently, the other attacks are not able to reach such results, so their data is localized in high-

constraint loss regions, easily rejected by the proposed technique, especially FAB-T, while Square actually fails in generating evident attacks. It is interesting to notice the D-shaped white region over the origin. It is an area in which constraints are almost fulfilled and the loss function can reach significantly non-null values, but no attacks fall there. This suggests that it is not straightforward to increase the supervision loss without violating the constraints. However, there are more extreme APCG-CE configurations with the largest supervision losses that also fulfill the knowledge (Figure 6.7, top-left area). Of course, this depends on several factors, such as the type of domain knowledge that is available, the way we selected to convert it into polynomial constraints, and the constraint enforcement scheme, thus opening to future improvements. Moving to the CIFAR-100 dataset, Figure 6.8, we observe different patterns with respect to the case of ANIMALS. This was clearly expected since the two datasets differ both in terms of the problem they consider, the number of classes, and in terms of the known relationships among such classes, described by the dataset-specific domain knowledge and embedded into the constraint loss. However, we can still observe the D-shaped region over the origin, even if in a less significant manner. On this dataset, the rejection rates are generally lower than ANIMALS. This is mostly due to the fact the constraint loss is larger also on the unaltered data, due to the already mentioned different problem and different type of domain knowledge. As a matter of fact, we have also a larger reject threshold $\theta$. In this case, the behavior of the different attack strategies is more coherent, remarking previous considerations on the role of knowledge in shaping the distribution of the attacks.

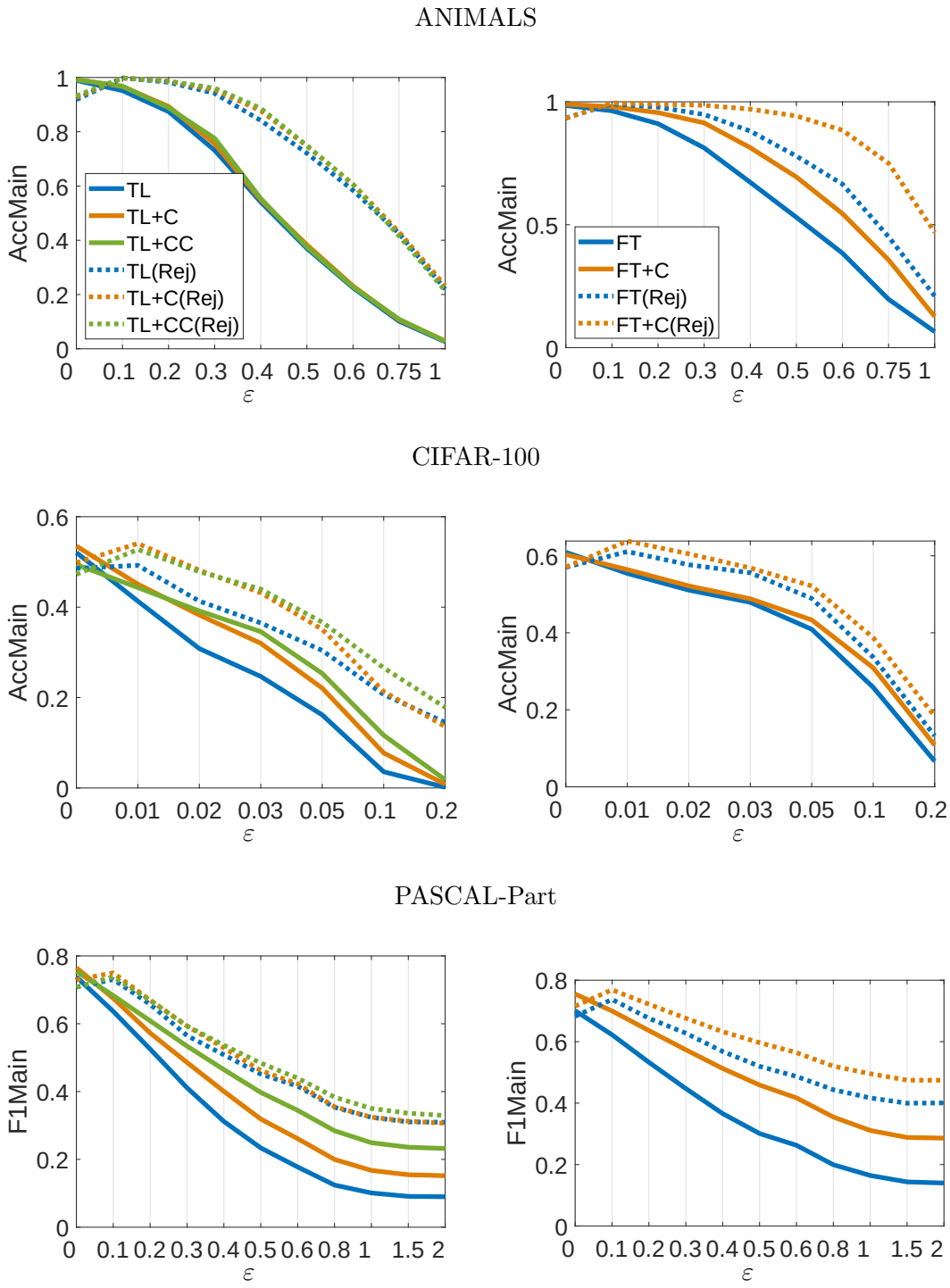


Figure 6.3: Black-box attacks. Classification quality of vanilla and knowledge-constrained models in function of $\varepsilon$. Dotted plots include rejection (Rej) of inputs that are detected to be adversarial.

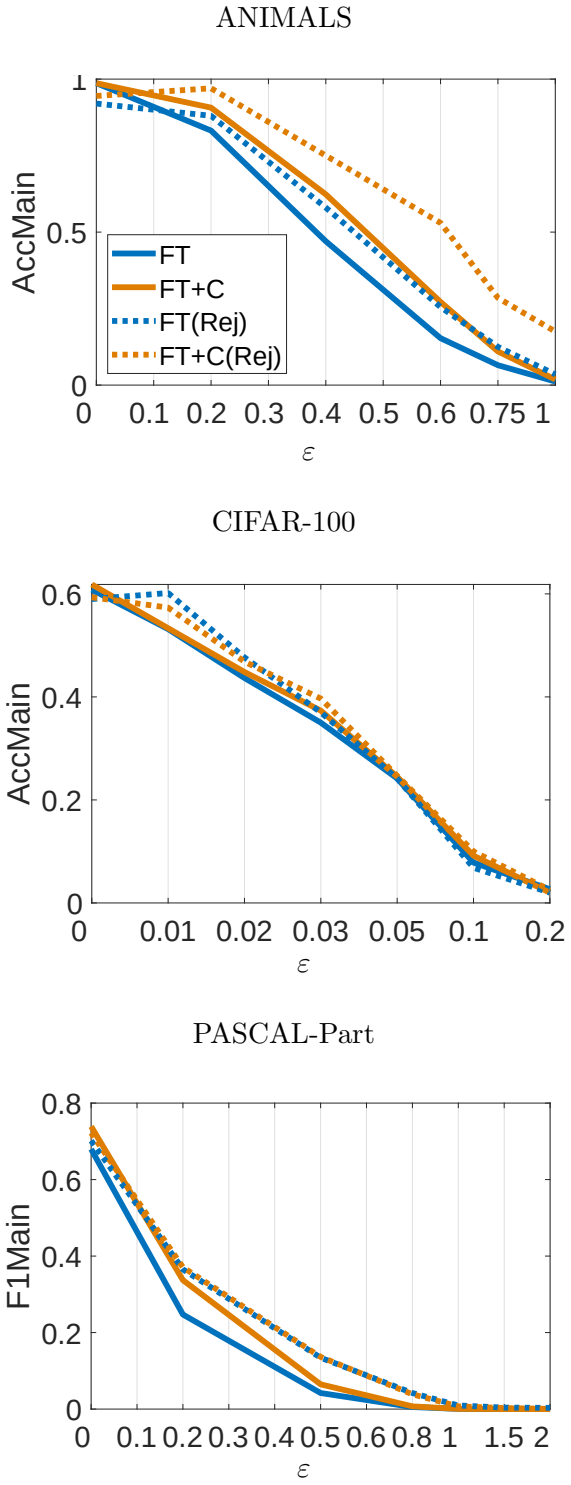


Figure 6.4: White-box attacks in the case of the FT classifiers. Classification quality of vanilla and knowledge-constrained models in function of $\varepsilon$. Dotted plots include rejection (Rej) of inputs that are detected to be adversarial.

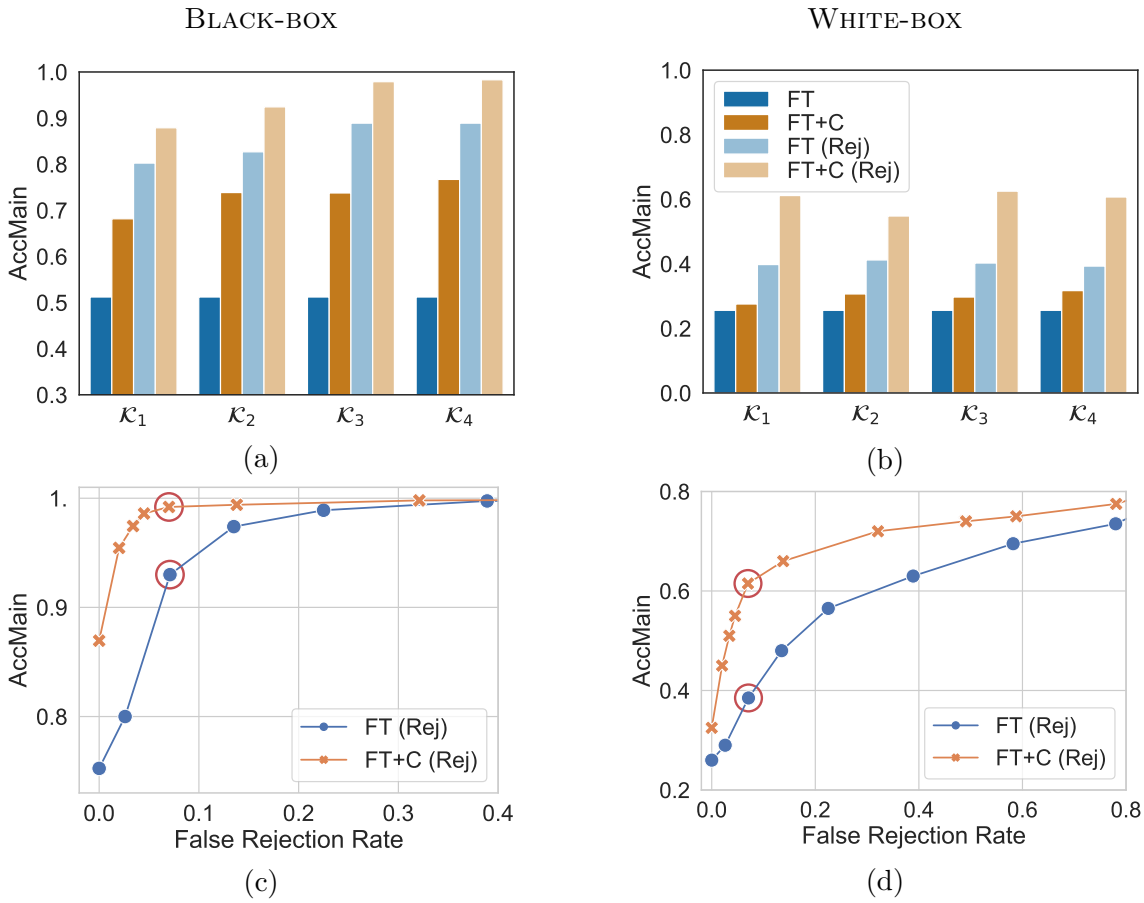


Figure 6.5: Further analysis of the proposed approach in the ANIMALS dataset ($\varepsilon = 0.5$), in black-box (left) and white-box (right) settings; (a,b): increasing amounts of domain knowledge $\mathcal{K}_1, \ldots, \mathcal{K}_4$ - legend reported only on the latter, for better readability; (c,d): different values of the rejection threshold $\theta$ (from larger to smaller values, left-to-right).

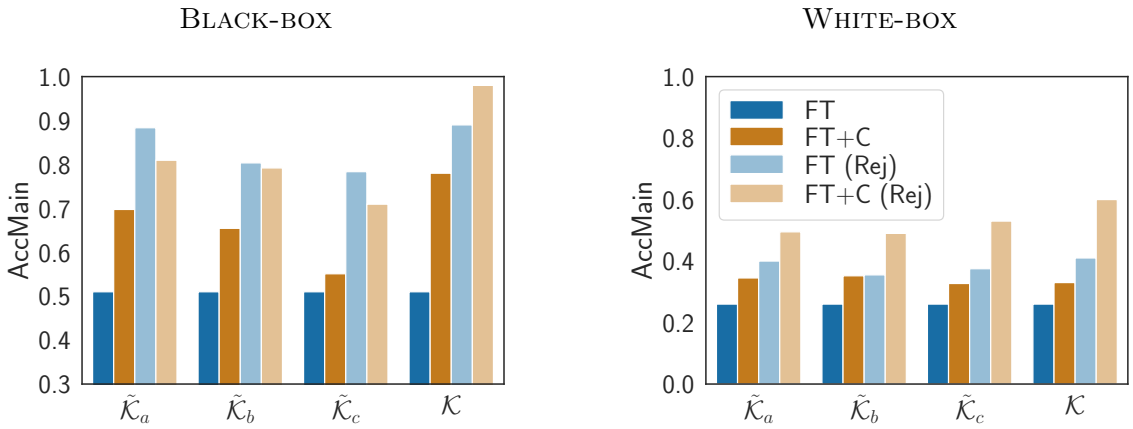


Figure 6.6: Noisy domain knowledge. Analysis of the proposed approach in the same setup of Figure 6.5, when exploiting different noisy knowledge bases $\tilde{\mathcal{K}}_a$, $\tilde{\mathcal{K}}_b$, $\tilde{\mathcal{K}}_c$ (see Sect 3.1 text for details). $\mathcal{K}$ is the original noise-free knowledge.

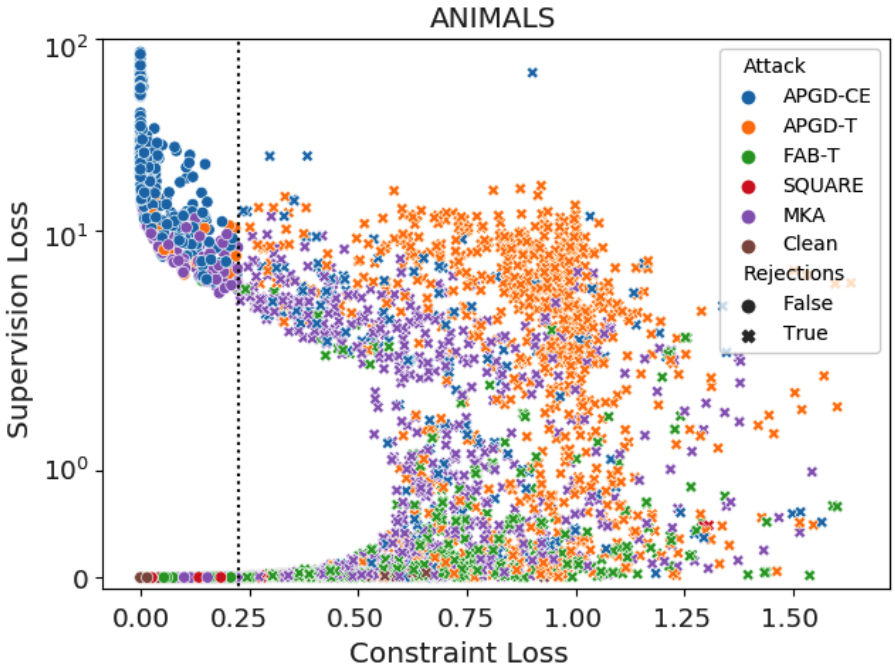


Figure 6.7: Adversarial data generated ($\varepsilon = 0.5$) by different attacks – ANIMALS, TL+C(Rej), black-box. Examples that are rejected/not-rejected by the proposed knowledge-based criterion are depicted with crosses/circles ("Clean" indicates unaltered examples from the test set; the vertical line is the reject threshold).

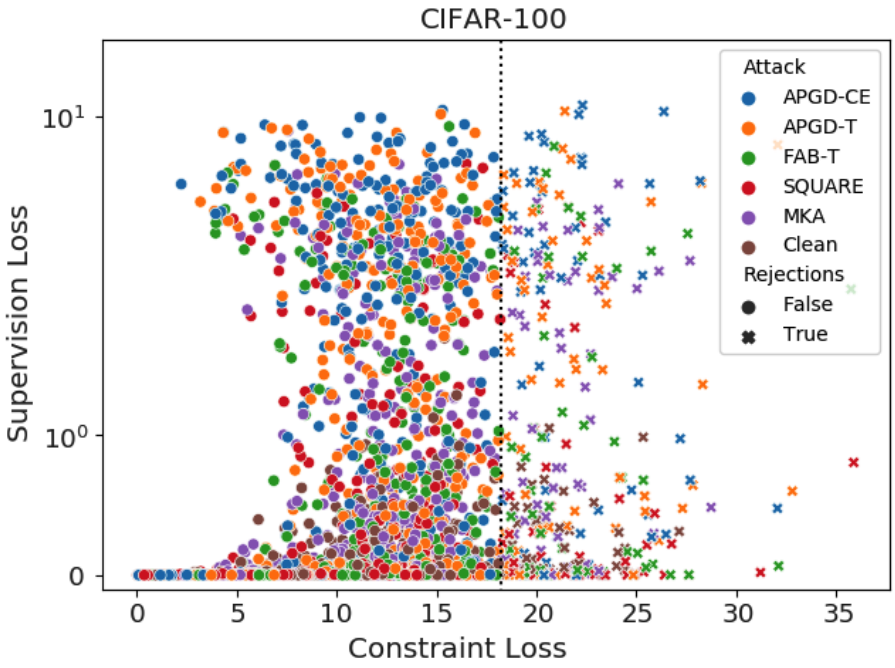


Figure 6.8: Adversarial data generated ($\varepsilon = 0.03$) by different attacks – CIFAR-100, TL+C(Rej), black-box. See Figure 6.7.

## 6.2 Detecting Adversarial Examples in Inner DNN layers

In this section, we report the experiments we ran to evaluate the DNR defense mechanism proposed in Chapter 4.
Section 6.2.1 describes the experimental setup, and Section 6.2.2 discusses obtained results.

### 6.2.1 Experimental setup

**Datasets.** We run experiments on MNIST and CIFAR10 datasets. MNIST handwritten digit data consists of 60,000 training and 10,000 test gray-scale 28x28 images. CIFAR10 consists of 50,000 training and 10,000 test RGB 32x32 images. We normalized the images of both datasets in [0,1] by simply dividing the input pixel values by 255.

**Train-test splits.** We average the results on five different runs. In each run, we consider 10,000 training samples and 1,000 test samples randomly drawn from the corresponding datasets. To avoid overfitting, we train the DNR combiner on the outputs of the base SVMs computed on a separate validation set, using a procedure known as *stacked generalization* (Wolpert, 1992). We use a 3-fold cross-validation procedure to subdivide the training dataset into three folds. We then learn the base SVMs on two folds and classify the remaining (validation) fold, repeating it three times, and alternating the folds. We then concatenate the predicted values for each validation fold and use such values to train the combiner. The deep neural networks (DNNs) used in our experiments are pre-trained on a training dataset (different from the ones that we use to train the SVMs) of 30,000 and 40,000 training samples, respectively, for MNIST and CIFAR10.

**Classifiers.** We compare the DNR approach (which implements rejection here based on the representations learned by three different network layers) against an undefended DNN (without any rejection mechanism) and against the NR defense by Melis et al. (2017) (which implements rejection on top of the representation learned by the output network layer). To implement the undefended DNNs for the MNIST dataset, we used the same architecture suggested by Carlini and Wagner (2017b). For CIFAR10, instead, we consider a lightweight network that, despite its size, allows obtaining high performances. The two considered architectures are shown in Tables 6.7 and 6.8, where the three layers considered by the DNR classifier are highlighted in bold. In particular, we consider (after excluding the Softmax layer) the last three layers for the MNIST network, and the last layer, the last batch-norm layer, and the second-to-last max-pooling layer for the CIFAR10 network (choice aimed at obtaining a reasonable amount of features). Finally, Table 6.9 reports the hyperparameters used for both networks' training.

| Id | Layer Type | Dimension |
|---|---|---|
| `relu1` | Conv. + ReLU | 64 filters (5x5) |
| **`relu2`** | **Conv. + ReLU** | **64 filters (3x3)** |
| **`relu3`** | **Conv. + ReLU** | **64 filters (3x3)** |
| **`relu4`** | **Fully Connected + ReLU** | **32 units** |
| `dropout` | Dropout ($p = 0.5$) | |
| `softmax` | Softmax | 10 units |

Table 6.7: Model architecture of the MNIST neural network (Carlini and Wagner, 2017a). The layers used by DNR and FADER detectors are highlighted in bold.

**Security Evaluation.** We compare these classifiers in terms of their security evaluation curves (Biggio and Roli, 2018), reporting classification accuracy against an increasing $\ell_2$-norm perturbation size $\varepsilon$, used to perturb all test samples. In particular, classification accuracy is computed as follows:

- in the absence of adversarial perturbation (i.e., for $\varepsilon = 0$), classification accuracy is computed as usual, but considering rejects as errors;
- in the presence of adversarial perturbation (i.e., for $\varepsilon > 0$), all test samples become adversarial examples, and we consider them correctly classified if they are assigned either to the rejection class or to their original class (which typically happens when the perturbation is too small to cause a misclassification).

We consider a significant interval of $\varepsilon \in [0, 5]$ and $\varepsilon \in [0, 2]$ for the attacks against MNIST and CIFAR10 datasets, respectively. For DNR and NR, we also report the rejection rates, computed by dividing the number of rejected samples by the number of test samples. Note that the difference between accuracy and rejection rate at each $\varepsilon > 0$ corresponds to the fraction of adversarial examples which are not rejected but still correctly assigned to their original class. Accordingly, under this setting, classifiers exhibiting higher accuracies under attack ($\varepsilon > 0$) can be retained more robust.

**Parameter Setting.** For all the two datasets, DNR detectors' best configuration is looked for in $C \in \{10^{-2}, \ldots, 10^{2}\}$ and $\gamma \in \{10^{-4}, \ldots, 10^{2}\}$ by performing a 3-fold cross-validation procedure to maximize classification accuracy on the unperturbed training data. DNR optimal configurations for each dataset are reported in Table 6.10.
As DNR layer classifiers and combiners are not independently optimized during training, the NR best configuration can be obtained by lookup Table 6.10 for the layer of interest.
We set the rejection threshold $\theta$ for NR and DNR to reject 10% of the samples when no attack is performed (at $\varepsilon = 0$).

| Id | Layer Type | Dimension |
|---|---|---|
| `relu1` | Conv. + Batch Norm. + ReLU | 64 filters (3x3) |
| `relu2` | Conv. + Batch Norm. + ReLU | 64 filters (3x3) |
| `drop1` | Max Pooling + Dropout ($p = 0.1$) | 2x2 |
| `relu3` | Conv. + Batch Norm. + ReLU | 128 filters (3x3) |
| `relu4` | Conv. + Batch Norm. + ReLU | 128 filters (3x3) |
| `drop2` | Max Pooling + Dropout ($p = 0.2$) | 2x2 |
| `relu5` | Conv. + Batch Norm. + ReLU | 256 filters (3x3) |
| `relu6` | Conv. + Batch Norm. + ReLU | 256 filters (3x3) |
| **`drop3`** | **Max Pooling + Dropout ($p = 0.3$)** | 2x2 |
| **`relu7`** | **Conv. + Batch Norm. + ReLU** | 512 filters (3x3) |
| `drop4` | Max Pooling + Dropout ($p = 0.4$) | 2x2 |
| **`linear`** | **Fully Connected** | 512 units |
| `softmax` | Softmax | 10 units |

Table 6.8: Model architecture of the CIFAR10 neural network. The layers used by DNR and FADER detectors are highlighted in bold.

| Parameter | MNIST | CIFAR10 |
|---|---|---|
| Learning Rate | 0.1 | 0.01 |
| Momentum | 0.9 | 0.9 |
| Dropout | 0.5 | (see Table 6.8) |
| Batch Size | 128 | 100 |
| Epochs | 50 | 75 |

Table 6.9: Parameters used to train the MNIST and CIFAR10 DNNs.

### 6.2.2 Experimental Results

The results are reported in Figure 6.9. In the absence of attack ($\varepsilon = 0$), the undefended DNNs slightly outperform NR and DNR since the latter wrongly reject also some unperturbed samples.

Under attack, ($\varepsilon > 0$), when the amount of injected perturbation is exiguous, the rejection rate of both NR and DNR increases jointly with $\varepsilon$, as the adversarial examples are located far from the rest of the training classes in the representation space (i.e., the intermediate representations learned by the neural network). For larger $\varepsilon$, both NR and DNR can no longer correctly detect the adversarial examples, as they tend to become indistinguishable from the rest of the training samples (in the representation space in which NR and DNR operate). Both defenses outperform the undefended DNNs on the adversarial samples, and DNR slightly outperforms

| MNIST | | | CIFAR10 | | |
|---|---|---|---|---|---|
| Layer | C | $\gamma$ | Layer | C | $\gamma$ |
| `relu2` | 10 | 1e-3 | `drop3` | 10 | 1e-3 |
| `relu3` | 10 | 1e-2 | `relu7` | 1.0 | 1e-3 |
| `relu4` | 1.0 | 1e-2 | `linear` | 1.0 | 1e-2 |
| `combiner` | 1e-1 | 1.0 | `combiner` | 1e-4 | 1.0 |

Table 6.10: DNR configurations for MNIST (left) and CIFAR10 (right) datasets.

NR, exhibiting a more graceful decrease in performance. Although NR tends to reject more samples for $\varepsilon \in [0.1, 1]$ on CIFAR and for $\varepsilon = 0.5$ on MNIST, its accuracy is lower than DNR. The reason is that DNR remains more accurate than NR when classifying samples that are not rejected. This also means that DNR provides tighter boundaries closer to the training classes than NR, thanks to the exploitation of lower-level network representations, which makes the corresponding defended classifier more difficult to evade.

Finally, in Figure 6.10 and 6.11, we show how the selection of the rejection threshold $\theta$ allows us to trade security against adversarial examples (i.e., accuracy on the y-axis) for a more accurate classifier on the unperturbed samples (reported in terms of the rejection rate of unperturbed samples on the x-axis). In particular, increasing (decreasing) the rejection threshold amounts to increasing (decreasing) the fraction of correctly-detected adversarial examples, and to increasing (decreasing) the rejection rate when no attack is performed.

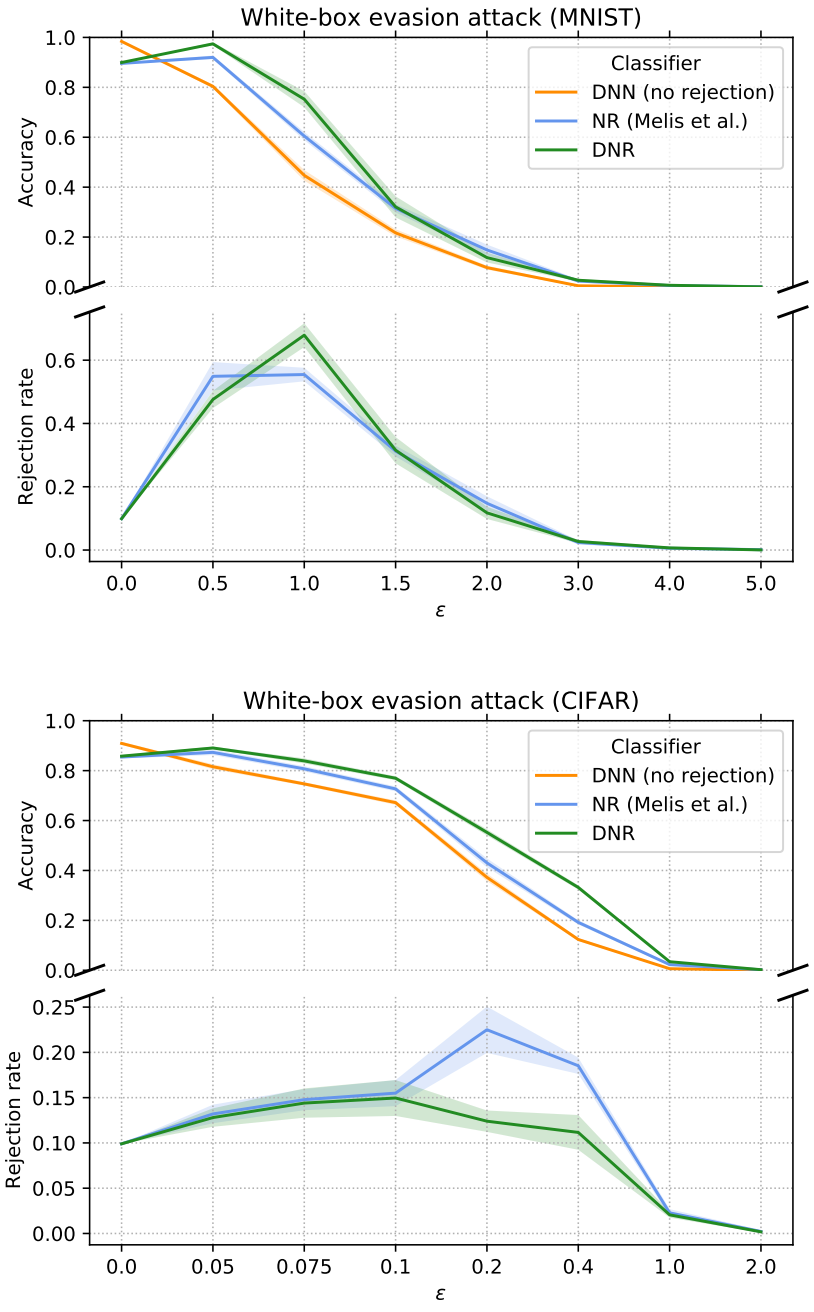


Figure 6.9: Security evaluation curves for MNIST (top) and CIFAR10 (bottom) data, reporting mean accuracy (and standard deviation) against $\varepsilon$-sized attacks.

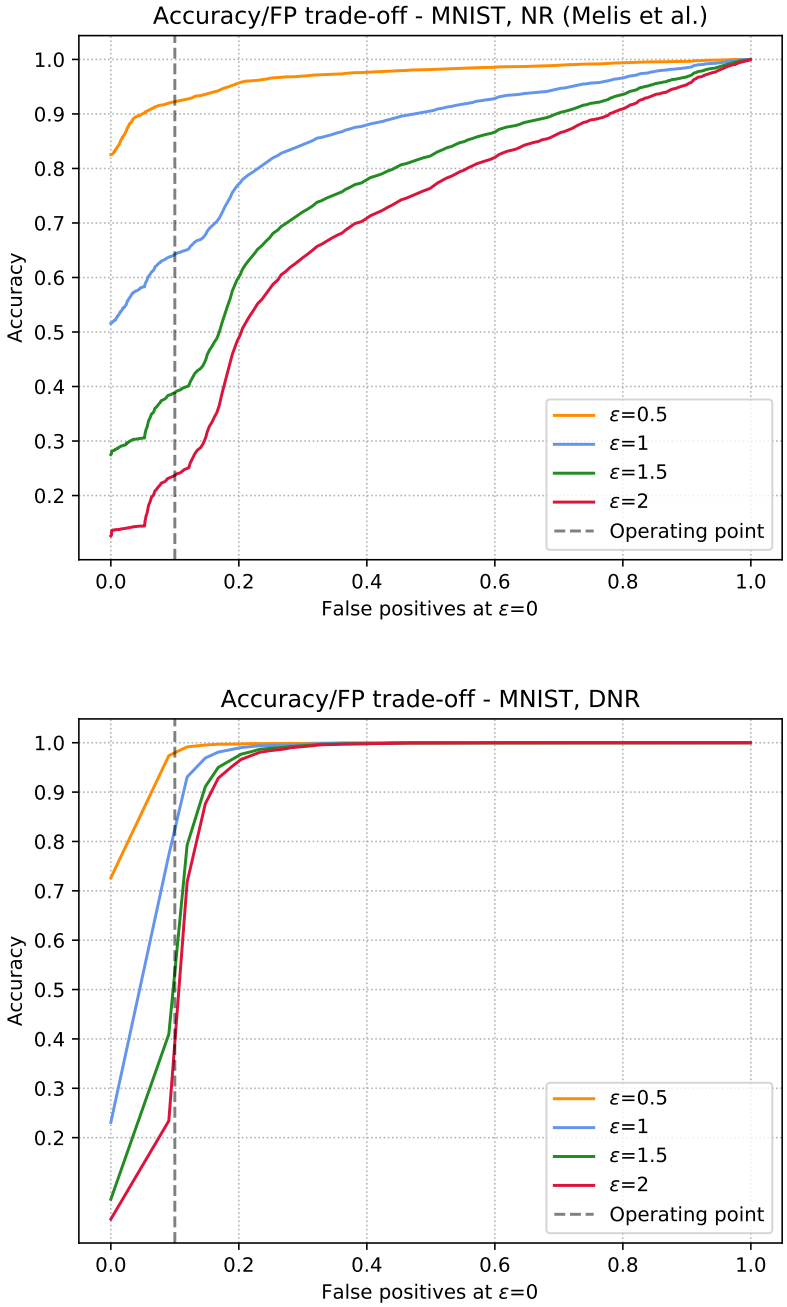


Figure 6.10: Influence of the rejection threshold $\theta$ on classifier accuracy under attack (y-axis) vs false rejection rate (i.e., fraction of wrongly-rejected unperturbed samples) on MNIST for NR (top) and DNR (bottom), for different $\varepsilon$-sized attacks. The dashed line highlights the performance at a 10% false rejection rate (i.e., the operating point used in our experiments).

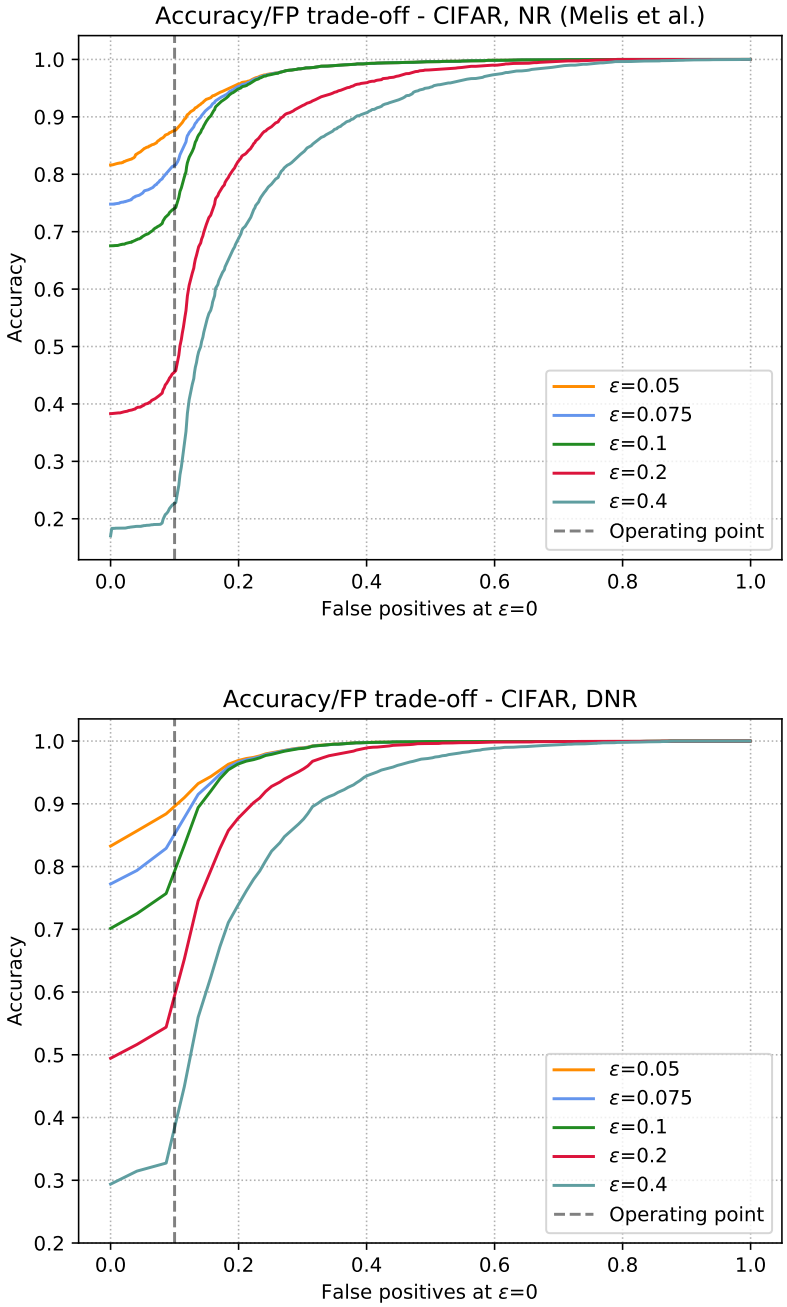


Figure 6.11: Influence of the rejection threshold $\theta$ on classifier accuracy under attack (y-axis) vs false rejection rate (i.e., fraction of wrongly-rejected unperturbed samples) on CIFAR10 for NR (top) and DNR (bottom), for different $\varepsilon$-sized attacks. The dashed line highlights the performance at a 10% false rejection rate (i.e., the operating point used in our experiments).

## 6.3 Improving the Efficiency of Prototypes-based Detectors

In this section, we empirically evaluate the security of the proposed FADER defense mechanism (i) against defense-aware adversarial examples and (ii) in a black-box setting where the attacker is essentially unaware of the defense mechanism used to protect the DNN classifier. After detailing our experimental setup (Section 6.3.1), we report the classifier's performance under attack by comparing it with NR and DNR detectors (Section 6.3.2). We also consider an additional set of experiments to validate the effectiveness of the proposed adaptive attack (PGD-LS) against standard PGD attacks in Section 6.3.3.

### 6.3.1 Experimental Setup

We discuss here the datasets we use to evaluate our defense method and the classifiers we compare the performance with. The undefended DNNs run on the GPU, while all the attached detectors are executed on the CPU.

**Datasets.** Our analysis is performed on three image recognition datasets: MNIST, CIFAR10, and a ten-class subset of ImageNet (Deng et al., 2009) referred to as *ImageNet10*. The first two datasets were already introduced in Section 6.2.1, whereas the ImageNet10 dataset is obtained by extending the Imagenette[9] dataset with a disjoint set of samples from the ImageNet validation split, so to compensate the lack of a publicly available test split for Imagenette. In particular, we selected 50 images per class coming from the ImageNet validation split for a total of 500 test samples. ImageNet RBG samples are resized to 256x256 pixels and center-cropped to 224x224 pixels. For all datasets, images are normalized in $[0, 1]$ by dividing the input pixel values by 255.

**Classifiers.** We compare the performance of an undefended DNN (i.e., not implementing any rejection mechanism), which represents our baseline, with NR and DNR defense methods and their *fast* variants, i.e., the FADER technique is applied. To implement the undefended DNNs for MNIST and CIFAR10 datasets, we use the same architectures described in Section 6.2.1. For ImageNet10, we rely on the pre-trained AlexNet (Krizhevsky, 2014) DNN available on TorchVision.[10] We keep only outputs related to the ten classes of interest. As for the detectors, we consider the single-layer rejection mechanism on top of the pre-softmax activation layer, in the form of an SVM-RBF for NR (Melis et al., 2017) and the DNR defense approach (see Chapter 4) employing SVMs with RBF kernel as both layer detectors and the top combiner. For both NR and DNR, we provide *fast* variants, employing size-controlled RBF networks, denoted as NR-RBF and DNR-RBF, respectively. As for

[9] https://github.com/fastai/imagenette
[10] https://pytorch.org/vision

DNR implementation, we limit the number of layers inspected by the deep detectors to three. For the MNIST and CIFAR10 networks, we consider the same layers as in Section 6.2.1, whereas for the AlexNet network on ImageNet10, we select the sixth last, fourth last, and third last layers.

**Training-test Splits.** We follow the same procedures described in Section 6.2.1 for the training of the MNIST and CIFAR10 networks and the experiment repetitions. Regarding experiments on ImageNet10, we use images from the ImageNet validation set as test data, obtaining 500 test samples. As a pre-trained network, the ImageNet10 DNN is trained on the whole ImageNet training set. As such, we exploit the Imagenette samples to train NR and DNR detectors. To avoid intersections between train and test data, we exclude from the Imagenette dataset a small number of images that were also among the ImageNet validation split.

**Parameter Setting.** For all three datasets, we use the procedure in Section 6.2.1 to find the DNR detectors' best configurations. The optimal configuration for the ImageNet10 dataset is reported in Table 6.11 (the NR best configuration can be obtained by lookup at Table 6.11 for the layer of interest).

The architectures of FADER-based solutions are designed to maximize the prototype reduction rate while achieving comparable performances on clean test samples (see Tables 6.12 and 6.13). In terms of training, RBF-based neurons are optimized using `pytorch`[11] Adam optimizer with default settings for 250 epochs. Rejection threshold $\theta$ is set, in all the considered cases, to reject 10% of the samples when no attack is performed (at $\varepsilon = 0$).

| ImageNet10 | | |
|---|---|---|
| Layer | C | $\gamma$ |
| `clf1` | 10 | 1e-4 |
| `clf3` | 1 | 1e-4 |
| `clf4` | 1 | 1e-4 |
| `combiner` | 1 | 1e-1 |

Table 6.11: DNR configuration for ImageNet10 dataset.

**Security Evaluation.** We compare the aforementioned undefended neural networks and the rejection-based architectures in terms of their security evaluation curves (Biggio and Roli, 2018), using the same approach described in Section 6.2.1. For ImageNet10, we perform an evaluation at fixed $\varepsilon = 1.0$ in a white-box setting as a representative value for evaluating detectors' performance under attack.

[11] https://pytorch.org

### 6.3.2 Experimental Results

**Adversarial Robustness Evaluation.** The security evaluation curves for both white-box (defense-aware) and black-box settings against MNIST and CIFAR10 datasets are reported in Figures 6.12 and 6.13, top subplots. In the bottom subplots, we report the corresponding rejection rates at increasing $\ell_2$-norm perturbation size. We also report in Table 6.16 the performance of all classifiers for the ImageNet10 dataset against our attack (PGD-LS) at distance $\varepsilon = 1.0$.
In the absence of an attack ($\varepsilon = 0$), the undefended DNN slightly outperforms all rejection-based detectors due to a portion of samples that are incorrectly flagged as positive. Under attack ($\varepsilon > 0$), all detectors show improved robustness to adversarial examples compared to the standard DNN, as their accuracy decreases more gracefully. Notably, the performance of all detectors even increases for low values of $\varepsilon$, as the slightly modified testing images immediately become blind-spot adversarial examples, ending up in a region that is far from the rest of the data. As the input perturbation increases, such samples are gradually drifted inside a different class, making them indistinguishable for rejection-based defenses (Melis et al., 2017).
Interestingly, for the MNIST case, we notice a similar increase of accuracy for NR, DNR, and NR-RBF detectors tested in the black-box setting at the highest values of $\varepsilon$. Due to the imperfect attacker's knowledge of the target classifier, when the $\ell_2$ distance is large, the adversarial samples end up again once farther from the rest of the data and are detected by the defenses. This behavior is also confirmed by the corresponding rejection rate, which increases jointly with $\varepsilon$. By comparing the average adversarial robustness of the considered detectors, we show that FADER provides comparable effectiveness against adversarial examples as original solutions.

**Prototype Reduction.** Tables 6.12, 6.13, and 6.14 report for MNIST, CIFAR10, and ImageNet10 datasets, respectively, the number of prototypes employed in each detector component along with the estimated prototype reduction rate for FADER-based defenses. Notably, up to a 73× prototype reduction rate can be achieved for NR and 20× for DNR without performance drops on both natural and adversarial data for the MNIST dataset. The same holds for the other considered datasets: FADER detector variants achieve up to a 50× and 82× prototype reduction rate against original detectors for CIFAR10 and ImageNet10, respectively. This prototype reduction rate has a substantial impact in terms of space complexity; in particular, if we only consider the reference prototypes that are stored in memory to perform classification, FADER variants allow us to save a considerable amount of space. We quantify this in Tables 6.12, 6.13, and 6.14, considering a (conservative) memory consumption estimate of just 8 bytes for each floating-point number.

**Runtime Complexity.** To effectively quantify the speedup induced at runtime by FADER variants over the SVM-based detectors, a benchmark evaluation is considered here. Figure 6.14 reports the time spent by the undefended DNN and each detector, executed on our workstation (as detailed in the experimental setup), to

| Detector | # prototypes | | | | | | memory | Accuracy |
|---|---|---|---|---|---|---|---|---|
| | `relu2` | `relu3` | `relu4` | `combiner` | `total` | `reduction` | | |
| NR | - | - | 736 | - | 736 | - | 188 KB | 0.984 |
| **NR-RBF** | - | - | **10** | - | **10** | **$\sim$ 73×** | **3 KB** | **0.985** |
| DNR | 2304 | 2375 | 736 | 9152 | 11527 | - | 43 MB | 0.961 |
| **DNR-RBF** | **250** | **250** | **50** | **10** | **560** | **$\sim$ 20×** | **4** MB | **0.989** |
| # features | 1600 | 576 | 32 | 30 | | | | |

Table 6.12: Comparison of the number of prototypes used by each component of the rejection-based defense architectures (FADER in bold) on the MNIST dataset. We also report the mean accuracy of each detector at $\varepsilon = 0$ and the memory consumption related to the stored reference prototypes.

| Detector | # prototypes | | | | | | memory | Accuracy |
|---|---|---|---|---|---|---|---|---|
| | `drop3` | `relu7` | `linear` | `combiner` | `total` | `reduction` | | |
| NR | - | - | 5257 | - | 5275 | - | 22 MB | 0.915 |
| **NR-RBF** | - | - | **100** | - | **100** | **$\sim$ 50×** | **410 KB** | **0.911** |
| DNR | 7198 | 3100 | 5257 | 10000 | 25555 | - | 311 MB | 0.913 |
| **DNR-RBF** | **500** | **300** | **100** | **100** | **1000** | **$\sim$ 28×** | **22 MB** | **0.892** |
| # features | 4096 | 2048 | 512 | 30 | | | | |

Table 6.13: Comparison of the number of prototypes used by each component of the rejection-based defense architectures (FADER in bold) on the CIFAR10 dataset. We also report the mean accuracy of each detector at $\varepsilon = 0$ and the memory consumption related to the stored reference prototypes.

classify an increasing number of test samples, averaged over 10 repetitions, for each dataset. As detailed in Section 5.2, the runtime complexity scales linearly with the test set size. The results for the last point of the curve, corresponding to 5000 test samples, are also reported in Table 6.15 (standard deviation in brackets) to better quantify the time reduction. The expected speedup due to a reduced number of prototypes is confirmed by the time measurements on all three datasets, reaching up to 3.46× on CIFAR10.
To conclude, even though the reduction of time complexity is not as large as the prototype reduction rates, FADER enables a significant reduction in space complexity, which makes it well suited to low-power, embedded devices or edge-cloud systems.

### 6.3.3 Comparison with PGD

In this section, we compare the performance of our attack PGD-LS (Algorithm 2) against the standard Projected Gradient Descent (PGD) with normalized step proposed in Madry et al. (2018). The performance under the white-box (defense-aware)

| Detector | # prototypes | | | | | | | Accuracy |
|---|---|---|---|---|---|---|---|---|
| | `clf1` | `clf3` | `clf4` | `combiner` | `total` | `reduction` | memory | |
| NR | - | - | 4103 | - | 4103 | - | 134 MB | 0.894 |
| **NR-RBF** | - | - | **50** | - | **50** | **$\sim 82\times$** | **2 MB** | **0.882** |
| DNR | 6729 | 6590 | 4103 | 1535 | 14854 | - | 571 MB | 0.894 |
| **DNR-RBF** | **200** | **200** | **100** | **50** | **550** | **$\sim 27\times$** | **6 MB** | **0.874** |
| # features | 4096 | 4096 | 4096 | 30 | | | | |

Table 6.14: Comparison of the number of prototypes used by each component of the rejection-based defense architectures (FADER in bold) on the ImageNet10 dataset. We also report the mean accuracy of each detector at $\varepsilon = 0$ and the memory consumption related to the stored reference prototypes.

| Classifier | MNIST | CIFAR10 | ImageNet10 |
|---|---|---|---|
| DNN | $557 \pm 4$ | $1214 \pm 1$ | $15453 \pm 280$ |
| NR | $667 \pm 6$ | $2575 \pm 5$ | $20543 \pm 750$ |
| **NR-RBF** | $\mathbf{568 \pm 7}$ **(1.17×)** | $\mathbf{1263 \pm 1}$ **(2.03×)** | $\mathbf{15843 \pm 639}$ **(1.30×)** |
| DNR | $5281 \pm 20$ | $16927 \pm 67$ | $73336 \pm 3324$ |
| **DNR-RBF** | $\mathbf{2053 \pm 32}$ **(2.57×)** | $\mathbf{4891 \pm 121}$ **(3.46×)** | $\mathbf{50069 \pm 615}$ **(1.46×)** |

Table 6.15: Expected time ± standard deviation (in milliseconds) to classify 5000 samples, averaged over 10 runs. The reduction attained by each FADER detector (in bold), computed as the ratio between the time spent by the SVM-based detector and the corresponding FADER variant, is reported in parenthesis.

attack against MNIST, CIFAR10, and ImageNet10 datasets, using 1000 test samples (500 for ImageNet10), is reported in Table 6.16. It is worth pointing out that PGD-LS especially outperforms PGD when optimizing attacks becomes more challenging, e.g., due to the presence of wide flat local optima, which are difficult to escape if the step size is not sufficiently large. This happens when attacking NR and NR-BRF on MNIST, and DNR on MNIST and CIFAR10. In these cases, PGD would only give a false sense of security (i.e., induce one to think that such methods are more robust), whereas PGD-LS successfully evades them with a higher probability, providing a more reliable robustness evaluation (Athalye et al., 2018a; Carlini et al., 2019).

| Classifier | MNIST ($\varepsilon = 1.5$) | | CIFAR10 ($\varepsilon = 0.2$) | | ImageNet10 ($\varepsilon = 1.0$) | |
|---|---|---|---|---|---|---|
| | PGD-LS | PGD | PGD-LS | PGD | PGD-LS | PGD |
| DNN | 0.20 | 0.25 | 0.42 | 0.38 | 0.23 | 0.24 |
| NR | 0.60 | 0.82 | 0.46 | 0.43 | 0.41 | 0.43 |
| **NR-RBF** | **0.59** | **0.84** | **0.46** | **0.40** | **0.40** | **0.51** |
| DNR | 0.71 | 1.00 | 0.53 | 0.95 | 0.55 | 0.58 |
| **DNR-RBF** | **0.62** | **0.74** | **0.45** | **0.40** | **0.55** | **0.56** |

Table 6.16: Classification accuracy under white-box attack at fixed values of $\varepsilon$ for the different classifiers (FADER in bold), MNIST data ($\varepsilon = 1.5$), CIFAR10 data ($\varepsilon = 0.2$) and ImageNet10 data ($\varepsilon = 1.0$), using our PGD with Line Search (PGD-LS, Algorithm 2), and a standard PGD with normalized step (Madry et al., 2018). PGD-LS outperforms PGD when optimizing attacks is more challenging, e.g., when attacking NR and NR-BRF on MNIST, and DNR on MNIST and CIFAR10.

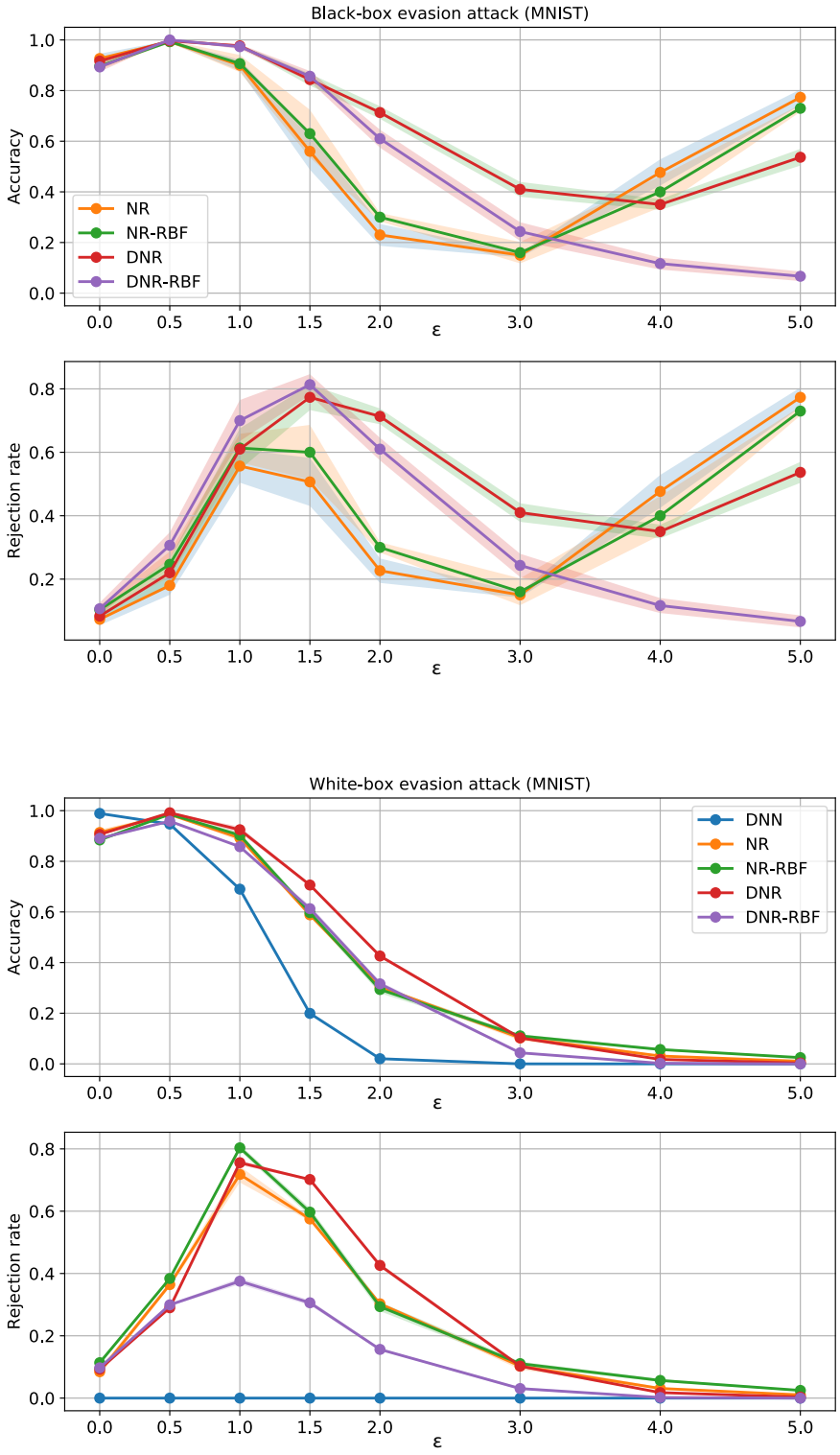


Figure 6.12: Security evaluation curves for MNIST data, under black-box (top) and white-box (bottom) settings. Mean accuracy at increasing $\ell_2$-norm perturbation size is reported in the top subplots, while the bottom subplots show the corresponding rejection rates.

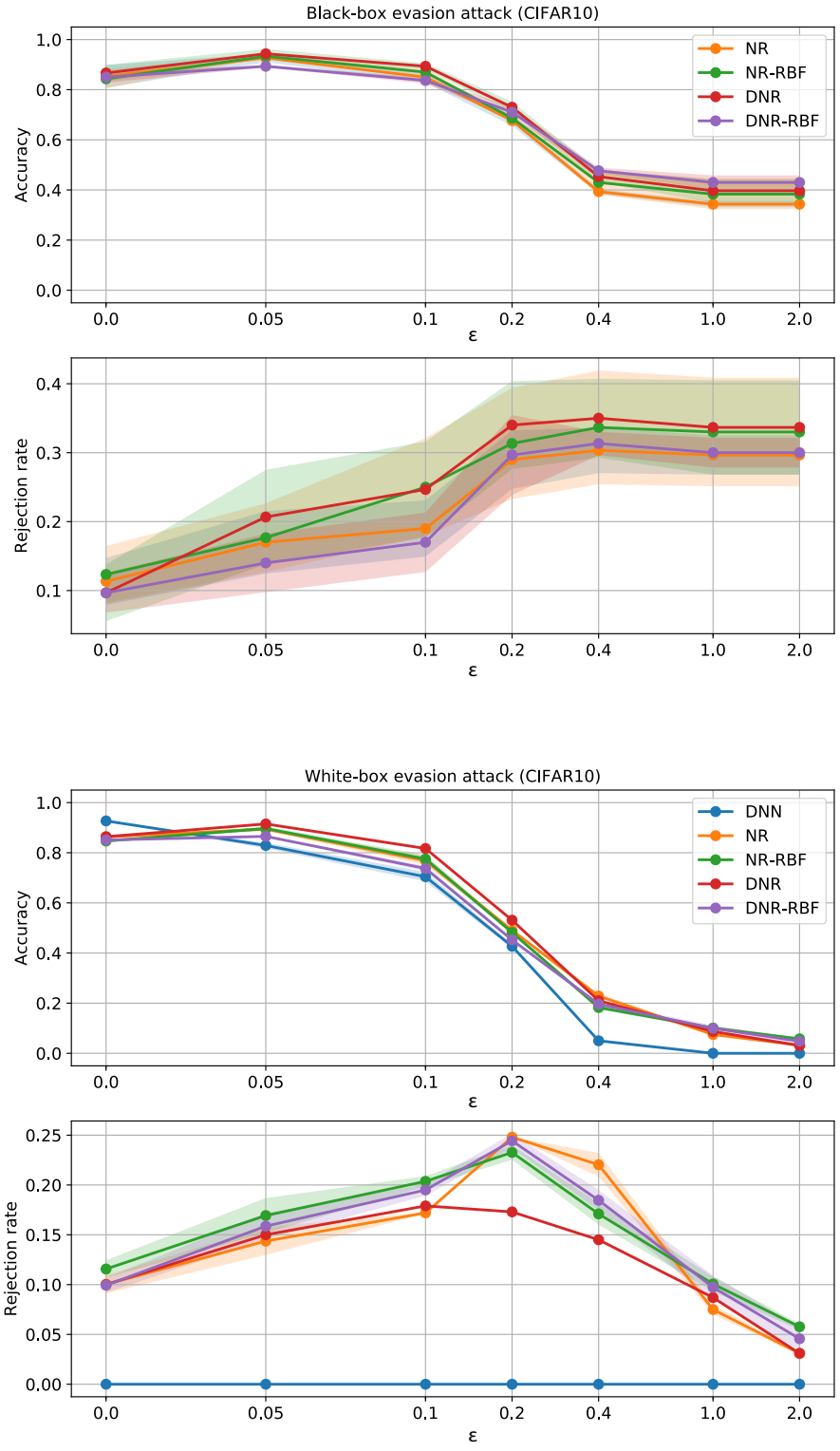


Figure 6.13: Security evaluation curves for CIFAR10 data, under black-box (top) and white-box (bottom) settings. Mean accuracy at increasing $\ell_2$-norm perturbation size is reported in the top subplots, while the bottom subplots show the corresponding rejection rates.

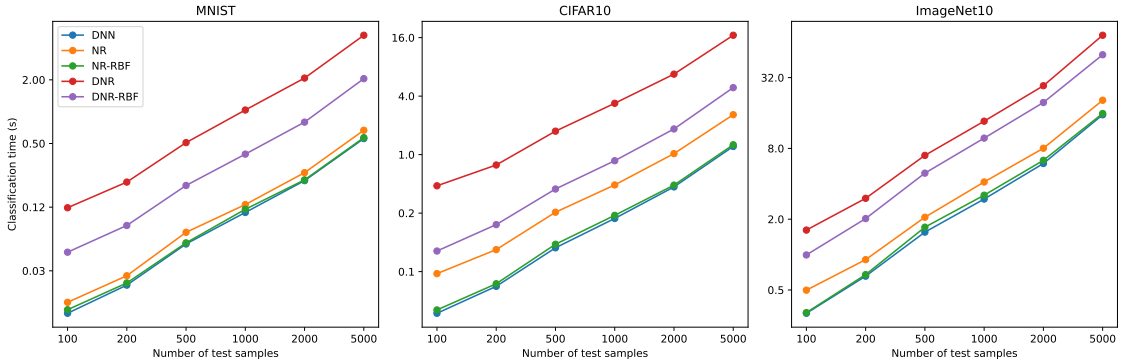


Figure 6.14: Classification time (in seconds) against an increasing number of test samples, averaged on 10 runs, using batches of 256 samples. Similar linear dependencies are also found for batch sizes of 32, 64, 128, 512, and 1024. FADER gives a consistent advantage over NR and DNR, as also quantified in Table 6.15.

# Chapter 7

# Conclusions

In this thesis, we proposed different detection methods to increase the robustness of machine learning models against adversarial examples, and we evaluated them showing that, as for all the state-of-the-art defenses, they can be fooled by an attacker that has full-knowledge of them and performs adaptive strategies, injecting stronger perturbations.

We first investigated the role of domain knowledge in adversarial settings. Focusing on multi-label classification, we injected knowledge expressed by First-Order Logic in the training stage of the classifier, not only with the aim of improving its quality but also as a means to build a detector of adversarial examples at no additional cost. We proposed a multi-label attack procedure and showed that knowledge-constrained classifiers could improve their robustness against both black-box and white-box attacks, and, using the same knowledge, they can detect adversarial attacks. The proposed adversarial example rejection scheme is based on the idea of dealing with classifiers that fulfill the knowledge-related constraints over the space regions in which the non-malicious data are distributed, not guaranteeing such fulfillment in the rest of the space.

We then proposed *Deep Neural Rejection* (DNR), i.e., a multi-layer rejection mechanism that, differently from other rejection approaches against adversarial examples, does not require generating adversarial examples at training time, and it is less computationally demanding. Our approach can be applied to pre-trained network architectures to implement a defense against adversarial attacks at test time. The base classifiers and the combiner used in our DNR approach are trained separately. To properly evaluate DNR's response to adversarial attacks, we also designed a novel attack algorithm that takes into account the defense to avoid overestimating the performance under attack.

As a follow-up of our work, we presented FADER (*Fast Adversarial Example Rejection*), i.e., a technique to speed up rejection-based defenses against adversarial examples. FADER exploits RBF networks to control the number of reference prototypes required for predictions, resulting in accuracy vs. detection time efficiency

gain. In our experiments, we demonstrated a 73× prototypes reduction with respect to analyzed detectors for the MNIST dataset, up to 50× prototypes reduction for the CIFAR10 image recognition task, and up to 82× in the case of the ImageNet10 dataset, while maintaining comparable performance on both clean and adversarial data. This can have a strong impact on real-world scenarios involving adversarial-example detection on low-capability (e.g., edge) devices. We further provided a comprehensive review of multiple detector-based adversarial detection techniques from the literature, framing them in the form of a proposed adversarial-example detection framework designed to accommodate both existing and newer methods to come. Experimental results on different image classification tasks highlight FADER-based defenses as more *efficient* solutions than original ones in terms of required prototypes while maintaining comparable performances both on clean data and under attack.

## 7.1 Limitations and Future Works

All the proposed defenses share a common limitation: against adaptive white-box attackers, they are able to increase the robustness of DNNs against adversarial examples only with small amounts of adversarial perturbation, while their effectiveness gradually decreases with higher perturbations. This makes our defenses effective only under specific constraints, thus they might not always be useful. It is also worth remarking that these results come from a worst-case evaluation, whereas in a real-world application scenario, the attacker usually does not know the presence and the behavior of the defense mechanism. It remains an open issue to understand how hard for an attacker would be to infer them in practical cases, especially considering the domain knowledge exploited in the first approach.

To improve the adversarial robustness of our defenses, we will consider combining them with adversarial training. In particular, for the first approach, adversarial training can be intermixed with knowledge constraints to strengthen the violation of the constraints out of the distribution of the real data. Furthermore, we are also interested in testing all the proposed approaches on the out-of-distribution samples detection problem, for which the classifier could not provide sufficient confidence for its decisions. In several application domains, it might be important to avoid the system outputs such decisions, preferring to abstain.

Regarding the first proposed approach, we plan to design a learnable model that decides whether to reject or not in function of the fulfillment of each specific logic formula, going beyond a simple-but-effective threshold on the cumulative constraint loss.

As an additional future work on DNR, it would be interesting to perform an end-to-end training of the proposed classifier similarly to the approaches proposed in Thulasidasan et al. (2019) and Geifman and El-Yaniv (2019). Another research direction may be that of testing our defense against training-time poisoning at-

tacks (Biggio et al., 2012; Jagielski et al., 2018; Xiao et al., 2015; Mei and Zhu, 2015; Biggio and Roli, 2018).
Moreover, we aim to improve FADER performance under attack by employing proper input gradient regularization (Demontis et al., 2019; Simon-Gabriel et al., 2019) and testing novel FADER architectural variants to further reduce the computational overhead of adversarial-example detection schemes.

## 7.2 Closing Remarks

To sum up, after presenting three detection methods for adversarial examples, we provide insights into their effectiveness under worst-case scenarios. We believe that these findings will contribute to understanding how to build more effective and efficient adversarial examples detection mechanisms that can be combined with other defense techniques. We also provide new adaptive optimization strategies to thoroughly evaluate them. Finally, we believe that the first proposed defense will open the investigation of domain knowledge as a feature to further improve the robustness of multi-label classifiers against adversarial attacks.

### Acknowledgements

This work has been supported by BMK, BMDW, and the Province of Upper Austria in the frame of the COMET Programme managed by FFG in the COMET Module S3AI.

# Bibliography


S. Akcay, A. Atapour-Abarghouei, and T. P. Breckon. Ganomaly: Semi-supervised anomaly detection via adversarial training. In *Asian Conference on Computer Vision*, pages 622–637. Springer, 2018.

J.-B. Alayrac, J. Uesato, P.-S. Huang, A. Fawzi, R. Stanforth, and P. Kohli. Are labels required for improving adversarial robustness? In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

M. Andriushchenko, F. Croce, N. Flammarion, and M. Hein. Square Attack: A Query-Efficient Black-Box Adversarial Attack via Random Search. In A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, editors, *Computer Vision – ECCV 2020*, pages 484–501, Cham, 2020. Springer International Publishing. ISBN 978-3-030-58592-1.

A. Araujo, L. Meunier, R. Pinot, and B. Negrevergne. Robust Neural Networks using Randomized Adversarial Training. *arXiv preprint, arXiv:1903.10219*, 2020.

A. Athalye, N. Carlini, and D. Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pages 274–283. PMLR, 10–15 Jul 2018a.

A. Athalye, L. Engstrom, A. Ilyas, and K. Kwok. Synthesizing robust adversarial examples. In *ICLR*, 2018b.

R. Babbar and B. Schölkopf. Adversarial extreme multi-label classification. *arXiv preprint, arXiv:1803.01570*, 2018.

P. Barbiero, G. Ciravegna, F. Giannini, P. Li'o, M. Gori, and S. Melacci. Entropy-based logic explanations of neural networks. In *AAAI*, 2022.

M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar. Can machine learning be secure? In *Proc. ACM Symp. Information, Computer and Comm. Sec.*, ASIACCS '06, pages 16–25, New York, NY, USA, 2006. ACM.

A. Bendale and T. E. Boult. Towards open set deep networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1563–1572, 2016.

A. N. Bhagoji, D. Cullina, and P. Mittal. Dimensionality reduction as a defense against evasion attacks on machine learning classifiers. *arXiv preprint arXiv:1704.02654*, 2, 2017.

B. Biggio and F. Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 84:317–331, 2018.

B. Biggio, G. Fumera, I. Pillai, and F. Roli. Image spam filtering by content obscuring detection. In *Fourth Conference on Email and Anti-Spam (CEAS)*, Microsoft Research Silicon Valley, Mountain View, California, 2-3 August 2007a.

B. Biggio, G. Fumera, I. Pillai, and F. Roli. Image spam filtering using visual information. In *14th International Conference on Image Analysis and Processing*, pages 105–110, Modena, Italy, 10-14 September 2007b. IEEE Computer Society.

B. Biggio, G. Fumera, I. Pillai, and F. Roli. Improving image spam filtering using image text features. In *Fifth Conference on Email and Anti-Spam (CEAS)*, Mountain View, CA, USA, 21 August 2008.

B. Biggio, B. Nelson, and P. Laskov. Poisoning attacks against support vector machines. In J. Langford and J. Pineau, editors, *29th Int'l Conf. on Machine Learning*, pages 1807–1814. Omnipress, 2012.

B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli. Evasion attacks against machine learning at test time. In H. Blockeel, K. Kersting, S. Nijssen, and F. Železný, editors, *Machine Learning and Knowledge Discovery in Databases (ECML PKDD), Part III*, volume 8190 of *LNCS*, pages 387–402. Springer Berlin Heidelberg, 2013.

W. Brendel, J. Rauber, M. Kümmerer, I. Ustyuzhaninov, and M. Bethge. *Accurate, Reliable and Fast Robustness Evaluation*. Curran Associates Inc., Red Hook, NY, USA, 2019.

M. Brückner, C. Kanzow, and T. Scheffer. Static prediction games for adversarial learning problems. *J. Mach. Learn. Res.*, 13:2617–2654, September 2012.

J. Buckman, A. Roy, C. Raffel, and I. Goodfellow. Thermometer encoding: One hot way to resist adversarial examples. In *International Conference on Learning Representations*, 2018.

N. Carlini and D. Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proceedings of the ACM Workshop on Artificial Intelligence and Security*, pages 3–14, 2017a.

N. Carlini and D. A. Wagner. Defensive distillation is not robust to adversarial examples. *ArXiv*, abs/1607.04311, 2016.

N. Carlini and D. A. Wagner. Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy*, pages 39–57. IEEE Computer Society, 2017b.

N. Carlini, A. Athalye, N. Papernot, W. Brendel, J. Rauber, D. Tsipras, I. Goodfellow, A. Madry, and A. Kurakin. On evaluating adversarial robustness. *arXiv preprint, arXiv:1902.06705*, 2019.

Y. Carmon, A. Raghunathan, L. Schmidt, P. Liang, and J. C. Duchi. Unlabeled data improves adversarial robustness. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pages 11192–11203, 2019.

F. Carrara, R. Becarelli, R. Caldelli, F. Falchi, and G. Amato. Adversarial examples detection in features distance spaces. In *The European Conference on Computer Vision (ECCV) Workshops*, September 2018.

O. Chapelle. Training a support vector machine in the primal. *Neural Comput.*, 19 (5):1155–1178, 2007.

P.-Y. Chen, H. Zhang, Y. Sharma, J. Yi, and C.-J. Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *10th ACM Workshop on Artificial Intelligence and Security*, AISec '17, pages 15–26, New York, NY, USA, 2017. ACM.

Z. Chen and X. Huang. End-to-end learning for lane keeping of self-driving cars. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 1856–1860, June 2017. doi: 10.1109/IVS.2017.7995975.

J. Chenou, G. Hsieh, and T. Fields. Radial basis function network: Its robustness and ability to mitigate adversarial examples. In *Proceedings - 6th Annual Conference on Computational Science and Computational Intelligence, CSCI 2019*, pages 102–106. Institute of Electrical and Electronics Engineers Inc., dec 2019. ISBN 9781728155845. doi: 10.1109/CSCI49370.2019.00024.

G. Ciravegna, F. Giannini, M. Gori, M. Maggini, and S. Melacci. Human-driven fol explanations of deep learning. In C. Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 2234–2240. International Joint Conferences on Artificial Intelligence Organization, 7 2020. doi: 10.24963/ijcai.2020/309. Main track.

M. Claesen, F. D. Smet, J. A. K. Suykens, and B. D. Moor. Fast prediction with svm models containing rbf kernels. *ArXiv*, abs/1403.0736, 2014.

J. Cohen, E. Rosenfeld, and Z. Kolter. Certified adversarial robustness via randomized smoothing. In *International Conference on Machine Learning*, pages 1310–1320. PMLR, 2019.

F. Crecchi, D. Bacciu, and B. Biggio. Detecting adversarial examples through nonlinear dimensionality reduction. In *ESANN '19*, 2019.

F. Crecchi, M. Melis, A. Sotgiu, D. Bacciu, and B. Biggio. Fader: Fast adversarial example rejection. *Neurocomputing*, 470:257–268, 2022. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2021.10.082. URL https://www.sciencedirect.com/science/article/pii/S0925231221015708.

F. Croce and M. Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In *International Conference on Machine Learning*, pages 1–12, 2020a.

F. Croce and M. Hein. Minimally distorted adversarial examples with a fast adaptive boundary attack. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pages 2196–2205. PMLR, 13–18 Jul 2020b.

N. Dalvi, P. Domingos, Mausam, S. Sanghai, and D. Verma. Adversarial classification. In *Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 99–108, Seattle, 2004.

A. S. d'Avila Garcez, M. Gori, L. C. Lamb, L. Serafini, M. Spranger, and S. N. Tran. Neural-symbolic computing: An effective methodology for principled integration of machine learning and reasoning. *Journal of Applied Logics - IfCoLog Journal*, 6(4):611–632, 2019.

L. De Alfaro. Neural Networks with Structural Resistance to Adversarial Attacks. Technical report, 2018. URL https://github.com/lucadealfaro/rbfi.

A. Demontis, M. Melis, B. Biggio, G. Fumera, and F. Roli. Super-sparse learning in similarity spaces. *IEEE Computational Intelligence Magazine*, 11(4):36–45, Nov 2016.

A. Demontis, M. Melis, B. Biggio, D. Maiorca, D. Arp, K. Rieck, I. Corona, G. Giacinto, and F. Roli. Yes, machine learning can be more secure! a case study on android malware detection. *IEEE Transactions on Dependable and Secure Computing*, 16(4):711–724, July 2019. ISSN 1545-5971. doi: 10.1109/TDSC.2017.2700270.

A. Demontis, M. Melis, M. Pintor, M. Jagielski, B. Biggio, A. Oprea, C. Nita-Rotaru, and F. Roli. Why do adversarial attacks transfer? Explaining transferability of evasion and poisoning attacks. In *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, 2019.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

L. Deng and X. Li. Machine Learning Paradigms for Speech Recognition: An Overview. *IEEE Transactions on Audio, Speech, and Language Processing*, 21 (5):1060–1089, May 2013. doi: 10.1109/TASL.2013.2244083.

G. S. Dhillon, K. Azizzadenesheli, J. D. Bernstein, J. Kossaifi, A. Khanna, Z. C. Lipton, and A. Anandkumar. Stochastic activation pruning for robust adversarial defense. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=H1uR4GZRZ.

M. Diligenti, M. Gori, and C. Sacca. Semantic-based regularization for learning and inference. *Artificial Intelligence*, 244:143–165, 2017.

I. Donadello, L. Serafini, and A. D. Garcez. Logic tensor networks for semantic image interpretation. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, IJCAI'17, page 1596–1602. AAAI Press, 2017. ISBN 9780999241103.

J. Duchi, S. Shalev-Shwartz, Y. Singer, and T. Chandra. Efficient projections onto the l1-ball for learning in high dimensions. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 272–279, New York, NY, USA, 2008. ACM.

R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley-Interscience Publication, 2000.

K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song. Robust physical-world attacks on deep learning visual classification. In *CVPR*, pages 1625–1634. IEEE Computer Society, 2018.

R. Feinman, R. R. Curtin, S. Shintre, and A. B. Gardner. Detecting adversarial samples from artifacts. *arXiv preprint arXiv:1703.00410*, 2017.

P. Fogla, M. Sharif, R. Perdisci, O. Kolesnikov, and W. Lee. Polymorphic blending attacks. In *USENIX-SS'06: Proceedings of the 15th conference on USENIX Security Symposium*, pages 241–256, Berkeley, CA, USA, 2006. USENIX Association.

Y. Geifman and R. El-Yaniv. Selective classification for deep neural networks. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4878–4887. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/7073-selective-classification-for-deep-neural-networks.pdf.

Y. Geifman and R. El-Yaniv. Selectivenet: A deep neural network with an integrated reject option. In *International Conference on Machine Learning*, pages 2151–2159, 2019.

A. Globerson and S. T. Roweis. Nightmare at test time: robust learning by feature deletion. In W. W. Cohen and A. Moore, editors, *Proceedings of the 23rd International Conference on Machine Learning*, volume 148, pages 353–360. ACM, 2006.

G. Gnecco, M. Gori, S. Melacci, and M. Sanguineti. Foundations of support constraint machines. *Neural computation*, 27(2):388–480, 2015.

I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015.

M. Gori and S. Melacci. Constraint verification with kernel machines. *IEEE Transactions on Neural Networks and Learning Systems*, 24(5):825–831, 2013.

K. Grosse, P. Manoharan, N. Papernot, M. Backes, and P. McDaniel. On the (statistical) detection of adversarial examples. *arXiv preprint, arXiv:1702.06280*, 2017.

C. Guo, M. Rana, M. Cisse, and L. van der Maaten. Countering adversarial images using input transformations. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=SyJ7ClWCb.

P. Habib Zadeh, R. Hosseini, and S. Sra. Deep-RBF Networks Revisited: Robust Classification with Rejection. Technical report, 2019.

K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

M. Hein, M. Andriushchenko, and J. Bitterwolf. Why relu networks yield high-confidence predictions far away from the training data and how to mitigate the problem. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 41–50, 2019.

D. Hendrycks and K. Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. In *International Conference on Learning Representations*, 2017a. URL https://openreview.net/forum?id=Hkg4TI9xl.

D. Hendrycks and K. Gimpel. Early methods for detecting adversarial images. In *International Conference on Learning Representations*, 2017b. URL https://openreview.net/forum?id=B1dexpDug.

G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

L. Huang, A. D. Joseph, B. Nelson, B. Rubinstein, and J. D. Tygar. Adversarial machine learning. In *4th ACM Workshop on Artificial Intelligence and Security (AISec 2011)*, pages 43–57, Chicago, IL, USA, 2011.

M. Jagielski, A. Oprea, B. Biggio, C. Liu, C. Nita-Rotaru, and B. Li. Manipulating machine learning: Poisoning attacks and countermeasures for regression learning. In *IEEE Symposium on Security and Privacy*, SP '18, pages 931–947. IEEE CS, 2018. doi: 10.1109/SP.2018.00057. URL doi.ieeecomputersociety.org/10.1109/SP.2018.00057.

A. Jalal, A. Ilyas, C. Daskalakis, and A. G. Dimakis. The robust manifold defense: Adversarial training using generative models. *arXiv e-prints*, pages arXiv–1712, 2017.

A. D. Joseph, B. Nelson, B. I. P. Rubinstein, and J. Tygar. *Adversarial Machine Learning*. Cambridge University Press, 2018.

A. Joshi, A. Mukherjee, S. Sarkar, and C. Hegde. Semantic adversarial attacks: Parametric transformations that fool deep classifiers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4773–4783, 2019.

A. Kantchelian, J. D. Tygar, and A. D. Joseph. Evasion and hardening of tree ensemble classifiers. In *33rd ICML*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 2387–2396. JMLR.org, 2016.

N. Kato, Z. M. Fadlullah, B. Mao, F. Tang, O. Akashi, T. Inoue, and K. Mizutani. The Deep Learning Vision for Heterogeneous Network Traffic Control: Proposal, Challenges, and Future Perspective. *IEEE Wireless Communications*, 24(3):146–153, June 2017. ISSN 1536-1284. doi: 10.1109/MWC.2016.1600317WC.

E. P. Klement, R. Mesiar, and E. Pap. *Triangular norms*, volume 8. Springer Science & Business Media, 2013.

A. Krizhevsky. One weird trick for parallelizing convolutional neural networks. *ArXiv*, abs/1404.5997, 2014.

A. Kurakin, I. J. Goodfellow, and S. Bengio. Adversarial machine learning at scale. In *ICLR*, 2017. URL https://arxiv.org/abs/1611.01236.

A. Lamb, J. Binas, A. Goyal, D. Serdyuk, S. Subramanian, I. Mitliagkas, and Y. Bengio. Fortified Networks: Improving the Robustness of Deep Networks by Modeling the Manifold of Hidden Representations. 2018. URL http://arxiv.org/abs/1804.02485.

X. Li and F. Li. Adversarial examples detection in deep networks with convolutional filter statistics. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 5775–5783, 2017. doi: 10.1109/ICCV.2017.615.

D. Lowd and C. Meek. Adversarial learning. In *Proc. 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 641–647, Chicago, IL, USA, 2005a. ACM Press.

D. Lowd and C. Meek. Good word attacks on statistical spam filters. In *Second Conference on Email and Anti-Spam (CEAS)*, Mountain View, CA, USA, 2005b.

J. Lu, T. Issaranon, and D. Forsyth. Safetynet: Detecting and rejecting adversarial examples robustly. In *The IEEE International Conference on Computer Vision (ICCV)*, 2017.

X. Ma, B. Li, Y. Wang, S. M. Erfani, S. Wijewickrema, G. Schoenebeck, M. E. Houle, D. Song, and J. Bailey. Characterizing adversarial subspaces using local intrinsic dimensionality. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=B1gJ1L2aW.

A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJzIBfZAb.

S. Mei and X. Zhu. Using machine teaching to identify optimal training-set attacks on machine learners. In *29th AAAI Conf. Artificial Intelligence (AAAI '15)*, 2015.

S. Melacci and M. Belkin. Laplacian Support Vector Machines Trained in the Primal. *Journal of Machine Learning Research*, 12:1149–1184, March 2011. ISSN 1532-4435.

S. Melacci, A. Globo, and L. Rigutini. Enhancing modern supervised word sense disambiguation models by semantic lexical resources. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

S. Melacci, G. Ciravegna, A. Sotgiu, A. Demontis, B. Biggio, M. Gori, and F. Roli. Domain knowledge alleviates adversarial attacks in multi-label classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2021. doi: 10.1109/TPAMI.2021.3137564.

M. Melis, A. Demontis, B. Biggio, G. Brown, G. Fumera, and F. Roli. Is deep learning safe for robot vision? Adversarial examples against the iCub humanoid. In *ICCVW Vision in Practice on Autonomous Robots (ViPAR)*, pages 751–759. IEEE, 2017.

M. Melis, D. Maiorca, B. Biggio, G. Giacinto, and F. Roli. Explaining black-box android malware detection. In *26th European Signal Processing Conf.*, EUSIPCO, pages 524–528, Rome, Italy, 2018. IEEE, IEEE.

D. Meng and H. Chen. MagNet: a two-pronged defense against adversarial examples. In *24th ACM Conf. Computer and Comm. Sec. (CCS)*, 2017.

J. H. Metzen, T. Genewein, V. Fischer, and B. Bischoff. On detecting adversarial perturbations. In *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, 2017.

D. J. Miller, Z. Xiang, and G. Kesidis. Adversarial learning targeting deep neural network classification: A comprehensive review of defenses against attacks. *Proceedings of the IEEE*, 108(3):402–433, 2020.

T. Miyato, S.-i. Maeda, M. Koyama, K. Nakae, and S. Ishii. Distributional smoothing with virtual adversarial training. In *International Conference on Learning Representations*, 2016. URL https://arxiv.org/pdf/1507.00677.pdf.

T. Miyato, S.-i. Maeda, M. Koyama, and S. Ishii. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(8):1979–1993, 2018.

S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 2574–2582, 2016.

P. Morgado and N. Vasconcelos. Semantically consistent regularization for zero-shot recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6060–6069, 2017.

V. Munusamy Kabilan, B. Morris, and A. Nguyen. Vectordefense: Vectorization as a defense to adversarial examples. 2018. arXiv preprint arXiv:1804.08529.

A. Najafi, S.-i. Maeda, M. Koyama, and T. Miyato. Robustness to adversarial perturbations in learning from incomplete data. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

M. M. Naseer, S. H. Khan, M. H. Khan, F. Shahbaz Khan, and F. Porikli. Cross-domain transferability of adversarial perturbations. *Neural Information Processing Systems*, 32, 2019.

A. B. Nassif, I. Shahin, I. Attili, M. Azzeh, and K. Shaalan. Speech Recognition Using Deep Neural Networks: A Systematic Review. *IEEE Access*, 7:19143–19165, 2019. doi: 10.1109/ACCESS.2019.2896880.

B. Nelson, M. Barreno, F. J. Chi, A. D. Joseph, B. I. P. Rubinstein, U. Saini, C. Sutton, J. D. Tygar, and K. Xia. Exploiting machine learning to subvert your spam filter. In *LEET'08: Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, pages 1–9, Berkeley, CA, USA, 2008. USENIX Association.

J. Newsome, B. Karp, and D. Song. Paragraph: Thwarting signature learning by training maliciously. In *Recent Advances in Intrusion Detection*, LNCS, pages 81–105. Springer, 2006.

A. M. Nguyen, J. Yosinski, and J. Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 427–436. IEEE, 2015.

T. Pang, C. Du, and J. Zhu. Max-mahalanobis linear discriminant analysis networks. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 4013–4022, 2018. URL http://proceedings.mlr.press/v80/pang18a.html.

N. Papernot and P. Mcdaniel. Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning. *ArXiv*, abs/1803.04765, 2018.

N. Papernot, P. McDaniel, and I. Goodfellow. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv preprint, arXiv:1605.07277*, 2016a.

N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 582–597, May 2016b. doi: 10.1109/SP.2016.41.

N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '17, pages 506–519, New York, NY, USA, 2017. ACM.

D. Park, H. Khan, A. Khan, A. Gittens, and B. Yener. Output randomization: A novel defense for both white-box and black-box adversarial models. *ArXiv*, abs/2107.03806, 2021.

S. Park, J. Park, S.-J. Shin, and I.-C. Moon. Adversarial dropout for supervised and semi-supervised learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, volume 32, Apr. 2018.

T. Pi, X. Li, and Z. M. Zhang. Boosted zero-shot learning with semantic correlation regularization. In *Proceedings of the Twenty-Sixth International Joint Conference*

*on Artificial Intelligence, IJCAI-17*, pages 2599–2605, 2017. doi: 10.24963/ijcai.2017/362. URL https://doi.org/10.24963/ijcai.2017/362.

I. Pillai, G. Fumera, and F. Roli. Multi-label classification with a reject option. *Pattern Recognition*, 46(8):2256 – 2266, 2013.

M. Pintor, D. Angioni, A. Sotgiu, L. Demetrio, A. Demontis, B. Biggio, and F. Roli. Imagenet-patch: A dataset for benchmarking machine learning robustness against adversarial patches. *Pattern Recognition*, page 109064, 2022a. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2022.109064. URL https://www.sciencedirect.com/science/article/pii/S0031320322005441.

M. Pintor, L. Demetrio, A. Sotgiu, A. Demontis, N. Carlini, B. Biggio, and F. Roli. Indicators of attack failure: Debugging and improving optimization of adversarial examples. *Advances in Neural Information Processing Systems*, 35, 2022b.

M. Pintor, L. Demetrio, A. Sotgiu, M. Melis, A. Demontis, and B. Biggio. secml: Secure and explainable machine learning in python. *SoftwareX*, 18:101095, 2022c. ISSN 2352-7110. doi: https://doi.org/10.1016/j.softx.2022.101095. URL https://www.sciencedirect.com/science/article/pii/S2352711022000656.

N. Popovic, D. P. Paudel, T. Probst, and L. V. Gool. Gradient obfuscation checklist test gives a false sense of security. *ArXiv*, abs/2206.01705, 2022.

A. Prakash, N. Moran, S. Garber, A. DiLillo, and J. A. Storer. Deflecting adversarial attacks with pixel deflection. In *CVPR*, 2018.

E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. Nicholas. Malware detection by eating a whole exe. In *AAAI Workshop on Artificial Intelligence for Cyber Security*, 2018.

Y. A. U. Rehman, L. M. Po, and M. Liu. Deep learning for face anti-spoofing: An end-to-end approach. In *2017 Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA)*, pages 195–200, Sept. 2017. doi: 10.23919/SPA.2017.8166863.

J. Rony, L. G. Hafemann, L. S. Oliveira, I. B. Ayed, R. Sabourin, and E. Granger. Decoupling direction and norm for efficient gradient-based l2 adversarial attacks and defenses. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4322–4330, 2019.

A. S. Ross and F. Doshi-Velez. Improving the adversarial robustness and interpretability of deep neural networks by regularizing their input gradients. In *AAAI*. AAAI Press, 2018.

S. Rota Bulò, B. Biggio, I. Pillai, M. Pelillo, and F. Roli. Randomized prediction games for adversarial machine learning. *IEEE Transactions on Neural Networks and Learning Systems*, 28(11):2466–2478, 2017.

P. Russu, A. Demontis, B. Biggio, G. Fumera, and F. Roli. Secure kernel machines against evasion attacks. In *9th ACM Workshop on Artificial Intelligence and Security*, AISec '16, pages 59–69, New York, NY, USA, 2016. ACM.

P. Samangouei, M. Kabkab, and R. Chellappa. Defense-GAN: Protecting classifiers against adversarial attacks using generative models. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=BkJ3ibb0-.

C. Saunders, A. Gammerman, and V. Vovk. Transduction with confidence and credibility. In *IJCAI International Joint Conference on Artificial Intelligence*, 1999.

W. Scheirer, L. Jain, and T. Boult. Probability models for open set recognition. *IEEE Trans. Patt. An. Mach. Intell.*, 36(11):2317–2324, 2014.

W. J. Scheirer, A. Rocha, R. Michaels, and T. E. Boult. Meta-recognition: The theory and practice of recognition score analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 33:1689–1695, 2011.

L. Schmidt, S. Santurkar, D. Tsipras, K. Talwar, and A. Madry. Adversarially robust generalization requires more data. 31, 2018. URL https://proceedings.neurips.cc/paper/2018/file/f708f064faaf32a43e4d3c784e6af9ea-Paper.pdf.

L. Schott, J. Rauber, M. Bethge, and W. Brendel. Towards the first adversarially robust neural network model on mnist. In *International Conference on Learning Representations*, 2018.

A. Shafahi, W. R. Huang, C. Studer, S. Feizi, and T. Goldstein. Are adversarial examples inevitable? In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=r1lWUoA9FQ.

G. Shafer and V. Vovk. A tutorial on conformal prediction. *Journal of Machine Learning Research*, 2008. ISSN 15324435.

M. Sharif, S. Bhagavatula, L. Bauer, and M. K. Reiter. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1528–1540. ACM, 2016.

S. Shen, G. Jin, K. Gao, and Y. Zhang. Ape-gan: Adversarial perturbation elimination with gan. *arXiv preprint arXiv:1707.05474*, 2017.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 2016. ISSN 14764687. doi: 10.1038/nature16961.

C.-J. Simon-Gabriel, Y. Ollivier, L. Bottou, B. Schölkopf, and D. Lopez-Paz. First-order adversarial vulnerability of neural networks and input dimension. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5809–5817. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/simon-gabriel19a.html.

Q. Song, H. Jin, X. Huang, and X. Hu. Multi-label adversarial perturbations. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 1242–1247, 2018.

Y. Song, T. Kim, S. Nowozin, S. Ermon, and N. Kushman. Pixeldefend: Leveraging generative models to understand and defend against adversarial examples. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJUYGxbCW.

A. Sotgiu, A. Demontis, M. Melis, B. Biggio, G. Fumera, X. Feng, and F. Roli. Deep neural rejection against adversarial examples. *EURASIP Journal on Information Security*, 2020:1–10, 2020.

A. Sotgiu, M. Pintor, and B. Biggio. Explainability-based debugging of machine learning for vulnerability discovery. In *Proceedings of the 17th International Conference on Availability, Reliability and Security*, ARES '22, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450396707. doi: 10.1145/3538969.3543809. URL https://doi.org/10.1145/3538969.3543809.

I. Steinwart. Sparseness of support vector machines. *J. Mach. Learn. Res.*, 4(11): 1071–1105, 2003.

C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014. URL http://arxiv.org/abs/1312.6199.

S. Teso. Does symbolic knowledge prevent adversarial fooling? *arXiv preprint, arXiv:1912.10834*, 2019.

S. Thulasidasan, T. Bhattacharya, J. Bilmes, G. Chennupati, and J. Mohd-Yusof. Knows when it doesn't know: Deep abstaining classifiers. 2019. URL https://openreview.net/forum?id=rJxF73R9tX.

F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. Stealing machine learning models via prediction apis. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 601–618, Austin, TX, 2016. USENIX Association. ISBN 978-1-931971-32-4.

F. Tramer, N. Carlini, W. Brendel, and A. Madry. On adaptive attacks to adversarial example defenses. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1633–1645. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/11f38f8ecd71867b42433548d1078e38-Paper.pdf.

L. Van Der Maaten and G. Hinton. Visualizing Data using t-SNE. Technical report, 2008. URL http://www.jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

V. Vovk, A. Gammerman, and C. Saunders. Machine-learning applications of algorithmic randomness. In *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, page 444–453, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. ISBN 1558606122.

N. Šrndic and P. Laskov. Practical evasion of a learning-based classifier: A case study. In *Proc. 2014 IEEE Symp. Security and Privacy*, SP '14, pages 197–211, Washington, DC, USA, 2014. IEEE CS.

P. H. Winston and B. K. Horn. *Lisp.* Addison Wesley Pub., Reading, MA, 1986.

G. L. Wittel and S. F. Wu. On attacking statistical spam filters. In *First Conference on Email and Anti-Spam (CEAS)*, Mountain View, CA, USA, 2004.

D. H. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.

E. Wong, L. Rice, and J. Z. Kolter. Fast is better than free: Revisiting adversarial training. In *International Conference on Learning Representations*, 2019.

Y. Wu, D. Bamman, and S. Russell. Adversarial training for relation extraction. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1778–1783, 2017.

C. Xiao, P. Zhong, and C. Zheng. Enhancing adversarial defense by k-winners-take-all. In *8th International Conference on Learning Representations*, 2020.

H. Xiao, B. Biggio, G. Brown, G. Fumera, C. Eckert, and F. Roli. Is feature selection secure against training data poisoning? In F. Bach and D. Blei, editors, *JMLR W&CP - Proc. 32nd Int'l Conf. Mach. Learning (ICML)*, volume 37, pages 1689–1698, 2015.

C. Xie, J. Wang, Z. Zhang, Z. Ren, and A. Yuille. Mitigating adversarial effects through randomization. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=Sk9yuql0Z.

X. Yin, S. Kolouri, and G. Rohde. Gat: Generative adversarial training for adversarial example detection and robust classification. In *ICLR*, 2020.

M. Yu and M. Dredze. Improving lexical embeddings with semantic knowledge. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 545–550, 2014.

T. Yu, S. Hu, C. Guo, W. Chao, and K. Weinberger. A new defense against adversarial images: Turning a weakness into a strength. In *Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS 2019)*, Oct. 2019.

H. Zhang, H. Chen, C. Xiao, S. Gowal, R. Stanforth, B. Li, D. Boning, and C.-J. Hsieh. Towards stable and efficient training of verifiably robust neural networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=Skxuk1rFwB.

# Appendix A

# Adversarial Examples

In order to better support our experimental analysis, we show here several adversarial examples generated against the considered defenses during their evaluation. In particular, for the evaluation in Section 6.1.4, we report some samples generated by the APGD-CE algorithm of the AutoAttack (Croce and Hein, 2020a) library, ANIMALS dataset. In Figure A.1 and Figure A.2, we plot the two adversarial examples with the highest supervision loss (and low constraint loss) and with the highest constraint loss (and low supervision loss) (see also Figure 6.7 of the thesis). No evident visual pattern is noticeable to distinguish the two cases.

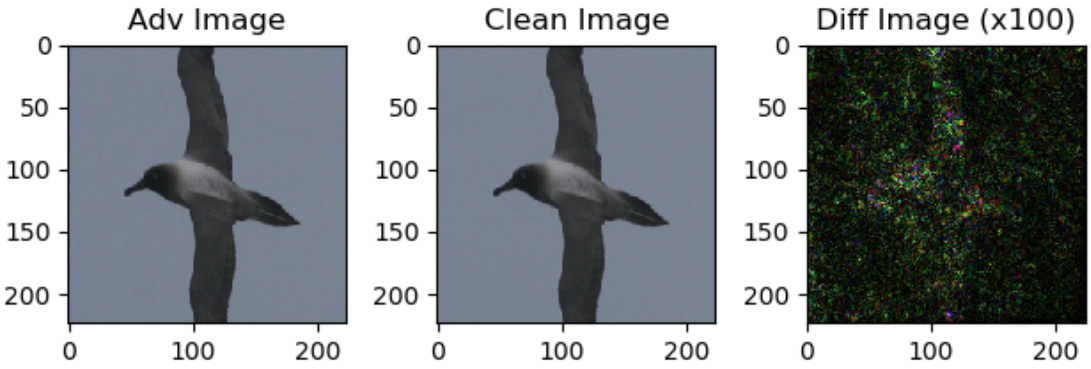


Figure A.1: Adversarial examples with highest supervision loss (low constraint loss), APGD-CE attack, ANIMALS dataset.

In Figure A.3 and Figure A.4, we show some adversarial examples computed respectively on the MNIST and CIFAR10 datasets during the evaluation in Section 6.2. Finally, we report in Figure aA.5 few ImageNet samples obtained using the two considered attack algorithms for evaluation in Section 6.3 against the different detectors, together with the relative induced adversarial perturbation: PGD-LS tends to produce finer perturbation than standard PGD attack, yet being effective (Table 6.16).

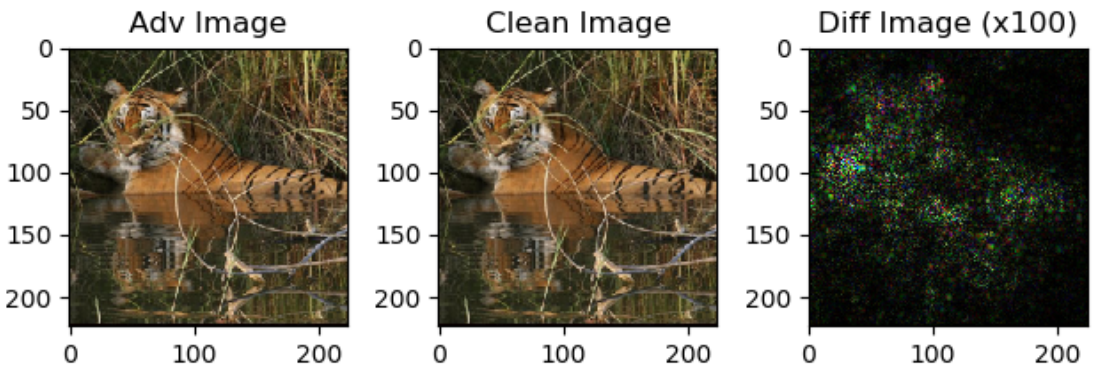


Figure A.2: Adversarial examples with highest constraint loss (low supervision loss), APGD-CE attack, ANIMALS dataset.

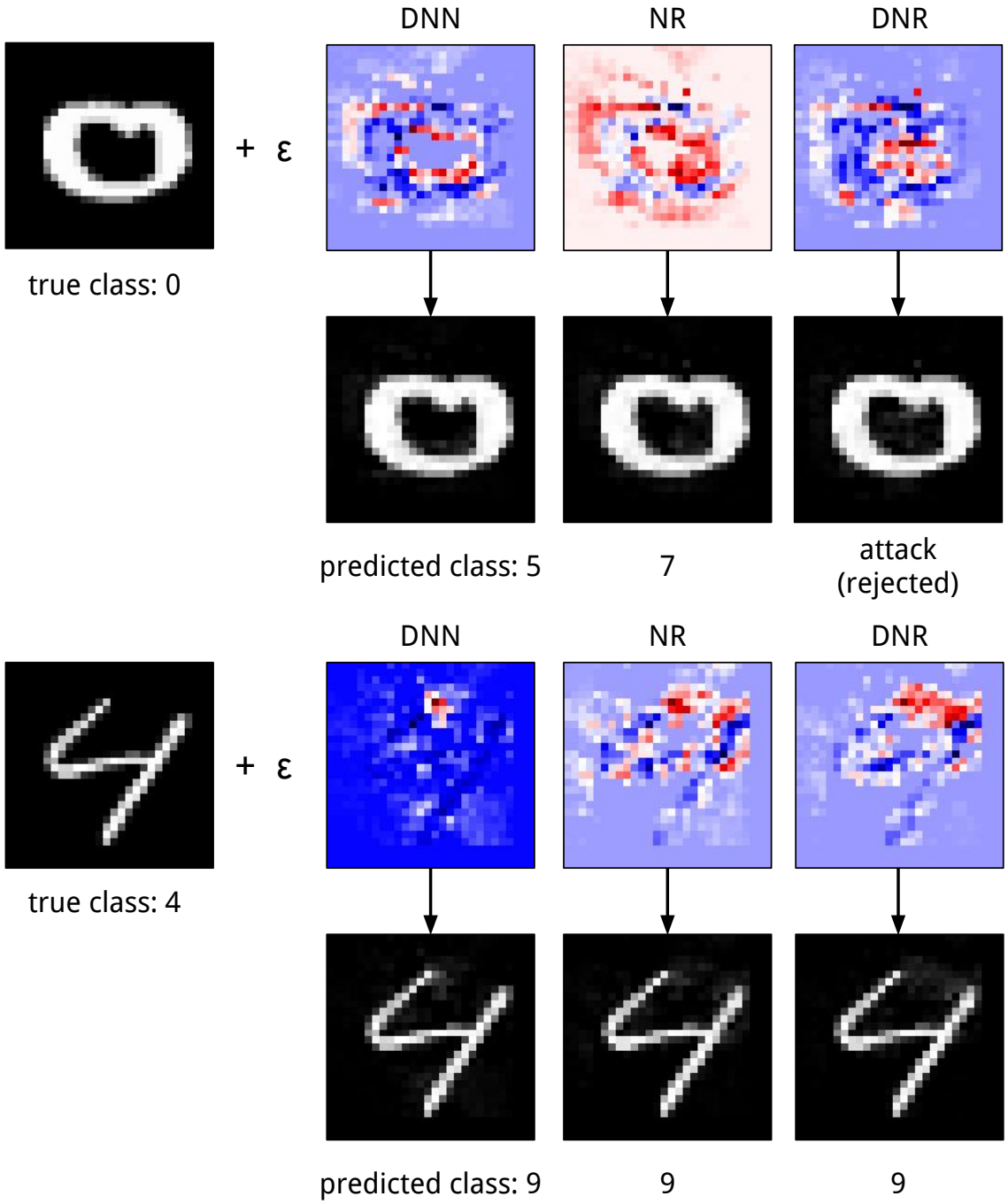


Figure A.3: Adversarial examples computed on the MNIST data to evade the undefended DNN, NR, and DNR. The source image is reported on the left, followed by the (magnified) adversarial perturbation crafted with $\varepsilon = 1$ against each classifier and the resulting adversarial examples. We remind the reader that the attacks considered in this work are untargeted, i.e., they succeed when the attack sample is not correctly assigned to its true class.

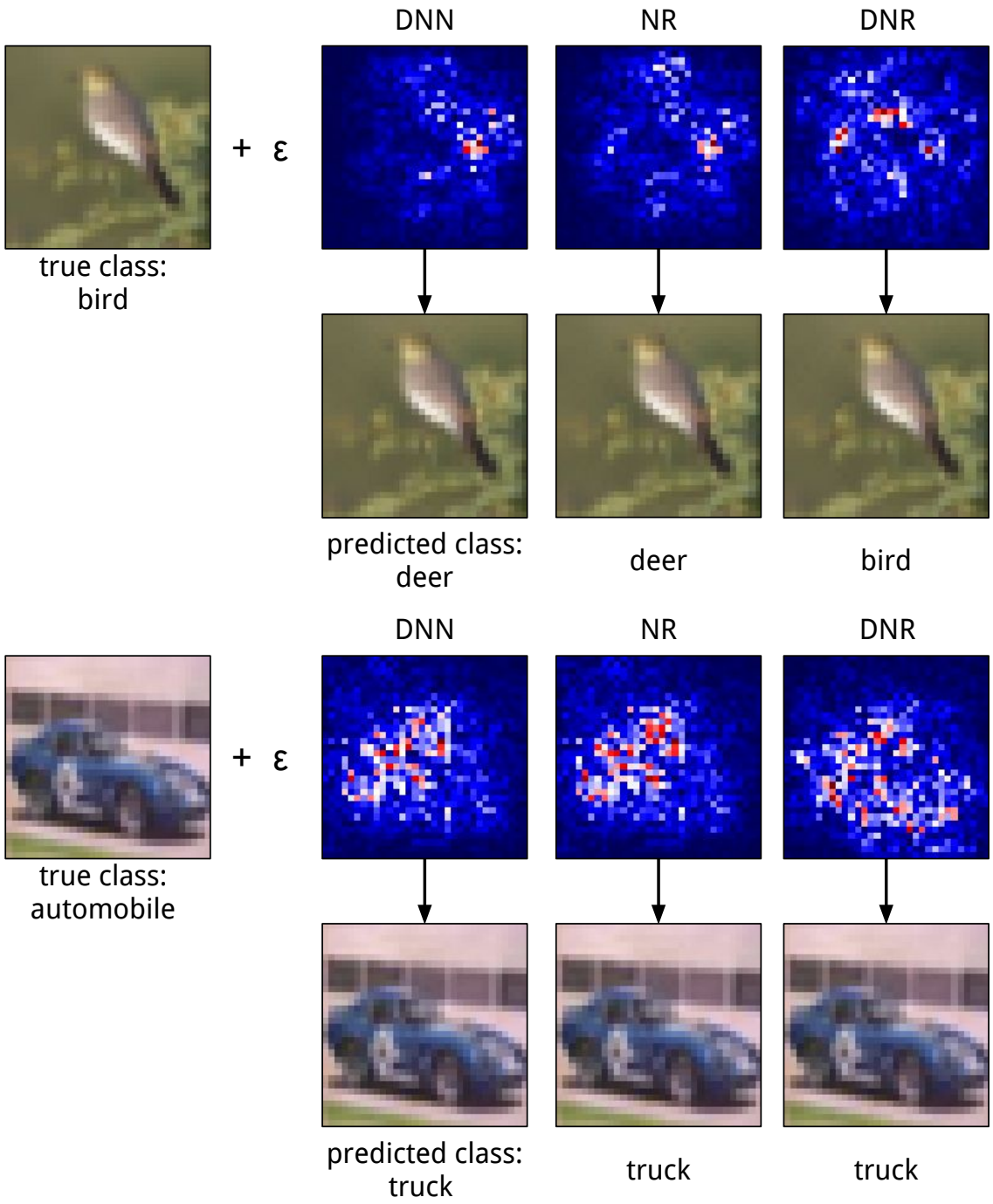


Figure A.4: Adversarial examples computed on the CIFAR10 dataset adding a perturbation computed with $\varepsilon = 0.2$. See the caption of FigureA.3 for further details.

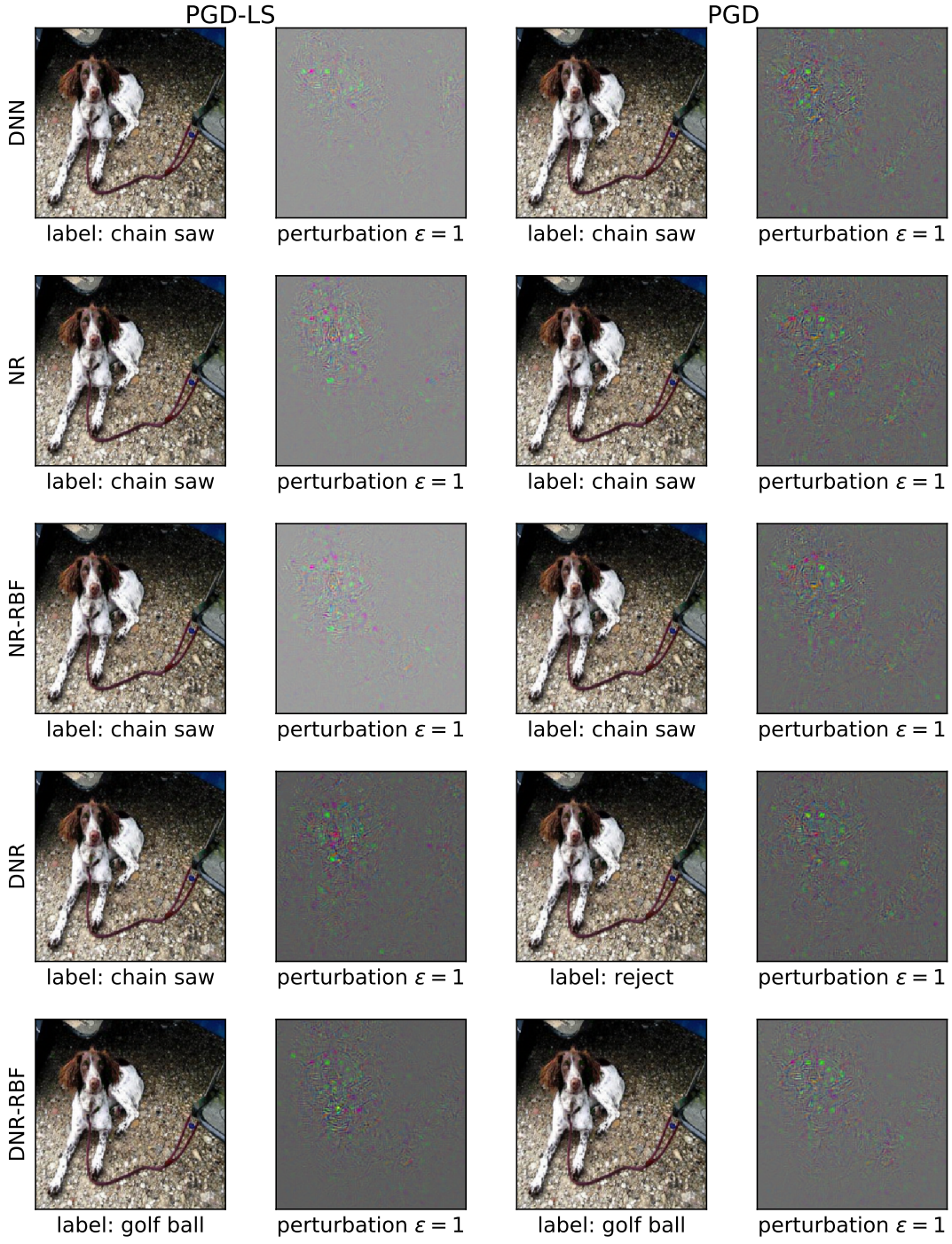


Figure A.5: Perturbed samples from the ImageNet10 dataset produced by attacking each classifier using PGD-LS (left columns) and PGD (right columns) algorithms. The maximum size of the $\ell_2$ perturbation is equally set to $\varepsilon = 1$.

# Appendix B

# Domain Knowledge

We report here the complete list of the domain-knowledge constraints available for the considered datasets, for the approach described in Chapter 3.

Each dataset is composed of a set of classes that, for convenience, we associate with logic predicates. Such predicates participate in First-Order Logic (FOL) formulas that model the available domain knowledge. The FOL formulas that define the domain knowledge of the ANIMALS, CIFAR-100, and PASCAL-Part data are reported in Table B.1, Table B.2, and Table B.4, respectively, where each predicate is indicated with capital letters. In each table (bottom part) we also report those rules that are about activating at least one of the classes of each level of the hierarchy. Following the nomenclature used in the thesis, the main classes of the ANIMALS dataset are ALBATROSS, GIRAFFE, CHEETAH, OSTRICH, PENGUIN, TIGER, ZEBRA, while the other categories are MAMMAL, HAIR, MILK, FEATHERS, BIRD, FLY, LAYEGGS, MEAT, CARNIVORE, POINTEDTEETH, CLAWS, FORWARDEYS, HOOFS, UNGULATE, CUD, EVENTOED, TAWNY, BLACKSTRIPES, LONGLEGS, LONGNECK, DARKSPOTS, WHITE, BLACK, SWIM, BLACKWHITE, GOODFLIER. In the case of the CIFAR-100 dataset, the main classes are the ones associated with the predicates of Table B.2 that belong to the premises of the shortest FOL formulas (i.e., the formulas in the form $\text{A}(x) \Rightarrow \text{B}(x)$, where the main class is A). Formulas in PASCAL-Part are relationships between objects and object parts. The same part can belong to multiple objects, and in each object, several parts might be visible. See Table B.4 for the list of classes (the main classes are in the premises of the second block of formulas).

In ANIMALS and CIFAR-100, a mutual exclusion predicate is imposed on the main classes. As a matter of fact, in these two datasets, each image is only about a single main class. The `mutual_excl(p_1, p_2, ..., p_n)` predicate defined below can be devised in different ways. The first, straightforward approach consists in

considering the disjunction of the true cases in the truth table of the predicate:

$$mutual\_excl(p_1, p_2, ..., p_n) = \bigvee_{i=0}^{n} \left( p_i(x) \wedge \bigwedge_{j=0, j\neq i}^{n} \neg p_j(x) \right), \quad i, j \in M, \tag{B.1}$$

where $M$ is the set of the main classes, with cardinality $n$ and $p_i(x)$ is the logic predicate corresponding to the $i$-th output of the network $f_i(x)$. This formulation of the `mutual_excl` predicate is what we used in the ANIMALS dataset. When there are several classes, as in CIFAR-100, this formulation leads to optimization issues since it turned out to be complicated to find a good balance between the effect of this constraint and the supervision-fitting term. For this reason, the mutual exclusivity in CIFAR-100 was defined as a disjunction of the main classes followed by a set of implications that are used to implement the mutual exclusion of the predicates,

$$mutual\_excl(p_1, p_2, ..., p_n) = \begin{cases} \bigvee_{i=0}^{n} p_i(x), \\ p_i(x) \Rightarrow \bigwedge_{j=0, j\neq i}^{n} \neg p_j(x), \quad \forall i \in M, \end{cases} \tag{B.2}$$

that resulted easier to tune since we have multiple soft constraints that could eventually be violated to accommodate the optimization procedure.

In the case of the ANIMALS dataset, we also considered a noisy setting in which we artificially altered the FOL rules of Table B.1 in order to make them not fully coherent with the (real) domain knowledge. We describe the resulting noisy knowledge bases in Table B.5, Table B.6, and Table B.7, reporting only the changes with respect to Table B.1. The knowledge base of Table B.5 has been obtained by altering four of the existing rules, while knowledge of Table B.6 is the outcome of *adding* four new rules. In both cases, we considered two implications whose conclusions are about main classes and two other implications whose conclusions are about auxiliary classes. Finally, Table B.7 is about a noisy knowledge base where we relaxed the main-class-oriented conclusions of four implications. Such knowledge has been created by manually extending the conclusions using the disjunction operator, thus tolerating multiple configurations of the main classes.

Table B.1: Domain knowledge, ANIMALS dataset.

| |
|---|
| $\forall x$ HAIR($x$) $\Rightarrow$ MAMMAL($x$) |
| $\forall x$ MILK($x$) $\Rightarrow$ MAMMAL($x$) |
| $\forall x$ FEATHER($x$) $\Rightarrow$ BIRD($x$) |
| $\forall x$ FLY($x$) $\wedge$ LAYEGGS($x$) $\Rightarrow$ BIRD($x$) |
| $\forall x$ MAMMAL($x$) $\wedge$ MEAT($x$) $\Rightarrow$ CARNIVORE($x$) |
| $\forall x$ MAMMAL($x$) $\wedge$ POINTEDTEETH($x$) $\wedge$ CLAWS($x$) $\wedge$ FORWARDEYES($x$) $\Rightarrow$ CARNIVORE($x$) |
| $\forall x$ MAMMAL($x$) $\wedge$ HOOFS($x$) $\Rightarrow$ UNGULATE($x$) |
| $\forall x$ MAMMAL($x$) $\wedge$ CUD($x$) $\Rightarrow$ UNGULATE($x$) |
| $\forall x$ MAMMAL($x$) $\wedge$ CUD($x$) $\Rightarrow$ EVENTOED($x$) |
| $\forall x$ CARNIVORE($x$) $\wedge$ TAWNY($x$) $\wedge$ DARKSPOTS($x$) $\Rightarrow$ CHEETAH($x$) |
| $\forall x$ CARNIVORE($x$) $\wedge$ TAWNY($x$) $\wedge$ BLACKSTRIPES($x$) $\Rightarrow$ TIGER($x$) |
| $\forall x$ UNGULATE($x$) $\wedge$ LONGLEGS($x$) $\wedge$ LONGNECK($x$) $\wedge$ TAWNY($x$) $\wedge$ DARKSPOTS($x$) $\Rightarrow$ GIRAFFE($x$) |
| $\forall x$ BLACKSTRIPES($x$) $\wedge$ UNGULATE($x$) $\wedge$ WHITE($x$) $\Rightarrow$ ZEBRA($x$) |
| $\forall x$ BIRD($x$) $\wedge$ $\neg$FLY($x$) $\wedge$ LONGLEGS($x$) $\wedge$ LONGNECK($x$) $\wedge$ BLACK($x$) $\Rightarrow$ OSTRICH($x$) |
| $\forall x$ BIRD($x$) $\wedge$ $\neg$FLY($x$) $\wedge$ SWIM($x$) $\wedge$ BLACKWHITE($x$) $\Rightarrow$ PENGUIN($x$) |
| $\forall x$ BIRD($x$) $\wedge$ GOODFLIER($x$) $\Rightarrow$ ALBATROSS($x$) |
| $\forall x$ `mutual_excl`(ALBATROSS($x$), GIRAFFE($x$), CHEETAH($x$), OSTRICH($x$), PENGUIN($x$), TIGER($x$), ZEBRA($x$)) |
| $\forall x$ MAMMAL($x$) $\vee$ HAIR($x$) $\vee$ MILK($x$) $\vee$ FEATHERS($x$) $\vee$ BIRD($x$) $\vee$ FLY($x$) $\vee$ LAYEGGS($x$) $\vee$ MEAT($x$) $\vee$ CARNIVORE($x$) $\vee$ POINTEDTEETH($x$) $\vee$ CLAWS($x$) $\vee$ FORWARDEYS($x$) $\vee$ HOOFS($x$) $\vee$ UNGULATE($x$) $\vee$ CUD($x$) $\vee$ EVENTOED($x$) $\vee$ TAWNY($x$) $\vee$ BLACKSTRIPES($x$) $\vee$ LONGLEGS($x$) $\vee$ LONGNECK($x$) $\vee$ DARKSPOTS($x$) $\vee$ WHITE($x$) $\vee$ BLACK($x$) $\vee$ SWIM($x$) $\vee$ BLACKWHITE($x$) $\vee$ GOODFLIER($x$) |

Table B.2: Domain knowledge, CIFAR-100 dataset.

$\forall x$ AQUATIC MAMMALS($x$) $\Rightarrow$ (BEAVER($x$) $\vee$ DOLPHIN($x$) $\vee$ OTTER($x$) $\vee$ SEAL($x$) $\vee$ WHALE($x$))
$\forall x$ BEAVER($x$) $\Rightarrow$ AQUATIC MAMMALS($x$)
$\forall x$ DOLPHIN($x$) $\Rightarrow$ AQUATIC MAMMALS($x$)
$\forall x$ OTTER($x$) $\Rightarrow$ AQUATIC MAMMALS($x$)
$\forall x$ SEAL($x$) $\Rightarrow$ AQUATIC MAMMALS($x$)
$\forall x$ WHALE($x$) $\Rightarrow$ AQUATIC MAMMALS($x$)

$\forall x$ FISH($x$) $\Rightarrow$ (AQUARIUM FISH($x$) $\vee$ FLATFISH($x$) $\vee$ RAY($x$) $\vee$ SHARK($x$) $\vee$ TROUT($x$))
$\forall x$ AQUARIUM_FISH($x$) $\Rightarrow$ FISH($x$)
$\forall x$ FLATFISH($x$) $\Rightarrow$ FISH($x$)
$\forall x$ RAY($x$) $\Rightarrow$ FISH($x$)
$\forall x$ SHARK($x$) $\Rightarrow$ FISH ($x$)
$\forall x$ TROUT($x$) $\Rightarrow$ FISH($x$)

$\forall x$ FLOWERS($x$) $\Rightarrow$ (ORCHID($x$) $\vee$ POPPY($x$) $\vee$ ROSE($x$) $\vee$ SUNFLOWER($x$) $\vee$ TULIP($x$))
$\forall x$ ORCHID($x$) $\Rightarrow$ FLOWERS($x$)
$\forall x$ POPPY($x$) $\Rightarrow$ FLOWERS($x$)
$\forall x$ ROSE($x$) $\Rightarrow$ FLOWERS($x$)
$\forall x$ SUNFLOWER($x$) $\Rightarrow$ FLOWERS($x$)
$\forall x$ TULIP($x$) $\Rightarrow$ FLOWERS($x$)

$\forall x$ FOOD_CONTAINERS($x$) $\Rightarrow$ (BOTTLE($x$) $\vee$ BOWL($x$) $\vee$ CAN($x$) $\vee$ CUP($x$) $\vee$ PLATE($x$))
$\forall x$ BOTTLE($x$) $\Rightarrow$ FOOD_CONTAINERS ($x$)
$\forall x$ BOWL($x$) $\Rightarrow$ FOOD_CONTAINERS ($x$)
$\forall x$ CAN($x$) $\Rightarrow$ FOOD_CONTAINERS ($x$)
$\forall x$ CUP($x$) $\Rightarrow$ FOOD_CONTAINERS ($x$)
$\forall x$ PLATE($x$) $\Rightarrow$ FOOD_CONTAINERS ($x$)

$\forall x$ FRUIT_AND_VEGETABLES($x$) $\Rightarrow$ (APPLE($x$) $\vee$ MUSHROOM($x$) $\vee$ ORANGE($x$) $\vee$ PEAR($x$) $\vee$ SWEET_PEPPER($x$))
$\forall x$ APPLE($x$) $\Rightarrow$ FRUIT_AND_VEGETABLES($x$)
$\forall x$ MUSHROOM($x$) $\Rightarrow$ FRUIT_AND_VEGETABLES($x$)
$\forall x$ ORANGE($x$) $\Rightarrow$ FRUIT_AND_VEGETABLES($x$)
$\forall x$ PEAR($x$) $\Rightarrow$ FRUIT_AND_VEGETABLES($x$)
$\forall x$ SWEET_PEPPER($x$) $\Rightarrow$ FRUIT_AND_VEGETABLES($x$)
$\forall x$ HOUSEHOLD_ELECTRICAL_DEVICES($x$) $\Rightarrow$ (CLOCK($x$) $\vee$ KEYBOARD($x$) $\vee$ LAMP($x$) $\vee$ TELEPHONE($x$) $\vee$ TELEVISION($x$))
$\forall x$ CLOCK($x$) $\Rightarrow$ HOUSEHOLD_ELECTRICAL_DEVICES($x$)
$\forall x$ KEYBOARD($x$) $\Rightarrow$ HOUSEHOLD_ELECTRICAL_DEVICES($x$)
$\forall x$ LAMP($x$) $\Rightarrow$ HOUSEHOLD_ELECTRICAL_DEVICES($x$)
$\forall x$ TELEPHONE($x$) $\Rightarrow$ HOUSEHOLD_ELECTRICAL_DEVICES($x$)
$\forall x$ TELEVISION($x$) $\Rightarrow$ HOUSEHOLD_ELECTRICAL_DEVICES($x$)

$\forall x$ HOUSEHOLD_FURNITURE($x$) $\Rightarrow$ (BED($x$) $\vee$ CHAIR($x$) $\vee$ COUCH($x$) $\vee$ TABLE($x$) $\vee$ WARDROBE($x$))
$\forall x$ BED($x$) $\Rightarrow$ HOUSEHOLD_FURNITURE($x$)
$\forall x$ CHAIR($x$) $\Rightarrow$ HOUSEHOLD_FURNITURE($x$)
$\forall x$ COUCH($x$) $\Rightarrow$ HOUSEHOLD_FURNITURE($x$)
$\forall x$ TABLE($x$) $\Rightarrow$ HOUSEHOLD_FURNITURE($x$)
$\forall x$ WARDROBE($x$) $\Rightarrow$ HOUSEHOLD_FURNITURE($x$)

$\forall x$ INSECTS($x$) $\Rightarrow$ (BEE($x$) $\vee$ BEETLE($x$) $\vee$ BUTTERFLY($x$) $\vee$ CATERPILLAR($x$) $\vee$ COCKROACH($x$))
$\forall x$ BEE($x$) $\Rightarrow$ INSECTS($x$)
$\forall x$ BEETLE($x$) $\Rightarrow$ INSECTS($x$)
$\forall x$ BUTTERFLY($x$) $\Rightarrow$ INSECTS($x$)
$\forall x$ CATERPILLAR($x$) $\Rightarrow$ INSECTS($x$)
$\forall x$ COCKROACH($x$) $\Rightarrow$ INSECTS($x$)

$\forall x$ LARGE_CARNIVORES($x$) $\Rightarrow$ (BEAR($x$) $\vee$ LEOPARD($x$) $\vee$ LION($x$) $\vee$ TIGER ($x$) $\vee$ WOLF($x$))
$\forall x$ BEAR($x$) $\Rightarrow$ LARGE_CARNIVORES($x$)

$\forall x$ LEOPARD($x$) $\Rightarrow$ LARGE_CARNIVORES($x$)
$\forall x$ LION($x$) $\Rightarrow$ LARGE_CARNIVORES($x$)
$\forall x$ TIGER($x$) $\Rightarrow$ LARGE_CARNIVORES($x$)
$\forall x$ WOLF($x$) $\Rightarrow$ LARGE_CARNIVORES($x$)

$\forall x$ LARGE_MAN-MADE_OUTDOOR_THINGS($x$) $\Rightarrow$ (BRIDGE($x$) $\vee$ CASTLE($x$) $\vee$ HOUSE($x$) $\vee$ ROAD($x$) $\vee$ SKYSCRAPER($x$))
$\forall x$ BRIDGE($x$) $\Rightarrow$ LARGE_MAN-MADE_OUTDOOR_THINGS($x$)
$\forall x$ CASTLE($x$) $\Rightarrow$ LARGE_MAN-MADE_OUTDOOR_THINGS($x$)
$\forall x$ HOUSE($x$) $\Rightarrow$ LARGE_MAN-MADE_OUTDOOR_THINGS($x$)
$\forall x$ ROAD($x$) $\Rightarrow$ LARGE_MAN-MADE_OUTDOOR_THINGS($x$)
$\forall x$ SKYSCRAPER($x$) $\Rightarrow$ LARGE_MAN-MADE_OUTDOOR_THINGS($x$)
$\forall x$ LARGE_NATURAL_OUTDOOR_SCENES($x$) $\Rightarrow$ (CLOUD($x$) $\vee$ FOREST($x$) $\vee$ MOUNTAIN($x$) $\vee$ PLAIN($x$) $\vee$ SEA($x$))
$\forall x$ CLOUD($x$) $\Rightarrow$ LARGE_NATURAL_OUTDOOR_SCENES($x$)
$\forall x$ FOREST($x$) $\Rightarrow$ LARGE_NATURAL_OUTDOOR_SCENES($x$)
$\forall x$ MOUNTAIN($x$) $\Rightarrow$ LARGE_NATURAL_OUTDOOR_SCENES($x$)
$\forall x$ PLAIN($x$) $\Rightarrow$ LARGE_NATURAL_OUTDOOR_SCENES($x$)
$\forall x$ SEA($x$) $\Rightarrow$ LARGE_NATURAL_OUTDOOR_SCENES($x$)

$\forall x$ LARGE_OMNIVORES_AND_HERBIVORES($x$) $\Rightarrow$ (CAMEL($x$) $\vee$ CATTLE($x$) $\vee$ CHIMPANZEE($x$) $\vee$ ELEPHANT($x$) $\vee$ KANGAROO($x$))
$\forall x$ CAMEL($x$) $\Rightarrow$ LARGE_OMNIVORES_AND_HERBIVORES($x$)
$\forall x$ CATTLE($x$) $\Rightarrow$ LARGE_OMNIVORES_AND_HERBIVORES($x$)
$\forall x$ CHIMPANZEE($x$) $\Rightarrow$ LARGE_OMNIVORES_AND_HERBIVORES($x$)
$\forall x$ ELEPHANT($x$) $\Rightarrow$ LARGE_OMNIVORES_AND_HERBIVORES($x$)
$\forall x$ KANGAROO($x$) $\Rightarrow$ LARGE_OMNIVORES_AND_HERBIVORES($x$)

$\forall x$ MEDIUM_MAMMALS($x$) $\Rightarrow$ (FOX($x$) $\vee$ PORCUPINE($x$) $\vee$ POSSUM($x$) $\vee$ RACCOON($x$) $\vee$ SKUNK($x$))
$\forall x$ FOX($x$) $\Rightarrow$ MEDIUM_MAMMALS($x$)
$\forall x$ PORCUPINE($x$) $\Rightarrow$ MEDIUM_MAMMALS($x$)
$\forall x$ POSSUM($x$) $\Rightarrow$ MEDIUM_MAMMALS($x$)
$\forall x$ RACCOON($x$) $\Rightarrow$ MEDIUM_MAMMALS($x$)
$\forall x$ SKUNK($x$) $\Rightarrow$ MEDIUM_MAMMALS($x$)

$\forall x$ NON-INSECT_INVERTEBRATES($x$) $\Rightarrow$ (CRAB($x$) $\vee$ LOBSTER($x$) $\vee$ SNAIL($x$) $\vee$ SPIDER($x$) $\vee$ WORM($x$))
$\forall x$ CRAB($x$) $\Rightarrow$ NON-INSECT_INVERTEBRATES($x$)
$\forall x$ LOBSTER($x$) $\Rightarrow$ NON-INSECT_INVERTEBRATES($x$)
$\forall x$ SNAIL($x$) $\Rightarrow$ NON-INSECT_INVERTEBRATES($x$)
$\forall x$ SPIDER($x$) $\Rightarrow$ NON-INSECT_INVERTEBRATES($x$)
$\forall x$ WORM($x$) $\Rightarrow$ NON-INSECT_INVERTEBRATES($x$)

$\forall x$ PEOPLE($x$) $\Rightarrow$ (BABY($x$) $\vee$ MAN($x$) $\vee$ WOMAN($x$) $\vee$ BOY($x$) $\vee$ GIRL($x$))
$\forall x$ BABY($x$) $\Rightarrow$ PEOPLE($x$)
$\forall x$ BOY($x$) $\Rightarrow$ PEOPLE($x$)
$\forall x$ GIRL($x$) $\Rightarrow$ PEOPLE($x$)
$\forall x$ MAN($x$) $\Rightarrow$ PEOPLE($x$)
$\forall x$ WOMAN($x$) $\Rightarrow$ PEOPLE($x$)

$\forall x$ REPTILES($x$) $\Rightarrow$ (CROCODILE($x$) $\vee$ DINOSAUR($x$) $\vee$ LIZARD($x$) $\vee$ SNAKE($x$) $\vee$ TURTLE($x$))
$\forall x$ CROCODILE($x$) $\Rightarrow$ REPTILES($x$)
$\forall x$ DINOSAUR($x$) $\Rightarrow$ REPTILES($x$)
$\forall x$ LIZARD($x$) $\Rightarrow$ REPTILES($x$)
$\forall x$ SNAKE($x$) $\Rightarrow$ REPTILES($x$)
$\forall x$ TURTLE($x$) $\Rightarrow$ REPTILES($x$)

$\forall x$ SMALL_MAMMALS($x$) $\Rightarrow$ (HAMSTER($x$) $\vee$ MOUSE($x$) $\vee$ RABBIT($x$) $\vee$ SHREW($x$) $\vee$ SQUIRREL($x$))
$\forall x$ HAMSTER($x$) $\Rightarrow$ SMALL_MAMMALS($x$)
$\forall x$ MOUSE($x$) $\Rightarrow$ SMALL_MAMMALS($x$)
$\forall x$ RABBIT($x$) $\Rightarrow$ SMALL_MAMMALS($x$)
$\forall x$ SHREW($x$) $\Rightarrow$ SMALL_MAMMALS($x$)
$\forall x$ SQUIRREL($x$) $\Rightarrow$ SMALL_MAMMALS($x$)

$\forall x$ TREES($x$) $\Rightarrow$ (MAPLE_TREE($x$) $\vee$ OAK_TREE($x$) $\vee$ PALM_TREE($x$) $\vee$ PINE_TREE($x$) $\vee$ WILLOW_TREE($x$))

$\forall x$ MAPLE_TREE($x$) $\Rightarrow$ TREES($x$)
$\forall x$ OAK_TREE($x$) $\Rightarrow$ TREES($x$)
$\forall x$ PALM_TREE($x$) $\Rightarrow$ TREES($x$)
$\forall x$ PINE_TREE($x$) $\Rightarrow$ TREES($x$)
$\forall x$ WILLOW_TREE($x$) $\Rightarrow$ TREE($x$)

$\forall x$ VEHICLES1($x$) $\Rightarrow$ (BIKE($x$) $\vee$ BUS($x$) $\vee$ MOTORBIKE($x$) $\vee$ PICKUP_TRUCK($x$) $\vee$ TRAIN($x$))
$\forall x$ BIKE($x$) $\Rightarrow$ VEHICLES1($x$)
$\forall x$ BUS($x$) $\Rightarrow$ VEHICLES1($x$)
$\forall x$ MOTORBIKE($x$) $\Rightarrow$ VEHICLES1($x$)
$\forall x$ PICKUP($x$) $\Rightarrow$ VEHICLES1($x$)
$\forall x$ TRAIN($x$) $\Rightarrow$ VEHICLES1($x$)

$\forall x$ VEHICLES2($x$) $\Rightarrow$ (LAWN MOWER($x$) $\vee$ ROCKET($x$) $\vee$ STREETCAR($x$) $\vee$ TANK($x$) $\vee$ TRACTOR($x$))
$\forall x$ LAWN MOWER($x$) $\Rightarrow$ VEHICLES2($x$)
$\forall x$ ROCKET($x$) $\Rightarrow$ VEHICLES2($x$)
$\forall x$ STREETCAR($x$) $\Rightarrow$ VEHICLES2($x$)
$\forall x$ TANK($x$) $\Rightarrow$ VEHICLES2($x$)
$\forall x$ TRACTOR($x$) $\Rightarrow$ VEHICLES2($x$)

---

$\forall x$ `mutual_excl`( APPLE($x$), AQUARIUM FISH($x$), BABY($x$), BEAR($x$), BEAVER ($x$), BED($x$), BEE($x$), BEETLE($x$), BICYCLE($x$), BOTTLE($x$), BOWL ($x$), BOY($x$), BRIDGE($x$), BUS($x$), BUTTERFLY($x$), CAMEL($x$), CAN($x$), CASTLE($x$), CATERPILLAR($x$) , CATTLE($x$), CHAIR($x$) CHIMPANZEE($x$), CLOCK($x$), CLOUD($x$) , COCKROACH($x$), COUCH($x$), CRAB($x$), CROCODILE($x$) , CUP($x$), DINOSAUR($x$), DOLPHIN($x$), ELEPHANT($x$), FLATFISH($x$), FOREST($x$), FOX($x$), GIRL($x$), HAMSTER($x$), HOUSE($x$), KANGAROO($x$), KEYBOARD($x$), LAMP($x$) , LAWN_MOWER($x$), LEOPARD($x$), LION($x$), LIZARD($x$), LOBSTER($x$), MAN($x$), MAPLE_TREE($x$) , MOTORCYCLE($x$), MOUNTAIN($x$), MOUSE($x$), MUSHROOM($x$), OAK_TREE($x$), ORANGE($x$), ORCHID($x$), OTTER($x$), PALM_TREE($x$), PEAR($x$), PICKUP_TRUCK($x$) , PINE_TREE($x$), PLAIN($x$), PLATE($x$), POPPY($x$), PORCUPINE($x$), POSSUM($x$), RABBIT($x$), RACCOON($x$), RAY($x$), ROAD($x$), ROCKET($x$), ROSE($x$), SEA($x$), SEAL($x$), SHARK($x$), SHREW($x$), SKUNK($x$) $\vee$ SKYSCRAPER($x$), SNAIL($x$), SNAKE($x$), SPIDER($x$), SQUIRREL($x$), STREETCAR($x$), SUNFLOWER($x$), SWEET_PEPPER($x$), TABLE($x$), TANK($x$), TELEPHONE($x$), TELEVISION($x$), TIGER($x$), TRACTOR($x$), TRAIN($x$), TROUT($x$), TULIP($x$), TURTLE($x$), WARDROBE($x$), WHALE($x$), WILLOW_TREE($x$), WOLF($x$) WOMAN($x$), WORM($x$) )

$\forall x$ `mutual_excl`( AQUATIC MAMMALS($x$), FISH($x$), FLOWERS($x$), FOOD CONTAINERS($x$), FRUIT AND VEGETABLES($x$), HOUSEHOLD ELECTRICAL ($x$), HOUSEHOLD FURNITURE($x$), INSECTS($x$) , LARGE CARNIVORES($x$), MAN-MADE OUTDOOR ($x$), NATURAL OUTDOOR SCENES($x$), OMNIVORES AND HERBIVORES($x$), MEDIUM MAMMALS($x$), INVERTEBRATES($x$) , PEOPLE($x$) , REPTILES($x$) , SMALL MAMMALS($x$), TREES($x$), VEHICLES1($x$), VEHICLES2($x$) )

Table B.4: Domain knowledge, PASCAL-Part dataset.

$\forall x$ SCREEN($x$) $\Rightarrow$ (TVMONITOR)
$\forall x$ COACH($x$) $\Rightarrow$ (TRAIN($x$))
$\forall x$ TORSO($x$) $\Rightarrow$ (PERSON($x$)$\vee$ HORSE($x$)$\vee$ COW($x$)$\vee$ DOG($x$)$\vee$ BIRD($x$)$\vee$ CAT($x$)$\vee$ SHEEP($x$))
$\forall x$ LEG($x$) $\Rightarrow$ (PERSON($x$)$\vee$ HORSE($x$)$\vee$ COW($x$)$\vee$ DOG($x$)$\vee$ BIRD($x$)$\vee$ CAT($x$)$\vee$ SHEEP($x$))
$\forall x$ HEAD($x$) $\Rightarrow$ (PERSON($x$)$\vee$ HORSE($x$)$\vee$ COW($x$)$\vee$ DOG($x$)$\vee$ BIRD($x$)$\vee$ CAT($x$)$\vee$ SHEEP($x$))
$\forall x$ EAR($x$) $\Rightarrow$ (PERSON($x$)$\vee$ HORSE($x$)$\vee$ COW($x$)$\vee$ DOG($x$)$\vee$ CAT($x$)$\vee$ SHEEP($x$))
$\forall x$ EYE($x$) $\Rightarrow$ (PERSON($x$)$\vee$ COW($x$)$\vee$ DOG($x$)$\vee$ BIRD($x$)$\vee$ CAT($x$)$\vee$ HORSE($x$)$\vee$ SHEEP($x$))
$\forall x$ EBROW($x$) $\Rightarrow$ (PERSON($x$))
$\forall x$ MOUTH($x$) $\Rightarrow$ (PERSON($x$))
$\forall x$ HAIR($x$) $\Rightarrow$ (PERSON($x$))
$\forall x$ NOSE($x$) $\Rightarrow$ (PERSON($x$)$\vee$ DOG($x$)$\vee$ CAT($x$))
$\forall x$ NECK($x$) $\Rightarrow$ (PERSON($x$)$\vee$ HORSE($x$)$\vee$ COW($x$)$\vee$ DOG($x$)$\vee$ BIRD($x$)$\vee$ CAT($x$)$\vee$ SHEEP($x$))
$\forall x$ ARM($x$) $\Rightarrow$ (PERSON($x$))
$\forall x$ MUZZLE($x$) $\Rightarrow$ (HORSE($x$)$\vee$ COW($x$)$\vee$ DOG($x$)$\vee$ SHEEP($x$))
$\forall x$ HOOF($x$) $\Rightarrow$ (HORSE($x$))
$\forall x$ TAIL($x$) $\Rightarrow$ (HORSE($x$)$\vee$ COW($x$)$\vee$ DOG($x$)$\vee$ BIRD($x$)$\vee$ SHEEP($x$)$\vee$ CAT($x$)$\vee$ AEROPLANE($x$))
$\forall x$ BOTTLE BODY($x$) $\Rightarrow$ (BOTTLE($x$))
$\forall x$ PAW($x$) $\Rightarrow$ (DOG($x$)$\vee$ CAT($x$))
$\forall x$ AEROPLANE BODY($x$) $\Rightarrow$ (AEROPLANE($x$))
$\forall x$ WING($x$) $\Rightarrow$ (AEROPLANE($x$)$\vee$ BIRD($x$))
$\forall x$ WHEEL($x$) $\Rightarrow$ (AEROPLANE($x$)$\vee$ CAR($x$)$\vee$ BICYCLE($x$)$\vee$ BUS($x$)$\vee$ MOTORBIKE($x$))
$\forall x$ STERN($x$) $\Rightarrow$ (AEROPLANE($x$))
$\forall x$ CAP($x$) $\Rightarrow$ (BOTTLE($x$))
$\forall x$ HAND($x$) $\Rightarrow$ (PERSON($x$))
$\forall x$ FRONTSIDE($x$) $\Rightarrow$ (CAR($x$)$\vee$ BUS($x$)$\vee$ TRAIN($x$))
$\forall x$ RIGHTSIDE($x$) $\Rightarrow$ (CAR($x$)$\vee$ BUS($x$)$\vee$ TRAIN($x$))
$\forall x$ ROOFSIDE($x$) $\Rightarrow$ (CAR($x$)$\vee$ BUS($x$)$\vee$ TRAIN($x$))
$\forall x$ BACKSIDE($x$) $\Rightarrow$ (CAR($x$)$\vee$ BUS($x$)$\vee$ TRAIN($x$))
$\forall x$ LEFTSIDE($x$) $\Rightarrow$ (CAR($x$)$\vee$ TRAIN($x$)$\vee$ BUS($x$))
$\forall x$ DOOR($x$) $\Rightarrow$ (CAR($x$)$\vee$ BUS($x$))
$\forall x$ MIRROR($x$) $\Rightarrow$ (CAR($x$)$\vee$ BUS($x$))
$\forall x$ HEADLIGHT($x$) $\Rightarrow$ (CAR($x$)$\vee$ BUS($x$)$\vee$ TRAIN($x$)$\vee$ MOTORBIKE($x$)$\vee$ BICYCLE($x$))
$\forall x$ MOTORBIKE($x$) $\Rightarrow$ (WHEEL($x$)$\vee$ HEADLIGHT($x$)$\vee$ HANDLEBAR($x$)$\vee$ SADDLE($x$))
$\forall x$ WINDOW($x$) $\Rightarrow$ (CAR($x$)$\vee$ BUS($x$))
$\forall x$ PLATE($x$) $\Rightarrow$ (CAR($x$)$\vee$ BUS($x$))
$\forall x$ ENGINE($x$) $\Rightarrow$ (AEROPLANE($x$))
$\forall x$ FOOT($x$) $\Rightarrow$ (PERSON($x$)$\vee$ BIRD($x$))
$\forall x$ CHAINWHEEL($x$) $\Rightarrow$ (BICYCLE($x$))
$\forall x$ SADDLE($x$) $\Rightarrow$ (BICYCLE($x$)$\vee$ MOTORBIKE($x$))
$\forall x$ HANDLEBAR($x$) $\Rightarrow$ (BICYCLE($x$)$\vee$ MOTORBIKE($x$))
$\forall x$ TRAIN HEAD($x$) $\Rightarrow$ (TRAIN($x$))
$\forall x$ BEAK($x$) $\Rightarrow$ (BIRD($x$))
$\forall x$ POT($x$) $\Rightarrow$ (POTTEDPLANT($x$))
$\forall x$ PLANT($x$) $\Rightarrow$ (POTTEDPLANT($x$))
$\forall x$ HORN($x$) $\Rightarrow$ (COW($x$)$\vee$ SHEEP($x$))

---

$\forall x$ TVMONITOR($x$) $\Rightarrow$ (SCREEN($x$))
$\forall x$ TRAIN($x$) $\Rightarrow$ (COACH($x$)$\vee$ LEFTSIDE($x$)$\vee$ TRAIN HEAD($x$)$\vee$ HEADLIGHT($x$)$\vee$ FRONTSIDE($x$) $\vee$RIGHTSIDE($x$)$\vee$ BACKSIDE($x$)$\vee$ ROOFSIDE($x$))
$\forall x$ PERSON($x$) $\Rightarrow$ (TORSO($x$)$\vee$ LEG($x$)$\vee$ HEAD($x$)$\vee$ EAR($x$)$\vee$ EYE($x$)$\vee$ EBROW($x$)$\vee$ MOUTH($x$)$\vee$ HAIR($x$) $\vee$NOSE($x$)$\vee$ NECK($x$)$\vee$ ARM($x$)$\vee$ HAND($x$)$\vee$ FOOT($x$))
$\forall x$ HORSE($x$) $\Rightarrow$ (HEAD($x$)$\vee$ EAR($x$)$\vee$ MUZZLE($x$)$\vee$ TORSO($x$)$\vee$ NECK($x$)$\vee$ LEG($x$)$\vee$ HOOF($x$)$\vee$ TAIL($x$)$\vee$ EYE($x$))
$\forall x$ COW($x$) $\Rightarrow$ (HEAD($x$)$\vee$ EAR($x$)$\vee$ EYE($x$)$\vee$ MUZZLE($x$)$\vee$ TORSO($x$)$\vee$ NECK($x$)$\vee$ LEG($x$)$\vee$ TAIL($x$)$\vee$ HORN($x$))
$\forall x$ BOTTLE($x$) $\Rightarrow$ (BOTTLE BODY($x$)$\vee$ CAP($x$))
$\forall x$ DOG($x$) $\Rightarrow$ (HEAD($x$)$\vee$ EAR($x$)$\vee$ TORSO($x$)$\vee$ NECK($x$)$\vee$ LEG($x$)$\vee$ PAW($x$)$\vee$ EYE($x$)$\vee$ MUZZLE($x$) $\vee$ NOSE($x$)$\vee$ TAIL($x$))
$\forall x$ AEROPLANE($x$) $\Rightarrow$ (AEROPLANE BODY($x$)$\vee$ WING($x$)$\vee$ WHEEL($x$)$\vee$ STERN($x$)$\vee$ ENGINE($x$)$\vee$ TAIL($x$))
$\forall x$ CAR($x$) $\Rightarrow$ (FRONTSIDE($x$)$\vee$ RIGHTSIDE($x$)$\vee$ DOOR($x$)$\vee$ MIRROR($x$)$\vee$ HEADLIGHT($x$)$\vee$ WHEEL($x$) $\vee$ WINDOW($x$)$\vee$ PLATE($x$)$\vee$ ROOFSIDE($x$)$\vee$ BACKSIDE($x$)$\vee$ LEFTSIDE($x$))
$\forall x$ BUS($x$) $\Rightarrow$ (PLATE($x$)$\vee$ FRONTSIDE($x$)$\vee$ RIGHTSIDE($x$)$\vee$ DOOR($x$)$\vee$ MIRROR($x$)$\vee$ HEADLIGHT($x$) $\vee$ WINDOW($x$)$\vee$ WHEEL($x$)$\vee$ LEFTSIDE($x$)$\vee$ BACKSIDE($x$)$\vee$ ROOFSIDE($x$))
$\forall x$ BICYCLE($x$) $\Rightarrow$ (WHEEL($x$)$\vee$ CHAINWHEEL($x$)$\vee$ SADDLE($x$)$\vee$ HANDLEBAR($x$)$\vee$ HEADLIGHT($x$))
$\forall x$ BIRD($x$) $\Rightarrow$ (HEAD($x$)$\vee$ EYE($x$)$\vee$ BEAK($x$)$\vee$ TORSO($x$)$\vee$ NECK($x$)$\vee$ LEG($x$)$\vee$ FOOT($x$)$\vee$ TAIL($x$)$\vee$ WING($x$))
$\forall x$ CAT($x$) $\Rightarrow$ (HEAD($x$)$\vee$ EAR($x$)$\vee$ EYE($x$)$\vee$ NOSE($x$)$\vee$ TORSO($x$)$\vee$ NECK($x$)$\vee$ LEG($x$)$\vee$ PAW($x$)$\vee$ TAIL($x$))
$\forall x$ MOTORBIKE($x$) $\Rightarrow$ (WHEEL($x$)$\vee$ HEADLIGHT($x$)$\vee$ HANDLEBAR($x$)$\vee$ SADDLE($x$))
$\forall x$ SHEEP($x$) $\Rightarrow$ (HEAD($x$)$\vee$ EAR($x$)$\vee$ EYE($x$)$\vee$ MUZZLE($x$)$\vee$ TORSO($x$)$\vee$ NECK($x$)$\vee$ LEG($x$)$\vee$ TAIL($x$)$\vee$ HORN($x$))
$\forall x$ POTTEDPLANT($x$) $\Rightarrow$ (POT($x$)$\vee$ PLANT($x$))

---

$\forall x$ TVMONITOR($x$)$\vee$ TRAIN($x$)$\vee$ PERSON($x$)$\vee$ BOAT($x$)$\vee$ HORSE($x$)$\vee$ COW($x$)$\vee$ BOTTLE($x$)$\vee$ DOG($x$) $\vee$ AEROPLANE($x$)$\vee$ CAR($x$)$\vee$ BUS($x$)$\vee$ BICYCLE($x$)$\vee$ TABLE($x$)$\vee$ CHAIR($x$)$\vee$ BIRD($x$)$\vee$ CAT($x$) $\vee$ MOTORBIKE($x$)$\vee$ SHEEP($x$)$\vee$ SOFA($x$)$\vee$ POTTEDPLANT($x$)

Table B.5: First noisy domain knowledge ($\tilde{\mathcal{K}}_a$), ANIMALS dataset, obtained by altering the clean knowledge of Table B.1. We report only the altered rules, highlighting the changes that make them not-coherent with the ANIMALS domain.

| | |
|---|---|
| $\forall x$ | FEATHER($x$) $\Rightarrow$ ~~BIRD($x$)~~ **MAMMAL(x)** |
| $\forall x$ | MAMMAL($x$) $\wedge$ ~~MEAT($x$)~~ **BIRD(x)** $\Rightarrow$ CARNIVORE($x$) |
| $\forall x$ | CARNIVORE($x$) $\wedge$ TAWNY($x$) $\wedge$ ~~DARKSPOTS($x$)~~ $\Rightarrow$ CHEETAH($x$) |
| $\forall x$ | BLACKSTRIPES($x$) $\wedge$ UNGULATE($x$) $\wedge$ WHITE($x$) $\Rightarrow$ ~~ZEBRA($x$)~~ **TIGER(x)** |

Table B.6: Second noisy domain knowledge ($\tilde{\mathcal{K}}_b$), ANIMALS dataset, obtained by adding new rules to the clean knowledge of Table B.1. We report only the added rules, that were explicitly created to be not-coherent with the ANIMALS domain.

| | |
|---|---|
| $\forall x$ | FLY($x$) $\Rightarrow$ MAMMAL($x$) |
| $\forall x$ | MAMMAL($x$) $\wedge$ EVENTOED($x$) $\Rightarrow$ FEATHER($x$) |
| $\forall x$ | BLACKSTRIPES($x$) $\wedge$ WHITE($x$) $\Rightarrow$ PENGUIN($x$) |
| $\forall x$ | CARNIVORE($x$) $\wedge$ DARKSPOTS($x$) $\Rightarrow$ TIGER($x$) |

Table B.7: Third noisy domain knowledge ($\tilde{\mathcal{K}}_c$), ANIMALS dataset, obtained by altering the clean knowledge of Table B.1. We report only the altered rules, highlighting the changes that make them not-fully-coherent with the ANIMALS domain. They all involve main-class-oriented conclusions.

| | |
|---|---|
| $\forall x$ | CARNIVORE($x$) $\wedge$ TAWNY($x$) $\wedge$ DARKSPOTS($x$) $\Rightarrow$ (CHEETAH($x$) $\vee$ **GIRAFFE(x)**) |
| $\forall x$ | UNGULATE($x$) $\wedge$ LONGLEGS($x$) $\wedge$ LONGNECK($x$) $\wedge$ TAWNY($x$) $\wedge$ DARKSPOTS($x$) $\Rightarrow$ (GIRAFFE($x$) $\vee$ **ZEBRA(x)**) |
| $\forall x$ | BLACKSTRIPES($x$) $\wedge$ UNGULATE($x$) $\wedge$ WHITE($x$) $\Rightarrow$ (ZEBRA($x$) $\vee$ **TIGER(x)**) |
| $\forall x$ | BIRD($x$) $\wedge$ $\neg$FLY($x$) $\wedge$ SWIM($x$) $\wedge$ BLACKWHITE($x$) $\Rightarrow$ (PENGUIN($x$) $\vee$ **OSTRICH(x)**) |

# Ringraziamenti

Grazie a tutte le persone che hanno condiviso con me anche solo una piccola parte del percorso di vita in questi anni: quello che sono oggi lo devo soprattutto alle esperienze vissute con loro, e già questo le rende speciali.

Grazie a Battista, che mi ha seguito costantemente motivandomi e trasmettendomi sia la sua passione che le sue conoscenze, e tutte le persone con le quali ho avuto occasione di collaborare durante il dottorato: Fabio, Ambra, Maura, Luca, Daniele e gli altri membri del PRALab, Guido, Enrico, Luca, Stefania, Davide e gli altri componenti del team di Pluribus One. Grazie anche a Giancarlo, Andrea, Gianluca e alle altre persone che ho incontrato durante i sei mesi al CISPA. Se ripenso a questo percorso, ho la sensazione di essermi divertito. La pandemia e altre vicissitudini mi hanno costretto a lavorare quasi interamente da remoto, e il tempo per coltivare i rapporti umani è stato davvero ridotto. Ma questo non mi ha impedito di trovare delle persone fantastiche dalle quali, oltre che imparare tantissimo, ho ricevuto tanto.

Grazie a tutti i miei amici e alla mia grande famiglia, da cui ho ricevuto affetto, supporto, aiuto e tanti bei momenti. In particolare, grazie ai miei genitori che mi hanno sempre supportato in tutto, anche quando questo non era scontato.

Grazie a Giulia, che è sempre stata al mio fianco e mi ha sopportato e sostenuto in ogni momento e in ogni modo, a volte prima che io stesso mi rendessi conto di averne bisogno. E grazie a Giovanni che, ancor prima di nascere, ci riempie di gioia: non vediamo l'ora di conoscerti.

*The research reported in this PhD thesis has been partly supported by BMK, BMDW, and the Province of Upper Austria in the frame of the COMET Programme managed by FFG in the COMET Module S3AI.*