DEM: A Method for Certifying Deep Neural Network Classifier Outputs in Aerospace

Guy Katz
The Hebrew University of Jerusalem
Jerusalem, Israely
gkatz@mail.huji.ac.il

Natan Levy
The Hebrew University of Jerusalem
Jerusalem, Israely
natan.levy1@mail.huji.ac.il

Idan Refaeli
The Hebrew University of Jerusalem
Jerusalem, Israely
idan.refaeli@mail.huji.ac.il

Raz Yerushalmi
The Weizmann Institute of Science
Rehovot, Israel
The Hebrew University of Jerusalem
Jerusalem, Israely
raz.yerushalmi@weizmann.ac.il

2024 AIAA DATC/IEEE 43rd Digital Avionics Systems Conference (DASC) | 979-8-3503-4961-0/24/$31.00 ©2024 IEEE | DOI: 10.1109/DASC62030.2024.10748779

***Abstract*—Air transportation, a critical component of modern life, faces significant challenges concerning efficiency, environmental sustainability, and safety. Addressing these challenges necessitates innovative solutions, such as using deep neural networks (DNNs). However, although DNNs demonstrate remarkable performance, they remain susceptible to tiny perturbations in their inputs, which may result in misclassification. The vulnerability, known as adversarial inputs, may trigger a chain of events that could result in significant, or even catastrophic, failures. In this regard, adversarial inputs constitute a crucial obstacle to the integration of DNNs into safety-critical aerospace systems.**

In the present work, we introduce a novel, output-centric method for certifying DNNs that addresses this challenge. This method utilizes statistical techniques to flag out specific inputs for which the DNN's output might be unreliable so that a human expert may examine them. In contrast to existing techniques, which typically attempt to certify the entire DNN, the proposed method certifies specific outputs. Moreover, the proposed method uses the DNN as a black box and makes no assumptions about its topology.

We developed a proof-of-concept tool called *DEM* to demonstrate the feasibility of the proposed method. For evaluation, we tested the proposed method on a VGG-16 model, trained on the CIFAR16 dataset, showing that the proposed method achieved a high percentage of success in detecting adversarial inputs. We believe this work constitutes another step towards integrating deep neural networks in safety-critical applications — especially in the aerospace domain, where high standards of quality and reliability are crucial.

***Index Terms*—Aerospace Certification, Deep Neural Networks, Enable Monitor, Statistical Verification.**

This work was partially funded by the European Union (ERC, VeriDeL, 101112713). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them. The work was further partially funded by an NSFC-ISF grant to David Harel, issued jointly by the National Natural Science Foundation of China (NSFC) and the Israel Science Foundation (ISF grant 3698/21), and in part by the Minerva grant 714132. Additional support was provided by a research grant to DH from Louis J. Lavigne and Nancy Rothman, the Carter Chapman Shreve Family Foundation, Dr. and Mrs. Donald Rivin, and the Estate of Smigel Trust.

I. Introduction

Air transportation handles billions of passengers annually, and requires advanced solutions for addressing long-standing efficiency, environmental impact, and safety challenges [1]. The deep learning revolution shows significant potential for addressing these requirements, e.g., through the use of deep neural networks (DNNs) [2], [3]. However, although DNNs have demonstrated remarkable performance, they remain susceptible to tiny perturbations in their inputs, which might result in misclassification. This vulnerability, referred to as *adversarial inputs* [4], [5], could trigger a chain of events that might result in serious injury, extensive damage to valuable assets, or irreparable damage to delicate ecosystems [6]. Consequently, adversarial inputs pose a significant barrier to the integration of DNNs in a variety of fields, such as safety critical systems.

In the aerospace industry, certification authorities such as the European Aviation Safety Agency (EASA) and the Federal Aviation Administration (FAA) play a key role in advocating civil aviation safety and environmental preservation. These authoritative bodies acknowledge the immense value of incorporating DNNs into safety-critical applications [1], yet also emphasize that conventional software certification guidelines, such as DO-178 [7], are not applicable to DNNs. In recognition of the inherent challenges of certifying DNNs within the aerospace domain, EASA recently released its comprehensive AI roadmap [1], which outlines seven pivotal requirements for ensuring artificial intelligence trustworthiness. Of these, *robustness to adversarial conditions* is mentioned as one of the fundamental prerequisites. Thus, it is evident that ensuring a DNN's robustness to adversarial inputs will play a crucial role in any future certification process.

During the preliminary stage of an aerospace certification process, the SAE ARP-4754 guidelines [8] call for conducting a *safety analysis* procedure for all aircraft systems and subsystems. For each subsystem, the safety engineering team must determine the permissible probability of failure, with which the design engineer is required to comply. In order to carry out such a process for a system that incorporates a DNN, the engineering team must demonstrate that the probability of a failure related to the DNN does not exceed the threshold established in the safety analysis. Unfortunately, there is currently a shortage of techniques for performing this analysis that are both sufficiently precise and scalable.

Here, we seek to bridge this crucial gap by introducing a novel DNN certification approach, *DNN Enable Monitor* (*DEM*). Inspired by wireless network management concepts [9]–[11], instead of certifying the DNN as a whole, *DEM* focuses on certifying the concrete DNN's outputs. For each DNN output, *DEM* calculates during inference time the probability of misclassification — and if this probability is below a certain acceptable threshold, the output is considered reliable. Otherwise, the output is flagged and passed on for further analysis, e.g., by a human expert. The motivation here is that well-trained DNNs, working under normal conditions, are generally correct; and so, identifying the few cases where the DNN might be wrong, and further analyzing only these cases, can relieve a significant cognitive load off the human experts, compared to a manual approach. Further, if the automated certification is accurate enough, the entire process becomes nearly fully automatic — and also sufficiently accurate to meet the bars specified in the safety analysis phase. The proposed method is in line with other semi-automatic safety analysis and optimization methods for safety-critical applications [12].

In order to effectively distinguish between regular input and an adversarial input, *DEM* leverages the concept of *probabilistic global categorial robustness*, presented as part of the *gRoMA* method for statistically evaluating the global robustness of a classifier DNN, per output category [13]. To gain intuition, consider an input $\vec{x_0}$, which is mapped by the DNN to label l_0. If this classification is correct, most input points in the robustness region around $\vec{x_0}$ would also be classified as l_0 [13]. If, however, $\vec{x_0}$ is an adversarial input, which can intuitively be interpreted as a "glitch" in the DNN, then other points around $\vec{x_0}$ would be classified differently. *DEM* is used to effectively distinguish between these two cases.

For evaluation purposes, we created a proof-of-concept implementation of *DEM*, and used it to evaluate VGG16 [14] and Resnet [15] DNN models. Our evaluation shows that *DEM* outperforms state-of-the-art methods in adversarial input detection, and that it successfully distinguishes adversarial inputs from genuine inputs with a very high success rate in some categories. For these categories, success rates are sufficiently high to meet the aerospace regulatory requirements.

A major advantage of *DEM* is that it computes different thresholds for certifying outputs for the different classifier categories — making it flexible and accurate enough to handle cases where it is impossible to select a uniform threshold for all output categories. To the best of our knowledge, this is the first effort at certifying the categorial robustness of DNN outputs, for safety-critical DNNs.

The rest of the paper is organized as follows: We begin in Sec. II with a description of prior work, and then present the required background on adversarial robustness in Sec. III. In Sec. IV we describe our proposed method for measuring adversarial robustness, followed by a description of our evaluation in Sec. V. Finally, in Sec. VI, we summarize and discuss our results.

II. Related Work

The certification of DNNs and their integration into safety-critical applications has been the subject of extensive research [16]–[18]. In general, DNN certification rests on two main foundations: certification guidelines and DNN robustness.

Certification Guidelines. A widely accepted cornerstone in system and software certification within the aerospace domain are the "ARP-4754 — Guidelines for Development of Civil Aircraft Systems and Equipment Certification" guidelines [8]. These guidelines were authored by SAE International, a global association of aerospace engineers and technical experts. The guidelines focus on safety considerations when developing civil aircraft and their associated systems. However, these guidelines do not directly apply to DNN components.

A recent work [19] proposed an approach to assessing DNN robustness through an alternative Functional Hazard Analysis (FHA) method. This method seeks to show that the likelihood that a neural network performs unexpectedly is below an acceptable threshold, defined by the user, with a confidence level of 99%. *DEM*, on the other hand, measures the probability of a failure condition, given a specific input. *DEM* meets the probabilistic goals of the ARP-4754 guidelines objectives, and could potentially be used to certify DNN-based components without further adjustments.

Another recent study [16] developed a comprehensive framework of principles for DNN certification, which is aligned with ARP-4754 — but which does not include a concrete method or tool that can be used. While ARP-4754 focuses on system level certification, the influential DO-178 guidelines for airborne systems focus on software certification [7]. The DO-178 guidelines supplement ARP-4754's safety requirements with 5 levels (A-E) for software failure conditions. Initial work has discussed certifying DNNs to level D [18], [20] or C [17], [21], but to date none have reached Level A — the most severe level, required where failures may lead to fatalities, which is the case in aerospace. We hope that our approach can be used to satisfy Level A requirements in the future.

DNN Robustness. An alternative approach to the one proposed here, which has received significant attention, is the formal certification of DNN robustness. The idea is to conclude, a-priori, that a DNN is safe to use, at least for certain regions of its input space.

One approach for drawing such conclusions is to *formally verify* DNNs [5], [22], i.e., mathematically prove that the DNN behavior adheres to a given specifications. DNN verification methods are highly accurate sound and often complete [23], but suffer from limited scalability, require white-box access to the DNN in question, and pose certain limitations on the DNN's topology and activation functions.

Another approach attempts to circumvent these limitations by certifying a DNN's robustness with significant margins of error [19], [24]–[26]. This kind of attempt may be inadequate for aerospace certification, where large error margins are usually not acceptable.

Finally, there exist statistical approaches for evaluating the probability that a specific input is being misclassified by the network (i.e., is an adversarial input) [27]–[31]. *DEM* is statistical at its core as well. It can be configured to be more conservative, i.e., to prefer type II error over type I error, or less, depending on the use case. A detailed comparison between *DEM* and a state-of-the-art technique [31] is given in section V.

III. Background

Deep Neural Networks (DNNs). A deep neural network (DNN) N is a function $N : \mathbb{R}^n \rightarrow \mathbb{R}^m$, which maps an input vector $\vec{x} \in \mathbb{R}^n$ to an output vector $\vec{y} \in \mathbb{R}^m$. In this paper we focus on classification DNNs, where $\vec{x}$ is classified as class c if the c'th entry of $N(\vec{x})$ has the highest score: $\arg\max(N(\vec{x})) = c$.

Local Adversarial Robustness. Local adversarial robustness is a measure of how resilient a DNN is to perturbations around specific input points [33]:

Definition III.1. A DNN N is ϵ-locally-robust at input point $\vec{x_0}$ iff

$$\forall \vec{x}. ||\vec{x} - \vec{x_0}||_\infty \leq \epsilon \Rightarrow \arg\max(N(\vec{x})) = \arg\max(N(\vec{x_0}))$$

Intuitively, Definition III.1 states that for input vector $\vec{x}$, the network assigns to $\vec{x}$ the same label that it assigns to $\vec{x_0}$, as long as the distance of $\vec{x_0}$ from $\vec{x}$ is at most ϵ (using the L_∞ norm).

Probabilistic Local Robustness. Definition III.1 is Boolean: given ϵ and $\vec{x_0}$, the DNN is either robust or not robust. However, in real-world settings, and specifically in aerospace applications, systems could still be determined to be sufficiently robust if the likelihood of encountering adversarial inputs is greater than zero, but is sufficiently low. Federal agencies, for example, provide guidance that a likelihood that does not exceed 10^{-9} (per operational hour under normal conditions) for an extremely improbable failure conditions event is acceptable [34]. Consequently, prior research suggested an adjustment to Definition III.1 that allows to reason about a DNN's robustness in terms of its probabilistic-local-robustness (plr) [35], which is a real value that indicates a DNN's resilience to adversarial perturbations imposed on specific inputs. More formally:

Definition III.2. The ϵ-probabilistic-local-robustness (PLR) score of a DNN N at input point $\vec{x_0}$, abbreviated $\text{plr}_\epsilon(N, \vec{x_0})$, is defined as:

$$\text{plr}_\epsilon(N, \vec{x_0}) \triangleq P_{\vec{x}:||\vec{x}-\vec{x_0}||_\infty \leq \epsilon} \\ [\arg\max(N(\vec{x})) = \arg\max(N(\vec{x_0}))]$$

Intuitively, the definition measures the probability that for a specific input $\vec{x_0}$, an input $\vec{x}$ drawn at random from the ϵ-region around $\vec{x_0}$ will have the same label as $\vec{x_0}$.

Probabilistic Global Caterorial Robustness (PGCR). Although Definition III.2 is more realistic than Definition III.1, it suffers from two drawbacks. The first: recent studies indicate a significant disparity in robustness among the output categories of a DNN [13], [35], which is extremely relevant to safety-critical applications. However, Definition III.2 does not distinguish between output classes. The second drawback is that Definition III.2 considers only local robustness, i.e., robustness around a single input point, in a potentially vast input space; whereas it may be more realistic to evaluate robustness on large, continuous chunks of the input space. To address these drawbacks, we use the following definition [13]:

Definition III.3. Let N be a DNN, let $l \in L$ be an output label. The (ϵ, δ)-PGCR score for N with respect to l, denoted $pgcr_{\delta,\epsilon}(N, l)$, is defined as:

$$pgcr_{\delta,\epsilon}(N, l) \triangleq P_{\vec{x_1},\vec{x_2} \in \mathbb{R}^n, ||\vec{x_1}-\vec{x_2}||_\infty \leq \epsilon} \\ [|N(\vec{x_1})[l] - N(\vec{x_2})[l]| < \delta]$$

This definition captures the probability that for an input $\vec{x_1}$, and for an input $\vec{x_2}$ that is at most ϵ apart from $\vec{x_1}$, the confidence scores for inputs $\vec{x_1}$ and $\vec{x_2}$ will differ by at most δ for the label l.

Hypothesis Testing. Hypothesis testing is the task of accepting the *null hypothesis* H_0, or rejecting it in favor of an alternative hypothesis H_1. To achieve this, we employ a statistical test h, which is a function of the measurements. We further select a threshold τ, such that $h > \tau$ implies that H_0 is rejected in favor of H_1, whereas $h \leq \tau$ implies that H_0 is accepted.

It is common to consider the *significance* and *power* of a test in order to assess its usefulness:

- The significance α is given as $\alpha = P(h > \tau \mid H_0)$, i.e.., the probability for a false positive, leading to the rejection of H_0 although it is in fact true (type I error).
- The power is given by $1 - \beta$, where $\beta = P(h \leq \tau \mid H_1)$ is the probability for a false negative, i.e., of wrongfully rejecting H_1 (type II error). The power can also be formulated directly as $P(h > \tau \mid H_1)$, i.e., the probability for a true positive, or of correctly rejecting H_0, given that H_1 is true.

In general, we seek to reduce both α and β. Following the Neyman-Pearson lemma [36], one may maximize the power (minimize β) given α:

$$h = \frac{\mathcal{L}(H_1 \mid \text{measurements})}{\mathcal{L}(H_0 \mid \text{measurements})}$$

where $\mathcal{L}(H \mid \text{measurements})$ is the likelihood of H given the measurements, and is equal to the conditional probability $P(\text{measurements} \mid H)$. In this work, we employ such likelihood testing to decide between the hypotheses, to determine whether a given input is adversarial or not.

IV. The Proposed Method

A. The Inference Phase

Given an input point $\vec{x}_0$, our objective is to quantify the reliability of the DNN's particular prediction, $N(\vec{x}_0)$; i.e., to accept or reject the H_0 hypothesis that the input is adversarial. Our method for achieving this consists of the following steps:

- Execute the DNN on input $\vec{x}_0$, and obtain the prediction $N(\vec{x}_0)$.
- Generate k random perturbations around $\vec{x}_0$, denoted $\vec{x}'_1 \ldots \vec{x}'_k$, sampled uniformly at random from an ϵ-region around $\vec{x}_0$.
- For each sampled perturbation point $\vec{x}'_i$, compute the prediction $N(\vec{x}'_i)$ and check whether it coincides with $N(\vec{x}_0)$. Count the number of such matches ("number of hits"), denoted as h.
- If h is *above* a certain threshold $\mathcal{T}$, certify the prediction of $N(\vec{x}_0)$ as correct, i.e., reject the H_0 hypothesis. Otherwise, flag it as suspicious.

The motivation for this approach is based on III.3. Intuitively, for a genuine prediction over the input $\vec{x}_0$, the network would produce stable predictions in the ϵ-region around $\vec{x}_0$; in such a case, the number of hits will be high and will exceed the threshold $\mathcal{T}$. Conversely, if $\vec{x}_0$'s prediction is not genuine, i.e. $\vec{x}_0$ is an adversarial input for N, then the network would produce different predictions in the ϵ-region around $\vec{x}_0$, and the number of hits will be lower.

Naturally, in order for this approach to work, care must be taken in determining the two key parameters: ϵ, which determines the size of the region around $\vec{x}_0$ from which perturbations are sampled, and the threshold $\mathcal{T}$ for deciding whether a prediction is correct. Below we present a method for selecting these values.

B. Dataset Preparation and Calibration

Our approach requires an offline calibration phase, to adjust the method's parameters to the DNN at hand. Recall that, during inference, random samples are procured from an ϵ-region around the input point $\vec{x}_0$; and that our goal is to select ϵ in such a way that many of these samples will be classified the same way as $\vec{x}_0$ when it is genuine, but only a few will meet this criterion when $\vec{x}_0$ is adversarial. As it turns out, different ϵ values can cause dramatic differences in the number of hits; see Fig. 1 for an illustration. In order to select an appropriate value of ϵ, we use an empirical approach that leverages Levy et al.'s method [13] for computing PGCR values: we test multiple potential ϵ values, and select the one that produces the most accurate result.

Dataset Creation. First, we obtain a set of n correctly classified (genuine) inputs from the test data, $X_g = \{\vec{x}^g_1, \ldots, \vec{x}^g_n\}$. From this set, we construct a set $X_a = \{\vec{x}^a_1, \ldots, \vec{x}^a_{n_a}\}$ of adversarial inputs, obtained from the points in X_g by applying state-of-the-art attacks, such as PGD [37], [38]. We also consider a set of m potential ϵ values, $E = \{\epsilon_1, \ldots, \epsilon_m\}$, from among which we will select the ϵ value to be used during inference.

Next, for each $\epsilon \in E$ and for each input point $\vec{x} \in X = X_g \cup X_a$, we sample k perturbed inputs $\vec{x}'_1 \ldots \vec{x}'_k$ from the L_∞ ϵ-region around $\vec{x}$ [13], [35]. We then evaluate N on $\vec{x}$ and on its perturbations $\vec{x}'_1 \ldots \vec{x}'_k$, obtaining the outputs $N(\vec{x})$ for $\vec{x}$ and $N(\vec{x}'_1) \ldots N(\vec{x}'_k)$ for the perturbations. We count the number of *hits*, $h^{\vec{x}}_\epsilon = |\{i \mid \arg\max N(\vec{x}'_i) = \arg\max N(\vec{x})\}|$, and store it for later use.

We note that instead of using the measure of hits as a predictor for whether the input being examined is adversarial, one could try to discover the distribution of adversarial inputs, and then attempt to decide whether an input is likely to belong to that distribution. However, determining the distribution of adversarial inputs is difficult, and presently cannot be achieved using state-of-the-art statistical tools [39].

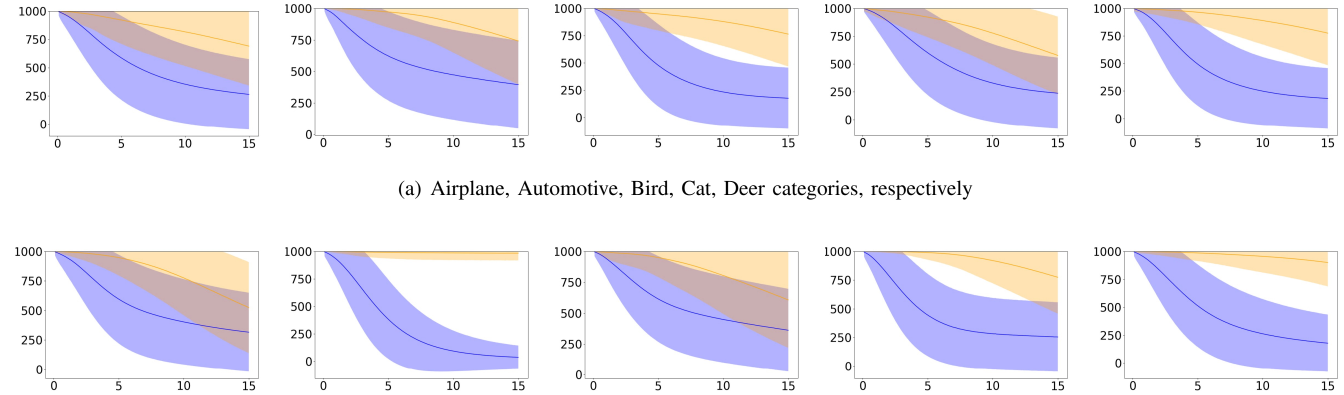
C. Maximal-Recall-Oriented Calibration

Using the collected data for various ϵ values, we present Alg. 1 for selecting an optimal ϵ value, and its corresponding threshold $\mathcal{T}$. The sought-after ϵ is optimal in the sense that, when used in the inference phase, it makes the fewest classification mistakes. The selection algorithm is brute-force: we consider each $\langle \epsilon, \mathcal{T} \rangle$ pair in turn, compute its success rate, and then pick the most successful pair.

The algorithm receives as input the set of potential ϵ values E; the sets of genuine and adversarial points, X_g and X_a, respectively; the measured number of hits, $h^{\vec{x}}_\epsilon$, for each input point $\vec{x}$ and each ϵ; and also the hyperparameter w_g, discussed later. The algorithm's outputs are the selected ϵ value, denoted ϵ^*, and its corresponding threshold of hits, $\mathcal{T}$, to be used during the inference phase.

Algorithm 1 Maximal-Recall-Oriented Calibration

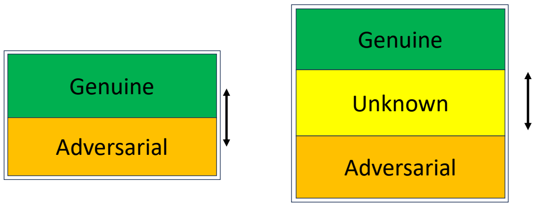
```
Input: $E$, $X_g$, $X_a$, $\{h^{\vec{x}}_\epsilon\}$, $w_g$
Output: $\epsilon^*, \mathcal{T}$
$\epsilon^*, bestScore, \mathcal{T} \leftarrow 0, 0, 0$
for all $\epsilon \in E$ do
    for $t := 0 \ldots k$ do
        $r_g = |\{\vec{x} \in X_g \mid h^{\vec{x}}_\epsilon > t\}| \;/\; |X_g|$
        $r_a = |\{\vec{x} \in X_a \mid h^{\vec{x}}_\epsilon < t\}| \;/\; |X_a|$
        $score = w_g \cdot r_g + (1 - w_g) \cdot r_a$
        if $score > bestScore$ then
            $\epsilon^* \leftarrow \epsilon, bestScore \leftarrow score, \mathcal{T} \leftarrow t$
        end if
    end for
end for
return $\epsilon^*, \mathcal{T}$
```



(a) Airplane, Automotive, Bird, Cat, Deer categories, respectively

(b) Dog, Frog, Horse, Ship, Truck categories, respectively

Fig. 1. An illustration of CIFAR-10 classifier performance. Each plot corresponds to a single output category. The Y-axes in the plots show the average number of hits (bold line) and standard deviation (faded area), for $k = 1000$ perturbations around genuine (orange) and adversarial (blue) inputs; whereas the X-axes represent different values of ϵ (in percents). The goal is to have as small overlap as possible between the two distributions, observing the significance of ϵ and the variance of the distributions between different categories.



(a) Recall Calibration Goals (b) Precision Calibration Goals

Fig. 2. Different optimization goals: the recall-oriented algorithm seeks the optimal threshold value that maximizes the separation between genuine and adversarial instances. The precision-oriented algorithm strives to minimize the "yellow" area, by determining the lowest threshold that maximizes adversarial input detection and the highest threshold that maximizes genuine input detection, e.g., minimizing the "unknown" cases for both thresholds.

Line 2 is the algorithm's main loop, iterating over all potential ϵ values in order to pick the optimal one. For each such ϵ, we iterate over all possible threshold values in Line 3; since each pair $\langle \epsilon, \vec{x} \rangle$ was tested over k perturbed inputs, the number of hits may vary between 0 and k, and we seek the best choice. Line 4 calculates the fraction of genuine inputs for which the threshold was reached, whereas Line 5 computes the fraction of adversarial inputs for which the threshold was not reached; these values are called the *recall rates*, and are denoted r_g and r_a, respectively. Then, Line 6 computes a real-valued score for this candidate ϵ. The score is a weighted average between r_g and r_a, where the hyperparameter w_g is used in prioritizing between a more conservative classification of inputs (w_g is lower, leading to fewer adversarial inputs mistakenly classified as genuine), or a less conservative one. The best score, and its corresponding ϵ and threshold, are stored in Line 8; and the selected ϵ and threshold are returned in Line 12.

Note. Alg. 1 computes a single ϵ value to be used during inference. However, in practice we used different ϵ value for each of the different output classes (see Fig. 1). E.g., during inference, if the input at hand is classified as "Dog", one ϵ value is used; whereas if it is classified as "Cat", another is selected. To make this adjustment, Alg. 1 needs to be run once for each output class, and the sets X_a and X_g need to be partitioned by output class, as well. For brevity, we omit these adjustments here, as well as in subsequent algorithms.

D. Maximal-Precision-Oriented Calibration

Alg. 1 is geared for recall; i.e., during inference, it classifies every input as either genuine (if the number of hits exceeds the threshold), or as adversarial. However, in our studies we observed that often, a division into *three* classes was more appropriate: in many cases, inputs encountered during inference would either demonstrate a very high number of hits, in which case they are likely genuine; a very low number of hits, in which case they are likely adversarial; or a medium number of hits, in which case it is unclear whether they were adversarial or not. While in Alg. 1 a human would have to inspect all inputs classified as adversarial, a more subtle division into three classes could allow the human to only inspect those inputs in the third, "unknown" category. See Fig. 2 for an illustration.

To support this variant, we present Alg. 2, which is geared towards increased *precision*. It is similar to Alg. 1, but instead of selecting an ϵ and returning a single threshold $\mathcal{T}$, it now returns a couple of thresholds — $\mathcal{T}_g$, which indicates a lower bar of hits for genuine inputs, and $\mathcal{T}_a$, which indicates an upper bar of hits for adversarial inputs. When this algorithm is used, the final inference step is changed to:

- If h is *above* a certain threshold $\mathcal{T}_g$, certify the prediction of $N(\vec{x}_0)$ as correct. If h is *below* a certain threshold $\mathcal{T}_a$, mark it as adversarial. Otherwise, flag it as "unknown".

We observe that this three-output scheme has an inherent tension between precision and recall. For example, remembering that genuine inputs tend to have a high number of hits and adversarial inputs a lower number, one could set $\mathcal{T}_g$ to be very high and $\mathcal{T}_a$ to be very low. This would result in excellent precision (i.e., the algorithm would make only a few mistakes), but many inputs would fall into the "unknown" category, causing poor recall. In terms of Fig. 2, this would result in a very wide yellow strip. Alternatively, one could set these two thresholds to be near-identical (resulting in a narrow yellow strip), which would simply mimic Alg. 1, with its excellent recall but sub-optimal precision. To resolve this, we allow the user to specify a-priori the desired precision, by setting minimal values $p_g^{\min}$ and $p_a^{\min}$ for the precision over genuine and adversarial inputs, respectively. Our algorithm then automatically selects the ϵ and threshold values that meet these requirements, and which achieve the maximum recall among all candidates.

More concretely, the inputs to Alg. 2 are the same as those to Alg. 1, plus the two threshold parameters, $p_g^{\min}$ and $p_a^{\min}$. For each candidate ϵ, the algorithm iterates over all possible thresholds (Line 4), from high to low, each time computing the recall rates (Lines 5–6) and the precision rate over genuine inputs (Line 7). If the precision rate meets the specified threshold, the threshold is stored as t_1 in Line 9. At the end of this loop, the algorithm has identified the lowest threshold for genuine inputs that meet the requirements. Next, the algorithm begins a symmetrical process for the threshold for adversarial inputs, this time iterating from low to high (Line 14), until it identifies the greatest threshold that satisfies the requirements (t_2, Line 19). Once both thresholds are discovered, the algorithm uses the recall ratios to compute a score (Line 24) — in order to compare the various ϵ values. The most optimal ϵ and its thresholds are stored in Line 25, and are finally returned in Line 29.

Algorithm 2 Maximal-Precision-Oriented Calibration

```
    Input: $E$, $X_g$, $X_a$, $\{h_\epsilon^{\vec{x}}\}$, $w_g$, $p_g^{\min}$, $p_a^{\min}$
    Output: $\epsilon^*$, $\mathcal{T}_g$, $\mathcal{T}_a$
 1: $\epsilon^*, bestScore, \mathcal{T}_g, \mathcal{T}_a, r_{\mathcal{T}_g}, r_{\mathcal{T}_a} \leftarrow 0,0,0,0,0,0$
 2: for all $\epsilon \in E$ do
 3:     $r_1, r_2 \leftarrow 0, 0$
 4:     for $t := k \dots 0$ do
 5:         $r_g = |\{\vec{x} \in X_g \mid h_\epsilon^{\vec{x}} > t\}| \,/\, |X_g|$
 6:         $r_a = |\{\vec{x} \in X_a \mid h_\epsilon^{\vec{x}} < t\}| \,/\, |X_a|$
 7:         $p_g = r_g / (r_g + (1 - r_a))$
 8:         if $p_g > p_g^{\min}$ then
 9:             $t_1 \leftarrow t$, $r_1 \leftarrow r_g$
10:         else
11:             break
12:         end if
13:     end for
14:     for $t := 1 \dots t_1$ do
15:         $r_g = |\{\vec{x} \in X_g \mid h_\epsilon^{\vec{x}} > t\}| \,/\, |X_g|$
16:         $r_a = |\{\vec{x} \in X_a \mid h_\epsilon^{\vec{x}} < t\}| \,/\, |X_a|$
17:         $p_a = r_a / (r_a + (1 - r_g))$
18:         if $p_a > p_a^{\min}$ then
19:             $t_2 \leftarrow t$, $r_2 \leftarrow r_a$
20:         else
21:             break
22:         end if
23:     end for
24:     $score = w_g \cdot r_1 + (1 - w_g) \cdot r_2$
25:     if $(score > bestScore)$ then
26:         $\epsilon^* \leftarrow \epsilon, bestScore \leftarrow score, \mathcal{T}_g \leftarrow t_1, \mathcal{T}_a \leftarrow t_2$
27:     end if
28: end for
29: return $\epsilon^*, \mathcal{T}_g, \mathcal{T}_a$
```

V. Evaluation

Implementation. In order to evaluate our method, we created a proof-of-concept implementation of Algs. 1 and 2 in a tool called DEM, which is available online [40]. Our tool is written in Python, and supports DNNs in the common PyTorch format.

Baseline. In our experiments we compared the two algorithms within DEM (recall-oriented and precision-oriented), and also compared DEM's recall-oriented algorithm to the Local Intrinsic Dimension (LID) method [31], which is the state of the art in detecting adversarial examples. Like *DEM*, LID has a calibration phase, in which the network is evaluated on multiple genuine and adversarial inputs. LID uses these evaluations to examine the assignments to the various neurons of the DNN, and then trains a regression model to predict, during inference and based on these values, whether an input is adversarial.

Since LID was not originally designed for categorical data, we trained it using two separate methods: (i) using examples from all classes, as originally suggested [31]; and (ii) training ten LID detectors, one for each output class. The second approach improved LID's performance slightly, and we used it as benchmark in the experiments described next.

Benchmarks. We used VGG16 [14] and Resnet [15] DNN models, trained on the CIFAR10 dataset [41]. The Resnet and VGG models achieved 88.35% and 83.75% accuracy, respectively. We then took the CIFAR10 test set, and removed from it any inputs that were misclassified by the trained models. We split the remaining inputs into two sets: a calibration set with 80% of the inputs, and an evaluation set with the remaining 20%. The adversarial inputs for the calibration and evaluation phases were obtained using PGD [37], with a maximal modification distance of 0.005 (i.e., 0.5%).

In the preliminary calibration phase, we created 1000 perturbed inputs around each genuine and adversarial input. We empirically observed that 1000 samples was a suitable choice — adding additional samples did not change our algorithm's performance significantly, whereas using significantly fewer samples led to degraded performance. For both calibration methods, we set the genuine recall hyperparameter to $w_g = 0.3$. This conservative choice prioritizes avoiding adversarial inputs wrongly misclassified as genuine. For the set E of candidate ϵ values, we arbitrarily chose distance values ranging from 0.001 to 0.15, with 0.001 resolution. While RoMA and gRoMA used an ϵ value of 0.04 [13], [35], we decided to use a higher value to ensure we do not miss a better calibration value.

We note that large values of ϵ reduce the number of hits for genuine inputs, because the region includes legitimate inputs with true different classifications; we empirically found that a distance greater than 0.15 typically causes this number to drop steeply.

We conducted our evaluation on a standard laptop, equipped with an AMD Ryzen 9 6900HX CPU, an NVIDIA GeForce RTX 3070 GPU, and 16GB of RAM. In order to create a virtual running environment, we used Python 3.11 and PyTorch 2.11. The calibration and data preparation for each algorithm took less than four hours per class. Each sample was analyzed in less than 0.003 seconds. The code for the creation of adversarial examples, calibration, and evaluation is available online [40].

A. Evaluating the Recall-Oriented Calibration Algorithm

Table I provides a comparison of our recall-oriented calibration algorithm with LID [31]. It demonstrates that the proposed method is highly effective at distinguishing between genuine and adversarial inputs. Compared to LID, we observe that LID identifies genuine examples with a slightly better recall; however, *DEM* is significantly better at identifying adversarial inputs for all classes. This is a favorable result, as the mistake of classifying an adversarial input as a genuine one is considered to be much more serious, in the use-cases we consider, than the second kind of mistake — classifying a genuine input as an adversarial.

TABLE I
EVALUATING THE RECALL-ORIENTED CALIBRATION ALGORITHM, IN COMPARISON TO LID. THE EVALUATION INCLUDES THE RESNET (MARKED AS RES) AND VGG (MARKED AS VGG) MODELS. THE TWO PARTS OF THE TABLE PRESENT GENUINE AND ADVERSARIAL RECALL: THE NUMBER OF SAMPLES, THE RECALL ACHIEVED BY *DEM*, AND THE RECALL ACHIEVED BY LID.

Class	Genuine Recall			Adversarial Recall		
	#Samples RES, VGG	*DEM*	LID	#Samples RES, VGG	*DEM*	LID
Airplane	180,177	63%,76%	81%,88%	393,125	92%,86%	54%,30%
Automotive	190,185	65%,65%	80%,91%	290,80	94%,87%	43%,52%
Bird	169,160	79%,67%	80%,84%	441,159	91%,91%	44%,35%
Cat	149,146	51%,50%	74%,85%	547,246	91%,89%	52%,31%
Deer	179,176	80%,79%	76%,89%	526,186	93%,97%	47%,37%
Dog	164,135	56%,60%	74%,87%	465,183	91%,92%	44%,30%
Frog	183,174	99%,91%	76%,84%	452,138	99%,97%	54%,34%
Horse	183,174	65%,74%	84%,88%	404,125	88%,87%	44%,38%
Ship	188,185	88%,91%	79%,92%	260,64	96%,92%	57%,56%
Truck	185,184	86%,80%	83%,87%	331,100	95%,95%	45%,44%

Examining the results, we observe the notable effectiveness of *DEM* when applied to the specific classes of the Resnet model. For instance, note the Frog class, where detection rates consistently surpass 90% across numerous samples. Nevertheless, some of the other classes demonstrate lesser effectiveness, and consequently these results, as a whole, do not meet the aerospace certification standards.

B. Evaluating the Precision-Oriented Calibration Algorithm

We conducted another similar experiment, this time using precision-oriented calibration, with precision thresholds set to 85% for genuine examples on Resnet and VGG, and 80% for adversarial inputs with both models. The w_g hyperparameter remained the same as in the recall evaluation. Table II depicts the results. The recall results reported therein appear alongside their corresponding precision results, for the same ϵ values. Except for one case, all recall values were above 50%, and most of them were over 70%. That second algorithm produces a detector with the required precision value in most cases.

TABLE II
EVALUATION OF THE PRECISION-ORIENTED CALIBRATION ALGORITHM. RESNET RESULTS ARE MARKED AS RES, AND VGG RESULTS AS VGG.

Class	Genuine Examples			Adversarial Inputs		
	#Samples RES, VGG	Precision	Recall	#Samples RES, VGG	Precision	Recall
Airplane	180,177	88%,85%	61%,80%	393,125	80%,81%	68%,86%
Automotive	190,185	91%,82%	68%,73%	290,80	77%,76%	91%,84%
Bird	169,160	89%,87%	79%,78%	441,159	81%,81%	91%,79%
Cat	149,146	91%,83%	41%,53%	547,246	77%,73%	69%,55%
Deer	179,176	90%,85%	82%,91%	526,186	84%,90%	91%,84%
Dog	164,135	88%,80%	50%,68%	465,183	74%,75%	74%,53%
Frog	183,174	88%,83%	100%,99%	452,138	100%,99%	86%,80%
Horse	183,174	85%,84%	64%,79%	404,125	75%,80%	80%,85%
Ship	188,185	90%,84%	95%,94%	260,64	94%,93%	90%,83%
Truck	185,184	89%,87%	91%,91%	331,100	91%,91%	88%,86%

Table II highlights the fact that taking into account both Precision and Recall affords a more favorable analysis of sample robustness.

VI. DISCUSSION AND FUTURE WORK

We presented here the *DEM* tool, which uses a novel, efficient probabilistic certification approach to determine the trustworthiness of individual DNN predictions. The proposed technique has several advantages: (i) it is applicable to black-box DNNs; (ii) it is computationally efficient during inference, using pre-calibrated parameters; and (iii) our evaluation results highlight *DEM*'s success in identifying adversarially distorted inputs. *DEM* could enable the selective use of DNN outputs when they are safe, while requesting human oversight when needed.

Moving forward, we plan to extend the study to cover more specific aviation use cases and related DNNs. In the long run, we hope that *DEM* will aid in creating certified DNN co-pilots. We further hope that including the *DEM* variance profile within functional hazard analyses may facilitate regulatory acceptance of classifier DNNs, allowing us to realize more of the AI potential in the aerospace domain.

References

[1] European Union Aviation Safety Agency, "EASA Artificial Intelligence Roadmap 2.0," 2023.

[2] P. Simard, D. Steinkraus, and J. Platt, "Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis," in *Proc. 7th Int. Conf. on Document Analysis and Recognition (ICDAR)*, 2003.

[3] C. Xu, D. Chai, J. He, X. Zhang, and S. Duan, "InnoHAR: A Deep Neural Network for Complex Human Activity Recognition," *IEEE Access*, vol. 7, pp. 9893–9902, 2019.

[4] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and Harnessing Adversarial Examples," 2014, technical Report. http://arxiv.org/abs/1412.6572.

[5] G. Katz, C. Barrett, D. Dill, K. Julian, and M. Kochenderfer, "Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks," in *Proc. 29th Int. Conf. on Computer Aided Verification (CAV)*, 2017, pp. 97–117.

[6] J. Knight, "Safety Critical Systems: Challenges and Directions," in *Proc. 24th Int. Conf. on Software Engineering (ICSE)*, 2002, pp. 547–550.

[7] Federal Aviation Administration, "RTCA DO-178C Software Considerations in Airborne Systems and Equipment Certification," 2013, https://nla.gov.au/nla.cat-vn4510326.

[8] SAE International, "ARP4754A Guidelines for Development of Civil Aircraft and Systems," 2010, https://www.sae.org/standards/content/arp4754a.

[9] P.-W. Tsai, C.-W. Tsai, C.-W. Hsu, and C.-S. Yang, "Network Monitoring in Software-Defined Networking: A Review," *IEEE Systems Journal*, vol. 12, no. 4, pp. 3958–3969, 2018.

[10] E. Levy, N. Maman, A. Shabtai, and Y. Elovici, "AnoMili: Spoofing prevention and explainable anomaly detection for the 1553 military avionic bus," *arXiv preprint arXiv:2202.06870*, 2022.

[11] A. Ashrov and G. Katz, "Enhancing Deep Learning with Scenario-Based Override Rules: a Case Study," in *Proc. 11th Int. Conf. on Model-Driven Engineering and Software Development (MODELSWARD)*, 2023, pp. 253–268.

[12] P. Munk, A. Abele, E. Thaden, A. Nordmann, R. Amarnath, M. Schweizer, and S. Burton, "Semi-Automatic Safety Analysis and Optimization," in *Proc. 55th Annual Design Automation Conf. (DAC)*, 2018, pp. 1–6.

[13] N. Levy, R. Yerushalmi, and G. Katz, "gRoMA: a Tool for Measuring Deep Neural Networks Global Robustness," *Proc. 12th Int. Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA)*, 2023.

[14] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *Proc. 3rd Int. Conf. on Learning Representations (ICLR)*, 2015, pp. 1–14.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Iimage Recognition," in *Proc. 29th IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[16] M. Gariel, B. Shimanuki, R. Timpe, and E. Wilson, "Framework for Certification of AI-Based Systems," *arXiv preprint arXiv:2302.11049*, 2023.

[17] K. Dmitriev, J. Schumann, and F. Holzapfel, "Towards Design Assurance Level C for Machine-Learning Airborne Applications," in *2022 IEEE/AIAA 41st Digital Avionics Systems Conference (DASC)*. IEEE, 2022, pp. 1–6.

[18] K. Dmitriev and J. Schumann, "Toward certification of machine-learning systems for low criticality airborne applications," in *2021 IEEE/AIAA 40th Digital Avionics Systems Conference (DASC)*. IEEE, 2021, pp. 1–7.

[19] C. Huang, Z. Hu, X. Huang, and K. Pei, "Statistical Certification of Acceptable Robustness for Neural Networks," in *Internet Corporation for Assigned Names and Numbers (ICANN)*, 2021, pp. 79–90.

[20] K. Dmitriev, J. Schumann, and F. Holzapfel, "Toward Design Assurance of Machine-Learning Airborne Systems," in *AIAA SCITECH 2022 Forum*, 2022, p. 1134.

[21] K. Dmitriev, J. Schumann, I. Bostanov, M. Abdelhamid, and F. Holzapfel, "Runway Sign Classifier: A DAL C Certifiable Machine Learning System," *arXiv preprint arXiv:2310.06506*, 2023.

[22] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Formal Security Analysis of Neural Networks Using Symbolic Intervals," in *27th USENIX Security Symposium (USENIX)*, 2018, pp. 1599–1614.

[23] G. Amir, H. Wu, C. Barrett, and G. Katz, "An SMT-Based Approach for Verifying Binarized Neural Networks," in *Proc. 27th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2021, pp. 203–222.

[24] T. Zhang, W. Ruan, and J. Fieldsend, "PRoA: A Probabilistic Robustness Assessment Against Functional Perturbations," 2022, technical Report. http://arxiv.org/abs/2207.02036.

[25] L. Weng, P.-Y. Chen, L. Nguyen, M. Squillante, A. Boopathy, I. Oseledets, and L. Daniel, "PROVEN: Verifying Robustness of Neural Networks with a Probabilistic Approach," in *Proc. Int. Conf. on Machine Learning (ICML)*, 2019.

[26] B. Anderson and S. Sojoudi, "Data-Driven Assessment of Deep Neural Networks with Random Input Uncertainty," 2020, technical Report. http://arxiv.org/abs/2010.01171.

[27] S. Webb, T. Rainforth, Y. Teh, and M. Pawan Kumar, "A Statistical Approach to Assessing Neural Network Robustness," 2018, http://arxiv.org/abs/1811.07209.

[28] K. Tit, T. Furon, and M. Rousset, "Efficient Statistical Assessment of Neural Network Corruption Robustness," in *Proc. 35th Conf. on Neural Information Processing Systems (NeurIPS)*, 2021.

[29] R. Mangal, A. Nori, and A. Orso, "Robustness of Neural Networks: A Probabilistic and Practical Approach," in *Proc. 41st Int. Conf. on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, 2019, pp. 93–96.

[30] J. Cohen, E. Rosenfeld, and Z. Kolter, "Certified Adversarial Robustness via Randomized Smoothing," in *Proc. Int. Conf. on Machine Learning (ICML)*, 2019.

[31] X. Ma, B. Li, Y. Wang, S. Erfani, S. Wijewickrema, G. Schoenebeck, D. Song, M. Houle, and J. Bailey, "Characterizing Adversarial Subspaces Using Local Intrinsic Dimensionality," 2018, technical Report. http://arxiv.org/abs/1801.02613.

[32] A. Aldahdooh, W. Hamidouche, S. A. Fezza, and O. Déforges, "Adversarial Example Detection for Dnn Models: A Review and Experimental Comparison," *Artificial Intelligence Review*, vol. 55, no. 6, pp. 4403–4462, 2022.

[33] O. Bastani, Y. Ioannou, L. Lampropoulos, D. Vytiniotis, A. Nori, and A. Criminisi, "Measuring Neural Net Robustness with Constraints," in *Proc. 30th Conf. on Neural Information Processing Systems (NIPS)*, 2016.

[34] A. Landi and M. Nicholson, "ARP4754A/ED-79A-Guidelines for Development of Civil Aircraft and Systems-Enhancements, Novelties and Key Topics," *SAE International Journal of Aerospace*, vol. 4, no. 2011-01-2564, pp. 871–879, 2011.

[35] N. Levy and G. Katz, "RoMA: a Method for Neural Network Robustness Measurement and Assessment," *Proc. 29th Int. Conf. on Neural Information Processing (ICONIP), pp. 92-105*, 2021.

[36] J. Neyman and E. Pearson, "On the Use and Interpretation of Certain Test Criteria for Purposes of Statistical Inference: Part I," *Biometrika*, pp. 175–240, 1928.

[37] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards Deep Learning Models Resistant to Adversarial Attacks," 2017, technical Report. http://arxiv.org/abs/1706.06083.

[38] M. Müller, C. Brix, S. Bak, C. Liu, and T. Johnson, "The Third International Verification Of Neural Networks Competition (VNN-COMP 2022): Summary And Results," 2022, technical Report. http://arxiv.org/abs/2212.10376.

[39] SAS, "SAS JMP Website," 2001, site. https://www.jmp.com/en_us/home.html.

[40] G. Katz, N. Levy, I. Refaeli, and R. Yerushalmi, "DEM: Code and Experiments," 2023, https://drive.google.com/drive/folders/1UxCOCLenJspDLRpXR6LY7Zn3_OBgZ76E?usp=sharing.

[41] A. Krizhevsky and G. Hinton, "Learning Multiple Layers of Features from Tiny Images," 2009.