Black-Box Multi-Robustness Testing for Neural Networks

Mara Downing
University of California Santa Barbara, CA, USA
maradowning@cs.ucsb.edu

Tevfik Bultan
University of California Santa Barbara, CA, USA
bultan@cs.ucsb.edu

***Abstract*—Neural networks are increasingly prevalent in day-to-day life, including in safety-critical applications such as self-driving cars and medical diagnoses. This prevalence has spurred extensive research into testing the robustness of neural networks against adversarial attacks, most commonly by determining if misclassified inputs can exist within a region around a correctly classified input. While most prior work focuses on robustness analysis around a single input at a time, in this paper we look at simultaneous analysis of multiple robustness regions. Our approach finds robustness violating inputs away from expected decision boundaries, identifies varied types of misclassifications by increasing confusion matrix coverage, and effectively discovers robustness violating inputs that do not violate input feasibility constraints. We demonstrate the capabilities of our approach on multiple networks trained from several datasets, including ImageNet and a street sign identification dataset.**

***Index Terms*—Neural Networks, Safety-Critical Systems, Robustness, Automated Testing**

I. Introduction

With the growing prevalence of neural networks, especially in safety critical domains such as self-driving cars or medical diagnoses, evaluating their correctness has become crucial. While accuracy measures on a previously unseen dataset give one metric for correctness, neural networks have been shown to be vulnerable to adversarial attacks in which a correctly classified input is perturbed in some small manner to make the network produce a misclassification that a human analyzing the same information would not [1]. Thus, testing the *robustness* of neural networks against these types of attacks is an important part of assuring their safety and dependability.

In this paper, we present a novel approach for multi-robustness analysis in which we leverage multiple inputs from the existing user-labeled training and test sets to automatically determine areas for which input classifications are uncertain (robustness conflict regions close to decision boundaries, as shown in Fig. 1) and do not report inputs within those regions as misclassifications, while automatically mutating inputs and providing a scalable multi-robustness testing approach.

We analyze robustness around multiple inputs simultaneously, which differentiates us from most existing robustness analysis approaches (Section V) which analyze a region around a singular input and thus must be run multiple times to obtain generalizable results about the whole network under analysis.

This research is supported by the NSF under Awards #2124039 and #2008660.

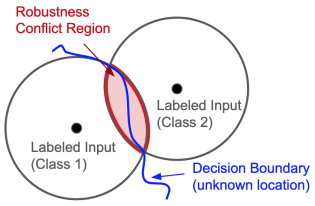

Fig. 1. Diagram of an area (robustness conflict region) with uncertain classification given known human-classified inputs and a perturbation radius defining a hypersphere around each input

Additionally, we leverage existing labeled training and test set inputs to avoid areas of uncertainty near expected decision boundaries (Fig. 1). Other work [2]–[5] analyzes multiple inputs simultaneously but does not leverage them to identify expected changes in classification, which allows the potential crossing of expected decision boundaries, the same as robustness around a singular input.

We incorporate feasibility constraints on the input values to the network (e.g., maximum and minimum pixel values) and sample within these feasibility constraints, which differentiates our approach from sampling tools such as PROVERO [6] which do not implement such constraints.

Additionally, we present a method and metric for reporting the variety of robustness violations based on confusion matrices, and demonstrate how this approach can be used to visualize and measure the variety of robustness violations found by analyzing multiple robustness regions. We use the L_2 (Euclidean) distance metric for defining our robustness radius, as it is a common metric for robustness [5]–[7]. However, the approach can be used for any robustness region shape for which the outer edge can be uniformly sampled.

Our key contributions in this paper and the sections in which we discuss them are as follows: (1) Formalization of multi-robustness claims and metrics for multi-robustness testing (Section II). (2) An automated black-box method for multi-robustness testing that maximizes the number of robustness faults found within a given amount of testing time by focusing on inputs on the edges of the robustness regions (Section III-A). (3) Bounds analysis to avoid analyzing infeasible samples (Section III-B). (4) Adjacency analysis to avoid reporting robustness faults in areas of uncertainty (Section III-C). (5) Implementation of our approach as a tool MuL_2 (**Mul**ti $\mathbf{L}_2$ Robustness) and its experimental evaluation (Section IV).

II. Multi-Robustness

In this section, we provide an overview of neural network robustness and confusion matrices for neural network accuracy analysis and define multi-robustness and metrics for multi-robustness testing.

A. Robustness

Robustness in neural networks is the ability of the network to maintain a correct result despite changes (perturbations) to the input that preserve its meaning (semantic-preserving). In the image domain, for instance, perturbations can involve changing one or more pixels to create an image that appears identical to human eyes but may cause a misclassification [1], or more complicated mutations such as translation, rotation, or adding rain [4], [8]–[17]. A common method of robustness analysis is to designate a region containing a correctly classified input which, with any allowed perturbation in the region, is expected to still maintain the original classification. This region can then be analyzed using various methods, including formal verification [18], [19], abstract interpretation [20], or sampling [6].

The most common approaches to robustness focus solely on one region or set of mutations about a singular input. This limits the knowledge gained by the analysis—with larger perturbations the network is expected to produce a different classification as inputs cross a decision boundary. One contribution of our approach is to identify, using multiple correctly classified inputs from the training and test set, where expected decision boundaries and thus *uncertain* samples lie and remove these from consideration (Fig. 1).

For a given classifier neural network $\mathcal{N}$ with K possible classifications and an input $X \in \mathbb{F}^n$, where n is the number of input features in X and $\mathbb{F}$ is the set of floating point numbers, $\arg\max(\mathcal{N}(X))$ denotes the classification for input X.

Given a classifier neural network $\mathcal{N}$, a robustness claim $R(\mathcal{N}, X, r)$ consists of a specific input X and a perturbation region $S_{X,r}$ (with radius r, centered around correctly classified input X) that should not contain any inputs with different classifications. We formalize this in Equation 1. In particular, we define our perturbation regions as the L_2 ball of radius r centered on X, using $\langle x_0, \cdots, x_{n-1} \rangle$ as the values of the individual input features that make up X. Then, the L_2 robustness claim $R(\mathcal{N}, X, r)$ is formalized as follows:

$$R(\mathcal{N}, X, r) \equiv \forall X' \in S_{X,r} \;.\; \arg\max(\mathcal{N}(X')) = \arg\max(\mathcal{N}(X)) \quad \text{where} \quad S_{X,r} = \{X' \mid \sqrt{\sum_{i=0}^{n-1}(x_i - x'_i)^2} \leq r\} \tag{1}$$

Here we define the term *fault* as an input to the network X' that demonstrates the violation of the robustness claim $R(\mathcal{N}, X, r)$:

$$X' \in S_{X,r} \wedge \arg\max(\mathcal{N}(X')) \neq \arg\max(\mathcal{N}(X)) \tag{2}$$

Now we can define the concept of multi-robustness. As mentioned above, robustness for neural networks is often measured using local robustness—a small region, generally about a singular input, for which the network should not produce different results. However, analyzing a single input may not give enough information to generalize to the full network. Hence, we define multi-robustness as a strategy where multiple local perturbation regions are analyzed together.

Given a set of inputs I_R for which individual robustness is expected to hold, we formalize a multi-robustness claim $R(\mathcal{N}, I_R, r)$ for a given network $\mathcal{N}$, set of expected robust inputs I_R, and robustness radius r as follows:

$$R(N, I_R, r) \equiv \forall X \in I_R \;.\; \forall X' \in S_{X,r} \;.\; \arg\max(\mathcal{N}(X')) = \arg\max(\mathcal{N}(X)) \quad \text{where} \quad S_{X,r} = \{X' \mid \sqrt{\sum_{i=0}^{n-1}(x_i - x'_i)^2} \leq r\} \tag{3}$$

Our black-box testing approach seeks to find as many faults as possible that violate this claim (as discussed in Section II-C and III-A) while maximizing the variety of faults reported (as discussed in Section II-B, II-C, III-B) and avoiding infeasible samples (as discussed in Section III-B) and cases that are near the expected decision boundaries (as discussed in Section III-C).

B. Confusion Matrices

To demonstrate and quantify the results we obtain from our multi-robustness testing approach, we use confusion matrices. Confusion matrices are a common tool for evaluating the testing performance of a neural network [21]. These matrices show, given a test set, the number of correctly and incorrectly classified instances, grouped by the expected classification (indicated by the column) and output classification (indicated by the row). For a network $\mathcal{N}$ with K classifications, the full confusion matrix will be of size $K \times K$. Fig. 2 shows example confusion matrices for a network classifying images as either a cat, dog, or rabbit. Entries on the diagonal indicate success—the number of correctly classified instances—while entries off the diagonal indicate the number of classification failures, grouped both by the expected and output classification.

The confusion matrices shown in Fig. 2 show two test sets with the same number of misclassifications. The test set for the matrix on the left shows more variety of faults. We formalize these concepts as *fault coverage* and *fault count* below and construct a *robustness confusion matrix* where the test set is obtained from the perturbation regions around a set of correctly classified inputs.

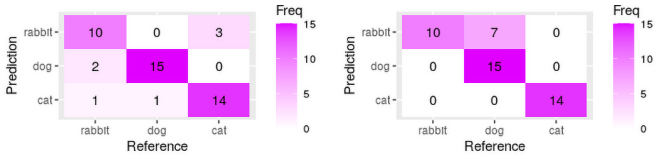

Fig. 2. Two sample confusion matrices for a network that classifies images as dogs, cats, or rabbits

C. Metrics for Multi-Robustness

Given a test set T, we define two key metrics for multi-robustness analysis: fault count and fault coverage. Fault count is the number of misclassified inputs within the perturbation regions ($S_{X,r}$), computed as:

$$\textit{fault count} \equiv \sum_{X' \in T} [X' \in S_{X,r} \wedge \arg\max(\mathcal{N}(X')) \neq \arg\max(\mathcal{N}(X))] \quad (4)$$

where $[expr]$ evaluates to 1 if $expr$ is true and 0 otherwise. Let C^T denote the confusion matrix created from the elements of the test set T. We define the fault coverage as follows:

$$\textit{fault coverage} \equiv \sum_{i=1}^{K} \sum_{j=1}^{K} [i \neq j \wedge C^T_{i,j} \neq 0] \quad (5)$$

Fault count denotes the number of misclassifications found, whereas fault coverage denotes the number of non-diagonal, non-zero entries in the robustness confusion matrix.

Our black-box testing approach (discussed in Section III) generates a test set T such that:

$$T \subseteq \bigcup_{X \in I_R} S_{X,r} \subseteq \mathbb{F}^n \quad (6)$$

containing only samples from the robustness regions of the elements in I_R (inputs for which the robustness claim is expected to hold), while aiming to maximize both fault count and fault coverage.

For the remainder of this paper, we will use the term *input* to refer to elements of I_R, and *sample* to refer to elements of T (the test set generated to test robustness).

III. Black-Box Multi-Robustness Testing

We present the components of our sampling-based black-box multi-robustness testing approach below. Our approach samples only the edges of the perturbation region around each input, and accounts for cases where samples within the perturbation region are infeasible and cases where the perturbation regions of inputs overlap. Our algorithm requires only black-box access to the network under analysis.

A. Surface Sampling Strategy

When looking for samples that show faults, especially since our multi-robustness testing strategy is looking at the perturbation regions around multiple inputs, we focus the search on cases that are more likely to be misclassified. It is known that perturbed samples which move further from an input are more likely to be misclassified than those close in, and other works have explored this strategy to look for misclassifications in images [8]. For our approach we also adopt this idea by sampling only the edges of the perturbation region around each input rather than the entire area. We distinguish this idea with two terms: the *robustness ball* designates the entire volume of the spehere with radius r (with distance r or closer to an input), whereas the *robustness sphere* denotes only the surface of that volume (i.e., the edges of the perturbation region) consisting of samples that are furthest away from the input within the perturbation region. We demonstrate a comparison between these two approaches in Section IV.

To uniformly sample on the surface of the L_2 volume of radius r (i.e. the surface of the perturbation region around the input), we use Algorithm 1. This algorithm, in simple terms, uniformly samples angles about the origin [22], then creates points distance r from the center at the sampled angle.

We present a refinement of this initial surface sampling algorithm in Algorithm 2, and we will discuss the additions in Sections III-B and III-C below.

B. Handling Bounded Inputs

When sampling along the L_2 sphere around each input (i.e., the surface of the perturbation region), in some cases (especially in image classification), there are bounds on the values that each input feature can take. For instance, in the MNIST dataset, images are composed of pixels ranging from black (0) to white (255), and pixel values outside of that range have no corresponding meaning.

We specify feasibility constraints as upper and lower bounds for each input feature. This type of constraints is sufficient to describe many scenarios, such as feasible pixel values in images and constraints on physical measurement input features (for example, when a value cannot be negative).

To make sure that our approach finds feasible misclassifications (after all, if an input cannot exist it is irrelevant if it is misclassified), we must take into account these feasibility bounds on each input feature while sampling. To begin, we show our approach to *complete bounds*—cases where the input lies on one or more feasibility bounds. Fig. 3 shows a set of inputs with two input features, with 2, 1, or 0 complete bounds.

Avoiding sampling outside of complete bounds is simple—we can avoid generating random inputs in those quadrants. We do this using lines 4–6 in Algorithm 2. However, a more difficult type of bound to handle is one which cuts off less than a full half of the available sampling space. For this purposes of this paper we will call this type of bound a *partial bound*.

Algorithm 1 SAMPLE($\mathcal{V}_{input}, r$)

▷ Generates a randomly selected sample with distance r from $\mathcal{V}_{input}$, with uniform distribution over the surface area.
▷ Calls function RANDNORM() to generate a randomly chosen number with Normal distribution, and DISTANCE($\mathcal{V}_1, \mathcal{V}_2$) to compute the Euclidean distance between two vectors.

Input: $\mathcal{V}_{input}$: vector of coordinates corresponding to a singular input under examination, where each coordinate is the value of an input feature; r: robustness radius.
Output: $\mathcal{V}_{sample}$: vector containing a sample, distance r from $\mathcal{V}_{input}$.

1: $V_{sample} \leftarrow \mathcal{V}_{zeroes}$ ▷ vector with length of $\mathcal{V}_{input}$ containing zeroes
2: **for** i=1 **to** *length*($\mathcal{V}_{input}$) **do**
3: $V_{sample}[i] \leftarrow$ RANDNORM()
4: **end for**
5: $d \leftarrow$ DISTANCE($\mathcal{V}_{zeroes}, \mathcal{V}_{sample}$)
6: **for** i=1 **to** *length*($\mathcal{V}_{input}$) **do**
7: $\mathcal{V}_{sample}[i] \leftarrow \mathcal{V}_{sample}[i] \times (r/d) + \mathcal{V}_{input}[i]$
8: **end for**
9: **return** $\mathcal{V}_{sample}$

Fig. 4 (Left) shows an example of an input with two input features, and some bounds on each input feature (2 complete and 1 partial). The yellow shaded area is the area of the perturbation region within the bounds and within the robustness radius, and the purple line shows the valid inputs on the surface of the perturbation region to sample. The bound on the right of the circle is a partial bound. We analyze two strategies for managing these partial bounds.

Algorithm 2 SAMPLE$_{final}(\mathcal{V}_{input}, r, B_u, B_l)$

▷ Generates a randomly selected sample from $\mathcal{V}_{input}$, satisfying bounds constraints and rejecting samples in robustness conflict regions.

▷ Calls function RANDNORM() to generate a randomly chosen number with Normal distribution, DISTANCE$(\mathcal{V}_1, \mathcal{V}_2)$ to compute the Euclidean distance between two vectors, and GETADJACENT$(\mathcal{V}_{input})$ to get a list of inputs adjacent to $\mathcal{V}_{input}$.

Input: $\mathcal{V}_{input}$: vector of coordinates corresponding to a singular input under examination, with each coordinate being the value of an input feature (for example, one pixel); r: robustness radius; B_u and B_l: vectors of (upper and lower) bounds for each input feature.
Output: $\mathcal{V}_{sample}$: vector containing a sample.

1: $V_{sample} \leftarrow \mathcal{V}_{zeroes}$ ▷ vector with length of $\mathcal{V}_{input}$ containing zeroes
2: **for** i=1 **to** *length*$(\mathcal{V}_{input})$ **do**
3: $V_{sample}[i] \leftarrow$ RANDNORM()
4: **if** $\mathcal{V}_{sample}[i]$ violates complete bound **then**
5: $\mathcal{V}_{sample}[i] \leftarrow -1 \times \mathcal{V}_{sample}[i]$
6: **end if**
7: **end for**
8: $d \leftarrow$ DISTANCE$(\mathcal{V}_{zeroes}, \mathcal{V}_{sample})$
9: **for** i=1 **to** *length*$(\mathcal{V}_{input})$ **do**
10: $\mathcal{V}_{sample}[i] \leftarrow \mathcal{V}_{sample}[i] \times (r/d) + \mathcal{V}_{input}[i]$
11: **if** *Strategy = Constrained (C)* **then**
12: **if** $\mathcal{V}_{sample}[i] > B_u[i] \vee \mathcal{V}_{sample}[i] < B_l[i]$ **then**
13: **return** "Rejected Sample"
14: **end if**
15: **else if** *Strategy = Constrained Adjusted (CA)* **then**
16: **if** $\mathcal{V}_{sample}[i] > B_u[i]$ **then**
17: $\mathcal{V}_{sample}[i] \leftarrow B_u[i]$
18: **else if** $\mathcal{V}_{sample}[i] < B_l[i]$ **then**
19: $\mathcal{V}_{sample}[i] \leftarrow B_l[i]$
20: **end if**
21: **end if**
22: **end for**
23: **for** $\mathcal{V}_{adj}$ **in** GETADJACENT$(\mathcal{V}_{input})$ **do**
24: **if** DISTANCE$(\mathcal{V}_{sample}, \mathcal{V}_{adj}) < r$ **then**
25: **return** "Rejected Sample"
26: **end if**
27: **end for**
28: **return** $\mathcal{V}_{sample}$

The first strategy is to generate samples along the available radius and reject those which do not fall within the bounds. Fig. 4 (Center Left) illustrates this approach, where the light purple line indicates the samples which are rejected and the dark purple line indicates the ones which are kept and tested in the network. We show the algorithmic addition as lines 11–14 in Algorithm 2 and name this the C bounds strategy as it is **C**onstrained.

The main benefit of this strategy is that it preserves uniformity of sampling. However, we discover experimentally (Section IV) that this approach rejects a large number of samples which wastes sample generation effort and thus time.

Our alternative approach to handling these partial bounds is more practical from a fault finding perspective, and we demonstrate in Section IV that it improves *fault count* and *fault coverage*, although it does not preserve uniformity of sampling. In this approach, we adjust inputs that are generated outside partial bounds such that they lie on the partial bound instead. Fig. 4 (Center Right) shows the samples created by this technique, along the purple line.

We do this adjustment by replacing any input feature value outside its bound with the bound value itself, as shown in Fig. 4 (Right). This approach does yield samples which are more tightly clustered in the section of the partial bound furthest from the original input, but it avoids the drawback of the C bounds approach above—since it does not reject the inputs outside of the partial bounds it does not face high rejection rates. We show this addition in lines 15–20 in Algorithm 2 and name this the CA bounds strategy for **C**onstrained **A**djusted. We compare applications of these strategies in our experimental evaluation in Section IV.

C. Handling Adjacent Inputs and Uncertain Regions

The correct location of decision boundaries for a classifier neural network is unknown—otherwise, the classifier could be implemented using those boundaries with 100% correctness and without training a neural network. Thus, we leverage the available information—the labels of training and test data—to approximate the locations of expected decision boundaries.

Within our approach, if two inputs, each with different expected classifications, have overlapping robustness regions, we consider the overlapping area to be an uncertain region for which we cannot specify faults. We demonstrate in Fig. 1 how we use this as a proxy for expected locations of intended decision boundaries.

In our approach, we reject any generated sample which lies within multiple inputs' perturbation regions and would otherwise be considered a fault. We examine in Section IV-B the benefits and drawbacks to rejecting samples within distance r of two inputs in I_R with different classifications before or

Fig. 3. Perturbation regions around three different inputs in two dimensions, with 2, 1, or 0 complete bounds

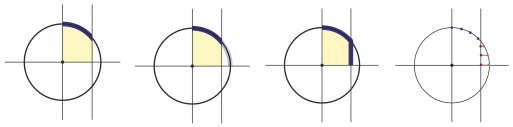

Fig. 4. Left: Perturbation region for an input constrained by three bounds—two complete and one partial; Center-Left: sampling when partial bounds are used to reject infeasible inputs (light purple); Center-Right: Sampling when infeasible inputs outside partial bounds are adjusted to lie along partial bounds; Right: Adjustment of infeasible inputs (light purple) to feasible inputs along partial bound.

after testing within the neural network. We additionally analyze the benefits and drawbacks of rejecting samples within distance r of two inputs in I_R with the same classification, as these are less likely to be misclassified.

This adjacency analysis is a key advantage of our multi-robustness analysis compared to iteratively running a single robustness tool multiple times—a single robustness tool will not account for the parts of the perturbation region that overlap with perturbation regions of inputs with different classifications, and so can flag faults that are not meaningful (for example cases where the perturbation region crosses over an expected decision boundary and thus the change in classification does not correspond to a fault). We demonstrate quantitatively in Section IV this drawback of single-robustness and how our approach remedies it. This addresses a prior criticism of robustness—a neural network cannot be fully robust because decision boundaries must exist somewhere [23]. By analyzing multiple inputs at once, we are able to avoid alerting to faults that are in actuality expected changes in classification.

A version of this check (for which any sample within distance r of two inputs in I_R is rejected) is added algorithmically as lines 23–27 in Algorithm 2. In Section IV we analyze whether this check should be done before or after running a sample in the network and whether adjacent inputs with the same classification should be considered. We compute the adjacencies for each input, to be retrieved by GETADJACENT($\mathcal{V}_{input}$), at the beginning of the multi-robustness analysis.

IV. Experimental Evaluation

In this section we present experiments and answer five research questions:

RQ1: Does sampling only at the edges of the robustness region improve fault finding?

RQ2: How do various strategies of handling infeasible inputs impact the fault discovery process?

RQ3: Can identifying adjacent inputs aid in meaningful fault discovery by avoiding marking faults near expected decision boundaries?

RQ4: Does MuL_2 discover varied neural network robustness faults in a scalable manner across multiple inputs simultaneously?

RQ5: Does MuL_2 improve fault discovery beyond existing techniques and tools?

Full data and code are available in our artifact at https://github.com/mara-downing/MuL_2.

A. Ablation Study: Sampling Strategy and Bounds Handling

In this section we answer **RQ1** and **RQ2**. We perform an ablation study of our approach (Table I) using the MNIST and CIFAR10 datasets, with 1000 correctly classified inputs in I_R to sample around (randomly chosen from the set of all correctly classified inputs from the respective dataset with the given network). Our variants are a combination of two pieces: Ball/Sphere and possible bounds strategies C or CA. Ball and Sphere indicate whether the entire perturbation region (Ball) or just the furthest edges (Sphere) are sampled. C and CA are the feasibility bounds strategies described in Section III-B; if no bounds strategy is listed, no bounds checking or adjustment is done during sample generation.

For MNIST we use two published networks, LeNet1 and LeNet5. We additionally choose the same set of 1000 correctly classified inputs for both of these networks (of the subset of inputs which both networks classify correctly). For CIFAR10 we use the published network ResNet20. All three of these networks are obtained from Xie et al. [4].

Results are presented in Table I. We chose small radii such that only around 2% of the samples per test run show faults. One key use case of MuL_2 is shown here—comparison of robustness between networks—as we can see that LeNet5 is more robust than LeNet1 on the same input set.

From this data we can see that bounds handling is imperative to obtaining feasible inputs to test—the first two rows of Table I, without bounds handling, had the highest sample rejection rates. For the Ball + C and Sphere + C approaches we can see that *fault coverage* and *fault count* are overall lower than the Ball + CA and Sphere + CA results.

In all of these tests, we see similar results between the Ball + CA and Sphere + CA options. The Sphere + CA option is slightly better on average—this small distinction is reasonable given geometric considerations: in an L_2 ball of 784 dimensions (the number of pixels in an MNIST image) and radius 4, approximately 86% of the volume is farther than distance 3.99 from the center.

Fig. 5 shows a bar chart demonstrating the difference in fault count and fault coverage between the four options with bounds handling. These experiments show that Sphere + CA is the preferred strategy for maximizing both *fault count* and *fault coverage* as defined in Section II. This answers **RQ1** and **RQ2** by showing the merits of sampling on the edge of the region and comparing bounds handling approaches to find that CA bounds handling performs best.

B. Adjacent Inputs Strategy Comparison

In this section we answer **RQ3** as well as analyze different strategies for when and how to handle sample rejection due to adjacency. For this analysis, we use the same three networks as in Section IV-A, each with a higher robustness radius for which there are a higher number of adjacencies between inputs in I_R. We present our results in Table II.

The rows of Table II are as follows: *No Adj. Rej.* signifies that no samples were rejected due to adjacency to other inputs until after testing in the network. *Diff. Adj. Rej.* signifies that

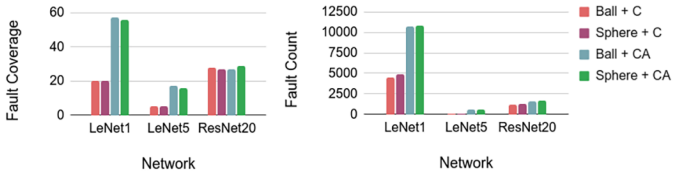

Fig. 5. Fault Coverage For Ball + C, Sphere + C, Ball + CA, and Sphere + CA Strategies.

TABLE I
ROBUSTNESS SAMPLING RESULTS FOR 2 HOURS ON MNIST AND CIFAR10 DATASET NETWORKS—ABLATION STUDY TO ANALYZE IMPACT OF SAMPLING AT EDGES OF PERTURBATION REGION AND BOUNDS HANDLING STRATEGIES; BLUE SHADING INDICATES BEST RESULT IN COLUMN

	LeNet1 (MNIST): r = 4			LeNet5 (MNIST): r = 4			ResNet20 (CIFAR10): r = 100		
MuL_2 Variant	% Rejected	# Faults	Fault Coverage	% Rejected	# Faults	Fault Coverage	% Rejected	# Faults	Fault Coverage
Ball	100.00	0	0	100.00	0	0	52.21	1186	25
Sphere	100.00	0	0	100.00	0	0	52.23	1173	26
Ball + C	98.53	4477	20	98.54	122	5	45.84	1188	28
Sphere + C	98.54	4897	20	98.54	123	5	46.02	1242	27
Ball + CA	0.00	10665	57	0.00	534	17	0.00	1561	27
Sphere + CA	0.00	10798	56	0.00	554	16	0.00	1655	29

the samples which fell into robustness conflict regions (within r of two inputs of different classes) were rejected before testing but that inputs within r of two inputs of the same class were kept and tested. *Same Adj. Rej.* signifies the opposite—samples which fell into robustness conflict regions (within r of two inputs of different classes) were kept but inputs within r of two inputs of the same class were rejected. Finally, *All Adj. Rej.* is where samples are rejected if they are within r of two or more inputs, regardless of classification.

Rep(orted) faults counts all misclassifications identified, False Pos(itives) indicates how many of those misclassifications fell within a robustness conflict region and were rejected after testing in the network (rather than before), and Real Faults is the difference of those two values.

We can see that our approach is able to find (and reject) faults which would otherwise be false positives, which would not be possible with a robustness tool which does not consider expected decision boundaries.

Rejecting samples within r of multiple inputs with the same classification has no effect on the number of false positives since these are not robustness conflict regions. More faults are found when they are not rejected as it is faster to run the sample in the network than to check these adjacencies.

When rejecting samples within r distance of multiple inputs with different classifications, some false positives occur and must be thrown out. However, waiting until after a sample has been run in the network to decide if it is a false positive is preferable, as it saves time on any case in which a sample within a robustness conflict region is classified the same as the input it was generated from and thus would not be considered and counted as a fault.

With these results, we are able to answer **RQ3** and show that our adjacency analysis is able to avoid marking faults within robustness conflict regions. We additionally find which strategy of adjacency rejection is best for discovering the most faults—only looking at adjacencies between differently classified inputs, and only checking if a sample is in a robustness conflict region after running it in the network.

C. Scalability to Large Datasets

We have already seen the the variation in faults found, but to demonstrate the scalability of our approach to larger robustness problems and complete our answer for **RQ4**, we analyze robustness of two networks trained on the ImageNet [24] dataset: Inception-V3 [25] and ResNet50 [26]. Inception-V3 expects inputs in the form of 299x299 pixel RGB images, and ResNet50 expects images in the form of 224x224 pixel RGB images. There are 1000 possible classifications which can be obtained from these images, and we give a set of 2000 correctly classified inputs in I_R.

We choose a small radius for each network (the networks expect differently sized inputs) and show our results in Table III. Additionally, we show sample images of the faults we found in Fig. 6. With these results, we answer **RQ4** and show that our approach is scalable to large inputs and input sets.

D. Critical Use Case: Street Sign Identification

To show practical utility, we apply our best strategy (Sphere + CA) to a street sign identification problem using the TSRD dataset [27] made up of images of street signs from China. This dataset contains images of 58 types of signs.

We present here a subset of the confusion matrix produced by a network trained to 98.08% accuracy on this dataset (Fig. 7). We test using $r = 5$ for 2 hours. Due to the number of available classifications, we simplify our confusion matrix to only show the classes which contributed to fault count or fault coverage.

With these results we demonstrate that our approach can be used to find meaningful faults in safety-critical networks.

E. Comparison with Symbolic Techniques, Domain-Specific Mutations, and Adversarial Attack Strategies

Finally, we set up comparison experiments with some of the most similar fault finding tools and compare their efficacy with our approach, to answer **RQ5**.

1) Scalability Comparison with White-Box Robustness via Concolic Execution: Sun et al. [2], [3] present DeepConcolic, which natively handles multiple correctly classified inputs at once and uses concolic execution to explore the search space. While symbolic reasoning typically struggles to scale to

Fig. 6. Example faults found from ResNet50 (Left) and Inception-V3 (Right) Networks for classifying the ImageNet Dataset. Left two images: original image of Tractor, followed by perturbed image, classified as Custard Apple. Right two images: original image of Fur Coat, followed by perturbed image, classified as Bow Tie.

TABLE II
ADJACENCY STRATEGY ANALYSIS, 20 MINUTES PER RUN; BLUE SHADING INDICATES BEST RESULT IN REAL FAULTS COLUMN

	LeNet1, $r = 11$			LeNet5, $r = 11$			ResNet20, $r = 3500$		
Strategy	Rep. Faults	False Pos.	Real Faults	Rep. Faults	False Pos.	Real Faults	Rep. Faults	False Pos.	Real Faults
No Adj. Rej.	15,573	279	15,294	12098	342	11756	5,158	184	4,974
Diff. Adj. Rej.	13,666	0	13,666	6,826	0	6,826	4,457	0	4,457
Same Adj. Rej.	13,957	61	13,896	9,230	69	9,161	4,526	77	4,449
All Adj. Rej.	11,826	0	11,826	5,329	0	5,329	4,034	0	4,034

TABLE III
IMAGENET ROBUSTNESS RESULTS, 2 HOURS FOR EACH TEST

Network	Fault Count	Fault Coverage
ResNet50, $r = 4$	1310	132
Inception-V3, $r = 15$	909	194

larger networks, concolic execution leverages concrete inputs alongside symbolic analysis.

We experiment using the `mnist_complicated.h5` network which they use and give as a sample in their code repository [2], [3], and run their tool with 100 randomly chosen correctly classified inputs, the NC criterion (which they found to be most effective for MNIST) and an L_∞ radius of 3. For our comparison, we use the same 100 inputs and an L_2 radius of 3, which is strictly encompassed by the L_∞ radius [28]. We ran each tool for 10 hours.

There is a major difference in the way in which faults are considered between MuL_2 and DeepConcolic—DeepConcolic [2], [3] is not bound to the seed inputs and reports faults as cases for which two samples with different classifications are within the provided L_∞ distance of each other—in essence, input pairs which demonstrate expected changes in classification across decision boundaries, the exact inputs which our approach avoids marking as faults due to such a classification change being expected.

Despite these differences, we can present quantitative results in Fig. 8 demonstrating that although we have less search space, we are able to find more faults. Due to the different interpretation of what constitutes a fault, further analysis into the types of faults discovered (fault coverage) is incomparable.

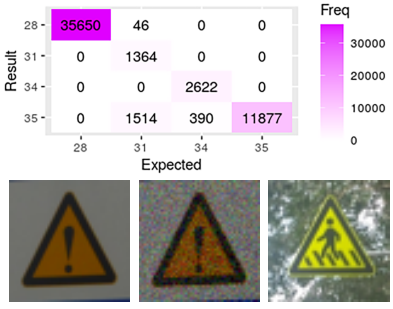

Fig. 7. Top: subset of Robustness Confusion Matrix for TSRD Classification Network with Two Hidden Layers of Size 5000. Bottom-Left: Original image of classification 34 (exclamation point). Bottom-Center: One of the 390 misclassifications of an image in the radius of 34 to 35 (crosswalk). Bottom-Right: Sample image of what classification 35 should look like.

2) Fault Coverage Comparison with Neuron-Coverage-Based Fuzzing: Next, we compare our tool with DeepHunter [4] which uses image mutations to fuzz neural networks for misclassifications. We choose the LeNet5 network for analysis as well as the most optimal combination of parameters (*prob* seed selection and NC coverage) from [4] for that network. We use the same 10 input images for each approach, which are taken from the DeepHunter experiments [4].

Within Fig. 9 (Left) we can see that as we increase the radius, we are able to find greater fault coverage than DeepHunter. However, we also know that at a certain point the images obtained will not be readily identifiable, so we present in Fig. 10 a few misclassifications we obtain at radius 9.5 to demonstrate they are still reasonable inputs. We also present in Fig. 9 (Right) how fault coverage increases over time for each approach, using radii 9 and 9.5 for MuL_2, which are above and below the fault coverage of DeepHunter in this time frame.

From these experiments we see that our approach can improve fault coverage over DeepHunter [4] by increasing the radius where we search for faults, and that these radius increases still produce meaningful misclassifications that a human can identify correctly. We also show with Fig. 9 (Right) that while DeepHunter increases fault coverage rapidly within the first 5 minutes, it levels off whereas MuL_2 is able to continue improving fault coverage further. We additionally exceed the fault count of DeepHunter at $r = 11$.

Finally, MuL_2 can be applied to any classification network, whereas DeepHunter and similar image fuzzing approaches rely on the domain-specific mutations.

3) Comparison with Adversarial Attack Tools: We present here a brief experimental comparison with the DeepRover query-efficient L_2 adversarial attack tool [5]. Used in its intended format, the DeepRover tool starts with a set of inputs to attack and removes each from the set when an attack is found against it. This approach leads to a high fault coverage, as different inputs with different classifications are successfully attacked, but a limited fault count, as each input can only be

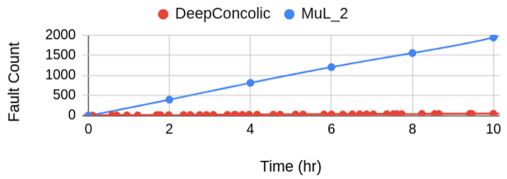

Fig. 8. Fault count of MuL_2 compared with DeepConcolic over 10 hours, using L_2 radius of 3 for MuL_2 and L_∞ radius of 3 for DeepConcolic.

attacked once. However, this approach of finding a singular fault per input does not show information such as which types of faults are more common, or whether there are classifications more or less prone to robustness faults. Thus, we set up an experiment where we use DeepRover to attempt our goal of multi-robustness analysis.

For this experiment, we select 2000 correctly classified inputs at random for I_R, using the ResNet50 network. We select an L_2 radius of 4, and use DeepRover in a loop to select one input at random from I_R, run an attack, and report either a singular fault found or a failure after a set number of attempts. We vary this attempts parameter, and demonstrate our results in Fig. 11, where both our approach (MuL_2) and DeepRover were given 2 hours.

We can see that our approach achieves both higher *fault count* and higher *fault coverage* in the same amount of time, even as we vary the number of DeepRover iterations per input.

With these three comparisons we answer **RQ5** by showing that MuL_2 improves fault discovery beyond three existing tools: DeepConcolic [2], [3], DeepHunter [4], and DeepRover [5].

F. Discussion

Through the experiments we have answered all of our research questions. We discover robustness faults in a scalable manner across multiple inputs and demonstrate the necessity of considering multiple inputs in omitting false positives in the discovered faults.

For both the DeepConcolic [2], [3] and DeepHunter [4] comparisons we chose the best set of parameters found in the original research papers (respectively) for the network and dataset. We chose a network for each that was available in their artifact. For DeepRover [5], as there was no clear choice for the number of iterations, we ran multiple tests with different numbers of iterations to compare.

Extending this approach to regression networks is feasible future work, with a different output format as confusion matrices are dependent on discrete classifications.

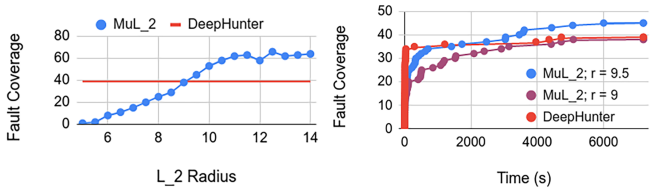

Fig. 9. Left: Fault coverage with increasing L_2 radius with MuL_2 compared with fault coverage computed by DeepHunter; Right: Fault coverage increase with time for MuL_2 using L_2 radii 9 and 9.5 compared with DeepHunter

Fig. 10. Misclassifications found by MuL_2 at L_2 radius 9.5

V. Related Work

There is much prior work on both quantitative and non-quantitative analysis of neural networks [2], [4]–[6], [8], [10], [10]–[20], [23], [29]–[75]. Within this paper, we align most with a smaller proportion of these which produce quantitative results [6], [40], [42], [49], [51]—how many misclassified inputs exist within a perturbation radius. We expand on these works with our introduction of multi-robustness, which includes identification of robustness conflict regions, as well as our introduced *fault coverage* metric.

One closely related tool, PROVERO [6], uses sampling within a single robustness region around an input to prove if robustness is above or below a threshold. Our Ball approach implemented in Table I (row 1) implements the same sampling strategy as PROVERO's L_2 robustness region. The PROVERO tool does not implement feasibility bounds, whereas we show that without these bounds implemented, many if not all samples must be rejected as they do not correspond to an image.

Addressing limitations of singular input robustness by clustering inputs with the same classification to construct larger and more complex regions expected to be robust has been explored [35]. However, in larger domains such as MNIST this clustering struggles to scale. We have demonstrated that our approach can scale to larger domains.

While our goal is not to search for adversarial attacks by input [5], [7], [76]–[80] but rather quantify the prevalence of faults, we modify the adversarial attack tool DeepRover [5] towards our goal, and show how we improve upon both fault count and fault coverage.

VI. Conclusion

We demonstrate in this work our black-box approach to analyzing L_2 robustness of multiple inputs to a network simultaneously. Through this paper, we demonstrate the scalability of our approach and our ability to find varied faults in neural networks. Additionally, we show how feasibility bounds analysis is necessary for producing valid faults and effectively limiting the number of samples that need to be rejected due to infeasibility. We demonstrate how analyzing multiple inputs simultaneously allows for automatic rejection of "false positive" faults that are in actuality changes in classification across expected decision boundaries. Finally, we show that our implementation in our tool MuL_2 performs favorably against three existing tools for fault discovery.

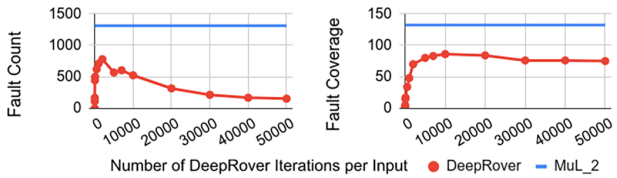

Fig. 11. Fault count (left) and coverage (right) with MuL_2 compared with DeepRover

REFERENCES

[1] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[2] Y. Sun, M. Wu, W. Ruan, X. Huang, M. Kwiatkowska, and D. Kroening, "Concolic testing for deep neural networks," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 109–119.

[3] Y. Sun, X. Huang, D. Kroening, J. Sharp, M. Hill, and R. Ashmore, "Deepconcolic: Testing and debugging deep neural networks," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 2019, pp. 111–114.

[4] X. Xie, L. Ma, F. Juefei-Xu, M. Xue, H. Chen, Y. Liu, J. Zhao, B. Li, J. Yin, and S. See, "Deephunter: a coverage-guided fuzz testing framework for deep neural networks," in *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2019, pp. 146–157.

[5] F. Zhang, X. Hu, L. Ma, and J. Zhao, "Deeprover: A query-efficient blackbox attack for deep neural networks," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2023, pp. 1384–1394.

[6] T. Baluta, Z. L. Chua, K. S. Meel, and P. Saxena, "Scalable quantitative verification for deep neural networks," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 312–323.

[7] M. Andriushchenko, F. Croce, N. Flammarion, and M. Hein, "Square attack: a query-efficient black-box adversarial attack via random search," in *European conference on computer vision*. Springer, 2020, pp. 484–501.

[8] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of the 40th international conference on software engineering*, 2018, pp. 303–314.

[9] B. C. Hu, L. Marsso, K. Czarnecki, R. Salay, H. Shen, and M. Chechik, "If a human can see it, so should your system: Reliability requirements for machine vision components," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 1145–1156.

[10] X. Xie, P. Yin, and S. Chen, "Boosting the revealing of detected violations in deep learning testing: A diversity-guided method," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, 2022, pp. 1–13.

[11] X. Gao, Z. Wang, Y. Feng, L. Ma, Z. Chen, and B. Xu, "Benchmarking robustness of ai-enabled multi-sensor fusion systems: Challenges and opportunities," *arXiv preprint arXiv:2306.03454*, 2023.

[12] F. Toledo, D. Shriver, S. Elbaum, and M. B. Dwyer, "Deeper notions of correctness in image-based dnns: Lifting properties from pixel to entities," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2023, pp. 2122–2126.

[13] Y. Tian, W. Zhang, M. Wen, S.-C. Cheung, C. Sun, S. Ma, and Y. Jiang, "Finding deviated behaviors of the compressed dnn models for image classifications," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 5, pp. 1–32, 2023.

[14] P. Zhang, B. Ren, H. Dong, and Q. Dai, "Cagfuzz: coverage-guided adversarial generative fuzzing testing for image-based deep learning systems," *IEEE Transactions on Software Engineering*, vol. 48, no. 11, pp. 4630–4646, 2021.

[15] M. Cheng, Y. Zhou, and X. Xie, "Behavexplor: Behavior diversity guided testing for autonomous driving systems," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2023, pp. 488–500.

[16] I. Dunn, H. Pouget, D. Kroening, and T. Melham, "Exposing previously undetectable faults in deep neural networks," in *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2021, pp. 56–66.

[17] S. Hu, H. Wu, P. Wang, J. Chang, Y. Tu, X. Jiang, X. Niu, and C. Nie, "Atom: Automated black-box testing of multi-label image classification systems," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE Computer Society, 2023, pp. 230–242.

[18] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljić *et al.*, "The marabou framework for verification and analysis of deep neural networks," in *International Conference on Computer Aided Verification*. Springer, 2019, pp. 443–452.

[19] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: a calculus for reasoning about deep neural networks," *Formal Methods in System Design*, pp. 1–30, 2021.

[20] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev, "Ai2: Safety and robustness certification of neural networks with abstract interpretation," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 3–18.

[21] O. Caelen, "A bayesian interpretation of the confusion matrix," *Annals of Mathematics and Artificial Intelligence*, vol. 81, no. 3-4, pp. 429–450, 2017.

[22] G. Marsaglia, "Choosing a point from the surface of a sphere," *The Annals of Mathematical Statistics*, vol. 43, no. 2, pp. 645–646, 1972.

[23] A. Kabaha and D. D. Cohen, "Verification of neural networks' global robustness," *Proceedings of the ACM on Programming Languages*, vol. 8, no. OOPSLA1, pp. 1010–1039, 2024.

[24] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, pp. 211–252, 2015.

[25] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[26] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[27] L. Huang, "Chinese traffic sign database." [Online]. Available: http://www.nlpr.ia.ac.cn/pal/trafficdata/recognition.html

[28] R. van de Geijn and M. Myers, "Advanced linear algebra: Foundations to frontiers," *Creative Commons NonCommercial (CC BY-NC)*, 2022.

[29] P. Ashok, V. Hashemi, J. Křetínskỳ, and S. Mohr, "Deepabstract: Neural network abstraction for accelerating verification," in *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2020, pp. 92–107.

[30] A. Boopathy, T.-W. Weng, P.-Y. Chen, S. Liu, and L. Daniel, "Cnn-cert: An efficient framework for certifying robustness of convolutional neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 3240–3247.

[31] E. Botoeva, P. Kouvaros, J. Kronqvist, A. Lomuscio, and R. Misener, "Efficient verification of relu-based neural networks via dependency analysis," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 3291–3299.

[32] R. Bunel, P. Mudigonda, I. Turkaslan, P. Torr, J. Lu, and P. Kohli, "Branch and bound for piecewise linear neural network verification," *Journal of Machine Learning Research*, vol. 21, no. 2020, 2020.

[33] Y. Y. Elboher, J. Gottschlich, and G. Katz, "An abstraction-based framework for neural network verification," in *International Conference on Computer Aided Verification*. Springer, 2020, pp. 43–65.

[34] M. Fazlyab, M. Morari, and G. J. Pappas, "Safety verification and robustness analysis of neural networks via quadratic constraints and semidefinite programming," *IEEE Transactions on Automatic Control*, 2020.

[35] D. Gopinath, G. Katz, C. S. Pasareanu, and C. Barrett, "Deepsafe: A data-driven approach for checking adversarial robustness in neural networks," *arXiv preprint arXiv:1710.00486*, 2017.

[36] I. B. Kadron, D. Gopinath, C. S. Păsăreanu, and H. Yu, "Case study: Analysis of autonomous center line tracking neural networks," in *Software Verification*, R. Bloem, R. Dimitrova, C. Fan, and N. Sharygina, Eds. Cham: Springer International Publishing, 2022, pp. 104–121.

[37] M. Lecuyer, V. Atlidakis, R. Geambasu, D. Hsu, and S. Jana, "Certified robustness to adversarial examples with differential privacy," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 656–672.

[38] J. Li, J. Liu, P. Yang, L. Chen, X. Huang, and L. Zhang, "Analyzing deep neural networks with symbolic propagation: Towards higher precision and faster verification," in *International Static Analysis Symposium*. Springer, 2019, pp. 296–319.

[39] W. Lin, Z. Yang, X. Chen, Q. Zhao, X. Li, Z. Liu, and J. He, "Robustness verification of classification deep neural networks via linear programming," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 11 418–11 427.

[40] R. Mangal, A. V. Nori, and A. Orso, "Robustness of neural networks: A probabilistic and practical approach," in *2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*. IEEE, 2019, pp. 93–96.

[41] M. Mirman, T. Gehr, and M. Vechev, "Differentiable abstract interpretation for provably robust neural networks," in *International Conference on Machine Learning*. PMLR, 2018, pp. 3578–3586.

[42] C. Păsăreanu, H. Converse, A. Filieri, and D. Gopinath, "On the probabilistic analysis of neural networks," in *Proceedings of the IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 2020, pp. 5–8.

[43] L. H. Sena, I. V. Bessa, M. R. Gadelha, L. C. Cordeiro, and E. Mota, "Incremental bounded model checking of artificial neural networks in cuda," in *2019 IX Brazilian Symposium on Computing Systems Engineering (SBESC)*. IEEE, 2019, pp. 1–8.

[44] G. Singh, T. Gehr, M. Mirman, M. Püschel, and M. T. Vechev, "Fast and effective robustness certification," *NeurIPS*, vol. 1, no. 4, p. 6, 2018.

[45] G. Singh, T. Gehr, M. Püschel, and M. Vechev, "Boosting robustness certification of neural networks," in *International Conference on Learning Representations*, 2018.

[46] ——, "An abstract domain for certifying neural networks," *Proceedings of the ACM on Programming Languages*, vol. 3, no. POPL, pp. 1–30, 2019.

[47] H.-D. Tran, D. M. Lopez, P. Musau, X. Yang, L. V. Nguyen, W. Xiang, and T. T. Johnson, "Star-based reachability analysis of deep neural networks," in *International Symposium on Formal Methods*. Springer, 2019, pp. 670–686.

[48] H.-D. Tran, X. Yang, D. M. Lopez, P. Musau, L. V. Nguyen, W. Xiang, S. Bak, and T. T. Johnson, "Nnv: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems," in *International Conference on Computer Aided Verification*. Springer, 2020, pp. 3–17.

[49] M. Usman, D. Gopinath, and C. S. Păsăreanu, "Quantifyml: How good is my machine learning model?" *arXiv preprint arXiv:2110.12588*, 2021.

[50] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Formal security analysis of neural networks using symbolic intervals," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018, pp. 1599–1614.

[51] S. Webb, T. Rainforth, Y. W. Teh, and M. P. Kumar, "A statistical approach to assessing neural network robustness," *arXiv preprint arXiv:1811.07209*, 2018.

[52] L. Weng, P.-Y. Chen, L. Nguyen, M. Squillante, A. Boopathy, I. Oseledets, and L. Daniel, "Proven: Verifying robustness of neural networks with a probabilistic approach," in *International Conference on Machine Learning*. PMLR, 2019, pp. 6727–6736.

[53] L. Weng, H. Zhang, H. Chen, Z. Song, C.-J. Hsieh, L. Daniel, D. Boning, and I. Dhillon, "Towards fast computation of certified robustness for relu networks," in *International Conference on Machine Learning*. PMLR, 2018, pp. 5276–5285.

[54] E. Wong, F. Schmidt, J. H. Metzen, and J. Z. Kolter, "Scaling provable adversarial defenses," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[55] H. Wu, A. Ozdemir, A. Zeljic, K. Julian, A. Irfan, D. Gopinath, S. Fouladi, G. Katz, C. Pasareanu, and C. Barrett, "Parallelization techniques for verifying neural networks," in *Proc. 20th Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD)*, vol. 1. TU Wien Academic Press, 2020, pp. 128–137.

[56] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel, "Efficient neural network robustness certification with general activation functions," *arXiv preprint arXiv:1811.00866*, 2018.

[57] M. von Stein and S. Elbaum, "Finding property violations through network falsification: Challenges, adaptations and lessons learned from openpilot," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, 2022, pp. 1–5.

[58] Y. Zhong, R. Wang, and S.-C. Khoo, "Expediting neural network verification via network reduction," *arXiv preprint arXiv:2308.03330*, 2023.

[59] B. Paulsen and C. Wang, "Example guided synthesis of linear approximations for neural network verification," in *International Conference on Computer Aided Verification*. Springer, 2022, pp. 149–170.

[60] Z. Xue, S. Liu, Z. Zhang, Y. Wu, and M. Zhang, "A tale of two approximations: Tightening over-approximation for dnn robustness verification via under-approximation," *arXiv preprint arXiv:2305.16998*, 2023.

[61] M. Casadio, E. Komendantskaya, M. L. Daggitt, W. Kokke, G. Katz, G. Amir, and I. Refaeli, "Neural network robustness as a verification property: a principled case study," in *International Conference on Computer Aided Verification*. Springer, 2022, pp. 219–231.

[62] Z. Zhao, G. Chen, J. Wang, Y. Yang, F. Song, and J. Sun, "Attack as defense: Characterizing adversarial examples using robustness," in *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2021, pp. 42–55.

[63] J. Wang, H. Qiu, Y. Rong, H. Ye, Q. Li, Z. Li, and C. Zhang, "Bet: black-box efficient testing for convolutional neural networks," in *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2022, pp. 164–175.

[64] P. Huang, Y. Yang, M. Liu, F. Jia, F. Ma, and J. Zhang, "ε-weakened robustness of deep neural networks," in *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2022, pp. 126–138.

[65] Y. Yuan, Q. Pang, and S. Wang, "Revisiting neuron coverage for dnn testing: A layer-wise and distribution-aware criterion," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 1200–1212.

[66] K. Pei, Y. Cao, J. Yang, and S. Jana, "Deepxplore: Automated whitebox testing of deep learning systems," in *proceedings of the 26th Symposium on Operating Systems Principles*, 2017, pp. 1–18.

[67] A. Odena, C. Olsson, D. Andersen, and I. Goodfellow, "Tensorfuzz: Debugging neural networks with coverage-guided fuzzing," in *International Conference on Machine Learning*. PMLR, 2019, pp. 4901–4911.

[68] Y. Huang, L. Ma, and Y. Li, "Patchcensor: Patch robustness certification for transformers via exhaustive testing," *ACM Transactions on Software Engineering and Methodology*, 2023.

[69] S. Lee, S. Cha, D. Lee, and H. Oh, "Effective white-box testing of deep neural networks with adaptive neuron-selection strategy," in *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2020, pp. 165–176.

[70] J. Guo, Q. Zhang, Y. Zhao, H. Shi, Y. Jiang, and J. Sun, "Rnn-test: Towards adversarial testing for recurrent neural network systems," *IEEE Transactions on Software Engineering*, vol. 48, no. 10, pp. 4167–4180, 2021.

[71] A. Zolfagharian, M. Abdellatif, L. C. Briand, M. Bagherzadeh, and S. Ramesh, "A search-based testing approach for deep reinforcement learning agents," *IEEE Transactions on Software Engineering*, 2023.

[72] Z. Li, X. Wu, D. Zhu, M. Cheng, S. Chen, F. Zhang, X. Xie, L. Ma, and J. Zhao, "Generative model-based testing on decision-making policies," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE Computer Society, 2023, pp. 243–254.

[73] H. You, Z. Wang, J. Chen, S. Liu, and S. Li, "Regression fuzzing for deep learning systems," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 82–94.

[74] J. Yu, S. Duan, and X. Ye, "A white-box testing for deep neural networks based on neuron coverage," *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[75] L. Wang, X. Xie, X. Du, M. Tian, Q. Guo, Z. Yang, and C. Shen, "Distxplore: Distribution-guided testing for evaluating and enhancing deep learning systems," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2023, pp. 68–80.

[76] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, 2017, pp. 506–519.

[77] W. Chen, Z. Zhang, X. Hu, and B. Wu, "Boosting decision-based black-box adversarial attacks with random sign flip," in *European Conference on Computer Vision*. Springer, 2020, pp. 276–293.

[78] P.-Y. Chen, H. Zhang, Y. Sharma, J. Yi, and C.-J. Hsieh, "Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models," in *Proceedings of the 10th ACM workshop on artificial intelligence and security*, 2017, pp. 15–26.

[79] F. Zhang, S. P. Chowdhury, and M. Christakis, "Deepsearch: A simple and effective blackbox attack for deep neural networks," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 800–812.

[80] S. Moon, G. An, and H. O. Song, "Parsimonious black-box adversarial attacks via efficient combinatorial optimization," in *International conference on machine learning*. PMLR, 2019, pp. 4636–4645.