Reducing DNN Properties to Enable Falsification with Adversarial Attacks

David Shriver
Department of Computer Science
University of Virginia
Charlottesville, Virginia, USA
dls2fc@virginia.edu

Sebastian Elbaum
Department of Computer Science
University of Virginia
Charlottesville, Virginia, USA
selbaum@virginia.edu

Matthew B. Dwyer
Department of Computer Science
University of Virginia
Charlottesville, Virginia, USA
matthewbdwyer@virginia.edu

Abstract—**Deep Neural Networks (DNN) are increasingly being deployed in safety-critical domains, from autonomous vehicles to medical devices, where the consequences of errors demand techniques that can provide stronger guarantees about behavior than just high test accuracy. This paper explores broadening the application of existing adversarial attack techniques for the falsification of DNN safety properties. We contend and later show that such attacks provide a powerful repertoire of scalable algorithms for property falsification. To enable the broad application of falsification, we introduce a semantics-preserving reduction of multiple safety property types, which subsume prior work, into a set of equivalid correctness problems amenable to adversarial attacks. We evaluate our reduction approach as an enabler of falsification on a range of DNN correctness problems and show its cost-effectiveness and scalability.**

Index Terms—**falsification, formal methods, neural nets**

I. Introduction

As the performance and applicability of Deep Neural Networks (DNN) continues to increase, their deployment has been explored for use in safety-critical domains, such as autonomous vehicles [9], [15], [35] and medicine [17], [23], [56]. As a result, checking the correctness of DNNs has emerged as a key challenge to ensuring the safety of these systems. For example, for the DroNet DNN that predicts a steering angle and a probability of collision for an autonomous quadrotor system [35], one such correctness property may specify that if the probability of collision is low, then the steering angle should not be extremely large.

One approach to checking correctness of DNN models is verification. In the past 3 years alone, dozens of DNN verification techniques have been introduced for proving properties of DNN models [7], [11], [13], [19]–[21], [24], [28]–[30], [43], [44], [46], [47], [51], [54], [55], [57]–[59]. Unfortunately, these approaches are often limited in their applicability, due to their simplifying assumptions about DNN structure or to high computational cost, making it difficult to apply them to real models [45], [61]. For example, due to the large size and complex computation graph of DroNet, existing verifiers fail to run or cannot finish in a reasonable time.

A complementary approach to verification is that of falsification, which attempts to find violations of a property. While verifiers can show that a property is true, falsifiers can often find violations more quickly than verifiers when the property is false. Falsification is an active area of research spanning property types and application domains [1], [4], [16], [18]. In the context of DNNs, one can view adversarial attacks [26], [32], [36], [37], [49], [50] as falsifiers for DNN local robustness properties. While these techniques often scale to real-world DNNs, they are currently limited in the range of properties they can falsify. For the DroNet example, adversarial attacks support the scale and complex structure of the DNN, but are designed only to find violations of robustness properties and not safety properties like those relating speed and probability of collision. As we discuss in §II-A, there is a broad range of properties that could benefit from the speed and applicability of these falsifiers.

Driven in part by the cost of DNN verification, and in part by the limited property support for DNN falsification, we identify the key insight that many valuable property types could be reduced into the more commonly supported form of local robustness properties. We build on this insight to develop an approach for reducing properties in an expressive, general form to an equivalid set of local robustness properties, which have wide support among both falsifiers and verifiers. Such translation has the potential to bring existing techniques to bear on falsifying DNN properties, leaving verifiers to focus on proving that a property holds.

Our community has exploited property reduction for verification and program analysis for decades. Perhaps best known is the reduction from stateful safety properties to reachability properties. For example, partial order reductions were broadened in applicability by reducing stateful properties to a form of deadlock [25], and both data flow analyses [40], [48] and SAT solving [8] were applied to verify stateful properties by formulating the reachability of error states. A second use of reductions is to enable more efficient algorithmic methods to be employed. For instance, the `-safety` option of the SPIN model checker permits it to use a significantly faster reachability algorithm [27]. Such reductions are now considered standard in verification and program analysis. In the new domain of DNN verification and falsification, however, the lessons of such reductions have not yet taken root.

In this paper, we introduce an approach for reducing a DNN and an associated safety property – which we refer to as a correctness problem – into an equivalid set of correctness

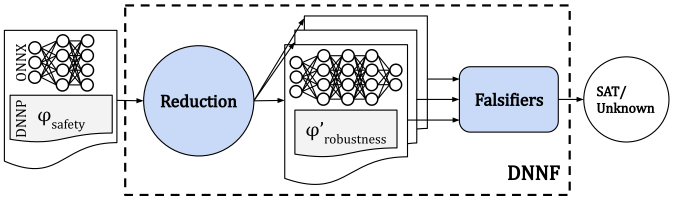


Fig. 1: Proposed approach reduces a DNN and its safety property into an equivalid set of correctness problems formulated with robustness properties that can be processed by falsifiers.

problems formulated with robustness properties that can be processed by existing adversarial techniques. Figure 1 provides an overview of the approach. By preserving validity, the translation supports both falsification approaches, such as adversarial attack algorithms, and existing verification techniques. The approach is fully automated which allows developers to specify properties in a convenient form while leveraging the complementary strengths of falsification and verification algorithms.

The primary contributions of this work are: (1) an automated approach for the reduction of DNN correctness problems as an equivalid set of robustness problems; (2) an implementation of our approach that employs a portfolio of falsifiers; and (3) a study demonstrating that property reduction yields cost-effective violations of general DNN correctness problems.

II. Background

This section presents prior work on DNN property specification and approaches for their falsification.

A. Properties of DNNs

Given a DNN, $\mathcal{N} : \mathbb{R}^n \to \mathbb{R}^m$, a property, $\phi(\mathcal{N})$, defines a set of constraints over the inputs, $\phi_\mathcal{X}$ – the pre-condition, and a set of constraints over the outputs, $\phi_\mathcal{Y}$ – the post-condition. Checking property $\phi(\mathcal{N})$ attempts to prove or falsify: $\forall x \in \mathbb{R}^n : \phi_\mathcal{X}(x) \to \phi_\mathcal{Y}(\mathcal{N}(x))$.

A survey on verification of neural networks [33] identifies different representations for the input and output constraints used by verification techniques; two of these representations are particularly useful in this work. A *hyperrectangle* is an n-dimensional rectangle where constraints are formulated as $(x_i \geq lb_i) \wedge (x_i \leq ub_i)$, where $lb_i, ub_i \in \mathbb{R}$ and $0 \leq i < n$ define the lower and upper bounds on the value of each dimension of x, respectively. A special case of hyperrectangles used in our approach is the *unit hypercube* which is a hyperrectangle where $\forall i.(lb_i = 0) \wedge (ub_i = 1)$; an n-dimensional hypercube is denoted $[0, 1]^n$. A *halfspace-polytope* is a polytope which can be represented as a set of linear inequality constraints, $Ax \leq b$, where $A \in \mathbb{R}^{k\times n}$, $b \in \mathbb{R}^k$, k is the number of constraints and n is the dimension of x.

Using such encodings, researchers have specified a range of desirable properties of DNNs. Here we distinguish three broad categories: robustness, reachability, and differential properties.

Robustness properties originated with the study of adversarial examples [50], [62]. Robustness properties apply to classification models and specify that inputs from a specific region of the input space must all produce the same output class. Robustness properties can be further classified as either local or global robustness; the former asserts robustness in a local region of the input domain and the latter over the entire input domain. Detecting violations of robustness properties has been widely studied, and they are a common type of property for evaluating verifiers [21], [46], [47], [51], [54].

Reachability properties define the post-condition using constraints over output values rather than output classes, and are thus not limited to classification models. Such properties have been used to evaluate DNN verifiers [29], [54]. Reachability properties specify that inputs from a given region of the input space must produce outputs that lie in a given region of the output space. For example, a DNN model controlling the velocity of an autonomous vehicle may have a safety property specifying that the model never produces a desired velocity value greater than the vehicles maximum physical speed for any input in the input domain. Similarly to robustness, reachability properties can be further classified as local or global. For example, a global halfspace-polytope reachability (GHPR) property would specify a halfspace-polytope constraint on network output values that must hold for all inputs.

Differential properties are the most recently introduced DNN property type [42]. These properties specify a difference (or lack thereof) between outputs of multiple DNNs. One type of differential property is equivalence, which states that for every input, two DNN models produce the same output. Such a property can be used to check that DNN semantics are preserved after some modification, such as quantization or pruning. Differential properties can be supported by combining multiple DNNs into a single network and expressing properties over their combined input and output domains.

In addition to these three categories and as alluded earlier, properties can also be classified by the form of their input pre-condition. *Global* properties have the most permissive pre-condition, enforcing the post-condition for any input in the input domain of the DNN. For example, a DNN that operates on images may accept values in $[0, 1]^n$. The pre-condition of a global property would not restrict this domain any further. *Local* properties only enforce the post-condition for inputs within a designated region of the input domain. For example, a local property for an image processing network may have the precondition that inputs are within distance ε of some given input x. This is especially common in robustness properties.

B. Adversarial Attacks and Fuzzing

One approach to checking properties of DNNs is through the use of algorithms that seek to find examples that violate a given specification for a given model. Two categories of techniques have been developed for DNNs that can be used to falsify DNN property specifications.

Adversarial attacks are methods that are optimized to detect violations of robustness properties [2], [62]. In general, adversarial attacks take in a DNN model and an initial input, and attempt to produce a perturbation that, when applied to the input, will change the class predicted by the given model.

These perturbations are often also subject to some constraints, such as remaining within a given distance of some original input. A perturbed input, commonly known as an adversarial example is a violation to a local robustness property. To our knowledge, adversarial attacks are a method of falsification that only supports the falsification of robustness properties. Adversarial attacks can be classified based on characteristics of the attack, such as if they are white-box [26], [32], [36], [37], [50] or black-box [49]; targeted [26], [50] or untargeted [32], [37]; iterative [32], [36], [37] or one-shot [26], [50]; or by their perturbation constraint (e.g., L_0 [49], L_2 [14], or L_∞ [26], [50]). A more exhaustive taxonomy and description of existing adversarial attacks is available in the literature [2], [62].

Fuzzing involves randomly generating inputs within a given input region (often the full input space), and checking whether the outputs they produce violate a specified post-condition. Fuzzing is more general than adversarial attacks, in that it can support the falsification of more than robustness properties, but requires specifying input mutation functions and objective functions (essentially an output oracle), for every type of property that needs support. Examples of existing fuzzing techniques include TensorFuzz [39] and DeepHunter [60].

III. Approach

The primary goal of our approach is to amplify the power of falsifiers, such as adversarial attacks, by increasing their applicability. Our approach takes in a correctness problem comprised of a DNN and a property, and encodes it as an equivalid set of robustness problems, which then enables us to run a portfolio of methods that are applicable to this restricted problem class to uncover general property violations.

A. Defining Property Reduction

A *correctness problem* is a pair, $\psi = \langle \mathcal{N}, \phi \rangle$, of a DNN, $\mathcal{N}$, and a property specification ϕ, formed to determine whether $\mathcal{N} \models \phi$ is *valid* or *invalid*.

Reduction, $reduce : \Psi \to P(\Psi)$, aims to transform a correctness problem, $\langle \mathcal{N}, \phi \rangle = \psi \in \Psi$, to an equivalid form, $reduce(\psi) = \{\langle \mathcal{N}_1, \phi_1 \rangle, \ldots, \langle \mathcal{N}_k, \phi_k \rangle\}$, in which property specifications define robustness properties:

$$\phi_i = \forall x. x \in [0,1]^n \to \mathcal{N}_i(x)_0 > \mathcal{N}_i(x)_1$$

and networks have input domains defined as unit hypercubes, and output domains consist of two values – indicating property satisfaction and falsification:

$$\forall \langle \mathcal{N}_i, \phi_i \rangle \in reduce(\psi). \mathcal{N}_i : [0,1]^n \to \mathbb{R}^2$$

As we demonstrate in §IV, reduction enables the application of a broad array of efficient DNN analysis techniques to compute problem validity and/or invalidity.

As defined, reduction has two key properties. The first property is that the set of resulting problems is equivalid with the original correctness problem (a proof of this theorem is included in Appendix A).

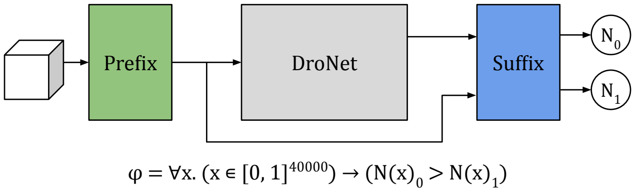


Fig. 2: One of the robustness problems generated by reduction.

Theorem 1. *Reduction maps an arbitrary correctness problem to an equivalid set of correctness problems.*

$$\mathcal{N} \models \psi \Leftrightarrow \forall \langle \mathcal{N}_i, \phi_i \rangle \in reduce(\psi). \mathcal{N}_i \models \phi_i$$

The second property is that the resulting set of problems all use the same property type, i.e., robustness; they all assert that $\mathcal{N}(x)_0$ is the output class for all inputs. Applying reduction enables verifiers or falsifiers to support a large set of correctness problems by implementing support for this single property type. We chose to reduce to robustness properties due to their broad support among existing falsifiers and verifiers.

B. Overview

To illustrate, consider a property for DroNet [35]; a DNN for controlling an autonomous quadrotor. Inputs to this network are 200 by 200 pixel grayscale images with pixel values between 0 and 1. For each image, DroNet predicts a steering angle and a probability that the drone is about to collide with an object. The property states that for all inputs, if the probability of collision is no greater than 0.1, then the steering angle is capped at ±5 degrees and is specified as:

$$\forall x.((x \in [0,1]^{40000}) \wedge (\mathcal{N}(x)_{Pcoll} \leq 0.1)) \to (-5^\circ \leq \mathcal{N}(x)_{Steer} \leq 5^\circ)$$

Adversarial attacks cannot be used off the shelf to falsify this property, since it is not a robustness property.

To enable the application of adversarial attacks, we reduce the property to a set of correctness problems with robustness properties, such as the one shown in Figure 2. This particular example is reduced to two correctness problems with robustness properties. Each of the problems pair a robustness property (shown in the bottom of Figure 2) with a modified version of the original DNN. The new DNN is created through two key transformations. First, incorporating a prefix network (shown in green in Figure 2) to reduce the input domain to a unit-hypercube. This modification ensures that the properties for reduced problems can all use the same pre-condition. Second, incorporating a suffix network (shown in blue in Figure 2) that takes in the inputs and outputs of the original DNN and classifies whether they constitute a violation of the original property. This suffix transforms the network into a classifier for which violations of a robustness property correspond to violations of the original property.

C. Reduction Transformation

We rely on three assumptions to transform a correctness problem into a reduced form. First, the constraints on the network inputs must be represented as a union of convex polytopes. Second, the constraints on the outputs of the

Algorithm 1: Property Reduction

```
Input: Correctness problem $\langle \mathcal{N}, \phi \rangle$
Output: A set of robustenss problems
        $\{\langle \mathcal{N}_1, \phi_1 \rangle, ..., \langle \mathcal{N}_i, \phi_i \rangle\}$
1  begin
2      $\phi' \leftarrow DNF(\neg\phi)$
3      $\Psi \leftarrow \emptyset$
4      for $disjunct \in \phi'$ do
5          $hpoly \leftarrow$ disjunct_to_hpolytope($disjunct$)
6          $prefix \leftarrow$ construct_prefix($hpoly$)
7          $\mathcal{N}' \leftarrow \mathcal{N}' : x \mapsto \text{concat}(\mathcal{N}(x), x)$
8          $suffix \leftarrow$ construct_suffix($hpoly$)
9          $\mathcal{N}'' \leftarrow suffix \circ \mathcal{N}' \circ prefix$
10         $\phi' \leftarrow \forall x.(x \in [0,1]^n \implies \mathcal{N}''(x)_0 > \mathcal{N}''(x)_1)$
11         $\Psi \leftarrow \Psi \cup \langle \mathcal{N}'', \phi' \rangle$
12     return $\Psi$
```

Algorithm 2: disjunct_to_hpolytope

```
Input: Conjunction of linear inequalities $\phi_i$
Output: Halfspace polytope $H$
1  begin
2      $H \leftarrow (A, b)$ where $A$ is an $(|\phi_i|) \times (m+n)$ matrix where
         columns 0 to $m-1$ correspond to output variables $N(x)_0$ to
         $N(x)_{m-1}$ and columns $m$ to $m+n-1$ correspond to input
         variables $x_0$ to $x_{n-1}$
3      for $ineq_j \in \phi_i$ do
4          if $ineq_j$ uses $\geq$ then
5              swap lhs and rhs; switch inequality to $\leq$
6          else if $ineq_j$ uses $>$ then
7              swap lhs and rhs; switch inequality to $<$
8          move variables to lhs; move constants to rhs
9          if $ineq_j$ uses $<$ then
10             decrement rhs; switch inequality to $\leq$
11         $A_j \leftarrow$ coefficients of variables on lhs
12         $b_j \leftarrow$ rhs constant
13     return $H$
```

Algorithm 3: construct_prefix

```
Input: Halfspace polytope $H$
Output: A fully-connected layer $P$
1  begin
2      $lb = (-\infty, ..., -\infty)$
3      $ub = (\infty, ..., \infty)$
4      for $constraint \in H$ do
5          if constraint is over only input variables then
6              for $x_i \in x$ do
7                  $lb_i \leftarrow \max\{\min_{x_i} constraint, lb_i\}$
8                  $ub_i \leftarrow \min\{\max_{x_i} constraint, ub_i\}$
9      $W \leftarrow \text{diag}(ub - lb)$
10     $b \leftarrow lb$
11     $P \leftarrow \text{FullyConnectedLayer}(W, b)$
12     return $P$
```

Algorithm 4: construct_suffix

```
Input: Halfspace polytope $H = (A, b)$
Output: A DNN with 2 fully connected layers $S$
1  begin
2      $S_h \leftarrow \text{ReLU}(\text{FullyConnectedLayer}(A, -b))$
3      $W \leftarrow \begin{bmatrix} 1 & 1 & ... & 1 \\ 0 & 0 & ... & 0 \end{bmatrix}$
4      $S_o \leftarrow \text{FullyConnectedLayer}(W, \vec{0})$
5      $S \leftarrow S_o \circ S_h$
6      return $S$
```

network must be represented as a union of convex polytopes. Third, we assume that each convex polytope is represented as a conjunction of linear inequalities. Complying with these assumptions still enables properties to retain a high degree of expressiveness as unions of polytopes are extremely general and subsume other geometric representations, such as intervals and zonotopes. §IV-A shows that these assumptions are sufficient to support existing DNN correctness problems.

Algorithm 1 defines the reduction transformation at a high level. We present each step of the algorithm and describe their application to the DroNet example described above.

1) Reformat the Property: Reduction first negates the original property specification and converts it to disjunctive normal form (DNF) – line 2. Negating the specification means that a satisfying model falsifies the original property. The DNF representation allows us to construct a property for each disjunct, such that if any are violated, the negated specification is satisfied and thus the original specification is falsified. For each of these disjuncts the approach defines a new robustness problem, as described below.

2) Transform into halfspace-polytopes: Constraints in each disjunct are converted to halfspace-polytope constraints, defined over the concatenation of the input and output domains – $disjunct\_to\_hpolytope()$ on line 5. This conversion is described in Algorithm 2. A halfspace-polytope can be represented in the form $Ax \leq b$, where A is a matrix of k rows, where each row represents 1 constraint, and d columns, one for each variable. In this case, d is equal to $m+n$, the size of the output space, plus the size of the input space. This representation facilitates the transformation of constraints into network operations. To build the matrix A and vector b, we first transform all inequalities in the conjunction to $\leq$ inequalities with variables on the left-hand-side and constants on the right-hand-side. The transformation first converts $\geq$ to $\leq$ and $>$ to $<$ – lines 4-7 of Algorithm 2. Then, all variables are moved to the left-hand-side and all constants to the right-hand-side – line 8. Next, $<$ constraints are converted to $\leq$ constraints by decrementing the constant value on the right-hand-side – lines 9-10. This transformation assumes that there exists a representable number with greatest possible value that is less than the right-hand-side. Finally, each inequality is converted to a row of A and value in b – lines 11-12.

3) Prefix Construction: Using the constructed halfspace-polytope, Algorithm 1 next constructs a prefix to the original network to ensure the input domain of the resulting network is $[0,1]^n$, where n is the input dimensionality of the original network – $construct\_prefix()$ on line 6. The algorithm to construct the prefix is shown in Algorithm 3. The prefix is constructed by first extracting lower and upper bounds for every input variable – lines 2-8. This extracts the minimal axis-aligned bounding hyperrectangle. The lower and upper bounds can then be used to construct the prefix network, which is a single n-dimensional fully-connected layer, with no activation function, which has a diagonal weight matrix with values equal to the ranges of the input variables, and biases equal to the lower bounds of each input. The prefix operates on unit hypercubes, reducing the input space to the correctness problems. The prefix also encodes any interval constraints over the original input space, allowing them to be removed before suffix construction, which simplifies the suffix networks. For the DroNet example, the diagonal of this matrix is a vector of ones, while the biases are all 0.

Next, the original input values are forwarded to the end of the original network and concatenated with the original output layer – line 7. Because constraints will be encoded as a network suffix that classifies whether inputs are property violations, this step is necessary to enable the encoding of constraints over the inputs.

4) Suffix Construction: The suffix subnetwork classifies whether inputs satisfy the specification – $construct\_suffix()$ on line 8. The algorithm for constructing the suffix from the halfspace-polytope constraints is shown in Algorithm 4. The constructed suffix has two layers, a hidden fully-connected layer with ReLU activations, and dimension equal to the number of constraints in the halfspace-polytope defined by the current disjunct, and a final output layer of size 2.

The hidden layer of the suffix has a weight matrix equal to the constraint matrix, A, of the halfspace-polytope representation, and a bias equal to $-b$ – line 2. With this construction, each neuron will only have a value greater than 0 if the corresponding constraint is not satisfied, otherwise it will have a value less than or equal to 0, which becomes equal to 0 after the ReLU activation is applied. In the DroNet problem for example, one of the constraints for a disjunct is $(\mathcal{N}(x)_S \leq -5°)$. For this conjunct we define the weights for one of the neurons to have a weight of 1 from $\mathcal{N}(x)_S$, a weight of 0 from $\mathcal{N}(x)_P$, and a bias of $5°$.

The output layer of the suffix has 2 neurons, each with no activation function. The first of these neurons is the sum of all neurons in the previous layer, and has a bias value of 0. Because the neurons in the previous layer each represent a constraint, and each of these neurons is 0 only when the constraint is satisfied, if the sum of all these neurons is 0, then the conjunction of the constraints is satisfied, indicating that a violation has been found. The second of these neurons has a constant value of 0 – all incoming weights and bias are 0. The resulting network will predict class 1 if the input satisfies the corresponding disjunct and class 0 otherwise.

5) Correctness Problem Construction: Lines 9-11 of Algorithm 1 define the reduced subproblem comprised of the network that we have constructed and a robustness property. The robustness property specification is always the same and states that the network should classify all inputs in the d-dimensional hypercube as class 0 – no violations. If a violation is found to this property, then, according to Theorem 2, the original property is violated by the unreduced input that violated the robustness property. In the end, we have generated a set of correctness problems such that, if any of the problems is violated, then the original problem is also violated. This comes from our construction of a property for each disjunct in the DNF of the negation of the original property.

D. Properties Over Multiple Networks

While Algorithm 1 is defined over properties with a single network, it can easily be applied to properties over multiple networks, by combining those networks into a single large network. This is specially relevant to check for equivalence properties. This can be done by concatenating their input and output vectors. This results in a single large network with a computation path for each network. The transformation algorithm can then be applied as before.

E. Implementation.

We implemented our approach in a system named DNNF[1], which accepts a DNN property specification and corresponding DNN as input, and returns whether a violation is found. Whereas the reduction algorithm in §III applies to properties with unions of polytopes as input constraints, the current implementation works on unions of hyperrectangles in the input space. This was a convenience choice to simplify the implementation while still accommodating most properties in the verification literature, as demonstrated in §IV-A.

IV. Empirical Evaluation

We now assess the cost-effectiveness of reducing DNN properties for falsification by applying it to a range of DNN property benchmarks that provide diversity in terms of property types and DNN complexity. Our evaluation will attempt to answer the following research questions:

- RQ1: How expressive are the properties supported by property reduction?
- RQ2: How cost-effective is falsification at finding property violations?
- RQ3: How scalable is falsification?

A. RQ1: On the Expressiveness of Reduction

We first evaluate whether the assumptions about the property specification required by reduction, namely that the original property is specified as a logical formula of linear inequalities, is expressive enough to support DNN correctness properties that have been proposed in existing work.

1) Setup: To evaluate the expressiveness of properties supported by our reduction, we analyze and catalog the benchmarks used by the five verifiers used in our later study, as well as the benchmarks of a recent DNN verifier competition, VNN-Comp [34]. Additionally, we surveyed published papers on DNN verification in the past two years identifying 4 additional works [6], [22], [52], [53]. Finally, we include the 2 new benchmarks introduced in this work.

2) Results: We summarize the results in Table I, which lists the benchmarks used in each work, the type and number of properties in the benchmark and whether the properties are supported by Algorithm 1 and our current implementation. The property types use abbreviated names with the following encoding: the first symbol indicates whether the property is global (G) or local (L); the second symbol indicates whether

[1] https://github.com/dlshriver/DNNF

the input constraint can be represented as a hyper-rectangle (□) or not (☒); the third symbol indicates whether the property is a robustness (r) property, a reachability (R) property, or a differential (D) property. Each cell under a property type indicates the number of properties in the corresponding benchmark of that type. The bolded benchmarks are used later in the study for the evaluation of RQ2 and RQ3. We describe the details of these benchmarks in more detail below.

The first benchmark is ACAS Xu, introduced for the study of the Reluplex verifier [29], and used extensively since [6], [12], [13], [30], [34], [54], [55]. The benchmark consists of 10 properties. Properties ϕ_1, ϕ_2, ϕ_3, ϕ_4, ϕ_7 and ϕ_8 are reachability properties, while ϕ_5, ϕ_6, ϕ_9, and ϕ_{10} are traditional class robustness properties. All 10 properties have hyper-rectangles constraints over the inputs and are fully supported by our property reduction.

The next benchmark is from the evaluation of the Planet verifier. First is the Collision Avoidance benchmark, which consists of 500 safety properties that check the robustness of a network that classifies whether 2 simulated vehicles will collide, given their current state. All 500 properties are L□r properties, and are all fully supported. Second is a set of 7 properties on an MNIST network. The first 4 of these are G□R properties, while the next 2 are L□r properties, and the final property is an L☒r property. In addition to restricting the amount of noise that can be added to each pixel in the input image, the final property constrains the difference in the noise between neighboring pixels. DNNF currently supports 6 of these properties, while the final is supported by Algorithm 1.

The Neurify verifier was evaluated on the ACAS Xu benchmark and on properties of 4 MNIST networks, 3 android app malware classification networks, and 1 self-driving car network. The evaluation on MNIST used 500 L□r properties across 4 networks, all of which we support. Neurify was also evaluated on 3 networks trained on the Drebin dataset [5] to classify apps as benign or malicious. This benchmark also includes 500 L□r properties, which are fully supported. Finally, Neurify was evaluated on local reachability properties for a modified version of the DAVE self-driving car network [10]. This benchmark consists of 200 local reachability properties, with 4 different types of input constraints (50 properties of each type). The first type of input constraint is an L_∞ constraint, which is equivalent to a hyper-rectangle constraint. The second type of input constraint is an L_1 constraint, which can be written as a halfspace polytope constraint. The third and fourth type of input constraint is an image brightness and contrast, which can also be written as a halfspace polytope constraints. DNNF currently supports the first 50 of these properties and the remainder are supported by Algorithm 1.

The DeepZono abstract domain of the ERAN verifier used in our study [46], was evaluated on 3300 L□r properties applied to a set of 24 MNIST networks and 13 CIFAR10 networks. All of the properties in this benchmark are fully supported by our approach.

The ReluDiff verifier was designed to support differential properties in order to show equivalance between two networks [42]. The verifier was evaluated on L□D properties. Each property was defined over a network, $\mathcal{N}$ and a modified version of the same network with quantized weights, $\mathcal{N}'$. The property checked whether $|\mathcal{N}(x) - \mathcal{N}'(x)| < \epsilon$ held in a local region of the input space. 14 of these properties were verified over networks from the ACAS Xu benchmark [29], 200 properties on networks trained with the MNIST dataset, and 100 properties on a network trained for Human Activity Recognition [3]. All 314 differencing properties are fully supported by our approach.

The recent VNN-Comp competition used 3 benchmarks. The first is a benchmark with properties applied to networks with piecewise linear activation functions. This benchmark consists of the ACAS Xu benchmark [29] with 4 L□r properties and 6 L□R properties, as well as a set of 50 local robustness properties with hyper-rectangle input constraints applied to 3 MNIST networks. All of these properties are supported by our approach. The second is a set of 300 local robustness properties with hyper-rectangle input constraints applied to convolutional neural networks trained on MNIST and CIFAR10. All of these properties are supported by our approach. The final benchmark is a set of 32 local robustness properties with hyper-rectangle input constraints applied to neural networks with non-linear activation functions ($sigmoid$ and $tanh$) trained on MNIST. All of these properties are supported by our approach.

Several DNN verifiers have been introduced recently. The nnenum verifier [6] and an abstraction-refinement approach for DNN verification [22] were evaluated on the ACAS Xu benchmark. The reachability set representation of ImageStars [52] was evaluated on two benchmarks of local robustness properties applied to MNIST and ImageNet networks. The benchmark on the MNIST networks used a version of 900 local robustness where pixels could be independently darkened, enabling input constraints to be represented as hyper-rectangles. The benchmark on the ImageNet networks uses 6 properties created from an original image and a corresponding adversarial example. The properties specify that for a given region along the line between the original image and adversarial example, all inputs along the segment are classified as the correct class. While the MNIST benchmark is supported by our current reduction implementation, the ImageNet benchmark requires polytope constraints in the input space and is therefore supported just by Algorithm 1. The NNV verifier [53] also introduced a benchmark with an adaptive cruise control (ACC) system. It checks a temporal property not currently supported by Algorithm 1, but we see the potential to support such properties through unrolling in future work.

> Overall, we find that the property specifications accepted by Algorithm 1 are rich enough to express 7 of the 8 property types found in the explored benchmarks.

When considering the listed benchmarks, Algorithm 1 fully supports 16 of the 17 benchmarks. Our current implementation completely supports the properties from 13 of the 17 bench-

TABLE I: Property types of existing benchmarks and their support by reduction. The property type names use the following encoding: the first symbol indicates a global (G) or local (L) property; the second symbol indicates whether the input constraint can be represented as a hyper-rectangle (□) or not (⊠); the third symbol indicates the property class as robustness (r), reachability (R), or differential (D). Bolded benchmarks are used later in the study to evaluate RQ2 and RQ3.

Benchmark	**# of Property of Each Type**								**Support**	
	L□r	**L⊠r**	**G□R**	**L□R**	**L⊠R**	**G□D**	**L□D**	**Other**	**Algorithm 1**	**Implementation**
ACAS Xu [29]	4			6					10	10
Collision Avoidance [21]	500								500	500
Planet-MNIST [21]	2	1	4						7	6
Neurify-MNIST [54]	500								500	500
Neurify-Drebin [54]	500								500	500
Neurify-DAVE [54]				50	150				200	50
ERAN-MNIST [46]	1700								1700	1700
ERAN-CIFAR [46]	1600								1600	1600
ReluDiff ACAS [42]							14		14	14
ReluDiff-MNIST [42]							200		200	200
ReluDiff-HAR [42]							100		100	100
VNN-COMP-CNN [34]	300								300	300
VNN-COMP-PWL [34]	54			6					60	60
VNN-COMP-NLN [34]	32								32	32
ImageStars-MNIST [52]	900								900	900
ImageStars-ImageNet [52]		6							6	0
NNV-ACC [53]								2	0	0
GHPR			20						20	20
CIFAR-EQ						91	200		291	291

marks, and supports a subset of the properties in 2 additional benchmarks. Our results also show that the current space of DNN properties has limited diversity, with most benchmarks consisting primarily of local robustness properties. This points to the value added of the new benchmarks we introduce. It is also expected, as has happened in the verification community in the past, that as verification and falsification techniques improve, developers will want to apply them to reason about a broader range of correctness properties. The proposed algorithm will enable them to do that, even if verifiers and falsifiers do not directly support them.

B. RQ2: On the Cost-Effectiveness of Reduction-Enabled Falsification

To evaluate the cost-effectiveness of falsification enabled by the proposed reduction, we identify a set of falsifiers and verifiers to compare their complementary performance, problem benchmarks, and metrics that constitute the basis for the studies around RQ2 and RQ3.

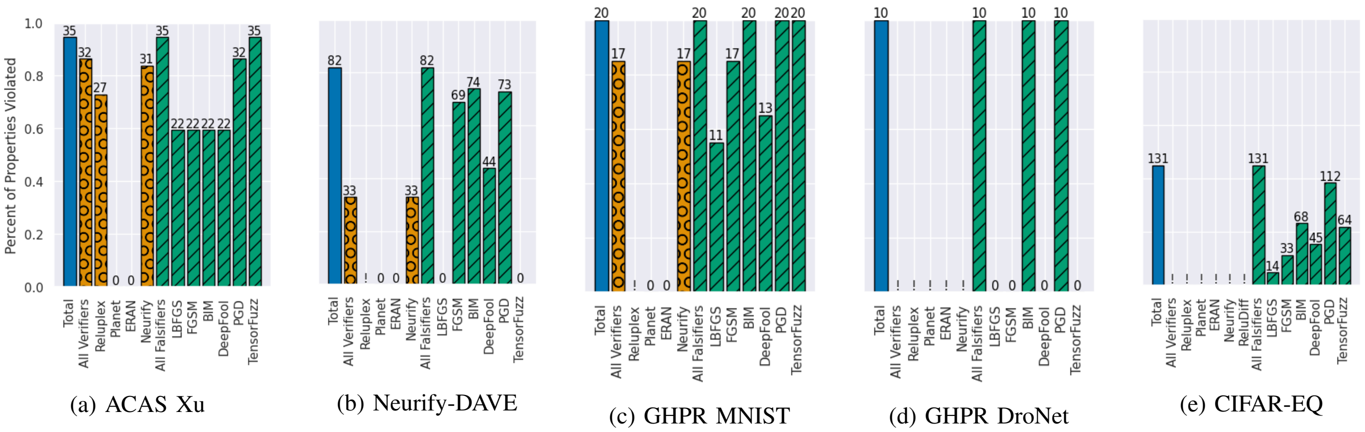
1) Setup: **Falsifiers.** As falsification methods, we will use several common adversarial techniques, as well as a DNN fuzzing tool. For adversarial attacks, we choose a subset of the methods from two surveys [2], [62]. We select the methods common to both surveys with L_∞ input constraints (which matches our implementation) and with implementations available in the cleverhans tool [41]. The chosen adversarial attacks are LBFGS [50], FGSM [26], Basic Iterative Method (BIM) [32], and DeepFool [37]. Of these attacks, none use random initialization, and thus will produce the same result over multiple runs. In order to observe the potential benefits of random initialization, we also include Projected Gradient Descent (PGD) [36], which was only included in one of the surveys. Therefore, we run each attack, except PGD, once, and if no adversarial example is found, we return an *unknown* result. For PGD, if no adversarial example is found, we try again, until one is found, or the given time limit is reached. We use the default parameters for each attack, as specified by cleverhans. For DNN fuzzing, we use TensorFuzz [39] for its easily accessible implementation [38]. TensorFuzz requires the definition of an oracle for recognizing property violations. We provide a version of TensorFuzz with an oracle that identifies violations by checking whether $\mathcal{N}(x)_0 \leq \mathcal{N}(x)_1$[2].

Verifiers. For comparison to verification, we select four verifiers: Reluplex [29], Planet [21], ERAN using the DeepZono abstract domain [46], and Neurify [54]. Neurify and ERAN have been shown to be fastest and most accurate in recent studies [61], and all four verifiers are supported by DNNV, which makes them easy to run and allows us to use a common property specification for all verifiers and falsifiers. For differential properties we also consider ReluDiff [42] since it is currently the only verifier built to handle such properties.

Portfolios. In addition to the individual falsifiers and verifiers, we simulate portfolios of these methods, which run analyses in parallel and return the first result. We use 3 portfolios: *All Falsifiers*, which includes the 6 falsifiers described above; *All Verifiers* which includes all verifiers run on each benchmark; *Total* which includes all methods used in this study. To simulate running each portfolio, we take the union of the violations found by each method in the portfolio, and consider the time to find each violation to be the fastest time among the methods in the portfolio which found that violation.

Problem Benchmarks. We evaluate our approach on two common and representative benchmarks from the verification literature, and two created for this work to provide a range of networks and property types. Our selection criteria was meant to achieve two objectives. First, we wanted to select enough benchmarks to explore all property types with hyper-rectangle input constraints. Second, we wanted to select benchmarks

[2] https://github.com/dlshriver/tensorfuzz



(a) ACAS Xu

(b) Neurify-DAVE

(c) GHPR MNIST

(d) GHPR DroNet

(e) CIFAR-EQ

Fig. 3: The number of violations found by each falsifier and verifier, reduced by the total number of potentially falsifiable properties. The number above each bar gives the total number of violations found. An exclamation point indicates that a verifier could not be run on a property due to the structure of the network.

with networks that varied in both size and structure since these factors have been shown to affect verifier performance [61].

From the verification literature, we select *ACAS Xu*, the most commonly used benchmark, and a slightly modified version of the *Neurify-DAVE* benchmark. For Neurify-DAVE, we select the 50 L□R properties supported by our current implementation, and we augment the benchmark with an additional network. The new network is the original DAVE DNN on which the smaller network in the benchmark was based. While the small DNN has 10277 neurons, the original DAVE network that we add has 82669 neurons, which will allow us to explore the scalability of reduction and falsification. The two networks in this benchmark are convolutional networks and are much larger than the networks in the ACAS Xu benchmark.

We developed 2 new benchmarks to cover property types that are not yet covered by existing benchmarks. The *GHPR* benchmark is a new DNN property benchmark that contains G□R properties applied to several network architectures of varying size and structure. It consists of 30 correctness problems, 20 of which are 10 GHPR properties applied to 2 MNIST networks, and 10 of which are GHPR properties applied to the DroNet DNN described previously. The DroNet DNN is one of the largest in our study, with more than 475,000 neurons. The MNIST properties are of the form: for all inputs, the output values for classes a and b are closer to one another than either is to the output value of class c. The DroNet properties are of the form: for all inputs, if the probability of collision is between p_{min} and p_{max}, then the steering angle is within d degrees of 0. These properties are described in more detail in the supplementary material[3].

The *CIFAR-EQ* benchmark is a new DNN property benchmark with differential properties applied to large networks with complex structures. It contains a mix of both global and local equivalence properties. It is the only benchmark to contain G□D properties, which were absent in the property benchmarks that we found. It consists of 291 properties: 91 global equivalence properties and 200 local equivalence properties. Of the global properties, 1 is untargeted, while the other 90 are targeted equivalence properties. Of the 200 local properties, 20 are untargeted, while the other 180 are targeted equivalence properties. The properties are applied to a pair of neural networks trained on the CIFAR dataset [31]. The first network is a large convolutional network with 62,464 neurons and the second is a ResNet-18 network with over 588,000 neurons. These properties are described in more detail in the supplementary material. Because the verifiers do not support the multiple computation path structure formed during network composition, we do not run them on this benchmark.

Metrics. For each verification and falsification approach, we will measure the number of property violations found and the total time to find each violation. The total time to find a violation includes both the time to transform the property, as well as to run the falsifier on the resulting properties.

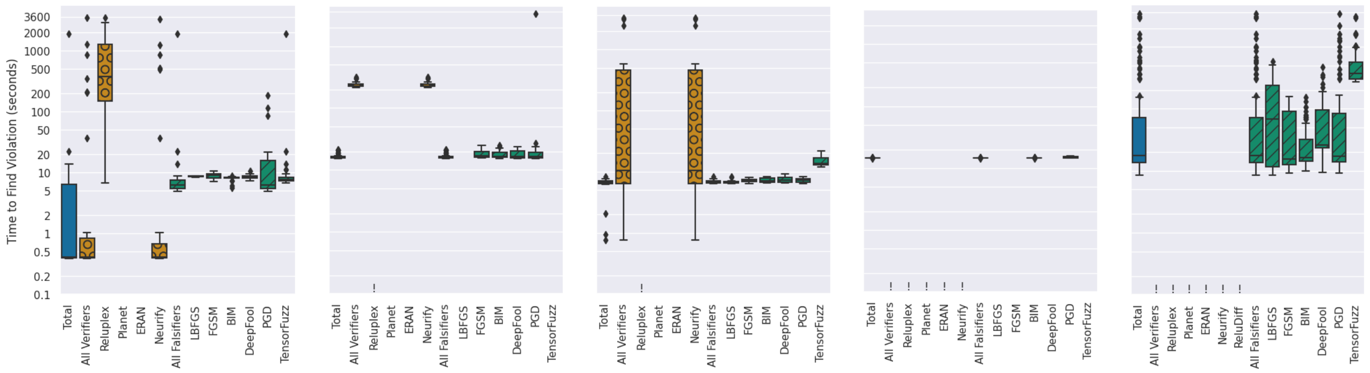
Computing resources. Experiments were run on compute nodes with Intel Xeon Silver 4214 processers at 2.20 GHz and 512GB of memory. Jobs were allowed to use up to 8 processor cores, 64GB of memory, with a time limit of 1 hour.

2) Results: Figure 3 shows the number of violations found by each verifier and falsifier method on the five benchmarks. The y-axis is the proportion of non-verified properties for which the techniques could find violations. We eliminated correctness problems that were known to be unfalsifiable. For ACAS this leaves 37 correctness problems, and does not reduce any of the other benchmarks. The number above each bar in the plots indicates the number of violations found. An exclamation point indicates that the verifier could not be run due to the architecture of the networks being verified.

The ACAS Xu benchmark with its simple and small DNN models, often used in verifier evaluation, showcases where verifiers perform best today. However, even in this benchmark we notice that falsification can complement verification, finding an additional 3 violations.

On the Neurify-DAVE benchmarks, the verifiers find only 33 violations from the 100 DNN correctness problems, while

[3] https://github.com/dlshriver/DNNF/blob/main/docs/_static/appendix.pdf



(a) ACAS Xu (b) Neurify-DAVE (c) GHPR MNIST (d) GHPR DroNet (e) CIFAR-EQ

Fig. 4: The times, in seconds, to find violations for each verifier and falsifier. An exclamation point indicates that a verifier could not be run on a property due to the structure of the network.

the falsifiers find 82 violations, subsuming the 33 violations from the verifiers. The best performing falsification method on this benchmark was BIM, with 74 violations found. PGD and FGSM follow closely with 73 and 69 violations found, respectively. TensorFuzz, the top performing falsification method on the ACAS benchmark, does not find any violations. We conjecture that this is due to the much larger input space. While the ACAS Xu networks have an input dimension of 5, the DAVE networks have an input dimension of 30000, which is more difficult to cover by random fuzzing.

On the GHPR MNIST benchmark, the verifiers can find 17 violations for the 20 properties, while 3 falsifiers, BIM, PGD, and TensorFuzz, can find violations for every property.

On the GHPR DroNet benchmark, the verifiers cannot find any violations, due to not supporting the residual block structures present in the DroNet network. Many of the falsifiers also struggle on these properties, except for PGD and BIM, which can find violations to all 10 properties.

Finally, on the CIFAR-EQ benchmark, the verifiers did not find any violations because they could not be run. Reluplex, Planet, ERAN, and Neurify do not support properties over multiple networks or networks with multiple computation paths, while ReluDiff is limited to networks with only fully-connected layers. Additionally, while PGD finds the most violations, it is complemented by the other falsification approaches, with BIM, DeepFool, and TensorFuzz each finding violations for at least 1 unique property. We conjecture that much of PGD's success is due to its random initialization, which allows it to be run multiple times with different results, increasing the chance of finding a violation.

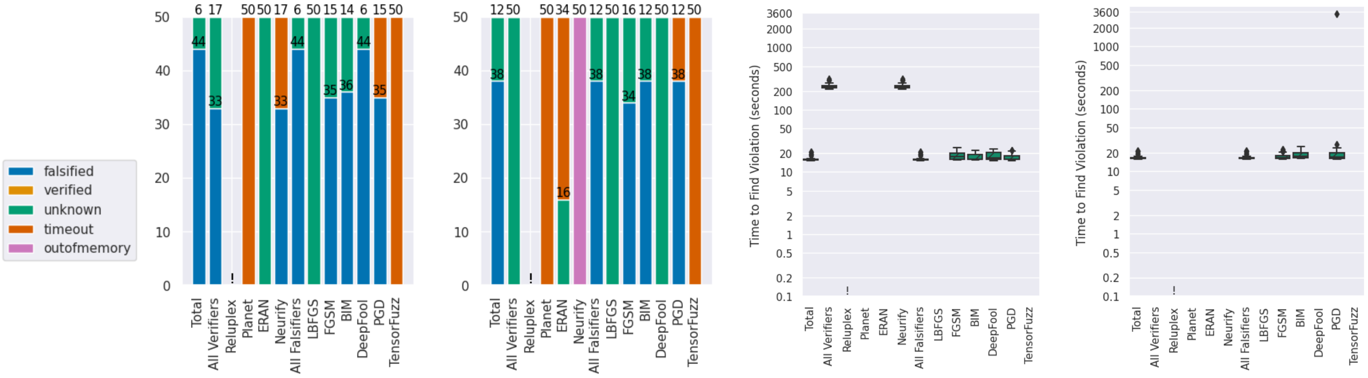
Note that the Planet and ERAN verifiers find no violations for any benchmark. For these benchmarks ERAN cannot find violations since its algorithmic approach focuses on proving that a property holds, which suggests its complementarity with falsification methods. Planet fails to find violations due to the complexity of the problem, and internal tool errors that cause Planet to crash on almost 20% of the correctness problems. We also see that the Reluplex verifier only finds violations for the ACAS Xu benchmark. It cannot find violations on the other benchmarks, since it does not support the architectures of the networks in those benchmarks.

Overall, we find that falsifiers can detect many property violations usually complementing those found by verification, that applying them in a parallel portfolio can leverage their unique strengths, and that they successfully scale to more complex benchmarks.

Box plots of the distributions of time to find violations for each method are shown in Figure 4. Figure 4a shows that the verifiers can be effective on the ACAS Xu benchmark, with Neurify often out performing the falsifiers. This is likely due to the extremely small size of the ACAS Xu networks enabling verification to run efficiently. These plots also show the efficiency struggle of the verifiers as the network get larger. For example, on the Neurify-DAVE benchmark, even when the verifiers can find a property violation, the falsifiers can find violations an order of magnitude faster. For more complex benchmarks, the verifiers cannot find violations within the timeout, so we only report the time for the falsifiers.

We find that falsification can efficiently find property violations even for the most complex benchmarks, with a median time to find a violation across all benchmarks and falsifiers of 16 seconds.

Figures 3 and 4 also reveal that no single falsifier always outperforms the others. While PGD performs well for the benchmarks studied here, we can still increase the number of violations by running mutliple falsifiers. Additionally, the falsifiers that find the most violations, do not necessarily always find them the fastest. Based on these two observations, we recommend using a portfolio approach, running many falsifiers in parallel and stopping as soon as a violation is found by any technique, such as the *All Falsifiers* method shown in the previous figures. This approach finds all the violations found by the verifiers as quickly as the fastest falsifier. We also recommend using falsifiers in conjunction with verifiers, since while falsifiers can often quickly find violations, they cannot prove when a property holds.



(a) Small DAVE

(b) Large DAVE

(c) Small DAVE

(d) Large DAVE

Fig. 5: The number of violations and time to find each violation for the Neurify-DAVE benchmark. The number above each bar gives the total number of property check results in the bar below it. An exclamation point indicates that a verifier could not be run on a property due to the structure of the network.

C. *RQ3: On the Scalability of Reduction-Enabled Falsification*

1) Setup: To explore the scalability of falsification with reduction, we want to evaluate a set of properties across networks that vary in size. To do this we applied the Neurify-DAVE properties to both the small DAVE network [54] and the original larger DAVE network [10]. This will allow us to see how performance of the verifiers and falsifiers change with respect to the size of the network being verified.

2) Results: We present the results of checking the properties in Figures 5a and 5b, as well as the box plots of the times to find violations for each method in Figures 5c and 5d.

On the smaller DAVE network, the verifiers struggle to verify the properties. Reluplex does not run at all, due to its lack of support for convolutional layers, while Planet does run, but reaches the timeout for all properties. The ERAN verifier does not timeout on the small network, but cannot verify any of the properties. Neurify was the only verifier that returned accurate results on the small network, successfully falsifying 33 of the 50 properties, and reaching the time limit on the other 17. While only a single verifier was able to falsify any properties, 4 of the 6 falsification approaches were able to falsify properties, all of them finding more violations than Neurify. The falsifiers were also faster than Neurify, finding violations almost an order of magnitude faster than Neurify on the small DAVE network.

While one verifier was able to find violations on the smaller network, none of the verifiers were able to find violations on the larger DAVE network, which has more than 8 times more neurons. Similar to the small network, Reluplex does not support the network structure, while Planet reaches the time limit for all properties. However, ERAN and Neurify both perform slightly differently. While ERAN was previously able to finish its analysis on the small network, it reaches the time limit for 34 properties on the large network, indicating that it could not scale to the larger network size. Similarly, while Neurify previously found property violations for the small network, it reaches the memory limit on the large network before any violations are found. The falsifiers on the other hand still perform well, with 3 of the 6 verifiers finding property violations. Surprisingly, the DeepFool falsifier goes from 44 violations found on the small network, to 0 violations on the large DAVE network. We conjecture that this may be due to the use of the default parameters for DeepFool, and that adjusting these parameters may yield better results. Additionally, the falsifiers show only a minor increase in the time needed to find a violation, from a median time of 20.2 seconds to 20.7 seconds.

Overall we find that, on the benchmarks explored here, DNN property reduction scales well to larger networks, and enables the application of scalable falsification approaches such as adversarial example generation.

V. Conclusion

In this work we present an approach for reducing DNN correctness problems to facilitate the application of falsifiers, in particular adversarial attacks, for finding DNN property violations. We implement our approach and apply it to a range of correctness problem benchmarks and find that 1) the reduction approach covers a rich set of properties, 2) reducing problems enables falsifiers to find property violations, and 3) since falsifiers tend to have different strengths, a portfolio approach can increase the violation finding ability. In future work we plan to extend DNNF to support the full range of properties supported by our algorithmic approach, perform a systematic evaluation of what factors may influence falsifier performance, and explore how reduction can also broaden the applicability of verifiers.

Data Availability

We make DNNF available at https://github.com/dlshriver/DNNF, and we provide an artifact containing the tool, as well as the data and scripts required to replicate our study at https://doi.org/10.5281/zenodo.4439219.

Acknowledgments

This material is based in part upon work supported by National Science Foundation awards 1900676 and 2019239.

Appendix

A. Proofs

Lemma 1. *Let ϕ be a conjunction of linear inequalities over the variables x_i for i from 0 to $n-1$. We can construct an H-polytope $H = (A, b)$ with Alg. 2 s.t. $(Ax \leq b) \Leftrightarrow (x \models \phi)$.*

Proof. Let $f(x) = \sum_0^{n-1} a_i x_i$ We first show that every lin. ineq. in the conjunction can be reformulated to the form $f(x) \leq b$. It is trivial to show the ineq. can be manipulated to have variables on lhs and a constant on rhs, that $\geq$ can be manipulated to an equivalent form with $\leq$, and $>$ can be manipulated to become $<$. The $<$ comparison can be changed to a $\leq$ comparison by decrementing the rhs constant from b to b' where b' is the largest representable number less than b. We prove ineq. with $<$ can be reformulated to use $\leq$ by contradiction. Assume either $f(x) < b$ and $f(x) > b'$ or $f(x) \geq b$ and $f(x) \leq b'$. Either $b' < f(x) < b$, a contradiction, since $f(x)$ cannot be both larger than the largest representable number less than b and also less than b.[4] Or $b \leq f(x) \leq b'$, a contradiction, since $b' < b$ by definition.

Given a conjunction of lin. ineq. in the form $f(x) \leq b$, Alg. 2 constructs A and b with a row in A and value in b corresponding to each conjunct. There are two cases: $(Ax \leq b) \rightarrow (x \models \phi)$ and $(x \models \phi) \rightarrow (Ax \leq b)$.

We prove case 1 by contradiction. Assume $(Ax \leq b)$ and $(x \not\models \phi)$. By construction of H in Alg. 2, each conjunct of ϕ is exactly 1 constraint in H. If $Ax \leq b$, then all constraints in H must be satisifed, and thus all conjuncts in ϕ must be satisfied and $x \models \phi$, a contradiction.

We prove case 2 by contradiction. Assume $(x \models \phi)$ and $(Ax \not\leq b)$. By construction of H in Alg. 2, each conjunct of ϕ is exactly 1 constraint in H. If $x \models \phi$, then all conjuncts in ϕ must be satisfied, and thus all constraints in H must be satisifed and $Ax \leq b$, a contradiction. □

Lemma 2. *Let $H = (A, b)$ be an H-polytope s.t. $Ax \leq b$. Alg. 4 constructs a DNN, $\mathcal{N}_s$, that classifies whether inputs satisfy $Ax \leq b$. Formally, $x \in H \Leftrightarrow \mathcal{N}_s(x)_0 \leq \mathcal{N}_s(x)_1$.*

Proof. There are 2 cases:

1) $x \in H \rightarrow \mathcal{N}_s(x)_0 \leq \mathcal{N}_s(x)_1$
2) $\mathcal{N}_s(x)_0 \leq \mathcal{N}_s(x)_1 \rightarrow x \in H$

We prove case 1 by contradiction. Assume $x \in H$ and $\mathcal{N}_s(x)_0 > \mathcal{N}_s(x)_1$. From Alg. 4, each neuron in the hidden layer of $\mathcal{N}_s$ corresponds to one constraint in H. The weights of each neuron are the values in the corresponding row of A, and the bias is the negation of the corresponding value of b. If input x satisfies the constraint, then the neuron value will be at most 0, otherwise it will be greater than 0. After the ReLU, each neuron will be equal to 0 if the corresponding constraint is satisfied by x and greater than 0 otherwise. The first output neuron sums all neurons in the hidden layer, while the second has a constant value of 0. If $x \in H$, then all neurons in the hidden layer after activation must have a value of 0 since all constraints are satisfied. However, if all neurons have value 0, then their sum must also be 0, and therefore $\mathcal{N}_s(x)_0 = \mathcal{N}_s(x)_1$, a contradiction.

We prove case 2 by contradiction. Assume $\mathcal{N}_s(x)_0 \leq \mathcal{N}_s(x)_1$ and $x \notin H$. If $x \notin H$, at least one neuron in the hidden layer must have a value greater than 0 after the ReLU since at least one constraint is not satisfied. Because some neuron has a value greater than 0, their sum must also be greater than 0, and therefore $\mathcal{N}_s(x)_0 > \mathcal{N}_s(x)_1$, a contradiction. □

Lemma 3. *Let $H = (A, b)$ be an H-polytope s.t. $Ax \leq b$. Alg. 3 constructs a DNN, $\mathcal{N}_p$, that maps values from the n-dim. unit hypercube to the axis aligned hyperrectangle that minimally bounds H. The range of this mapping does not exclude any x s.t. $Ax \leq b$. Formally, $\forall x \in H. \exists z \in [0, 1]^n . x = \mathcal{N}_p(z)$.*

Proof. The proof is by contradiction. Let the axis aligned hyperrectangle that minimally bounds H be specified by lower bounds $\vec{lb}$ and upper bounds $\vec{ub}$ s.t. $\forall x \in H. \forall i. x_i \in [lb_i, ub_i]$. Alg. 3 constructs a DNN, $\mathcal{N}_p$, that computes $\mathcal{N}_p(z) = Wz + b$, where $W = \text{diag}(ub - lb)$ and $b = lb$. This function is invertible: $\mathcal{N}_p^{-1}(x) = W^{-1}(x - b) = W^{-1}x - W^{-1}b$. Assume $\exists x \in H. \exists i. (z = \mathcal{N}_p^{-1}(x)) \wedge ((z_i < 0) \vee (z_i > 1))$. From the def. of $\mathcal{N}_p^{-1}$, we get $\mathcal{N}_p^{-1}(lb)_i \leq z_i \leq \mathcal{N}_p^{-1}(ub)_i$ and $W_{i,i}^{-1}(lb_i) - W_{i,i}^{-1}(lb_i) = 0 \leq z_i \leq W_{i,i}^{-1}(ub_i) - W_{i,i}^{-1}(lb_i) = (\frac{1}{ub_i - lb_i}(ub_i) - \frac{1}{ub_i - lb_i}(lb_i)) = 1$. Therefore $(lb_i \leq x_i \leq ub_i) \rightarrow (0 \leq z_i \leq 1)$, a contradiction. □

Theorem 2. *Let $\psi = \langle \mathcal{N}, \phi \rangle$ be a correctness problem with its property defined as a formula of disjunctions and conjunctions of linear inequalities over the inputs and outputs of $\mathcal{N}$. Property Reduction (Alg. 1) maps ψ to an equivalid set of correctness problems $reduce(\psi) = \{\langle \mathcal{N}_1, \phi_1 \rangle, \ldots, \langle \mathcal{N}_k, \phi_k \rangle\}$.*

$$\mathcal{N} \models \psi \Leftrightarrow \forall \langle \mathcal{N}_i, \phi_i \rangle \in reduce(\psi). \mathcal{N}_i \models \phi_i$$

Proof. A model that satisfies any disjunct of $DNF(\neg\phi)$ falsifies ϕ. If ϕ is falsifiable, then at least one disjunct of $DNF(\neg\phi)$ is satisfiable.

Alg. 1 constructs a correctness problem for each disjunct. For each disjunct, Alg. 1 constructs an H-polytope, H, which is used to construct a prefix network, $\mathcal{N}_p$, and suffix network, $\mathcal{N}_s$. The algorithm then constructs networks $\mathcal{N}'(x) = concat(\mathcal{N}(x), x)$ and $\mathcal{N}''(x) = \mathcal{N}_s(\mathcal{N}'(\mathcal{N}_p(x)))$. Alg. 1 pairs each constructed network with the property $\phi = \forall x. x \in [0, 1]^n \rightarrow \mathcal{N}''(x)_0 > \mathcal{N}''(x)_1$. A violation occurs only when $\mathcal{N}''(x)_0 \leq \mathcal{N}''(x)_1$. By Lemmas 1, 2, and 3, we get that $\mathcal{N}''(x)_0 \leq \mathcal{N}''(x)_1$ if and only if $\mathcal{N}'(x) \in H$. If $\mathcal{N}'(x) \in H$ then $\mathcal{N}'(x)$ satisfies the disjunct and is therefore a violation of the original property. □

B. On Existance of a Bounded Largest Representable Number

Our proof that property reduction generates a set of robustness problems equivalid to an arbitrary problem relies on the assumption that strict inequalities can be converted to non-strict inequalities. To do so we rely on the existance of a largest representable number that is less than some given value. While this is not necessarily true for all sets of numbers (e.g., $\mathbb{R}$), it is true for for most numeric representations used in computation (e.g., IEEE 754 floating point arithmetic).

[4] We discuss the assumption that such a number exists in Appendix B.

References

[1] Arvind Adimoolam, Thao Dang, Alexandre Donzé, James Kapinski, and Xiaoqing Jin. Classification and coverage-based falsification for embedded control systems. In *International Conference on Computer Aided Verification*, pages 483–503. Springer, 2017.

[2] N. Akhtar and A. Mian. Threat of adversarial attacks on deep learning in computer vision: A survey. *IEEE Access*, 6:14410–14430, 2018.

[3] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, and Jorge Luis Reyes-Ortiz. A public domain dataset for human activity recognition using smartphones. In *21st European Symposium on Artificial Neural Networks, ESANN 2013, Bruges, Belgium, April 24-26, 2013*, 2013.

[4] Yashwanth Annpureddy, Che Liu, Georgios Fainekos, and Sriram Sankaranarayanan. S-taliro: A tool for temporal logic falsification for hybrid systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 254–257. Springer, 2011.

[5] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, and Konrad Rieck. DREBIN: effective and explainable detection of android malware in your pocket. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*. The Internet Society, 2014.

[6] Stanley Bak, Hoang-Dung Tran, Kerianne Hobbs, and Taylor T. Johnson. Improved geometric path enumeration for verifying relu neural networks. In Shuvendu K. Lahiri and Chao Wang, editors, *Computer Aided Verification*, pages 66–96, Cham, 2020. Springer International Publishing.

[7] Osbert Bastani, Yani Ioannou, Leonidas Lampropoulos, Dimitrios Vytiniotis, Aditya V. Nori, and Antonio Criminisi. Measuring neural net robustness with constraints. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 2621–2629, USA, 2016. Curran Associates Inc.

[8] Armin Biere, Alessandro Cimatti, Edmund M Clarke, Masahiro Fujita, and Yunshan Zhu. Symbolic model checking using sat procedures instead of bdds. In *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, pages 317–320, 1999.

[9] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. In *NIPS 2016 Deep Learning Symposium*, 2016.

[10] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016.

[11] Akhilan Boopathy, Tsui-Wei Weng, Pin-Yu Chen, Sijia Liu, and Luca Daniel. Cnn-cert: An efficient framework for certifying robustness of convolutional neural networks. In *AAAI*, Jan 2019.

[12] Elena Botoeva, Panagiotis Kouvaros, Jan Kronqvist, Alessio Lomuscio, and Ruth Misener. Efficient verification of relu-based neural networks via dependency analysis. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 3291–3299. AAAI Press, 2020.

[13] Rudy R. Bunel, Ilker Turkaslan, Philip H. S. Torr, Pushmeet Kohli, and Pawan Kumar Mudigonda. A unified view of piecewise linear neural network verification. In *NeurIPS*, pages 4795–4804, 2018.

[14] Nicholas Carlini and David A. Wagner. Towards evaluating the robustness of neural networks. *CoRR*, abs/1608.04644, 2016.

[15] F. Codevilla, M. Miiller, A. López, V. Koltun, and A. Dosovitskiy. End-to-end driving via conditional imitation learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–9, May 2018.

[16] Cas JF Cremers. The scyther tool: Verification, falsification, and analysis of security protocols. In *International conference on computer aided verification*, pages 414–418. Springer, 2008.

[17] Jeffrey De Fauw, Joseph R. Ledsam, Bernardino Romera-Paredes, Stanislav Nikolov, Nenad Tomasev, Sam Blackwell, Harry Askham, Xavier Glorot, Brendan O'Donoghue, Daniel Visentin, George van den Driessche, Balaji Lakshminarayanan, Clemens Meyer, Faith Mackinder, Simon Bouton, Kareem Ayoub, Reena Chopra, Dominic King, Alan Karthikesalingam, Cían O. Hughes, Rosalind Raine, Julian Hughes, Dawn A. Sim, Catherine Egan, Adnan Tufail, Hugh Montgomery, Demis Hassabis, Geraint Rees, Trevor Back, Peng T. Khaw, Mustafa Suleyman, Julien Cornebise, Pearse A. Keane, and Olaf Ronneberger. Clinically applicable deep learning for diagnosis and referral in retinal disease. *Nature Medicine*, 24(9):1342–1350, Sep 2018.

[18] Tommaso Dreossi, Alexandre Donzé, and Sanjit A Seshia. Compositional falsification of cyber-physical systems with machine learning components. *Journal of Automated Reasoning*, 63(4):1031–1053, 2019.

[19] Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. Output range analysis for deep feedforward neural networks. In *NFM*, volume 10811 of *Lecture Notes in Computer Science*, pages 121–138. Springer, 2018.

[20] Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy Mann, and Pushmeet Kohli. A dual approach to scalable verification of deep networks. In *Proceedings of the Thirty-Fourth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-18)*, pages 162–171, Corvallis, Oregon, 2018. AUAI Press.

[21] Rüdiger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *Automated Technology for Verification and Analysis - 15th International Symposium, ATVA 2017, Pune, India, October 3-6, 2017, Proceedings*, pages 269–286, 2017.

[22] Yizhak Yisrael Elboher, Justin Gottschlich, and Guy Katz. An abstraction-based framework for neural network verification. In Shuvendu K. Lahiri and Chao Wang, editors, *Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part I*, volume 12224 of *Lecture Notes in Computer Science*, pages 43–65. Springer, 2020.

[23] Andre Esteva, Brett Kuprel, Roberto A. Novoa, Justin Ko, Susan M. Swetter, Helen M. Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115–118, Feb 2017.

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

[25] Patrice Godefroid and Pierre Wolper. Using partial orders for the efficient verification of deadlock freedom and safety properties. In *International Conference on Computer Aided Verification*, pages 332–342. Springer, 1991.

[26] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[27] Gerard J. Holzmann. The model checker spin. *IEEE Transactions on software engineering*, 23(5):279–295, 1997.

[28] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *CAV (1)*, volume 10426 of *Lecture Notes in Computer Science*, pages 3–29. Springer, 2017.

[29] Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*, pages 97–117, 2017.

[30] Guy Katz, Derek A Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, et al. The Marabou framework for verification and analysis of deep neural networks. In *International Conference on Computer Aided Verification*, pages 443–452, 2019.

[31] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[32] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial examples in the physical world. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.net, 2017.

[33] Changliu Liu, Tomer Arnon, Christopher Lazarus, Clark W. Barrett, and Mykel J. Kochenderfer. Algorithms for verifying deep neural networks. *CoRR*, abs/1903.06758, 2019.

[34] Changliu Liu and Taylor T. Johnson. Vnn-comp. https://sites.google.com/view/vnn20/vnncomp.

[35] Antonio Loquercio, Ana Isabel Maqueda, Carlos R. Del Blanco, and Davide Scaramuzza. Dronet: Learning to fly by driving. *IEEE Robotics and Automation Letters*, 2018.

[36] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[37] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: A simple and accurate method to fool deep neural networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 2574–2582. IEEE Computer Society, 2016.

[38] Augustus Odena, Catherine Olsson, David Andersen, and Ian Goodfellow. TensorFuzz, 2019. https://github.com/brain-research/tensorfuzz.

[39] Augustus Odena, Catherine Olsson, David Andersen, and Ian Goodfellow. TensorFuzz: Debugging neural networks with coverage-guided fuzzing. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 4901–4911, Long Beach, California, USA, 09–15 Jun 2019. PMLR.

[40] Kurt M. Olender and Leon J. Osterweil. Interprocedural static analysis of sequencing constraints. *ACM Trans. Softw. Eng. Methodol.*, 1(1):21–52, January 1992.

[41] Nicolas Papernot, Fartash Faghri, Nicholas Carlini, Ian Goodfellow, Reuben Feinman, Alexey Kurakin, Cihang Xie, Yash Sharma, Tom Brown, Aurko Roy, Alexander Matyasko, Vahid Behzadan, Karen Hambardzumyan, Zhishuai Zhang, Yi-Lin Juang, Zhi Li, Ryan Sheatsley, Abhibhav Garg, Jonathan Uesato, Willi Gierke, Yinpeng Dong, David Berthelot, Paul Hendricks, Jonas Rauber, and Rujun Long. Technical report on the cleverhans v2.1.0 adversarial examples library. *arXiv preprint arXiv:1610.00768*, 2018.

[42] Brandon Paulsen, Jingbo Wang, and Chao Wang. Reludiff: Differential verification of deep neural networks. In *Proceedings of the 42nd International Conference on Software Engineering, ICSE 2020*, 2020.

[43] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. In *ICLR*. OpenReview.net, 2018.

[44] Wenjie Ruan, Xiaowei Huang, and Marta Kwiatkowska. Reachability analysis of deep neural networks with provable guarantees. In *IJCAI*, pages 2651–2659. ijcai.org, 2018.

[45] David Shriver, Dong Xu, Sebastian G. Elbaum, and Matthew B. Dwyer. Refactoring neural networks for verification. *CoRR*, abs/1908.08026, 2019.

[46] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. Fast and effective robustness certification. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 10802–10813. Curran Associates, Inc., 2018.

[47] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin T. Vechev. An abstract domain for certifying neural networks. *PACMPL*, 3(POPL):41:1–41:30, 2019.

[48] Robert E. Strom and Daniel M Yellin. Extending typestate checking using conditional liveness analysis. *IEEE Transactions on Software Engineering*, 19(5):478–485, 1993.

[49] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *CoRR*, abs/1710.08864, 2017.

[50] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[51] Vincent Tjeng, Kai Y. Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. In *International Conference on Learning Representations*, 2019.

[52] Hoang-Dung Tran, Stanley Bak, Weiming Xiang, and Taylor T. Johnson. Verification of deep convolutional neural networks using imagestars. In Shuvendu K. Lahiri and Chao Wang, editors, *Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part I*, volume 12224 of *Lecture Notes in Computer Science*, pages 18–42. Springer, 2020.

[53] Hoang-Dung Tran, Xiaodong Yang, Diego Manzanas Lopez, Patrick Musau, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, and Taylor T. Johnson. NNV: the neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In Shuvendu K. Lahiri and Chao Wang, editors, *Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part I*, volume 12224 of *Lecture Notes in Computer Science*, pages 3–17. Springer, 2020.

[54] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Efficient formal safety analysis of neural networks. In *NeurIPS*, pages 6369–6379, 2018.

[55] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Formal security analysis of neural networks using symbolic intervals. In *USENIX Security Symposium*, pages 1599–1614. USENIX Association, 2018.

[56] X. Wang, Y. Peng, L. Lu, Z. Lu, M. Bagheri, and R. M. Summers. Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3462–3471, 2017.

[57] Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Luca Daniel, Duane S. Boning, and Inderjit S. Dhillon. Towards fast computation of certified robustness for relu networks. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 5273–5282. PMLR, 2018.

[58] Eric Wong and J. Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 5283–5292. PMLR, 2018.

[59] W. Xiang, H. Tran, and T. T. Johnson. Output reachable set estimation and verification for multilayer neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 29(11):5777–5783, Nov 2018.

[60] Xiaofei Xie, Lei Ma, Felix Juefei-Xu, Minhui Xue, Hongxu Chen, Yang Liu, Jianjun Zhao, Bo Li, Jianxiong Yin, and Simon See. Deephunter: A coverage-guided fuzz testing framework for deep neural networks. In *28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2019.

[61] Dong Xu, David Shriver, Matthew B. Dwyer, and Sebastian Elbaum. Systematic generation of diverse benchmarks for dnn verification. In *Computer Aided Verification CAV*, 2020.

[62] X. Yuan, P. He, Q. Zhu, and X. Li. Adversarial examples: Attacks and defenses for deep learning. *IEEE Transactions on Neural Networks and Learning Systems*, 30(9):2805–2824, 2019.