IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 13 March 2023, accepted 22 March 2023, date of publication 29 March 2023, date of current version 6 April 2023.

*Digital Object Identifier 10.1109/ACCESS.2023.3263145*

RESEARCH ARTICLE

# Abstract Layer for LeakyReLU for Neural Network Verification Based on Abstract Interpretation

**OMAR EL MELLOUKI[1], MOHAMED IBN KHEDHER[2], AND MOUNIM A. EL-YACOUBI[3], (Member, IEEE)**

[1]ENSTA IP Paris, 91120 Palaiseau, France
[2]IRT-SystemX, 91120 Palaiseau, France
[3]Samovar, CNRS, Telecom SudParis, Institut Polytechnique de Paris, 91120 Palaiseau, France

Corresponding author: Mohamed Ibn Khedher (mohamed.ibn-khedher@irt-systemx.fr)

This work was supported in part by the French Government under the ''France 2030'' Program, as part of the SystemX Technological Research Institute; and in part by the EPI Project (EPI for ''AI-Based Decision Making Systems' Performance Evaluation'').

**ABSTRACT** Deep neural networks have been widely used in several complex tasks such as robotics, self-driving cars, medicine, etc. However, they have recently shown to be vulnerable in uncertain environments where inputs are noisy. As a consequence, the robustness of neural networks has become an essential property for their application in critical systems. Robustness is the capacity to take the same decision even when inputs are disturbed under different types of perturbations, including adversarial attacks. The great difficulty today is providing a formal guarantee of robustness, which is the context of this paper. To do so, abstract interpretation, a popular state-of-the-art method, consisting of converting the layers of the neural network into abstract layers, has been recently proposed. An abstract layer can act on a geometric abstract object or shape comprising implicitly an infinite number of inputs rather than an individual input. In this paper, we propose a new mathematical formulation of an abstract transformer to convert a LeakyReLU activation layer to an abstract layer. Moreover, we implement and integrate our transformer into the ERAN tool. For validation, we assess the performance of our transformer according to the LeakyReLU hyperparameter, and we study the robustness of the neural network according to the input perturbation intensity. Our approach is evaluated on three different datasets: MNIST, Fashion and a robotic dataset. The obtained results demonstrate the efficacy of our abstract transformer in terms of mathematical formulation and implementation.

**INDEX TERMS** Neural network verification, robustness, abstract interpretation, abstract transformer, LeakyReLU.

## I. INTRODUCTION

During the past decade, Artificial Intelligence (AI) and, in particular, Machine Learning have achieved a dramatic performance increase for a variety of critical tasks. In particular, Deep Neural Networks (DNN) have revolutionized machine learning and achieved spectacular performance across a wide range of complex applications. These applications include computer vision [1], cybersecurity [2], robotics [3], and control of autonomous systems [4], etc.

In recent years, researchers have focused on studying the robustness of neural networks in uncertain environments, and several research areas, as a result, have emerged (Figure 1). These areas are usually related to each other, but often each research work focuses on the challenges of a particular domain. The first area is called ''robustness improvement of neural networks''. The principle is to apply defense techniques in order to obtain robust models, in the sense that their performance does not change if the inputs are perturbed. At the end of this phase, robustness is not guaranteed, as it depends on the effectiveness of the defense technique. The second research area is ''testing the robustness'' by studying the behavior of the neural network against input perturbations. Indeed, as mentioned before, after the application of defense techniques, robustness is not guaranteed. Therefore, when we judge that our neural network is robust, we study its behavior against powerful, well calculated perturbations, called ''adversarial attacks''. If the network fails against the

The associate editor coordinating the review of this manuscript and approving it for publication was Prakasam Periasamy.

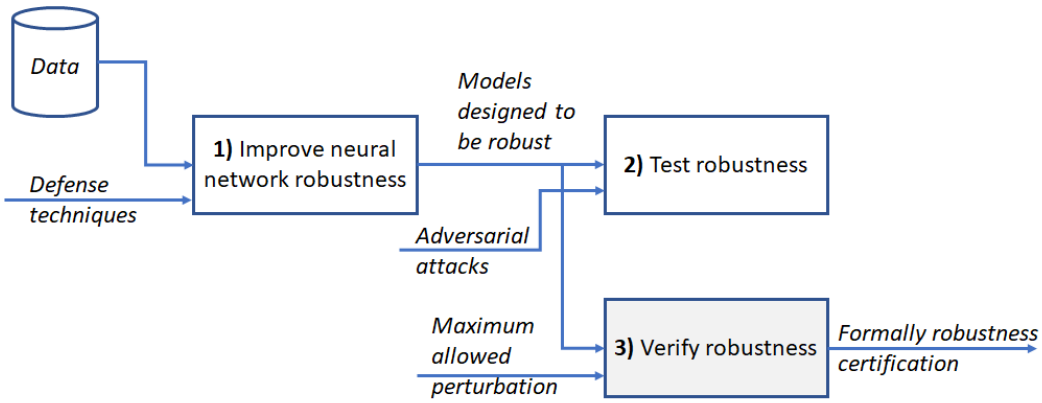



**FIGURE 1.** Research areas related to neural network robustness.

adversarial attacks, we turn around again to the robustification phase. The third research area is the ''formal verification of neural networks''. Indeed, the statement ''a neural network is robust against adversarial attacks'' does not mean that it is robust to all perturbations allowed on the inputs, but rather that it has a high chance to be robust. In this respect, a formal proof of robustness should be provided, and it is in this context that our work fits.

Despite the power of Deep Neural Networks to process high dimensional inputs, and resolve complex problems in several critical applications, it has been shown recently that small perturbations in the input space can trigger incorrect decisions [4]. Concretely, it has been observed that DNN can be easily fooled, by making their predictions change, when slightly modifying the inputs. This modification is carefully chosen, and the modified samples are known as adversarial examples. These findings have been instrumental in raising the awareness that a fundamental challenge today is ensuring that machine learning systems, and deep neural networks in particular, behave as intended, when dealing with disturbed inputs.

Adversarial examples are typically obtained by slightly perturbing an input, that was originally correctly classified by the network, in such a way that the network misclassifies the disturbed input. The adversarial examples are not randomly generated, but carefully computed. There are several techniques to generate these examples, but most of them rely on minimizing the distance between the adversarial example and the original one, while ensuring that the prediction is incorrect. Some techniques require access to all the classifier parameters (white-box attacks). In contrast, other techniques require access only to the prediction function (black-box attacks).

To generate the adversarial examples, several methods have been proposed in the literature. These techniques include the Fast Gradient Sign Method (FGSM), proposed by Goodfellow et al. [5], [6], and based on the gradient descent method. The Basic Iterative Method (BIM), proposed by Kurabin et al. in [7], is an extension of FGSM. A second extension of FGSM is the Projected Gradient Descent (PGD), proposed in [8]. In addition to gradient-based attacks, the Jacobian Saliency Map Attack (JSMA), proposed by Parpernot et al. in [9], consists in disturbing a minimal number of pixels. Another popular attack, proposed by Moosavi-Dezfooli et al. in [10], is DeepFool, which consists of finding the closest distance from the original input to the decision boundary.

**FIGURE 2.** Adversarial attacks. left: initial image, right: Blurring [11].

These adversarial samples can be dangerous as they may pose a risk to human lives. Take autonomous driving as an example. An autonomous driving system should demonstrate robustness against environmental perturbations such as lighting variation, visibility, etc. A misinterpretation of such perturbations may lead to a wrong decision, with potentially catastrophic consequences. To guarantee an acceptable robustness level, the system has to be evaluated in the face of several challenges, such as: *i*) the uncertainty of the uncontrolled driving environment, *ii*) occlusion/absence (partially or totally) of perception, etc.

As an illustration, consider the images of Fig.2 in the context of autonomous vehicles. The image on the left is an ordinary image of a limit speed sign *20 km/h*, while a blurring perturbation is added to the image on the right. Despite the degraded quality of the second image, both appear to be the same to humans, i.e. a limit speed sign to *20 km/h*. It is therefore surprising that the DNN, developed for traffic sign classification, misclassifies the right image as a *80 km/h* limit speed sign [11].

An adversarial attack, whatever its type, is just an example of a well-chosen perturbation that could be injected into

inputs. Given the criticality of certain applications, it is necessary to *formally* verify the robustness of the DNN to all possible perturbations, including adversarial attacks, before making it available to users. The robustness verification is a task independent of adversarial attacks and assumes that each DNN input (pixel) belongs to an interval reflecting the perturbation. It consists of verifying the capacity of the DNN to take the same correct decision for all possible perturbations.

To assess robustness, three major approaches have been proposed in the literature: satisfiability, reachability and optimization approaches. Satisfiability approaches consist of transforming the neural network into a feasibility problem to prove the existence of a counterexample. If a counterexample is found, the neural network is considered as not safe; if none is found, the neural network is safe. On the other side, the reachability approaches consist of calculating the reachable set (outputs) of all inputs and checking if it is included in the desired set. Regarding optimization approaches, they involve formalizing the neural network as a system of equations of conjunction or disjunction. Typically, solutions to these systems are achieved through the use of mathematical programming algorithms.

One of the main advantages of the reachability-based approaches is their scalability to input dimension. In this category, an interesting approach, named *Abstract Interpretation*, proposed in [12], aims to assess systems for resistance to unsatisfied specifications, by checking that the DNN still outputs the same label even if the inputs become noisy. A key phase of the mathematical formulation of the abstract interpretation is to convert the DNN layers into abstract layers. To the best of our knowledge, only the following activation functions have been proposed in the literature: Tanh, Sigmoid and ReLU.

The objective of this paper is to extend the mathematical formulation of the Abstract Interpretation to the LeakyReLU activation function, a recently popular activation function in the area of neural networks thanks to its better ability to learn. This enables the assessment of the robustness of neural networks containing LeakyReLU layers, that is missing in the state-of-the-art. To achieve this objective, our strategy consists of the following two contributions[1]:

- We propose a new abstract transformer that allows the use of Abstract Interpretation for verifying the robustness of DNN with LeakyReLU activation. The current state of the art does not include this transformer.
- We integrate the transformer into ERAN (ETH Robustness Analyzer for Neural Networks), which is one of the most popular formal tools. Then, several experiments for classification tasks on different datasets are performed. In addition, we note that the formulation of our transformer is generic and could be implemented under any formal tool used for abstract interpretation or interfaced with ERAN as the tool DNNV [13].
- We investigate the robustness of the neural networks according to two variables: the hyperparameter of the ReLU activation ($\alpha$) and the radius of the maximum allowed perturbation ($\epsilon$).

[1]A preliminary version had been accepted in two workshops: DataIA 2020 (https://www.dataia.eu/ws-safety-ai) and Workshop on Machine Learning and Certified Systems 2021 (https://mlcertifiedsystems.deel.ai/), but no papers had been published.

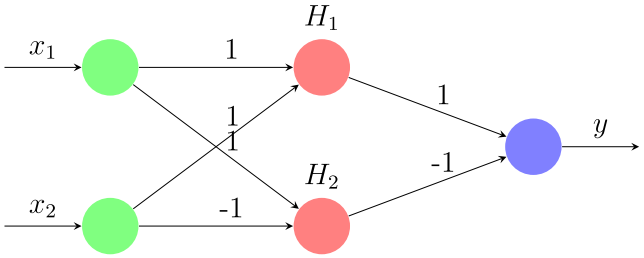



**FIGURE 3.** Example of neural network.

The rest of the paper is organized as follows. In section II, the current state of the art on methods for verifying the robustness of neural networks is given. In section III, we introduce adversarial attacks and robustness. In section IV-A, the main principles of Abstract Interpretation are described. Our approach is described in detail in sectionIV-B. Section V describes the experimental results, and section VI concludes the paper.

## II. STATE OF THE ART

To formally verify the robustness of DNNs against input perturbations, the state-of-the art approaches can be grouped according to the formulation of the problem. In this section, we introduce first the Neural Networks Verification (NNV) problem. Then, we focus on the three following formulations: satisfiability problem, reachability problem, and optimization problem.

### A. NEURAL NETWORKS VERIFICATION (NNV) PROBLEM

Given a neural network $N: x \rightarrow y$, a set of properties $P$ covering the inputs, and a set of properties $Q$ covering the outputs, the NNV problem formulation is to seek an answer to the following question: Is there an input $x$ resulting into an output $y = N(x)$, verifying $P$ and failing $Q$ ?

For more details, let us take the example of a physical system, represented by the neural network of Fig.3. This neural network takes as inputs the measurements ($x_1$ and $x_2$) of two sensors and outputs the prediction $y$ of an actuator measure, that should be greater than $(-5)$. To verify the robustness of the input $x = (x_1, x_2) = (0, 0)$, let us assume that the inputs can be disturbed by a maximum perturbation of radius 2. Verifying the robustness in this case amounts to checking the following constraint: $\forall(x_1 > -2) \wedge (x_1 < 2) \wedge (x_2 > -2) \wedge (x_2 < 2)$, we have $(y > -5)$

In this example, $P$ is then the constraints that describe the possible perturbation on the input sensor measurements: $(x_1 > -2) \wedge (x_1 < 2) \wedge (x_2 > -2) \wedge (x_2 < 2)$. As for $Q$, it is the constraint of the desired value of the actuator: $(y > -5)$

### B. SATISFIABILITY APPROACHES

The verification of a neural network can be formulated as a feasibility problem. It consists in transforming the neural network into a problem for the existence of a counterexample. Based on the example of Fig.3, the verification problem and its formulation are presented as follows: *Prove that for all possible values of inputs ($x_1$ and $x_2$), are there values of output $y$ that do not satisfy the property $Q$ of $y$ ?*

In more details, the feasibility problem associated with the example of Fig.3 consists of checking whether there exists an $x_1 \in [-2, 2]$ and $x_2 \in [-2, 2]$, where $N(x_1, x_2) <= -5$. Formally, a feasibility problem consists of taking all constraints on inputs and the negation of constraints on outputs. In our example, the feasibility problem becomes: $(x_1 > -2) \wedge (x_1 < 2) \wedge (x_2 > -2) \wedge (x_2 < 2) \wedge (y \leqslant -5)$. If the problem admits a solution, the network $N$ is considered not robust; otherwise it is robust.

In [14], the authors propose Reluplex. It is the abbreviation of ReLU for the simplex algorithm. The simplex algorithm is an algorithm for solving linear optimization problems. Its objective is to minimize a function on a set defined by inequalities. Reluplex's principle consists in formalizing the neural network by a set of equations. To solve this system of equations, starting from an initial assignment, it tries to correct some constraints violated at each step. The specification of this approach is that, from one iteration to another, the constraints between the variables can be violated.

In [15], the authors propose PLANET, ''a Piece-wise LineAr feed-forward NEural network verification Tool''. Its principle consists first of replacing the non-linear neural network functions by a set of linear equations. It then tries to find a solution to the system of equations. The approach supports both types of nodes: ReLU and Max Pooling.

In [16], the authors proposed the linearization of the non-convex ReLU activation using the technique Big-M.

### C. REACHABILITY APPROACHES

In computer science, the reachability problem in a system is the problem of determining whether a final situation is reachable from an initial situation.

Given a neural network $N$ and an input set $X$, the reachability set $Y$ is defined as all the possible outputs: $Y = \{y \mid y = N(x), \forall x \in X\}$. If the reachable set is included in the desired set, the neural network is declared as robust. Otherwise, if the reachable set is not included, totally or partially, in the desired set, the neural network is declared as not robust.

The approaches proposed in the literature consist of performing an exact [17] or approximate [18], [19] reachability analysis to determine the set of outputs. In [18], the reachability set search problem is formulated as a chain of optimization problems to compute the maximum and minimum sensitivity values for each neural network node. In [19], the authors propose a verification scheme called ''Abstract Interpretation for Artificial Intelligence'', abbreviated as AI$^2$. The main idea of AI$^2$ is to model the neural network inputs by zonohedron-based geometric shapes (the zonohedron is a special case of the polyhedron geometric shape). A set of abstract operators is then defined to propagate the evolution of the zonohedron through the neural network layers. The approach AI$^2$ is adapted in [20] to verify the robustness of neural networks against geometric transformations. Finally, the approach presented in [17] consists of exactly representing the neural network inputs by the union of polyhedra. Then, the computation of the reachability set is performed at each layer of the network using a polyhedron manipulation tool. It is worth mentioning that our approach falls under the umbrella of reachability approaches.

### D. OPTIMIZATION APPROACHES

The optimization-based formulation consists in formalizing the neural network by a conjunction or disjunction of linear properties. These properties model the relations between the successive layers. Hereafter, an example of the translation of a ''Fully-connected'' layer by a set of conjunctions is given.

First, we introduce variables $\vec{x}_i$ that represent the output vectors of layer $i$. For each layer, we encode the calculation of $\vec{x}_i$ knowing $\vec{x}_{i-1}$ by constraint $C_i$ as $C_i \equiv \{x_i^j = w_i^j * x_{i-1} + b_i^j\}$, where $w_i^j$ is the $j^{th}$ weight component of $w_i$.

After encoding all the neural network layers by a set of linear properties (equalities or inequalities), several solutions were proposed to solve the problem. These solutions can be split into two groups: *i*) Primary problem formulation and *ii*) Dual problem formulation.

Primal formulation consists in directly solving the system of equations using linear programming. The approach of [21], for instance, encodes the neural network by a set of constraints. Then, the Gurobi algorithm is applied to find a solution. The approach of [22], by contrast, looks for the maximum perturbation allowing to distort the neural network using Mixed-Integer Linear Programming (MILP).

Dual formulation, on the other hand, consists in using relaxations (approximations) of linear equations to solve the optimization problem. The approach of [23], for instance, uses Lagrangian relaxation to approximate the boundaries of the neural network nodes. In [24], the boundaries of nodes are estimated using convex relaxation, while in [25], a positive semi-definite optimization technique is used to approximate the boundaries.

## III. ADVERSARIAL ATTACKS AND ROBUSTNESS

In this section, we introduce adversarial attacks and describe the most popular one, namely FGSM. Then, we introduce the task of verifying the robustness of neural networks using abstract interpretation.

### A. ADVERSARIAL ATTACKS

An adversarial attack is defined as a small perturbation, injected into the inputs to fool the classifier's prediction. If the neural network is resistant to adversarial attacks, it is likely to be resistant to other types of perturbations. Since the latter

claim is not guaranteed, we need to formally verify the robustness of the neural network against all possible perturbations.

We recall that the goal of this paper is to verify robustness and not to study adversarial attacks. However, it is useful to describe how these attacks are generated. In this context, we refer to our research papers [26], [27], [28] for more information.

There are many ways to generate adversarial attacks. We distinguish between two types of attacks depending on the attacker's goal: targeted attacks and untargeted attacks:

- **Targeted attack:** aims to misclassify the input sample away from its original class to a specific target class.
- **Untargeted attack:** aims to misclassify the input sample away from its original class, regardless of the new output class.

Thereafter, an example of the most popular adversarial attack (FGSM) with its two versions: targeted and untargeted, is described. Fast Gradient Sign Method (FGSM) is considered as one of the first proposed attacks to fool neural networks. Goodfellow et al. [5], [6] have developed a method for generating adversarial examples based on the gradient descent technique. Given an original sample $x$, each of its components is modified by adding or subtracting a small perturbation $\epsilon$.

The method consists in considering the sign of the loss function gradient $\nabla_x \mathcal{L}(x, y)$:

- if $\nabla_x \mathcal{L}(x, y)$ is positive, then it means that the increase of $x$ increases the loss function $\mathcal{L}$.
- if $\nabla_x \mathcal{L}(x, y)$ is negative, then it means that the increase of $x$ decreases the loss function $\mathcal{L}$.

FGSM can be targeted or untargeted. For the targeted version, the adversarial function $\psi$ is expressed as in (1):

$$\begin{aligned} \psi \quad &: \mathcal{X} \times \mathcal{Y} \longrightarrow \mathcal{X} \\ &(x, y) \longmapsto -\varepsilon \,.\, \text{sign}(\nabla_x \mathcal{L}(x, y)) \end{aligned} \tag{1}$$

The adversarial sample $x'$ is then generated as in (2):

$$x' = x - \varepsilon \,.\, \text{sign}(\nabla_x \mathcal{L}(x, y)) \tag{2}$$

where $y$ is the target label of the input $x$.

Regarding the untargeted version, the adversarial function $\rho$ is expressed as in (3):

$$\begin{aligned} \rho \quad &: \mathcal{X} \times \mathcal{Y} \longrightarrow \mathcal{X} \\ &(x, y) \longmapsto \varepsilon \,.\, \text{sign}(\nabla_x \mathcal{L}(x, y)) \end{aligned} \tag{3}$$

The adversarial sample $x'$ is then generated as in (4):

$$x' = x + \varepsilon \,.\, \text{sign}(\nabla_x \mathcal{L}(x, y)) \tag{4}$$

where $y$ is the ground truth of the input $x$.

FGSM requires the computation of the loss function gradient, which makes it a simple method. On the other hand, the only hyperparameter of FGSM is $\varepsilon$, the maximum allowed perturbation.

### B. ROBUSTNESS

To introduce robustness, take the example of Fig.4 where an input has been disrupted to the point of completely changing the network decision. This is the FGSM attack, consisting of adding a noise that is proportional to the gradient sign of the loss function. The input, which was previously correctly classified as Panda (with a confidence level of 57.7%), has its output label completely changed once disturbed. It is now classified as Gibbon, with a confidence level of 99.3%, whereas for a human, the difference between the disturbed and the origin images is not perceptible. Consequently, this error could have disastrous consequences depending on the context in which it occurs.

It is therefore necessary to be able to check whether a neural network is able to correctly classify inputs even if they are disturbed. This is called the verification of the *robustness*. To verify the robustness, a Neural Network Verification (NNV) approach is needed. Thereafter, a description of the principles of a NNV problem is presented.

The perturbations are of various natures and intensities. The principle of NNV problem consists in verifying that, given an initially correctly classified sample, all its possible perturbations that can be generated remain correctly classified. For example, to verify the robustness of a neural network, for image classification, against pixels' illumination, it would be necessary, for any initially correctly classified image, to generate all possible images resulting from different kinds of illumination. Then, an assessment phase should be conducted to verify whether these images are still correctly classified.

It is not possible, however, to generate this set of modified images, as it is in fact infinite. This is where the abstract interpretation comes in, as it allows, instead of working on each disturbed image separately, considering an abstract geometrical shape that contains them all. A neural network adapted to work with this kind of abstract inputs is then used to process this abstract shape to show whether it verifies the robustness properties.

The verification of robustness takes place after the neural network has already been trained in its standard form (non-abstract way). Therefore, as mentioned in Section II-C, $AI^2$ does not allow us to correct a possible non-robustness of the neural network but simply to verify whether the latter is robust or not.

An abstract domain, representing the space of possible perturbations on the input, is then generated (shown in Fig.5 on the left by the blue parallelepiped in a simplified form for readability). This abstract domain is given as input to the neural network, which will be *adapted* so that it can act on abstract sets rather than on individual inputs. This adaptation of the neural network, i.e. the *abstraction* of the neural network, is an important part of this work. The abstract domain is then propagated through the neural network, where it is transformed by its abstract layers as it passes through them. Finally, the output of the neural network becomes an abstract shape representing the outputs of all possible perturbations

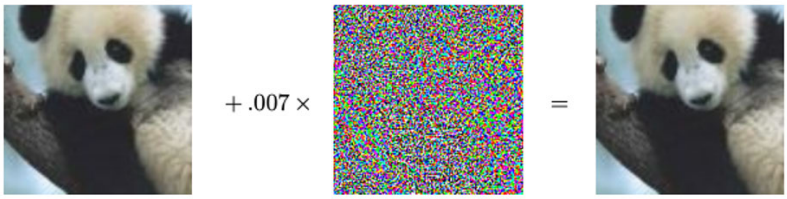



**FIGURE 4.** Impact of a FGSM attack on image classification [29]. From left to right: an original image classified as Panda with a confidence of 57.7%. A noise generated using the FGSM technique. A resulting image classified as Gibbon with a confidence of 99.3%.

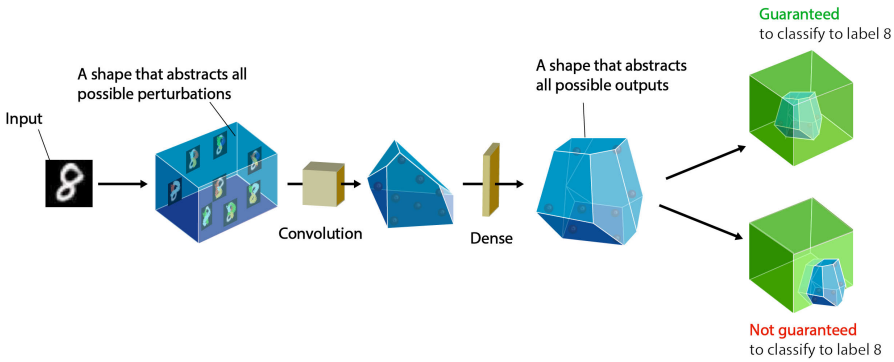



**FIGURE 5.** Abstract interpretation for neural network verification [19].

encapsulated in the input abstract shape. The output shape is then checked whether it is included in the desired set. If it is entirely included, the neural network is declared as robust.

There are various methods for constructing the abstract domain. Each one has its own characteristics, and affects differently the speed and the precision of the robustness verification. The authors of [19] studied three shapes to construct the abstract domain: the box, the zonotope and the polyhedron. We refer to the [19] for more information.

## IV. THEORY OF ABSTRACT INTERPRETATION AND APPLICATION ON LeakyReLU

This section is composed of two parts. In the first part IV-A, we present the principle of abstract interpretation applied on the ReLU activation function. In the second section IV-B, we present our extension of abstract interpretation to LeakyReLU.

### A. THEORY OF ABSTRACT INTERPRETATION

Given an input $A$, correctly classified by the Neural Network (NN), we want to check whether the NN is robust if $A$ is disturbed.

Let us represent the image $A$ by its pixels, $A = \{(x_1, x_2, \ldots, x_n), x_i \in [0, 1], \forall i \in [1, n]\}$, and consider a perturbation consisting in building a ball in $L_\infty$, of radius $\epsilon$, centered on $A$. Then, the input of $[0, 1]^n$ is converted to an abstract domain $X$, defined by $\times_{i=1}^{n}[l_i, u_i]$, where $l_i = x_i - \epsilon$ and $u_i = x_i + \epsilon$. These two quantities are called respectively *lower bound* and *upper bound*.

#### 1) THE APPROXIMATION OF ABSTRACT TRANSFORMERS

For the NN to use abstract domains, it is necessary to convert their operators into abstract transformers. An abstract domain is then obtained by applying an abstract transformer to a NN layer. In this section, to introduce theses notions, we consider the NN example of Fig.3. For technical reasons, the neurons of dense layers have been decomposed into two successive operations: an affine transformation and then the application of the ReLU function, defined by $x \mapsto \max(0, x)$.

##### a: CONSTRUCTION OF ABSTRACT DOMAIN

To proceed with the abstraction, each neuron is associated with: 1) two *linear constraints* that define the polyhedron and 2) two values representing the set of values that can be reached by this neuron. These four equalities/inequalities completely define the abstract domain at each step of the neural network processing, and they are the ones generated and processed by the abstract transformers. The abstract NN of the example in Fig.3 is presented in Fig.6.

Indeed, for each neuron, an abstract domain is generated, defined by four equalities/inequalities:

- The *upper bound* and the *lower bound*.
- The *linear constraints*: the upper constraint $a_i^{\leqslant}$ and the lower constraint $a_i^{\geqslant}$.

##### b: ABSTRACT TRANSFORMER

An abstract transformer $T_f^{\#}$ is an operator acting on the abstract domain $< a^{\geqslant}, a^{\leqslant}, l, u >$ to transform it into a new abstract domain $< a'^{\geqslant}, a'^{\leqslant}, l', u' >$. It can be defined as in (5):

$$T_f^{\#} :< a^{\geqslant}, a^{\leqslant}, l, u > \mapsto < a'^{\geqslant}, a'^{\leqslant}, l', u' > \quad (5)$$

As the NN can contain non-linear layers, it would be possible to obtain non-convex abstract domains. However, convexity is necessary for optimization. Indeed, a valid abstract transformer should guarantee the convexity.

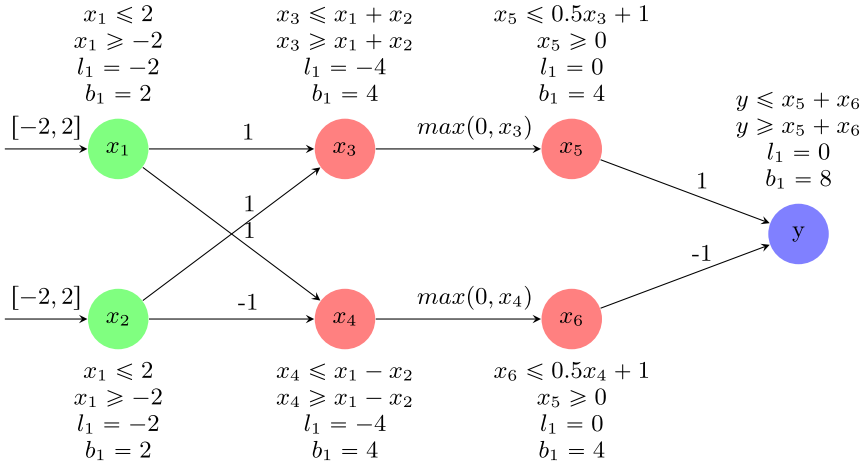



**FIGURE 6.** Example of abstract neural network.

Figure 6 represents an example of an abstract domain construction applied to the ReLU function: $x_j = \max(0, x_i)$, where $x_i \in [u_i, l_i]$. Since the ReLU curve is not convex, it is therefore necessary to complete it to make it convex by transforming the ReLU curve of Fig.7(a) into the triangle of Fig.7(b).

At this stage, the linear constraints appear to construct the abstract domain. A triangle is defined by three linear constraints. To respect the definition of an abstract domain presented in section IV-A1.a, only two linear constraints are selected. This is detailed in the next section.

#### 2) ILLUSTRATIVE EXAMPLE: PROPAGATION OF LINEAR CONSTRAINTS THROUGH THE NN

In this section, we illustrate the notion of abstract domain through the example in Fig.6. The NN is composed of one hidden layer with two neurons and ReLU activation. This processing in two steps is decomposed into two operations as it is explained in the beginning of the section IV-A1.

Thereafter, we illustrate the propagation of the abstract domain through the NN. At the input layer, $x_1$ varies between -2 and 2. The lower constraint is then ($l_1 = -2$) and the upper constraint is ($u_1 = 2$). As for constraints, the lower constraint $a^{\geqslant}$ is ($x \geqslant 2$) and the upper constraint $a^{\leqslant}$ is ($x \leqslant 2$). The same reasoning is valid for $x_2$.

At the second layer, the affine transformation ($y = w^T x + b$) is applied. As this transformation is linear, the convexity of the input is preserved in the output, and it is not necessary to approximate this result. Therefore, the constraints become ($x_3 = x_1 + x_2$) and ($x_4 = x_1 - x_2$), and the bounds are calculated using those of $x_1$ and $x_2$. However, although this calculation is exact and can be defined by strict equality, two linear constraints are created (upper and lower constraints) to respect the definition of an abstract domain. For this reason, instead of using ($x_3 = x_1 + x_2$), the equality is represented by two equivalent inequalities: ($x_3 \geqslant x_1 + x_2$) and ($x_3 \leqslant x_1 + x_2$).

At the third layer, the ReLU activation is applied to $x_3$ and $x_4$ to give respectively $x_5$ and $x_6$. The calculation of the bounds is different in this case, and it will be detailed in Section IV-A4.

At the last layer, the condition of good classification is verified. For example, assuming that the input belongs to class 1, defined by $y > 0$, we check if the condition ( $y > 0$ i.e $x_5 > x_6$) is true for all values of $x_1$ and $x_2$.

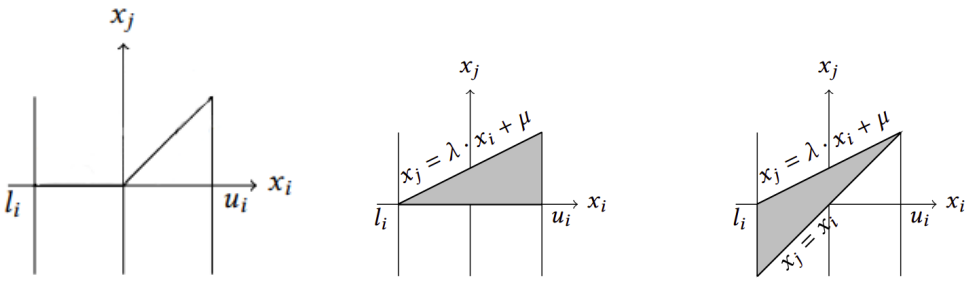
#### 3) IMPORTANT DETAILS TO APPLY $AI^2$

To apply $AI^2$, two points are important. The first remark is about the necessity to have *exactly* two linear constraints. During the propagation of the abstract domain, the number of constraints depends, at each neuron, on the number of constraints and the number of neurons in the previous layer. Indeed, if more than two constraints are considered in each neuron, there will be twice as many at the next layer. This exponential explosion of the number of constraints will require very long computations, and when processing images of several hundred pixels, the computations can quickly become unmanageable. For this reason, only two constraints are maintained at each step.

The second remark is about the *backsubstitution* principle [30]. This means that linear constraints are expressed as functions of inputs, with the best approximation. Backsubstitution is performed *every time* that a nonlinear transformation is performed. It reduces the loss of information and thus improves accuracy, which is crucial to avoid incorrect robustness verification.

#### 4) ABSTRACT TRANSFORMER FOR ReLU

The abstract transformer for ReLU is proposed in [30]. ReLU is composed of two segments, a segment with slope 1, corresponding to the case where the input is strictly positive ($l_i > 0$), and a null segment, corresponding to a negative or null input ($u_i \leqslant 0$). In these two situations, the output, i.e. the curve segment is *convex*, and does not need to be approximated, the output in this case is *exact*. The problematic case is when the output can be either positive or negative: $l_i < 0$ and $u_i > 0$. In this case, an approximation is necessary.




(a) ReLU function (b) Abstract domains depending on the constraint chosen to remove

**FIGURE 7.** ReLU function: real and abstract domains.

In summary, there are three cases to calculate the output of ReLU:

- If $u_i \leqslant 0$, the value of the neuron is necessarily negative and the output is null. The upper and lower constraints delimiting the abstract domain are: $0 \leqslant x_i$ and $x_i \leqslant 0$. $x_i$ is therefore null and no approximation is needed.
- Similarly, if $0 \leqslant l_i$, the value of the neuron can only take positive values. In this case, the output is identical to the input and neither the constraints nor the bound values are changed.
- If $l_i \leqslant 0$ and $0 \leqslant u_i$, the result will not be accurate. Considering the ReLU curve as an abstract domain, the underlying shape is not convex. By contrast, the approximation in the two previous cases is exact and does not need to be approximated.

The last case is the only problematic one. Since convexity should be ensured, it is necessary to approximate the ReLU curve by keeping the maximum amount of information. Indeed, three constraints define the abstract domain, two of which are bound by $x_j$, and one of them should be removed (Fig.7(b)). On the left, the abstract domain can be obtained by removing the constraint ($x_j \geqslant x_i$), and on the right, the abstract domain is obtained by removing the constraint ($x_j \geqslant 0$). The choice is made according to the area of the abstract domain. The one with the smallest area allows the least amount of information to be lost.

The computation of parameters of the abstract domain curves (Fig.7(b)) is a problem with two unknowns. The obtained coefficients are as in (6):

$$\lambda = \frac{u_j}{u_j - l_j}$$
$$\mu = \frac{-l_j u_j}{u_j - l_j} \tag{6}$$

To conclude, the abstract transformer, $T_f^{\#}$, of ReLU activation is formalized as follows:

- First case: If $u_i \leqslant 0$, $a'^{\geqslant} = a'^{\leqslant} = 0$, and $l' = u' = 0$; the output is equal to 0.
- Second case: If $0 \leqslant l_i$, $a'^{\geqslant} = a^{\geqslant}$, $a'^{\leqslant} = a^{\leqslant}$, $l' = l$, and $u' = u$, the output is the same as the input.
- Third case: If $u_i \geqslant 0$ and $l_i \leqslant 0$, $a'^{\geqslant} = \lambda . x_i + \mu$, and $a'^{\leqslant} = 0$ or $a'^{\leqslant} = x_i$, depending on the configuration associated with the least loss of information, where $\lambda = \frac{u_j}{u_j - l_j}$ and $\mu = \frac{-l_j u_j}{u_j - l_j}$

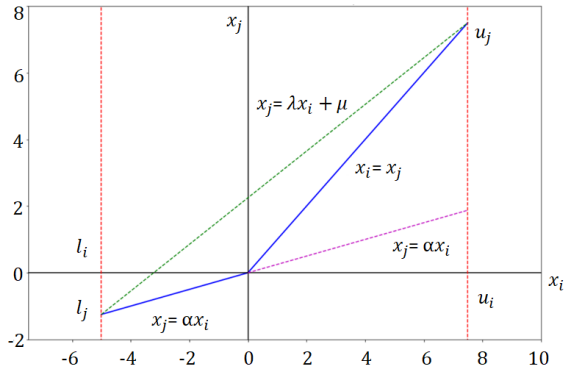



**FIGURE 8.** Abstract LeakyReLU.

### B. CONTRIBUTION TO LeakyReLU

In this section, we present our contribution, namely the proposition of an abstract transformer for the LeakyReLU activation function, the blue curve of Fig.8, defined as:

$$x \mapsto x \text{ if } x > 0, x \mapsto \alpha x \text{ otherwise, where } \alpha \in \mathbf{R}.$$

To obtain a convex abstract domain, we have connected the two extreme points by the green dashed segment, and we have extended the curve for negative $x_i$ (magenta segment). The obtained domain is the triangle of vertices: $(u_i, u_j)$, $(u_i, \alpha u_i)$ and $(l_i, l_j)$.

There are different ways to construct the abstract domain. In Fig.8, the latter is created by extending the slope line $\alpha$ and connecting the end points of the LeakyReLU. It could also be generated by extending the line $x_j = x_i$. In this case, comparing the twp areas generated by the two extensions above, we would retain the one with minimum loss information.

#### 1) ABSTRACT DOMAIN CONSTRAINTS

The constraints of the abstract domain are expressed as follows:

- Lower constraint $a_j^{\leqslant} = (x_j \geqslant \alpha x_i)$
- Upper constraint $a_j^{\geqslant} = (x_j \leqslant \lambda x_i + \mu)$.

After fixing the bounds values of $x_j$, $l_j$ and $u_j$, the parameters $\lambda$ and $\mu$ are calculated as in (7):

$$\begin{aligned} (i)\ l_j &= \alpha l_i = \lambda l_i + \mu \\ (ii) u_j &= u_i = \lambda u_i + \mu \\ (i) \text{ and } (ii) \quad \alpha l_i - u_i &= \lambda(l_i - u_i) \\ \text{Hence} \quad \lambda &= \frac{\alpha l_i - u_i}{l_i - u_i} \end{aligned} \tag{7}$$

The same reasoning is applied to find $\mu$ as in (8):

$$\mu = (\alpha - \lambda) l_j \tag{8}$$

Then, using (8), $\mu$ is given in (9).

$$\mu = \frac{l_i u_i (1 - \alpha)}{l_i - u_i} \tag{9}$$

By taking $\alpha = 0$, we find exactly the values of $\lambda$ and $\mu$ associated with classic ReLU. This proves the consistency of our abstract domain with [30]. To summarize, the abstract transformer $T_f^*(< a^{\leqslant}, a^{\geqslant}, l, u >)$ of LeakyReLU is expressed as follows:

- If $u_i \leqslant 0$, $a_i^{\leqslant}(x) = a_j^{\leqslant}(x) = \alpha x_i$, $l_j = l_i$, $u_j = u_i$
- If $l_i \geqslant 0$, $a_i^{\leqslant}(x) = a_j^{\leqslant}(x) = x_i$, $l_j = l_i$, $u_j = u_i$
- If $u_i > 0$ and $l_i < 0$ the transformer approximates the result by an abstract domain constructed by two linear constraints and the interval $[l_j,u_j]$:
  - $x_j \geqslant \alpha x_i$
  - $x_j \leqslant \frac{(\alpha l_i - u_i) x_i + l_i u_i (1-\alpha)}{l_i - u_i}$
  - $l_j = \alpha l_i$
  - $u_j = u_i$.

#### 2) ABSTRACT TRANSFORMER PROPERTIES

An abstract transformer should be verified for two properties: 1) the solidity and 2) the preservation of the invariant. To accomplish this, we should define the concretization function $\gamma_n$ [30] as in (10):

$$\begin{aligned} &\gamma_n : \mathbf{A}_n \longrightarrow \mathbf{R}^n \\ &\quad a \mapsto \{x \in \mathbf{R}^n | \forall i \in \{1, 2, \ldots, n\}, a_i^{\leqslant} \leqslant x_i \wedge\ a_i^{\geqslant} \geqslant x_i\} \end{aligned} \tag{10}$$

where $\mathbf{A}_n$ is a set of abstract domains. The concretization function returns all the elements encapsulated in the abstract domain $\mathbf{A}_n$. This function is a purely theoretical object because this set is *infinite* in practice.

##### a: PROPERTY OF SOLIDITY

The first property to verify for an abstract transformer is the property of solidity. The abstract transformer $T_f^{\#}$ of the function $f : \mathbf{A}_m \mapsto \mathbf{A}_n$ is solid if $T_f(\gamma_m(a)) \subseteq \gamma_n(T_f^{\#}(a))$ for all $a \in \mathbf{A}_m$, where $T_f$ is the *concrete* transformer. Unlike the abstract transformer, $T_f$ acts on concrete elements rather than abstract domains, and $T_f(X) = \{f(X) | x \in X\}$. This property means that the abstract transformer over-approximates the behavior of the concrete transformer.

*Proposition 4.1: The abstract transformer defined for the LeakyReLU activation verifies the solidity property:* $T_f(\gamma_m(a)) \subseteq \gamma_n(T_f^{\#}(a))$.

##### b: PROPERTY OF PRESERVING THE INVARIANT

The second property to verify is the preservation of the invariant. It is defined as follows: given an abstract transformer $T_f^{\#}$ such that $T_f^{\#}(a) = a'$, we have $\gamma_i(a') \subseteq \times_{j=1}^{i}[l'_j, u'_j]$. This property indicates that each element has a lower and an upper bound.

*Proposition 4.2: The abstract transformer $T_f^{\#}$ defined for LeakyReLU activation such that $T_f^{\#}(a) = a'$, verifies $\gamma_j(a') \subseteq \times_{i=1}^{j}[l'_i, u'_i]$.*

The demonstrations of 4.1 and 4.2 are detailed in the Appendix section.

## V. EXPERIMENTAL RESULTS

### A. DATASETS

Our approach was validated on three datasets, including two popular image benchmarks: MNIST [31] and Fashion [32]. The MNIST dataset consists of 60.000 training and 10.000 test grayscale images of handwritten digits, whose resolution is 28 × 28 pixels. The images show white digits on a black background.

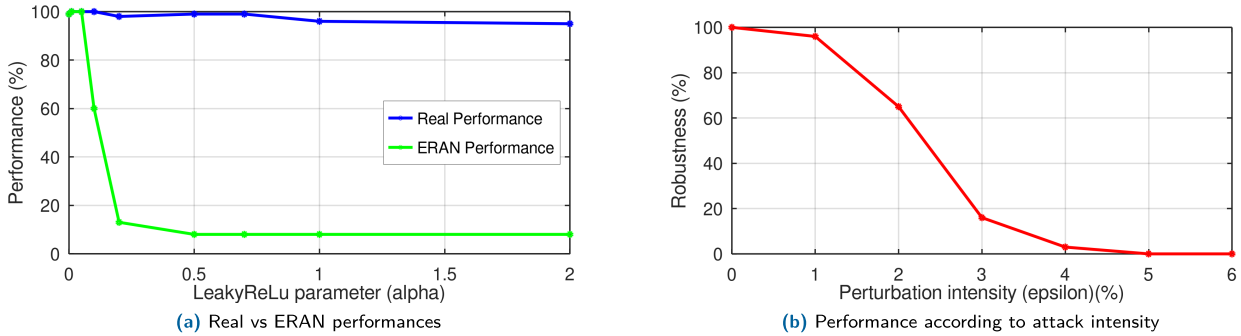
The Fashion dataset consists of 60.000 training grayscale images and 10.000 test images, with a resolution of 28 × 28 pixels. Each example is assigned to one of the following labels: T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag and Ankle boot.

The third dataset, named Robotics, is a collection of sensor readings obtained by a robot during its navigation inside a room. For robot navigation, 24 ultrasound sensors (US) were used, and arranged circularly around its waist with an arc distance of 15 degrees. The dataset contains 5456 observations. The possible decisions of the robot are: 1) Move-Forward, 2) Slight-Right-Turn, 3) Slight-Left-Turn and 4) Sharp-Right-Turn.

### B. EVALUATION TOOL/PROTOCOL

To test our contribution, we have used the ERAN tool, the Robustness Analyzer for Neural Networks tool, implemented in [33]. ERAN allows for the verification of the robustness of a trained neural network against input perturbation. It uses ELINA (ETH LIibrary for Numerical Analysis, [34]), a C-library that handles the heavy numerical computations involved in the processing of abstract domains. As LeakyReLU activation is missing from ERAN, our abstract transformer was implemented and integrated into ERAN to be able to evaluate the robustness of neural networks that contain LeakyReLU activation layers.

In the evaluation step, the network's robustness is verified for each correctly classified sample. Indeed, verifying robustness for a misclassified sample is unnecessary. First, ERAN generates the input abstract domain that will be propagated through the network. The input abstract domain includes all




(a) Real vs ERAN performances

(b) Performance according to attack intensity

**FIGURE 9.** Results on the MNIST dataset.

**TABLE 1.** Protocol evaluation on used datasets.

| Dataset | Description | Used network architecture |
|---|---|---|
| MNIST | 100 test images | 3 fully-connected layer with 50 neurons each |
| Fashion | 100 test images | |
| Robotic | 1637 test samples | |

the possible perturbations. Then, for each network layer, the associated abstract transformer is applied to propagate the abstract domain. The ELINA library provides mathematical tools to represent and propagate abstract domains.

Our experimental results are represented in terms of two metrics: performance and robustness.

- The *performance metric* is the neural network's classification rate.
- The *robustness metric* is the rate of samples that retain their (correct) output labels after perturbation.

Regarding the first metric, we recall that the abstract domain is an over-approximation of a geometric form that contains all potential input perturbations. Our formalization of the abstract domain relies on the hyperparameter $\alpha$ of the LeakyReLU activation. The more precise the approximation, the more accurate the analysis of robustness. Investigating the effect of the $\alpha$ value on the quality of the approximation of the abstract domain, is therefore relevant to assess the NN robustness in a comprehensive way.

Regarding the second metric, we recall that our abstract transformer is constructed and integrated into ERAN so that the robustness of neural networks with a LeakyReLU activation layer can be validated. It would be interesting to examine the robustness of neural networks w.r.t the maximal intensity of perturbation $\epsilon$. Regardless of the value of $\epsilon$, our implementation should not produce issues.

As evaluation methodology, we ensure that the neural network's classification rate remains the same when subjected to a null perturbation as when assessed outside ERAN. This indicates that the implementation of the abstract domains is valid. This assessment is carried out on a collection of eight neural networks with *alpha* values ranging from 0 to 2. The obtained results are shown on the left side of figures (9, 10 and 11).

To asses the effect of *epsilon*, we select one of the 8 evaluated neural networks (one having a specific *alpha*) and evaluate its robustness to input perturbation. For a comprehensive analysis, we have assessed a range of perturbation intensity (*epsilon*) values up to 10%.

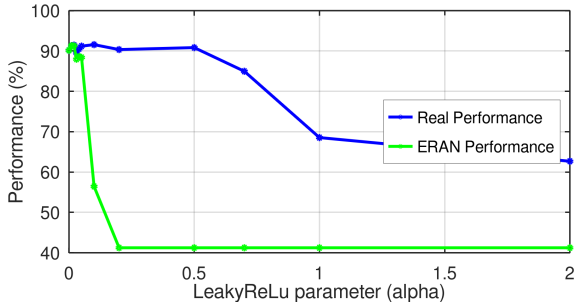
### C. RESULTS

The goal of our experiments is to validate the implementation of our abstract domain for LeakyReLU. For this purpose, we trained several neural networks. Each neural network is composed of three fully connected layers followed by a LeakyReLU activation with a different value of the $\alpha$ hyperparameter. For the same network, we used the same value of $\alpha$ for all layers. Table 1 summarizes the architecture of the used neural networks. Figures 9, 10 and 11 show respectively the obtained results on the three datasets: MNIST, Robotics and Fashion. These figures show the performance of neural networks according to the value of the $\alpha$ hyperparameter of LeakyReLU. These figures include two curves: the blue curve is associated with the performance of the neural network outside ERAN, while the green curve shows its performance inside ERAN.

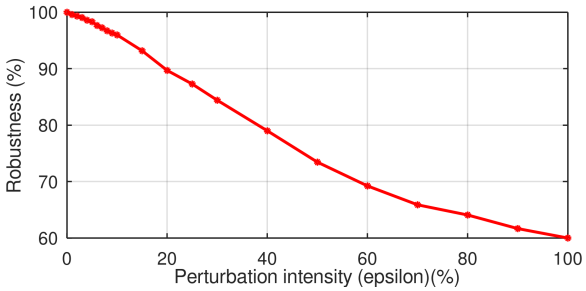
Two evaluation results are presented: 1) a quality evaluation of the abstract domain approximation and 2) a robustness evaluation of the neural network according to the maximal perturbation *epsilon*. The obtained results are shown on the right side of the figures (9, 10 and 11).

Several observations can be drawn from the obtained results. The first observation concerns the results in terms of the abstract domain's quality. Below a specific value of *alpha* (*alpha* $=$ 0.05 in the MNIST and Robotics datasets and *alpha* $=$ 0.09 in the fashion dataset), the quality of the abstract domain is always ensured. This is supported by the similar performance of neural networks within and outside of ERAN. For high *alpha* values, the over-approximation of the abstract domain becomes a coarse representation, and ERAN performance consequently degrades.

As the approximation of the abstract domain is dependent on *alpha*, for high *alpha* values, the geometrical form representing the potential perturbations becomes too large relative to reality, and the approximation of the abstract domain becomes too coarse. This remains a case to examine in order to offer more concise approximations for in the future.

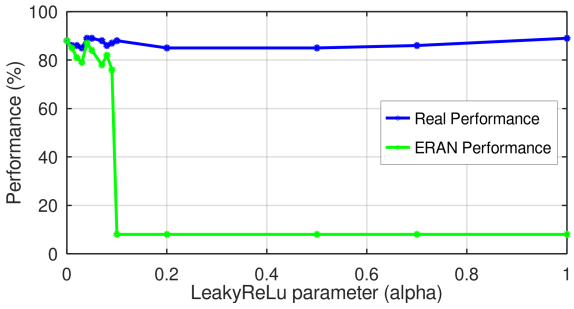



(a) Real vs ERAN performances

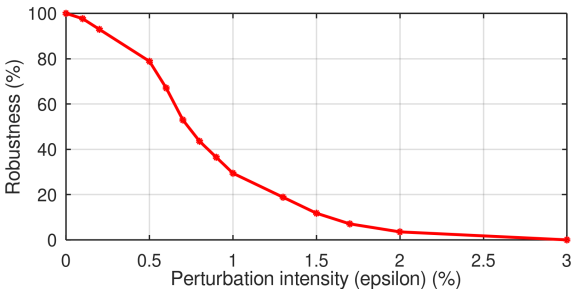



(b) Performance according to attack intensity

**FIGURE 10.** Results on the robotics dataset.




(a) Real vs ERAN performances




(b) Performance according to attack intensity

**FIGURE 11.** Results on the fashion dataset.

The second observation concerns the neural network's robustness to input perturbations. In this assessment, one model is chosen and evaluated for each dataset. The results demonstrate that the models trained on the three datasets respond differently to a perturbation of intensity *epsilon*. The robotic model is the most robust in general. In fact, 90% of robustness is always ensured when *epsilon* is less than 0.2. On the other hand, the fashion model is moderately robust; a *epsilon* value of 0.03 decreases robustness to 16%. At the same *epsilon* value (0.03) for MNIST, the model is not robust.

This evaluation demonstrates that our implementation is correct and that no issues were produced for any *epsilon* value. We recall that the purpose of this work is not to enhance the robustness of already-trained models, but rather to validate their robustness. To strengthen the robustness of these models against input perturbations, researchers have devised defensive strategies, with data augmentation being the most common.

## VI. CONCLUSION AND PERSPECTIVES

In this paper, we have studied the robustness of neural networks against input perturbations. The principal challenge was to provide a formal guarantee of robustness when the input is disturbed. To do this, we have used the abstract interpretation framework consisting of converting the neural network layers into abstract layers. Our contribution has been the proposition of a mathematical formulation of an abstract transformer to convert the LeakyReLU activation layer to an abstract layer. Moreover, our abstract transformer has been implemented and integrated into the ERAN tool.

In the evaluation step, we have investigated the effectiveness of our abstract transformer according to the LeakyReLU hyperparameter $\alpha$. On the other hand, we have studied the robustness of neural networks according to the input perturbation $\epsilon$. Our abstract transformer was tested on three public datasets: MNIST, Fashion and a robotic dataset. The obtained results showed that below a certain value of $\alpha$, our transformer is effective. However, for large values of $\alpha$, the approximation is too coarse and should be improved. Moreover, results showed that our implementation is correct and no bugs were generated for all values of $\epsilon$.

In future work, we plan to improve the approximation of our abstract domain in the case of large values of $\alpha$. Then, we will extend our study to other activation functions. Furthermore, we intend to validate our proposed approach with real-world use cases such as self-driving cars.

## APPENDIX
## PROOF OF SOLIDITY PROPERTY

*Proof:* To compute $LeakyReLU_\alpha(x_i)$, we distinguish three cases according to values of $u_i$ and $l_i$.

For the first case where $u_i \leqslant 0$, $T_f(\gamma_{j-1}(a))$ can be written as in (11), shown at the top of the next page.

For the second case where $l_j \geqslant 0$, $T_f(\gamma_{j-1}(a))$ can be written as in (12), shown at the top of the next page.

At this stage we therefore have proof that in both cases the abstract transformer verify the solidity. It should also be noted that it is sufficient to demonstrate only the inclusion but in both cases the **equality** has been demonstrated.

$$\begin{aligned} T_f(\gamma_{j-1}(a)) &= \{x \in \mathbf{R}^j \,|\, (\forall k \in [1, j-1], a_k^{\leqslant} \leqslant x_k \wedge a_k^{\geqslant} \geqslant x_k) \wedge (x_j = LeakyReLU_\alpha(x_i))\} \\ &= \{x \in \mathbf{R}^j \,|\, (\forall k \in [1, j-1], a_k^{\leqslant} \leqslant x_k \wedge a_k^{\geqslant} \geqslant x_k) \wedge (x_j = \alpha x_j)\} \\ &= \{x \in \mathbf{R}^j \,|\, \forall k \in [1, j], a_k'^{\leqslant} \leqslant x_k \wedge a_k'^{\geqslant} \geqslant x_k\} \\ &= \gamma_j(T_f^{\#}(a)) \end{aligned} \tag{11}$$

$$\begin{aligned} T_f(\gamma_{j-1}(a)) &= \{x \in \mathbf{R}^j \,|\, (\forall k \in [1, j-1], a_k^{\leqslant} \leqslant x_k \wedge a_k^{\geqslant} \geqslant x_k) \wedge (x_j = LeakyReLU_\alpha(x_i))\} \\ &= \{x \in \mathbf{R}^j \,|\, (\forall k \in [1, j-1], a_k^{\leqslant} \leqslant x_k \wedge a_k^{\geqslant} \geqslant x_k) \wedge (x_j = x_i)\} \\ &= \{x \in \mathbf{R}^j \,|\, \forall k \in [1, j], a_k'^{\leqslant} \leqslant x_k \wedge a_k'^{\geqslant} \geqslant x_k\} \\ &= \gamma_i(T_f^{\#}(a)) \end{aligned} \tag{12}$$

$$\begin{aligned} T_f(\gamma_{j-1}(a)) &= \{x \in \mathbf{R}^j \,|\, (\forall k \in [1, j-1], a_k^{\leqslant} \leqslant x_k \wedge a_k^{\geqslant} \geqslant x_k) \wedge (x_j = LeakyReLU_\alpha(x_i))\} \\ &\subseteq \{x \in \mathbf{R}^j \,|\, (\forall k \in [1, j-1], a_k^{\leqslant} \leqslant x_k \wedge a_k^{\geqslant} \geqslant x_k) \wedge (x_j \geqslant \alpha x_i) \wedge \\ &\quad (x_j \leqslant \frac{(\alpha l_i - u_i)x_i + l_i u_i(1-\alpha)}{l_i - u_i})\} \\ &= \{x \in \mathbf{R}^j \,|\, \forall k \in [1, j], a_k'^{\leqslant} \leqslant x_k \wedge a_k'^{\geqslant} \geqslant x_k\} \\ &= \gamma_j(T_f^{\#}(a)) \end{aligned} \tag{13}$$

Regarding the third case where $l_i < 0$ and $u_i > 0$, the $T_f(\gamma_{j-1}(a))$ can be written as in (13), shown at the top of the page.

The inclusion in the second line is the result of the approximation made to correct the non-convexity of the abstract domain, and it represents the loss of information in the neural network.

In conclusion, the solidity property for the abstract transformer of the LeakyReLU activation was successfully proved. This property is proved for one of the two different ways of constructing the abstract domain, but for the other way, the proof is similar. □

## PROOF OF THE PROPERTY OF PRESERVING THE INVARIANT

*Proof:*

We should proof that the concretisation of the abstract domain resulting from the abstract transformer of LeakyReLU is included in a product of interval form $[l_i', u_i']$.

If $u_i \leqslant 0$:

We have $(\forall k \in [1, i-1], a_k^{\leqslant} \leqslant x_k \wedge a_k^{\geqslant} \geqslant x_k)$ implies that $l_i' = \alpha l_i \leqslant \alpha x_i = a_j^{\leqslant}(x_i) \leqslant x_j \leqslant a_j^{\geqslant}(x_i) = \alpha x_i \leqslant \alpha u_i = u_i'$.

if $l_i \geqslant 0$:

we have $(\forall k \in [1, i-1], a_k^{\leqslant} \leqslant x_k \wedge a_k^{\geqslant} \geqslant x_k)$ implies that $l_i' = l_i \leqslant x_i = a_j^{\leqslant}(x_i) \leqslant x_j \leqslant a_j^{\geqslant}(x_i) = x_i \leqslant u_i$.

if $u_i > 0$ and $l_i < 0$:

we have $(\forall k \in [1, i-1], a_k^{\leqslant} \leqslant x_k \wedge a_k^{\geqslant} \geqslant x_k)$ implies that $l_i'$ $\alpha l_i = \alpha x_i = a_j^{\leqslant}(x_i) \leqslant x_j \leqslant a_j^{\geqslant}(x_i) = \frac{(\alpha l_i - u_i)x_i + l_i u_i(1-\alpha)}{l_i - u_i} \leqslant u_i'$.

□

## ACKNOWLEDGMENT

The authors would like to thank Gwenaëlle Berthier, EPI Project Manager, for her administrative help.

## REFERENCES

[1] M. I. Khedher, H. Jmila, and M. A. E. Yacoubi, "Fusion of interest point/image based descriptors for efficient person re-identification," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Rio de Janeiro, Brazil, Jul. 2018, pp. 1–7, doi: 10.1109/IJCNN.2018.8489111.

[2] H. Jmila, M. I. Khedher, G. Blanc, and M. A. El-Yacoubi, "Siamese network based feature learning for improved intrusion detection," in *Neural Information Processing* (Lecture Notes in Computer Science), vol. 11953, T. Gedeon, K. W. Wong, and M. Lee, Eds. Sydney, NSW, Australia: Springer, Dec. 2019, pp. 377–389, doi: 10.1007/978-3-030-36708-4_31.

[3] M. Khedher, M. Mziou, and M. Hadji, "Improving decision-making-process for robot navigation under uncertainty," in *Proc. 13th Int. Conf. Agents Artif. Intell. (ICAART)*, A. P. Rocha, L. Steels, and H. J. van den Herik, Eds. Setúbal, Portugal: SciTePress, Feb. 2021, pp. 1105–1113, doi: 10.5220/0010323311051113.

[4] R. Bunel, I. Turkaslan, P. H. Torr, P. Kohli, and M. P. Kumar, "A unified view of piecewise linear neural network verification," in *Proc. 32nd Int. Conf. Neural Inf. Process. Syst. (NIPS)*. Red Hook, NY, USA: Curran Associates, 2018, pp. 4795–4804. [Online]. Available: http://dl.acm.org/citation.cfm?id=3327345.3327388

[5] Y. Dong, F. Liao, T. Pang, H. Su, J. Zhu, X. Hu, and J. Li, "Boosting adversarial attacks with momentum," *CoRR*, vol. abs/1710.06081v3, pp. 1–12, Mar. 2018.

[6] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2014, *arXiv:1412.6572*.

[7] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," 2016, *arXiv:1607.02533*.

[8] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," 2017, *arXiv:1706.06083*.

[9] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," 2015, *arXiv:1511.07528*.

[10] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "DeepFool: A simple and accurate method to fool deep neural networks," *CoRR*, vol. abs/1511.04599, pp. 1–9, Nov. 2015.
[11] A. M. Aung, Y. Fadila, R. Gondokaryono, and L. Gonzalez, "Building robust deep neural networks for road sign detection," *CoRR*, vol. abs/1712.09327, pp. 1–9, Dec. 2017.
[12] P. Cousot, "The verification grand challenge and abstract interpretation," in *Verified Software: Theories, Tools, Experiments*. Berlin, Germany: Springer, 2008, pp. 189–201, doi: 10.1007/978-3-540-69149-5_21.
[13] D. Shriver, S. Elbaum, and M. B. Dwyer, "DNNV: A framework for deep neural network verification," in *Computer Aided Verification*, A. Silva and K. R. M. Leino, Eds. Cham, Switzerland: Springer, 2021, pp. 137–150.
[14] G. Katz, C. W. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient SMT solver for verifying deep neural networks," *CoRR*, vol. abs/1702.01135, pp. 1–31, Feb. 2017.
[15] R. Ehlers, "Formal verification of piece-wise linear feed-forward neural networks," *CoRR*, vol. abs/1705.01320, pp. 1–15, May 2017.
[16] M. Khedher, H. Ibn-Khedher, and M. Hadji, "Dynamic and scalable deep neural network verification algorithm," in *Proc. 13th Int. Conf. Agents Artif. Intell.*, vol. 2, A. P. Rocha, L. Steels, and H. J. van den Herik, Eds. Setúbal, Portugal: SciTePress, Feb. 2021, pp. 1122–1130, doi: 10.5220/0010323811221130.
[17] W. Xiang, H. Tran, and T. T. Johnson, "Reachable set computation and safety verification for neural networks with ReLU activations," *CoRR*, vol. abs/1712.08163, pp. 1–19, Dec. 2017.
[18] W. Xiang, H. D. Tran, and T. T. Johnson, "Output reachable set estimation and verification for multi-layer neural networks," *CoRR*, vol. abs/1708.03322, pp. 1–8, Aug. 2017.
[19] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev, "AI2: Safety and robustness certification of neural networks with abstract interpretation," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2018, pp. 3–18.
[20] M. Mziou-Sallami, M. I. Khedher, A. Trabelsi, S. Kerboua-Benlarbi, and D. Bettebghor, "Safety and robustness of deep neural networks object recognition under generic attacks," in *Neural Information Processing* (Communications in Computer and Information Science), vol. 1142, T. Gedeon, K. W. Wong, and M. Lee, Eds. Sydney, NSW, Australia: Springer, Dec. 2019, pp. 274–286, doi: 10.1007/978-3-030-36808-1_30.
[21] A. Lomuscio and L. Maganti, "An approach to reachability analysis for feed-forward ReLU neural networks," *CoRR*, vol. abs/1706.07351, pp. 1–10, Jun. 2017.
[22] V. Tjeng, K. Xiao, and R. Tedrake, "Evaluating robustness of neural networks with mixed integer programming," in *Proc. 7th Int. Conf. Learn. Represent. (ICLR)*, New Orleans, LA, USA, May 2019, pp. 1–21. [Online]. Available: https://openreview.net/forum?id=HyGIdiRqtm
[23] K. Dvijotham, R. Stanforth, S. Gowal, T. A. Mann, and P. Kohli, "A dual approach to scalable verification of deep networks," *CoRR*, vol. abs/1803.06567, pp. 1–14, Mar. 2018.
[24] E. Wong and J. Z. Kolter, "Provable defenses against adversarial examples via the convex outer adversarial polytope," in *Proc. 35th Int. Conf. Mach. Learn. (ICML)*, vol. 80, J. G. Dy and A. Krause, Eds., Stockholm, Sweden, Jul. 2018, pp. 5283–5292. [Online]. Available: http://proceedings.mlr.press/v80/wong18a.html
[25] A. Raghunathan, J. Steinhardt, and P. Liang, "Certified defenses against adversarial examples," in *Proc. 6th Int. Conf. Learn. Represent. (ICLR)*, Vancouver, BC, Canada, Apr./May 2018, pp. 1–15. [Online]. Available: https://openreview.net/forum?id=Bys4ob-Rb
[26] H. Jmila and M. I. Khedher, "Adversarial machine learning for network intrusion detection: A comparative study," *Comput. Netw.*, vol. 214, Sep. 2022, Art. no. 109073. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128622002146
[27] M. I. Khedher and M. Rezzoug, "Analyzing adversarial attacks against deep learning for robot navigation," in *Proc. 13th Int. Conf. Agents Artif. Intell.*, vol. 2, A. P. Rocha, L. Steels, and H. J. van den Herik, Eds. Setúbal, Portugal: SciTePress, Feb. 2021, pp. 1114–1121, doi: 10.5220/0010323611141121.
[28] H. Ibn-Khedher, M. I. Khedher, and M. Hadji, "Mathematical programming approach for adversarial attack modelling," in *Proc. 13th Int. Conf. Agents Artif. Intell.*, vol. 2, A. P. Rocha, L. Steels, and H. J. van den Herik, Eds. Setúbal, Portugal: SciTePress, Feb. 2021, pp. 343–350, doi: 10.5220/0010324203430350.
[29] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *CoRR*, vol. abs/1412.6572, pp. 1–11, Dec. 2014.
[30] G. Singh, T. Gehr, M. Püschel, and M. Vechev, "An abstract domain for certifying neural networks," in *Proc. ACM Program. Lang.*, vol. 3, Jan. 2019, pp. 1–30, doi: 10.1145/3290354.
[31] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
[32] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms," 2017, *arXiv:1708.07747*.
[33] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev. ETH-SRI/ERAN. ERAN. Accessed: 2020. [Online]. Available: https://github.com/eth-sri/eran
[34] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev. ETH Library for Numerical Analysis. Department of Computer Science, ETH Zurich, ELINA. Accessed: 2021. [Online]. Available: http://elina.ethz.ch/

**OMAR EL MELLOUKI** received the engineering degree in computer science from the National School of Advanced Techniques (ENSTA), France, in 2021. He is currently a Researcher and a Developer in the computer vision field using deep-learning algorithms for anomaly detection in video and depth estimation in stereo images.

**MOHAMED IBN KHEDHER** received the engineering and master's degrees in computer science from the National School of Computer Sciences, Tunisia, in 2007 and 2009, respectively, where he implemented a biometric system for face and gait recognition, and the Ph.D. degree in computer science from Telecom SudParis with the collaboration of the University of Evry, France. During his internship, he studied video complexity estimation in norm H.264. During the Ph.D. degree, he developed software for person re-identification from the video sequence. Since 2015, he has been a Software Engineer with the research center specializing in advanced driver assistance systems. His main research interests include neural network verification, machine learning, video coding standards, video surveillance, biometrics, human gesture recognition, and handwriting analysis.

**MOUNIM A. EL-YACOUBI** (Member, IEEE) received the Ph.D. degree in signal processing and telecommunications from the University of Rennes I, France, in 1996. During the Ph.D. degree, he was with Service de Recherche Technique de la Poste (SRTP), Nantes, France, where he developed software for handwritten address recognition for automatic French mail sorting. He was a Visiting Scientist with the Centre for Pattern Recognition and Machine Intelligence (CENPARMI), Montreal, Canada, for 18 months. He became an Associate Professor with the Catholic University of Parana (PUC-PR), Curitiba, Brazil, from 1998 to 2000. From 2001 to 2008, he was a Senior Software Engineer with Parascript, Boulder, CO, USA, a world leader company in the automatic processing of handwritten and printed documents (mail, checks, and forms). Since June 2008, he has been a Professor with Telecom SudParis, University of Paris Saclay. His main research interests include machine learning, human gesture and activity recognition, human–robot interaction, and video.