# SpecAttack: Specification-Based Adversarial Training for Deep Neural Networks

**Fabian Bauer-Marquart[1], David Boetius[1], Stefan Leue[1], and Christian Schilling[2]**

[1]University of Konstanz, Konstanz, Germany
[1]Aalborg University, Aalborg, Denmark

**Abstract**

Safety specification-based adversarial training aims to generate examples violating a formal safety specification and therefore provides approaches for repair. The need for maintaining high prediction accuracy while ensuring the save behavior remains challenging. Thus we present SpecAttack, a query-efficient counter-example generation and repair method for deep neural networks. Using SpecAttack allows specifying safety constraints on the model to find inputs that violate these constraints. These violations are then used to repair the neural network via re-training such that it becomes provably safe. We evaluate SpecAttack's performance on the task of counter-example generation and repair. Our experimental evaluation demonstrates that SpecAttack is in most cases more query-efficient than comparable attacks, yields counter-examples of higher quality, with its repair technique being more efficient, maintaining higher functional correctness, and provably guaranteeing safety specification compliance.

## Introduction

Deep neural networks (DNNs) are increasingly applied in safety-critical domains, such as self-driving cars, unmanned aircraft, medical diagnosis, and face recognition based security protocols. Due to this widespread adoption, it is even more important to ensure that these neural networks behave as expected–that is, with respect to a formal safety specification. Unfortunately, it has been shown that neural networks are vulnerable to–sometimes intentionally crafted–adversarial examples. These inputs, also called counter-examples, cause the neural network to show unsafe behavior. Finding inputs leading to such an error is thus necessary to identify the limitations of current machine learning models and suggest ways to provably repair these models.

In this paper, we focus on the efficient generation of high-quality counter-examples, as well as leveraging them to perform a repair that produces provably safe neural networks. This approach is related to adversarial training as performed by (Goodfellow, Shlens, and Szegedy 2015; Madry et al. 2018), but considers formal safety specifications instead of a defence against robustness attacks. Therefore, we discuss three threads of work towards counter-example generation and repair of DNNs in the context of formal safety specifications:

**Verification.** The formal method community has come up with techniques that verify a neural network's safety in a provable way: either giving a formal guarantee that the specification for given input is fulfilled, or a counter-example is returned for unsafe inputs. Formal methods, such as (Huang et al. 2017), (Katz et al. 2019), or (Botoeva et al. 2020), solve an NP-hard problem and thus do not scale to large networks. ERAN (Singh et al. 2019) uses over-approximation to make verification more scalable. However, for the task of finding a counter-example, all methods mentioned above require an exact logic encoding which needs to be solved using SMT or MILP solvers.

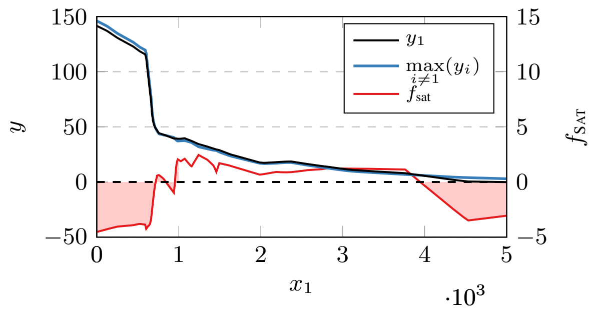

Figure 1: The satisfaction function $f_{\text{sat}}$ maps safety violations to values below zero, shaded in red. This specification $f_{\text{sat}} = y_1 - \max_{i\neq 1}(y_i)$: "$y_1$ is never the maximum value".

**Adversarial Attacks.** The machine learning community has designed several attack algorithms to generate counter-examples. Most only consider *robustness*, which is a special case of specification concerned with preventing that two very similar inputs are assigned to different labels. Such attacks have been performed by Ilyas et al. (2018) using optimization and by Brendel, Rauber, and Bethge (2018) using a random walk. For example, Moon, An, and Song (2019) use an optimization procedure for image classifiers that perturbs only parts of the input image in order to accelerate search, and Chen, Jordan, and Wainwright (2020) generate counter-examples optimized for the $L_2$ and $L_\infty$ norms. The approach DL2 by Fischer et al. (2019) can express specifications beyond robustness and sends queries to a basin-hopping optimizer; in the evaluation, we show that our approach is more scalable due to a lower number of queries.

**Repair of Neural Networks.** Verification and adversarial attacks alone only analyze DNNs statically. The ultimate goal, however, is repairing the models such that they become provably safe, while at the same time maintaining high prediction accuracy. Madry et al. (2018) perform adversarial training by enhancing the training data with adversarial perturbations, yielding networks with higher robustness. DL2 by Fischer et al. (2019) improves on the former as it supports logic specifications and integrates them as optimization constraints. Out of the box, it does not give formal guarantees that the resulting network satisfies these logic constraints. Sinitsin et al. (2020) patch faulty behaviour by fine-tuning a network to correctly handle a set of input data points gathered during deployment. Goldberger et al. (2020) (minimal modification) use a verifier to directly modify network parameters such that it satisfies a given formal specification. A neuron fixation-based (NF) repair approach by (Dong et al. 2020) uses a verifier to generate counter-examples. After modifying individual neurons, the network is verified again until no counter-example is found; in the evaluation, we show that our approach is faster because of our more efficient attack method.

Up to our knowledge, existing adversarial training procedures consider almost exclusively robustness. Only two methods mentioned above are concerned with safety specifications, but either rely on slow, formal verifiers for each repair step, or do not give any formal guarantee for the resulting network.

To address this challenge, we propose *SpecAttack*, a query-efficient, specification-based counter-example generation and repair technique for deep neural networks. **First**, we define the *satisfaction function*, which combines the original network and the formal specification to facilitate counter-example generation. This resulting function becomes negative for exactly those inputs that violate the safety property. **Second**, we turn the counter-example generation problem into an optimization problem. The specification-based attack is then carried out by a global optimization procedure. **Third**, we introduce an automated repair mechanism that uses the counter-examples from the second step, performing additional training iterations on an already-trained network and eliminating the counter-examples in the process. Therefore, the algorithm enjoys the advantages that training algorithms provide. Specification compliance of the repaired network is then guaranteed by a verifier. **Finally**, we demonstrate the efficiency of SpecAttack over several state-of-the-art attacks and repair mechanisms. Among other experimental results, SpecAttack needs fewer queries to the networks to find counter-examples, and for successful repair cases achieves better accuracy in fewer repair steps than comparable methods.

## Background

A *deep neural network* $N : \mathbb{R}^n \to \mathbb{R}^m$ assigns a given input $\mathbf{x} \in \mathbb{R}^n$ to confidence values $\mathbf{y} \in \mathbb{R}^m$ for $m$ class labels. It is comprised $k$ layers that are sequentially composed such that $N = f_1 \circ f_2 \circ \cdots \circ f_k$. Each layer is assigned an activation function $\sigma$ and learnable parameters $\boldsymbol{\theta}$ consisting of $W$ (weight) and $\mathbf{b}$ (bias), such that $f(\mathbf{x}) = \sigma(W \cdot \mathbf{x} + \mathbf{b})$. Common activation functions are linear $\sigma(x) = x$, $\mathrm{ReLU}(x) = \max(0, x)$, and convolutional (LeCun et al. 1989).

| | |
|---|---|
| $X_\varphi$ | $[55947.691, \infty] \times \mathbb{R}^2 \times [1145, \infty] \times [-\infty, 60]$ |
| $Y_\varphi$ | $\{\mathbf{y} \mid y_1 < \max_{i \neq 1} y_i\}$ |

Table 1: An example safety property, from (Katz et al. 2017)
.

We consider *formal safety specifications* composed from input-output properties. A specification consists of multiple properties: $\Phi = \{\varphi_1, \ldots, \varphi_s\}, s \in \mathbb{N}$. A property $\varphi = (X_\varphi, Y_\varphi)$ specifies that for all points in an input set $X_\varphi$, a network needs to predict outputs that lie in an output set $Y_\varphi$:

$$X_\varphi = \left\{ \prod_{i=1}^{n} [l_i, u_i] \middle| l_i, u_i \in \mathbb{R}, l_i \leq u_i \right\},$$

$$Y_\varphi = \left\{ \mathbf{y} \middle| \mathbf{y} \in \mathbb{R}^m, \mathbf{y} \models \bigwedge_{j_1=1}^{a_1} \bigvee_{j_2=1}^{a_2} B_{j_1,j_2} \right\}$$

where $a_1, a_2 \in \mathbb{N}$. Term $B_{j_1,j_2}$ has one of the following forms:

$$\begin{array}{ll} \mathbf{y}_{i_1} \leq c & \mathbf{y}_{i_1} < c \\ \mathbf{y}_{i_1} \geq c & \mathbf{y}_{i_1} > c \\ \mathbf{y}_{i_1} \geq \mathbf{y}_{i_2} & \mathbf{y}_{i_1} > \mathbf{y}_{i_2} \end{array}$$

Here, $\mathbf{y}$ is the network output with $i_1, i_2 \in \{1, \ldots, m\}$ and $c \in \mathbb{R}$ is a constant.

A network $N$ satisfies a specification $\Phi$ if the following holds:

$$N \models \varphi \Leftrightarrow \forall \mathbf{x} \in X_\varphi : N(\mathbf{x}) \in Y_\varphi$$
$$N \models \Phi \Leftrightarrow \forall \varphi \in \Phi : N \models \varphi.$$

This enables us to specify that outputs of a neural network satisfy a logical formula. In particular, we can compare individual outputs to constants, express robustness properties, and specify which output should be the minimum/maximum. The example in Table 1 is such a property, and Figure 1 shows the topology of the network outputs compared to the satisfaction function.

## An Optimization View on Safety Specifications

We now present how to map safety properties to an objective function, which we call the *satisfaction function*.

The satisfaction function $f_{\text{sat}}$ function uses the following building blocks:

$$\begin{array}{ll} B : y_{i_1} \leq c & f_{\text{sat}B}(\mathbf{y}) := c - y_{i_1} \\ B : y_{i_1} \geq c & f_{\text{sat}B}(\mathbf{y}) := y_{i_1} - c \\ B : y_{i_1} \geq y_{i_2} & f_{\text{sat}B}(\mathbf{y}) := y_{i_1} - y_{i_2} \end{array}$$

The satisfaction function of given property is then defined:

$$f_{\text{sat}}(\mathbf{x}) := \min_{j_1 \in \{1 \ldots a_1\}} \max_{j_2 \in \{1 \ldots a_2\}} f_{\text{sat}B_{j_1,j_2}}(\mathbf{x})$$

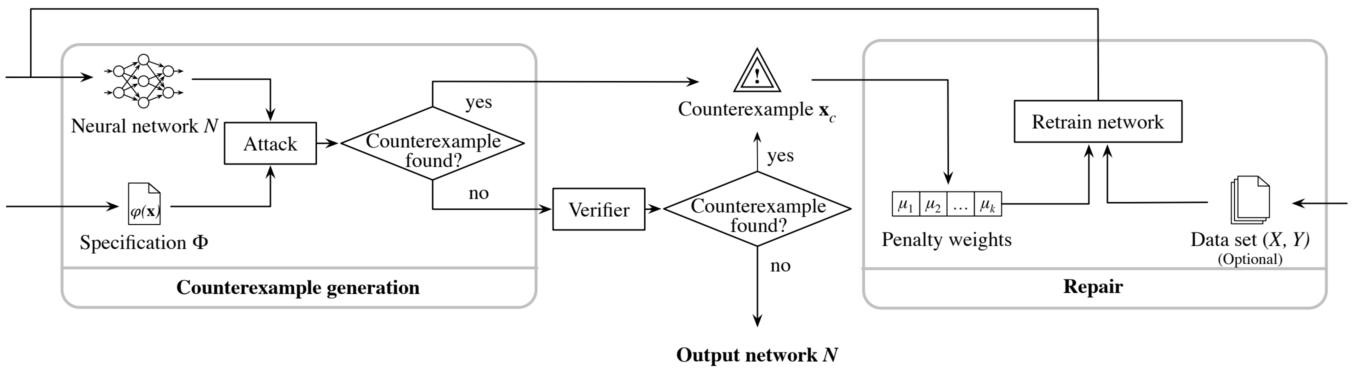

Figure 2: Overall SpecAttack architecture.

**Theorem 1.** *Given a satisfaction function $f_{\text{SAT}}$ obtained from a network $N$ and a property $\varphi$ we have*

$$N \not\models \varphi \Leftrightarrow \exists \mathbf{x} : f_{\text{SAT}}(\mathbf{x}) < 0.$$

For detecting inputs that violate the specification, which we also refer to as *counter-examples*, we can hence minimize the function $f_{\text{sat}}$ searching for values below zero.

## Optimization

The satisfaction function $f_{\text{sat}}$ enables us to turn the problem of finding specification-based counter-examples in a neural network into a multivariate optimization problem. The choice of a suited optimization routine depends on several factors. Considerations concern the unavailability of the network's gradient, good runtime, quality of the counter-examples, and a low number of function evaluations. This leaves us with four candidates:

*Basin-hopping* (Olson et al. 2012) works best for objective functions which are "funnel-like". *differential evolution* (Storn and Price 1997) and *dual annealing* (Xiang and Gong 2000) rely on stochastic minimization instead of analyzing the function landscape. Finally, *simplicial homology global optimization (SHGO)* (Endres, Sandrock, and Focke 2018) samples points on the function surface to approximate it as a triangular mesh. The last method is well-suited for optimizing functions with unknown topology. Note that none of these optimization methods guarantees the detection of the global minimum due to their stochastic nature. Experimental results by the authors show that SHGO outperforms basin-hopping and differential evolution, which itself has been shown to be more efficient than dual annealing (Price 2013). This is also supported by our evaluation in Figure 3. Overall, integrating SHGO into a general specification-based attack for neural networks seems promising since it is efficient, black-box, and not based on gradients, and therefore can deal with non-differentiable activation functions such as the popular ReLU.

We enhance the optimization procedure by increasing the sampling density in certain critical reasons: because the minimum/maximum property introduces a relatively high amount of objective value fluctuations in unrobust regions, we iteratively increase the amount of starting points for the optimizer where the output label in question has the minimal/maximal confidence, respectively. This strategy's outcome is shown in our evaluation (Table 2).

## Repair Framework

Our repair technique can be summarized as follows: *minimize* losing the correct functionality of the network *such that* the network does not violate the specification for a set of counter-examples.

Neural network training algorithms are unconstrained optimization algorithms. Penalty functions (Smith et al. 1997) allows us to introduce constraints into the training procedure. We repeat the definition here for the convenience of the reader:

$$\arg\min_{\boldsymbol{\theta}} \; f(\boldsymbol{\theta}) + \sum_i \mu_i \cdot p(c_i(\boldsymbol{\theta}))$$

where $\mu_i$ is a positive penalty weight and $p : \mathbb{R} \to \mathbb{R}$ is a penalty function that is greater than zero if constraint $c_i$ is satisfied, and zero if it is violated:

$$c_i(\boldsymbol{\theta}) < 0 \Rightarrow p(c_i(\boldsymbol{\theta})) > 0,$$
$$c_i(\boldsymbol{\theta}) \geq 0 \Rightarrow p(c_i(\boldsymbol{\theta})) = 0.$$

Here, we use the $\ell_1$ penalty function:

$$p(c_i(\boldsymbol{\theta})) = \max(0, -c_i(\boldsymbol{\theta}))$$

We now enhance the training procedure using a penalized loss functions. Training then incorporates both model accuracy and decreasing the violation of the counter-examples. After each training iteration, penalty weights are updated and model parameters are used as starting points for subsequent iterations. A detailed view of the method is given in Algorithm 1. Here, $\epsilon$ is satisfaction constant, $10^{-4}$ by default.

The top-level repair algorithm (Figure 2) repeatedly generates counter-examples and fixes them. Because SpecAttack is designed to return counter-examples that violate the specification most, i.e. minimal points of the satisfaction function $f_{\text{sat}}$, which also avoids spurious counter-examples during verification that can appear when fixing counter-examples occurring at the boundary.

**Algorithm 1:** $\ell_1$ penalty function repair.

```
Input: DNN $N : \mathbb{R}^n \to \mathbb{R}^m$, parameters $\boldsymbol{\theta}$,
       counterexamples and properties
       $(\mathbf{x}_1^c, \varphi_1), (\mathbf{x}_2^c, \varphi_2), \dots, (\mathbf{x}_v^c, \varphi_v)$, loss
       function $J : \mathbb{R}^{\#\text{param}(N)} \to \mathbb{R}$
Data: Penalty weights $\mu_1, \mu_2, \dots, \mu_v$, constraint
      functions $c_i : \mathbb{R}^{\#\text{param}(N)} \to \mathbb{R}$ for
      $i \in \{1, \dots, v\}$, penalized loss function
      $J' : \mathbb{R}^{\#\text{param}(N)} \to \mathbb{R}$
foreach $i \in \{1, \dots, v\}$ do
    $\mu_i \leftarrow 1$;
    $g_i(\boldsymbol{\theta}') \leftarrow (\text{violation\_function}(\varphi_i, \mathbf{x}_i^c))(\boldsymbol{\theta}') - \epsilon$
while $\exists i \in \{1, \dots, v\} : N(\mathbf{x}_i^c) \notin Y_{\varphi_i}$ do
    $J'(\boldsymbol{\theta}') \leftarrow J(\boldsymbol{\theta}') + \sum_{i=1}^{v} \mu_i \cdot \max(0, -g_i(\boldsymbol{\theta}'))$;
    train$(N, J')$;
    foreach $i \in \{1, 2, \dots, v\}$ do
        if $N(\mathbf{x}_i^c) \notin Y_{\varphi_i}$ then
            $\mu_i \leftarrow 2\mu_i$
```

## Evaluation

We now present a thorough experimental evaluation showing the effectiveness of SpecAttack for counter-example generation and repairing neural networks according to safety specifications. Our system is implemented in PyTorch and evaluated on an Intel Xeon E5-2680 CPU with 2.4 GhZ and 170 GB of memory. The final verification step after the repair phase is based on ERAN (Singh et al. 2019).

### Experimental Setup

Our experiments use 49 networks for the tasks of collision avoidance for aircraft and image classification. In detail:

- *ACAS Xu* (Julian, Kochenderfer, and Owen 2018) is a collision avoidance system for unmanned aircraft, consisting of 45 fully connected DNNs. The inputs are five parameters describing the relative position and speed between the two aircraft. The outputs are five advisories (clear-of-conflict, weak right, strong right, weak left, strong left), the advisory taken corresponds to the minimum output. Each of the 45 networks has 6 hidden layers with 50 ReLU nodes. To ensure the safe behavior of the DNNs, 10 safety properties have been formulated (Katz et al. 2017), which we use to evaluate our method.
- *MNIST* (LeCun and Cortes 2010) contains 70k grayscale images of handwritten digits from 0 to 9, with $28 \times 28$ pixels. We train two networks: a convolutional network (CNN) with two convolutional layers followed by a hidden dense layer with 1024 units. Replicating Chen, Jordan, and Wainwright (2020), two convolutional layers have 32 and 64 filters respectively, each followed by a max-pooling layer. The second network is a fully connected DNN with DiffAI-defended training (Mirman, Gehr, and Vechev 2018), with five hidden dense layers with 100 units each.

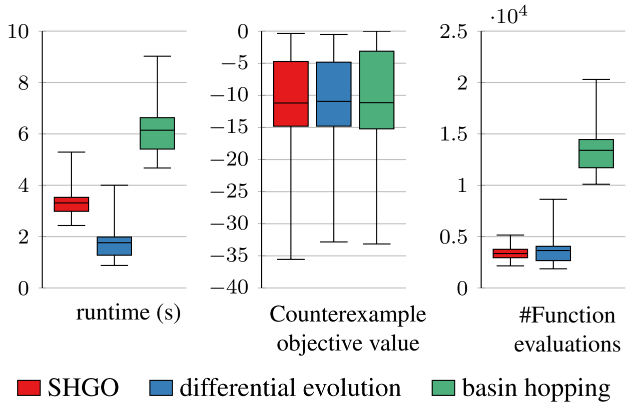

Figure 3: Comparison of optimization methods.

- *CIFAR10* (Krizhevsky, Hinton et al. 2009) contains 60k color images showing 10 different objects, with $32 \times 32$ pixels. We train two networks: a 20-layer ResNet (He et al. 2016) replicating Chen, Jordan, and Wainwright (2020), and a CNN adapted from ERAN (Singh et al. 2019) with two convolutional layers and a max-pooling layer, with this structure repeated once with 24 and 32 channels, respectively, followed by two dense layers with 100 units each.

### Specification-Based Attack

As a preliminary experiment, we evaluate three optimization methods: SHGO, differential evolution, and basin hopping 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 $f_{\text{sat}}$ function for the counter-example) which gives an estimate about its suitability for repair, and the number of function evaluations, shown in Figure 3.

For the collision avoidance task, we determine if SpecAttack can detect the failed properties among all 45 networks and all 10 properties. We report the ground truth and the SpecAttack result in Table 2. In addition, we also evaluate the naïve version of SpecAttack, using only one iteration of the optimizer. The non-convex properties $\varphi_7$, and $\varphi_8$ were split into convex parts and separately queried with SpecAttack. SpecAttack finds a counter-example in all cases, i.e. $\varphi_2$ (Table 1), $\varphi_7$, and $\varphi_8$.

For the image classification task, we compare SpecAttack to two state-of-the-art attacks: DL2 (Fischer et al. 2019) and HopSkipJumpAttack/HSJA (Chen, Jordan, and Wainwright 2020), which has been shown to outperform gradient-based attacks such as (Carlini and Wagner 2017). Two types of attack are performed: For the untargeted attack, we randomly sample images from the test set. We create a set of $L_\infty$ robustness properties for each classification label, with $L_\infty$ distance thresholds ranging from $10^{-2}$ to $10^{1}$. The resulting satisfaction function $f_{\text{sat}}$ is minimal where the original label becomes smaller than any of the others. For the targeted attack, we change the properties to specify the target label for

| Spec | Models | Ground Truth | | naïve SpecAttack | | SpecAttack | |
|---|---|---|---|---|---|---|---|
| | | #UNSAT | #SAT | #UNSAT | #UNKNOWN | #UNSAT | #UNKNOWN |
| $\varphi_1$ | all | 0 | 45 | 0 | 45 | 0 | 45 |
| $\varphi_2$ | $N_{2,1}$ to $N_{5,9}$ | 35 | 1 | **30** | **6** | **35** | **1** |
| $\varphi_3$ | all except $N_{1,7}$ to $N_{1,9}$ | 0 | 42 | 0 | 42 | 0 | 42 |
| $\varphi_4$ | all except $N_{1,7}$ to $N_{1,9}$ | 0 | 42 | 0 | 42 | 0 | 42 |
| $\varphi_5$ | $N_{1,1}$ | 0 | 1 | 0 | 1 | 0 | 1 |
| $\varphi_6$ | $N_{1,1}$ | 0 | 1 | 0 | 1 | 0 | 1 |
| $\varphi_7$ | $N_{1,9}$ | 1 | 0 | **0** | **1** | **1** | **0** |
| $\varphi_8$ | $N_{2,9}$ | 1 | 0 | **0** | **1** | **1** | **0** |
| $\varphi_9$ | $N_{3,3}$ | 0 | 1 | 0 | 1 | 0 | 1 |
| $\varphi_{10}$ | $N_{4,5}$ | 0 | 1 | 0 | 1 | 0 | 1 |

Table 2: Specification compliance analysis on the ACAS Xu benchmark (Katz et al. 2017), conducted on networks $N_{1,1}$ to $N_{5,9}$, a total of 45 networks. We compare naïve SpecAttack, and SpecAttack (this work). Some properties apply to all networks, while others apply only to a subset of these networks (second column). The ground truth is obtained using ERAN (Singh et al. 2019). UNSAT means that the specification is violated, i.e., a counterexample exists. The numbers in bold mark a difference between naïve SpecAttack and SpecAttack.

the misclassification. This modification ensures that $f_{\text{sat}}$ becomes minimal in regions where the target label is maximal.

The robustness analysis is performed analogously for the collision avoidance task. However, due to the lack of a test set, we instead sample inputs with the desired labels to create the robustness properties. We evaluate the success threshold against a) number of queries, and b) runtime.

**Results.** Figure 3 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 SpecAttack: SHGO, differential evolution, and basin hopping. While SHGO's runtime lies between the other methods, the counter-examples it finds are of better quality due to their lower objective value. This corresponds to a higher deviation from the safety specification and is important during the repair phase of SpecAttack. Also, SHGO needs the fewest function evaluations, making it the most suitable to perform attacks on networks to which access is limited.

Table 2 shows the results of our increased sampling density in unrobust input intervals. SpecAttack correctly falsifies for all networks, while the naïve implementation using only the off-the-shelf optimizer finds no counter-examples for 7 cases.

Figure 4 shows the number of queries (top) and runtime (bottom) against the median success $L_\infty$ threshold (on a log scale). It contains results for CIFAR10 with ResNet, MNIST with CNN, and $N_{2,2}$ of ACAS Xu.

In general, SpecAttack needs fewer function evaluations and shorter runtime when tightening the distance threshold compared to HSJA and DL2. For image classification, SpecAttack performs better for targeted attacks than for untargeted attacks, while for ACAS Xu, the untargeted attack needs fewer function evaluations. For HSJA, the behavior is mirrored. We explain this due to the $f_{\text{SAT}}$ construction, which uses a "$\max$" that introduces more non-differentiable sections in the function the more outputs the DNN has. We capped the number of function evaluations for DL2 at 2,000 (CIFAR10, MNIST), and at 1,000 (ACAS Xu); otherwise, DL2 needed between 3,000 to 50,000 function evaluations for an $L_\infty$ distance of 1.0 on the ACAS Xu network for example. For image classification, DL2's number of queries grows fast for $L_\infty$ distances below 0.1.

## Counter-Example-Based Repair

We compare SpecAttack against three state-of-the-art repair techniques: DL2 (Fischer et al. 2019), from which we extracted the repair step and added a verification step, minimal modification (Goldberger et al. 2020), and neuron fixation/NF (Dong et al. 2020).

For the collision avoidance task, we repair 36 models according to 3 different properties. Additionally, we form a combined specification $\Phi = \{\phi_1, \phi_2, \phi_3, \phi_4, \phi_8\}$. We evaluate the accuracy and mean average error after repair to measure the level of correct functionality. Due to the lack of the original training and test data set, we instead use the grid fidelity compared to the original network.

For the image classification task, we repair a total of 100 cases concerning an $\ell_\infty$ robustness specification, with a distance of 0.03, replicating the robustness experiment by Dong et al. (2020). We compare the number of successful repairs, test accuracy (to measure maintenance of the model's functionality), verification time, number of repair steps, and percentage of newly introduced counterexamples.

**Results.** Table 3 shows the repair results for the collision avoidance task, measuring accuracy and mean average error after repair. SpecAttack performs best, maintaining the highest level of correct functionality, having the highest classification accuracy after repair and the smallest mean average error. DL2, while successfully repairing more cases, yields qualitatively worse networks, with lower classification accuracy and higher error. Neuron fixation performs the worst, as it only repairs a single network ($N_{2,7}$) with an acceptable

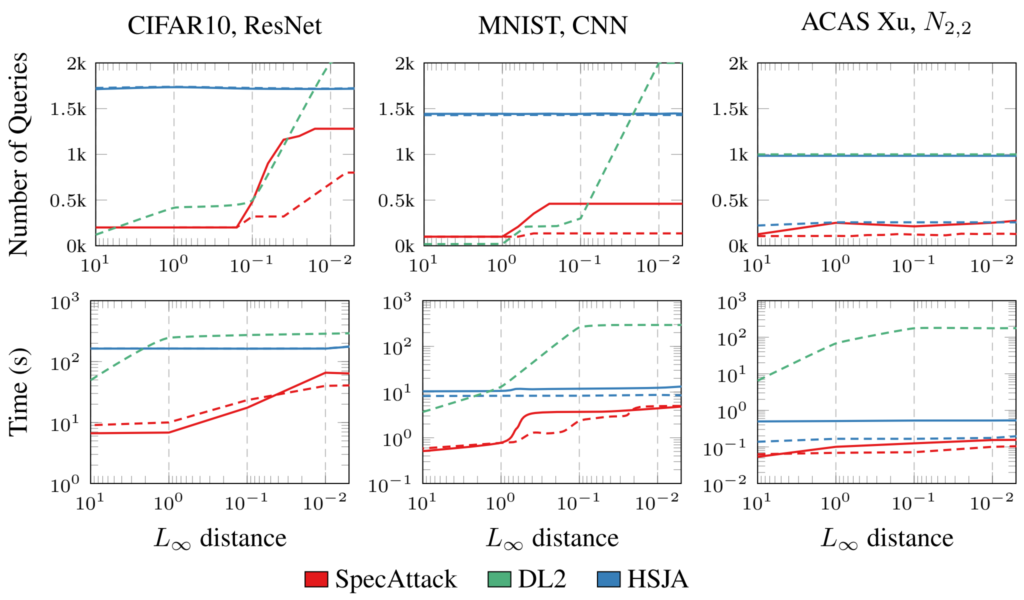

Figure 4: Query complexity and runtime versus median $L_\infty$ distance. Untargeted (solid line) and targeted (dashed line) perturbed model queries on CIFAR-10 with ResNet, MNIST with CNN, and ACAS Xu with network $N_{2,2}$ (log scale).

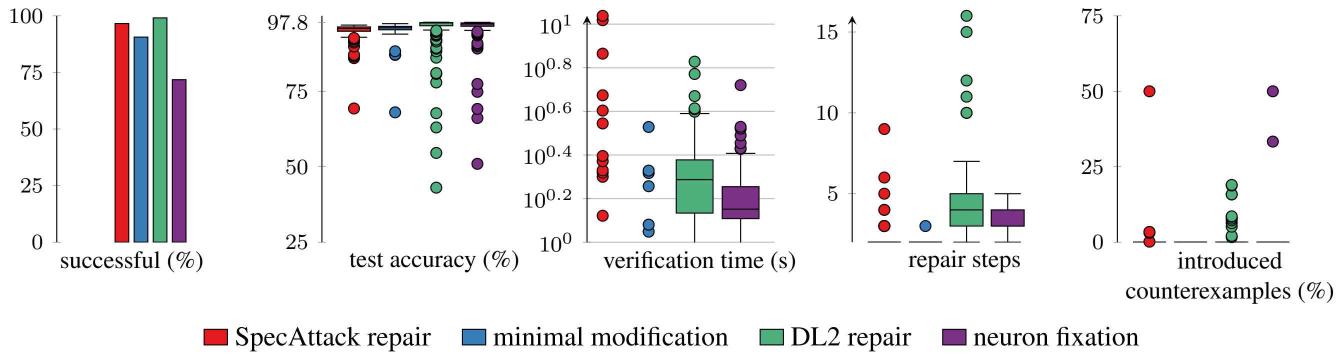

Figure 5: Repairing robustness of a DiffAI-defended MNIST CNN: Aggregated results.

mean average error. We explain this with the method's design being targeted towards small regions of the input space.

For the image classification task, Figure 5 shows the aggregated repair results for a DiffAI-defended MNIST classifier. SpecAttack introduces the smallest aggregated number of counter-examples during the repair. For successful repairs, SpecAttack ranks second, while DL2 performs best. This corresponds to the hypothesis by Sotoudeh and Thakur (2019) that DiffAI-defended networks behave more linearly, thus being easier to modify without changing much of their functionality. However, the quality of DL2's repair results is lacking, as seen in its high number of new counter-examples introduced by its repair procedure.

Figure 6 shows the aggregated repair results for CIFAR10. SpecAttack's repair success ranks second but outperforms the other approaches in terms of higher test accuracy, shorter verification time, and fewest repair steps. Minimal modification performs best in the number of successful repairs, but has worse test accuracy.

## Discussion

SpecAttack is able to craft counter-examples that substantially deviate from the safety specification, and also successfully attacks models that incorporate a defence mechanism against adversarial attacks. This shows the importance to not only statically verify models but also integrate safety assurance into the training phase to benefit from this analysis. We have demonstrated that SpecAttack achieves comparable or even superior performance to state-of-the-art repair methods on several types of networks. The performance of SpecAttack for the collision avoidance task is the best among the methods compared in the evaluation, suggesting that it is highly suitable for safety-critical applications.

Our attack method considers output properties that can compare against constants, and specify which outputs

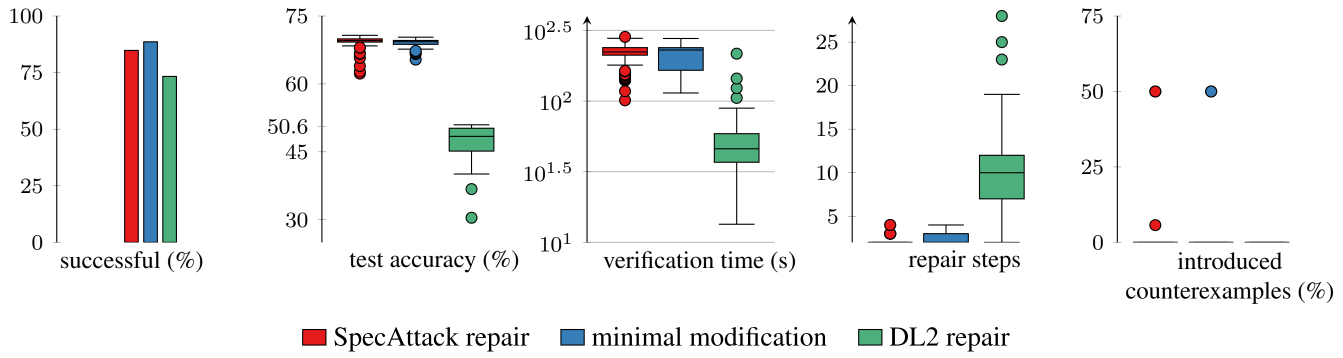

Figure 6: Repairing robustness of a CIFAR10 CNN: Aggregated results.

| Spec | Model | Accuracy SpecAttack | Accuracy DL2 | Accuracy NF | Mean average error SpecAttack | Mean average error DL2 | Mean average error NF |
|---|---|---|---|---|---|---|---|
| | $N_{2,1}$ | **99.1** | 21.1 | 83.9 | **0.22** | 0.77 | 9.08 |
| | $N_{2,2}$ | **98.7** | 96.8 | $\times$ | **0.23** | 0.37 | $\times$ |
| | $N_{2,3}$ | **99.3** | 97.6 | 83.5 | **0.13** | 0.19 | 26.55 |
| | $N_{2,4}$ | **99.5** | 99.1 | 83.6 | **0.09** | 0.14 | 2335.63 |
| | $N_{2,5}$ | 🔔 | 30.4 | **84.1** | 🔔 | **0.42** | 1023.09 |
| | $N_{2,6}$ | 🔔 | **97.1** | 85.6 | 🔔 | **0.15** | 1501.14 |
| | $N_{2,7}$ | 14.5 | 14.8 | **14.9** | **0.15** | 0.25 | 0.41 |
| | $N_{2,8}$ | **99.6** | 29.6 | $\times$ | **0.14** | 0.78 | $\times$ |
| | $N_{2,9}$ | **99.8** | 99.2 | 88.6 | **0.13** | 0.22 | 1211.91 |
| | $N_{3,1}$ | **98.6** | 80.9 | 8.8 | **0.27** | 0.57 | 3.23 |
| | $N_{3,2}$ | **99.9** | 99.8 | 86.5 | **0.10** | 0.13 | 2378.94 |
| | $N_{3,4}$ | **99.5** | 98.3 | 83.7 | **0.10** | 0.23 | 127.51 |
| | $N_{3,5}$ | **99.5** | 32.4 | 84.2 | **0.09** | 0.38 | 2441.65 |
| | $N_{3,6}$ | ? | 79.5 | **81.8** | ? | 0.21 | 2410.51 |
| | $N_{3,7}$ | **99.7** | 96.8 | $\times$ | **0.11** | 0.30 | $\times$ |
| | $N_{3,8}$ | **99.7** | $\times$ | 87.9 | **0.09** | $\times$ | 1024.48 |
| $\varphi_2$ | $N_{3,9}$ | 🔔 | **88.4** | $\times$ | 🔔 | **0.48** | $\times$ |
| | $N_{4,1}$ | **99.8** | 99.5 | 87.7 | **0.11** | 0.14 | 1564.80 |
| | $N_{4,3}$ | **99.4** | 98.7 | 9.6 | **0.13** | 0.22 | 1.96 |
| | $N_{4,4}$ | **99.5** | 98.7 | 10.4 | **0.10** | 0.20 | 1.03 |
| | $N_{4,5}$ | **99.4** | 98.5 | 12.1 | **0.08** | 0.18 | 2.22 |
| | $N_{4,6}$ | **99.6** | 96.5 | 10.2 | **0.07** | 0.35 | 2.68 |
| | $N_{4,7}$ | **98.3** | 62.0 | 88.9 | **0.14** | 0.57 | 33.47 |
| | $N_{4,8}$ | **99.1** | 95.5 | $\times$ | **0.16** | 0.27 | $\times$ |
| | $N_{4,9}$ | **99.5** | 90.9 | 88.9 | **0.06** | 0.29 | 2438.07 |
| | $N_{5,1}$ | **99.5** | 98.5 | 87.5 | **0.11** | 0.46 | 11.87 |
| | $N_{5,2}$ | **99.7** | 98.9 | 87.6 | **0.10** | 0.25 | 845.28 |
| | $N_{5,4}$ | **99.6** | 99.2 | 87.8 | **0.09** | 0.13 | 1064.02 |
| | $N_{5,5}$ | 🔔 | **99.3** | 10.8 | 🔔 | **0.15** | 1.12 |
| | $N_{5,6}$ | **99.5** | 98.9 | 89.6 | **0.12** | 0.16 | 526.73 |
| | $N_{5,7}$ | **98.1** | 90.8 | $\times$ | **0.16** | 0.30 | $\times$ |
| | $N_{5,8}$ | **99.4** | 15.0 | 87.7 | **0.11** | 0.97 | 1871.85 |
| | $N_{5,9}$ | **98.1** | 80.6 | 87.9 | **0.13** | 0.42 | 1977.43 |
| $\varphi_7$ | $N_{1,9}$ | ? | $\times$ | **0.04** | ? | $\times$ | **400.22** |
| $\varphi_8$ | $N_{2,9}$ | 🔔 | 🔔 | **88.6** | 🔔 | 🔔 | **32.86** |
| $\Phi_1$ | $N_{2,9}$ | 🔔 | ? | $\times$ | 🔔 | ? | $\times$ |
| #successes | | 28 | **32** | 29 | | | |
| median | | **99.5** | 96.8 | 84.2 | **0.11** | 0.26 | 526.73 |

Table 3: Repair results of the ACAS Xu DNNs. SpecAttack (this work), DL2, and neuron fixation. 🔔 indicates a timeout, '?' marks cases where the verifier returned UNKNOWN, and $\times$ indicates unsuccessful repairs.

should or should not have the minimal/maximal score. For future work, extending the satisfaction function to include convex polyhedra would enable SpecAttack to perform attacks and repairs for general linear properties.

One limitation of our repair approach is that for image classification tasks, it does not always return with a successfully repaired network. This is because it relies on a verifier to provide a formal guarantee of a successful repair, which is computationally expensive, but crucial for safety specification compliance. One explanation is that other penalties are better suited for convolutional neural networks. Future research may seek the combination of SpecAttack with different penalty functions during re-training to gain insights into the quality of repair results when applied to different network architectures.

## Conclusion

We presented SpecAttack, an efficient technique for generating counter-examples and repairing neural networks such that they comply with formal safety specifications. SpecAttack supports expressive logical properties and translates these into an objective function, which becomes negative for all network inputs that violate the safety property. Our attack detects counter-examples using a global optimization method which we enhance by increasing sampling density in unrobust regions of the neural network. The proposed repair technique utilizes these counter-examples to make the networks safe via penalized re-training, giving a safety guarantee for the resulting networks using a verifier. Experimental results demonstrate that SpecAttack can be used effectively for both counter-example generation and repairing neural networks, generating useful counter-examples, achieving a high-quality of repair, and demonstrating competitive performance compared to existing approaches.

## References

Botoeva, E.; Kouvaros, P.; Kronqvist, J.; Lomuscio, A.; and Misener, R. 2020. Efficient Verification of ReLU-Based Neural Networks via Dependency Analysis. In *AAAI*, 3291–3299. AAAI Press.

Brendel, W.; Rauber, J.; and Bethge, M. 2018. Decision-

Based Adversarial Attacks: Reliable Attacks Against Black-Box Machine Learning Models. In *ICLR*.

Carlini, N.; and Wagner, D. A. 2017. Towards Evaluating the Robustness of Neural Networks. In *IEEE Symposium on Security and Privacy*, 39–57. IEEE Computer Society.

Chen, J.; Jordan, M. I.; and Wainwright, M. J. 2020. HopSkipJumpAttack: A Query-Efficient Decision-Based Attack. In *IEEE Symposium on Security and Privacy*, 1277–1294. IEEE.

Dong, G.; Sun, J.; Wang, J.; Wang, X.; and Dai, T. 2020. Towards Repairing Neural Networks Correctly. *CoRR*, abs/2012.01872.

Endres, S. C.; Sandrock, C.; and Focke, W. W. 2018. A simplicial homology algorithm for Lipschitz optimisation. *J. Glob. Optim.*, 72(2): 181–217.

Fischer, M.; Balunovic, M.; Drachsler-Cohen, D.; Gehr, T.; Zhang, C.; and Vechev, M. T. 2019. DL2: Training and Querying Neural Networks with Logic. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, 1931–1941. PMLR.

Goldberger, B.; Katz, G.; Adi, Y.; and Keshet, J. 2020. Minimal Modifications of Deep Neural Networks using Verification. In *LPAR*, volume 73 of *EPiC Series in Computing*, 260–278. EasyChair.

Goodfellow, I. J.; Shlens, J.; and Szegedy, C. 2015. Explaining and Harnessing Adversarial Examples. In *ICLR (Poster)*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Identity Mappings in Deep Residual Networks. In *ECCV (4)*, volume 9908 of *Lecture Notes in Computer Science*, 630–645. Springer.

Huang, X.; Kwiatkowska, M.; Wang, S.; and Wu, M. 2017. Safety Verification of Deep Neural Networks. In *CAV (1)*, volume 10426 of *Lecture Notes in Computer Science*, 3–29. Springer.

Ilyas, A.; Engstrom, L.; Athalye, A.; and Lin, J. 2018. Black-box Adversarial Attacks with Limited Queries and Information. In *ICML*, volume 80 of *PMLR*, 2142–2151.

Julian, K. D.; Kochenderfer, M. J.; and Owen, M. P. 2018. Deep Neural Network Compression for Aircraft Collision Avoidance Systems. *CoRR*.

Katz, G.; Barrett, C. W.; Dill, D. L.; Julian, K.; and Kochenderfer, M. J. 2017. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *CAV*, volume 10426 of *LNCS*, 97–117. Springer.

Katz, G.; Huang, D. A.; Ibeling, D.; Julian, K.; Lazarus, C.; Lim, R.; Shah, P.; Thakoor, S.; Wu, H.; Zeljic, A.; Dill, D. L.; Kochenderfer, M. J.; and Barrett, C. W. 2019. The Marabou Framework for Verification and Analysis of Deep Neural Networks. In *CAV (1)*, volume 11561 of *Lecture Notes in Computer Science*, 443–452. Springer.

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

LeCun, Y.; Boser, B. E.; Denker, J. S.; Henderson, D.; Howard, R. E.; Hubbard, W. E.; and Jackel, L. D. 1989. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Comput.*, 1(4): 541–551.

LeCun, Y.; and Cortes, C. 2010. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/.

Madry, A.; Makelov, A.; Schmidt, L.; Tsipras, D.; and Vladu, A. 2018. Towards Deep Learning Models Resistant to Adversarial Attacks. In *ICLR (Poster)*. OpenReview.net.

Mirman, M.; Gehr, T.; and Vechev, M. T. 2018. Differentiable Abstract Interpretation for Provably Robust Neural Networks. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, 3575–3583. PMLR.

Moon, S.; An, G.; and Song, H. O. 2019. Parsimonious Black-Box Adversarial Attacks via Efficient Combinatorial Optimization. In *ICML*, volume 97 of *PMLR*, 4636–4645.

Olson, B. S.; Hashmi, I.; Molloy, K.; and Shehu, A. 2012. Basin Hopping as a General and Versatile Optimization Framework for the Characterization of Biological Macromolecules. *Adv. Artif. Intell.*, 2012: 674832:1–674832:19.

Price, K. V. 2013. Differential Evolution. In *Handbook of Optimization - From Classical to Modern Approach*, volume 38, 187–214. Springer.

Singh, G.; Gehr, T.; Püschel, M.; and Vechev, M. T. 2019. An abstract domain for certifying neural networks. *POPL*, 3: 41:1–41:30.

Sinitsin, A.; Plokhotnyuk, V.; Pyrkin, D.; Popov, S.; and Babenko, A. 2020. Editable Neural Networks. In *ICLR*. OpenReview.net.

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

Sotoudeh, M.; and Thakur, A. V. 2019. Computing Linear Restrictions of Neural Networks. In *NeurIPS*, 14132–14143.

Storn, R.; and Price, K. V. 1997. Differential Evolution - A Simple and Efficient Heuristic for global Optimization over Continuous Spaces. *J. Glob. Optim.*, 11(4): 341–359.

Xiang, Y.; and Gong, X. 2000. Efficiency of generalized simulated annealing. *Physical Review E*, 62(3).