SpecRepair: Counter-Example Guided Safety Repair of Deep Neural Networks

Fabian Bauer-Marquart¹, David Boetius¹, Stefan Leue¹, and Christian Schilling²
¹ University of Konstanz, Germany
² Aalborg University, Denmark

Abstract. Deep neural networks (DNNs) are increasingly applied in safety-critical domains, such as self-driving cars, unmanned aircraft, and medical diagnosis. It is of fundamental importance to certify the safety of these DNNs, i.e. that they comply with a formal safety specification. While safety certification tools exactly answer this question, they are of no help in debugging unsafe DNNs, requiring the developer to iteratively verify and modify the DNN until safety is eventually achieved. Hence, a repair technique needs to be developed that can produce a safe DNN automatically. To address this need, we present SpecRepair, a tool that efficiently eliminates counter-examples from a DNN and produces a provably safe DNN without harming its classification accuracy. SpecRepair combines specification-based counter-example search and resumes training of the DNN, penalizing counter-examples and certifying the resulting DNN. We evaluate SpecRepair’s effectiveness on the ACAS Xu benchmark, a DNN-based controller for unmanned aircraft, and two image classification benchmarks. The results show that SpecRepair is more successful in producing safe DNNs than comparable methods, has a shorter runtime, and produces safe DNNs while preserving their classification accuracy.

Keywords: Neural networks · safety repair · safety specification.

1 Introduction

Autonomous systems are increasingly steered by machine-learned controllers. The moment these controllers are integrated into self-driving cars [28], unmanned drones [14], or software for medical diagnosis [3], they become safety-critical. Machine learning models, such as deep neural networks (DNNs), have been shown to not be robust against small modifications to the input [33]. These inputs, which we call counter-examples, can radically change the classification outcome, thus leading to safety hazards. Consequently, various safety certification tools have been proposed to show the absence of counter-examples [13]. However, if the controller is not safe, using these tools results in a tedious process of iteratively verifying and modifying the controller until a safe version, free of any counter-examples, is eventually obtained. While several methods have addressed
this problem by focusing only on classification robustness \cite{8,24}, it is essential to target the more general formal safety properties \cite{19} instead. Such logic properties are crucial when analyzing and verifying these safety-critical systems.

To address the issues mentioned above, we introduce SpecRepair, a safety repair tool for DNNs that renders manual iterative modification obsolete. This is achieved by a counter-example search algorithm tailored to formal safety properties, a repair procedure that balances accuracy and counter-example elimination, and a final safety certification.

**Related Work.** We summarize three threads of work towards counter-example search and repair of DNNs in the context of formal safety specifications:

**Certification.** The verification community has developed techniques that provably determine a DNN’s safety \cite{22}: either giving a formal guarantee that the specification is satisfied or finding a counter-example. This problem is NP-hard, and approaches such as \cite{12,16} solve the problem precisely and thus are only suitable for relatively small DNNs. ERAN \cite{29} uses an abstract interpretation to make verification more scalable. To find a counter-example, all the above approaches require a logic encoding of the DNN to use SMT or MILP solvers, which limits their scalability, whereas we found that our way to obtain counter-examples even scales to large DNNs.

**Adversarial attacks and adversarial search.** The machine-learning community has designed several algorithms to find counter-examples. Most algorithms only consider classification robustness, which expresses that a classifier assigns the same label to similar inputs. Goodfellow et al. \cite{8} proposed the fast-gradient sign method (FGSM), one of the first such attack algorithms. Moon et al. \cite{26} accelerate search via an optimization procedure for image classifiers that perturbs only parts of the input image, and Chen et al. \cite{1} generate counter-examples optimized for the $\ell_2$ and $\ell_\infty$ norms. DL2 by Fischer et al. \cite{6} can express specifications beyond robustness and send queries to a basin-hopping optimizer. Its repair capabilities are discussed in the next paragraph. Some works monitor a DNN for adversarial attacks but do not target other safety properties \cite{11,23,2}.

**Formal safety repair of neural networks.** Verification and adversarial attacks alone only analyze DNNs statically. The ultimate goal, however, is to repair the DNNs such that they become provably safe and at the same time maintain high classification accuracy (the ratio of correct classifications amongst all inputs). We explicitly distinguish the concept of formal safety repair from ‘repairs’ that mainly target improving a model’s test accuracy, as done in \cite{34}. We also need to distinguish formal safety repair from adversarial defence techniques, such as \cite{24,10}, as these do not lead to any guarantees. DL2 by Fischer et al. \cite{6} integrates logic constraints into the DNN training procedure, but does not give formal guarantees that the resulting DNN ultimately satisfies these logic constraints. Goldberger et al. \cite{7} (minimal modification) use the verifier Marabou \cite{16} to directly modify network weight parameters to satisfy a given specification; the technique is based on SMT solving and hence suffers from limited scalability. Also, the modification of such parameters may harm the DNN’s accuracy. The approach nRepair by Dong et al. \cite{4} iteratively generates counter-examples using
a verifier. Instead of modifying the DNN directly, violating inputs are sent to a copy of the original DNN with modified parameters. Then, the combined model is verified again until no counter-example is found. The method only handles fully-connected feed-forward DNNs and does not support convolutional neural networks (CNNs). In the evaluation, we show that our approach is more efficient. Finally, Sotoudeh and Thakur [31], similarly to [7], aim to minimally modify a given DNN according to a formal specification using an LP solver. However, the specifications that the method supports are limited: for ACAS Xu-sized DNNs, only two-dimensional input regions are supported.

In conclusion, existing repair procedures mainly consider robustness, without giving any safety guarantees. Only four methods are concerned with safety specifications: however, these either have scalability issues [31, 7, 4], or do not give any formal guarantee for the resulting network [6].

Contributions. To address the lack of scalable and performance-preserving neural network repair methods, we propose SpecRepair, an efficient and effective technique for specification-based counter-example guided repair of DNNs.

First, we define the satisfaction function, an objective function that combines the function represented by the original DNN with the formal safety specification. This facilitates the search for counter-examples, i.e. inputs that lead to unsafe behavior due to violating the specification. Second, we propose an approach to find these counter-examples. For that, we turn the counter-example generation problem into an optimization [17] problem. A global optimizer then carries out the specification-based counter-example search. Third, we introduce an automated repair mechanism that uses the original DNN’s loss function and the counter-examples from the second step to create a penalized training loss function. Additional training iterations are performed on the DNN and eliminate the counter-examples in the process while preserving high accuracy. A verifier then checks specification compliance of the repaired network. Crucially, the verifier is typically used only once. Finally, we demonstrate the performance of SpecRepair compared to several state-of-the-art approaches. The experimental results show that SpecRepair efficiently finds counter-examples in the DNNs and successfully repairs more DNNs while also achieving better classification accuracy for the repaired DNNs.

2 Background

In this work, we study deep neural networks (DNNs). While our approach is independent of the particular application, to simplify the presentation, we restrict our attention to classification tasks. A deep neural network \( N : \mathbb{R}^n \rightarrow \mathbb{R}^m \) assigns a given input \( \mathbf{x} \in \mathbb{R}^n \) to confidence values \( \mathbf{y} \in \mathbb{R}^m \) for \( m \) class labels. A DNN comprises \( k \) layers that are sequentially composed such that \( N = f_k \circ \cdots \circ f_1 \). Each layer \( i \) is assigned an activation function \( \sigma_i \) and learnable parameters \( \theta_i \), consisting of a weight matrix \( W \) and a bias vector \( b \), such that the output of the \( i \)th layer is a function \( f_i : \mathbb{R}^{k_{i-1}} \rightarrow \mathbb{R}^{k_i} \) with \( f_i(\mathbf{z}) = \sigma_i(W_i \mathbf{z} + b_i) \).
We consider formal specifications \( \Phi = \{ \varphi_1, \ldots, \varphi_s \} \) composed of \( s \) input-output properties. Such a property \( \varphi = (X_\varphi, Y_\varphi) \) specifies that for all points in an input set \( X_\varphi \), the network needs to predict outputs that lie in an output set \( Y_\varphi \). For simplicity, we consider interval input sets \( X_\varphi \) and assume that \( Y_\varphi \) is a Boolean combination of constraints given in conjunctive normal form (CNF).

\[
X_\varphi = \prod_{i=1}^{n} [l_i, u_i] \quad l_i, u_i \in \mathbb{R}, l_i \leq u_i,
\]

\[
Y_\varphi = \{ y \in \mathbb{R}^m \mid y = \bigwedge_{j_1=1}^{a_\varphi} \bigvee_{j_2=1}^{b_\varphi} B_{j_1,j_2} \},
\]

where \( a_\varphi, b_\varphi \in \mathbb{N} \) are the total number of logical conjunctions and disjunctions, respectively. The atomic constraints \( B_{j_1,j_2} \) are of the form

\[
B_{j_1,j_2} \equiv g_{j_1,j_2}(y) \geq 0
\]

where the \( g_{j_1,j_2} : \mathbb{R}^m \rightarrow \mathbb{R} \) are computable functions of the output values \( y \). Common examples of output constraints include linear constraints (such as comparing two outputs \( y_1 \leq y_2 \); see Table 1 for constraints used in the running example).

An \( \ell_\infty \) robustness property \( \varphi_\epsilon \) is a special case of the above class of specifications. Such a property specifies stable classification for all inputs from a hypercubic neighborhood around a given input \( x \) with radius \( \epsilon \) and is defined for the desired class \( c \) (i.e., the value of the corresponding output neuron is \( y_c \)):

\[
X_{\varphi_\epsilon} = \prod_{i=1}^{n} [x_i - \epsilon, x_i + \epsilon], \\
Y_{\varphi_\epsilon} = \{ y \in \mathbb{R}^m \mid y_c = \max_j y_j \}.
\]

A DNN \( N \) satisfies a property \( \varphi \), resp. a specification \( \Phi \), if the following holds:

\[
N \models \varphi \iff \forall x \in X_\varphi : N(x) \in Y_\varphi \\
N \models \Phi \iff \forall \varphi \in \Phi : N \models \varphi.
\]

### 2.1 Running example

ACAS Xu [14] is a system for collision avoidance of two aircraft, consisting of 45 fully connected DNNs. Five inputs describe the relative position and speed of the two aircraft, while the outputs are five advisories, shown in Figure 1. Two additional parameters, the time until loss of vertical separation \( \tau \) and the previous advisory \( a_{\text{prev}} \), are used to index which of the 45 DNNs applies to the specific scenario. The advisory that is suggested corresponds to the DNN output with the minimum value.

To specify the safe behavior of the system, 10 safety properties have been formulated [15] (see Appendix A). Two example properties are given in Table 1. Figure 2 illustrates the goal of our paper using the ACAS Xu example: eliminate counter-examples from a given DNN by performing an automated specification-based repair. In the figure we see that the original network (left) gives an unsafe advisory in the red region, while the repaired network (right) only gives safe advisories.
Fig. 1: Running example description. Left: Input variables for the ACAS Xu DNNs [4]: Distance from ownship to intruder $\rho$, angle to intruder relative to ownship heading direction $\theta$, heading angle of intruder relative to ownship heading direction $\psi$, speed of ownship $v_{\text{own}}$, and speed of intruder $v_{\text{int}}$. Right: Output advisories: Clear-of-conflict (COC), weak left (WL), weak right (WR), strong left (SL), and strong right (SR). Both aircraft are in the same horizontal plane. Crossrange is perpendicular to the flight direction, while downrange is horizontal to the flight direction.

| Spec | $X_\varphi$ | $Y_\varphi$ |
|------|-------------|-------------|
| $\varphi_1$ | $[55947.691, \infty] \times \mathbb{R}^2 \times [1145, \infty] \times [-\infty, 60]$ | $\{ y \mid y_1 \leq 1500 \}$ |
| $\varphi_2$ | $[55947.691, \infty] \times \mathbb{R}^2 \times [1145, \infty] \times [-\infty, 60]$ | $\{ y \mid y_1 \leq \max_{\varphi \neq \varphi_1} y_1 \}$ |

Table 1: Running example properties. ACAS Xu safety properties; If the intruder is distant and is significantly slower than the ownship, $\varphi_1$ “Clear-of-conflict (COC, $y_1$) is always below 1500” and $\varphi_2$ “Clear-of-conflict is never the maximum output”, taken from [13].

3 SpecRepair Overview

In this section, we give a high-level overview of our approach called SpecRepair. A detailed explanation follows in the later sections. The general structure of SpecRepair is depicted in Figure 3. SpecRepair iterates back and forth between the two main components counter-example generation and repair until it terminates after a fixed number of repair steps.

The counter-example generation component takes a formal specification $\Phi = \{ \varphi_1, \ldots, \varphi_s \}$ and a DNN $N$ and produces $s$ counter-examples $x_1^s, \ldots, x_s^s$. Subsequently, the repair component retrains the DNN, for which it uses the original data set $(X, Y)$ that $N$ was trained with, the counter-examples obtained in the last step, and a penalty weight $\mu_i$, which steers counter-example removal, starting with the original parameters $\theta$. This strategy balances counter-example removal from the DNN and classification accuracy. After re-training has taken place, the counter-example generation component is executed again. If no counter-example
Fig. 2: Least advised actions of ACAS Xu network $N_{2,1}$ before (left) and after repair (right). The advised actions are described in Figure 1. Both aircraft are in the same horizontal plane. Crossrange is perpendicular to the flight direction, while downrange is horizontal to the flight direction. This figure visualizes property $\phi_2$ before and after repair. Here, $\tau = 0$, $a_{\text{prev}} = \text{weak left}$, $\psi = -70^\circ$, $v_{\text{own}} = 1185.0$, and $v_{\text{int}} = 7.5$. The red area shows unsafe behavior according to $\phi_2$ and thus constitutes counter-examples.

is detected, as a final step, we attempt to verify the DNN using a formal verification method. Since the previous counter-example search is fast but incomplete (i.e., may miss counter-examples), the verifier may still find a counter-example, in which case SpecRepair goes back to the repair component. Otherwise, the repaired DNN $N$ is verified and returned by SpecRepair.

In Section 4 we describe how counter-example generation for a DNN $N$ with respect to a formal specification $\Phi$ is performed. In Section 5 we explain the counter-example guided repair approach.

4 Finding Violations of Safety Specifications

In this section, we show how the existence of a counter-example can be cast as an optimization problem. This allows us to use an optimization procedure to find counter-examples.

4.1 An Optimization View on Safety Specifications

Here we show how to map a specification to an objective function, which we call the satisfaction function $f_{\text{sat}}$.

**Definition 1.** The satisfaction function for an atomic constraint $B_{j_1,j_2}$ of the form $g_{j_1,j_2}(y) \geq 0$ from (3) is defined as

$$f_{\text{sat}}B_{j_1,j_2}(y) = g_{j_1,j_2}(y).$$ 

(6)
The satisfaction function for a given specification, i.e., set of input-output properties \( \varphi = (X_\varphi, Y_\varphi) \) of the form \( (1) \) and \( (2) \), is defined as

\[
    f_{\text{sat}}(x) := \min_{\varphi \in \Phi} \min_{j_1 \in \{1 \ldots a_\varphi\}} \max_{j_2 \in \{1 \ldots b_\varphi\}} f_{\text{sat}_{B_{j_1,j_2}}}(N(x)).
\] (7)

In the following we focus on a single property \( \varphi \). Given a property \( \varphi \), the satisfaction function is negative if and only if \( \varphi \) is violated, which is summarized in the following theorem.

**Theorem 1.** Given a satisfaction function \( f_{\text{sat}} \) obtained from a network \( N \) and an input-output property \( \varphi = (X_\varphi, Y_\varphi) \), we have

\[
    N \not\models \varphi \iff \exists x \in X_\varphi : f_{\text{sat}}(x) < 0.
\]

**Proof.** Fix a network \( N \) and a property \( \varphi = (X_\varphi, Y_\varphi) \). Clearly, we have

\[
    f_{\text{sat}_{B_{j_1,j_2}}}(y) \text{ is negative if and only if } B_{j_1,j_2} \text{ is violated for } y. \tag{\ast}
\]

First suppose that \( N \not\models \varphi \). According to \( (5) \), there exists an input \( x \in X_\varphi \) such that \( N(x) \notin Y_\varphi \). Since the output constraints \( Y_\varphi \) in \( (2) \) are given in conjunctive normal form, one of the disjunctions and hence all corresponding disjuncts must be violated. By \( (\ast) \) we have that \( f_{\text{sat}_{B}} \) is negative for all these disjuncts. Thus the max in \( (7) \) and hence the image of \( f_{\text{sat}} \) itself is negative too.

Now suppose that \( N \models \varphi \). Then for each input \( x \in X_\varphi \) we have that \( N(x) \in Y_\varphi \). By a similar argument as above, in each disjunction there is at least one disjunct that is satisfied. Using \( (\ast) \), we know that \( f_{\text{sat}_{B}} \) is non-negative for this disjunct. Finally, from \( (7) \) we get that \( f_{\text{sat}}(x) \) is non-negative. \( \square \)
Property $\varphi_1$. $f_{\text{sat}} = 1500 - y_1$ encodes the specification $\varphi_1$: “The score for $y_1$ is always below 1500”. The specification also includes input constraint $l_1 \leq x_1 \leq u_1$.

Fig. 4: Running example. Safety properties on the ACAS Xu DNNs are mapped to satisfaction functions $f_{\text{sat}}$, which map counter-examples (safety violations) to negative values, shaded in red. The two examples illustrate the properties $\varphi_1$ and $\varphi_2$ from Table 1.

4.2 Using Optimization To Find Counter-examples

Using Theorem 1 for detecting counter-examples we can now equivalently minimize the function $f_{\text{sat}}$ in search of values below zero. The examples in Figure 4 show this for the topology of the DNN outputs, compared to the satisfaction function for the properties $\varphi_1$ and $\varphi_2$ from the running example in Table 1.

The satisfaction function $f_{\text{sat}}$ enables us to turn the problem of finding counter-examples witnessing a specification violation of a DNN into a multivariate optimization problem. Note that both the DNN and the specification are fully captured by $f_{\text{sat}}$ and hence we can call any off-the-shelf black-box optimization algorithm with $X_{\varphi}$ as the input bounds and $f_{\text{sat}}$ as the objective function to be minimized. The optimization procedure used here was chosen experimentally. For details refer ??.

5 Repair Framework

In the previous section, we have seen how to find counter-examples that violate the specification of a DNN. In this section, we build a framework around that algorithm to repair the DNN. By “repair” we mean to modify the network parameters such that the new DNN satisfies the specification. However, modifying the network parameters generally changes the accuracy of the DNN as well. Thus, as a second goal, we intend to preserve the accuracy of the DNN as much as possible. Our repair technique can be summarized as follows: minimize the loss in accuracy of the DNN such that the DNN satisfies the given specification.
DNN training uses unconstrained optimization of a loss function \[^9\]. In contrast, constrained optimization problems can be stated as follows:

$$\text{minimize } f(\theta) \text{ such that } c_i(\theta) \geq 0 \text{ for all } i \in \{1, \ldots, v\}.$$  

Here, \(f\) is a loss function and the \(c_i\) are constraints under which a point \(\theta\) is admissible to the problem.

We introduce constraints into the training procedure by incorporating penalty functions \[^{30}\] into the loss function. We want to minimize this loss function such that it satisfies the additional constraints \(c_i\), which are assigned a positive penalty weight \(\mu_i\), defining the penalized objective function problem as

$$\arg\min_{\theta} f(\theta) + \sum_{i=1}^{v} \mu_i \cdot c_i^+(\theta)$$  \hspace{1cm} (8)

where penalty function \(c_i^+\) is defined as \(c_i^+(\theta) = \max(0, -c_i(\theta))\). Intuitively, the penalty function forces the unconstrained optimization algorithm, which is used to solve Equation (8), to minimize the constraint violation: If a constraint is violated, it adds a large positive term to the objective function.

By enhancing the training procedure using the penalized loss function defined in Equation (8), training both incorporates model accuracy (since the old loss function is part of the new loss function) and decreases the violation of the counter-examples \(x_{c_1}^c, \ldots, x_{c_v}^c\). After each training iteration, the penalty weights \(\mu\) are updated, and the current model parameters are used as starting points for subsequent iterations.

Algorithm 1 gives a detailed view of one repair step. As inputs, the algorithm takes a DNN \(N\) to repair with weights \(\theta\), the original data set to train \(N\) (if not available, a uniform sampling of \(N\) can be used), a set of counter-examples \(x_{c_i}^c\) for the set of safety properties \(\phi_i\) that have been found during the counter-example generation step, and the training loss function \(\lambda\) originally used to train \(N\).

The algorithm first iterates over all the counter-example/property pairs that it was given, assigning each counter-example \(x_{c_i}^c\) an initial penalty weight \(\mu_i = 1\). Then, we build the constraint function \(c_i\) by using the \(f_{\text{sat}}\) function and applying it to the counter-example and current network weights (line 3).

In the second loop, the DNN \(N\) is iteratively trained. In line 6, each counter-example is converted into a penalized constraint. Each unsatisfied constraint adds a positive term to the loss function’s objective value. Therefore, the loss function is likely not minimal when there are any unsatisfied constraints. This updated loss function is then used to re-train the DNN. After training, we check if the counter-examples still occur in the re-trained DNN. If so, the penalty weight is doubled; otherwise, the successfully repaired DNN with new weights \(\theta'\) is returned.

The \textit{counter-example generation} component outputs counter-examples and hands them over to the repair component. This process is repeated iteratively. As outlined in Section 4, the satisfaction function \(f_{\text{sat}}\) is input to a global optimization algorithm. Intuitively, it would be possible to exit the optimization
Algorithm 1: Penalty function repair.

**Input:** DNN \( N : \mathbb{R}^n \rightarrow \mathbb{R}^m \) with parameters \( \theta \), data set \( (X,Y) \), counter-examples and properties \( (x_1^c, \varphi_1), (x_2^c, \varphi_2), \ldots, (x_v^c, \varphi_v) \), loss function \( \lambda : \mathbb{R}^{\dim(\theta)} \rightarrow \mathbb{R} \).

**Data:** Penalty weights \( \mu_1, \mu_2, \ldots, \mu_v \), constraint functions \( c_i : \mathbb{R}^{\dim(\theta)} \rightarrow \mathbb{R} \) for \( i \in \{1, \ldots, v\} \), penalized loss function \( \lambda' : \mathbb{R}^{\dim(\theta)} \rightarrow \mathbb{R} \).

**Output:** Repaired DNN \( N \) with new parameters \( \theta' \).

1. foreach \( i \in \{1, \ldots, v\} \) do
2. \( \mu_i \leftarrow 1; \) // default initial penalty weight
3. \( c_i(\theta') \leftarrow f_{\text{sat}}(\varphi_i(N\theta(x_i^c))) ; \)
4. end
5. while \( \exists j \in \{1, \ldots, v\} : N(x_j^c) \notin Y_{\varphi_j} \) do
6. \( \lambda(\theta') \leftarrow \lambda(\theta') + \sum_{i=1}^{v} \mu_i \cdot c_i(\theta') ; \)
7. train\( (N, (X, Y), \lambda') \);
8. foreach \( i \in \{1, \ldots, v\} \) do
9. if \( N(x_i^c) \notin Y_{\varphi_i} \) then
10. \( \mu_i \leftarrow 2\mu_i; \) // default penalty increase strategy
11. end
12. end
13. end

routine early when any negative value is detected. However, in our experiments we found that taking the counter-examples at the minimum, expressing a higher violation severity, ultimately results in more successful repairs.

6 Evaluation

This section presents our experimental evaluation, demonstrating the algorithm’s effectiveness in repairing a neural network subject to a safety specification. Our implementation of SpecRepair uses PyTorch for DNN interactions. We conduct the experiments on an Intel Xeon E5-2680 CPU with 2.4GhZ and 170GB of memory. For the final verification step, our implementation uses the verifier ERAN [29]. We note that ERAN uses an internal timeout and may hence return Unknown. This can be circumvented by increasing the timeout, but in the evaluation we use the default settings of ERAN and give up with the result Unknown instead.

6.1 Experimental Setup

Our experiments use 36 networks for the tasks of aircraft collision avoidance and image classification, where we replicate the benchmark networks from [29] for the latter. In detail:

- The collision avoidance system ACAS Xu [14] consists of 45 fully connected DNNs, \( N_{1,1} \) to \( N_{5,9} \). The inputs and outputs are described in Table A.1 in
Appendix A Each of the 45 networks has 6 hidden layers with 50 ReLU nodes. We use the 34 networks that were shown to violate at least one of the safety properties from [15] to evaluate our method. Because the training data is not openly available, we resort to a uniform sampling of the original model and compare the repaired model to it in terms of classification accuracy (reminder: the percentage of correct classifications) and mean average error (MAE) between the classification scores of the original and repaired models.

- **MNIST [20]** contains 70k grayscale images, showing a handwritten digit from 0 to 9, with $28 \times 28$ pixels. We use a fully connected DNN trained to a test accuracy of 97.8% using DiffAI-defended training [25], with five hidden dense layers of 100 units each.

- **CIFAR10 [18]** contains 60k color images, showing an object from one of ten possible classes, with $32 \times 32$ pixels. We use the benchmark CNN from [29], which was trained to an accuracy of 58.6%. It has two convolutional layers [21] and a max-pooling layer, repeated once with 24 and 32 channels, respectively, followed by two dense layers with 100 units each.

### 6.2 Counter-Example-Based Repair

We compare SpecRepair against three state-of-the-art repair techniques that we described in Section 1: minimal modification (MM) by Goldberger et al. [7], nRepair (NR) by Dong et al. [4], and DL2 by Fischer et al. [6]. We do not compare against [31] because its specification encoding is only applicable to two dimensions for small-scale networks and thus neither supports the ACAS Xu [15] nor any image classification robustness properties. We run DL2 with 5 different values for the DL2 weight parameter: 0.01, 0.05, 0.1, 0.2, and 0.5 (for more details see [6]). We analyze all final repair outcomes with ERAN to assess whether the repairs produced by the tools are genuinely safe.

For the **collision avoidance task**, we repair different problem instances: 34 DNNs subject to three different properties. As an additional challenge, we also create a combined specification $\Phi = \{\varphi_1, \varphi_2, \varphi_3, \varphi_4, \varphi_8\}$ consisting of five properties. We evaluate the classification accuracy (percentage of correct classifications) and mean average error (MAE) after repair to measure the level of correct functionality. Because we have no access to the original training and test data sets, we use a uniform sampling from the original network as test data for calculating accuracy and MAE. We set a timeout of three hours for all techniques.

For the **image classification task**, we repair a total of 100 cases for an $\ell_\infty$ robustness specification, with a robustness parameter $\epsilon = 0.03$, replicating the robustness experiment by Dong et al. [4]. Additionally, for MNIST, we compare batch repair of 10 and 25 counter-examples at the same time. This, however, is only applicable to SpecRepair and nRepair (NR), because MM and DL2 do not have this functionality. We do not compare against minimal modification (MM) [7] because of problems that let the internally used Marabou solver [16] fail to generate any counter-examples. We have reported this error to the Marabou de-
Table 2: Safety repair results of the ACAS Xu DNNs: SpecRepair (this work), DL2 [6], nRepair (NR) [4], and minimal modification (MM) [7]. We compare the cumulative repair outcome for all 35 instances, test accuracy and mean average error (MAE) after repair, and the median runtime. ‘✓’ indicates successful repairs, ‘×’ indicates failed repairs, ‘?’ indicates a timeout, and ‘?’ marks cases where the verifier returned Unknown.

Developers[^3] Also, the authors of MM themselves already described their technique to perform sub-optimally for CNF properties because it relies on the exact encoding needed by Marabou. Furthermore, because nRepair (NR) does not support convolutional layers, we cannot evaluate it on the network for CIFAR10.

We compare the number of successful repairs, test accuracy (to measure preservation of the model’s functionality), and runtime.

**Results.** Table 2 shows the aggregated repair results for the collision avoidance task, counting the number of successful repairs, failures, unknown outcomes, and timeouts. Additionally, it shows the accuracy and mean average error (MAE) of the DNNs after they have been repaired by the respective method. Results per benchmark instance are given in Table 3.

SpecRepair successfully repairs 28 of the 36 instances. The DNNs that have been repaired by SpecRepair achieve the highest classification accuracy with 99.5 and lowest mean average error with 0.1. Two times ERAN (and hence SpecRepair) terminates with a result of Unknown. Six times, SpecRepair could not repair the DNN within the time limit.

nRepair (NR) repairs one more instance than SpecRepair, but at the cost of yielding the lowest test accuracy of the three successful tools. Also, the mean average error (MAE) is extremely high: NR does not consider the classification scores but instead is only concerned with maintaining the correct class, leading to large deviations from the original policy.

While DL2’s repair accuracy is still reasonably high, with a median value of 93.4 and MAE of 6.03, it only delivers three successful repairs and fails to repair the other 33 instances. This is likely due to DL2’s algorithm design, which includes a hard-coded cross-entropy loss function and no termination criteria beyond performing a large number of iterations. Experiments using DL2 with a task loss function did not result in any successful repair.

[^3]: https://github.com/NeuralNetworkVerification/Marabou/issues/494
### Table 3: Safety repair results of the ACAS Xu DNNs: SpecRepair (this work), nRepair (NR) [4], and DL2 [5]. Each row shows the results for one benchmark instance with the property/specification in the first column (see Appendix A) and the DNN in the second column (names taken from [14]). ‘✓’ indicates successful repairs, ‘✗’ indicates failed repairs, ‘?’ indicates a timeout, and ‘?’ marks cases where the verifier returned UNKNOWN.

| Spec Model | Repair Outcome | Accuracy [%] | MAE |
|------------|----------------|--------------|-----|
| Φ₂ N₂₁ | ✓ ✓ ✗ | 99.1 83.9 | 0.22 2242.6 |
| Φ₂ N₂₂ | ✓ ✓ ✓ | 98.7 85.1 93.4 | 0.23 2279.3 6.29 |
| Φ₂ N₂₃ | ✓ ✓ ✗ | 99.3 83.5 | 0.13 2420.6 |
| Φ₂ N₂₄ | ✓ ✗ ✗ | 99.5 - | 0.09 - |
| Φ₂ N₂₅ | ✗ ✓ ✗ | - 84.1 | - 2433.8 |
| Φ₂ N₂₆ | ✗ ✓ ✗ | - 85.6 | - 2303.7 |
| Φ₂ N₂₇ | ✓ ✓ ✗ | 14.5 87.0 89.5 | 0.15 1644.8 5.97 |
| Φ₂ N₂₈ | ✓ ✓ ✗ | 99.6 87.3 | 0.14 663.6 |
| Φ₂ N₂₉ | ✓ ✓ ✗ | 99.8 88.6 | 0.13 2405.1 |
| Φ₂ N₃₀ | ✓ ✓ ✗ | 98.6 77.5 | 0.27 6.1 |
| Φ₂ N₃₁ | ✓ ✗ ✗ | 99.9 - | 0.10 - |
| Φ₂ N₃₂ | ✓ ✗ ✗ | 99.5 - | 0.10 - |
| Φ₂ N₃₃ | ✓ ✗ ✗ | 99.5 84.2 | 0.09 2384.2 |
| Φ₂ N₃₄ | ✓ ✗ ✗ | 99.5 - | - 2387.0 |
| Φ₂ N₃₅ | ✓ ✓ ✗ | 99.7 87.0 | 0.11 2251.7 |
| Φ₂ N₃₆ | ✓ ✓ ✗ | 99.7 87.9 | 0.09 1311.1 |
| Φ₂ N₃₇ | ✗ ✓ ✗ | - 87.2 | - 2442.5 |
| Φ₂ N₃₈ | ✓ ✓ ✗ | 99.8 87.7 | 0.11 1939.3 |
| Φ₂ N₃₉ | ✓ ✓ ✗ | 99.4 87.8 96.0 | 0.13 2419.4 6.03 |
| Φ₂ N₄₀ | ✓ ✓ ✗ | 99.5 87.9 | 0.10 1090.5 |
| Φ₂ N₄₁ | ✓ ✓ ✗ | 99.4 87.5 | 0.08 2.7 |
| Φ₂ N₄₂ | ✓ ✓ ✗ | 99.6 89.8 | 0.07 1329.1 |
| Φ₂ N₄₃ | ✓ ✓ ✗ | 98.3 88.9 | 0.14 1996.1 |
| Φ₂ N₄₄ | ✓ ✓ ✗ | 99.1 88.6 | 0.16 584.3 |
| Φ₂ N₄₅ | ✓ ✓ ✗ | 99.5 88.8 | 0.06 2292.2 |
| Φ₂ N₄₆ | ✗ ✓ ✗ | 99.5 87.5 | 0.11 2227.2 |
| Φ₂ N₄₇ | ✓ ✓ ✗ | 99.7 87.6 | 0.10 2438.8 |
| Φ₂ N₄₈ | ✓ ✓ ✗ | 99.6 87.8 | 0.09 405.1 |
| Φ₂ N₄₉ | ✗ ✓ ✗ | - 87.9 | - 749.8 |
| Φ₂ N₅₀ | ✓ ✗ ✗ | 99.5 - | 0.12 - |
| Φ₂ N₅₁ | ✓ ✓ ✗ | 98.4 88.0 | 0.16 957.7 |
| Φ₂ N₅₂ | ✓ ✓ ✗ | 99.4 87.7 | 0.11 382.5 |
| Φ₂ N₅₃ | ✓ ✓ ✗ | 98.1 87.9 | 0.13 181.2 |
| Φ₇ N₁₀ | ✗ ✗ ✗ | - - - | - - |
| Φ₈ N₂₉ | ✗ ✗ ✗ | - - - | - - |
| Φ₁ N₂₉ | ✗ ✗ ✗ | - - - | - - |

| Success frequency | median | MAE | median |
|-------------------|--------|-----|--------|
| 28 29 3 | 99.5 87.6 93.4 | 0.1 | 1996.1 6.03 |
Table 4: Robustness repair results of the DiffAI-defended [25] MNIST DNN: SpecRepair (this work), nRepair (NR) [4], and DL2 [8]. We compare the cumulative repair outcome for all 100 instances, the test accuracy (minimum, median, and maximum) after repair, and the median runtime.

| Tool     | Success | Fail | Unknown | Timeout | Accuracy [%] | Runtime [s] |
|----------|---------|------|---------|---------|--------------|-------------|
| SpecRepair | 100     | 0    | 0       | 0       | 96.0         | 163.6       |
| NR       | 84      | 16   | 0       | 0       | 97.3         | 36.0        |
| DL2      | 10      | 82   | 0       | 8       | 91.9         | 18839.2     |

Table 5: Collective robustness repair results of the DiffAI-defended [25] MNIST DNN: SpecRepair (this work), and nRepair (NR) [4]. We compare the cumulative repair outcome for all 100 instances, supplemented by two partitions into groups of 10 and 25 points per instance, and the test accuracy (minimum, median, and maximum) after repair.

| Tool     | Points | Inst. | Success | Fail | Unknown | Timeout | Accuracy [%] |
|----------|--------|-------|---------|------|---------|---------|--------------|
| SpecRepair       | 1      | 100   | 100     | 0    | 0       | 0       | 96.0         |
| SpecRepair      | 10     | 10    | 10      | 0    | 0       | 0       | 93.1         |
| SpecRepair      | 25     | 4     | 3       | 0    | 1       | 1       | 93.5         |
| NR              | 1      | 100   | 84      | 16   | 0       | 0       | 97.3         |
| NR              | 10     | 10    | 10      | 0    | 0       | 0       | –            |
| NR              | 25     | 4     | 4       | 0    | 0       | 0       | –            |

Minimal modification (MM) times out for every instance. We explain this with the high computational cost using the SMT-based method Marabou [16] to modify network parameters directly.

For repairing the DiffAI-defended [25] MNIST DNNs, the results are presented in Table 4. The non-aggregated data is provided in Table C.5. SpecRepair successfully repairs all 100 instances, with a median test accuracy after repair of 96%. Table 5 shows collective repair results, with SpecRepair successfully repairing 10 counter-examples at once, and for three out of four instances, it repairs 25 counter-examples in one run. nRepair repairs only 84 instances, with a slightly higher accuracy of 97.3%, and better runtime. Yet, it fails when tasked to collectively repair more than one counter-example in one execution. DL2 only repairs ten instances, failing to repair 82, and timing out on eight. Also, it achieves the lowest median repair accuracy of 91.9% for the successfully repaired cases. This indicates that DL2’s counter-example generation cannot handle the DiffAI de-
Table 6: **Robustness repair results of the CIFAR10 CNN**: SpecRepair (this work), minimal modification (MM) [7], and DL2 [6]. We compare the cumulative repair outcome for all 100 instances, the test accuracy (minimum, median, and maximum) after repair, and the median runtime.

| Tool     | Repair Outcome | Accuracy [%] | Runtime [s] |
|----------|----------------|--------------|-------------|
|          | SUCCESS | FAIL | UNKNOWN | TIMEOUT | median | median |
| SpecRepair | 88      | 5       | 1       | 6       | 69.6   | 4269.7 |
| DL2      | 100     | 0       | 0       | 0       | 61.8   | 26724.1 |

For the CIFAR10 CNN, we present the results in Table 6 [8] with non-aggregated data given in Tables C.8 to C.10. SpecRepair and DL2 are successful for 88 and 100 instances, respectively. SpecRepair maintains the highest mean test accuracy after repair with a value of 69.6%, with DL2 sacrificing quantity over accuracy, only achieving 61.6%. Also, DL2 is six times slower than SpecRepair. We observe a lower test accuracy than for the collision avoidance task and the MNIST network for all approaches. These results suggest that the higher input dimension of CIFAR10 (32 × 32 × 3 versus 28 × 28 with MNIST) is a limiting factor not only for SpecRepair but all the repair methods that we have evaluated.

### 7 Discussion

In Section 3 we gave an overview of our procedure. Maintaining the correct functionality of the repaired DNN is a fundamental challenge: a successful repair is worthless if we compromise accuracy for it. We have demonstrated that SpecRepair consistently achieves high performance on several types of networks, often outperforming state-of-the-art repair methods. The performance of SpecRepair for the collision avoidance and image classification tasks is the best among the methods compared in the evaluation in terms of successful repairs, accuracy, or scalability, while still providing formal safety guarantees. This demonstrates that SpecRepair is highly suitable for safety-critical applications.

The quality and success of our repair technique stems from its algorithmic design. Instead of relying on accurate yet computationally expensive encodings backed by SMT or linear programming, we use heuristics based on global optimization to produce counter-examples fast.

The insights gained in our experiments also support the hypothesis that using a task loss function that integrates into standard DNN training procedures, as in our approach or in [6], is not only more efficient, but also better in preserving the DNN’s accuracy. Although the approaches in [7, 31] try to keep the modifications of DNN parameters minimal, original training data is not considered, and the
experiments demonstrate that there may still be a significant negative impact on the model’s test accuracy.

One limitation of our repair approach is that, for image classification tasks, it does not always return with a successfully repaired DNN within the specified time limit. Future research may seek the combination of SpecRepair with different penalty functions during re-training to gain insights into the quality of repair results when applied to other network architectures.

8 Conclusion

We presented SpecRepair, an efficient technique for generating counter-examples and repairing deep neural networks (DNNs) such that they comply with a formal specification. Due to its black-box nature, SpecRepair supports arbitrary DNNs and specifications. Our technique consists of two main components. The first component (counter-example generation) translates the specification into an objective function, which becomes negative for all network inputs that violate the specification, and then detects counter-examples using a global optimization method. The second component (repair) utilizes these counter-examples to make the DNN safe via penalized re-training. SpecRepair finally gives a safety guarantee for the resulting DNN using a verifier. Experimental results demonstrate that SpecRepair can be used effectively for both counter-example generation and repair of DNNs, generating useful counter-examples, achieving a high quality of repair, and outperforming existing approaches.

Acknowledgments

This research was partly supported by DIREC - Digital Research Centre Denmark and the Villum Investigator Grant S40S.
Bibliography

[1] J. Chen, M. I. Jordan, and M. J. Wainwright. HopSkipJumpAttack: A query-efficient decision-based attack. In IEEE Symposium on Security and Privacy, pages 1277–1294. IEEE, 2020. URL https://doi.org/10.1109/SP40000.2020.00045.

[2] C. Cheng. Provably-robust runtime monitoring of neuron activation patterns. In DATE, pages 1310–1313. IEEE, 2021. URL https://doi.org/10.23919/DATE51398.2021.9473957.

[3] G. R. Djavanshir, X. Chen, and W. Yang. A review of artificial intelligence’s neural networks (deep learning) applications in medical diagnosis and prediction. IT Prof., 23(3):58–62, 2021.

[4] G. Dong, J. Sun, X. Wang, X. Wang, and T. Dai. Towards repairing neural networks correctly. In QRS, pages 714–725. IEEE, 2021.

[5] S. C. Endres, C. Sandrock, and W. W. Focke. A simplicial homology algorithm for Lipschitz optimisation. J. Glob. Optim., 72(2):181–217, 2018. URL https://doi.org/10.1007/s10898-018-0645-y.

[6] M. Fischer, M. Balunovic, D. Drachsler-Cohen, T. Gehr, C. Zhang, and M. T. Vechev. DL2: training and querying neural networks with logic. In ICML, volume 97 of Proceedings of Machine Learning Research, pages 1931–1941. PMLR, 2019.

[7] B. Goldberger, G. Katz, Y. Adi, and J. Keshet. Minimal modifications of deep neural networks using verification. In LPAR, volume 73 of EPiC Series in Computing, pages 260–278. EasyChair, 2020.

[8] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. In ICLR (Poster), 2015.

[9] I. J. Goodfellow, Y. Bengio, and A. C. Courville. Deep Learning. Adaptive computation and machine learning, MIT Press, 2016. ISBN 978-0-262-03561-3. URL http://www.deeplearningbook.org/.

[10] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio. Generative adversarial networks. Commun. ACM, 63(11):139–144, 2020.

[11] T. A. Henzinger, A. Lukina, and C. Schilling. Outside the box: Abstraction-based monitoring of neural networks. In ECAI, volume 325 of FAIA, pages 2433–2440. IOS Press, 2020. https://doi.org/10.3233/FAIA200375.

[12] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu. Safety verification of deep neural networks. In CAV (1), volume 10426 of LNCS, pages 3–29. Springer, 2017.

[13] X. Huang, D. Kroening, W. Ruan, J. Sharp, Y. Sun, E. Thamo, M. Wu, and X. Yi. A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. Comput. Sci. Rev., 37:100270, 2020.
[14] K. D. Julian, J. Lopez, J. S. Brush, M. P. Owen, and M. J. Kochenderfer. Policy compression for aircraft collision avoidance systems. In 2016 IEEE/AIAA 55th Digital Avionics Systems Conference (DASC), pages 1–10. IEEE, 2016.

[15] G. Katz, C. W. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In CAV, volume 10426 of LNCS, pages 97–117. Springer, 2017. URL https://doi.org/10.1007/978-3-319-63387-9_5.

[16] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljic, D. L. Dill, M. J. Kochenderfer, and C. W. Barrett. The Marabou framework for verification and analysis of deep neural networks. In CAV (1), volume 11561 of LNCS, pages 443–452. Springer, 2019.

[17] M. J. Kochenderfer and T. A. Wheeler. Algorithms for optimization. Mit Press, 2019.

[18] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images, 2009.

[19] L. Lamport. Proving the correctness of multiprocess programs. IEEE Trans. Software Eng., 3(2):125–143, 1977.

[20] Y. LeCun and C. Cortes. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/, 2010. URL https://yann.lecun.com/exdb/mnist/.

[21] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. Neural Comput., 1(4):541–551, 1989.

[22] C. Liu, T. Arnon, C. Lazarus, C. A. Strong, C. W. Barrett, and M. J. Kochenderfer. Algorithms for verifying deep neural networks. Found. Trends Optim., 4(3-4):244–404, 2021.

[23] A. Lukina, C. Schilling, and T. A. Henzinger. Into the unknown: Active monitoring of neural networks. In RV, volume 12974 of LNCS, pages 42–61. Springer, 2021. URL https://doi.org/10.1007/978-3-030-88494-9_3.

[24] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks. In ICLR (Poster). OpenReview.net, 2018.

[25] M. Mirman, T. Gehr, and M. Vechev. Differentiable abstract interpretation for provably robust neural networks. In International Conference on Machine Learning (ICML), 2018. URL https://www.icml.cc/Conferences/2018/Schedule?showEvent=2477.

[26] S. Moon, G. An, and H. O. Song. Parsimonious black-box adversarial attacks via efficient combinatorial optimization. In ICML, volume 97 of PMLR, pages 4636–4645, 2019. URL http://proceedings.mlr.press/v97/moon19a.html.

[27] B. S. Olson, I. Hashmi, K. Molloy, and A. Shehu. Basin hopping as a general and versatile optimization framework for the characterization of biological macromolecules. Adv. Artif. Intell., 2012:674832:1–674832:19, 2012. URL https://doi.org/10.1155/2012/674832.
[28] T. Onishi, T. Motoyoshi, Y. Suga, H. Mori, and T. Ogata. End-to-end learning method for self-driving cars with trajectory recovery using a path-following function. In *IJCNN*, pages 1–8. IEEE, 2019.

[29] G. Singh, T. Gehr, M. Püschel, and M. T. Vechev. An abstract domain for certifying neural networks. *POPL*, 3:41:1–41:30, 2019. URL https://doi.org/10.1145/3290354.

[30] A. E. Smith, D. W. Coit, T. Baeck, D. Fogel, and Z. Michalewicz. Penalty functions. *Handbook of evolutionary computation*, 97(1):C5, 1997.

[31] M. Sotoudeh and A. V. Thakur. Provable repair of deep neural networks. In *PLDI*, pages 588–603. ACM, 2021.

[32] R. Storn and K. V. Price. Differential evolution - A simple and efficient heuristic for global optimization over continuous spaces. *J. Glob. Optim.*, 11(4):341–359, 1997. URL https://doi.org/10.1023/A:1008202821328.

[33] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus. Intriguing properties of neural networks. In *ICLR*, 2014. URL http://arxiv.org/abs/1312.6199.

[34] M. Usman, D. Gopinath, Y. Sun, Y. Noller, and C. S. Pasareanu. NNrepair: Constraint-based repair of neural network classifiers. In *CAV (1)*, volume 12759 of *Lecture Notes in Computer Science*, pages 3–25. Springer, 2021.
A  ACAS Xu Properties

An overview about the ACAS Xu input variables and output advisories is given in Table A.1. The ten properties provided by Katz et al. [15] are listed in Table A.2.

| Input $x$ | Semantics | Output $y$ | Semantics |
|------------|-----------|------------|-----------|
| $\rho$    | Distance from ownship to intruder | $y_1$ | Clear-of-Conflict (COC) |
| $\theta$  | Angle to intruder relative to ownship heading direction | $y_2$ | Weak left (WL) |
| $\psi$    | Heading angle of intruder relative to ownship heading direction | $y_3$ | Weak right (WR) |
| $v_{own}$ | Speed of ownship | $y_4$ | Strong left (SL) |
| $v_{int}$ | Speed of intruder | $y_5$ | Strong right (SR) |

Table A.1: **ACAS Xu network parameters.** Input variables and output classes for the ACAS Xu DNNs [14].

| Spec Model | Definition |
|------------|------------|
| $\varphi_1$ | $X_0 = \mathbb{R}^2 \times [1145, \infty] \times \mathbb{R}^2 \times [0, 1500]$ |
| $\varphi_2$ | $X_0 = \mathbb{R}^2 \times [1145, \infty] \times \mathbb{R}^2 \times [0, 1500]$ |
| $\varphi_3$ | $X_0 = \mathbb{R}^2 \times [1145, \infty] \times \mathbb{R}^2 \times [0, 1500]$ |
| $\varphi_4$ | $X_0 = \mathbb{R}^2 \times [1145, \infty] \times \mathbb{R}^2 \times [0, 1500]$ |
| $\varphi_5$ | $X_0 = \mathbb{R}^2 \times [1145, \infty] \times \mathbb{R}^2 \times [0, 1500]$ |
| $\varphi_6$ | $X_0 = \mathbb{R}^2 \times [1145, \infty] \times \mathbb{R}^2 \times [0, 1500]$ |
| $\varphi_7$ | $X_0 = \mathbb{R}^2 \times [1145, \infty] \times \mathbb{R}^2 \times [0, 1500]$ |
| $\varphi_8$ | $X_0 = \mathbb{R}^2 \times [1145, \infty] \times \mathbb{R}^2 \times [0, 1500]$ |
| $\varphi_9$ | $X_0 = \mathbb{R}^2 \times [1145, \infty] \times \mathbb{R}^2 \times [0, 1500]$ |
| $\varphi_{10}$ | $X_0 = \mathbb{R}^2 \times [1145, \infty] \times \mathbb{R}^2 \times [0, 1500]$ |

Table A.2: **ACAS Xu specifications** by [15]. The output indices are assumed to correspond to the following actions: $y_1$: COC, $y_2$: WL, $y_3$: WR, $y_4$: SL, $y_5$: SR.
B  Choice of Global Optimization Method

As a preliminary experiment, we evaluate three optimization methods: basin hopping [27], SHGO [5], and differential evolution [32], on property $\varphi_2$ and all 45 networks of the collision avoidance task. For this purpose, we compare runtime, the counter-example’s objective value (the numerical output of the $f_{sat}$ function for the counter-example) which gives an estimate about its suitability for repair, and the number of function evaluations, shown in Figure B.1.

Figure B.1: Comparison of optimization methods.

Figure B.1 shows runtime, the objective value of the counter-examples found, and the number of function evaluations for state-of-the-art optimizers that can be used by SpecRepair: basin hopping, SHGO, and differential evolution. The runtime of differential evolution is best. The counter-example quality, measured as objective function value of the counter-example, where lower values are better than higher ones, is the lowest for SHGO. A low objective value corresponds to a higher deviation from the safety specification and is important during the repair phase of SpecRepair. Also, SHGO needs few function evaluations, making it the most suitable to perform attacks on networks to which access is limited.
C Non-Aggregated Data

In this section, we give the non-aggregated data for the collision avoidance and image classification tasks from the experiments. In the former, we repair 36 problem instances (consisting of a DNN and a safety property), and in the latter, we perform a robustness repair on a DNN for 100 distinct input images. Also, we give detailed information of the runtime per algorithm phase: counter-example fixing, verification, counter-example generation, and overall.

C.1 ACAS Xu

This section gives additional information about the repair results for the collision avoidance task, counting the number of successful repairs, failures, unknown outcomes, and timeouts.

The runtimes per algorithm phase is given in in Table

| Tool     | Rep. Outcome | Accuracy [%] | MAE | Runtime [s] |
|----------|--------------|--------------|-----|-------------|
|          | ✓ | × | ? | min | med | max | med | Cx fixing | Verification | Cx generation | Overall |
| SpecRepair | 28 | 0 | 2 | 6 | 14.5 | 99.5 | 99.9 | 0.1 | 13.0 | 264.7 | 6.5 | 299.6 |
| DL2      | 33 | 3 | 0 | 0 | 89.5 | 87.6 | 96.0 | 1996.1 | 10658.5 | 94.9 | – | 1996.1 |
| MM       | 0 | 0 | 0 | 35 | 77.5 | 93.4 | 89.8 | 6.03 | 8843.4 | 2528.2 | – | 10834.1 |
| NF       | 29 | 0 | 1 | 6 | – | – | – | 0.1 | 9.9 | – | 10.0 |

Table C.1: Safety repair results of the ACAS Xu DNNs: SpecRepair (this work), DL2 [6], nRepair (NR) [4], and minimal modification (MM) [7]. We compare the cumulative repair outcome for all 35 instances, median accuracy and mean average error (MAE) after repair, and the median runtime. ✓ indicates successful repairs, × indicates failed repairs, • indicates a timeout, and ‘?’ marks cases where the verifier returned UNKNOWN.
C.2 Diff-AI defended MNIST DNN

Table C.2: **Robustness repair results of the DiffAI-defended [25] MNIST DNN**: SpecRepair (this work), nRepair (NR) [4], and DL2 [6]. We compare the cumulative repair outcome for all 100 instances, the test accuracy (minimum, median, and maximum) after repair, and the median runtime by algorithm phase. ✓ indicates successful repairs, × indicates failed repairs, ♦ indicates a timeout, and '?' marks cases where the verifier returned UNKNOWN.

Verification (via ERAN) is a lot faster for SpecRepair because it more frequently confirms that a given property is verified. For DL2 and nRepair (NR), ERAN is (mis-)used to generate counter-examples, a task that ERAN is not designed for; it is designed to report satisfaction fast through the use of abstract interpretation.

The results for DL2, as shown in Table C.2, are the best results obtained by testing five different DL2 weight parameters: 0.01, 0.05, 0.1, 0.2 and 0.5. Timeouts are ranked above failures, i.e. if repair with any of the weight values times out, that case is counted as a timeout in the above table.

The runtime of one run of the DL2 tool is calculated as the sum of the runtime for the different weight values. Because of this the median runtime of the DL2 tool exceeds the timeout (10800) by more than two hours.

Table C.3: **Robustness repair results of the DiffAI-defended [25] MNIST DNN**: SpecRepair (this work), and nRepair (NR) [4]. We compare the cumulative repair outcome for all 100 instances, with two partitions into groups of 10 and 25 instances, and the test accuracy (minimum, median, and maximum) after repair.
| Point | 0.01 | 0.05 | 0.1 | 0.2 | 0.5 | 0.01 | 0.05 | 0.1 | 0.2 | 0.5 |
|-------|------|------|-----|-----|-----|------|------|-----|-----|-----|
| 0     | x    | x    | x   | x   | x   | x    | x    | x   | x   | x   |
| 1     | x    | x    | x   | x   | x   | x    | x    | x   | x   | x   |
| 2     | x    | x    | x   | x   | x   | 52   | x    | x   | x   | x   |
| 3     | x    | x    | x   | x   | x   | 53   | x    | x   | x   | x   |
| 4     | x    | x    | x   | x   | x   | 54   | x    | x   | x   | x   |
| 5     | x    | x    | x   | x   | x   | 55   | x    | x   | 77.32% | x   |
| 6     | x    | x    | x   | x   | x   | 56   | x    | x   | x   | x   |
| 7     | x    | x    | x   | x   | x   | 57   | x    | x   | x   | x   |
| 8     | x    | x    | x   | x   | x   | 58   | x    | x   | x   | x   |
| 9     | x    | x    | x   | x   | x   | 59   | x    | x   | x   | x   |
| 10    | x    | x    | x   | x   | x   | 60   | x    | x   | x   | x   |
| 11    | x    | x    | x   | x   | x   | x    | 61   | x    | 91.58% | x   |
| 12    | x    | x    | x   | x   | x   | x    | 62   | x    | 86.73% | x   |
| 13    | x    | x    | x   | x   | x   | x    | 63   | x    | x   | x   |
| 14    | x    | x    | x   | x   | x   | x    | 64   | x    | x   | x   |
| 15    | x    | x    | x   | x   | x   | x    | 65   | x    | x   | x   |
| 16    | x    | x    | x   | x   | x   | x    | 66   | x    | x   | x   |
| 17    | x    | x    | x   | x   | x   | 67   | x    | x   | x   | x   |
| 18    | x    | x    | x   | x   | x   | x    | 68   | x    | x   | x   |
| 19    | x    | x    | x   | x   | x   | x    | 69   | x    | x   | x   |
| 20    | x    | x    | x   | x   | x   | x    | 70   | x    | x   | x   |
| 21    | x    | x    | x   | x   | x   | x    | 71   | x    | x   | x   |
| 22    | x    | x    | x   | x   | x   | x    | 72   | x    | x   | x   |
| 23    | x    | x    | x   | x   | x   | x    | 73   | x    | 93.54% | x   |
| 24    | x    | x    | x   | x   | x   | 74   | x    | 96.58% | x   |
| 25    | x    | x    | x   | x   | x   | 75   | x    | x   | x   | x   |
| 26    | x    | x    | x   | x   | x   | 76   | x    | x   | x   | x   |
| 27    | x    | x    | x   | x   | x   | 77   | x    | x   | 92.16% | x   |
| 28    | x    | x    | x   | x   | x   | 78   | x    | x   | 96.51% | x   |
| 29    | x    | x    | x   | x   | x   | 79   | x    | x   | x   | x   |
| 30    | x    | x    | x   | x   | x   | x    | 80   | x    | 96.28% | x   |
| 31    | x    | x    | x   | x   | x   | x    | 81   | x    | 95.95% | x   |
| 32    | x    | x    | x   | x   | x   | x    | 82   | x    | x   | x   |
| 33    | x    | x    | x   | x   | x   | 83   | x    | x   | x   | x   |
| 34    | x    | x    | x   | x   | x   | 84   | x    | x   | x   | x   |
| 35    | x    | x    | x   | x   | x   | 85   | x    | x   | x   | x   |
| 36    | x    | x    | x   | x   | x   | 86   | x    | x   | x   | x   |
| 37    | x    | x    | x   | x   | x   | 87   | x    | x   | x   | x   |
| 38    | x    | x    | x   | x   | x   | 88   | x    | x   | x   | x   |
| 39    | x    | x    | x   | x   | x   | 89   | x    | x   | x   | x   |
| 40    | x    | x    | x   | x   | x   | 90   | x    | x   | x   | x   |
| 41    | x    | x    | x   | x   | x   | 91   | x    | x   | x   | x   |
| 42    | x    | x    | x   | x   | x   | 92   | x    | x   | x   | x   |
| 43    | x    | x    | x   | x   | x   | 93   | x    | x   | x   | x   |
| 44    | x    | x    | x   | x   | x   | 94   | x    | x   | x   | x   |
| 45    | x    | x    | x   | x   | x   | 95   | x    | x   | x   | x   |
| 46    | x    | x    | x   | x   | x   | 96   | x    | x   | x   | x   |
| 47    | x    | x    | x   | x   | x   | 97   | x    | x   | x   | x   |
| 48    | x    | x    | x   | x   | x   | 98   | x    | x   | x   | x   |
| 49    | x    | x    | x   | x   | x   | 99   | x    | x   | x   | x   |

Table C.5: Non-aggregated robustness repair results of the DiffAI-defended [25] MNIST DNN: DL2 [6]. Repair outcome for all 100 instances and the 5 DL2 weights, and median test accuracy.
C.3 CIFAR10 CNN

As for the MNIST experiment, the results for DL2 are the best results obtained by testing five different DL2 weight parameters: 0.05, 0.01, 0.1, 0.2, and 0.5. Refer to Tables C.8 to C.10.

| Tool       | SUCCESS | FAIL | UNKNOWN | TIMEOUT | Accuracy [%] | Runtime [s] |
|------------|---------|------|---------|---------|--------------|-------------|
|            |         |      |         |         | min          | med         |
|            |         |      |         |         | max          |             |
| SpecRepair | 88      | 5    | 1       | 6       | 62.3         | 69.6        |
| DL2        | 100     | 0    | 0       | 0       | 58.7         | 61.8        |

Table C.6: Robustness repair results of the CIFAR10 CNN: SpecRepair (this work), minimal modification (MM) [7], and DL2 [6]. We compare the cumulative repair outcome for all 100 instances, the test accuracy (minimum, median, and maximum) after repair, and the median runtime.
| Point | 0.01 | 0.05 | 0.1 | 0.2 | 0.5 |
|-------|------|------|-----|-----|-----|
| 0     | 64.69%/82.70% | 62.36%/75.78% | 60.55%/72.75% | 60.35%/72.39% |
| 1     | 61.40%/75.08% | 60.11%/72.74% | 60.30%/72.19% | 59.48%/71.17% |
| 2     | 63.11%/77.91% | 61.69%/74.97% | 59.27%/71.05% |
| 3     | 61.84%/75.63% | 59.75%/71.68% | 58.79%/70.16% |
| 4     | 58.65%/70.04% | 57.55%/67.97% | 58.33%/69.28% | 58.22%/68.74% |
| 5     | 63.48%/78.53% | 62.59%/77.12% | 61.79%/74.94% |
| 6     | 62.82%/76.65% | 60.58%/72.79% | 59.99%/71.98% |
| 7     | 61.19%/73.58% | 59.66%/71.55% | 59.17%/70.59% |
| 8     | 70.51%/94.99% | 68.36%/89.87% | 66.57%/86.32% |
| 9     | 63.93%/79.41% | 62.00%/75.42% | 61.74%/74.83% |
| 10    | 59.43%/71.20% | 59.65%/71.58% | 59.23%/70.29% |
| 11    | 59.55%/70.96% | 59.90%/71.04% |
| 12    | 61.11%/73.93% | 60.52%/72.53% | 58.77%/70.29% | 59.37%/70.24% |
| 13    | 61.16%/73.29% | 58.97%/70.50% | 58.48%/69.92% |
| 14    | 61.77%/74.57% | 59.90%/71.64% | 59.56%/71.06% | 59.06%/70.13% |
| 15    | 60.21%/71.72% | 59.37%/70.40% | 59.74%/70.48% | 59.32%/70.32% |
| 16    | 63.42%/78.45% | 61.09%/73.95% | 60.65%/72.89% | 60.17%/71.84% |
| 17    | 61.10%/73.76% | 59.51%/71.34% | 59.72%/71.49% |
| 18    | 60.63%/73.01% | 58.85%/70.38% | 59.15%/70.30% | 59.03%/70.19% |
| 19    | 60.91%/74.48% | 60.03%/73.32% | 59.68%/71.26% |
| 20    | 63.09%/77.81% | 62.19%/75.98% | 60.36%/72.11% | 59.98%/71.59% |
| 21    | 62.79%/76.65% | 60.58%/72.79% | 60.94%/72.88% | 60.70%/72.47% |
| 22    | 63.76%/79.21% | 62.19%/75.39% | 62.24%/74.97% | 62.10%/74.72% |
| 23    | 60.29%/72.36% | 59.99%/71.52% | 59.78%/71.41% |
| 24    | 61.62%/74.53% | 59.57%/71.47% | 59.54%/71.13% | 59.22%/70.63% |
| 25    | 61.78%/74.71% | 61.45%/70.65% | 68.34%/89.16% | 69.24%/91.76% | 67.60%/86.57% |
| 26    | 62.52%/74.52% | 61.25%/74.18% | 59.16%/70.85% | 59.09%/70.67% | 59.10%/70.74% |
| 27    | 62.52%/76.27% | 62.19%/76.05% | 61.94%/75.38% | 61.10%/73.72% |
| 28    | 61.87%/74.71% | 60.49%/71.54% | 60.49%/71.67% | 60.35%/71.23% | 60.28%/71.32% |
| 29    | 62.72%/77.17% | 60.59%/72.73% | 59.78%/71.48% | 59.89%/71.49% | 59.20%/70.83% |

Table C.8: Non-aggregated robustness repair results of the CIFAR10 CNN: DL2 [6]. Repair outcome for the instances 0 to 49 and the 5 DL2 weights, and median test accuracy.
| Point | 0.01 | 0.05 | DL2 weight | 0.1 | 0.2 | 0.5 |
|-------|------|------|-----------|-----|-----|-----|
| 50    | 63.79% / 79.83% | 61.59% / 74.98% | ▲ | 61.47% / 74.50% |
| 51    | 65.39% / 84.18% | 63.12% / 75.34% | ▲ | 59.23% / 70.78% |
| 52    | 63.71% / 78.90% | 62.58% / 75.52% | ▲ | 62.00% / 74.73% |
| 53    | 59.89% / 71.18% | 58.81% / 69.71% | ▲ | 58.43% / 69.49% |
| 54    | ▲ | 60.38% / 73.21% | ▲ | 59.64% / 71.25% |
| 55    | 62.26% / 76.07% | 60.55% / 72.94% | ▲ | 59.27% / 70.88% |
| 56    | 60.70% / 72.99% | 59.41% / 70.49% | ▲ | 59.01% / 70.06% |
| 57    | 64.54% / 80.08% | 62.25% / 75.91% | ▲ | 61.12% / 74.04% |
| 58    | 66.00% / 84.16% | 64.14% / 79.80% | ▲ | 60.00% / 73.39% |
| 59    | 61.91% / 74.97% | 60.19% / 71.53% | 60.05% / 71.54% | 60.08% / 71.47% | 60.07% / 71.48% |
| 60    | ▲ | ▲ | ▲ | ▲ | ▲ |
| 61    | ▲ | 60.64% / 72.81% | ▲ | 61.01% / 72.99% | 60.91% / 71.82% | 60.00% / 71.51% |
| 62    | 61.92% / 76.44% | 61.61% / 74.82% | 61.35% / 74.55% | 61.12% / 73.83% | 60.17% / 71.77% |
| 63    | 60.68% / 73.61% | 60.25% / 72.10% | 59.77% / 71.45% | 59.63% / 71.13% | 59.41% / 71.02% |
| 64    | ▲ | ▲ | ▲ | ▲ | ▲ |
| 65    | 59.61% / 70.67% | 59.01% / 70.15% | 59.32% / 70.66% | 59.34% / 70.68% | ▲ |
| 66    | 65.39% / 80.08% | 64.07% / 72.33% | 63.12% / 75.34% | 62.58% / 75.52% | ▲ |
| 67    | ▲ | ▲ | ▲ | ▲ | ▲ |
| 68    | ▲ | ▲ | ▲ | ▲ | ▲ |
| 69    | 61.42% / 74.71% | 60.25% / 72.16% | ▲ | ▲ | ▲ |
| 70    | 66.24% / 80.89% | 62.19% / 76.75% | ▲ | ▲ | ▲ |
| 71    | 66.44% / 80.42% | 65.46% / 78.94% | ▲ | ▲ | ▲ |
| 72    | 67.01% / 86.71% | 66.67% / 86.02% | ▲ | ▲ | ▲ |
| 73    | 59.66% / 71.01% | 59.65% / 71.33% | ▲ | ▲ | ▲ |
| 74    | ▲ | ▲ | ▲ | ▲ | ▲ |
| 75    | 61.02% / 74.60% | 60.14% / 71.88% | ▲ | ▲ | ▲ |
| 76    | 63.73% / 78.18% | 61.39% / 74.65% | ▲ | ▲ | ▲ |
| 77    | ▲ | ▲ | ▲ | ▲ | ▲ |
| 78    | ▲ | ▲ | ▲ | ▲ | ▲ |
| 79    | ▲ | ▲ | ▲ | ▲ | ▲ |
| 80    | 64.54% / 80.08% | 62.25% / 75.91% | ▲ | ▲ | ▲ |
| 81    | 65.39% / 80.08% | 64.07% / 72.33% | ▲ | ▲ | ▲ |
| 82    | 66.24% / 80.89% | 62.19% / 76.75% | ▲ | ▲ | ▲ |
| 83    | ▲ | ▲ | ▲ | ▲ | ▲ |
| 84    | ▲ | ▲ | ▲ | ▲ | ▲ |
| 85    | ▲ | ▲ | ▲ | ▲ | ▲ |
| 86    | ▲ | ▲ | ▲ | ▲ | ▲ |
| 87    | ▲ | ▲ | ▲ | ▲ | ▲ |
| 88    | ▲ | ▲ | ▲ | ▲ | ▲ |
| 89    | ▲ | ▲ | ▲ | ▲ | ▲ |
| 90    | ▲ | ▲ | ▲ | ▲ | ▲ |
| 91    | ▲ | ▲ | ▲ | ▲ | ▲ |
| 92    | ▲ | ▲ | ▲ | ▲ | ▲ |
| 93    | ▲ | ▲ | ▲ | ▲ | ▲ |
| 94    | ▲ | ▲ | ▲ | ▲ | ▲ |
| 95    | ▲ | ▲ | ▲ | ▲ | ▲ |
| 96    | ▲ | ▲ | ▲ | ▲ | ▲ |
| 97    | ▲ | ▲ | ▲ | ▲ | ▲ |
| 98    | ▲ | ▲ | ▲ | ▲ | ▲ |
| 99    | ▲ | ▲ | ▲ | ▲ | ▲ |

Table C.10: Non-aggregated robustness repair results of the CIFAR10 CNN: DL2 [6]. Repair outcome for the instances 50 to 99 and the 5 DL2 weights, and median test accuracy.