Increasing the Confidence of Deep Neural Networks by Coverage Analysis

Giulio Rossolini [ID], Alessandro Biondi [ID], *Member, IEEE*, and Giorgio Buttazzo [ID], *Fellow, IEEE*

Abstract—The great performance of machine learning algorithms and deep neural networks in several perception and control tasks is pushing the industry to adopt such technologies in safety-critical applications, as autonomous robots and self-driving vehicles. At present, however, several issues need to be solved to make deep learning methods more trustworthy, predictable, safe, and secure against adversarial attacks. Although several methods have been proposed to improve the trustworthiness of deep neural networks, most of them are tailored for specific classes of adversarial examples, hence failing to detect other corner cases or unsafe inputs that heavily deviate from the training samples. This paper presents a lightweight monitoring architecture based on coverage paradigms to enhance the model robustness against different unsafe inputs. In particular, four coverage analysis methods are proposed and tested in the architecture for evaluating multiple detection logic. Experimental results show that the proposed approach is effective in detecting both powerful adversarial examples and out-of-distribution inputs, introducing limited extra-execution time and memory requirements.

Index Terms—Neural networks coverage, DNNs robustness, adversarial examples detection

✦

1 INTRODUCTION

RECENT developments of machine learning algorithms exhibited superhuman performance to solve specific problems, as image classification, object detection, control, and strategy games. However, most of the AI algorithms developed today have been used for non-critical applications, as face aging, speech recognition, text prediction, gaming, image restoration and colorization, etc. Due to their excellent performance, there is a great industrial interest in using deep neural networks (DNNs) and, more in general, machine learning algorithms in autonomous systems, as robots and self-driving vehicles. When moving to such safety-critical application domains, several questions arise: can we trust machine-learning algorithms as they are? Are they prone to cyber-attacks? What to do if they fail? Are outputs generated within bounded response times? To address these questions, several issues need to be addressed at different levels of the architecture, as security, safety, explainability, and predictability.

This paper focuses on security and safety, which are quite intertwined. Several works have shown that DNN models are quite sensitive to small input variations, which can cause a DNN to produce a wrong prediction. This property has been used to generate adversarial examples (AEs), which are specially crafted inputs that appear genuine to humans but are incorrectly classified by the model with a high confidence.

Another serious problem is caused by inputs that are significantly different from those used to train the model. In such corner cases, the output of the model cannot be trusted. The problem can arise for several reasons. A common situation occurs when the training set contains a bias, introduced when most objects of a class appear on a similar background. In this case, the model can learn such irrelevant features, generating a wrong prediction with high confidence when an object belonging to a different class appears on the same background.

Another problem arises from the fact that, normally, DNNs for classification tasks are trained to classify m given classes using data sets containing only objects belonging to those classes. In this case, in the presence of a new object that does not belong to any of the m classes, the network cannot say "I don't know", and could be forced to take a decision by the softmax output layer, which imposes that the sum of the output values has to be 1.0. In general, when the feature distribution of the input is significantly different from those that characterize the data used to train the model, the output should not be trusted, because the input could activate unusual neural paths that could cause a wrong prediction with high confidence.

To improve the robustness of DDNs against adversarial attacks, several methods have been proposed in the literature. Some of these methods consist of augmenting the training data by suitable transformations, or adversarial examples. Other works enforced the robustness of DNNs by defenses applied at inference or testing time. They are reviewed in more detail in Section 2.

Contribution. Inspired by coverage metrics for DNNs studied in previous work for offline testing purposes, this work proposes new methods to enhance the trustworthiness of DNNs by providing a confidence value coupled to the prediction made by the network. The confidence value is obtained by coverage-based run-time monitoring.

- *The authors are with the Department of Excellence in Robotics & AI, Scuola Superiore Sant'Anna, 56127 Pisa, Italy. E-mail: {giulio.rossolini, alessandro.biondi, giorgio.buttazzo}@santannapisa.it.*
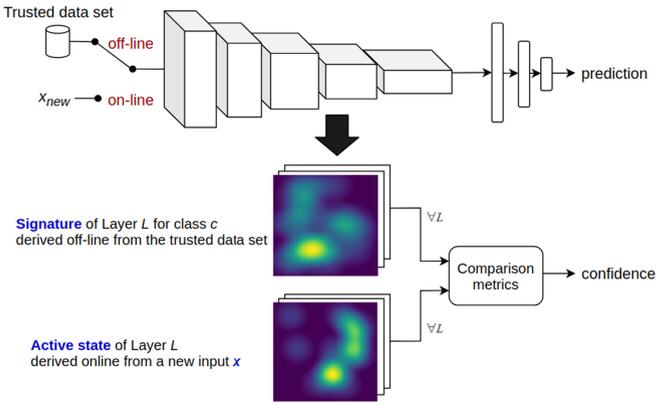
Fig. 1. Simplified scheme of the approach used to derive a confidence level for each prediction.

In particular, while the prediction score of DNNs is still associated with the model softmax output, the prediction confidence is computed by analyzing the activation state of a large number of internal neurons and comparing it with the corresponding state produced from a trusted data set. The proposed approach consists in two phases. In a preliminary (off-line) phase, the neuron outputs, in each layer and for each class, are analyzed and aggregated into a set of covered states, which all together represent a *signature* that describes how the network responded to those samples for each given class. Such a signature is derived using a trusted dataset, that is, a subset of the training samples for which that DNN generates a correct prediction with a softmax probability higher than a given threshold. Then, at runtime, each new input is subject to an evaluation phase, in which the activation state produced by the input in each layer is compared with the corresponding signature for the class predicted by the network. The approach is schematically illustrated in Fig. 1.

The adopted coverage methods are procedural, well-defined, and not based on learning-enabled components: as such, they provide solid foundations for checking the run-time behavior of DNNs in critical systems. The proposed approach can be used for detecting adversarial examples and, more in general, inputs that do not comply with the distribution of a pre-selected set of inputs used as a reference trusted data set. While many previous works rely on input transformations to accomplish the later tasks, this paper performs run-time monitoring by exploiting a white-box model analysis. Furthermore, the proposed approach proved to be suitable for resource-constrained devices, since it introduces limited extra latency at inference time and a modest memory footprint in most cases.

In summary, this paper provides the following contributions:

- It introduces a novel and flexible monitoring architecture for enhancing the trustworthiness of DNNs through coverage criteria.
- It presents four Coverage Analysis Methods (CAMs) to instantiate the proposed monitoring architecture.
- It presents an implementation of the proposed monitoring architecture and the four CAMs as GPU-accelerated extensions of the Caffe framework.

- It evaluates the proposed methods in terms of detection performance, additional inference time, and memory footprint. State-of-the-art methods to generate unsafe inputs have been used for the evaluation.

Paper Structure. The rest of the paper is organized as follows. Section 2 reviews the state-of-the-art methods focused on improving the robustness of DNNs. Section 3 introduces the notation adopted in the paper. Section 4 describes the proposed architecture with more details. Section 5 formally presents four CAMs and the corresponding algorithms. Section 6 illustrates the implementation of the proposed approach in Caffe. Section 7 reports the experimental results. Finally, Section 8 states the conclusions and future work.

2 RELATED WORK

The literature related to this work is quite vast and can be classified into (i) methods based on data augmentation, (ii) inference- and testing-time defense mechanisms, and (iii) methods to quantify the trustworthiness of DNNs.

Data Augmentation. To improve the robustness of DNNs against adversarial attacks (e.g., [1], [2], [3], [4]), various methodologies have been proposed in the literature. Most of them rely on data augmentation and work by enriching the training set with new samples generated via transformations of the available training inputs. This approach has been followed by Kurakin *et al.* [5], Pei *et al.* [6], and Shaham *et al.* [7]. Data augmentation has shown to significantly improve robustness of neural networks and their capability to generalize among new valid inputs. However, it does not help defend against unseen AEs.

Alternative approaches, such as those proposed by Sinha *et al.* [8] and Tramèr *et al.* [9], tried to make DNNs more robust by performing a specific type of data augmentation called adversarial training. Adversarial training is a type of defense, first proposed by Goodfellow *et al.* [1], that works by enriching the training set with pre-computed AEs and retraining the network until it learns to classify them correctly. Papernot *et al.* [10] have instead extended a DNN distillation technique [11] to devise a new training method capable of improving the network robustness with respect to AEs. Despite these interesting works, it is becoming more and more evident that avoiding unsafe behaviors of DNNs is a very complex or even impossible task [12], [13]. Therefore, it is preferable to accept the existence of unsafe inputs and focus on their detection by means of defense mechanisms.

Inference- and Testing-Time Defenses. Recent works tried to address these issues with new approaches based on enforcing the robustness of DNNs by means of defenses applied at inference or testing time. They rely on an online validation of the DNN behavior to establish whether a new input is likely to be dangerous. Most of the proposed detection methods are based on the observation that AEs typically lie in the proximity of the classification boundaries [14], [15]. Relevant examples of such works are those by Wang *et al.* [16], which attempts at validating network inputs using voting techniques on multiple mutations of the network model, and Srisakaokul *et al.* [17], which assembles multiple robust network models for improving the output accuracy. Biggio

et al. [18] pointed out the weaknesses of several types of Deep Learning algorithms using gradient-based attacks to question a general detection mechanism, examining different levels of knowledge of the attacked system. Also Carlini and Wagner [13] addressed these issues by showing the poor effectiveness of various detection approaches.

Other detection methods, such as Features Squeezing [19] and Vision-Guard [20] study the divergence of the probabilities score vector among network outputs when providing as network inputs multiple versions of the same image obtained with transformations (e.g., jpeg compression, rotation, etc.). These methods were conceived by focusing on the fragility of specific types of AEs, as also pointed out in [21]. Other approaches, such as DefenseGan [22] and MagNet [23], use generative adversarial networks and autoencoders, respectively, to remove possible malicious perturbations from the input sample, which could lead the network to fail the prediction.

A common drawback of several strategies mentioned above is that they are not explicitly conceived to improve the trustworthiness of the network in general, i.e., by deriving interpretable assessments of the model behavior at run-time, but rather to counteract specific classes of AEs. As such, they may fail in detecting other corner cases or other dangerous inputs (e.g., inputs that are unsafe due to over-generalization [14] or AEs that are not compliant with a norm-based definition [24], [25]). Section 7 shows how the approaches proposed in this work are capable of improving the detection performance against different types of unsafe inputs, also introducing less run-time overhead.

Defenses Based on Internal Network Analysis. Much closer to this paper are other works that investigated the internal behavior of DNNs to improve the trustworthiness of their outputs. For instance, Papernot *et al.* [26] introduced a new inference logic for DNN-based classifiers, called DkNN. According to this method, a classical inference is used for extracting the internal activations values of the network. Then, such activations are processed with a statistical analysis based on the k-Nearest Neighbors algorithm to decide the network output. This approach has been shown to improve the classification performance of DNNs in the presence of malicious inputs, possibly due to the fact that it relies on a more interpretable prediction score than a classical softmax-based output, which has frequently been cited as a poorly credible confidence metrics [26], [27], [28].

Note that most of the works cited above were not designed to evaluate the trustworthiness of the original DNN model. Rather, they drastically change the inference logic or the input data to improve the robustness of the model. Nevertheless, it has been shown that they can still fail to detect several unsafe inputs [29], [30]. Furthermore, it is important to remark that most of the works discussed are not suitable for being applied in real-time and/or on embedded devices. Indeed, they require accessory software components that considerably increase the memory footprint (e.g., [26]) and the running time of the inference phase (e.g., [22]).

Finally, it is worth mentioning other relevant approaches that improve the trustworthiness of DNNs by processing internal network activations through machine learning algorithms. Sotgiu *et al.* [31] proposed a complementary architecture to the original DNN, called Deep Neural Rejection (DNR), which quantifies the coherency of the neuron outputs generated by a new input with the respect to the one generated by training samples. Although the basic concept is very close to the DkNN [26], DNR uses Support Vector Machines with Radial Basis Kernels. Also, Carrara *et al.* [32] followed a similar approach, using a Long Short-Term Memory (LSTM) to detect unexpected activation patterns deemed as consequences of adversarial examples. A drawback of these approaches is that the machine learning used to improve the trustworthiness of DNNs may in turn be a source of untrustworthiness for the whole system, hence just shifting the problem.

Quantification of the DNN Trustworthiness. Significant attention has also been devoted to metrics that quantify the trustworthiness of DNNs, possibly with the end of matching certification requirements with formal guarantees. As Huang *et al.* [24] pointed out, it is useful to distinguish between verification and testing methods. The goal of verification methods is to derive formal robustness properties for neural networks [33], [34], [35]. However, due to inherently hardness of the corresponding verification problems, they suffer of scalability issues, especially when applied to modern DNN architectures [36]. For this reason, testing methods are usually preferred to approximate the certification process to work on a finite set of test inputs.

The completeness of the number and type of the generated test cases is usually measured by applying *coverage criteria* for DNNs, which are considerably different from those that are commonly used for classical software [37]. Pei *et al.* [6] first proposed neuron coverage as a criterion for testing DNNs. Subsequently, multiple testing metrics based on neurons were proposed [38].

In addition to the neuron-based criteria, more accurate criteria based on Modified Condition/Decision Coverage (MC/DC) [39] were proposed by Sun *et al.* [40] to improve the coverage performance. Coverage criteria for DNNs provide a reasonable compromise between formal robustness guarantees and their computational cost. They are also applied in both white-box, black-box, and concolic testing approaches to extract a large set of AEs for adversarial training [6], [38], [40]. It is worth noting that such coverage techniques for DNNs have all been conceived for testing purposes and, per se, do not constitute defense mechanisms. Indeed, they are not capable to detect unsafe inputs, even if applied online. Conversely, the coverage-based methods proposed in this work are not aimed at retrieving static trustworthiness measures off-line, but at accomplishing a run-time monitoring of the model behaviors against new inputs. As such, the methods proposed in this work are not comparable with the ones in [6], [38], [40] because they serve a different purpose.

Finally, it is also worth mentioning that several studies [41], [42] argue how coverage metrics for DNN testing are not correlated to adversarial examples. In particular, they state a limited relationship between inputs generated to meet coverage criteria and the ones obtained by popular adversarial attacks. Although our work still involves coverage metrics for DNNs, its final objective and the approach are completely different from the ones studied in [41], [42], making the corresponding observations not applicable. As a

matter of fact, the experimental results presented in Section 7 demonstrate the capability of our methods of detecting adversarial examples, meaning that it exists a substantial correlation between them and the proposed methods. Furthermore, as supplementary material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TSE.2022.3163682, we report additional tests to demonstrate the relationship between our approaches and the adversarial examples generated with popular attacks.

This Work. Differently from previous work, this paper presents a general approach to monitor the network state at inference time (white-box analysis), which is then used to employ new algorithms based on coverage methods to quantify the trustworthiness of the network output as well as detect dangerous inputs on-the-fly when they are presented to the network. Traditional, well-defined procedural algorithms are used for this purpose. The standard inference process of DNNs is retained and extended just to extract intermediate results such as the neuron outputs. The approach relies on a new kind of layer that runs a passive coverage verification to detect unsafe activation patterns with limited extra execution time.

3 TERMINOLOGY AND NOTATION

Before illustrating the proposed coverage monitoring pipeline, this section introduces the required formalism.

A feed-forward DNN is an architecture composed of a set of K *layers* $L = \{L_k \mid k \in \{1, \ldots, K\}\}$ linked by a series of weighted connections. Each layer L_k consists of a set N_k of l_k *neurons*, where the jth neuron is denoted by $n_{k,j} \in N_k$. The whole set of neurons (except the input ones) is defined as $N = \{n_{k,j} \mid k \in \{2, \ldots, K\}, j \in \{1, \ldots, l_k\}\}$. The neurons within each layer L_k are organized into $C_k \geq 1$ *channels*, where the set of neurons in the ith channel is denoted by $N_k^{\mathrm{ch}-i} \subseteq N_k$.

Each neuron $n_{k,j} \in N$ is associated with two variables $u_{k,j}$ and $v_{k,j}$ that denote its value before and after the activation function ϕ_k, respectively. The neuron output is hence defined as

$$v_{k,j} = \phi_k(u_{k,j}) \quad \text{with} \quad u_{k,j} = b_{k,j} + \sum_{h=1}^{l_{k-1}} w_{k,h,j} \cdot v_{k-1,h}, \quad (1)$$

where $w_{k,h,j}$ is the *weight* of the connection between neurons $n_{k-1,h}$ and $n_{k,j}$, and $b_{k,j}$ is the *bias* term of neuron $n_{k,j}$. In the following, it is sometimes required to distinguish between the output values of the same neuron when the network is provided with different inputs. To this purpose, the output of each neuron $n_{k,j}$ when the network input is x is denoted by $v_{k,j}(x)$.

This work focuses on classification tasks, where the DNN associates a generic input x to a class $\hat{y}$ belonging to a set of $m = l_K$ classes. Specifically, the network makes a prediction $\hat{y}$, which is the index of the neuron in the output layer with the largest value, that is $\hat{y} = \mathrm{argmax}_{1 \leq j \leq m} \{v_{K,j}\}$, where $\{1, \ldots, m\}$ is the set of classes and each value $v_{K,j}$ represents the prediction score generated by a softmax output.

To help the presentation of the following results, it is also convenient to introduce a vector notation for the DNN
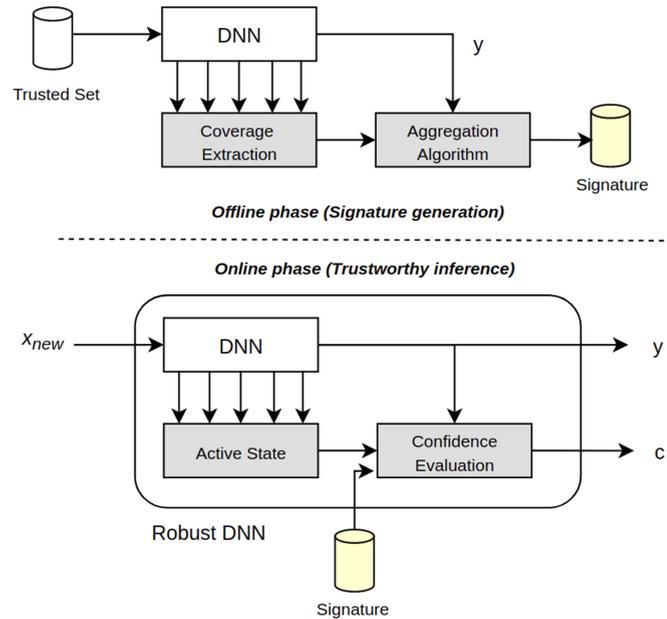

Fig. 2. Overview of the monitoring architecture with its offline and online phases. Grey boxes denote meta-algorithms, i.e., those whose behavior depend on the selected coverage criterion.

parameters. Let $V_k(x)$ be the vector denoting the activations of the neurons in layer l_k for an input x, and let $v_{k,j}(x)$ be the j-th element of $V_k(x)$. In this way, a DNN can be alternatively expressed as a function $f : \mathbb{R}^{l_1} \to \mathbb{R}^{l_K}$ such that $f(x) = \phi_{\mathbf{K}}(\phi_{\mathbf{K-1}}(\ldots \phi_{\mathbf{2}}(x)))$, where $\phi_{\mathbf{k}}$ is the vector-wise version of the activation function ϕ_k. Here, $\mathbb{R}^{l_1}$ is the input space (e.g., in the case of image classification, it represents all the possible configurations of pixels of the input image), while $\mathbb{R}^{l_1} \times \cdots \times \mathbb{R}^{l_K}$ is the network space, which includes all the output values produced by the neurons in N.

This work also considers convolutional neural networks (CNNs) [43], which can be studied as a special case of DNNs as modeled above.

4 MONITORING ARCHITECTURE

This section presents the monitoring architecture proposed to detect unsafe inputs at inference time. It is based on a novel way of applying coverage criteria. Despite coverage criteria were originally conceived for off-line testing purposes, in our work they are leveraged to identify a series of activation patterns that are deemed safe, because generated by the network on a set of trusted inputs (i.e., those producing a correct prediction with a score higher than a given threshold). Then, at runtime, such patterns are compared with those generated by a new input, measuring how they match according to a given confidence metric.

The workflow of the proposed architecture is illustrated in Fig. 2 and consists of an offline phase (also called *Signature generation*) and an online phase (also called *Trustworthy inference*).

Offline Phase (Signature Generation). In this phase, the network processes a set of trusted inputs, denoted by *Trusted Set*, that generate correct network outputs with a high prediction score. Each sample in the Trusted Set is used to perform inference on the target trained DNN. During inference, the intermediate results produced by the various

layers of the DNN (e.g., the neuron outputs) are recorded and used to apply a certain coverage criterion, denoted by *Coverage Analysis Method* (CAM) in the following. The coverage results across all inputs in the Trusted Set are then grouped by the corresponding output classes and aggregated to produce a representation of the covered activation patterns (by the Aggregation Algorithm). This representation is then compressed to produce a file called *DNN Signature*. The DNN Signature encodes all network activation patterns that are deemed safe for each output label, under a certain CAM. Both the Coverage Extraction and the Aggregation Algorithm depend on the selected CAM. This work considers four CAMs, which are presented in Section 5.

Online Phase (Trustworthy Inference). The network is deployed together with a DNN Signature. At inference time, given a new input x_{new}, the same CAM used to generate the DNN Signature is applied to extract the coverage result generated by x_{new}, which is referred to as *DNN Active State*. Given the class $\hat{y}$ predicted for x_{new}, a Confidence Evaluation Algorithm computes the matching level between the Active State stimulated by x_{new} and the trusted one encoded in the DNN Signature corresponding to class $\hat{y}$, producing a confidence value for the prediction. When such a confidence is below a given threshold, the input x_{new} is deemed unsafe. The configuration of such a threshold is addressed in Section 7. The Confidence Evaluation Algorithm also depends on the selected CAM and four options are presented in Section 5.

5 COVERAGE ANALYSIS METHODS

This section presents three CAMs together with the corresponding algorithms required to instantiate the monitoring architecture illustrated in Fig. 2. A fourth CAM, which did not achieve acceptable performance during our experimental evaluation, is presented in the supplementary material, available online.

Note that all state-of-the-art coverage metrics for DNNs introduced in Section 2 are conceived with a different purpose (i.e., DNN coverage testing). Nevertheless, some of them inspired the following CAMs, which are instead designed for being applied to the monitoring scheme proposed in Section 4.

5.1 Single-Range Coverage (SRC)

This CAM is based on the Neuron Boundary Coverage criterion [38] and works by analyzing the range of output values produced by each neuron. During the offline phase, the minimum and maximum output values produced by the neurons when testing the Trusted Set are recorded to set the range of the "typical" behavior of the network when stimulated by trusted inputs.

The Trusted Set S is split into m subsets $S_1, \ldots, S_m$, one for each class, where S_i denotes the set of inputs in S belonging to the ith class. The DNN Signature σ_i for the ith class (with $i = 1, \ldots, m$) is a collection of pairs $\sigma_{i,k,j} = (v_{i,k,j}^{\min}, v_{i,k,j}^{\max})$, where $v_{i,k,j}^{\min}$ and $v_{i,k,j}^{\max}$ denote the minimum and maximum output values produced by neuron $n_{k,j}$, respectively, over all inputs in S_i. The Aggregation Algorithm of SRC is summarized in Algorithm 1.

Algorithm 1. Aggregation Algorithm of SRC

Input Trusted Set S, trained DNN
Output DNN Signature σ

for $S_i \in S$ **do**
  $\sigma_i = \{\}$
  for $n_{k,j} \in N$ **do**
    $v_{i,k,j}^{\min} = \min_{x \in S_i}\{v_{k,j}(x)\}$
    $v_{i,k,j}^{\max} = \max_{x \in S_i}\{v_{k,j}(x)\}$
    $\sigma_{i,k,j} = (v_{i,k,j}^{\min}, v_{i,k,j}^{\max})$
    Add $\sigma_{i,k,j}$ to σ_i
  end
end
return $\sigma = \{\sigma_1, \ldots, \sigma_m\}$

During the online phase, given a new input x_{new} and the corresponding class $\hat{y}$ predicted by the DNN, the DNN Signature $\sigma_{\hat{y}}$ is compared against the output values produced by the neurons, denoted as the DNN Active State. The Confidence Evaluation Algorithm of SRC returns a confidence value that is computed as a function of the number of neurons whose output value $v_{k,j}(x_{\text{new}})$ is outside the range specified by pair $\sigma_{\hat{y},k,j}$. The confidence is computed as $c = \exp(-\frac{\eta \cdot ln(2)}{\tau_{\hat{y}}})$, where η is the number of neurons out of range and $\tau_{\hat{y}}$ is a class-dependent parameter, called *threshold*, which tunes the slope of the exponential function (the higher the threshold, the faster the function goes to zero). Note that, if $\eta = \tau_{\hat{y}}$, then $c = 0.5$. The specific values for parameters $\tau_{\hat{y}}$ are set with a calibration procedure presented in Section 7.3. The rationale of this formula is to map the computed coverage cost η to a confidence value between 0 and 1, to be compliant with typical softmax scores. A smooth exponential function is adopted also because it allows assigning a high confidence when just a few values fall outside the ranges in the signature.

This procedure is reported in Algorithm 2, assuming to monitor all the network's neurons N. In practice, to balance performance with computation time, all these algorithms can also be executed on a subset of N (see Section 6).

Algorithm 2. Confidence Evaluation Algorithm of SRC

Input x_{new}, DNN Signature σ, trained DNN, thresholds τ_i
Output confidence c

$\hat{y} = \text{argmax}_{1 \le j \le m}\{f(x_{\text{new}})\}$
$\eta = 0$
for $n_{k,j} \in N$ **do**
  Extract $\sigma_{\hat{y},k,j}$ from $\sigma_{\hat{y}}$
  $(v_{\hat{y},k,j}^{\min}, v_{\hat{y},k,j}^{\max}) = \sigma_{\hat{y},k,j}$
  if $v_{k,j}(x_{\text{new}}) \notin [v_{\hat{y},k,j}^{\min}, v_{\hat{y},k,j}^{\max}]$ **then**
    η ++
  end
end
return $c = \exp(-\frac{\eta \cdot ln(2)}{\tau_{\hat{y}}})$

5.2 Multi-Range Coverage (MRC)

This CAM is based on the K-Multi-Section coverage criterion [38] and extends SRC by introducing multiple ranges to analyze the output of the neurons. The offline phase

works as follows. First, as for SRC, the minimum and maximum output values produced by the neurons when testing the Trusted Set are recorded. Then, each output range is evenly split into Q sub-ranges. Finally, for all the inputs of the Trusted Set, the algorithm counts the number of times in which the neuron outputs fall in a given sub-range.

Formally, using the same notation introduced for SRC, for each output class with index i, the output range of each neuron $n_{k,j}$ is split into Q sub-ranges of size $\Delta_{i,k,j} = (v_{i,k,j}^{\max} - v_{i,k,j}^{\min})/Q$. For convenience, the last of such sub-ranges is defined as $[v_{i,k,j}^{\min} + (Q-1)\Delta_{i,k,j}, v_{i,k,j}^{\max}]$, while the others have open right endpoints.

The DNN Signature σ_i for the ith class is a collection of tuples $\sigma_{i,k,j} = (v_{i,k,j}^{\min}, v_{i,k,j}^{\max}, \lambda_{i,k,j}^1, \lambda_{i,k,j}^2, \dots, \lambda_{i,k,j}^Q)$, where $\lambda_{i,k,j}^q \in [0,1]$ (with $q = 1, \dots, Q$) is given by the number of times the output of neuron $n_{k,j}$ falls within the qth sub-range when the network is stimulated by inputs in S_i, divided by the cardinality of S_i itself. The Aggregation Algorithm of MRC, reported in Algorithm 3, first computes the minimum and maximum output values of the neurons, and then the corresponding occurrences to produce the DNN Signature.

Algorithm 3. Aggregation Algorithm of MRC

Input Trusted Set S, trained DNN, number of sections Q
Output DNN Signature σ

---

for $S_i \in S$ **do**
　$\sigma_i = \{\}$
　for $n_{k,j} \in N$ **do**
　　$v_{i,k,j}^{\min} = \min_{x \in S_i}\{v_{k,j}(x)\}$
　　$v_{i,k,j}^{\max} = \max_{x \in S_i}\{v_{k,j}(x)\}$
　　$\Delta_{i,k,j} = (v_{i,k,j}^{\max} - v_{i,k,j}^{\min})/Q$
　　$\lambda_{i,k,j}^1 = 0, \lambda_{i,k,j}^2 = 0, \dots, \lambda_{i,k,j}^Q = 0$
　　for $x \in S_i$ **do**
　　　$q = \max\{1, \lceil (v_{k,j}(x) - v_{i,k,j}^{\min})/\Delta_{i,k,j} \rceil\}$
　　　$\lambda_{i,k,j}^q{+}{+}$
　　end
　　$\lambda_{i,k,j}^1/ = |S_i|, \lambda_{i,k,j}^2/ = |S_i|, \dots, \lambda_{i,k,j}^Q/ = |S_i|$
　　$\sigma_{i,k,j} = (v_{i,k,j}^{\min}, v_{i,k,j}^{\max}, \lambda_{i,k,j}^1, \lambda_{i,k,j}^2, \dots, \lambda_{i,k,j}^Q)$
　　Add $\sigma_{i,k,j}$ to σ_i
　end
end
return $\sigma = \{\sigma_1, \dots, \sigma_m\}$

The idea behind the online phase of MRC is that the more a new input x_{new} produces neuron output values outside the sub-ranges matched by the inputs in the Trusted Set, the more x_{new} is likely to be unsafe. Given x_{new}, the DNN Active State is composed of the identifiers of the sub-ranges to which the output of the neurons belong to, if any. This is used to assign a cost $\Theta_{k,j}(x_{\text{new}}) \in [0,1]$ to each neuron $n_{k,j}$ that quantifies how much the neuron output is deemed unsafe. Being i the index of the class assigned to x_{new}, this cost is formally defined as

$$\Theta_{k,j}(x_{\text{new}}) = \begin{cases} 1, & \text{if } v_{n_{k,j}}(x_{\text{new}}) \notin [v_{i,k,j}^{\min}, v_{i,k,j}^{\max}] \\ 1 - \lambda_{i,k,j}^{q^*}, & \text{otherwise}, \end{cases} \quad (2)$$

where q^* denotes the index of the sub-range of $n_{k,j}$ to which $v_{n_{k,j}}(x_{\text{new}})$ belongs to.

The Confidence Evaluation Algorithm, reported in Algorithm 4, computes the sum of such costs $\Theta_{k,j}(x_{\text{new}})$ over all neurons, denoted here as η. Hence, the confidence score is computed as $c = \exp(-\frac{\eta \cdot \ln(2)}{\tau_{\hat{y}}})$.

Algorithm 4. Confidence Evaluation Algorithm of MRC

Input x_{new}, DNN Signature σ, trained DNN, thresholds τ_i, number of sections Q
Output confidence c

$\hat{y} = \operatorname{argmax}_{1 \le j \le m}\{f(x_{\text{new}})\}$
$\eta = 0$
for $n_{k,j} \in N$ **do**
　Extract $\sigma_{\hat{y},k,j}$ from σ_i
　Extract $v_{\hat{y},k,j}^{\min}$ and $v_{\hat{y},k,j}^{\max}$ from $\sigma_{\hat{y},k,j}$
　if $v_{k,j}(x_{\text{new}}) \notin [v_{\hat{y},k,j}^{\min}, v_{\hat{y},k,j}^{\max}]$ **then**
　　$\eta \mathrel{+}= 1$
　end
　else
　　$\Delta_{\hat{y},k,j} = (v_{\hat{y},k,j}^{\max} - v_{\hat{y},k,j}^{\min})/Q$
　　$q^* = \max\{1, \lceil (v_{k,j}(x_{\text{new}}) - v_{\hat{y},k,j}^{\min})/\Delta_{\hat{y},k,j} \rceil\}$
　　Extract $\lambda_{\hat{y},k,j}^{q^*}$ from $\sigma_{\hat{y},k,j}$
　　$\eta \mathrel{+}= 1 - \lambda_{\hat{y},k,j}^{q^*}$
　end
end
return $c = \exp(-\frac{\eta \cdot \ln(2)}{\tau_{\hat{y}}})$

5.3 k-Nearest Neighbors Coverage (kNNC)

This CAM is based on the DkNN algorithm presented in [26], which was originally proposed as an alternative inference logic and not as a way to detect unsafe inputs. The key idea of the kNNC is to compute a confidence score by applying the k-Nearest Neighbors (kNN) algorithm on the output values produced by the neurons of the various DNN layers.

For each output class with index i, the DNN Signature is composed of a collections of sets $\sigma_{i,k}$ of vectors, one for each layer L_k. Each set $\sigma_{i,k}$ is composed by aggregating the vectors $V_k(x)$ obtained from all inputs $x \in S_i$. The corresponding procedure is reported in Algorithm 5.

Algorithm 5. Aggregation Algorithm of kNNC

Input Trusted Set S, trained DNN
Output DNN Signature σ

---

for $S_i \in S$ **do**
　$\forall k = 1, \dots, K, \ \sigma_{i,k} = \{\}$
　for $x \in S_i$ **do**
　　for $L_k \in L$ **do**
　　　Add $V_k(x)$ to $\sigma_{i,k}$
　　end
　end
　$\sigma_i = \{\sigma_{i,1}, \dots, \sigma_{i,K}\}$
end
return $\sigma = \{\sigma_1, \dots, \sigma_K\}$

During the online phase of kNNC, given a new input x_{new} and its corresponding predicted class $\hat{y}$, the kNN algorithm is used to compute the confidence value. In the following, the parameter that controls the kNN algorithm is
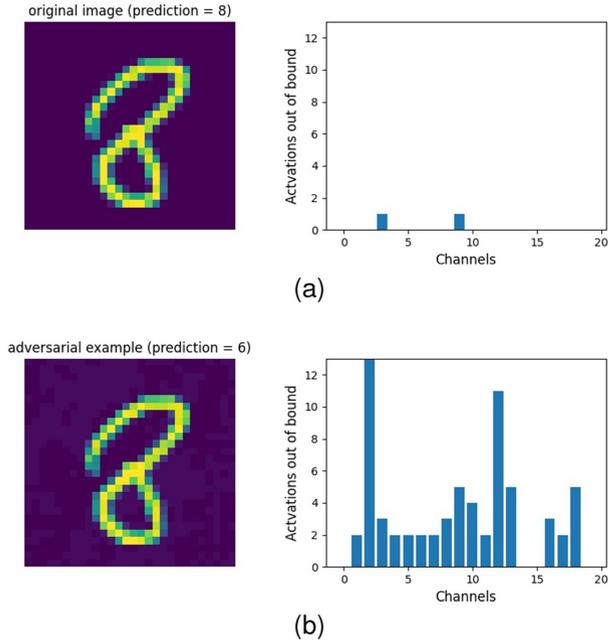
Fig. 3. (a) A genuine sample and its corresponding activation histogram with respect to a DNN Signature obtained in the first convolutional block of LeNet, under the SRC. (b) An adversarial example and its corresponding activation histogram against the same DNN Signature.

denoted as G, since k denotes the layer index. Hence, the G vectors $V_k(x)$ stored in the DNN Signature that are nearest to $V_k(x_{new})$ are identified by the kNN algorithm and their corresponding output class indexes are recorded and stored in a multiset Ω_k. Finally, the number of occurrences of the class index $\hat{y}$ in Ω_k is counted and denoted as η. Hence, the output confidence score is computed as $c = exp(-\frac{\eta \cdot ln(2)}{\tau_{\hat{y}}})$. The rationale behind this computation is the following. Given inputs x classified to the $\hat{y}$th class by the DNN, the more vectors $V_k(x)$ in the signature are among the G closest ones to $V_k(x_{new})$, the more the active state produced by x_{new} resembles the ones produced by the trusted inputs that the DNN classifies as for x_{new}, hence positively contributing to the confidence value. The overall procedure is reported in Algorithm 6.

Algorithm 6. Confidence Evaluation of kNNC

Input x_{new}, DNN Signature σ, trained DNN, thresholds τ_i, number of nearest neighbors G
Output confidence c

$\hat{y} = \text{argmax}_{1 \leq j \leq m}\{f(x_{new})\}$
$\eta = 0$
for $L_k \in L$ **do**
    $\Omega_k = \text{kNN}(G, V_k(x_{new}), \sigma_{1,k}, \ldots, \sigma_{m,k})$
    $\eta += |y \in \Omega_k : y = \hat{y}|$
end
return $c = exp(-\frac{\eta \cdot ln(2)}{\tau_{\hat{y}}})$

5.4 Illustrative Example

To illustrate the proposed approach, let us consider a simple example based on an inspired LeNet-4 [44] CNN trained on the MNIST dataset [45]. The Trusted Set consists of those MNIST samples that are correctly predicted by the network
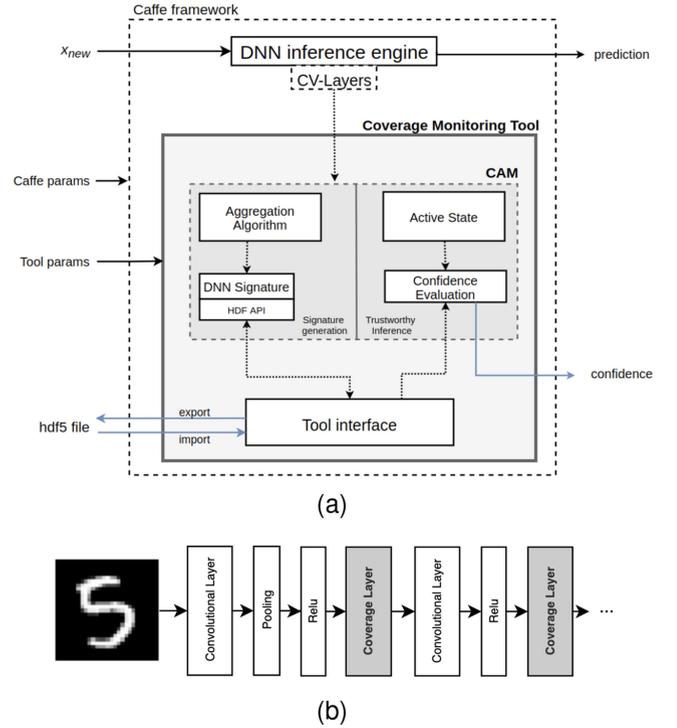

Fig. 4. (a) Overview of the tool architecture. (b) Example network architecture with two CV-Layers installed after two ReLu activation layers.

with a prediction score higher than 0.9. The selected CAM is the SRC presented in Section 5.1. Unsafe inputs are generated using the FGSM [1], setting $\epsilon = 0.05$ to obtain AEs with small perturbations.

For simplicity, the SRC method is only applied to the first convolutional block, including a convolutional layer, a pooling layer, and a ReLU activation layer. Fig. 3a illustrates a trusted input (i.e., a digit correctly classified as '8' with a prediction score of 0.974), which is not part of the Trusted Set, together with a histogram that reports the number of neurons per channel that have an output value outside the corresponding ranges $[v_{i,k,j}^{min}, v_{i,k,j}^{max}]$ representing the DNN Signature for class '8' under the SRC. In particular, note that only two neurons have outputs that do not match the signature, producing a prediction confidence $c = \exp(-\frac{2 \cdot ln(2)}{\tau_8}) = 0.8705$, where in this example all the thresholds τ_i are set to 10. Conversely, Fig. 3b illustrates an adversarial example, incorrectly classified as '6' (with a prediction score of 0.918), together with the corresponding histogram. In this case, there are 63 neurons in the first block whose activations are outside the ranges stored in the DNN Signature for label '6', resulting in a prediction confidence $c = \exp(-\frac{63 \cdot ln(2)}{\tau_6}) = 0.0126$.

6 IMPLEMENTATION IN CAFFE

The proposed architecture has been implemented as an extension of the Caffe framework [46] — see Fig. 4a. The extension introduces a new type of layer in Caffe, called *Coverage Layer* (CV-Layer), which operates in a transparent (i.e., pass-through) fashion, hence not altering neither the DNN performance nor the DNN hyperparameters. The CV-Layer applies a coverage criterion at inference time based on the tensor data it receives in input and is used during

both the offline and online phases. When installing a CV-Layer, given that it operates as a pass-through component, connections between two layers, say L_k and L_{k+1}, can be preserved by connecting (i) the output of L_k to the input of the CV-Layer and (ii) the output of the CV-Layer to the input of L_{k+1}. Multiple CV-Layers can be installed depending on the behavior of the selected CAM. For instance, CAMs that work by analyzing all neuron outputs of the network require the installation of a CV-Layer after each layer of the original DNN. An example installation of the CV-Layer is illustrated in Fig. 4b.

Fig. 4a illustrates the interconnection between the principal software blocks of the tool and the Caffe framework. Under Caffe, networks are distributed via a `prototxt` model file, which should include the CV-layers in the network architecture, and a `caffemodel` file, which contains the trained weights (both referred to as 'Caffe params' in the figure). Regarding the tool, the required parameters (referred to as 'Tool params' in the figure) are a specification of the Trusted Set and the identifier of the selected CAM (to be chosen among those supported by the tool). The tool will then instantiate the implementation of the monitoring functions associated to the selected CAM. Furthermore, CAM-specific parameters can be specified (see Section 5). Finally, the tool is capable of exporting and importing the DNN Signature using the hdf5 file format by means of the hierarchical data format (HDF) API. The tool provides an interface to invoke monitoring operations. It manages the interaction with the DNN Signature, as the export at the end of the offline phase, or the import at the beginning of the online phase. The tool also allows reading the confidence evaluation outcome during the online phase, which represents the coverage metric of the applied CAM.

Each CAM is developed both in CUDA and C++. Since the CV-Layers are totally compliant with the other Caffe layers, the DNN framework will take care of forwarding tensors to the selected implementation based on the available architecture (e.g., GPU or CPU). At the same time, it is up to the tool to select the device implementation of the Confidence Evaluation Algorithms, which are performed at the end of the DNN inference.

Another feature offered by the tool is the possibility of computing the signature in the offline phase using a mini-batch approach. The Trusted Set is split into multiple mini-batches where each one represents a single, but large input tensor that is analyzed by the CV-Layer on the same inference pass. The values extracted from the inputs of the mini-batch, according to the selected CAM, are then merged into the DNN Signature with the results obtained by previously-processed mini-batches. This approach significantly allows speeding up the creation of the DNN Signature creation time, especially when it is possible to parallelize the procedure.

7 EXPERIMENTAL EVALUATION

This section reports the results of a set of experiments conducted to assess the performance of the proposed methods in terms of running time, memory footprint, and capability of detecting unsafe network inputs. Several types of unsafe inputs generated with state-of-the-art methods have been considered: to make the paper self-contained, they are summarized in Section 7.1 and described in details in the supplementary material, available online. Before discussing the actual experimental results, Section 7.2 presents the experimental setting, whiles Section 7.3 presents the calibration of thresholds used.

7.1 Unsafe Inputs Generation

In recent years, many adversarial attacks [47] have been disclosed to efficiently generate AEs while minimizing the perturbation to be applied to turn a safe input into an unsafe one. Besides AEs, a network can also fail for other types of unsafe inputs. A recap of all the methods to generate unsafe inputs tested during our evaluation is reported next, while their complete description is provided in the supplementary material, available online, accompanied by illustrative examples.

Adversarial Examples. We considered the following attack methods under different settings for crafting both small and medium perturbations. The amount of perturbation is usually refereed with the ϵ parameter. Among the first and famous adversarial attacks, one is FGSM [1], which crafts adversarial perturbation in one shot. More advanced adversarial attacks have also been considered: they are PGD [48], BIM [2] and CW [4], which implement iterative methods capable of improving the attack effectiveness at each iteration. The number of iterations and the amount of perturbation introduced at each iteration are denoted by k and α, respectively. The attacks are configured as reported in Table 2.

Out-of-Distribution Unsafe Inputs. Another class of unsafe inputs are those that are far from the distribution of data used to train the network, but are still predicted with a high score by the model. They commonly exist because a DNN works as a global classifier for the whole input space (e.g., all possible images of a given size and format). To test this class of unsafe inputs, we generated new samples with large perturbations, obtained with a variant of the FGSM attack, which are applied to samples in the training set of the tested networks. Details on the generations are reported in the supplementary material, available online.

Adversarial Patches. Finally, we also studied another famous class of adversarial images, namely those using patches to make them adversarial. Adversarial patches are crafted with the PGD [48] method, but perturbing a limited area of the image only.

7.2 Experimental Setting

The datasets considered in the experimental evaluation are MNIST [45] and F-MNIST [49]. MNIST is a dataset of hand-written digits while F-MNIST is a dataset of clothes articles. Both datasets contain grayscale images of 28x28 pixels, and provide 60,000 images for training and 10,000 images for testing. F-MNIST is slightly more complex than MNIST and DNNs usually achieve lower accuracy in classifying its images. Both the datasets have been processed with a LeNet-4 [44] CNN trained on 8 epochs using the Adam optimization algorithm [50] and a cross entropy loss function.

Please note that the related works discussed in Section 2 that adopt a procedural-based internal analysis (e.g., [26],

TABLE 1
Network Models Applied for the Experimental Evaluation

Dataset	Model	Accuracy
MNIST	conv(20,5,1) - ReLU - **CV** - MaxPool(2,2) - conv(50,5,1) - ReLU - **CV** - fc(500) - ReLU - **CV** - fc(10)	0.9907
F-MNIST	conv(20,5,1) - ReLU - **CV** - MaxPool(2,2) - conv(50,5,1) - ReLU - **CV** - fc(500) - ReLU - **CV** - fc(10)	0.9106

'CV' indicates a Coverage Layer.

[31]), performed the experimental evaluation on small datasets (e.g., MNIST and CIFAR10) and DNNs with similar complexity to the ones mentioned above. The use of larger datasets and more complex network models will be part of a future work.

Three Coverage Layers per network have been installed (other network models with different installations of Coverage Layers are discussed later). Table 1 summarizes the network architectures and their corresponding classification accuracy on the original testing sets.

The experimental evaluation used the following sets of inputs:

- *Trusted Set*: it is the one used to generate the DNN Signature and has been obtained by selecting from the original *training set* those samples that the DNN classifies correctly with a prediction score (i.e., the softmax probability) larger than 0.9. This set contains 59309 and 52680 samples for MNIST and F-MNIST, respectively.
- *Trusted Test Set*: it is the one used to assess the classification performance of the DNNs when enhanced with one of the proposed CAMs. It is obtained by selecting from the original *testing set* those samples that the DNN classifies correctly with a prediction score > 0.9. This set contains 9000 samples.
- *Adversarial Set*: it is the one employed for evaluating the performance of the proposed methods in detecting unsafe inputs. It contains unsafe inputs for which the DNN makes a wrong prediction. Different definitions of this set have been tested depending on the selected attack method (see Section 7.1): they are

summarized in Table 2 for both MNIST and F-MNIST. A prediction is considered to be wrong when the DNN classifies such inputs in a wrong class with a prediction score larger than 0.8, for AEs and adversarial patches, or larger than 0.99, for out-of-distribution inputs. Such thresholds have been selected by empirically finding the largest values that allowed generating a sufficient number of unsafe inputs to perform the experiments. Note that the higher the thresholds the more difficult the generation of AEs. Conversely, when using lower thresholds, the generated AEs tend to be less relevant, as they could be simply discarded by comparing the softmax score produced by the network against a certain threshold. Further details are provided in the supplementary material, available online.

In the following, the experimental results are reported for four CAMs: SRC, two versions of MRC with $Q = \{16, 32\}$, and kNNC with $G = 75$. Furthermore, we also compared the results against state-of-the-art run-time methods used for detecting AEs: FeaturesSqueezing [19] and VisionGuard [20]. Details on their behavior and implementation are provided in the supplementary material, available online.

7.3 Threshold Calibration

This experimental evaluation was focused on a binary classification of the network inputs, i.e., either an input was deemed *safe* and the prediction made by the network was accepted, or the input was deemed *unsafe*, and the network prediction was rejected. This has been implemented by calibrating the tolerances τ_i used by the various Confidence Evaluation Algorithms so that an input is deemed safe if $c \geq 0.5$ and unsafe otherwise.

The calibration of thresholds has been performed using Receiver Operating Characteristic (ROC) analysis to compute the values τ_i, for each class with index $i = 1, \ldots, m$, that represent the best balance between minimizing the inputs that are wrongly rejected and accepted. In this regard, portions of the Trusted Test Set and the Adversarial Set introduced in the previous section have been used to

TABLE 2
Parameters and Settings Used to Generate the Adversarial Set

Attack method	MNIST			F-MNIST		
	Attack parameters	# of samples for calibration	# of samples for evaluation	Attack parameters	# of samples for calibration	# of samples for evaluation
FGSM-1	$\epsilon = 0.1$	500	500	$\epsilon = 0.05$	600	3400
FGSM-2	$\epsilon = 0.2$	800	3200	$\epsilon = 0.1$	600	1400
PGD-1	$(\epsilon, \alpha, k) = (0.1, 0.015, 40)$	800	1800	$(\epsilon, \alpha, k) = (0.03, 0.015, 40)$	600	3400
PGD-2	$(\epsilon, \alpha, k) = (0.18, 0.015, 40)$	800	3200	$(\epsilon, \alpha, k) = (0.10, 0.015, 40)$	600	3400
BIM-1	$(\epsilon, \alpha, k) = (0.05, 0.004, 10)$	450	500	$(\epsilon, \alpha, k) = (0.015, 0.004, 10)$	600	1500
BIM-2	$(\epsilon, \alpha, k) = (0.18, 0.004, 10)$	800	3200	$(\epsilon, \alpha, k) = (0.10, 0.004, 10)$	600	3400
CW	$k = 500$	450	500	$k = 500$	200	300
Out of Dis.	$\epsilon = 0.02, k = 80$	-	4000	$\epsilon = 0.02, k = 80$	-	4000
Patch	$\epsilon = 0.02, k = 200$	-	100	$\epsilon = 0.02, k = 200$	-	1000

TABLE 3
Detection Accuracy of All the Tested Methods on MNIST and F-MNIST

Input type	MNIST						F-MNIST					
	SRC	MRC-32	MRC-16	kNNC	VG	FS	SRC	MRC-32	MRC-16	kNNC	VG	FS
FGSM-1	0.998	0.976	0.946	0.958	0.986	0.976	0.855	0.848	0.827	0.915	0.880	0.881
FGSM-2	1.0	0.994	0.989	0.941	0.930	0.936	0.929	0.952	0.937	0.883	0.785	0.814
PGD-1	0.989	0.968	0.943	0.970	0.989	0.976	0.852	0.841	0.792	0.935	0.901	0.973
PGD-2	1.0	0.919	0.852	0.955	0.810	0.882	0.979	0.942	0.943	0.859	0.175	0.325
BIM-1	0.968	0.943	0.912	0.964	1.0	0.985	0.842	0.848	0.789	0.916	0.997	1.0
BIM-2	0.995	0.861	0.815	0.952	0.8425	0.870	0.986	0.924	0.939	0.831	0.161	0.239
CW	0.971	0.959	0.933	0.960	1.0	0.989	0.813	0.829	0.822	0.907	0.993	1.0
Out of Dis.	1.0	1.0	1.0	0.346	0.0	0.0	1.0	1.0	1.0	0.842	0.00	0.00
Patch	1.0	0.891	0.923	0.995	0.957	0.984	1.0	0.911	0.858	0.951	0.962	0.975
Safe Samples	0.991	0.951	0.919	0.955	0.925	0.938	0.89	0.892	0.851	0.921	0.671	0.732

VG and FS correspond to Vision Guard [20] and FeatureSqueezing [19], respectively.

test safe and unsafe inputs, respectively, during calibration. A more detailed discussion of the calibration phase and an illustration of the obtained ROC curves is presented in the supplementary material, available online.

7.4 Detection Performance
Experiments have been conducted to evaluate the performance of the four CAMs in correctly detecting whether an input is safe or unsafe using the thresholds τ_i calibrated as described above. Tables 3 reports the detection accuracy, defined as the ratio of inputs *correctly* classified as either safe or unsafe. The tables report the breakdown of unsafe inputs for all the methods introduced in Section 7.1 (see Table 2 for the number of samples used for the evaluation). The last rows of the tables are related to safe inputs, which correspond to the remaining inputs of the Trusted Test Set that have not been considered for calibration.

Although SRC is the simplest CAM, for inputs of the MNIST dataset it is often capable of detecting unsafe inputs better than the others CAMs. The only exception pertains to the BIM attack, but the performance of SRC is anyway extremely close to the one of kNNC. Note that SRC and MRC are also capable of detecting all unsafe out-of-distribution inputs, while kNNC exhibits very poor performance (close to coin tossing) for that kind of inputs.

The results for the F-MNIST dataset are quite different. In this case, there is no dominating CAM for AEs. Note that kNNC performs quite well with AEs generated using low perturbations (e.g., CW and BIM), while SRC significantly outperforms kNNC for unsafe inputs generated using the PGD attack. SRC and MRC are again capable of detecting all unsafe out-of-distribution inputs. This suggests that, given proper accelerated implementations, multiple CAMs could be used in parallel to improve the overall detection performance of unsafe inputs.

Table 3 also provides the results achieved with Vision Guard and FeatureSqueezing. They always obtain high performance with low-perturbation attacks on both MNIST and FMNIST, but fail when the magnitude of the perturbation increases (e.g., PGD-2 and BIM-2, which find more effective AEs). Moreover, they report worse performance on safe samples, since they are incorrectly rejected more times than the proposed CAMs. Most interestingly, Vision Guard and FeatureSqueezing totally fail with out-of-distributions

unsafe inputs, while three of the proposed CAMs are very effective.

Finally, to further evaluate the detection performance of the CAMs, additional tests are provided in the supplementary materials, available online, to better evaluate the relationship between the CAMs and the perturbations of AEs.

7.5 Running Time and Memory Footprint
With respect to a regular deployment of a DNN, each of the proposed CAMs introduces an additional inference time to execute the Confidence Evaluation Algorithm (implemented by CV-Layers) and an additional memory footprint to store the DNN Signature. This section is focused on evaluating these overheads, which in some cases may be very important in selecting the most appropriate CAM. For instance, to deploy a CAM on a resource-constrained, embedded device that has to operate in real time, it is essential to contain as much as possible the additional inference time and the memory footprint, even accepting reduced detection performance.

Additional Inference Time. The experiments for SRC, MRC, and NRC have been performed on a machine equipped with an Intel(R) Core(TM) i5-4670K CPU @ 3.40GHz, 8GB of RAM, and an NVIDIA GeForce GTX770 GPU. Due to their demanding memory requirements (see the results at the end of this section), the experiments for kNNC have been performed on a Nvidia DGX Station V100. The implementation presented in Section 6 has been used.[1] Fig. 5 reports the additional inference time introduced by the four CAMs together with the corresponding detection performance for a representative setting (F-MNIST dataset and unsafe inputs generated by the FGSM-2 attack). Note that the y-axis at the left of the figure has a logarithmic scale. The figure reports the results for different installations of the CV-Layers, denoted by the sets reported on the x-axis of the plot. They correspond to cases in which the CAMs operate on a subset of the entire set of CV-Layers reported in Table 1. The

---

1. Note that, due to implementation issues under resolution in integrating kNNC in our CUDA-based implementations of CV-Layers, the experiments for kNNC have been performed with a Python implementation of the tool presented in Section 6 that uses the Pytorch framework [51] (with GPU acceleration) and FAISS [52], which is notably the best-performing library that implements a GPU-accelerated exact nearest neighbors search.
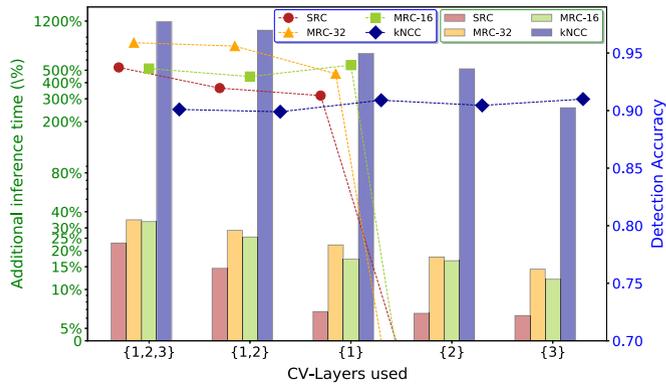
Fig. 5. Timing performance and detection accuracy for five installations of the CV-Layers. The sets on the x-axis denote which of the CV-Layers reported in Table 1 are installed (numbered in the order with which they appear in the table). For instance, $\{1, 2\}$ refers to the case in which only the first two CV-Layers are installed.

calibration procedure presented in Section 7.3 has been performed for each of them (i.e., each installation is assigned different thresholds).

As it can be noted from the figure, when all the 3 CV-Layers are installed ($\{1, 2, 3\}$ on the x-axis) the percentage increase of the interference time (with respect the original network inference) is of the same order of magnitude for SRC, MRC-16, and MRC-32. As one may easily expect, the best timing performance is achieved by SRC, with a percentage increase of 22.9%. The additional inference time introduced by kNNC is instead quite high and corresponds to a 1188% increase.

Interesting observations can be made by looking at the accuracy of the various installations and CAMs: SRC and MRC exhibit very poor detection performance if the first CV-Layer is not installed, while kNNC exhibits only slights variations of the detection performance when less CV-Layers are installed. Most interestingly, note that the detection performance may even increase when less CV-Layers are installed. For instance, these results reveal that it is not convenient to adopt kNNC with the first two CV-Layers installed, as the same detection performance can be achieved with just a 255% increase of the inference time, rather than a 1188% increase for the case in which all CV-Layers are installed.

Memory Footprint. Table 4 reports the memory footprint of the DNN Signature of the various CAMs for each CV-Layer. As it can be noted from the table, SRC and NRC have a very modicum memory footprint (in the order of 1MB in total), while kNNC is characterized by a huge memory footprint that even exceeds 3GB if all the three CV-Layers are installed. These results, together with the ones of Fig. 5,

TABLE 4
Memory Footprint of the DNN Signature (Exported HDF File) of the Various CAMs for Each CV-Layer

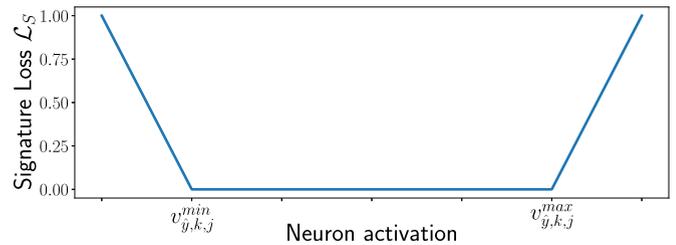|	CV-Layer1	CV-Layer2	CV-Layer3
SRC	944 kB	278 kB	62,8 kB
MRC-16	8,3 MB	2,3 MB	387 kB
MRC-32	15,7 MB	4,4 MB	707 kB
NRC	483,6 kB	150,8 kB	42,8 kB
kNNC	2.6 GB	732 MB	114 MB

Fig. 6. Illustration of the *Loss-Signature* for the case of a single neuron.

confirm that under kNNC it does not worth to install CV-Layers between shallower layers of the network, and that SRC is a very good choice to balance performance with overheads. The results for the kNNC may vary if different (e.g., approximate) methods to implement the nearest neighbors search are adopted.

Comparison With Other Detection Techniques. The average running times of both VG and FS were also measured. They introduce an additional latency of 138% and 237%, respectively, which correspond to the running time of multiple inferences, plus an additional overhead introduced by the used transformations. As far memory footprint is concerned, they are very lightweight approaches only when the multiple inferences they require are sequentially performed (as in the evaluated setting). Conversely, if they are performed in parallel, the overall memory footprint grows as a function of the amount of space needed for instantiating and running multiple replicas of the network model.

7.6 Adaptive Attack and Countermeasure

Although the presented CAMs are able to detect several kinds of unsafe inputs, ad-hoc attacks can still be devised to optimize unsafe inputs by directly exploiting the knowledge of the detection mechanism and the signatures.

Inspired by the famous gradient-based attacks (which are formally discussed in the supplementary material, available online), we designed a white-box attack method, called *Signature-Attack*, for crafting adversarial inputs while trying to keep the activation of neurons within the range of the signatures. This is accomplished by optimizing the adversarial perturbation through two loss functions, $\mathcal{L}_{CE}$ and $\mathcal{L}_S$. The former is the common cross-entropy loss, also involved in all the other attacks tested in this work. The optimization process shall maximize such a loss to intensify the adversarial effect of the final perturbation (i.e., increase the probability of misclassifying the perturbed input). The latter is a problem-specific loss function, named *Signature-Loss*, which is conceived to return a positive cost when the activation of a neuron is outside the signature range $[v_{\hat{y},k,j}^{\min}, v_{\hat{y},k,j}^{\max}]$, where $\hat{y}$ is the label associated by the network to the input to be perturbed. Technically speaking, the Signature-Loss is implemented as $\mathcal{L}_S = \sum_{n_{k,j} \in N} \text{ReLU}(v_{k,j} - v_{\hat{y},k,j}^{\max}) + \text{ReLU}(-v_{k,j} + v_{\hat{y},k,j}^{\min})$, where $\text{ReLU}(x) = max(0, x)$. Fig. 6 provides a sample illustration of function $\mathcal{L}_S$ in the case of a single neuron only. The Signature-Loss has to be minimized during the optimization process to reduce the chance of the adversarial example being detected by the proposed CAMs.

From a practical point of view, the optimization problem to accomplish the Signature-Attack is solved through an
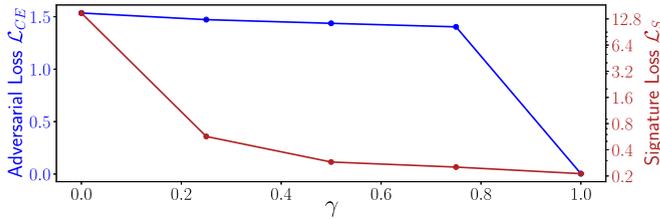
Fig. 7. Average losses computed with inputs from the MNIST data set when attacked with the Signature-Attack, under different configurations of the control parameter γ.

TABLE 5
Detection Accuracy on FMNIST and MNIST Against Adversarial Examples Crafted With the Signature-Attack Under Four Configurations

γ	MNIST			F-MNIST		
	SRC	MRC-32	MRC-16	SRC	MRC-32	MRC-16
0.0	0.984	0.957	0.935	0.968	0.908	0.912
0.25	0.779	0.775	0.701	0.881	0.805	0.785
0.50	0.756	0.769	0.690	0.879	0.797	0.781
0.75	0.749	0.742	0.687	0.874	0.793	0.778

iterative formulation similar to [48], but based on a specific loss function $\mathcal{L}$ that takes into account both $\mathcal{L}_{CE}$ and $\mathcal{L}_S$

$$x'_k = x'_{k-1} + \alpha \cdot sign(\nabla_{x'_{k-1}} \mathcal{L}(f(x'_{k-1}), \hat{y})),$$
$$s.t. \quad ||x'_k - x|| \le \epsilon, \quad (3)$$

where α and ϵ denote the step size and the overall perturbation magnitude, respectively, as in [48]. The gradient of the loss function $\mathcal{L}$ is defined as

$$\nabla_x \mathcal{L} = (1 - \gamma) \cdot \frac{\nabla_x \mathcal{L}_{CE}}{|| \nabla_x \mathcal{L}_{CE} ||_2} - \gamma \cdot \frac{\nabla_x \mathcal{L}_S}{|| \nabla_x \mathcal{L}_S ||_2}, \quad (4)$$

where γ is a parameter introduced to balance the importance of $\mathcal{L}_{CE}$ and $\mathcal{L}_S$. Since the gradients of $\mathcal{L}_{CE}$ and $\mathcal{L}_S$ may have very different scales, they are both subject to *norm-2* normalization to ensure that their effect can be properly balanced using a single parameter γ. Furthermore, it is also important to observe how the overall gradient of $\mathcal{L}$ goes in the same direction of $\nabla_x \mathcal{L}_{CE}$ and in the opposite direction of $\nabla_x \mathcal{L}_S$. Thus, optimizing the perturbation in the gradient direction of $\mathcal{L}$ means increasing the classification loss function $\mathcal{L}_{CE}$ while reducing the Signature-Loss $\mathcal{L}_S$.

Effects of the Signature-Attack. Multiple versions of the Signature-Attack were tested on both MNIST and FMNIST. Similar to the settings used in Table 2, we set α and ϵ to 0.004 and 0.1, respectively, and we evaluated five values for γ: 0.0, 0.25, 0.50, 0.75, and 1.0. The first and last values were tested for studying the behavior of the proposed attack in

two limit cases (i.e., when only one of the two loss functions is considered).

Fig. 7 reports the average classification loss $\mathcal{L}_{CE}$ and Signature-Loss $\mathcal{L}_S$ computed on the MNIST data set for the tested values of γ. Note that the classification loss (to be maximized) preserves a large value for the tested values of $\gamma < 1$, while the Signature-Loss (to be minimized) has a considerably low value for the tested values of $\gamma \ge 0.5$.

The effectiveness of the generated adversarial examples can be observed from the results reported in Table 5 as a function of γ (the results for $\gamma = 1.0$ are omitted since it was not possible to generate adversarial examples with this setting). As it can be noted from the table, the detection performance of the proposed CAMs is reduced, with respect to the results of Table 3, on both MNIST and FMNIST, demonstrating the effectiveness of the Signature-Attack with $\gamma > 0$.

Countermeasure. To counteract the Signature-Attack, we included a portion of the adversarial examples crafted with that attack to the calibration set in order to re-calibrate the thresholds. This approach helps improve the mechanism by tuning the thresholds at lower values such that also CAM-specific adversarial inputs can be detected without significantly reducing the accuracy of all the others.

To this purpose, we refined the set of inputs used for calibration (see Section 7.3) by adding 800 adversarial examples obtained by the Signature-Attack with $\gamma \in \{0.25, 0.75\}$. Other 8000 adversarial examples (2000 for each $\gamma \in \{0.0, 0.25,$

TABLE 6
Detection Accuracy After the Re-Calibration, Performed With Also Unsafe Inputs Crafted From the Proposed Signature-Attack

Input type	MNIST				F-MNIST			
	SRC	MRC-32	MRC-16	kNNC	SRC	MRC-32	MRC-16	kNNC
FGSM-1	1.0	0.986	0.962	0.969	0.839	0.845	0.829	0.922
FGSM-2	1.0	0.995	0.991	0.951	0.925	0.948	0.922	0.891
PGD-1	0.991	0.953	0.887	0.969	0.849	0.842	0.804	0.939
PGD-2	1.0	0.977	0.960	0.953	0.961	0.889	0.874	0.868
BIM-1	0.993	0.910	0.953	0.923	0.839	0.847	0.792	0.919
BIM-2	0.999	0.967	0.940	0.949	0.980	0.921	0.924	0.844
CW	0.987	0.977	0.953	0.960	0.813	0.825	0.822	0.907
Out of Dis.	1.0	1.0	1.0	0.34	1.0	1.0	1.0	0.851
Patch	1.0	0.889	0.921	0.988	1.0	0.907	0.853	0.954
Signature-Attack ($\gamma = 0.0$)	0.998	0.896	0.841	0.973	0.963	0.889	0.884	0.851
Signature-Attack ($\gamma = 0.25$)	0.949	0.939	0.782	0.972	0.935	0.861	0.852	0.852
Signature-Attack ($\gamma = 0.50$)	0.939	0.831	0.767	0.971	0.933	0.861	0.852	0.853
Signature-Attack ($\gamma = 0.75$)	0.925	0.820	0.763	0.971	0.931	0.858	0.847	0.851
Safe Samples	0.986	0.938	0.89	0.957	0.890	0.886	0.854	0.916

0.50, 0.75}) are appended to the test set introduced in Table 2. Table 6 shows the final results obtained after the re-calibration, i.e., by testing the detection mechanisms with the new thresholds. By comparing Table 6 with Table 5, it can be noted that the detection accuracy of the proposed CAMs is improved for all the tested versions of the Signature-Attack. Also note that the re-calibration of thresholds did not significantly affect the detection performance for the other kinds of unsafe inputs, which shows similar results with respect to Table 3.

8 CONCLUSION AND FUTURE WORK

This paper presented a monitoring architecture to enhance the trustworthiness of DNNs by generating a confidence value to be coupled to the network prediction. The workflow of the proposed architecture includes an off-line phase (to be performed after training) where a DNN Signature is generated starting from a set of trusted inputs, and an online phase (to be performed at inference time), where the DNN Active State generated by a new input is compared against the DNN Signature. Four different CAMs have been proposed to instantiate the proposed architecture. An implementation as an extension of the Caffe framework has also been presented.

The proposed CAMs have been compared in terms of detection performance and running time, using several state-of-the-art methods to generate unsafe inputs. The experimental results on MNIST and F-MNIST data sets showed that the CAMs based on coverage techniques better generalize among different types of unsafe inputs. Vision-Guard [20] and FeatureSqueezing [19], which proved to have competitive performance with respect to most of the proposed CAMs for low-perturbation AEs, lose their effectiveness with medium-perturbation attacks and out-of-distribution unsafe inputs. Also in terms of running time and memory footprint, most of the proposed CAMs exhibit better performance than previous work, proving to be more suitable for real-time applications.

The approach proposed in this paper opens several interesting future work directions. Among them, a particularly relevant research line is the investigation of techniques that combine the proposed CAMs to enhance the detection performance while balancing the implied additional inference time and memory footprint. Research efforts should also be spent on the design of other CAMs and techniques to prune unnecessary information stored by the DNN Signature. Such developments will hopefully make possible to apply the proposed approach to more complex DNNs, such as ResNet [53] and Inception [54], and larger and sparser data sets, as ImageNet [55].

REFERENCES

[1] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *Proc. 3rd Int. Conf. Learn. Representations*, 2015.

[2] A. Kurakin, I. J. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," in *Artificial Intelligence Safety and Security*. London, U.K./Boca Raton, FL, USA: Chapman and Hall/CRC, 2018, pp. 99–112.

[3] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *Proc. IEEE Eur. Symp. Secur. Privacy*, 2016, pp. 372–387.

[4] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *Proc. IEEE Symp. Secur. Privacy*, 2017, pp. 39–57.

[5] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial machine learning at scale," 2016, *arXiv:1611.01236*.

[6] K. Pei, Y. Cao, J. Yang, and S. Jana, "DeepXplore: Automated whitebox testing of deep learning systems," in *Proc. 26th Symp. Oper. Syst. Princ.*, 2017, pp. 1–18.

[7] U. Shaham, Y. Yamada, and S. Negahban, "Understanding adversarial training: Increasing local stability of supervised models through robust optimization," *Neurocomputing*, vol. 307, pp. 195–204, 2018.

[8] A. Sinha, H. Namkoong, R. Volpi, and J. Duchi, "Certifying some distributional robustness with principled adversarial training," 2017, *arXiv:1710.10571*.

[9] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, "Ensemble adversarial training: Attacks and defenses," 2017, *arXiv:1705.07204*.

[10] N. Papernot, P. D. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," 2015, *arXiv:abs/1511.04508*.

[11] G. Hinton et al., "Distilling the knowledge in a neural network," 2015, *arXiv:1503.02531*.

[12] A. Shafahi, W. R. Huang, C. Studer, S. Feizi, and T. Goldstein, "Are adversarial examples inevitable?," 2018, *arXiv:abs/1809.02104*.

[13] N. Carlini and D. Wagner, "Adversarial examples are not easily detected: Bypassing ten detection methods," in *Proc. 10th ACM Workshop Artif. Intell. Secur.*, 2017, pp. 3–14.

[14] M. Abbasi, A. Rajabi, A. S. Mozafari, R. B. Bobba, and C. Gagne, "Controlling over-generalization and its effect on adversarial examples generation and detection," 2018, *arXiv:1808.08282*.

[15] A. Mustafa, S. Khan, M. Hayat, R. Goecke, J. Shen, and L. Shao, "Adversarial defense by restricting the hidden space of deep neural networks," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 3384–3393.

[16] J. Wang, G. Dong, J. Sun, X. Wang, and P. Zhang, "Adversarial sample detection for deep neural network through model mutation testing," in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng.*, 2019, pp. 1245–1256.

[17] S. Srisakaokul, Y. Zhang, Z. Zhong, W. Yang, T. Xie, and B. Li, "MulDef: Multi-model-based defense against adversarial examples for neural networks," 2018, *arXiv:1809.00065*.

[18] B. Biggio et al., "Evasion attacks against machine learning at test time," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discov. Databases*, 2013, pp. 387–402.

[19] W. Xu, D. Evans, and Y. Qi, "Feature squeezing: Detecting adversarial examples in deep neural networks," in *Proc. 25th Annu. Netw. Distrib. Syst. Secur. Symp.*, 2018, pp. 1–15.

[20] Y. Kantaros et al., "VisionGuard: Runtime detection of adversarial inputs to perception systems," 2020, *arXiv:abs/2002.09792*.

[21] F. Nesti, A. Biondi, and G. Buttazzo, "Detecting adversarial examples by input transformations, defense perturbations, and voting," *IEEE Trans. Neural Netw. Learn. Syst.*, to be published.

[22] P. Samangouei, M. Kabkab, and R. Chellappa, "Defense-GAN: Protecting classifiers against adversarial attacks using generative models," in *Proc. 6th Int. Conf. Learn. Representations*, 2018.

[23] D. Meng and H. Chen, "MagNet: A two-pronged defense against adversarial examples," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 135–147.

[24] X. Huang et al., "A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability," *Comput. Sci. Rev.*, vol. 37, 2020, Art. no. 100270.

[25] T. B. Brown, D. Mané, A. Roy, M. Abadi, and J. Gilmer, "Adversarial patch," 2017, *arXiv:abs/1712.09665*.

[26] N. Papernot and P. D. McDaniel, "Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning," 2018, *arXiv:abs/1803.04765*.

[27] M. Sensoy, L. M. Kaplan, and M. Kandemir, "Evidential deep learning to quantify classification uncertainty," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 3183–3193.

[28] U. Ozbulak, W. D. Neve, and A. V. Messem, "How the softmax output is misleading for evaluating the strength of adversarial examples," 2018, *arXiv:abs/1811.08577*.

[29] N. Carlini and D. A. Wagner, "Magnet and "efficient defenses against adversarial attacks" are not robust to adversarial examples," 2017, *arXiv:abs/1711.08478*.

[30] X. Li, Y. Chen, Y. He, and H. Xue, "AdvKnn: Adversarial attacks on K-nearest neighbor classifiers with approximate gradients," 2019, *arXiv:abs/1911.06591*.

[31] A. Sotgiu *et al.*, "Deep neural rejection against adversarial examples," *EURASIP J. Inf. Secur.*, vol. 2020, 2020, Art. no. 5.

[32] F. Carrara, R. Becarelli, R. Caldelli, F. Falchi, and G. Amato, "Adversarial examples detection in features distance spaces," in *Proc. Workshops Comput. Vis.*, 2018, pp. 313–327.

[33] E. Wong and J. Z. Kolter, "Provable defenses against adversarial examples via the convex outer adversarial polytope," in *Proc. 35th Int. Conf. Mach. Learn.*, 2018, pp. 5283–5292.

[34] T. Weng *et al.*, "Evaluating the robustness of neural networks: An extreme value theory approach," in *Proc. 6th Int. Conf. Learn. Representations*, 2018.

[35] M. Hein and M. Andriushchenko, "Formal guarantees on the robustness of a classifier against adversarial manipulation," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 2266–2276.

[36] M. Borg *et al.*, "Safely entering the deep: A review of verification and validation for machine learning and a challenge elicitation in the automotive industry," 2018, *arXiv:abs/1812.05389*.

[37] H. Zhu, P. Hall, and J. May, "Software unit test coverage and adequacy," *ACM Comput. Surv.*, vol. 29, pp. 366–427, 1997.

[38] L. Ma *et al.*, "DeepGauge: Comprehensive and multi-granularity testing criteria for gauging the robustness of deep learning systems," 2018, *arXiv:abs/1803.07519*.

[39] K. J. Hayhurst, D. S. Veerhusen, J. J. Chilenski, and L. K. Rierson, "A practical tutorial on modified condition/decision coverage," NASA Langley Tech. Rep. Server, 2001. [Online]. Available: https://dl.acm.org/doi/10.5555/886632

[40] Y. Sun, X. Huang, and D. Kroening, "Testing deep neural networks," 2018, *arXiv:abs/1803.04792*.

[41] F. Harel-Canada , L. Wang, M. A. Gulzar, Q. Gu, and M. Kim, "Is neuron coverage a meaningful measure for testing deep neural networks?," in *Proc. 28th ACM Joint Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng. Virt. Event*, 2020, pp. 851–862.

[42] S. Yan *et al.*, "Correlations between deep neural network model coverage criteria and model quality," in *Proc. 28th ACM Joint Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng. Virt. Event*, 2020, pp. 775–787.

[43] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," 2015, *arXiv:abs/1511.08458*.

[44] L. Sun, Y. Wang, and L. Dai, "Convolutional neural network protection method of Lenet-5-like structure," in *Proc. 2nd Int. Conf. Comput. Sci. Artif. Intell.*, 2018, pp. 77–80.

[45] Y. LeCun, "The MNIST database of handwritten digits," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[46] Y. Jia *et al.*, "Caffe: Convolutional architecture for fast feature embedding," in *Proc. 22nd ACM Int. Conf. Multimedia*, 2014, pp. 675–678.

[47] C. Szegedy *et al.*, "Intriguing properties of neural networks," 2013, *arXiv:1312.6199*.

[48] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *Proc. 6th Int. Conf. Learn. Representations*, 2018.

[49] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms," 2017, *arXiv:abs/1708.07747*.

[50] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. 3rd Int. Conf. Learn. Representations*, 2015.

[51] A. Paszke *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 8024–8035.

[52] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with GPUs," *IEEE Trans. Big Data*, vol. 7, no. 3, pp. 535–547, Jul. 2021.

[53] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[54] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 2818–2826.

[55] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei , "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 2009, pp. 248–255.

Giulio Rossolini received the graduated (cum laude) degree in embedded computing systems engineering, and the master's degree jointly offered by the Scuola Superiore Sant'Anna of Pisa and the University of Pisa, Pisa, Italy. He is currently working toward the PhD degree in the Real-Time Systems (ReTiS) Laboratory, Scuola Superiore Sant'Anna of Pisa, Pisa, Italy. His current research interests include the design and implementation of software tools to support and increase the trustworthiness of machine learning algorithms used in computer vision applications and safety-critical systems.

Alessandro Biondi (Member, IEEE) received the graduated (cum laude) degree in computer engineering from the University of Pisa, Pisa, Italy, within the excellence program, and the PhD degree in computer engineering from the Scuola Superiore Sant'Anna, Pisa, Italy, under the supervision of Prof. Giorgio Buttazzo and Prof. Marco Di Natale. He is currently an assistant professor with the Real-Time Systems (ReTiS) Laboratory, Scuola Superiore Sant'Anna. In 2016, he has been visiting scholar with the Max Planck Institute for Software Systems (Germany). His research interests include design and implementation of real-time operating systems and hypervisors, schedulability analysis, cyber-physical systems, synchronization protocols, and safe and secure machine learning. He was recipient of six best paper awards, one Outstanding Paper Award, the ACM SIGBED Early Career Award 2019, and the EDAA Dissertation Award 2017.

Giorgio Buttazzo (Fellow, IEEE) received the graduate degree in electronic engineering from the University of Pisa, Pisa, Italy, the MS degree in computer science from the University of Pennsylvania, Philadelphia, Pennsylvania, and the PhD degree in computer engineering from the Scuola Superiore Sant'Anna of Pisa, Pisa, Italy. He is currently a full professor of computer engineering with the Scuola Superiore Sant'Anna of Pisa. He has been editor-in-chief of the *Real-Time Systems*, associate editor of the *ACM Transactions on Cyber-Physical Systems*. He has authored seven books on real-time systems and more than 300 papers in the field of real-time systems, robotics, and neural networks, receiving 13 best paper awards.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/csdl.