Robustness Distributions in Neural Network Verification

ANNELOT W. BOSMAN*, Leiden University, The Netherlands
AARON BERGER, RWTH Aachen University, Germany
HOLGER H. HOOS, RWTH Aachen University, Germany and Leiden University, The Netherlands
JAN N. VAN RIJN, Leiden University, The Netherlands

Neural networks are vulnerable to slight alterations to otherwise correctly classified inputs, leading to incorrect predictions. To rigorously assess the robustness of neural networks against such perturbations, verification techniques can be employed. Robustness is generally measured in terms of adversarial accuracy, based on an upper bound on the magnitude of perturbations commonly denoted by ε. For each input in a given set, a verifier determines whether a perturbation up to magnitude ε can deceive the network. In this work, we contribute novel analysis techniques for the verified robustness of neural networks for supervised classification problems and report on interesting findings we obtained using these techniques.

We utilise the notion of robustness distributions, specifically those built using the concept of critical ε values. Critical ε values are defined as the maximum amount of perturbation for which a given input is provably correctly classified, such that any larger perturbations can cause misclassification. To effectively estimate the critical ε values for each input in a given set, we utilise a variant of the binary search algorithm. We then analyse the distributions of these critical ε values over a given set of inputs for 12 MNIST classifiers widely used in the literature on neural network verification. Using a Kolmogorov-Smirnov test, we obtain support for the hypothesis that the critical ε values of 11 of these networks follow a log-normal distribution. Furthermore, we found no statistically significant differences between the critical ε distributions for training and testing data for 12 feed-forward neural networks on the MNIST dataset.

Generally, we find a strong positive correlation between the critical ε of an input image across various networks. However, in some cases, an input that is easily perturbed to deceive one network may require a considerably larger perturbation to deceive another. Furthermore, for a given input, the adversarial examples that we find differ across networks, with different predicted classes associated with them.

We investigate the effect adversarial training can have on the critical ε distribution of various neural networks for MNIST, CIFAR and GTSRB datasets. We also find that complete verification is expensive for some of the CIFAR and GTSRB networks, which limits the precision of the robustness distributions we were able to obtain. Nonetheless, we observe that most of the critical ε distributions of the networks obtained through adversarial training do not follow a log-normal distribution. Furthermore, adversarial training significantly improves the critical ε distributions for testing as well as training data in most cases.

Lastly, we provide a ready-to-use Python package available on GitHub that can be used for creating robustness distributions and enables others to build upon our work.

JAIR Associate Editor: Scott Sanner

*Corresponding Author.

Authors' Contact Information: Annelot W. Bosman, ORCID: 0009-0004-1050-5165, a.w.bosman@liacs.leidenuniv.nl, Leiden University, Leiden, The Netherlands; Aaron Berger, ORCID: 0009-0006-3322-5718, aaron.berger@rwth-aachen.de, RWTH Aachen University, Aachen, Germany; Holger H. Hoos, ORCID: 0000-0003-0629-0099, hh@cs.rwth-aachen.de, RWTH Aachen University, Aachen, Germany and Leiden University, Leiden, The Netherlands; Jan N. van Rijn, ORCID: 0000-0003-2898-2168, j.n.van.rijn@liacs.leidenuniv.nl, Leiden University, Leiden, The Netherlands.

DOI: 10.1613/jair.1.18403

JAIR Reference Format:
Annelot W. Bosman, Aaron Berger, Holger H. Hoos, and Jan N. van Rijn. 2025. Robustness Distributions in Neural Network Verification. *Journal of Artificial Intelligence Research* 83, Article 20 (July 2025), 41 pages. DOI: 10.1613/jair.1.18403

1 Introduction

It is well known that neural networks are vulnerable to producing incorrect outputs resulting from changes in inputs. One commonly researched type of vulnerability arises from adversarial attacks, based on small input perturbations. Various studies [19, 45] have shown how small, sometimes imperceptible changes to a given input can mislead state-of-the-art neural networks. Therefore, robustness against small input perturbations, also known as local robustness, is an important topic of investigation.

Various methods for assuring complete robustness against perturbations within a certain radius ε have since been developed [3, 5, 8, 15, 16, 17, 18, 20, 22, 46, 47, 48, 52]. Yearly neural network verification competitions have been held since 2020, aiming to assess the performance of different tools available on a selected set of networks and to incentivise their broad use using standardised formats [7, 36]. Verification approaches have improved in speed as well as scalability, in part by using algorithms from the field of optimisation and GPU acceleration. Neural network verification algorithms can be categorised as complete or incomplete. In contrast to incomplete algorithms, which do not guarantee to report a solution, complete verification methods, when run to completion, always provide formal proof for the robustness of a given neural network for a specific input [28]. This typically comes at a considerable computational cost.

In many studies, the maximum magnitude of allowable input perturbations, ε, is treated as an experimental setting determined in advance [28]. Following this notion, *robust accuracy* is defined as the percentage of inputs that will be classified correctly, regardless of what perturbation within the predefined bound ε is applied to any given input [13, 39, 55]. In some cases, this definition is adapted to ignore originally misclassified inputs [54].

However, we argue that robust accuracy is not able to fully capture important aspects of the robustness of a neural network, for the following reasons: (i) it requires the domain expert to pre-specify an acceptable perturbation level, which is knowledge that might not be available at that point; (ii) it does not indicate what level of perturbation is tolerated for an individual input before it will be misclassified; and (iii) when comparing two networks with similar robust accuracies, important nuances might not be revealed concerning the robustness of individual inputs against perturbations. For example, given two neural networks trained to solve the same classification problem, for one network, many of the inputs may tolerate only a perturbation slightly larger than the pre-defined ε value, while for the other, this margin might be much larger; information of this nature should be useful to anyone interested in solving the given classification problem robustly in practice.

To overcome these limitations, in this work, which is an extended version of a workshop contribution [4], we are interested in the distribution of robustness of a given neural network against adversarial perturbations. We focus specifically on neural networks that are employed for classification tasks. To study this, we utilise the concept of *critical ε value*, which is defined as the maximum amount of perturbation to a given input for which the predicted class will provably remain correct, in the context of local robustness verification. The critical ε value is sometimes also referred to as the *adversarial result* [29] and the *maximum perturbation bound* [9]. We use a complete verifier to provide the robustness guarantee. Furthermore, we utilise a method from the literature, k-binary search [11], to empirically estimate a lower bound on the critical ε value. By running multiple verification queries in parallel, we can make use of the information obtained from the query that finishes first. In particular, this helps us to determine whether we need to continue running the other verification queries, and/or produce a number of new queries.

This leads to the *critical ε distribution*, an empirical distribution over the critical ε values for a wide range of inputs to a given neural network. These critical ε distributions allow us to compare and analyse the robustness of neural networks systematically and in detail. Note that we refer to critical ε distributions as the empirical

distribution over instances measured specifically in terms of the critical ε, whereas we refer to robustness distributions as the more general concept, i.e., the empirical distribution over inputs measured using any robustness notion. In this article, we focus specifically on critical ε distributions. An advantage of using distributions to define the robustness of a neural network is that they enable assumptions about inputs for which we have not found the critical ε values. In parallel to our earlier work [4], J. Liu *et al.* [30] also utilise critical ϵ distributions, but rather than analysing them directly, they utilise them to develop a novel methodological contribution for verifying neural networks.

The main contributions of our work presented in this article are as follows:

(1) We elaborate on the concept of robustness distributions, which are empirical distributions over a measure of the robustness of a given neural network against input perturbations and propose to use them for robustness evaluation between networks. We calculate these distributions specifically using the critical ε values, which provide a *provable lower bound* to the amount of adversarial perturbation that a network can withstand for a given input. This results in so-called critical ε distributions.
(2) We utilise a variant of the binary search algorithm, k-binary search, to obtain the critical ε distribution for a given neural network. When searching for the critical ε for a given network and input, uncertain delays and possibly missing values are encoutered (due to time-outs and out-of-memory errors). k-binary search handles this by evaluating k verification queries with different ε values in parallel, potentially terminating queries early when new information becomes available.
(3) We analyse the critical ε distributions for 12 widely studied fully-connected MNIST neural networks on both training data and testing data, for correctly classified instances. We specifically note the importance of using widely used network architectures that have also been used in other robustness studies, to minimise the bias of selecting an architecture and training hyperparameters.
(4) Surprisingly, we observe that 11 of the robustness distributions of these neural networks correspond to log-normal distributions, which contradicts the claims by J. Liu *et al.* [30]. Furthermore, the robustness distributions that we investigated do not significantly differ between training and testing data, suggesting that these distributions generalise to previously unseen inputs.
(5) We investigate the correlation between the critical ε for a certain input across multiple networks, with the goal of understanding whether the size of the critical ε for an instance is related to the input or specific to a given network. We find that generally there is a strong positive correlation of critical ε values between images for training data. However, for testing data, this correlation is less pronounced.
(6) We analyse the effect of adversarial training on the robustness distributions of different fully-connected neural networks for the MNIST, CIFAR-10 and GTSRB datasets. We confirm that for all three datasets and all networks we considered, adversarial training has a significant positive effect on the robustness distributions and generally increases the computational resources required for verification.
(7) We analyse the robustness distributions of multiple convolutional neural architectures for MNIST, CIFAR-10 and GTSRB, which are all trained in three distinct ways, i.e., using a conventional training method and two different adversarial training methods. We observe that adversarial training can result in networks with significantly improved robustness distributions. We also see that complete verification is too expensive to generate dependable robustness distributions for the CIFAR-10 and GTSRB datasets. The robustness distributions of the networks used in this part of our study do not seem to follow log-normal distributions generally, independent of the training method. Furthermore, similar to the aforementioned fully-connected neural networks, we find that robustness distributions do not significantly differ between training and testing data for the adversarially trained networks as well as the conventionally trained networks considered.

(8) Finally, we provide a ready-to-use Python package for measuring robustness distributions.[1] The package is modular, such that any part can be changed, including the instance set under consideration, the robustness property or the verifiers used. This makes our results fully reprodicible and will help others build on our work. Furthermore, all our networks and data are available on GitHub.[2]

The remainder of this article is structured as follows. We discuss the main concepts related to neural network verification in Section 2. In Section 3, we define the concept of robustness distributions and describe the k-binary search algorithm. Following this, Section 4 presents the aforementioned empirical results on the conventionally trained fully-connected neural networks. In Section 5 we discuss the effect of adversarial training on the robustness distributions of fully-connected neural networks for three different datasets and similarly in Section 6 we analyse the effect on convolutional neural networks. Finally, in Section 7, we summarise the main findings from this work and briefly discuss avenues for future work.

2 Background

In the following, we review key aspects of local robustness verification and the critical ε that form the basis for our work and briefly explain the verification method we are building on. We also discuss incomplete verification as well as training methods for improving robustness and closely related work.

2.1 Local Robustness Verification

Verification methods for neural networks aim to formally evaluate whether a given model satisfies certain input-output properties. One such property is local robustness, which refers to the ability of a neural network to maintain the correct prediction for a single input, even when that input is slightly perturbed, possibly with malicious intent. To assess local robustness, verification algorithms analyse a set of inputs and seek to determine whether there exist small perturbations that could cause the model to produce incorrect outputs.

When such perturbations are applied to a previously correctly processed input, so-called *adversarial examples* are obtained. The size of these perturbations is usually restricted by a predefined maximum radius, typically denoted by ε, which limits the extent of changes that can be made to the original input image or each input variable.

Similar to much of the literature on local robustness verification, our work focuses on neural networks for supervised classification. Formally, a neural network classifier can be described by a function f_θ in $\mathbb{R}^n \rightarrow \mathbb{R}^m$, where θ is the set of trained parameters for f_θ, n is the number of input variables, and m is the number of possible classes.

Considering an input x_0 with correct label $\lambda(x_0)$ and a region around x_0 defined by $G_{p,\varepsilon}(x_0) = \{x : ||x - x_0||_p \leq \varepsilon\}$, using some p norm, local robustness verification aims to formally determine whether there exists a perturbed input $x \in G_{p,\varepsilon}(x_0)$ such that the predicted label of x is no longer equal to the predicted label of x_0. In our work, the perturbation is measured using the l_∞ norm, consistent with much of the literature.

In many cases, local robustness verification is modelled using mixed-integer program (MIP) formulations. In general, MIP solving is known to be NP-hard, but in practice, many instances can be solved within a reasonable time using state-of-the-art commercial MIP solvers, such as Gurobi or CPLEX [46]. These solvers are complete and therefore guaranteed to find any existing adversarial examples when run to completion.

Robustness verification systems have improved in speed as well as scalability, in part by utilising algorithms from the field of optimisation [5, 20, 22] as well as GPU acceleration [15, 41, 48]. However, as neural networks can have billions of trainable parameters, current verifiers do not scale to state-of-the-art networks. In contrast to the previously mentioned complete verifiers, there also exist incomplete verification approaches; however,

[1] See: https://github.com/ADA-research/VERONA
[2] See: https://github.com/ADA-research/NNV_JAIR_robustness_distributions

whilst these can be used to find adversarial examples for a given input and provide lower bounds on the critical ε, they do not guarantee to provide a result for each property.

2.2 Critical ε and Minimum Adversarial Example

The existing literature on complete neural network verification predominantly focuses on the binary question of whether an adversarial example exists within a specific perturbation radius ε. To create robustness distributions, we utilise ε^*, the critical ε value for a trained neural network f_θ and input x_0 with correct class $\lambda(x_0)$, defined as the maximum perturbation size such that any perturbed input $x \in \{x : ||x - x_0||_\infty \leq \varepsilon^*\}$ cannot lead to misclassification and some perturbations larger than ε^* will provably lead to misclassification. Other terms for the concept of ε^* include the adversarial radius [30], the adversarial result [29] and the maximum perturbation bound [49].

Tjeng *et al.* [46] observed that ε^* can be determined by applying a binary search algorithm over a range of ε values using any verification method, a technique later adopted, e.g., in work by J. Liu *et al.* [30]. Other work has investigated exploiting the structure of the neural network verification problem to directly determine ε^* [9, 44, 46]. However, Tjeng *et al.* [46] found that the gap between the minimum adversarial distortion and the certified lower bound is significant for each case they investigated with their proposed verifier, *MIPVerify*. Strong *et al.* [44] adapted the *Marabou* verifier [22] for direct optimisation for finding the smallest adversarial distortion. This adaptation, called *MarabouOpt*, is complementary to MIPVerify and can solve more instances on some benchmarks. In our study reported here, we have used a k-binary search algorithm (See section 3.3); this choice allows flexibility in the framework, enabling the use of any state-of-the-art verifier that can address the specific verification problem at hand, such as different layers or different verification properties. By not relying on a single verifier, our framework remains adaptable and broadly applicable.

A concept strongly related to ε^* is the minimum adversarial distortion: the smallest amount of perturbation necessary to create an adversarial example that can mislead a neural network, such that any smaller perturbation could not lead to a misclassification for a given instance. In theory, the ε^* and minimum adversarial distortion only differ from each other with an indistinguishable small amount. However, in practice, we discretise the search space, which ensures that the gap between these two quantities is at least the bin-width, and additionally, in some cases, there will be time-out errors and out-of-memory errors, which increase the gap between ε^* and minimum adversarial distortion even further. Finding minimum adversarial distortions using different algorithms (that generally give no guarantee for minimality) has been extensively studied, for example by Moosavi *et al.* [34], who proposed *DeepFool*, a method based on orthogonal projections. Weng *et al.* [49] aims to find the minimum adversarial distortion and consider the certifiably safe radius for an instance to be the lower bound to that. To provide formal guarantees, they use extreme value theory to find the Lipschitz constant and combine this in their framework called *CLEVER*. Another line of work uses adversarial attack methods, such as PGD, to find the minimum adversarial distortion without guarantees.

2.3 Neural Network Verifiers

In the first part of our experiments, we utilise a recent version of the branch-and-bound-based neural network verification framework (BaB) [8, 15]. BaB tackles the verification problem by converting it into a MIP formulation and subsequently solving it using the branch-and-bound algorithm. The branch-and-bound algorithm partitions the feasible region of a given MIP instance into smaller regions, making it easier to iteratively solve the instance.

In the second part of our experiments, we utilise a recent version of the GPU-accelerated α, β-CROWN framework [57]. α, β-CROWN is based on a branch-and-bound approach and leverages bound-propagation methods to iteratively tighten the feasible region of the exact verification problem. This verifier was also the winner of the 2021, 2022, 2023 and 2024 Neural Network Verification Competitions [2, 6, 36].

2.4 Training to Enhance Robustness

Adversarial training methods are used to enhance the robustness of neural networks against small input perturbations. While Goodfellow *et al.* [19] suggested retraining neural networks including adversarial examples to increase robustness, Madry *et al.* [31] found eevidence that this tends to cause overfitting on the adversarial examples. Madry *et al.* [31] introduced project gradient descent (PGD) for improving the robustness of neural networks against adversarial attacks.

PGD uses optimisation techniques for modelling various forms of adversarial attacks. After specifying the attack type, PGD aims to satisfy a guarantee that a network is robust against the specified attack, making use of a min-max formulation that controls the loss function of the network during training, to minimise the maximum adversarial loss created by an attack. Another adversarial training method, called DiffAI, leverages differentiable abstract interpretation [32] – an approach in which an approximation of the feasible region of the attack is created and incorporated into the loss function while retraining a given neural network. Even though these methods seem to work well in increasing adversarial robustness, robust training also tends to result in a significant reduction in the accuracy of the adversarially trained networks [31, 53].

Another class of methods used to enhance robustness is certified training. Unlike adversarial examples, which can overapproximate the robustness of a given network, certification methods provide a guaranteed lower bound on robustness [40]. However, certified training is computationally expensive and often exhibits unstable behaviour [40], which typically results in a significant trade-off in accuracy [37]. Despite these challenges, certified training has the potential to facilitate verification by making it easier to verify neural networks for specific properties [56].

In this work, we focus exclusively on adversarial training, as it is a more computationally efficient and accessible approach for improving the robustness of neural networks.

2.5 Other Related Work

Other work that mentions a concept similar to robustness distributions includes a study by J. Liu *et al.* [30], utilised an incomplete and a complete verifier for determining the ε^* for a fully-connected MNIST network. While both the work of J. Liu *et al.* [30] and ours adopt the notion of robustness distributions, we note that the main contribution of our work is the analysis of these distributions, whereas the work of J. Liu *et al.* [30] which utilises the robustness distribution to make a methodological contribution towards verifying neural networks.

J. Liu *et al.* [30] furthermore investigated a set of networks trained on both CIFAR-10 and MNIST, using incomplete verification to find the approximate ε^*. They reported that for some of the networks, the distributions of the approximate ε^* seem to follow the normal distribution, and they leveraged this to create a methodology for input validation. In our work, reported in the following, we find that many robustness distributions are more accurately characterised using log-normal distributions.

3 Robustness Distributions

In this section, we describe practical considerations for determining the critical ε (see Section 2.2), as well as k-binary search [12], the algorithm we utilise to determine the critical ε value for a given input. These two concepts are necessary to create the robustness distributions we are investigating in this work. We note that a critical ε distribution is a specific case of robustness distribution, and in principle, robustness distributions can also be constructed using alternative robustness measures.

3.1 An Empirical Lower-Bound on the Critical ε Value

The ε^* distribution or robustness distribution is the distribution of ε^* values over a set of instances I. In the context of this work, an instance corresponds to a neural network and an input, e.g., an image. As we perform this

investigation on image classification tasks, we will use the terms input, image and input image interchangeably. The ε^* distribution only includes the ε^* values for images that were originally correctly classified by the network.

The cost of determining the ε^* values depend on the given dataset. Computing exact values of the critical ε values is not always practically possible, due to time-outs, memory limits and other practical issues. However, in most cases, good lower bounds can be established using state-of-the-art verification techniques.

In the following, we use $\tilde{\varepsilon}^*$ to denote an empirical lower bound for ε^*. Assume we investigate a discretised set of ε values, $E = (\varepsilon_0, \ldots, \varepsilon_h)$, for a given network f_θ and input x_0, where $\forall i = 0 \ldots h-1 : \varepsilon_i < \varepsilon_{i+1}$. If there exists an adversarial example for ε_i, where $0 < i \leq h$, and no adversarial example for ε_{i-1}, $\tilde{\varepsilon}^*$ is determined at ε_{i-1}, the ε^* (which represents the exact critical ε) will be in the interval $[\varepsilon_{i-1}, \varepsilon_i)$. In the case where a perturbation with maximum amount ε_i does not lead to misclassification and a perturbation with maximum amount ε_{i+j} can lead to a misclassification – this amount is referred to as the minimum adversarial distortion – where $0 \leq i < h$ and $0 < j \leq h - i$ and the verifiers are not able to resolve the verification queries for ε values in between these two in the set (e.g., due to time-outs or out-of-memory errors), $\tilde{\varepsilon}^*$ will be determined at ε_i as a conservative lower bound.

To find the empirical robustness distributions investigated later in this work, we compute the conservative lower bound, $\tilde{\varepsilon}^*$, for a set of relevant instances. For example, if the exact ε^* equals 0.0035, we may not find this exact value due to discretisation. We will carry out a verification query at $\varepsilon = 0.003$, which does not lead to misclassification, and a query at $\varepsilon = 0.005$, for which an adversarial example is found, we determine $\tilde{\varepsilon}^* = 0.003$ as the empirical lower bound for ε^*. When we do this over a set I of instances, the empirical robustness distribution in itself provides a lower bound on the theoretical ε^* distribution.

To utilise the advantages of (binary) search methods, we will discretise the problem and investigate a predefined set of ε ranges. Verifying every possible value of ε is infeasible because verification can take considerable time as the underlying local robustness verification problem is NP-complete for ReLU-based networks [22].

3.2 Empirical Upper-Bounds and Verification Gaps

If the verifiers we use work as expected, we would find an adversarial example at ε_{i+1} and the $\tilde{\varepsilon}^*$ at ε_i. In practice, it could be that the verifiers cannot resolve a verification query leading to *verification gaps* between the $\tilde{\varepsilon}^*$ and smallest ε for which we find an adversarial example, which we will refer to as the *approximate minimum adversarial distortion* denoted by $\tilde{p}$. The verification gap for a single instance is then defined as $\tilde{p} - \tilde{\varepsilon}^*$.

In the case the verification system can verify all relevant queries, the verification gap is as large as the difference between two adjacent ε values (due to the discretisation of the search process for ε^*). In practice, however, the verification gap may get quite large when time-out errors or out-of-memory errors occur frequently, due to the complexity of the verification tasks.

Critical terminations. One consequence of using verification queries with a time-out is that we might encounter $\tilde{\varepsilon}^*$ values affected by time-outs. This means that we obtained one or more time-outs for ε in between two ε for which the smallest does not lead to an adversarial example and the larger does; we refer to this as *critical termination* and this contributes to the verification gap. Examining our dataset for $k = 2$, which consists of a total of 2301 instances in total for 12 classifiers and the testing data and training data, we encountered a total of 61 critical terminations (due to time-outs, out-of-memory errors or other running time errors) over all the testing and training queries, which corresponds to 2.6% of the total number of queries. In all other cases, we were able to estimate a tight lower bound on the ε^*, within the limits of the discretised ε values described previously. When we derive the $\tilde{\varepsilon}^*$ distributions of complex networks (such as the ones considered in Sections 5 and 6), more premature terminations are encountered, resulting in looser bounds. In this work, we consider robustness distributions to be of high quality when their bounds are tight and the vast majority of instances can be successfully solved.

3.3 k-binary Search

One proposed method for finding the ε^* is by employing binary search, which can be used as a wrapper method in combination with any existing verification method. However, for each verification query, the verification of whether or not a specific of a ε value with a specific network and input leads to an adversarial example takes an unknown amount of time, and queries that exceed a given time-out are terminated to limit the potential waste of computational resources. This means that a binary search approach experiences delays and potentially contains missing results if the time-out is reached. If we use regular binary search, as done by Cicalese *et al.* [30], its benefits may be nullified by the delays and missing values.

Instead, we propose a novel search method based on k-binary search, a modified version of binary search, based on the work of Cicalese *et al.* [11]. The key idea is to verify k queries in parallel, rather than just one at a time. In their work, Cicalese *et al.* [11] analysed the optimal strategy for binary search when there are unknown delays and at most one missing value. In k-binary search, we apply this strategy to concurrently verify multiple queries of the same network and input with different ε values. Standard binary search typically selects the midpoint of a sorted array, checks the answer and iteratively continues until the correct element is found. When verifying k queries in parallel, we divide the search space into $k + 1$ parts. In this research, we choose to use equal-sized parts. Using information from the queries that run in parallel but find a result faster, we can therefore reduce the total time necessary to find the $\tilde{\varepsilon}^*$ value for a given input.

For example, suppose that we are simultaneously verifying queries A and B with ε values of a and b, respectively, with $a < b$, for a given input and network. If one query terminates before the other, in some cases, we can immediately use the result of the former to infer the result of the latter. Specifically, if for $\varepsilon = b$, no adversarial example is found, we know that there will be no adversarial example found for $\varepsilon = a$, and we can terminate that process early. This allows us to save computing resources by avoiding unnecessary verification runs. The opposite is also true, i.e., if for $\varepsilon = a$ an adversarial example is found, we can terminate the query for $\varepsilon = b$.

Using k-binary search on the discretised problem will not provide us with the exact ε^*; instead, the procedure produces a range within which the exact ε^* lies. For example, if we find an adversarial example for $\varepsilon = 0.005$, but there does not exist one for 0.003, we take 0.003 as our empirical lower bound $\tilde{\varepsilon}^*$, knowing that the actual value lies within the interval $(0.003, 0.005]$. In case we find, for example, for $\varepsilon = 0.003$ there exists no adversarial example, and then, for one or more consecutive ε, we encounter errors or time-outs, and for the next ε, we find an adversarial example, we again choose $\tilde{\varepsilon}^* = 0.003$ as the lower bound for ε^*. We have thus discretised the problem of finding an estimation of the ε^* for a given input while speeding up the process of finding it using k-binary search.

3.4 The VERONA Tool for Neural Network Verification Experiments

Designing and executing neural network verification experiments can be challenging, involving tasks such as installing the verifier and defining the properties. To simplify this process, we have developed an open-source software package[3] for robustness verification experiments. The package, called VERONA, is object-oriented, to facilitate extensions and adaptations for specific use cases.

On a high level, the goal of the software architecture we have developed was to separate the file and dependency handling from the actual algorithm and verification execution. Towards that end, we created an experiment repository class that takes care of all the IO processes necessary to organise the experiment data on the file system and to store the required files, such as properties, for the verification algorithm. For handling the results, we created separate classes for the verification results and the algorithms to estimate the ε values. With these classes, all the necessary resources to compute a given property can be handed to an estimator in the form of a

[3] https://github.com/ADA-research/VERONA

class that keeps track of these resources, which we call the *verification context*. Then, the estimator takes care of computing the property e.g., using binary search for the critical ε value.

To enable the use of different verifiers within our package, auto-verify [43] has been wrapped into a verification module. This allows the use of all verifiers supported by auto-verify but also ensures that our package can be extended to accommodate other verifiers, which can easily be included using new wrapper modules. This greatly facilitates the integration of other verifiers and critical ε estimation techniques into VERONA. For creating the verification properties, we use the VNN-LIB[4] format, as this has become the standard in the yearly neural network verification competitions [7], yet the property generator class could be extended to various other formats to enable a variety of verifiers.

In this work, we have used VERONA mainly for creating and analysing robustness distributions. In the package, we have included examples for reproducing our experiments.

4 Empirical Investigation on Conventionally Trained Neural Networks

In this section, we focus on fully-connected neural networks that have been conventionally trained on MNIST data [27]. These constitute the simplest class of problems in our range of experiments. First, we explored whether k-binary search can determine reasonable lower bounds on the ε^* values. Next, we investigated whether the robustness distributions follow the same parametric distribution class for all fully-connected neural networks for the MNIST dataset. Then, we assessed whether the robustness of the training observations and the testing observations for a given neural network comes from the same distribution. Furthermore, we investigated the correlation of the $\tilde{\varepsilon}^*$ between networks trained on the same data.

4.1 Setup of Experiments

Choice of networks. We investigate the robustness distributions of 12 pre-trained MNIST neural networks with ReLU activation functions. These networks were also part of the work by König *et al.* [25] and are widely used in the literature on neural network verification. To study the robustness distributions of standard neural networks, in this first analysis, we only considered neural networks that were not adversarially trained. While the previously mentioned work investigated 15 neural networks, 3 of these led to various errors, caused by input inconsistency and out-of-memory issues, and we therefore omitted these from this part of our study. The test accuracy of these networks ranges from 0.757 to 0.996. Further details on the number of input images investigated for each network and the training and testing accuracy of these networks can be found in Appendix B, Table 5.

Choice of input images. Following the work of König *et al.* [25], we used the first 100 instances from the MNIST training and testing sets, respectively. We only considered an image for a specific network if the image was originally correctly classified, meaning that the set of images in the distribution might vary over networks. The number of input images considered for each network can be found in Appendix B, Table 5. We briefly compare the robustness distributions obtained for 100 instances and 400 instances for 5 networks in Appendix C.

In total, we endeavoured to determine $\tilde{\varepsilon}^*$ for 1 147 instances based on training images and 1 154 instances based on testing images for the 12 networks. Note that each of these instances required one or more verification queries to determine $\tilde{\varepsilon}^*$; The time-out for each of these queries was set to one hour. We found a $\tilde{\varepsilon}^*$ for each of the images we investigated for each of the networks.

Algorithm setup. We ran k-binary search with 200 ε values, ranging from 0.001 to 0.4, in intervals of 0.002, i.e., $(0.001, 0.003, \ldots, 0.397, 0.399)$. Consistent with previous work, we selected 0.001 as the smallest possible value [25, 36], and we opted for 0.4 as the highest possible value to ensure that all images could be perturbed in a manner that would cause a misclassification. Cicalese *et al.* [11] have analysed the exact maximum length n of values in

[4] https://www.vnnlib.org/

the range for k-binary search such that there exists an optimal strategy of a maximum of t queries when at most k queries are started simultaneously. Using their formula, we determined that using $k = 2$ simultaneous queries will result in a theoretical maximum of 11 verification queries with different ε, which is the lowest number of queries compared to all possible values for k, to find $\tilde{\varepsilon}^*$ within the 200 intervals.

However, we note that Cicalese *et al.* [11] assume a maximum of one time-out in the ε range. In our setting, we have an unknown number of time-outs, which renders the problem more complex. Using two parallel queries, we split the remaining search space into equal-sized parts every time. When one of the queries finishes while the other query is still running and we have no new information about the unfinished query, we will find a new ε that is at the midpoint of the new range of ε. The method used for verification in this section was BaB [15], which was chosen because it was easy to use out of the box and did not require dedicated GPU resources. The latter restriction was necessary since using k-binary search (for $k > 1$) with GPU acceleration led to several problems, including unstable communication between the root node that coordinates the binary search and the k different GPU nodes running the individual verification queries.

Execution environment. All experiments were carried out on a cluster of machines, each equipped with 2 Intel Xeon E5-2683 CPUs with 32 cores, 40MB cache size and 94GB of RAM. The amount of RAM available for each verification query for one image and network depended on the choice of k; we always used one dedicated CPU core per verification query and restricted each verification query to a time budget of one CPU hour. We used Python 3.10 with CentOS 7.0.

Kolmogorov-Smirnov test. In the following, we use the Kolmogorov-Smirnov test [24, 23, 42] as a *goodness-of-fit* and as a *two-sample* test. In the first case, the null hypothesis states that a sample stems from a given distribution, while in the second case, it states that two samples are drawn from the same underlying distribution.

The test statistic captures the largest vertical distance between two cumulative distribution functions. Specifically, if n is the sample size of the first sample, $F_n(x)$ and $F(x)$ are two cumulative distribution functions, the first of which is derived from a given sample, while the second corresponds to a specific theoretical distribution or to another sample. Formally, the Kolmogorov-Smirnov test statistic is then defined as follows:

$$K_n = \sqrt{n} \quad \cdot \sup_{-\infty<x<\infty} |F_n(x) - F(x)| \tag{1}$$

Theoretically, the ε^* values may stem from a real-valued probability distribution, yet in this work, we are using a binary search algorithm and we only consider certain ε values from a discretised set. The Kolmogorov-Smirnov test can still be applied, due to the test being applicable even if the random variables x are not from a continuous distribution. Interested readers can find a comprehensive overview of the history and development of the test in the work of Darling [14].

4.2 Performance of k-binary Search

To investigate the effectiveness of k-binary search, we performed an extensive experiment using the 12 networks and the training images. We analysed several aspects of the behaviour of k-binary search by comparing various values of k, specifically, $k \in \{1, 2, 4, 8, 16\}$, where the algorithm runs each of the k queries in parallel on a dedicated CPU.

Number of queries. Cicalese *et al.* [11] describe a theoretical maximum of queries necessary to find an item based on the length of the search array for any odd k or $k \in \{2, 4\}$. This theoretical maximum applies when there is at most one missing answer, whereas in our study, we have an unknown number of missing values, e.g., due to out-of-memory errors or time-outs. These theoretical maxima for the k values considered in this research are shown in Table 1.

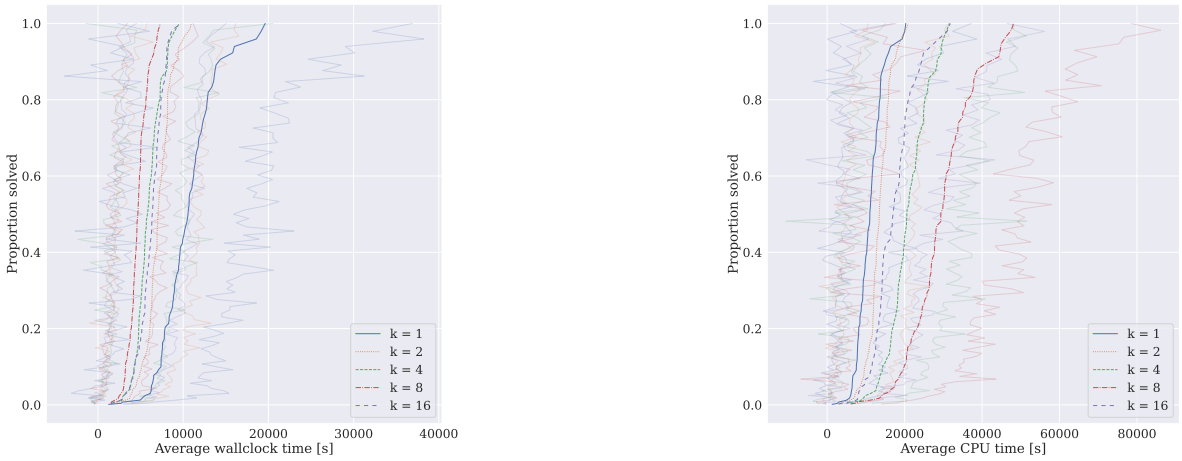
Table 1. Comparison of the efficiency of finding $\tilde{\varepsilon}^*$ using k-binary search with different values of k. The theoretical maximum number of queries per instance was calculated following Cicalese et al.(2004); because in practice, there is a possibility of encountering many time-outs, we observed different maximum values in our work. The table also shows the actual minimum and average number of queries required to find $\tilde{\varepsilon}^*$ for a single instance. The average number of time-outs is the average number of time-outs encountered when searching for $\tilde{\varepsilon}^*$ for a single image for one network, averaged across networks. The values in this table are based on all training images for all 12 conventionally trained MNIST networks considered in our study, using a 1-hour CPU time limit per verification query.

	k	1	2	4	8	16
number of queries per instance	Theoretical maximum	8	11	17	n/a	n/a
	Minimum	7	4	7	15	4
	Maximum	74	19	31	54	54
	Average	12.68	11.78	18.33	28.51	22.46
Time (1h time-out)	Average CPU time per instance [s]	10 162	12 750	19 582	27 234	15 657
	Average wallclock time per instance [s]	9 781	6 717	5 574	4 433	5 878
	Average number of time-outs per instance	0.10	0.17	1.07	1.46	0.19

Our empirical analysis, see Table 1, shows that the average number of verification queries exceeds the theoretical maximum in all cases for which a theoretical maximum exists. This suggests that the assumption of having at most one missing answer is not a realistic assumption for the setting of neural network verification with the selected time limit. Therefore, there is room for more theoretical work that studies the more complicated cases, in which there can be an arbitrary number of missing answers. In our experiments, considering the training data for the twelve conventionally trained networks, we investigated a total of 107 495 verification queries, of which 3 432 resulted in time-outs and 6 205 in out-of-memory errors. In finding the $\tilde{\varepsilon}^*$ for a given image and network, we encountered an average of 1.68 missing answers to queries, due to time-outs and out-of-memory errors, yet there might be more, since we did not need to verify all possible ε values for each network and image, due to the use of binary search. Table 4 in Appendix A provides further information about the number of time-outs and out-of-memory errors encountered for every k investigated in our experiments.

We can see that $k = 2$ resulted in the lowest average number of verification queries needed as well as the lowest maximum. However, on average, using $k = 2$ did result in a slightly higher number of time-outs per query. Using $k = 1$ resulted in the highest maximum number of queries while resulting only in the second-highest average number of queries. Only considering the number of queries, we can see that $k = 8$ performed worst in our empirical analysis, as it required 28.51 queries on average and a minimum of 15 queries. From these results, it appears that the theoretically best choice, $k = 2$, also works best in practice. It might seem counterintuitive that the wallclock time does not continue to decrease as k increases; however, every time we increase the number of queries that can be verified simultaneously, the amount of available memory per query decreases. We note that, in our experiments, all choices of k eventually led to the same $\tilde{\varepsilon}^*$.

Running time analysis. Our empirical analysis showed that when considering the need to minimise the number of queries to run, it is best to select $k = 2$. In practice, a resource that often should be minimised is time spent on



(a) (b)

Fig. 1. Cumulative distribution of the fraction of instances solved by k-binary search with different values of k within a certain amount of wallclock time (left) and CPU time (right) for the 12 networks considered in our study and the MNIST training images. For each instance, we have taken the average running time over all 12 considered networks. The lines in the background represent the standard deviation in running time or CPU time at each point in the cumulative distribution function.

solving the problem, or in this case, on creating the robustness distributions. In the following, we will analyse the efficiency of different k values in terms of wall clock and CPU time.

Figure 1 shows CDF plots of (a) the average wallclock time and (b) the average CPU time. We would expect that higher degrees of parallelism (expressed by the number of parallel queries k) would result in lower wallclock times but higher CPU times (due to the fact that there might be redundant queries). Based on the figures, we see that this pattern mostly holds, i.e., we observed it for all values except for $k = 16$. Surprisingly, $k = 16$ has a relatively higher wallclock time than expected compared to the other values of k, but a relatively lower CPU time. Note that a higher degree of parallelism also implies a lower amount of RAM per query (see Section 4.1). For this reason, we conclude that while a higher degree of parallelism potentially improves the wallclock time, as expected, there is a limit to which this can be beneficial. In the case of our experiments, this limit was encountered at $k = 8$.

4.3 Robustness Distributions on Training Data

In this section, we report the results from our analysis of the robustness distributions of fully connected ReLU neural networks on training data. Our goal in this investigation was to characterise the shape of the distributions and determine how the robustness distributions of different networks relate to each other.

Figure 2 shows a boxplot of the robustness distribution of the training set of the 12 MNIST classifiers. The networks are sorted by the minimal $\tilde{\varepsilon}^*$ from the empirical distribution. This data strongly indicates that, depending on the settings and the amount of input perturbation that is tolerated (as specified by ε), different networks would be preferable based on robust accuracy. For example, for an ε value of 0.012, which has been widely used in other work, network *relu_4_1024* has a robust accuracy of 93% and network *net* a better robust accuracy of 100%. However, had we chosen an ε value of 0.04, network *relu_4_1024* and *net* have a robust accuracy of 66%

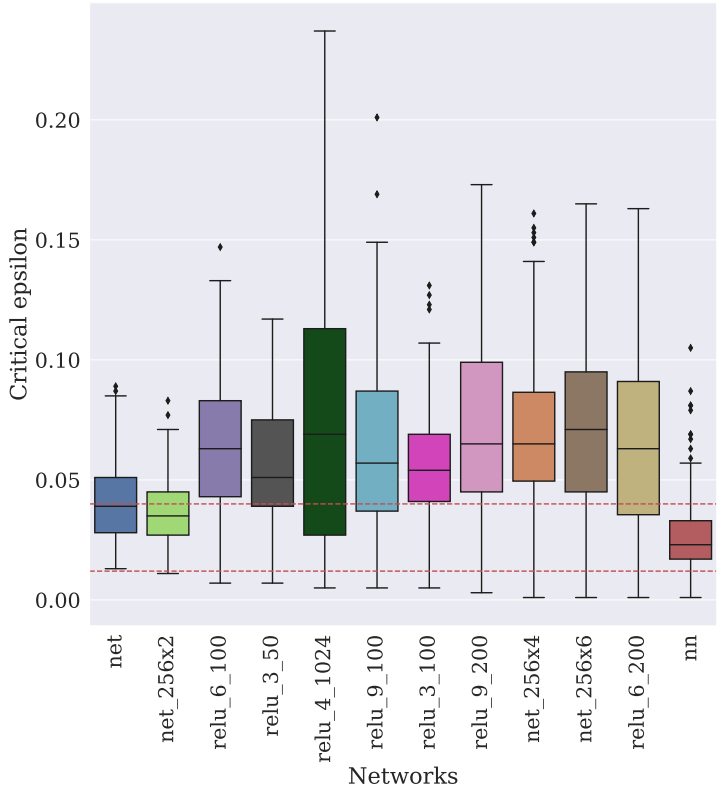
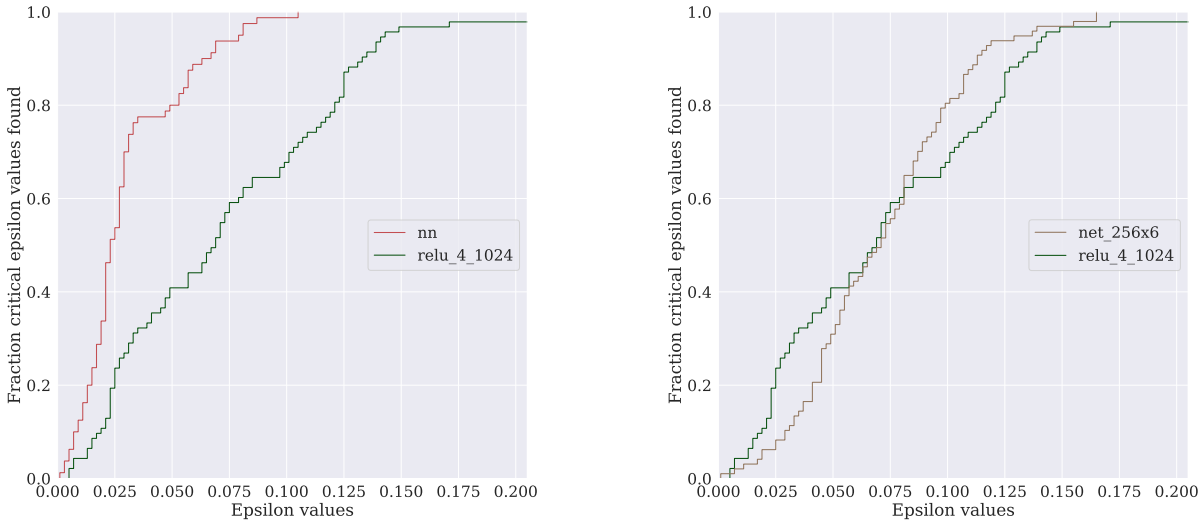


Fig. 2. Boxplots of the distributions of $\tilde{\epsilon}^*$ for 12 MNIST classifiers over their set of correctly classified inputs from the training set used in our experiments. The networks are ordered from the largest to the smallest minimal $\tilde{\epsilon}^*$ value over the images in the training set. We note that the individual distributions do not include images that were originally misclassified by the network. The plot contains two horizontal lines, at 0.012 and 0.04, to indicate the substantial differences in robust accuracy for different networks at those ϵ values.

and 49% respectively. There seems to be no clear relationship between the minimum and the median of a critical epsilon distribution. While the minimum of the critical ϵ distribution of network *relu_4_1024* is relatively low, it has the highest median robustness and is indeed one of the most robust networks over the entire training set.

Figure 3 shows cumulative distribution functions of the robustness distribution for the classifiers with the highest, second-highest and lowest median $\tilde{\epsilon}^*$. CDF plots are particularly useful, since the robust accuracy of a given network for arbitrary ϵ values can be easily determined from them (note that we consider the definition of robust accuracy where we omit originally misclassified images). We observed two qualitatively different scenarios: Figure 3a shows a situation where one network (*relu_4_1024*) consistently exhibited higher robustness for all ϵ values, while in Figure 3b, a case is seen where one network (*relu_4_1024*) achieved higher robustness for ϵ values below 0.06, whereas the other displays higher robustness for ϵ values of 0.08 and above.



(a) (b)

Fig. 3. Empirical CDF plots for the empirical robustness distributions for three neural networks on training data. The figure on the left shows the robustness distributions for the two networks with the robustness distributions with the highest median $\tilde{\varepsilon}^*$, and the figure on the right shows the robustness distributions of the networks with the highest and lowest median $\tilde{\varepsilon}^*$, respectively.

4.4 Robustness Distributions on Testing Data

For robustness distributions to eventually be useful for practical purposes, they need to generalise beyond the training data of a given neural network. To directly assess this, we compared robustness distributions for the same networks between training and testing data.

Figure 4 shows boxplots of the distributions of $\tilde{\varepsilon}^*$ across training and testing sets, respectively. Clearly, the boxplots for training and testing data show a high degree of similarity. Further evidence for the similarity between robustness distributions on training and testing data can be seen from the CDFs shown in Figures 5a and 5b, showing the network with the highest and lowest median $\tilde{\varepsilon}^*$, respectively.

Using the Kolmogorov-Smirnov test with a standard significance level of 0.05, we found no evidence that the minor differences in robustness distributions for the same network on training and testing data are statistically significant. This suggests that finding the robustness distribution for a given training set is sufficient for analysing the overall robustness of a network in a supervised learning scenario.

Furthermore, we found evidence that the robustness distributions for the networks considered could be characterised well by log-normal distributions, except for *net-256x4*; using the Kolmogorov-Smirnov test with a standard confidence level of 0.05, this distributional hypothesis was not rejected for any of the other 11 networks. This finding is significant, as it enables reasoning about the distribution of $\tilde{\varepsilon}^*$ values for unseen instances drawn from the same distribution as the testing images. Therefore, it should be possible to determine the robustness of a neural network without the need for costly evaluation of a large number of inputs.

A possible explanation of this phenomenon might be rooted in the fact that the gradients of conventionally trained neural networks tend to follow near log-normal distributions [10]. It is possible that the gradient loosely influences the size of ε^*, although the precise nature of this effect remains unknown at this time.

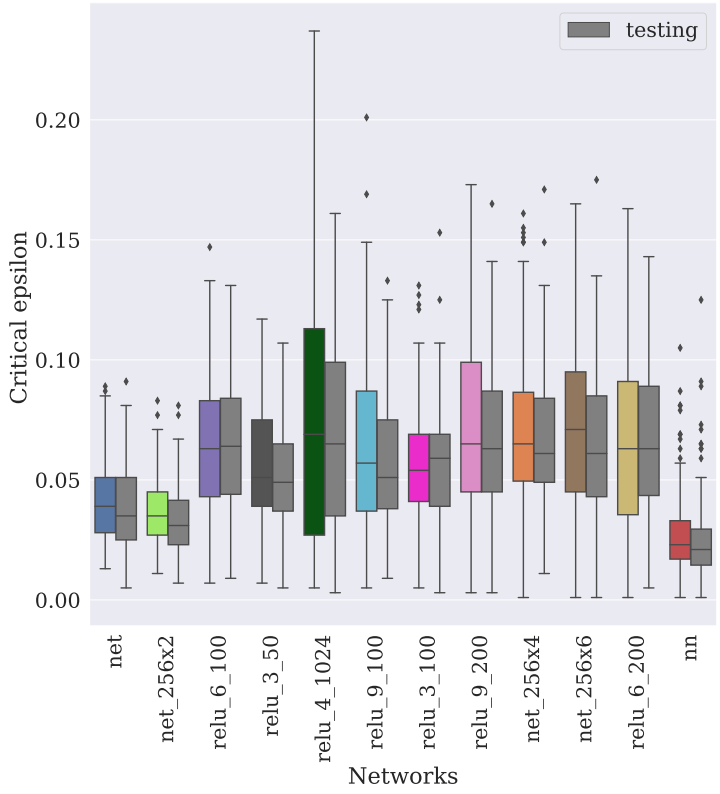
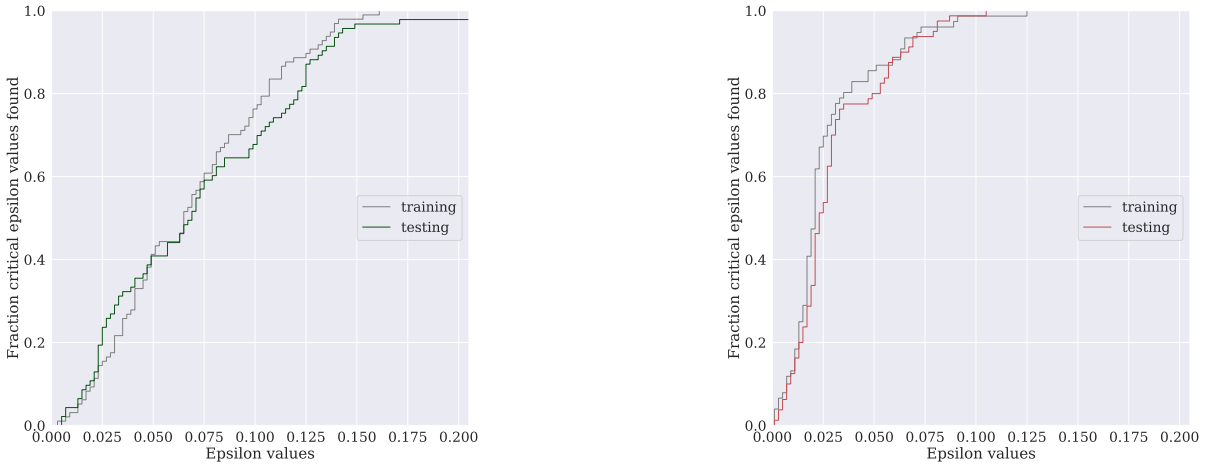


Fig. 4. Boxplots of the distributions of $\tilde{\epsilon}^*$ for 12 MNIST classifiers over their sets of correctly classified inputs on training and testing data. The networks are ordered from the largest to the smallest minimal $\tilde{\epsilon}^*$ value over the training sets. We note that the individual distributions do not include images that were originally misclassified by the network.

4.5 Correlation Analysis

We now investigate the correlation between the size of $\tilde{\epsilon}^*$ values of the same input for different networks. In particular, we would like to understand whether the same inputs are easily perturbed to be misclassified for different networks or whether relative robustness varies depending on the network. This is relevant, as there might be inputs that are more important and for which the networks must be robust to perturbations. An understanding of this might have a substantial impact on preferences for one network over another in real-world use cases.

Figure 6a shows the Spearman correlation coefficient r_s of $\tilde{\epsilon}^*$ for training images from each of the 12 conventionally trained networks. Spearman correlation coefficients were calculated only based on images that were correctly classified by both networks. When $r_s = 0$, there is no correlation between the size of perturbation necessary to make two neural networks misclassify a formerly correctly classified image, and when $r_s = 1$, there is a perfect monotonic relationship between the size of perturbation necessary for two different networks. We chose to use the Spearman correlation coefficient over the standard Pearson correlation coefficient since it does not require the assumption of a linear dependence between the two random variables describing the data to



(a) network: *relu_4_1024*

(b) network: *nn*

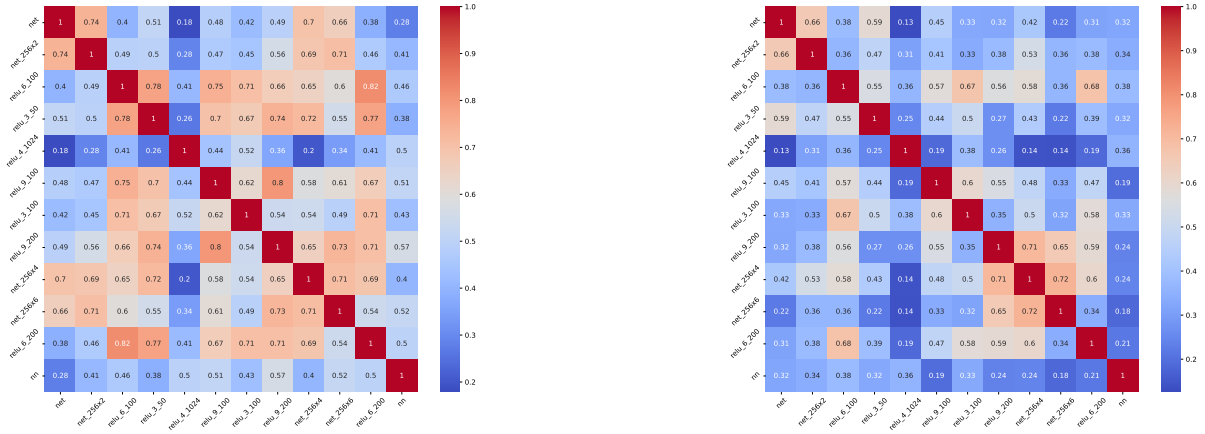
Fig. 5. Empirical CDF plots for the empirical robustness distributions for the neural networks with the highest (left) and lowest (right) median $\tilde{\varepsilon}^*$ on training and testing data. The colours of the training distribution of a network correspond to those used in Figure 2.

be analysed. In Figure 6a we can see that the correlation of $\tilde{\varepsilon}^*$ between networks varies from slight correlation ($r_s = 0.18$) to strong correlation ($r_s = 0.82$), and on average, we observed a correlation of $r_s = 0.55$.

We note that some networks that have a similar architecture, such as *relu_6_100*, *relu_6_200*, and *relu_9_100*, tend to have higher correlation amongst each other, while architectures that are very different from each other, such as *nn* compared to all other networks, tend to show lower correlation. This could indicate that the robustness of a given input against perturbation partially depends on the architecture of the given network. However, as the different networks considered here are trained similarly, we cannot rule out that this might not hold for similar architectures trained using varying regimens.

Figure 6b shows the correlation coefficient r_s of $\tilde{\varepsilon}^*$ for MNIST testing images between each of the 12 conventionally trained networks. We can see that for all pairs of networks except *net_256_4* and *relu_9_200*, the correlation coefficient is lower than in Figure 6a, with many pairs that would have a high correlation for training data showing moderate to low correlation for testing data; for example, the correlation coefficient between *relu_9_200* and *relu_3_50* is 0.74 and 0.27 for training and testing data, respectively. This is also reflected in the average correlation of $r_s = 0.39$ for testing data. This suggests that the training processes typically ensure that networks exhibit similar robustness for the same images, but less so for the robustness of test images; it does not imply that networks are less robust on test images (as we discussed in Section 4.4).

Figure 7 shows instance-level comparisons of $\tilde{\varepsilon}^*$ for the set of MNIST training and testing images for the networks with the highest and lowest correlation for training and testing data, respectively. We will first describe the figures based on training data, *i.e.*, Figure 7a and Figure 7b. In Figure 7a, we can see that even though there is some variation, the relationship between $\tilde{\varepsilon}^*$ of the two networks is quite strong. One possible explanation in this case is the similarity in architecture between these two neural networks. In Figure 7b, we observe a very different situation: The correlation coefficient of $\tilde{\varepsilon}^*$ of training images for *net* and *relu_4_1024* is 0.18, indicating a weak relationship between the difficulty of perturbing an image to be misclassified for the two networks.



(a) training data

(b) testing data

Fig. 6. Heatmap for the Spearman correlation between the $\tilde{\varepsilon}^*$ for training, depicted on the top, and testing images, depicted on the bottom. The networks depicted are 12 conventionally trained MNIST networks considered in our study. Spearman correlation coefficients were calculated only based on images that were correctly classified by both networks.

Figure 7c and 7d show instance-level comparisons of $\tilde{\varepsilon}^*$ for the set of MNIST testing images. In Figure 7c we see the considerable positive correlation between instance level $\tilde{\varepsilon}^*$ for *net* and *net_256x2*; various images have identical $\tilde{\varepsilon}^*$ for both networks. In Figure 7d the lack of correlation between the instance level $\tilde{\varepsilon}^*$ for *net* and *relu_4_1024* is recognisable by the large variation; there are, however, still several instances with similar $\tilde{\varepsilon}^*$ values.

It is well-known that certain adversarial examples transfer between networks (see e.g., [38]), meaning that a given adversarial example can mislead multiple networks. However, we note that when a given adversarial example misleads two networks, this does not necessarily imply that the critical $\tilde{\varepsilon}^*$ is similar for both networks. For this reason, our analysis is orthogonal to the findings of Papernot *et al.* [38].

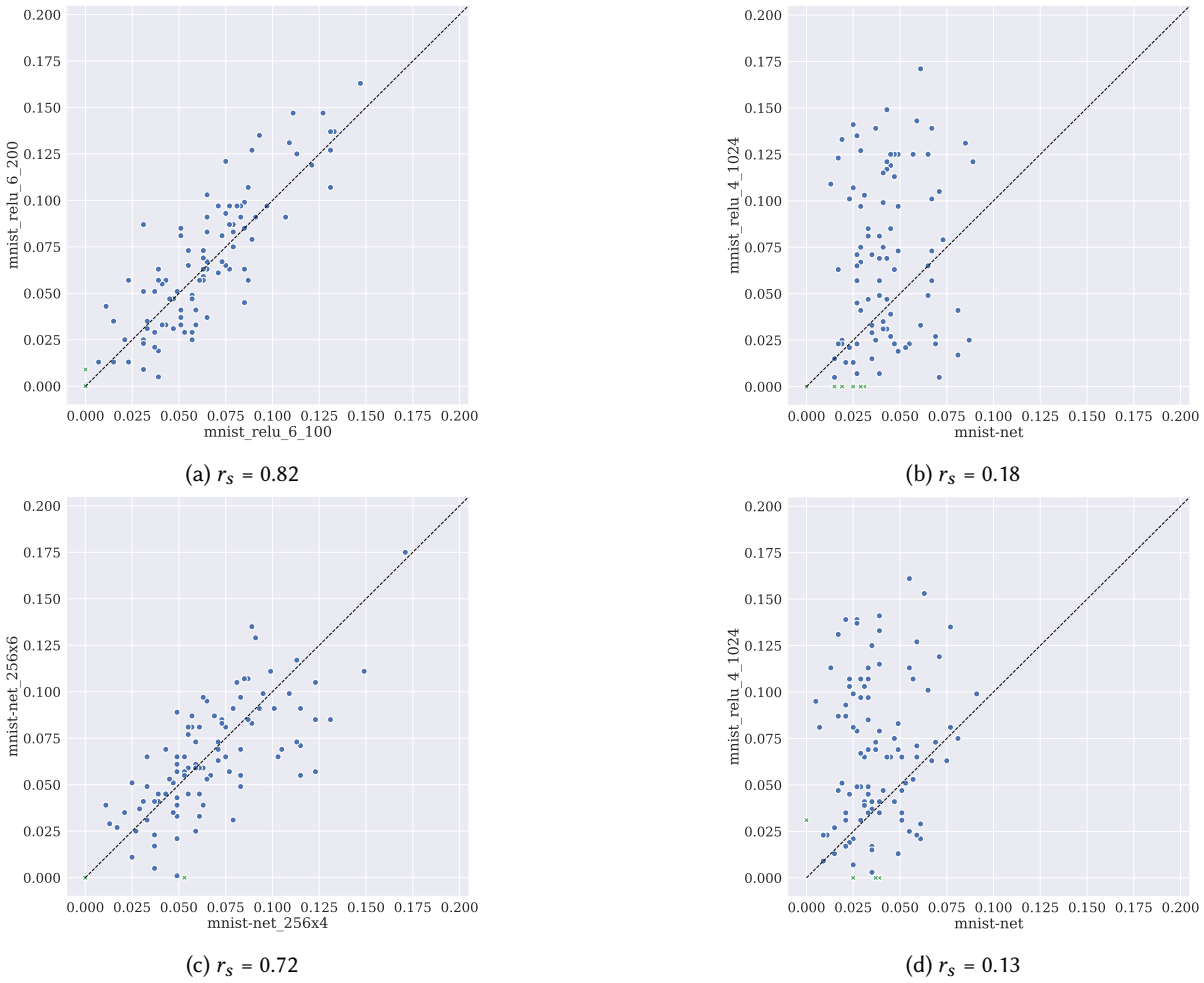
5 Empirical Investigation on Adversarially Trained Fully-Connected Neural Networks

In a second set of experiments, we investigated the effect of adversarial training on the robustness distributions of fully-connected neural networks. Adversarial training improves the robustness of a neural network, which should be clearly reflected in the respective robustness distributions. In addition to experiments with MNIST networks, we have obtained robustness distributions for networks trained on the CIFAR-10 [26] and GTSRB [21] datasets. Our main results of these experiments are discussed in Section 5.3 after we discussed the experimental set-up and the quality of the obtained robustness distributions.

5.1 Setup of Experiments

Choice of networks. In this section, we selected the *mnist_nn* and *mnist_relu_4_1024* architectures, as these have the lowest and highest median $\tilde{\varepsilon}^*$ in the robustness distributions from the previous section, respectively. This gives a versatile view of how robustness improves when networks are trained adversarially.

We additionally investigated networks trained on the CIFAR-10 and GTSRB datasets. CIFAR-10 is included because it is a more complex dataset commonly studied in neural network verification, while GTSRB was chosen for its practical application in traffic sign classification.



(a) $r_s = 0.82$

(b) $r_s = 0.18$

(c) $r_s = 0.72$

(d) $r_s = 0.13$

Fig. 7. Scatterplot of $\tilde{\epsilon}^*$ for different training and testing images, where each point corresponds to one image. Figures 7a and 7b show the $\tilde{\epsilon}^*$ of the networks with the highest and lowest Spearman correlation, respectively, for the training images. Figures 7c and 7d show the $\tilde{\epsilon}^*$ of the networks with the highest and lowest Spearman correlation, respectively, for the testing images. The images that were originally misclassified by at least one of the networks are coloured green and are represented as a cross, and the rest are coloured blue and represented as dots. Spearman correlation coefficients were calculated only based on images that were correctly classified by both networks.

For the CIFAR-10 dataset, we have selected *CIFAR_7_1024*, a regularly used architecture in the neural network verification literature [25, 36]. We included only this single fully-connected network due to the generally low accuracy of such models on CIFAR-10, but believe that this inclusion is necessary to provide a complete picture of the robustness of different architectures.

For the GTSRB experiments, the first fully-connected network we selected is inspired by the work of Mohapatra *et al.* [33], which introduced a verification approach against semantic perturbations. This is a 6-layer ReLU network

with 256 nodes in each hidden layer, and we refer to it as *GTSRB_6_256*. The second FFNN we consider has the same architecture as *CIFAR_7_1024*.

Adversarial training. Following Wong *et al.* [50], we use FGSM, as introduced in Section 2.4, training with non-zero initialisation. For MNIST, we used a perturbation of 0.2 and for CIFAR-10 and GTSRB 8/255. For PGD training, as introduced in Section 2.4, we used the training protocol from Madry *et al.* [31]. For MNIST, we used a perturbation of 0.3, and for CIFAR-10 and GTSRB, we used a perturbation of 8/255. For GTSRB, we used the same training protocol as for CIFAR-10. For all training methods, we performed hyperparameter optimisation using Optuna [1]; the final hyperparameter values can be found in Appendix I, Tables 21 and 22.

Choice of input images. We used the same MNIST instances as in Section 4.1. For both CIFAR-10 and GTSRB, we randomly selected 100 testing and training images each, with random seed 42. Given that the GTSRB dataset contains 42 classes, we performed stratified random selection.

The accuracy of the CIFAR-10 network is notably low, and the number of verified instances is also quite limited, particularly for the adversarially trained networks (refer to Appendix E, Table 9). However, we chose not to include additional instances solely based on their verifiability, as doing so could introduce bias towards easily verifiable images and result in robustness distributions that lack generalisability.

Algorithm setup. For MNIST, we used the same k-binary search setup as before (i.e., $k = 2$). The GTSRB and CIFAR-10 experiments were performed using α, β-CROWN for verification [57], which is considered state of the art in neural network verification. This deviation of verifier use from the previous section is motivated by the size of the architectures we employ for these datasets. We use GPU-accelerated α, β-CROWN, and as there are no methods currently known to us that allow for efficient and dependable status communication between multiple GPU nodes involved in verification tasks and a root CPU node that coordinates the k-binary search, we use 1-binary search in this section. Furthermore, for k-binary search to be as efficient as possible, verification queries need to be terminated while running, which turned out to be practically impossible in our GPU setup. Therefore, we used standard binary search ($k = 1$) for CIFAR-10 and GTSRB, using 102 ε values ranging from 0.00 to 0.4 in increments of $\frac{1}{255}$. This approach differs from the one used for MNIST, as our focus was solely on investigating full pixel differences.

Execution environment. The experiments for MNIST were run in the same execution environment as outlined in Section 4.1. The GTSRB and CIFAR-10 experiments were carried out on a second cluster of machines, each equipped with 4 NVIDIA GeForce GTX 1080 Ti GPUs and 11 GB VRAM per GPU. For both datasets, we set a time restriction of one hour of wallclock time per query; wallclock time was chosen because of our use of GPUs. We used Python 3.10 with CentOS 7.0

5.2 Verification Gaps and Quality of Robustness Distributions

In this section, we investigate the extent to which the added complexity of adversarial training and more sophisticated datasets, specifically CIFAR-10 and GTSRB, affect robustness distributions.

Table 2 shows an overview of quality measures for the robustness distributions determined in our experiments. We note that the percentage of cases in which a critical termination occurs for conventionally trained MNIST networks is higher than the value reported in Section 4.2, as there, the average was taken over all twelve networks and training as well as testing data. The percentage of instances for which a critical termination occurred increases significantly when the networks are trained adversarially, compared to conventional training. Furthermore, for the CIFAR-10 network, a large percentage of instances could not be solved at all (i.e., not a single query was successful). As the robustness distributions of this network have a large percentage of unsolved instances (making

it of low quality, see Section 3.2) and since the network has a low general accuracy, we have decided to exclude it from further analysis.

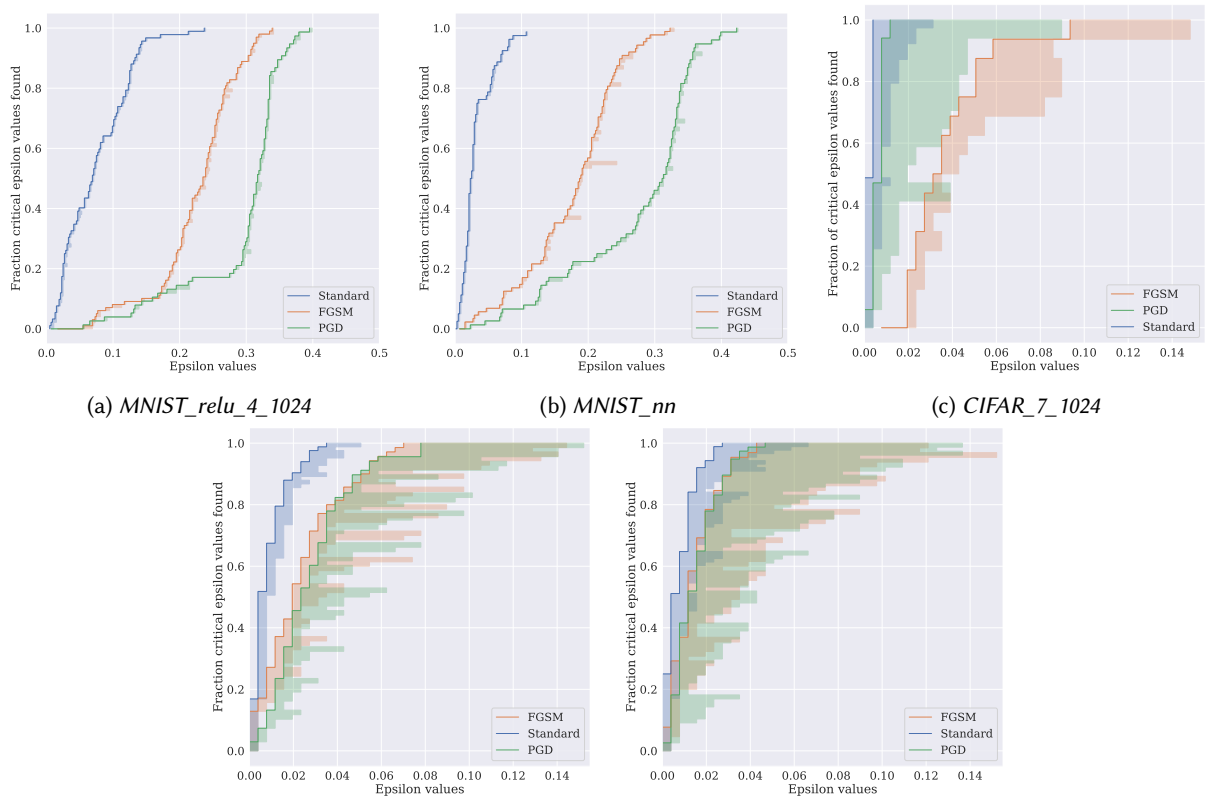
Table 2. Details on the quality of the robustness distributions of the fully-connected neural networks for the MNIST, CIFAR-10 and GTSRB datasets. Test accuracy was measured over the entire test set for each respective network. KS-test refers to the Kolmogorov-Smirnov test we used to test for log-normality with a significance level of 0.05. If an empirical robustness distribution does not pass this test, there is evidence that it significantly deviates from a log-normal distribution. A critical termination occurs when for an instance, there are one or more out-of-memory or time-out errors for queries falling between $\tilde{\varepsilon}^*$ and $\tilde{p}$. Unsolved instances refer to cases where we could not find a single solution for any verification query for a given instance due to out-of-memory or time-out errors. The average verification gap is the average gap between $\tilde{\varepsilon}^*$ and $\tilde{p}$ for all solved instances over the testing data. Note that the minimum interval between ε-values tested is 0.002 for the MNIST dataset, and 0.0039 for the CIFAR-10 and GTSRB datasets.

Network	Training	test accuracy	KS-test passed		% critical terminations	% unsolved instances	Average verification gap
			train	test	test	test	test
MNIST_nn	Standard	0.757	True	True	4.938	0	0.002
	FGSM	0.910	False	False	31.461	0	0.003
	PGD	0.802	False	False	35.065	0	0.003
MNIST_relu_4_1024	Standard	0.941	True	True	19.355	0	0.003
	FGSM	0.978	False	False	36	0	0.003
	PGD	0.782	True	False	31.169	0	0.003
CIFAR_7_1024	Standard	0.540	True	False	43.860	29.825	0.008
	FGSM	0.367	True	True	31.429	51.429	0.017
	PGD	0.510	False	False	29.787	61.702	0.023
GTSRB_6_256	Standard	0.800	False	False	25	0	0.005
	FGSM	0.604	True	True	66.197	0	0.019
	PGD	0.613	True	True	66.667	0	0.005
GTSRB_7_1024	Standard	0.794	False	False	62.921	0	0.010
	FGSM	0.586	True	False	75.757	0	0.028
	PGD	0.673	False	True	84.615	0	0.029

In Figure 8, we show the robustness distributions for the three datasets for the fully-connected neural networks with different training regimens. We can see that, indeed, the number of critical terminations for both CIFAR-10 and GTSRB networks is substantially higher than for the MNIST networks. Using the Kolmogorov-Smirnov test with a significance level of 0.05, we conclude that for the GTSRB and CIFAR-10 networks, the $\tilde{\varepsilon}^*$ and $\tilde{p}$ distributions are significantly different from each other, except for the CIFAR-10 architecture trained with FGSM. The large number of critical terminations is caused by the complexity of the verification queries around the transition phase, i.e., the ε values that are close to the boundary between safe values and values leading to adversarial examples. This indicates that the distributions are not of high quality as there is a high uncertainty in the actual shape of robustness distributions, except for the MNIST networks and CIFAR-10 architecture trained with FGSM, see Appendix K, Table 23.

5.3 Robustness Distributions of Adversarially Trained Fully-Connected Neural Networks

In this section, we report the main results of our empirical investigation of the influence of adversarial training on the robustness distributions of a fully-connected neural network. Our goal was to demonstrate that $\tilde{\varepsilon}^*$ distributions can give a nuanced picture of the effects of adversarial training on the robustness of a given neural network.



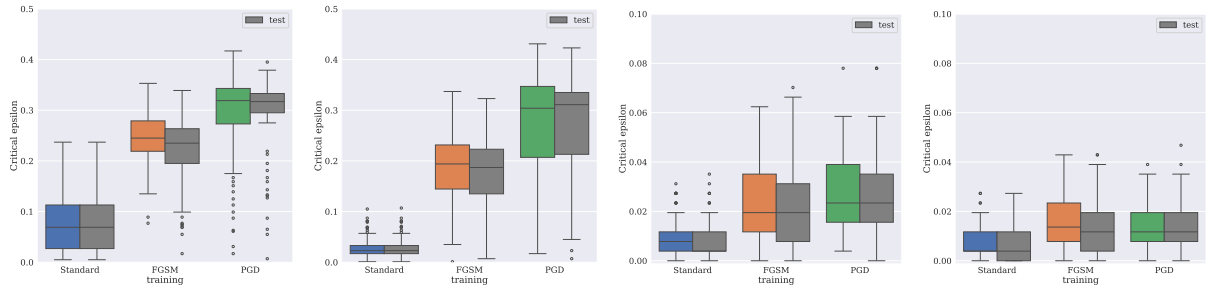
(a) *MNIST_relu_4_1024* (b) *MNIST_nn* (c) *CIFAR_7_1024*

(d) *GTSRB_6_256* (e) *GTSRB_7_1024*

Fig. 8. Empirical CDF for the empirical robustness distributions of neural networks across MNIST, CIFAR-10, and GTSRB datasets. The shaded area illustrates the difference between $\tilde{\epsilon}^*$ and the upper bound $\tilde{p}$ of the respective image. (a) and (b) show the MNIST networks with the highest and lowest median $\tilde{\epsilon}^*$. (c) illustrates the CIFAR-10 architecture trained with different methods. (d) and (e) show the GTSRB networks with the highest and lowest median $\tilde{\epsilon}^*$. Note that the axis scaling differs for different datasets.

Figure 8 shows that adversarial training increases the robustness of the networks we studied in allmost all cases. While this is clear for most networks, for *CIFAR_7_1024* and *GTSRB_7_1024* we visually confirm this due to overlap between the $\tilde{\epsilon}^*$ and $\tilde{p}$ of some distributions. The mean $\tilde{\epsilon}^*$ for all distributions were quite larger after using adversarial training, see also Appendix E. Figure 9 provides boxplots of the same distributions, and again, shows the increase in $\tilde{\epsilon}^*$ when using adversarial training. Surprisingly, we not only found that adversarial training increases the robustness of the least robust inputs, but we also observed an increase in the robustness for the entire set of verified inputs. The latter is clearly visible from the respective robustness distributions, which demonstrates once more how the nuanced view of robustness provided by these can produce new insights.

Table 2 shows whether the robustness distribution of a neural network passed the Kolmogorov-Smirnov test. Interestingly, we observed that although the robustness distributions of the standardly trained MNIST neural networks appear to follow a log-normal distribution, adversarial training changes this; for the GTSRB networks, we see the opposite as the robustness distributions of the adversarially trained networks seem to follow



(a) *MNIST_relu_4_1024* (b) *MNIST_nn* (c) *GTSRB_6_256* (d) *GTSRB_7_1024*

Fig. 9. Boxplots of the empirical robustness distributions for the MNIST and GTSRB neural networks in comparison to the adversarially retrained versions of these networks on training and testing data. Note that the axis scaling for the GTSRB networks differs from that for the MNIST networks. Using the KS-test with significance level 0.05 we did not find any significant differences between the distributions for training and testing data.

a log-normal distribution where that of the standardly trained does not. Additionally, the robustness distributions show distinct shapes across different datasets and training methods, though no consistent correlations were found between datasets.

Furthermore, we found evidence that adversarial training generally increases the computational complexity of verifying the networks, examined in this study while also enhancing their robustness. For example, for *MNIST_nn* the number of queries that timed out was 69 and 59 times higher for FGSM and PGD training, respectively compared to standard training (see also Appendix F).

6 Empirical Investigation on Adversarially Trained Convolutional Neural Networks

In a third set of experiments, we investigated the effect of adversarial training on the robustness distributions of convolutional neural networks. We did this in a way similar to the second set of experiments, except that we used a GPU-based method for the verification of the networks trained on MNIST as well, due to the nature of the network architectures we considered. The main results of these experiments are discussed in Section 6.3 after we discussed the experimental set-up and the quality of the obtained robustness distributions.

6.1 Setup of Experiments

In the following analysis, for each dataset, we use the same instances for verification as in Section 5.1 and we use the same methods for training the different architectures.

Choice of networks. We selected two architectures from the literature [32], *convSmall* and *convMedG*, which were both trained using three different methods. These networks were also part of the study performed by König *et al.* [25]. Both networks were trained on MNIST data and contained convolutional layers as well as ReLU activation functions. *convSmall* was trained by Mirman *et al.* [32] using conventional training, denoted *Point* in the literature, as well as DiffAI and PGDK adversarial training (see also Section 2). *convMedG* was trained using conventional training as well as PGDK with two different perturbation radii, namely 0.1 and 0.3, which specify the maximum size of the perturbation used in adversarial training. These networks were also considered in the work of König *et al.* [25] and achieve a testing accuracy ranging from 0.977 to 0.991. Additional specifics regarding the number of images used for each network, as well as comprehensive training and testing accuracy scores, can be found in Appendix E, Table 8.

For the analysis of the CIFAR-10 networks, we selected two convolutional networks, *ConvBig* and *resnet_4b*. While *ConvBig* was used by König *et al.* [25] and Müller *et al.*[35], *resnet_4b* was selected from the VNNCOMP 2021 CIFAR-10 benchmark [2].

Finally, we incorporated two convolutional neural networks trained on GTSRB. The architecture of these is based on the network from the work by Wu *et al.* [51], which addresses the maximum safe radius and feature robustness problems under the assumption of Lipschitz continuity. We refer to this architecture as *GTSRB_cnn_deep_game*. It is a 10-layer network, featuring 4 convolutional layers and a max-pooling layer. The authors used the same network architecture for CIFAR-10 and GTSRB, but since only the CIFAR-10 version is available online, we reconstructed and trained the network on the GTSRB dataset from scratch for this work.

The second CNN is inspired by a network from the VNNCOMP 2023 [6] GTSRB benchmark, which featured binarised neural networks. We included the architecture referred to as *Accuracy Efficient Architecture for GTSRB and Belgium Dataset*, although, for simplicity, we call it *GTSRB_cnn_vnncomp23*. This is a 5-layer network, featuring 3 convolutional layers. We retrained this network from scratch without binarisation, which we consider to fall outside of the scope of this work. The *GTSRB_cnn_vnncomp23* network trained with FGSM led to verification errors from α, β-CROWN, which is why we have excluded it from our analysis. Values for the training hyperparameters for the CIFAR-10 and GTSRB networks can be found in Appendix I, Tables 21 and 22

Algorithm setup. The neural networks for MNIST used in this part of the study contain convolutional layers, which could not generally be solved by the BaB verifier used in Section 4 and 5.1. This choice prompted us to use standard binary search for verification of all networks, as in Section 5.1 with α, β-CROWN for verification. We used the same range of ε values for the MNIST networks as we did in Section 5.1. The choices for the other two datasets are the same as described in Section 5.1.

Execution environment. All experiments in this part of our study were carried out on a cluster of machines, each equipped with 4 NVIDIA GeForce GTX 1080 Ti GPUs and 11 GB video memory per GPU. We provided the same time budget of an hour in wallclock time but did not impose any memory constraints. We used Python 3.10 with CentOS 7.0

6.2 Verification Gaps and Quality of the Robustness Distributions

In this section, we investigate the effect of the added complexity of using convolutional networks. In these distributions, we encountered more uncertainty considering the $\tilde{\varepsilon}^*$, due to time-outs and out-of-memory errors. Our goal was to understand the extent to which this uncertainty affects our empirical results.

Table 3 shows an overview of quality measures for the robustness distributions. The percentage of instances for which a critical termination exists is larger than for the fully-connected networks considered in the previous section, and, as before, we observed an increase for adversarially trained networks. Notably, for the CIFAR-10 networks, a large percentage of verification instances once again could not be solved.

Figure 10 shows empirical CDF plots of the robustness distributions for the convolutional neural networks with different training regimens. The robustness distributions are generally of lower quality than those presented in the previous section; specifically, the verification gaps for almost all convolutional networks are rather large. As a notable exception, for the *convSmall* network trained with DiffAI rather than Point or PGDK, we found that the verification time decreased and the verification gaps turned out to be relatively small. We noticed that, while in Section 5.2 a non-zero $\tilde{\varepsilon}^*$ could be found for most instances, for the GTSRB networks, the robustness distributions generally have lower bounds of 0, and for CIFAR-10, even the conventionally trained networks have robustness distributions with lower bounds of 0. For this reason, and because CIFAR-10 contains a large number of verification instances that cannot be solved, we excluded the networks trained on these datasets from further

Table 3. Details on the quality of the robustness distributions of the convolutional neural networks for the MNIST, CIFAR-10 and GTSRB datasets. The test accuracy is measured over the entire test set for each respective network. KS-test refers to the Kolmogorov-Smirnov test we used to test for log-normality with a significance level of 0.05. If an empirical robustness distribution does not pass this test, there is evidence that it significantly deviates from a log-normal distribution. A test marked as invalid indicates a situation where our data did not adhere to the assumptions of the KS-test; this happened when there were verification instances for which we found a ε^* of 0, as log-normal distributions consist of strictly positive random variables. A critical termination indicates a situation where a verification instance gave rise to one or more out-of-memory errors or time-out errors for queries that lay between the $\tilde{\varepsilon}^*$ and $\tilde{p}$. Unsolved instances refer to cases where we could not find a single solution for any verification query for a given instance due to out-of-memory or time-out errors. The average verification gap is the average gap between $\tilde{\varepsilon}^*$ and $\tilde{p}$ for all solved instances over the testing data. Note that the minimum interval between ε-values tested is 0.002 for the MNIST dataset and 0.0039 for the CIFAR-10 and GTSRB datasets.

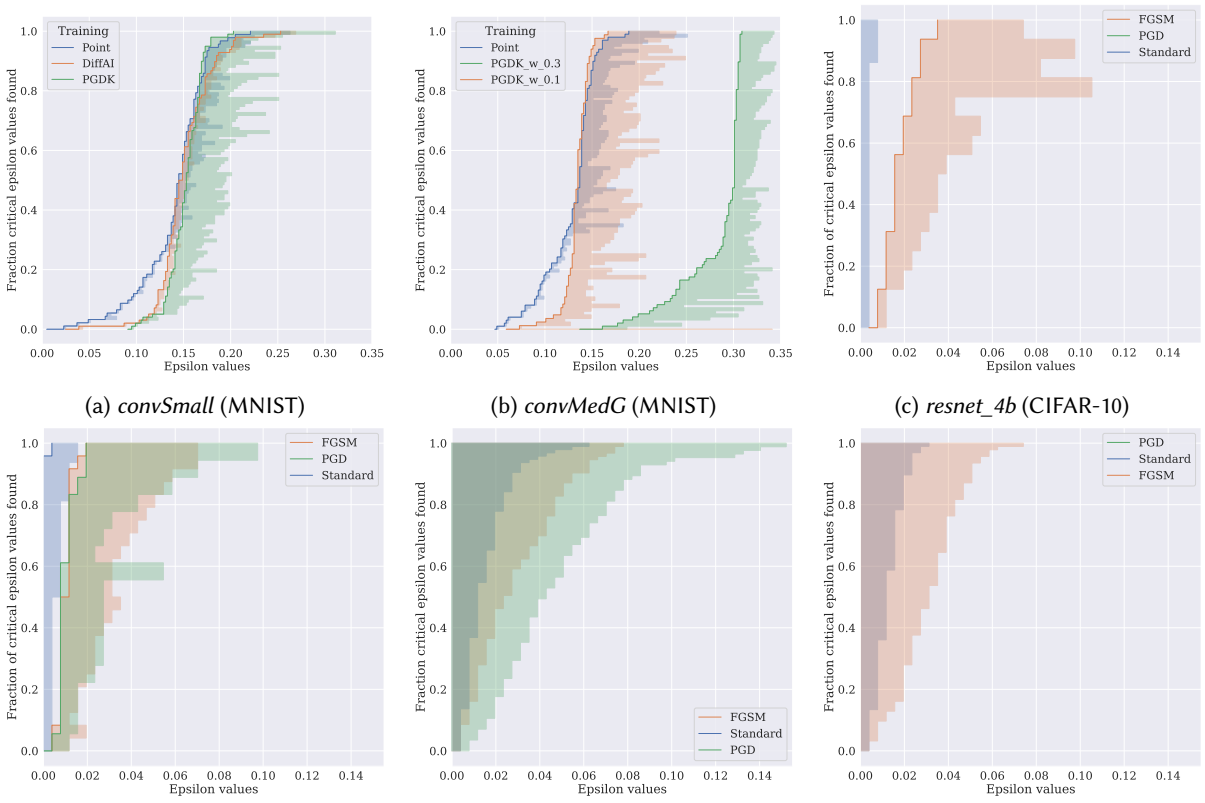
Network	Training	test accuracy	KS-test passed		% critical terminations	% unsolved instances	Average verification gap
			train	test	test	test	test
convSmall (MNIST)	point	0.983	False	False	76.344	0	0.012
	DiffAI	0.997	True	False	73.196	0	0.006
	PGDK	0.990	True	True	98	0	0.037
convMedG (MNIST)	point	0.986	False	False	78	0	0.016
	PGDK_0.1	0.991	False	False	98.881	0	0.053
	PGDK_0.3	0.998	False	False	100	0	0.036
conv_big (CIFAR-10)	Standard	0.650	False	True	39.063	21.875	0.007
	FGSM	0.588	True	True	44.444	53.704	0.024
	PGD	0.593	True	True	50.943	47.170	0.046
resnet_4b (CIFAR-10)	Standard	0.884	invalid	invalid	12.162	18.919	0.004
	FGSM	0.668	True	True	3.333	59.524	0.027
GTSRB_cnn_deep_game	Standard	0.927	invalid	invalid	85.416	0	0.016
	FGSM	0.857	invalid	invalid	90.425	0	0.029
	PGD	0.793	invalid	invalid	100	0	0.048
GTSRB_cnn_vnncomp23	Standard	0.947	invalid	invalid	85.714	0	0.013
	FGSM	0.880	invalid	invalid	95.745	3.614	0.031

investigation. In this part of our analysis, only the $\tilde{\varepsilon}^*$ and $\tilde{p}$ distributions of the *convSmall* architecture trained with standard training and DiffAI do not differ significantly from one another; see also Appendix K.

6.3 Robustness Distributions of Adversarially Trained Convolutional Neural Networks

We now report the results of our empirical investigation of the influence of adversarial training on the robustness distributions of a given neural network. Our goal was to demonstrate that $\tilde{\varepsilon}^*$ distributions can capture to what extent adversarial training affects the robustness of convolutional neural networks.

From Figure 11, we can see that for *convMedG*, the robustness improved significantly when trained with PGDK with weight 0.3, while for the other networks, we did not observe a significant improvement due to training. This increase in robustness comes at a marginal cost, as the training accuracy is only minimally worse than that of the conventionally trained network (0.995 vs 0.996), while the testing accuracy of the adversarially trained network turned out to be slightly better than that of the conventionally trained network (0.988 vs 0.986). The means of the distributions after adversarial training do significantly differ from those induced by conventional training, except for *convSmall* trained with DiffAI, see also Appendix E Table 8. We believe this lack of improvement can partially be explained due to the low quality of the robustness distributions; since the verification gap is quite large for most networks from any of the three datasets, it could be possible to detect a quantitative improvement from robustness distributions of a quality higher than that we were able to achieve here.



(a) *convSmall* (MNIST) (b) *convMedG* (MNIST) (c) *resnet_4b* (CIFAR-10)

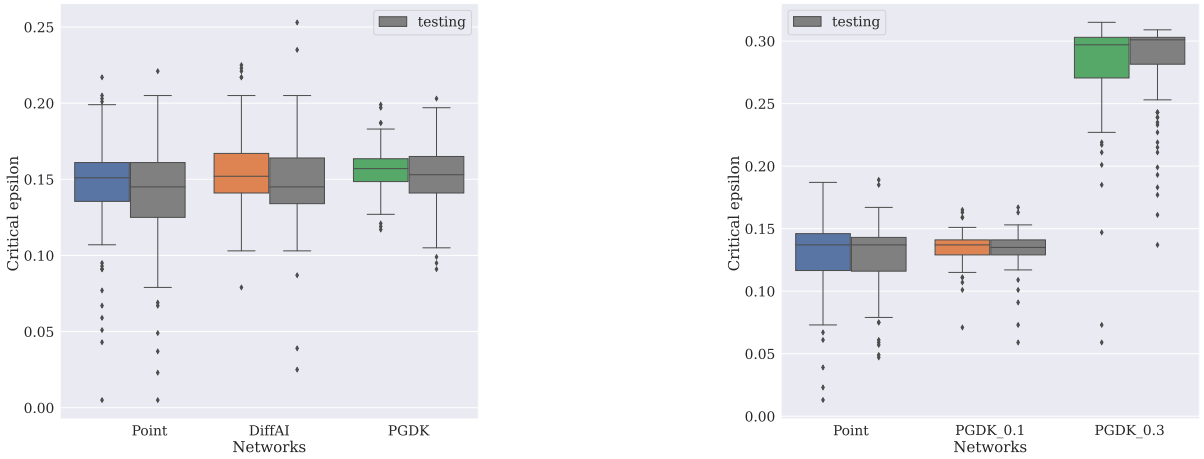
(d) *convBig* (CIFAR-10) (e) *GTSRB_cnn_deep_game* (f) *GTSRB_cnn_vnncomp23*

Fig. 10. Empirical CDF for the empirical robustness distributions of convolutional neural networks across the MNIST, CIFAR-10 and GTSRB datasets. The shaded area illustrates the difference between $\tilde{\varepsilon}^*$ and the upper bound $\tilde{p}$ of the respective image. (a) and (b) show the MNIST networks trained with different methods, (c) and (d) illustrate the CIFAR-10 networks trained with different methods. (e) and (f) show the GTSRB networks trained with different methods.

7 Discussion and Outlook

In this study, we have shown the limitations of assessing neural network robustness solely based on robust accuracy concerning a fixed perturbation radius, denoted by ε. In particular, the robustness of networks varies greatly under different perturbation radii. As an alternative, we propose robustness distributions, which provide a far more nuanced view of neural network robustness. Rather than a single robust accuracy value, our approach provides a considerably more informative distribution describing the robustness across different input data. The robustness distributions provide a holistic view of robustness across a wide range of different ε values, which can aid in better-informed model selection for more robust models. We determine the robustness distributions over a set of critical ε values. A critical ε value represents, for a given input, the maximum amount of perturbation for which we can prove robustness. While determining critical ε values comes at an additional computational cost, we show that, when utilising parallel k-binary search, they can be determined efficiently.

It is computationally challenging to calculate the critical epsilon distributions. We have developed measures to assess the quality of these distributions, notably the verification gap, the percentage of critical terminations and



(a) convSmall (b) convMedG

Fig. 11. Boxplots of the training and testing distribution of $\tilde{\varepsilon}^*$ for two MNIST networks, trained using conventional training as well as two adversarial training methods. Using the Kolmogorov-Smirnov test, we did not find significant differences between the distributions of training and testing data.

the percentage of unsolved instances. While for the MNIST networks (fully connected feed-forward networks, convolutional networks, both trained conventionally and adversarially), we often obtain small verification gaps, indicating that the verification engine was able to solve most queries, resulting in high-quality distributions. However, for the CIFAR-10 and GTSRB networks, more queries result in time-out and out-of-memory errors, resulting in lower-quality robustness distributions, from which we can not make general statements. This, in turn, indicates that further advances are needed to obtain robustness distributions for networks on challenging datasets.

Analysing these critical ε (or more generally: robustness) distributions, we have found evidence that, at least for the fully-connected MNIST ReLU networks we have analysed, they closely resemble log-normal distributions. Determining why these robustness distributions tend to follow a log-normal distribution is a question that is left for future work.

We also found evidence that the robustness distributions for inputs that the network was not trained on (testing data) do not differ significantly from those of the inputs used for training. This is notable since the critical epsilon value is directly related to the decision boundary. Where the training process has the opportunity to adapt the decision boundary based on the training data, this has not happened for the testing data. Generally, it is common that a relatively small, yet significant, number of observations already converge to a constant and reliable distribution. That leads us to believe that the use of critical ε distributions makes it possible to confidently make general statements of the robustness of a given network, without the need to determine the critical ε value for the entire dataset.

To further demonstrate the power of robustness distributions, we performed a correlation analysis on the critical ε values for different MNIST fully-connected networks. This analysis uncovered that there is generally a moderate to strong correlation in the perturbation size needed for an image to be able to deceive different networks. At the same time, we found that these networks may differ in the classes they mispredict for a given non-robust input.

We also investigated the effect of adversarial training on robustness distributions on multiple datasets and architectures. We found that the robustness distributions can capture the increased robustness produced by adversarial training for part of the datasets and architectures. By using robustness distributions, we were able to quantify the change in robustness in a statistically meaningful way.

At the same time, for part of the architectures we investigated (for example, all the CIFAR-10 networks), complete verification is too expensive, and we cannot find high-quality robustness distributions even with state-of-the-art verification methods. As we have shown the power of robustness distributions in giving insight into and comparing the robustness of networks, it is necessary to focus on methods that can find the critical ε of these more complex networks.

In this part of the investigation, we also found that adversarially trained neural networks do not seem to follow log-normal distributions. This warrants further research on what determines the shape of the robustness distributions.

In future work, we plan to explore ways to increase the efficiency of calculating critical ε distributions via k-binary search, e.g., via reusing information from other images. Additionally, investigating the effect of different norms, besides the l_∞ norm, on the shape of the robustness distribution would be interesting. To further understand the effect of training regimen and architecture on the shape of the robustness distributions, it is essential to do a more principled study of different training regimens for the same architecture. We also believe that it would be desirable to have a single robustness metric that is sound, generalisable and preferably computationally cheaper, based on the notion of the distribution of critical ε over input data. Finally, we are interested in analysing robustness distributions for a broader range of machine learning tasks, such as regression.

In this paper, we have introduced the concept of robustness distributions, which provides a nuanced view of the robustness of neural networks and thus enables new and powerful techniques for analysing the robustness of machine learning methods. While we have developed and studied this concept in the context of neural networks, it can, in principle, be applied more broadly and thus enable further progress in the broader area of robust machine learning.

Acknowledgements

We thank Matthias König, Bram Renting, Hadar Shavit, Nils Eberhardt and Konstantin Kaulen for their helpful feedback on an earlier version of this article. We are also grateful to Laurens Arp, Sietse Schröder, Hendrik Baacke, Konstantin Kaulen and Andreas Paraskeva for proofreading the manuscript before final submission. Finally, we thank the editor, Scott Sanner, and the reviewers for their time, effort and valuable input. This research was partially supported by TAILOR, a project funded by the EU Horizon 2020 research and innovation program under GA No. 952215, and by an Alexander-von-Humboldt Professorship in AI held by Holger Hoos.

References

[1] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. 2019. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM International Conference on Knowledge Discovery & Data Mining*. ACM, Anchorage, Alaska, USA, 2623–2631.

[2] S. Bak, C. Liu, and T. Johnson. 2021. The Second International Verification of Neural Networks Competition (VNN-COMP 2021): Summary and Results. (2021).

[3] O. Bastani, Y. Ioannou, L. Lampropoulos, D. Vytiniotis, A. Nori, and A. Criminisi. 2016. Measuring Neural Net Robustness with Constraints. In *Advances in Neural Information Processing Systems 29 (NeurIPS 2016)*. Neural Information Processing Systems Foundation, Inc. (NeurIPS), Barcelona, Spain, 2613–2621.

[4] A. W. Bosman, H. H. Hoos, and J. N. van Rijn. 2023. A Preliminary Study of Critical Robustness Distributions in Neural Network Verification. Paris, France, (2023).

[5] E. Botoeva, P. Kouvaros, J. Kronqvist, A. Lomuscio, and R. Misener. 2020. Efficient Verification of ReLU-based Neural Networks via Dependency Analysis. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI-20)*. AAAI Press, New York, New York, USA, 3291–3299.

[6] C. Brix, S. Bak, C. Liu, and T. T. Johnson. 2023. The Fourth International Verification of Neural Networks Competition (VNN-COMP 2023): Summary and Results. (2023).

[7] C. Brix, M. N. Müller, S. Bak, T. T. Johnson, and C. Liu. 2023. First three years of the international verification of neural networks competition (VNN-COMP). *International Journal on Software Tools for Technology Transfer*, 25, 3, 329–339.

[8] R. Bunel, I. Turkaslan, P. Torr, P. Kohli, and P. K. Mudigonda. 2018. A Unified View of Piecewise Linear Neural Network Verification. In *Advances in Neural Information Processing Systems 31 (NeurIPS 2018)*. Neural Information Processing Systems Foundation, Inc. (NeurIPS), Montreal, Canada, 1–10.

[9] C.-H. Cheng, G. Nührenberg, and H. Ruess. 2017. Maximum Resilience of Artificial Neural Networks. In *Automated Technology for Verification and Analysis: 15th International Symposium, ATVA 2017, 2017, Proceedings 15*. Springer, Pune, India, 251–268.

[10] B. Chmiel, L. Ben-Uri, M. Shkolnik, E. Hoffer, R. Banner, and D. Soudry. 2021. Neural Gradients are Near-Lognormal: Improved Quantized and Sparse Training. In *Proceedings of the 9th International Conference on Learning Representations (ICLR 2021)*. International Conference on Learning Representations, Vienna, Austria, 1–29.

[11] F. Cicalese, L. Gargano, and U. Vaccaro. 2004. On Searching Strategies, Parallel Questions, and Delayed Answers. *Discrete applied mathematics*, 144, 3, 247–262.

[12] F. Cicalese and U. Vaccaro. 2003. Binary Search with Delayed and Missing Answers. *Information Processing Letters*, 85, 5, 239–247.

[13] D. Cullina, A. N. Bhagoji, and P. Mittal. 2018. PAC-learning in the presence of adversaries. In *Advances in Neural Information Processing Systems 31 (NeurIPS 2018)*. Neural Information Processing Systems Foundation, Inc. (NeurIPS), Montreal, Canada, 1–12.

[14] D. A. Darling. 1957. The Kolmogorov-Smirnov, Cramér-von Mises Tests. *The Annals of Mathematical Statistics*, 28(4), 823–838.

[15] A. De Palma, R. Bunel, A. Desmaison, K. Dvijotham, P. Kohli, P. H. Torr, and M. P. Kumar. 2021. Improved Branch and Bound for Neural Network Verification via Lagrangian Decomposition. (2021).

[16] K. Dvijotham, R. Stanforth, S. Gowal, T. A. Mann, and P. Kohli. 2018. A Dual Approach to Scalable Verification of Deep Networks. In *Proceedings of the 38th Conference on Uncertainty in Artificial Intelligence (UAI 2018)*. Association for Uncertainty in Artificial Intelligence (AUAI), Eindhoven, The Netherlands, 550–559.

[17] R. Ehlers. 2017. Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks. In *Proceedings of the 15th International Symposium on Automated Technology for Verification and Analysis (ATVA 2017)*. Springer, Pune, India, 269–286.

[18] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev. 2018. AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation. In *Proceedings of the 39th IEEE Symposium on Security and Privacy (IEEE S&P 2018)*. IEEE, San Francisco, California, USA, 3–18.

[19] I. J. Goodfellow, J. Shlens, and C. Szegedy. 2014. Explaining and Harnessing Adversarial Examples. (2014).

[20] P. Henriksen and A. Lomuscio. 2020. Efficient Neural Network Verification via Adaptive Refinement and Adversarial Search. In *Proceedings of the 24th European Conference on Artificial Intelligence (ECAI 2020)*. IOS press, Santiago de Compostela, Spain, 2513–2520.

[21] S. Houben, J. Stallkamp, J. Salmen, M. Schlipsing, and C. Igel. 2013. Detection of traffic signs in real-world images: the German Traffic Sign Detection Benchmark. (2013).

[22] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. 2017. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *Proceedings of the 29th International Conference on Computer Aided Verification (CAV 2017)*. Springer, Heidelberg, Germany, 97–117.

[23] A. Kolmogoroff. 1941. Confidence Limits for an Unknown Distribution Function. *The annals of mathematical statistics*, 12, 4, 461–463.

[24] A. Kolmogorov. 1933. Sulla determinazione empirica di una legge didistribuzione. *Giorn Dell'inst Ital Degli Att*, 4, 89–91.

[25] M. König, A. W. Bosman, H. H. Hoos, and J. N. van Rijn. 2024. Critically Assessing the State of the Art in Neural Network Verification. *Journal of Machine Learning Research*, 25, 12, 1–53.

[26] A. Krizhevsky, V. Nair, and G. Hinton. 2009. Cifar-10 (canadian institute for advanced research). (2009). http://www.cs.toronto.edu/~kriz/cifar.html.

[27] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*. IEEE, unknown, 2278–2324.

[28] L. Li, T. Xie, and B. Li. 2023. SoK: Certified Robustness for Deep Neural Networks. In *2023 IEEE Symposium on Security and Privacy (SP 2023)*. IEEE Computer Society, San Francisco, California, USA, 94–115.

[29] C. Liu, T. Arnon, C. Lazarus, C. A. Strong, C. W. Barrett, and M. J. Kochenderfer. 2021. Algorithms for Verifying Deep Neural Networks. *Foundations and Trends® in Optimization*, 4, 3-4, 244–404.

[30] J. Liu, L. Chen, A. Miné, H. Yu, and J. Wang. 2023. Input Validation for Neural Networks via Local Robustness Verification. In *2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security Companion (QRS-C)*. Springer, Chiang Mai, Thailand, 237–246.

[31] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. 2017. Towards Deep Learning Models Resistant to Adversarial Attacks. (2017).

[32] M. Mirman, T. Gehr, and M. Vechev. 2018. Differentiable Abstract Interpretation for Provably Robust Neural Networks. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018* (Proceedings of Machine Learning Research). PMLR, Vienna, Austria, 3578–3586.

[33] J. Mohapatra, T.-W. Weng, P.-Y. Chen, S. Liu, and L. Daniel. 2020. Towards Verifying Robustness of Neural Networks against a Family of Semantic Perturbations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, Seattle, Washington, USA, 244–252.

[34] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. 2016. DeepFool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. IEEE, Las Vegas, Nevada, USA, 2574–2582.

[35] C. Müller, F. Serre, G. Singh, M. Püschel, and M. Vechev. 2021. Scaling Polyhedral Neural Network Verification on GPUs. *Proceedings of Machine Learning and Systems*, 3, 733–746.

[36] M. N. Müller, C. Brix, S. Bak, C. Liu, and T. T. Johnson. 2022. The Third International Verification of Neural Networks Competition (VNN-COMP 2022): Summary and Results. (2022).

[37] M. N. Müller, F. Eckert, M. Fischer, and M. Vechev. 2023. Certified Training: Small Boxes are All You Need. In *Proceedings of the 11th International Conference on Learning Representations (ICLR 2023)*. International Conference on Learning Representations, Kigali, Rwanda, 1–21.

[38] N. Papernot, P. McDaniel, and I. Goodfellow. 2016. Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples. (2016).

[39] L. Schmidt, S. Santurkar, D. Tsipras, K. Talwar, and A. Madry. 2018. Adversarially Robust Generalization Requires More Data. In *Advances in Neural Information Processing Systems 31 (NeurIPS 2018)*. Neural Information Processing Systems Foundation, Inc. (NeurIPS), Montreal, Canada, 1–13.

[40] Z. Shi, Y. Wang, H. Zhang, J. Yi, and C.-J. Hsieh. 2021. Fast Certified Robust Training with Short Warmup. *Advances in Neural Information Processing Systems 34 (NeurIPS 2021)*, 34, 18335–18349.

[41] G. Singh, T. Gehr, M. Püschel, and M. Vechev. 2019. Boosting Robustness Certification of Neural Networks. In *Proceedings of the 7th International Conference on Learning Representations (ICLR 2019)*. International Conference on Learning Representations, New Orleans, Louisiana, USA, 1–12.

[42] N. V. Smirnov. 1944. Approximate Laws of Distribution of Random Variables from Empirical Data. *Uspekhi Matematicheskikh Nauk*, 10, 179–206.

[43] C. Spek. 2023. *Auto-Verify: A framework for portfolio-based neural network verification*. Master's thesis. LIACS, Leiden University.

[44] C. A. Strong, H. Wu, A. Zeljić, K. D. Julian, G. Katz, C. Barrett, and M. J. Kochenderfer. 2023. Global Optimization of Objective Functions represented by ReLU Networks. *Machine Learning*, 112, 10, 3685–3712.

[45] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus. 2014. Intriguing Properties of Neural Networks. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR 2014)*. International Conference on Learning Representations, Banff National Park, Canada, 1–10.

[46] V. Tjeng, K. Xiao, and R. Tedrake. 2019. Evaluating Robustness of Neural Networks with Mixed Integer Programming. In *Proceedings of the 7th International Conference on Learning Representations (ICLR 2019)*. International Conference on Learning Representations, New Orleans, Louisiana, USA, 1–21.

[47] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana. 2018. Efficient Formal Safety Analysis of Neural Networks. In *Advances in Neural Information Processing Systems 31 (NeurIPS 2018)*. Neural Information Processing Systems Foundation, Inc. (NeurIPS), Montreal, Canada, 6369–6379.

[48] S. Wang, H. Zhang, K. Xu, X. Lin, S. Jana, C.-J. Hsieh, and J. Z. Kolter. 2021. Beta-CROWN: Efficient Bound Propagation with Per-neuron Split Constraints for Neural Network Robustness Verification. In *Advances in Neural Information Processing Systems 34 (NeurIPS 2021)*. Neural Information Processing Systems Foundation, Inc. (NeurIPS), virtual, 29909–29921.

[49] T.-W. Weng, H. Zhang, P.-Y. Chen, J. Yi, D. Su, Y. Gao, C.-J. Hsieh, and L. Daniel. 2018. Evaluating the Robustness of Neural Networks: An Extreme Value Theory Approach. In *Proceedings of the 7nd International Conference on Learning Representations (ICLR 2018)*. International Conference on Learning Representations, Vancouver, Canada, 1–18.

[50] E. Wong, L. Rice, and J. Z. Kolter. 2020. Fast is better than free: Revisiting adversarial training. In *Proceedings of the 8th International Conference on Learning Representations (ICLR 2020)*. International Conference on Learning Representations, Addis Ababa, Ethiopia, 1–17.

[51] M. Wu, M. Wicker, W. Ruan, X. Huang, and M. Kwiatkowska. 2020. A game-based approximate verification of deep neural networks with provable guarantees. *Theoretical Computer Science*, 807, 298–329.

[52] W. Xiang, H.-D. Tran, and T. T. Johnson. 2018. Output Reachable Set Estimation and Verification for Multilayer Neural Networks. In *IEEE Transactions on Neural Networks and Learning Systems*. Vol. 29. IEEE, Rhodes, Greece, 5777–5783.

[53] C. Xie, Y. Wu, L. v. d. Maaten, A. L. Yuille, and K. He. 2019. Feature Denoising for Improving Adversarial Robustness. In *Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition*. IEEE, Long beach, California, USA, 501–509.

[54] Y.-Y. Yang, C. Rashtchian, H. Zhang, R. R. Salakhutdinov, and K. Chaudhuri. 2020. A Closer Look at Accuracy vs. Robustness. In *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*. Neural Information Processing Systems Foundation, Inc. (NeurIPS), Vancouver, Canada, 8588–8601.

[55] H. Zhang, Y. Yu, J. Jiao, E. P. Xing, L. E. Ghaoui, and M. I. Jordan. 2019. Theoretically Principled Trade-off between Robustness and Accuracy. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019* (Proceedings of Machine Learning Research). K. Chaudhuri and R. Salakhutdinov, (Eds.) Vol. 97. PMLR, Long beach, California, USA, 7472–7482.

[56] H. Zhang, H. Chen, C. Xiao, S. Gowal, R. Stanforth, B. Li, D. Boning, and C. J. Hsieh. 2020. Towards Stable and Efficient Training of Verifiably Robust Neural Networks. In *Proceedings of the 8th International Conference on Learning Representations (ICLR 2020)*. International Conference on Learning Representations, Addis Ababa, Ethiopia, 1–25.

[57] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel. 2018. Efficient Neural Network Robustness Certification with General Activation Functions. In *Advances in Neural Information Processing Systems 31 (NeurIPS 2018)*. Neural Information Processing Systems Foundation, Inc. (NeurIPS), Montreal, Canada, 4939–4948.

A Details of k-binary Search Analysis

Table 4. Statistics for training data across different k values. Total queries include all ε values over all images and twelve networks. Time-out errors are queries exceeding 3,600 seconds; out-of-memory errors ran out of memory. *Cancelled queries* became redundant due to another query solving the same instance.

k	Total queries	Time-out errors		Out-of-memory errors		Cancelled queries	
	Absolute	Absolute	Ratio	Absolute	Ratio	Absolute	Ratio
1	14 547	119	0.008	5 643	0.388	206	0.014
2	13 516	191	0.141	131	0.010	1 961	0.145
4	21 025	1 228	0.058	140	0.007	6 499	0.309
8	32 646	1 677	0.051	86	0.003	15 949	0.438
16	25 761	217	0.008	205	0.008	12 398	0.481
total	107 495	3 432		6 205		37 013	

B Statistics Robustness Distributions MNIST Fully-Connected Conventionally Trained

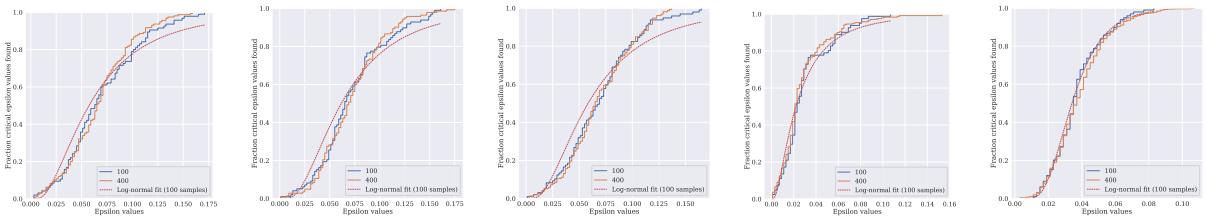
Table 5. Statistics of the $\tilde{\varepsilon}^*$ distributions and the testing and training accuracy over all testing and training instances for the 12 conventionally trained, fully connected ReLU networks. We report the minimum and mean of the largest safe radius that we found, $\tilde{\varepsilon}^*$, both for training and testing data. Additionally, we report the number of correctly classified images per network and the training and testing accuracy over all MNIST training and testing data respectively.

network	lowest $\tilde{\varepsilon}^*$		mean $\tilde{\varepsilon}^*$		instances		accuracy	
	training	testing	training	testing	training	testing	training	testing
net	0.013	0.005	0.042	0.038	99	99	0.996	0.996
net_256x2	0.011	0.007	0.037	0.034	99	100	0.995	0.995
relu_6_100	0.007	0.009	0.065	0.063	96	98	0.969	0.968
relu_3_50	0.007	0.005	0.058	0.052	97	98	0.964	0.962
relu_4_1024	0.005	0.003	0.073	0.068	93	97	0.945	0.941
relu_9_100	0.005	0.009	0.064	0.057	96	95	0.957	0.956
relu_3_100	0.005	0.003	0.057	0.057	98	97	0.971	0.968
relu_9_200	0.003	0.003	0.072	0.067	95	97	0.957	0.956
net_256x4	0.001	0.011	0.072	0.067	98	99	0.991	0.990
net_256x6	0.001	0.001	0.071	0.064	97	98	0.997	0.978
relu_6_200	0.001	0.005	0.067	0.067	98	98	0.972	0.969
nn	0.001	0.001	0.030	0.027	80	76	0.745	0.757

C Size of Sample for Robustness Distributions

Table 6. Comparison of the robustness distribution of the $\tilde{\varepsilon}^*$ of 100 random MNIST training images used in this work and the robustness distribution of the $\tilde{\varepsilon}^*$ of 400 additionally random MNIST training images (mean and standard deviation). This second set of 400 images strictly does not contain any of the images of the original 100 images. The table shows the outcomes of a two-sample Kolmogorov-Smirnov (KS) test at a significance level of 0.05. The test shows whether the respective $\tilde{\varepsilon}^*$ distributions are deemed from the same distribution, in all cases we cannot reject the H_0 of log-normality.

	$\tilde{\varepsilon}^*$ mean ± variance		p-value KS two-sample test
Network	N=100	N=400	
relu_9_200	0.072 ± 0.038	0.067 ± 0.033	0.716
net_256x4	0.072 ± 0.035	0.072 ± 0.032	0.640
net_256x6	0.071 ± 0.035	0.069 ± 0.031	0.995
nn	0.030 ± 0.022	0.028 ± 0.024	0.81
net_256x2	0.037 ± 0.015	0.039 ± 0.017	0.296



(a) relu_9_200 (b) net_256x4 (c) net_256x6 (d) nn (e) net_256x6

Fig. 12. Emperical CDF for the empirical robustness distributions for the 100 MNIST images of conventionally trained neural networks in comparison to another 400 random images for this architecture on training data. Note that we create the CDF with images that are correctly classified of these 100 and 400 images, this might mean that the distribution is created with less than 100 and 400 images respectively, depending on the accuracy of the network. We have added a red dotted line representing the log-normal distribution fitted on the 100 samples for each network. All robustness distributions, except for *net_256x4*, with 100 images tend to follow a log-normal distribution. While the distributions are highly similar (see Figure 12), the Kolmogorov-Smirnov test becomes conservative at large sample sizes and fails to detect that the distributions of 400 images follow a log-normal distribution, as expected. The Kolmogorov-Smirnov test becomes overly conservative for large samples, especially when considering discrete data.

D Statistics of Running Time Robustness Distributions MNIST Fully-Connected Conventionally Trained

Table 7. Information of the running time in CPU time in hours and wallclock time in hours of the experiments inferring the $\tilde{\epsilon}^*$ distributions for fully-connected conventionally trained MNIST networks. *Total time* $k = 2$ contains the running time for all experiments for $k = 2$ and *total time all* contains the total time for all different values for k. The total and average times consider both testing and training instances.

network	Total time k=2		Average per image		Average verification time [h]	Total time all	
	wallclock [h]	cpu [h]	wallclock [h]	cpu [h]		wallclock [h]	cpu [h]
net	159	272	0.8	1	0.12	440	1183
net_256x2	49	86	0.25	0.43	0.04	154	49
relu_6_100	430	829	2	4	0.38	1263	2988
relu_3_50	219	418	1	2	0.18	603	1566
relu_4_1024	490	932	3	5	0.41	1507	3742
relu_9_100	448	851	2	4	0.37	1281	3074
relu_3_100	409	782	2	4	0.34	1078	2819
relu_9_200	491	953	3	5	0.42	1568	3704
net_256x4	550	1022	3	5	0.43	1407	3691
nn	95	171	0.59	1	0.09	286	759
net_256x6	493	948	3	5	0.42	1462	3943
relu_6_200	450	868	2	4	0.39	1409	3392
total	4283	8132	2	4	0.29	12459	31260

E Statistics of Robustness Distributions for Fully-Connected Architectures including Adversarial Training

Table 8. Statistics of the $\tilde{\epsilon}^*$ distributions and the testing and training accuracy overall testing and training instances for the MNIST architectures. The total error contains the number of instances that could not be verified due to the verification process leading to only time-outs and errors. The bold values indicate that the mean of the adversarially trained distribution is significantly different from the conventionally trained distribution according to a t-test with a significance level of 0.05.

Network	Training method	Lowest $\tilde{\epsilon}^*$		Mean $\tilde{\epsilon}^*$		Instances		Total errors		Accuracy	
		train	test	train	test	train	test	train	test	train	test
mnist_nn	Standard	0.001	0.001	0.030	0.031	81	81	0	0	0.745	0.757
	FGSM	0.001	0.007	**0.188**	**0.174**	92	89	0	0	0.912	0.910
	PGD	0.017	0.007	**0.270**	**0.268**	84	77	0	0	0.799	0.802
mnist_relu_4_1024	Standard	0.005	0.005	0.073	0.073	93	93	0	0	0.945	0.941
	FGSM	0.077	0.017	**0.244**	**0.223**	97	100	0	0	0.980	0.978
	PGD	0.017	0.007	**0.290**	**0.290**	85	77	0	0	0.780	0.782
convSmall	Point	0.005	0.005	0.144	0.138	90	93	9	6	0.987	0.983
	DiffAI	0.079	0.025	0.155	0.149	98	99	0	0	0.981	0.977
	PGDK	0.117	0.019	**0.156**	**0.159**	92	100	7	0	0.995	0.990
convMedG	Point	0.013	0.047	0.127	0.127	100	100	0	0	0.996	0.986
	PGDK w 0.1	0.071	0.059	**0.135**	**0.133**	97	84	3	16	0.999	0.991
	PGDK w 0.3	0.059	0.137	**0.279**	**0.288**	96	98	3	1	0.995	0.988

Table 9. Statistics of the $\tilde{\epsilon}^*$ distributions and the testing and training accuracy over all testing and training instances for the included CIFAR architectures. The total error contains the number of instances that could not be verified due to the verification process leading to only time-outs and errors. The bold values indicate that the mean of the adversarially trained distribution is significantly different from the conventionally trained distribution according to a t-test with a significance level of 0.05.

Network	Training method	Lowest $\tilde{\epsilon}^*$		Mean $\tilde{\epsilon}^*$		Instances		Total errors		Accuracy	
		train	test	train	test	train	test	train	test	train	test
cifar_7_1024	Standard	0.000	0.000	0.002	0.002	38	40	17	24	0.641	0.540
	FGSM	0.000	0.0078	**0.025**	**0.036**	18	17	18	14	0.362	0.367
	PGD	0.000	0.000	**0.007**	**0.006**	17	18	29	47	0.663	0.510
conv_big	Standard	0.000	0.000	0.000	0.000	63	49	14	35	0.975	0.650
	FGSM	0.000	0.0039	**0.011**	**0.01**	20	25	29	48	0.668	0.588
	PGD	0.000	0.000	**0.010**	**0.010**	18	19	25	41	0.654	0.593
resnet_4b	Standard	0.000	0.000	0.000	0.000	81	59	14	19	0.987	0.884
	FGSM	0.000	0.0039	**0.013**	**0.017**	13	17	25	23	0.441	0.668

Table 10. Statistics of the $\tilde{\epsilon}^*$ distributions and the testing and training accuracy over all testing and training instances for the included GTSRB architectures. The total error contains the number of instances that could not be verified due to the verification process leading to only time-outs and errors. The bold values indicate that the mean of the adversarially trained distribution is significantly different from the conventionally trained distribution according to a t-test with a significance level of 0.05. For the instances where no $\tilde{\epsilon}^*$ could be found, we included an x.

Network	Training method	Lowest $\tilde{\epsilon}^*$		Mean $\tilde{\epsilon}^*$		Instances		Total errors		Accuracy	
		train	test	train	test	train	test	train	test	train	test
gtsrb_6_256	Standard	0.000	0.000	0.009	0.008	98	84	0	0	0.989	0.800
	FGSM	0.000	0.000	**0.023**	**0.023**	76	71	0	0	0.771	0.604
	PGD	0.004	0.000	**0.027**	**0.027**	70	69	0	0	0.767	0.613
gtsrb_7_1024	Standard	0.000	0.000	0.007	0.007	100	89	0	0	0.982	0.794
	FGSM	0.000	0.000	**0.015**	**0.014**	72	66	0	0	0.776	0.586
	PGD	0.000	0.000	**0.015**	**0.015**	89	78	0	0	0.875	0.673
gtsrb_cnn_deep_game	Standard	x	x	x	x	100	96	0	0	1.00	0.927
	FGSM	x	x	x	x	100	94	0	0	0.979	0.857
	PGD	x	x	x	x	96	86	0	0	0.952	0.793
gtsrb_cnn_vnncomp23	Standard	x	x	x	x	99	95	0	0	1.00	0.957
	FGSM	x	x	x	x	100	91	0	0	0.965	0.885
	PGD	x	x	x	x	91	79	0	0	0.901	0.724

F Statistics of Running Time Robustness Distributions with Different Training Methods

Table 11. Information of the running time in CPU time for the FFNN architectures and GPU time for CNN architectures in hours of the $\tilde{\epsilon}^*$ experiments. The total and average are all considering testing and training experiments. The total running times include time-outs and errors.

Network	Train method	Total running time [h]	Time-outs	Errors	Avg.Running time [h] per image	Avg.Running time [h] per query
mnist_nn	Standard	170.983	3	8	1.055	0.089
	FGSM	972.197	209	0	5.371	0.383
	PGD	758.036	178	2	4.708	0.340
mnist_nn_ relu_1024	Standard	931.710	40	54	5.009	0.409
	FGSM	1639.122	548	1	8.320	0.560
	PGD	924.545	227	1	5.707	0.415
Total FFNN		5436.432	1205	66		
convSmall	Point	537.199	484	627	2.699	0.224
	DiffAI	160.453	156	271	0.814	0.090
	PGDK	1956.685	1884	1791	10.138	0.421
convMedG	Point	722.142	656	867	3.611	0.259
	PGDK w 0.1	2062.852	2004	2954	10.915	0.357
	PGDK w 0.3	1618.928	1537	2136	8.135	0.346
Total CNN		6028.259	6721	8646		

Table 12. Information of the running time in GPU time in hours of the $\tilde{\epsilon}^*$ experiments for the CIFAR architectures. The total and average are all considering testing and training experiments. The total running times include time-outs and errors.

Network	Train method	Total running time [h]	Time-outs	Errors	Avg. Running time [h] per image	Avg. Running time [h] per query
cifar_7_1024	Standard	100.824	93	4223	0.847	0.021
	FGSM	109.764	101	3296	1.638	0.030
	PGD	231.360	219	7829	2.084	0.028
conv_big	Standard	77.299	61	5189	0.477	0.013
	FGSM	160.322	122	8084	1.314	0.019
	PGD	148.233	105	8602	1.235	0.017
resnet_4b	Standard	21.811	5	3728	0.126	0.005
	FGSM	1173.877	165	4944	2.201	0.033
Total		2022.943	871	103369		

Table 13. Information of the running time in GPU time in hours of the $\tilde{\varepsilon}^*$ experiments for the GTSRB architectures. The total and average are all considering testing and training experiments. The total running times include time-outs and errors.

Network	Train method	Total running time [h]	Time-outs	Errors	Avg. Running time [h] per image	Avg. Running time [h] per query
gstrb_6_256	Standard	79.161	70	0	0.435	0.062
	FGSM	643.098	620	0	4.375	0.439
	PGD	625.913	598	0	4.503	0.451
gtsrb_7_1024	Standard	300.614	291	0	1.591	0.204
	FGSM	907.038	900	0	6.573	0.554
	PGD	1179.660	1169	0	7.064	0.578
gtsrb_cnn _deep_game	Standard	8.656	0	564	0.044	0.005
	FGSM	13.497	0	1258	0.070	0.006
	PGD	19.240	0	2108	0.106	0.007
gtsrb_cnn _vnncomp23	Standard	4.953	0	635	0.025	0.003
	FGSM	8.396	0	1566	0.044	0.004
	PGD	4.970	0	721	0.029	0.003
Total		3795.196	3648	6852		

G Performance in terms of Accuracy for Adversarial Training and Adversarial Attacks

Table 14. Table with the accuracy scores for the MNIST networks. The training accuracy, standard accuracy and robust accuracies for the PGD adversary are listed.

network name	training	train acc	standard	PGD 0.2	PGD 0.3
convMedGRELU	PGDK w 0.1	99.9	99.2	40.6	0.5
	PGDK w 0.3	99.5	98.8	95.9	93.0
	Standard	99.6	98.6	10.8	0.0
convSmallRELU	DiffAI	98.1	97.7	14.8	0.3
	PGDK	99.5	98.9	49.1	2.5
	Standard	98.7	98.2	21.3	0.5
mnist_nn	FGSM 02	91.2	91.0	44.4	8.1
	PGD	79.9	80.2	62.3	51.9
	Standard	74.5	75.4	0.0	0.0
*mnist_relu*_4_1024	FGSM 02	98.0	97.8	64.7	15.4
	PGD	78.0	78.2	68.0	63.2
	Standard	94.5	93.8	0.1	0.0

Table 15. Table with the accuracy scores for the CIFAR-10 networks. The training accuracy, standard accuracy and robust accuracies for the PGD adversary are listed.

network name	training	train acc	standard	PGD 4/255	PGD 8/255
cifar_7_1024	Standard	64.1	54.0	11.4	1.6
	FGSM	36.3	36.7	30.8	25.6
	PGD	66.4	51.0	37.6	26.4
conv_big	Standard	97.6	65.0	3.0	0.4
	FGSM	66.8	58.8	45.6	33.1
	PGD	65.5	59.3	47.2	35.7
resnet_18	Standard	100.0	88.4	0.4	0.0
	FGSM	70.3	66.8	53.6	41.1
	PGD	80.5	71.1	56.6	42.2
resnet_4b	Standard	98.7	74.5	0.0	0.0
	FGSM	44.2	43.9	35.7	29.3
	PGD	53.4	52.0	41.5	32.4

Table 16. Table with the accuracy scores for the GTSRB networks. The training accuracy, standard accuracy and robust accuracies for the PGD adversary are listed.

network name	training	train acc	standard	PGD 4/255	PGD 8/255
gtsrb_6_256	Standard	98.9	80.0	19.7	8.4
	FGSM	77.1	60.4	45.1	32.7
	PGD	76.7	61.3	47.9	35.9
gtsrb_7_1024	Standard	98.2	79.4	28.2	15.5
	FGSM	77.6	58.6	42.7	31.3
	PGD	87.5	67.3	52.3	38.5
gtsrb_cnn_deep_game	Standard	100.0	92.7	26.6	12.9
	FGSM	98.0	85.7	50.4	32.9
	PGD	95.2	79.3	63.7	45.8
gtsrb_cnn_vnncomp23	Standard	100.0	95.7	21.7	5.3
	FGSM	96.5	88.5	64.1	42.5
	PGD	90.1	72.4	55.1	39.8

H Model Information

Table 17. Details about the MNIST model architectures

network	neurons	parameters	linear layers	conv layers
net	1K	669K	3	0
net_256x2	0.5K	269K	3	0
relu_6_100	0.5K	119K	6	0
relu_3_50	0.1K	42K	3	0
relu_4_1024	3K	2913K	4	0
relu_9_100	0.8K	150K	9	0
relu_3_100	0.2K	89K	3	0
relu_9_200	1.6K	440K	9	0
net_256x4	1K	400K	5	0
net_256x6	1.5K	532K	7	0
relu_6_200	1K	319K	6	0
nn	512	269K	3	0
convSmall	5K	89K	2	3
convMedG	8K	1587K	2	3

Table 18. Details about the CIFAR-10 model architectures.

network	neurons	parameters	linear layers	conv layers
cifar_7_1024	6K	8405K	7	0
resnet_4b	14K	123K	2	9
conv_big	62K	2466K	3	4
resnet_18	558K	11173K	1	17

Table 19. Details about the GTSRB model architectures.

network	neurons	parameters	linear layers	conv layers
gtsrb_6_256	1K	1060K	6	0
gtsrb_7_1024	6K	8438K	7	0
gtsrb_cnn_deep_game	221K	8438K	3	4
gtsrb_cnn_vnncomp23	260K	1775K	3	3

I Training parameters

Table 20. Training parameters for the MNIST models.

network	training	lr	step size	gamma	batch size	epochs
mnist_nn	FGSM	0.008940	5	0.893000	256	20
	PGD	0.000590	9	0.730000	256	100
mnist_relu_4_1024	FGSM	0.002020	2	0.990000	256	20
	PGD	0.001360	9	0.620000	256	100

Table 21. Training parameters for the CIFAR-10 models.

network	training	lr	step size	gamma	batch size	epochs
cifar_7_1024	Standard	0.000090	8	0.280000	128	30
	FGSM	0.000910	8	0.300000	128	15
	PGD	0.000200	2	0.980000	128	50
conv_big	Standard	0.001600	9	0.570000	128	30
	FGSM	0.000390	5	0.720000	128	15
	PGD	0.000580	9	0.017000	128	50
resnet_4b	Standard	0.002000	8	0.210000	128	30
	FGSM	0.010520	3	0.510000	128	15
	PGD	0.001490	8	0.041000	128	50

Table 22. Training parameters for the GTSRB models.

network	training	lr	step size	gamma	batch size	epochs
gtsrb_6_256	Standard	0.001630	4	0.630000	128	50
	FGSM	0.001430	3	0.877000	128	20
	PGD	0.001510	8	0.270000	128	50
gtsrb_7_1024	Standard	0.000960	10	0.980000	128	50
	FGSM	0.000660	10	0.467000	128	20
	PGD	0.000690	10	0.540000	128	50
gtsrb_cnn_deep_game	Standard	0.001200	9	0.030000	128	50
	FGSM	0.000747	9	0.864000	128	20
	PGD	0.000680	10	0.082800	128	50
gtsrb_cnn_vnncomp23	Standard	0.000921	9	0.047629	128	50
	FGSM	0.000777	7	0.666632	128	20
	PGD	0.001032	9	0.101743	128	50

J Verification Gaps

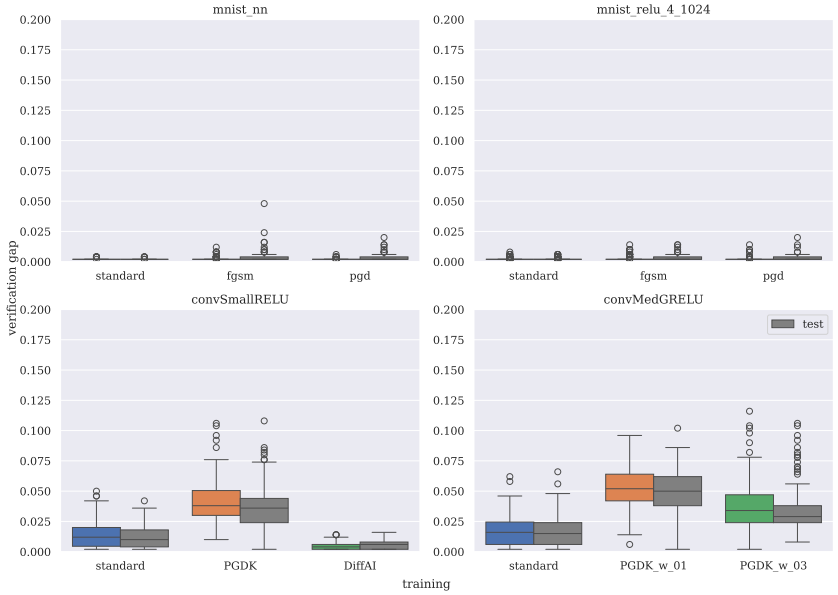


Fig. 13. Boxplots of the verification gaps of the MNIST networks.

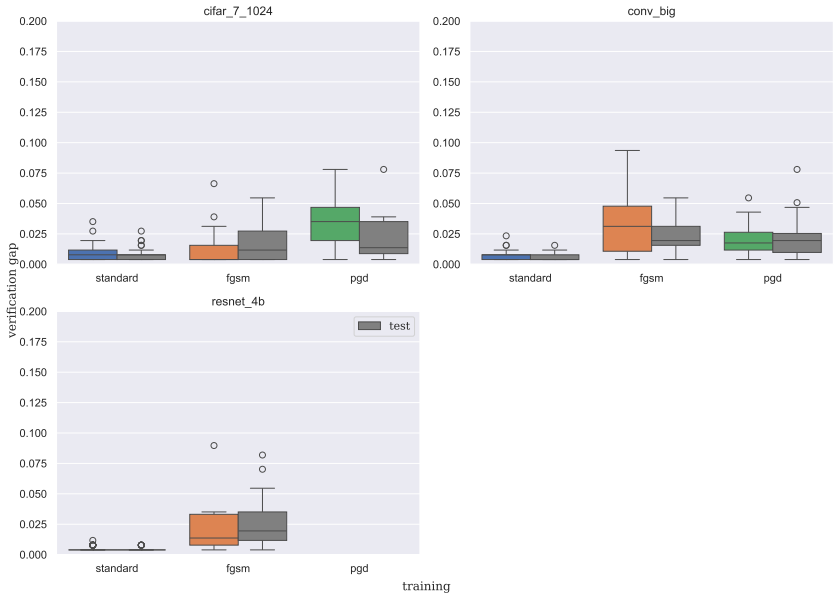


Fig. 14. Boxplots of the verification gaps of the CIFAR-10 networks.

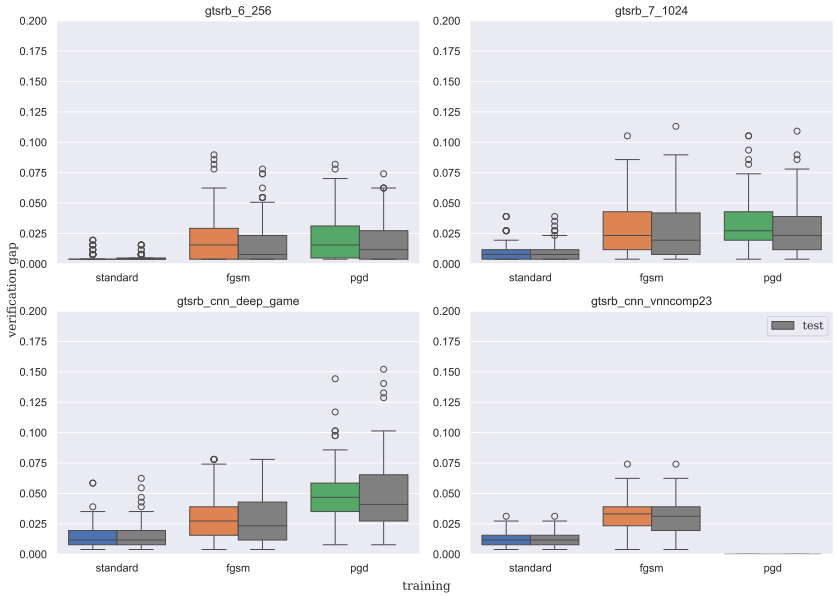


Fig. 15. Boxplots of the verification gaps of the GTSRB networks.

K Uncertainty in Determining the Exact ε^*

Table 23. This table shows, for each architecture and training method, for both the training and testing data distributions, a comparison of the distribution of the $\tilde{\varepsilon}^*$ and the distribution of the smallest ε-value, denoted by $\tilde{p}$, at which an adversarial example was found. Note that $\tilde{\varepsilon}^*$ is a strict under-estimation of ε^*, due to discretisation and verification gaps. When these distributions are similar, we have found the ε^* distribution with high precision. The table shows the p-value of the Kolmogorov-Smirnov test and the t-test with a significance value of 0.05. The tests that prove that the distributions are significantly different are bold, and the tests that could not be performed due to not reaching the criterion are replaced with an x.

Network	Training method	t-test		KS test	
		train	test	train	test
mnist_nn	Standard	x	x	0.588	0.588
	FGSM	0.799	0.703	1.0	0.999
	PGD	0.893	0.820	0.999	0.997
mnist_relu_4_1024	Standard	0.723	0.727	0.957	0.991
	FGSM	0.672	0.714	0.993	1.000
	PGD	0.839	0.801	0.985	0.975
convSmall	Point	0.015	0.046	**0.001**	0.011
	DiffAI	0.173	0.204	0.356	0.276
	PGDK	**0.000**	**0.000**	**0.000**	**0.000**
convMedG	Point	**0.001**	**0.001**	**0.000**	**0.000**
	PGDK w 0.1	**0.000**	**0.000**	**0.000**	**0.000**
	PGDK w 0.3	**0.000**	**0.000**	**0.000**	**0.000**
cifar_7_1024	Standard	**0.000**	**0.000**	**0.000**	**0.000**
	FGSM	0.149	0.094	0.191	0.751
	PGD	**0.000**	**0.000**	**0.000**	**0.000**
conv_big	Standard	**0.000**	**0.000**	0.067	0.585
	FGSM	**0.000**	**0.000**	**0.000**	**0.000**
	PGD	**0.000**	**0.000**	**0.000**	**0.000**
resnet_4b	Standard	**0.000**	**0.000**	x	x
	FGSM	**0.007**	**0.001**	**0.024**	**0.005**
gtsrb_6_256	Standard	**0.000**	**0.000**	0.649	**0.019**
	FGSM	**0.000**	**0.000**	**0.000**	0.072
	PGD	**0.000**	**0.000**	**0.000**	**0.007**
gtsrb_7_1024	Standard	**0.000**	**0.000**	**0.001**	**0.002**
	FGSM	**0.000**	**0.000**	**0.000**	**0.000**
	PGD	**0.000**	**0.000**	**0.000**	**0.000**
gtsrb_cnn_deep_game	Standard	**0.000**	**0.000**	x	x
	FGSM	**0.000**	**0.000**	x	x
	PGD	**0.000**	**0.000**	x	x
gtsrb_cnn_vnncomp23	Standard	**0.000**	**0.000**	x	x
	FGSM	**0.000**	**0.000**	x	x
	PGD	**0.000**	**0.000**	x	x

Received 10 April 2024; accepted 20 May 2025