EPiC Series in Computing

Volume 94, 2023, Pages 124–143

Proceedings of 24th International Conference on Logic for Programming, Artificial Intelligence and Reasoning

Tighter Abstract Queries in Neural Network Verification

Elazar Cohen[1]*, Yizhak Yisrael Elboher[1], Clark Barrett[2], and Guy Katz[1]

[1] The Hebrew University of Jerusalem, Jerusalem, Israel
{elazar.cohen1, yizhak.elboher, g.katz}@mail.huji.ac.il
[2] Stanford University, Stanford, USA
barrett@cs.stanford.edu

Abstract

Neural networks have become critical components of reactive systems in various domains within computer science. Despite their excellent performance, using neural networks entails numerous risks that stem from our lack of ability to understand and reason about their behavior. Due to these risks, various formal methods have been proposed for verifying neural networks; but unfortunately, these typically struggle with scalability barriers. Recent attempts have demonstrated that abstraction-refinement approaches could play a significant role in mitigating these limitations; but these approaches can often produce networks that are so abstract, that they become unsuitable for verification. To deal with this issue, we present `CEGARETTE`, a novel verification mechanism where both the system and the property are abstracted and refined simultaneously. We observe that this approach allows us to produce abstract networks which are both small and sufficiently accurate, allowing for quick verification times while avoiding a large number of refinement steps. For evaluation purposes, we implemented `CEGARETTE` as an extension to the recently proposed `CEGAR-NN` framework. Our results are highly promising, and demonstrate a significant improvement in performance over multiple benchmarks.

1 Introduction

Deep neural networks (DNNs) have become state-of-the-art technology in many fields [58], including image processing [37], computational photography [13], speech recognition [1, 35], natural language processing [23], video processing [10, 45], autonomous driving [16], and many others. Nowadays, they are even increasingly being used as critical components in various systems [55, 60, 74, 79], and society's reliance on them is constantly increasing.

Despite these remarkable achievements, neural networks suffer from multiple limitations, which undermine their reliability: the training process of DNNs is based on assumptions regarding the data, which may fail to hold later on [36, 43]; the training process might cause over-fitting of the DNN to specific data [77, 91]; and, independently of the above, there are various attacks that can fool a DNN into performing unwanted actions [72, 87].

*Equal Contribution

In order to overcome these difficulties and ensure the correctness and safety of DNNs, the formal methods community has been devising techniques for verifying them [26, 46, 48, 62, 83]. Techniques for neural network verification (NNV) receive as input a neural network and a set of constraints over its input and output, and check whether there exists an input/output pair that satisfies these constraints. Typically, the constraints encode the negation of some desirable property, and so such a pair constitutes a counter-example (the `SAT` case); whereas if no such pair exists, the desired property holds (the `UNSAT` case). NNV has been studied extensively in recent years, and many different verifiers have been proposed [24–26, 46, 48, 62, 67, 83]. However, scalability remains a fundamental barrier, both theoretical and practical, which limits the use of NNV engines: generally, increasing the number of neurons of the verified neural network implies an exponential increase in the complexity of the verification problem [40, 46].

In order to alleviate this scalability limitation, recently there have been attempts to apply *abstraction techniques* within NNV [11, 27, 28, 57, 69, 76], often focusing on the counter-example guided abstraction refinement (CEGAR) framework [21]. CEGAR is a well-known approach, aimed at expediting the solving of complex verification queries, by *abstracting* the model being verified into a simpler one — in such a way that if the simpler model is safe, i.e. the query is `UNSAT`, then so is the original model. In case of a `SAT` answer from the verifier, we check whether the satisfying assignment of the abstract model is also a satisfying assignment of the original. If so, the original query is declared `SAT`, the satisfying assignment is returned, and the process ends. Otherwise, the satisfying assignment is called *spurious* (or *erroneous*), indicating that the abstract query is too coarse, and should be refined into a slightly "less abstract" query, which is a bit closer to the original model (but is hopefully still smaller). CEGAR has been successfully used in various formal method applications [9, 15, 42, 65, 73], and also in the context of NNV [27, 28, 57]. Typically, these approaches generate an *abstract neural network*, which is smaller than the original, and is created by the merging of neurons of the original network. The refinement, accordingly, is performed by splitting previously merged neurons. Other related approaches have also considered abstracting the ranges of values obtained by neurons in the network [11, 69, 70, 76].

The general motivation for abstraction schemes is that smaller, abstract networks are easier to verify. While this is often true, smaller networks tend to be far less precise, and verifying them often requires multiple refinement steps [28, 70]. In extreme cases, these multiple refinement steps can render the verification process slower than directly verifying the original network [28]. Here, we seek to tackle this problem, by improving the abstract verification queries. We propose a novel verification scheme for DNNs, wherein abstraction and refinement operations include altering not only the network (as in [21, 27, 28, 57, 70]), but also the property being verified. The motivation is to render the abstract properties more *restrictive*, in a way that will reduce the number of spurious counter-examples encountered during the verification process; but at the same time, ensure that the abstract queries still maintain the over-approximation property: if the abstract query is `UNSAT`, the original query is `UNSAT` too. The key idea on which our approach is based is to compute a *minimal difference* between the outputs of the abstract network and the original network, and then use this minimal difference to tighten the property being verified, in a sound manner.

Our approach is not coupled to any specific DNN verification method, and can use multiple DNN verifiers as black-box backends. For evaluation purposes, we implemented it on top of the Marabou DNN verifier [48]. We then tested our approach on the ACAS-Xu benchmarks [44] for airborne collision avoidance, and also on MNIST benchmarks for digit recognition [54]. Our results indicate that property abstraction affords a significant increase in the number of queries that can be verified within a given timeout, as well as a sharp decrease in the number

of refinement steps performed.

To summarize, our contributions are as follows: (i) we present CEGARETTE, a novel CEGAR framework for DNN verification, that abstracts not only the network but also the property being verified; (ii) we provide a publicly available implementation of our approach, CEGARETTE-NN; and (iii) we use our implementation to demonstrate the practical usefulness of our approach.

The rest of this paper is organized as follows. In Section 2, we provide a brief background on neural networks and their verification, followed by an explanation of the CEGAR framework and its implementation for neural network verification. In Section 3, we describe our novel verification framework CEGARETTE. In Section 4, we discuss how to apply this framework for abstracting and refining DNNs, followed by an evaluation in Section 5. Related work is discussed in Section 6, and we conclude in Section 7.

2 Background

2.1 Neural Networks

A neural network [33] is a directed graph, comprised of a sequence of *layers*: an input layer, followed by one or more consecutive hidden layers, and finally an output layer. A layer is a collection of nodes, also referred to as *neurons*. Here we focus on *feed-forward* neural networks, where the values of neurons are computed based on values of neurons in preceding layers. Thus, when the network is evaluated, values are assigned to neurons in its input layer; and they are then propagated, layer after layer, through to the output layer.

We use $n_{i,j}$ to denote the j'th neuron of layer i. Typically, the value of neuron $n_{i,j}$, denoted as $v_{i,j}$, is given by the following formula:

$$v_{i,j} = act_{i,j}(b_{i,j} + \sum_{k=1}^{l_{i-1}} w^i_{k,j} \cdot v_{i-1,k})$$

where l_{i-1} is the number of neurons in the $i-1$'th layer, $act_{i,j}$ is a pre-defined (neuron-specific) activation function, $w^i_{k,j}$ is the weight of the outgoing edge from $n_{i-1,k}$ to $n_{i,j}$, $v_{i-1,k}$ is the value of the k'th neuron in the $i-1$'th layer, and $b_{i,j}$ is the bias value of the j'th neuron in the i'th layer. For simplicity, we assume here that the only activation function in use is the Rectified Linear Unit (ReLU) function [63], which is defined by $\text{ReLU}(x) = max(0, x)$, and is very common in practice.

Fig. 1 depicts a small neural network. The network has 3 layers, of sizes $l_1 = 1, l_2 = 2$ and $l_3 = 1$. Its weights are $w^2_{1,1} = 10$, $w^2_{1,2} = 1$, $w^3_{1,1} = 3$ and $w^3_{2,1} = 4$, and its biases are all zeros. For input $v_{1,1} = 21$, node $n_{2,1}$ evaluates to 210 and node $n_{2,2}$ evaluates to 21 (both are positive, and hence not changed by the ReLU activation function). The output node $n_{3,1}$ then evaluates to $3 \cdot 210 + 4 \cdot 20 = 714$.

2.2 Neural Network Verification

The goal of neural network verification (NNV) is to determine the satisfiability of a *verification query*. A query is typically defined to be a triple $\langle N, P, Q \rangle$, where: (i) N is a neural network; (ii) P is an input property, which is a conjunction of constraints on the input neurons; and (iii) Q is an output property, which is a conjunction of constraints on the output neurons [56]. The query is SAT if and only if there exists an input vector x_0 to N, such that $P(x_0)$ and $Q(N(x_0))$ both hold; in which case the verifier returns x_0 as the counter-example. As we

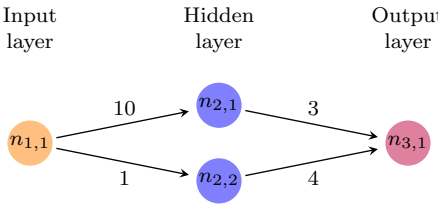

Figure 1: A simple, feed-forward neural network.

previously mentioned, Q typically represents some undesirable behavior of N on inputs from P, and so the goal is to obtain an `UNSAT` result.

Most existing verifiers focus primarily on ReLU activation functions, and we follow this line here. In addition, most existing verifiers assume that P is a conjunction of linear constraints on the input values, and we again take the same path. Finally, we make the simplifying assumption that N has a single output neuron y, and that the property Q is of the form $y > c$. This assumption may seem restrictive, but in fact, it does not incur any loss of generality [28], and is sufficient for expressing many properties of interest with arbitrary Boolean structure, via a simple reduction.

In recent years, various methods have been proposed for solving the verification problem (for a brief overview, see Section 6). Our abstraction-refinement mechanism is designed to be compatible with many of these techniques, as we later describe.

2.3 Counter-Example Guided Abstraction Refinement (CEGAR)

Counter-example guided abstraction refinement (CEGAR) [21] is frequently used as part of model-checking frameworks, and it has recently been applied to neural network verification, as well. The general framework, borrowed from [28], is presented in Algorithm 1. Given a *verification query* $\langle N, P, Q\rangle$, we begin by generating an *abstract network* N'. Then, the CEGAR loop starts, where in each iteration, we verify a query with the current abstract network, $\langle N', P, Q\rangle$. The abstract network N' is constructed in a way that makes $\langle N', P, Q\rangle$ an over-approximation of $\langle N, P, Q\rangle$: if the former is `UNSAT`, then so is the latter. Thus, if the underlying verifier returns `UNSAT` on the current query, we can stop and return `UNSAT`. Otherwise, the verifier returns a satisfying assignment x_0 for $\langle N', P, Q\rangle$, and we check whether this x_0 constitutes a satisfying assignment for $\langle N, P, Q\rangle$ as well. If so, we return `SAT` and x_0 as the satisfying assignment, and stop. Otherwise, we say that x_0 is a *spurious* counter-example, indicating that N' is too coarse; in which case, we *refine* N' into a new "tighter" network, N'', whose verification is more likely to produce accurate results. This refinement process is guided by the spurious counter-example x_0. This general framework can be instantiated in many ways, depending on the implementation of the `abstract` and `refine` operations.

There have been a few recent attempts to apply CEGAR in the context of NNV [27,28,57], all following a similar approach. At first, a preprocessing phase is performed, and every hidden neuron in the network is classified according to its effect on the network's output. Then, abstraction is carried out by repeatedly merging pairs of neurons with the same type, usually making the network significantly smaller than the original.

We focus here on one of these approaches, called `CEGAR-NN` [28]. There, the preprocessing phase initially splits each hidden neuron into 4 neurons, each belonging to one of 4 categories based on the effect of the neuron on values of both the next layer's neurons and the output:

Algorithm 1 Abstraction-based DNN Verification($\langle N, P, Q \rangle$)

1: $N' \leftarrow \texttt{abstract}(N)$
2: **if** *Verify*($\langle N', P, Q \rangle$) is `UNSAT` **then**
3: return `UNSAT`
4: **else**
5: Extract satisfying assignment x_0
6: **if** x_0 is a satisfying assignment for N **then**
7: return `SAT`, x_0
8: **else**
9: $N'' \leftarrow \texttt{refine}(N', N, x_0)$
10: $N' \leftarrow N''$
11: Goto step 2
12: **end if**
13: **end if**

the output edges can be all positive (*pos*) or all negative (*neg*), and the value of a neuron can increase the output when being increased (*inc*), or increase the output when being decreased (*dec*). The splitting process changes the network's architecture but does not change its output — i.e., the preprocessed network is completely equivalent to the original. After splitting the neurons and categorizing the new neurons, pairs of neurons from the same layer that share a category can be merged into a single neuron. The weights and biases of the new, merged neuron are determined by aggregating the weights and biases of its constituent neurons, in a way that depends on the category of these neurons, and which guarantees that the abstract network's (single) output is always greater than or equal to that of the original network when the two networks are evaluated on the same input. This, combined with our assumption that the output property is always of the form $y > c$, guarantees that the verification query on the abstract network constitutes an over-approximation of the original query. Finally, this pair-wise merging is then repeated, resulting in a much smaller network.

An example showing the result of applying this abstraction to the network from Fig. 1 appears in Fig. 2: $n_{2,1}$ and $n_{2,2}$ both already belong to the same category (in this tiny network, all other categories are empty). We merge them into a single abstract neuron, $n_{2,1+2}$. The output weights are aggregated by a sum operation, so we get an output weight of $3 + 4 = 7$. For the input weight aggregation, we take the maximal value of the two, so we get an input weight of $\max(10, 1) = 10$. The key property here is that for every input x, $N'(x) \geq N(x)$. For additional details, as well as a discussion of various heuristics for selecting which neurons should be merged and in what order, see [28, 57].

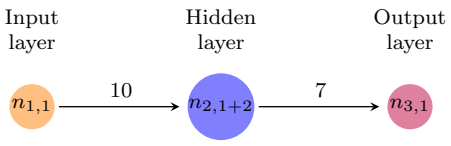

Figure 2: An abstract network, generated from the network from Fig. 1.

The refinement operation is then carried out by splitting an abstract neuron, which represents a set of original neurons, into two or more new neurons. In the example, the refinement

step would be to split the abstract neuron $n_{2,1+2}$ into the original neurons $n_{2,1}$ and $n_{2,2}$, and to restore their original weights.

Empirical evaluations of CEGAR-NN demonstrated its great potential to enhance the scalability of NNV engines, but as many experiments show [28], there is much room for improvement.

3 CEGARETTE: Tighter Abstract Queries

3.1 Motivation

Although CEGAR-NN is quite useful in many cases, it is also prone to producing spurious counter-examples, which in turn triggers multiple refinement steps that slow the process down. For example, observe again the network N from Fig. 1, and consider the verification query

$$v_1 = \langle N, n_{1,1} \in [20, 21], n_{3,1} > 800 \rangle$$

Here, a sound verifier will declare that v_1 is UNSAT, because for inputs in the range $[20, 21]$, network N can only produce outputs that are upper-bounded by $N(21) = 714$. If we attempt to verify this query using CEGAR-NN, we would generate the abstract query

$$\langle N', n_{1,1} \in [20, 21], n_{3,1} > 800 \rangle,$$

where N' is the network in Fig. 2. For this query, a sound verifier will return a satisfying assignment, such as $n_{1,1} = 20$. Of course, this assignment is spurious: although $N'(20) = 1400 \geq 800$, we get that $N(20) = 680 < 800$. Thus, refinement would be carried out, transforming N' back into the original network N, and the overall process would be slower than just verifying N directly. More broadly, we recognize the following issue with CEGAR-NN and related techniques:

Performance vs Accuracy. Neuron-merging-based abstraction techniques, such as CEGAR-NN, have an intrinsic trade-off between performance and accuracy. In order to avoid coarse model abstractions and the ensuing spurious counter-examples, it is desirable to look for *accurate* model abstractions. On one hand, a common approach in CEGAR-based verification is to heuristically guess an accurate initial abstract model, so that future satisfying assignments will not be spurious; and if a spurious assignment is discovered nonetheless, to try and heuristically select a refinement operation that will restore as much accuracy as possible. On the other hand, a model with a higher accuracy is almost always larger and hence verifying it is slower. Thus, these two requirements conflict with each other: generating an *accurate* abstract model restricts our ability to generate small abstractions, and results instead in larger models that take longer to verify.

3.2 Introducing Property Abstractions

In order to overcome the aforementioned issue, we introduce an extension to CEGAR, which we term CEGARETTE: CEGAR Enhanced by TighTEning. In CEGARETTE, when we are given a verification query $\langle N, P, Q \rangle$, instead of abstracting and refining only the network N, we may also alter the output property Q in order to produce an over-approximate query $\langle N', P, Q' \rangle$.

To set the stage, we introduce the following definitions:

Definition 1. *A verification query $\langle N', P, Q' \rangle$ is an over-approximation of another verification query $\langle N, P, Q \rangle$, if and only if the unsatisfiability of $\langle N', P, Q' \rangle$ implies the unsatisfiability of $\langle N, P, Q \rangle$.*

We refer to the process in which $\langle N', P, Q' \rangle$ is created from $\langle N, P, Q \rangle$ as *query abstraction*.

Definition 2. *Let $\langle N, P, Q \rangle$ be some base verification query. A query $\langle N'', P, Q'' \rangle$ is called a refinement of a query $\langle N', P, Q' \rangle$, if (i) $\langle N'', P, Q'' \rangle$ is an over-approximation of $\langle N, P, Q \rangle$; and (ii) $\langle N', P, Q' \rangle$ is an over-approximation of $\langle N'', P, Q'' \rangle$.*

To illustrate the effect of changing the property, we consider again the verification query from our running example: $v_1 = \langle N, n_{1,1} \in [20, 21], n_{3,1} > 800 \rangle$. Let us consider what happens to this query if we change its output property. Decreasing the constant 800, for example by setting $n_{3,1} > 400$, renders the property easier to satisfy (e.g., the output 600 satisfies the latter output property, but not the former). Therefore, if $\langle N, n_{1,1} \in [20, 21], n_{3,1} > 400 \rangle$ is `UNSAT`, then v_1 is also `UNSAT`. Consequently, we claim that decreasing the constant that appears in an output property Q results in an over-approximation.

While the aforementioned process is sound, it goes against the grain of our desired approach: decreasing the constant in the output property could potentially result in additional, not fewer, spurious counter-examples. Thus, what we would like to do is to *increase* the constant that appears in the output property, in order to rule out spurious counter-examples. More formally, there is a set of spurious inputs S, whose outputs satisfy the property in the abstract network: $\forall x \in S : (N'(x) > c)$, but not in the original network, where their outputs are smaller: $\forall x \in S : (N(x) \leq c)$. By increasing the constant in the output property, we seek to avoid these spurious examples.

As it turns out, there are cases in which we can increase the output bound, and still obtain an over-approximate query. Consider the query

$$v_2 = \langle N', n_{1,1} \in [20, 21], n_{3,1} > 1486 \rangle$$

In v_2, there are changes (with respect to v_1) in both the model N and the output property Q. By applying changes to the model N and generating N', the output increases: for every input x, $N(x) \leq N'(x)$. In other words, there is a minimal (non-negative) difference between the outputs:

$$\exists d \geq 0 : \forall x : N(x) + d \leq N'(x) \tag{1}$$

Given that the input property $P := x \in [20, 21]$ was not changed, and assuming that we can calculate d, we can *increase* the output constant by any number in $[0, d]$, and still get an over-approximate query; if the value in the abstract network is smaller than $c + d$, the output of the original network is bounded from above by $(c + d) - d = c$ (by Eq. 1).

Going back to our running example, we need to calculate the minimal difference between the respective outputs of the networks, for any input vector in the range specified by property P. For $P = [20, 21]$, it can be shown that the minimal difference is bounded by $d \geq N'(20) - N(21) = 1400 - 714 = 686$. Consequently, we can increase the constant in the output property from 800 (in v_1) to $800 + 686 = 1486$ (in v_2), and still get an over-approximate query: if $\forall x \in [20, 21] : N'(x) \leq 1486$, then (by difference of bounds) $\forall x \in [20, 21] : N(x) \leq 1486 - 686 = 800$. Making this adjustment rules out the spurious counter-example we saw before, namely $N'(20) = 1400$.

More broadly, the basic idea underlying `CEGARETTE` is that in order to produce a small but accurate abstraction, the output property (Q) should be tightened as much as possible in parallel to reducing the size of the neural network (N). The abstraction became more refined, and as a consequence, the counter example is more relevant, and the number of refinement steps should decrease. Naturally, the abstraction and tightening processes are linked, as the network determines the possible output properties.

The example shows how changing N enables tightening Q, but CEGARETTE is not limited to a specific order; it is also possible to first increase the constant of Q and only then change N, as long as the over-approximation property is maintained.

Algorithm 2 shows the general outline of the CEGARETTE framework. First, we generate an initial *abstract verification query* using queryAbstract. Then a loop starts, where in each iteration we verify the current abstract query. If the answer is UNSAT, we are guaranteed that the original query is also UNSAT (by the over-approximation property), and can stop and return UNSAT. Otherwise, the satisfying assignment x_0 is examined in the original model, and if it is also a satisfying assignment there, we return SAT and x_0. In the case where x_0 is not a satisfying assignment for $\langle N, P, Q\rangle$, the current abstract query is apparently too coarse and is thus refined using queryRefine — producing a more precise abstract query, for the next iteration.

Algorithm 2 CEGARETTE-based Verification(N, P, Q)

$\langle N', P, Q'\rangle \leftarrow$ queryAbstract$(\langle N, P, Q\rangle)$
if *Verify*$\langle N', P, Q'\rangle$ is UNSAT **then**
  return UNSAT
else
  Extract counterexample x_0
  if x_0 is a counterexample for $\langle N, P, Q\rangle$ **then**
    return SAT, x_0
  else
    $\langle N'', P, Q''\rangle \leftarrow$ queryRefine$(\langle N', P, Q'\rangle, N, x_0)$
    $\langle N', P, Q'\rangle \leftarrow \langle N'', P, Q''\rangle$
    Goto step 2
  end if
end if

The structure of CEGARETTE is similar to that of CEGAR-NN, but instead of invoking the abstract and refine operations as in Algorithm 1, which only abstract and refine the network, CEGARETTE invokes queryAbstract and queryRefine, which abstract and refine both the network and the output property. Therefore, Algorithm 1 is a special case of Algorithm 2, and CEGARETTE extends CEGAR.

4 DNN Verification Using CEGARETTE

In this section, we describe CEGARETTE-NN, which is our implementation of CEGARETTE for NNV. Specifically, we propose a particular implementation of the queryAbstract and queryRefine operators, which modify the model and the output property in a way that soundly produces over-approximations of the original query.

Given a verification query $\langle N, P, Q\rangle$, our proposed implementation of queryAbstract is shown as Algorithm 3. It begins by generating N', an abstract neural network, using the same abstract from the CEGAR-NN framework; and then proceeds to tighten the output property, by adding to the output constant a scalar d, which lower-bounds the minimal difference in outputs between N' and N.

In Algorithm 4 we describe how to compute the constant d. This is performed by calculating the lower bound of the abstract network's output, and the upper bound of the original network's output, and then subtracting the latter from the former. If the result is positive, it can be used in order to update the output property in the over-approximate verification query. The output

Algorithm 3 AbstractQueryGeneration(N, P, Q)

1: $N' \leftarrow \texttt{abstract}(N)$
2: $Q' \leftarrow$ TightenProperty(N', N, P, Q)
3: return $\langle N', P, Q' \rangle$

property is of the form $Q := y < c$ for some constant c (denoted by Q_c in Algorithm 4), and so the update is performed by setting $Q' := y < c + d$.

Algorithm 4 TightenProperty(N', N, P, Q)

1: Compute $l_{N'}$, a lower bound on the output of N'
2: Compute u_N an upper bound on the output of N
3: $d = \max(0, l_{N'} - u_N)$
4: $Q' := y > Q_c + d$
5: **return** Q'

Computing the lower and upper bounds of the abstract and the original networks with respect to a given input range in Algorithm 4 is based on *bound propagation methods*, which maintain and propagate tractable and sound bounds through neural networks. Bound propagation has been studied extensively, and there are many scalable methods intended for this purpose [30, 34, 50, 51, 75, 82, 83, 88, 92]. We give a simple example later on.

Lemma 1. *Algorithm 3 returns an over-approximation of the verification query passed to it as input.*

Proof. Algorithm 3 begins by generating N', an abstract network, from N. Then it invokes TightenProperty, described in Algorithm 4, which sets $Q' := y' \leq c + max(0, l_{N'} - u_N)$, where u_N is an upper bound for the output of N and $l_{N'}$ is a lower bound for the output of N'. We now wish to prove that if $\langle N', P, Q' \rangle$ is `UNSAT`, then $\langle N, P, Q \rangle$ is also `UNSAT`. Differently put, we need to show that

$$\forall x \in P : N'(x) \leq c + max(0, l_{N'} - u_N) \Rightarrow \forall x \in P : N(x) \leq c$$

This is equivalent, modus tollens, to proving that

$$\exists x \in P : N(x) > c \Rightarrow \exists x \in P : N'(x) > c + max(0, l_{N'} - u_N)$$

The last implication holds since we know that for every input x, the output increases after abstracting N to N' by at least $l_{N'} - u_N$; and we know also that the output cannot decrease. Overall, model abstraction increases x's output by at least $max(0, l_{N'} - u_N)$, and hence the implication holds. □

Next, `queryRefine` is implemented in Algorithm 5. First, we refine the previous abstraction of the network (by splitting neurons that had previously been merged), and then computing a new output property, from scratch, with respect to this new network.

Lemma 2. *Algorithm 5 returns an over-approximation of the verification query* $\langle N, P, Q \rangle$.

Proof. N'' is an abstract network generated from N. The remainder of the proof is the same as in the proof of Lemma 1. □

Algorithm 5 RefinedQueryGeneration($\langle N', P, Q' \rangle, N, x_0$)

1: $N'' \leftarrow \texttt{refine}(N', N, x_0)$
2: $Q'' = \text{TightenProperty}(N'', N, P, Q)$
3: return $\langle N'', P, Q'' \rangle$

Now that we have established the soundness of our approach via Lemmas 1 and 2, we proceed to prove that it always terminates.

Lemma 3. *For any verification query $\langle N, P, Q \rangle$, CEGARETTE-NN converges.*

Proof. CEGARETTE-NN implements Algorithm 2, using `queryAbstract` from Algorithm 3 and `queryRefine` from Algorithm 5. We show that with these implementations, Algorithm 2 converges.

Assume that Algorithm 2 does not converges, then Line 11 takes place infinite number of times. In that case, Lines 9-10 takes place infinite number of times too. Notice that after the abstraction in Line 1, the network N' only changes in Line 10. After finite number of calls to `queryRefine` which is implemented by Algorithm 5, the query $\langle N', P, Q' \rangle$ will be equal to the original query $\langle N, P, Q \rangle$; Algorithm 4 refine the network and then tighten the query. When N' is fully refined ($N' == N$), the difference between the lower bound of N' and the upper bound of N can't be positive and $d = 0$ which implies that $Q' == Q$. In the next iteration, both getting `UNSAT` or `SAT` will terminate the verification, since the current query is equal to the original query. In contradiction to our assumption.

□

Returning to our running example, given the verification query v_1 above, applying Algorithm 2 triggers Algorithm 3, which generates N' from N; and then, in turn, triggers Algorithm 4, which calculates d, which bounds the minimal difference between N and N'. For simplicity, we use a naïve method for calculating the lower and upper bounds of a network, called interval bound propagation, or `IBP` [34].

In `IBP`, bounds are propagated forward in the network, starting from the input layer, layer by layer, to the output layer. The method assumes that lower and upper bounds for the input neurons are provided a priori, and then uses a linear combination of the bounds of neurons from one layer to compute lower and upper bounds for neurons in the following layer. The linear combination is decided according to the weights of the edges that connect the two layers.

As an example, we show how to compute the bounds for neuron $n_{3,1}$ in the original network N. The range of the input neuron $n_{1,1}$ is $[20, 21]$. The range of the possible values of $n_{2,1}$ is $[20, 21] \cdot 10 = [200, 210]$, since its only input edge comes from $n_{1,1}$ and its weight is 10. Similarly, the possible values of $n_{2,2}$ are $[20, 21]$ since its only input edge comes from $n_{1,1}$ and its weight is 1. Moving to the output layer, the range of possible values of $n_{3,1}$ is calculated using the weighted sum of the ranges of $n_{2,1}$ and $n_{2,2}$: the lower bound is $200 \cdot 3 + 20 \cdot 4 = 680$, and the upper bound is $210 \cdot 3 + 21 \cdot 4 = 714$. Similarly, by propagating the input range to the output range in N' we get that the output range of $n_{3,1}$ in N' is $[1400, 1470]$. Therefore, the minimal difference is bounded by $d = N'(20) - N(21) = 1400 - 714 = 686$. Now, d can be used in updating the verification query into the over-approximate verification query $\langle N', n_{1,1} \in [20, 21], n_{3,1} < 800 + 686 = 1486 \rangle$, which is exactly v_2 above.

5 Implementation and Evaluation

Implementation. For evaluation purposes, we created a proof-of-concept implementation of `CEGARETTE`, referred to as `CEGARETTE-NN`, where the `queryAbstract` and `queryRefine` operations are implemented according to Algorithms 3 and 5, respectively. The neuron merging and splitting operations, `abstract` and `refine`, were taken from the publicly released code of [27], which implements the methods proposed in [28].

As our tool's underlying verification engine, we used the Marabou framework [48] (although other tools could be used, instead). Marabou is a sound and complete verifier [48, 84] that runs a Simplex solver at its core [46, 47], combined with abstract-interpretation enhancements [28, 66, 75, 82, 90], proof-production capabilities [39], advanced splitting heuristics [85], optimization techniques [78], and support for various activation functions [6]. The framework has previously been applied to perform various tasks, such as DNN repair [32, 71], explainability [14], reinforcement learning verification [2, 4, 5, 29], DNN simplification [31, 53], and DNN ensemble selection [7] and industrial needs [3].

For the lower and upper bound computation performed in Algorithm 4, we implemented the interval bound propagation (IBP) method proposed in [34]; and also used the symbolic bound tightening (SBT) method [82] and the DeepPoly method [75], both of which were already implemented as part of the Marabou engine [48]. Our tool is implemented in Python and is publicly available online, along with all benchmarks used in our evaluation.[1] All experiments reported below were conducted on x86-64 Gnu/Linux-based machines, using a single Intel(R) Xeon(R) Gold 6130 CPU @ 2.10GHz core.

Benchmarks. We conducted extensive experiments using two sets of benchmarks: ACAS-Xu [44] and MNIST [54].

ACAS-Xu is a set of 45 DNNs intended to operate as an airborne collision avoidance system. Each of these networks receives sensor information regarding the aircraft's trajectory and velocity, as well as those of other aircraft nearby; and produces a horizontal turning advisory, intended to reduce the chance of airborne collision. Each of these networks consists of an input layer with 5 neurons, followed by 6 hidden layers with 50 neurons each, and a final output layer with 5 additional neurons — yielding a total of 310 neurons.

For specifications, we used adversarial robustness queries, which are the de facto standard for DNN verification [12]. Each such query specifies an input point x_0 and a radius δ and states that any point within the δ-ball around x_0 must produce the same classification as x_0. Here, we used 20 previously proposed adversarial robustness properties [28], each with a 2-hour timeout.

For the second family of benchmarks, we used the MNIST dataset of grayscale images of hand-written digits between 0 and 9. We used 60000 images to train 3 different networks, whose topologies are listed in Table 1. These networks achieved high accuracy rates (although not as high as the state of the art [61]) on 10000 test images, as detailed in the "Accuracy" row in Table 1. For these networks, we again used adversarial robustness queries. Specifically, we selected 30 input points that were sampled uniformly at random. For δ, we used the values $\delta \in \{1e^{-3}, 1e^{-2}, 2e^{-2}, 5e^{-2}, 7e^{-2}, 9e^{-2}, 1e^{-1}\}$. For each candidate query, i.e., a pair of input point x_0 and a radius δ, we sampled 10000 random inputs in the δ-ball around x_0, and ensured that they were all correctly classified. If any random sample was misclassified, we discarded the query without verification (this was done in order to filter out very simple queries, where formal verification is not needed). After this filtering, the total number of queries remaining for each of the networks was 653, 797, and 560. Because these networks were more complex

[1] https://github.com/yizhake/cegarette_nn

Table 1: Network sizes and properties.

	Acas Xu	**MNIST 1**	**MNIST 2**	**MNIST 3**
Inputs	5	784	784	784
Outputs	5	10	10	10
Hidden layers	6	5	6	15
Total hidden neurons	300	144	320	224
Accuracy		96.94	97.13	95.43

Table 2: Comparing Bound propagation methods on the ACAS-Xu benchmarks.

	`IBP`	`SBT`	`DeepPoly`
#Finished	397	**851**	833
#Timeouts	463	**49**	67

than the ACAS Xu case, we set an arbitrary timeout value of 3 days for each query. The actual encoding as a Marabou query was performed according to the definition of *standard robustness* (Definition 2, [20]).

Evaluation. Recall that our approach depends on our ability to compute tight lower and upper output bounds as part of Algorithm 4. Multiple methods have been proposed for computing such bounds, with varying degrees of accuracy — and with the more accurate ones typically taking longer to run. Thus, in our first experiment, we set out to compare the different bound propagation approaches, namely `IBP`, `SBT`, and `DeepPoly`, and measure their usefulness as part of our framework.

Table 2 depicts the results obtained using the three approaches when applied to the ACAS-Xu benchmarks. `IBP`, which is the most lightweight but also the least precise among the approaches, performed the worst — leading to the highest number of timeouts. This indicates that the bounds it computed were fairly loose, triggering large sequences of spurious examples and refinement steps, eventually leading to the timeouts. `DeepPoly`, which is the most precise among the three but also the most computationally expensive, achieved better bounds, and consequently fewer timeouts. `SBT`, which is not as precise as `DeepPoly` but which is quicker to run, obtained the best results. Thus, we selected `SBT` as the best configuration for our tool and used it in the remaining experiments.

In our second experiment, we set out to measure the overall improvement afforded by our approach. Since it has already been established that abstraction-refinement often improves over direct verification [11,27,28,69], and because our approach extends the `CEGAR-NN` approach [28], we used `CEGAR-NN` as our baseline.

The results of comparing the two tools on all benchmark sets are displayed in Table 3. Most importantly, the results in the *Timeouts* and *Finished* columns confirm that `CEGARETTE-NN` significantly improves over `CEGAR-NN` in terms of finished experiments: a total of 2674 for `CEGARETTE-NN`, versus 1618 for `CEGAR-NN` — a 65% improvement. Because both tools use the same underlying verifier, and apply the same basic abstraction and refinement operators, this improvement stems directly from the property abstraction mechanism, and its ability to reduce the number of spurious counter-examples.

The next column, *Faster Verification Time*, counts the number of instances in which one

Table 3: Comparing CEGARETTE-NN and CEGAR-NN.

		Timeout	Finished	Faster Verification Time	Fewer Refinement Steps
ACAS-Xu	CEGAR-NN	501	397	**272**	**73**
	CEGARETTE-NN	**49**	**851**	115	2
MNIST-1	CEGAR-NN	268	396	127	1
	CEGARETTE-NN	**88**	**576**	**252**	**283**
MNIST-2	CEGAR-NN	716	230	83	0
	CEGARETTE-NN	**228**	**718**	**134**	**133**
MNIST-3	CEGAR-NN	**101**	**595**	195	6
	CEGARETTE-NN	167	529	**263**	**51**
TOTAL	CEGAR-NN	1506	1618	677	80
	CEGARETTE-NN	**532**	**2674**	**764**	**469**

tool outperformed the other. It shows that CEGARETTE-NN achieves an improvement of 12.85% over CEGAR-NN. Finally, the *Fewer Refinement Steps* column counts the number of experiments in which the total number of refinement steps was smaller, and shows that CEGARETTE-NN achieves an improvement of 586.25%. We note that all comparison metrics that we used (except *Timeouts*) consider only those benchmarks that both tools successfully solved. We also mention that, as with all SMT solvers, slight changes to the input can result in widely different search paths and runtimes, as demonstrated, e.g., in the case of the MNIST-3 network.

The graphs in Figure 3 show how many experiments were completed by each method at different time points. ACAS-Xu/MNIST (MNIST-1, MNIST-2 and MNIST-3 together) results are represented in the left/right graph respectively, and indicate that CEGARETTE-NN is close in performance to CEGAR-NN at short time constants, while achieving a significant advantage when the threshold time is extended.

6 Related Work

There are two main approaches to validating the robustness of deep neural networks. The first combines methods of dynamic analysis and heuristic search to find violations and test the system (e.g. [19, 68, 72, 80]). The second, which we follow in this work, is that of formal verification, where correctness is established using a rigorous, automated procedure [56].

Several DNN verification techniques have been studied in recent years, based on multiple approaches: SMT-based techniques, such as Marabou [48], Reluplex [46], and others [38, 49]; techniques based on mixed integer programming, including Planet [26], MIPVerify [81] and others [17, 18, 24, 25]; symbolic interval propagation techniques [82]; abstract interpretation techniques [30]; and many others (e.g., [8, 41, 52, 59, 64, 84, 86]). Our approach can be used to extend various abstraction and refinement mechanisms, and be integrated with many sound and complete DNN verifiers as backends. Incomplete verification techniques could also be used as part of our approach, potentially improving performance but at the cost of incompleteness; we leave this for future work.

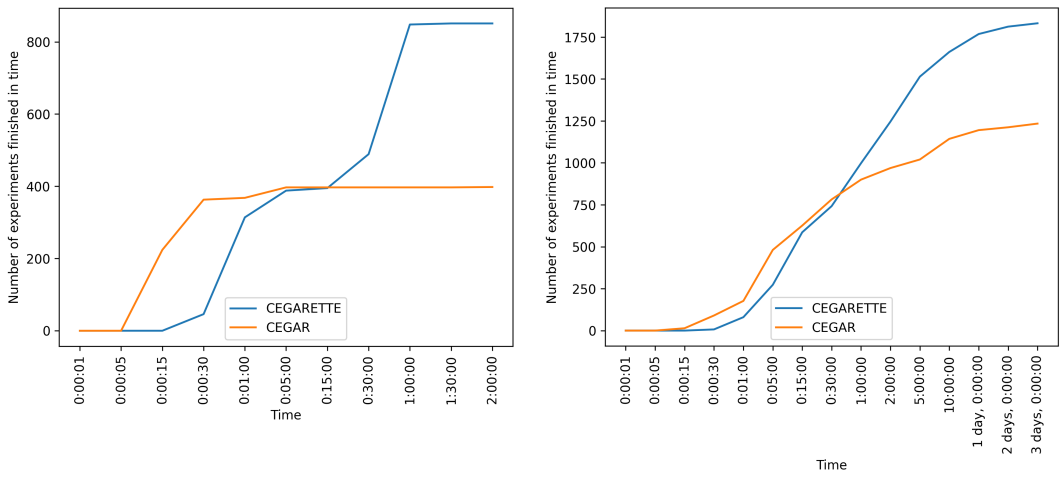

Figure 3: Comparing the numbers of finished experiments over time between `CEGARETTE-NN` and `CEGAR-NN` on ACAS-Xu (left) and MNIST (right) datasets.

There have been multiple attempts to utilize abstract interpretation [22] to decrease the complexity of DNN verification, and expedite the verification process [30,75,89]. These methods capture the behavior of propagated values in the network using abstract domains such as boxes, zonotopes, or polyhedra. Often, these methods rely on coarse over-approximations, and are incomplete; although various refinement methods, as well as integration with complete verifiers, have been proposed to mitigate this.

Apart from the DNN abstraction technique that we leveraged here [28], other incomplete abstraction techniques have been proposed [11]. These manipulate the neurons and the edges of the network, using semantic similarity; utilize clustering methods in order to merge similar neurons; or merge neurons and compute ranges of weights and biases that the merged neurons can take. Integrating our framework with these additional approaches should be possible, and is left for future work.

A few recent papers proposed to instrument CEGAR-based approaches for DNN verification [28,57]. The network is preprocessed such that, later on, multiple neurons can be merged while causing a strict increase in the output values, hence over-approximating the original network. Refinement is done by splitting past-merged neurons. Both are oblivious to the underlying verification engine as long as it is sound and complete.

The work in [27] uses residual reasoning to optimize CEGAR-based approaches, given that the backend applies case splitting. Another independent optimization for the CEGAR-based approach for neural network verification was recently proposed in [93], where testing methods are embedded during the formal verification process in order to quickly expose violations using adversarial attacks.

As far as we know, our work is the first formal verification scheme which extends the basic mechanism of CEGAR [21] by applying abstraction and refinement not only to the checked system but also to the (output) property. Therefore, it should be compatible with any of the aforementioned techniques.

7 Conclusion

Neural networks are gaining momentum in many areas, but using them entails various risks. Neural network verification tools seek to help overcome this issue, but are computationally expensive — and afford only limited scalability. Abstraction-based approaches hold great potential for expediting the verification process and allowing verifiers to scale to much larger networks. Here, we took a step in this direction, by applying abstraction and refinement not only to neural networks themselves but also to the properties being checked as part of the verification query. We demonstrated that our method dramatically improves performance, by reducing the number of spurious counter-examples encountered. The result is a significant boost to the scalability of existing verification technology.

Moving forward, we plan to pursue several research directions. First, we plan to explore additional techniques for property tightening given an abstract neural network. Second, we intend to develop novel abstraction and refinement heuristics, which are optimized for `CEGARETTE-NN` — i.e., which will allow better property tightening. Third, we plan to integrate our method with additional, recently-proposed CEGAR-based approaches [11, 69].

Acknowledgements. We thank Orna Kupferman and Orna Grumberg for their insightful comments. This project was partially supported by grants from the National Science Foundation (2211505), the Binational Science Foundation (2021769), and the Israel Science Foundation (683/18).

References

[1] M. Alam, M. Samad, L. Vidyaratne, A. Glandon, and K. Iftekharuddin. Survey on Deep Neural Networks in Speech and Vision Systems. *Neurocomputing*, 417:302–321, 2020.

[2] G. Amir, D. Corsi, R. Yerushalmi, L. Marzari, D. Harel, A. Farinelli, and G. Katz. Verifying Learning-Based Robotic Navigation Systems. In *Proc. 29th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 607–627, 2023.

[3] G. Amir, Z. Freund, G. Katz, E. Mandelbaum, and I. Refaeli. veriFIRE: Verifying an Industrial, Learning-Based Wildfire Detection System. In *Proc. 25th Int. Symposium on Formal Methods (FM)*, pages 648–656, 2023.

[4] G. Amir, O. Maayan, T. Zelazny, G. Katz, and M. Schapira. Verifying Generalization in Deep Learning. In *Proc. 34th Int. Conf. on Computer Aided Verification (CAV)*, 2023.

[5] G. Amir, M. Schapira, and G. Katz. Towards Scalable Verification of Deep Reinforcement Learning. In *Proc. 21st Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD)*, pages 193–203, 2021.

[6] G. Amir, H. Wu, C. Barrett, and G. Katz. An SMT-Based Approach for Verifying Binarized Neural Networks. In *Proc. 27th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 203–222, 2021.

[7] G. Amir, T. Zelazny, G. Katz, and M. Schapira. Verification-Aided Deep Ensemble Selection. In *Proc. 22nd Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD)*, pages 27–37, 2022.

[8] G. Anderson, S. Pailoor, I. Dillig, and S. Chaudhuri. Optimization and Abstraction: a Synergistic Approach for Analyzing Neural Network Robustness. In *Proc. 40th ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI)*, pages 731–744, 2019.

[9] Z. Andraus, M. Liffiton, and K. Sakallah. CEGAR-Based Formal Hardware Verification: a Case Study. *Ann Arbor*, 1001, 2007.

[10] E. Apostolidis, E. Adamantidou, A. Metsai, V. Mezaris, and I. Patras. Video Summarization using Deep Neural Networks: A Survey, 2021. Technical Report. http://arxiv.org/abs/2101.06072.

[11] P. Ashok, V. Hashemi, J. Kretinsky, and S. Mühlberger. DeepAbstract: Neural Network Abstraction for Accelerating Verification. In *Proc. 18th Int. Symposium on Automated Technology for Verification and Analysis (ATVA)*, pages 92–107, 2020.

[12] S. Bak, C. Liu, and T. Johnson. The Second International Verification of Neural Networks Competition (VNN-COMP 2021): Summary and Results, 2021. Technical Report. http://arxiv.org/abs/2109.00498.

[13] G. Barbastathis, A. Ozcan, and G. Situ. On the use of Deep Learning for Computational Imaging. *Optica*, 6:921–943, 2019.

[14] S. Bassan and G. Katz. Towards Formal XAI: Formally Approximate Minimal Explanations of Neural Networks. In *Proc. 29th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 187–207, 2023.

[15] D. Beyer and T. Lemberger. Symbolic Execution with CEGAR. In *Proc. 7th Int. Symposium on Leveraging Applications of Formal Methods (ISoLA)*, pages 195–211, 2016.

[16] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba. End to End Learning for Self-Driving Cars, 2016. Technical Report. http://arxiv.org/abs/1604.07316.

[17] R. Bunel, J. Lu, I. Turkaslan, P. Torr, P. Kohli, and M. Kumar. Branch and Bound for Piecewise Linear Neural Network Verification, 2019. Technical Report. https://arxiv.org/abs/1909.06588.

[18] R. Bunel, I. Turkaslan, P. Torr, P. Kohli, and M. Kumar. Piecewise Linear Neural Network Verification: A Comparative Study, 2017. Technical Report. https://arxiv.org/abs/1711.00455v1.

[19] N. Carlini and D. Wagner. Towards Evaluating the Robustness of Neural Networks, 2016. Technical

Report. http://arxiv.org/abs/1608.04644.

[20] M. Casadio, E. Komendantskaya, M. Daggitt, W. Kokke, G. Katz, G. Amir, and I. Refaeli. Neural Network Robustness as a Verification Property: A Principled Case Study. In *Proc. 34th Int. Conf. on Computer Aided Verification (CAV)*, pages 219–231, 2022.

[21] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-Guided Abstraction Refinement. In *Proc. 12th Int. Conf. on Computer Aided Verification (CAV)*, pages 154–169, 2010.

[22] P. Cousot and R. Cousot. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Proc. 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL)*, pages 238–252, 1977.

[23] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2018. Technical Report. http://arxiv.org/abs/1810.04805.

[24] S. Dutta, X. Chen, S. Jha, S. Sankaranarayanan, and A. Tiwari. Sherlock — A tool for Verification of Neural Network Feedback Systems. In *Proc. 22nd ACM Int. Conf. on Hybrid Systems: Computation and Control (HSCC)*, pages 262–263, 2019.

[25] S. Dutta, S. Jha, S. Sanakaranarayanan, and A. Tiwari. Output Range Analysis for Deep Neural Networks. In *Proc. 10th NASA Formal Methods Symposium (NFM)*, pages 121–138, 2018.

[26] R. Ehlers. Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks. In *Proc. 15th Int. Symp. on Automated Technology for Verification and Analysis (ATVA)*, pages 269–286, 2017.

[27] Y. Y. Elboher, E. Cohen, and G. Katz. Neural Network Verification using Residual Reasoning. In *Proc. 20th Int. Conf. on Software Engineering and Formal Methods (SEFM)*, pages 173–189, 2022.

[28] Y. Y. Elboher, J. Gottschlich, and G. Katz. An Abstraction-Based Framework for Neural Network Verification. In *Proc. 32nd Int. Conf. on Computer Aided Verification (CAV)*, pages 43–65, 2020.

[29] T. Eliyahu, Y. Kazak, G. Katz, and M. Schapira. Verifying Learning-Augmented Systems. In *Proc. Conf. of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, pages 305–318, 2021.

[30] T. Gehr, M. Mirman, D. Drachsler-Cohen, E. Tsankov, S. Chaudhuri, and M. Vechev. AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation. In *Proc. 39th IEEE Symposium on Security and Privacy (S&P)*, 2018.

[31] S. Gokulanathan, A. Feldsher, A. Malca, C. Barrett, and G. Katz. Simplifying Neural Networks using Formal Verification. In *Proc. 12th NASA Formal Methods Symposium (NFM)*, pages 85–93, 2020.

[32] B. Goldberger, Y. Adi, J. Keshet, and G. Katz. Minimal Modifications of Deep Neural Networks using Verification. In *Proc. 23rd Int. Conf. on Logic for Programming, Artificial Intelligence and Reasoning (LPAR)*, 2020.

[33] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[34] S. Gowal, K. Dvijotham, R. Stanforth, R. Bunel, C. Qin, J. Uesato, R. Arandjelovic, T. Mann, and P. Kohli. On the Effectiveness of Interval Bound Propagation for Training Verifiably Robust Models, 2018. Technical Report. http://arxiv.org/abs/1810.12715.

[35] A. Gulati, J. Qin, C.-C. Chiu, N. Parmar, Y. Zhang, J. Yu, W. Han, S. Wang, Z. Zhang, Y. Wu, and R. Pang. Conformer: Convolution-Augmented Transformer for Speech Recognition, 2020. Technical Report. http://arxiv.org/abs/2005.08100.

[36] J. Guo and C. Liu. Practical Poisoning Attacks on Neural Networks. In *Proc. 22nd European Conf. on Computer Vision (ECCV)*, pages 142–158, 2020.

[37] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *Proc.*

IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), pages 770–778, 2016.

[38] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu. Safety Verification of Deep Neural Networks. In *Proc. 29th Int. Conf. on Computer Aided Verification (CAV)*, pages 3–29, 2017.

[39] O. Isac, C. Barrett, M. Zhang, and G. Katz. Neural Network Verification with Proof Production. In *Proc. 22nd Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD)*, pages 38–48, 2022.

[40] O. Isac, Y. Zohar, C. Barrett, and G. Katz. DNN Verification, Reachability, and the Exponential Function Problem, 2023. Technical Report. https://arxiv.org/abs/2305.06064.

[41] Y. Jacoby, C. Barrett, and G. Katz. Verifying Recurrent Neural Networks using Invariant Inference. In *Proc. 18th Int. Symposium on Automated Technology for Verification and Analysis (ATVA)*, pages 57–74, 2020.

[42] M. Janota, W. Klieber, J. Marques-Silva, and E. Clarke. Solving QBF with Counterexample Guided Refinement. *Artificial Intelligence*, 234:1–25, 2016.

[43] J. Johnson and T. Khoshgoftaar. Survey on Deep Learning with Class Imbalance. *Journal of Big Data*, 6(1):1–54, 2019.

[44] K. Julian, J. Lopez, J. Brush, M. Owen, and M. Kochenderfer. Policy Compression for Aircraft Collision Avoidance Systems. In *Proc. 35th Digital Avionics Systems Conf. (DASC)*, pages 1–10, 2016.

[45] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-Scale Video Classification with Convolutional Neural Networks. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 1725–1732, 2014.

[46] G. Katz, C. Barrett, D. Dill, K. Julian, and M. Kochenderfer. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *Proc. 29th Int. Conf. on Computer Aided Verification (CAV)*, pages 97–117, 2017.

[47] G. Katz, C. Barrett, D. Dill, K. Julian, and M. Kochenderfer. Reluplex: a Calculus for Reasoning about Deep Neural Networks. *Formal Methods in System Design (FMSD)*, 2021.

[48] G. Katz, D. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljić, D. Dill, M. Kochenderfer, and C. Barrett. The Marabou Framework for Verification and Analysis of Deep Neural Networks. In *Proc. 31st Int. Conf. on Computer Aided Verification (CAV)*, pages 443–452, 2019.

[49] Y. Kazak, C. Barrett, G. Katz, and M. Schapira. Verifying Deep-RL-Driven Systems. In *Proc. 1st ACM SIGCOMM Workshop on Network Meets AI & ML (NetAI)*, 2019.

[50] Z. Kolter and E. Wong. Provable Defenses Against Adversarial Examples via the Convex Outer Adversarial Polytope. In *Proc. 16th IEEE Int. Conf. on Machine Learning and Applications (ICML)*, 2018.

[51] D. Krishnamurthy, R. Stanforth, S. Gowal, T. Mann, and P. Kohli. A Dual Approach to Scalable Verification of Deep Networks, 2018. Technical Report. http://arxiv.org/abs/1803.06567.

[52] L. Kuper, G. Katz, J. Gottschlich, K. Julian, C. Barrett, and M. Kochenderfer. Toward Scalable Verification for Safety-Critical Deep Networks, 2018. Technical Report. https://arxiv.org/abs/1801.05950.

[53] O. Lahav and G. Katz. Pruning and Slicing Neural Networks using Formal Verification. In *Proc. 21st Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD)*, pages 183–192, 2021.

[54] Y. LeCun. The MNIST Database of Handwritten Digits, 1998. http://yann.lecun.com/exdb/mnist/.

[55] T. Lee, S. Mckeever, and J. Courtney. Flying Free: A Research Overview of Deep Learning in Drone Navigation Autonomy. *Drones*, 5(2), 2021.

[56] C. Liu, T. Arnon, C. Lazarus, C. Barrett, and M. Kochenderfer. Algorithms for Verifying Deep Neural Networks, 2019. Technical Report. http://arxiv.org/abs/1903.06758.

[57] J. Liu, Y. Xing, X. Shi, F. Song, Z. Xu, and Z. Ming. Abstraction and Refinement: Towards

Scalable and Exact Verification of Neural Networks, 2022. Technical Report. https://arxiv.org/abs/2207.00759.

[58] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. Alsaadi. A Survey of Deep Neural Network Architectures and their Applications. *Neurocomputing*, 234:11–26, 2017.

[59] A. Lomuscio and L. Maganti. An Approach to Reachability Analysis for Feed-Forward ReLU Neural Networks, 2017. Technical Report. https://arxiv.org/abs/1706.07351.

[60] T. Malekzadeh, M. Abdollahzadeh, H. Nejati, and N.-M. Cheung. Aircraft Fuselage Defect Detection using Deep Neural Networks, 2017. Technical Report. http://arxiv.org/abs/1712.09213.

[61] V. Mazzia, F. Salvetti, and M. Chiaberge. Efficient-CapsNet: capsule network with self-attention routing. *Scientific Reports*, 11(1), jul 2021.

[62] M. Müller, G. Makarchuk, G. Singh, M. Püschel, and M. Vechev. PRIMA: General and Precise Neural Network Certification via Scalable Convex Hull Approximations. In *Proc. 49th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL)*, 2022.

[63] V. Nair and G. Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proc. 27th Int. Conf. on Machine Learning (ICML)*, pages 807–814, 2010.

[64] N. Narodytska, S. Kasiviswanathan, L. Ryzhyk, M. Sagiv, and T. Walsh. Verifying Properties of Binarized Deep Neural Networks, 2017. Technical Report. http://arxiv.org/abs/1709.06662.

[65] J. Nellen, E. Ábrahám, and B. Wolters. A CEGAR Tool for the Reachability Analysis of PLC-Controlled Plants Using Hybrid Automata. In *Proc. 3rd IEEE Int. Workshop on Formal Methods Integration (FMi)*, pages 55–78, 2015.

[66] M. Ostrovsky, C. Barrett, and G. Katz. An Abstraction-Refinement Approach to Verifying Convolutional Neural Networks. In *Proc. 20th. Int. Symposium on Automated Technology for Verification and Analysis (ATVA)*, 2022.

[67] C. Paterson, H. Wu, J. Grese, R. Calinescu, C. Pasareanu, and C. Barrett. DeepCert: Verification of Contextually Relevant Robustness for Neural Network Image Classifiers. In *Proc. 40th Int. Comf. on Computer Safety, Reliability, and Security (SAFECOMP)*, pages 3–17, 2021.

[68] K. Pei, Y. Cao, J. Yang, and S. Jana. DeepXplore: Automated Whitebox Testing of Deep Learning Systems. *Communications of the ACM (CACM)*, pages 137–145, 2019.

[69] P. Prabhakar and Z. Afzal. Abstraction based Output Range Analysis for Neural Networks, 2020. Technical Report. http://arxiv.org/abs/2007.09527.

[70] L. Pulina and A. Tacchella. An Abstraction-Refinement Approach to Verification of Artificial Neural Networks. In *Proc. 22nd Int. Conf. on Computer Aided Verification (CAV)*, pages 243–257, 2010.

[71] I. Refaeli and G. Katz. Minimal Multi-Layer Modifications of Deep Neural Networks. In *Proc. 5th Workshop on Formal Methods for ML-Enabled Autonomous Systems (FoMLAS)*, 2022.

[72] K. Ren, T. Zheng, Z. Qin, and X. Liu. Adversarial Attacks and Defenses in Deep Learning. *Engineering*, 6(3):346–360, 2020.

[73] H. Riener, R. Ehlers, and G. Fey. CEGAR-Based EF Synthesis of Boolean Functions with an Application to Circuit Rectification. In *Proc. 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 251–256, 2017.

[74] S. Shamshirband, M. Fathi, A. Dehzangi, A. Chronopoulos, and H. Alinejad-Rokny. A Review on Deep Learning Approaches in Healthcare Systems: Taxonomies, Challenges, and Open Issues. *Journal of Biomedical Informatics*, 113, 2021.

[75] G. Singh, T. Gehr, M. Puschel, and M. Vechev. An Abstract Domain for Certifying Neural Networks. In *Proc. 46th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL)*, 2019.

[76] M. Sotoudeh and A. Thakur. Abstract Neural Networks, 2020. Technical Report. http://arxiv.org/abs/2009.05660.

[77] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A Sim-

ple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[78] C. Strong, H. Wu, A. Zeljić, K. Julian, G. Katz, C. Barrett, and M. Kochenderfer. Global Optimization of Objective Functions Represented by ReLU Networks. *Journal of Machine Learning*, pages 1–28, 2021.

[79] H. Su, M. Chai, L. Chen, and J. Lv. Deep Learning-Based Model Predictive Control for Virtual Coupling Railways Operation. In *Proc. 24th IEEE Int. Intelligent Transportation Systems Conf (ITSC)*, pages 3490–3495, 2021.

[80] Y. Tian, K. Pei, S. Jana, and R. Baishakhi. DeepTest: Automated Testing of Deep-Neural-Network-driven Autonomous Cars. In *Proc. 40th Int. Conf. on Software Engineering (ICSE)*, pages 303–314, 2018.

[81] V. Tjeng, K. Xiao, and R. Tedrake. Evaluating Robustness of Neural Networks with Mixed Integer Programming. In *Proc. 7th Int. Conf. on Learning Representations (ICLR)*, 2019.

[82] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana. Formal Security Analysis of Neural Networks using Symbolic Intervals. In *Proc. 27th USENIX Security Symposium*, 2018.

[83] S. Wang, H. Zhang, K. Xu, X. Lin, S. Jana, C.-J. Hsieh, and Z. Kolter. Beta-CROWN: Efficient Bound Propagation with Per-Neuron Split Constraints for Complete and Incomplete Neural Network Verification. In *Proc. 35th Conf. on Neural Information Processing Systems (NeurIPS)*, 2021.

[84] H. Wu, A. Ozdemir, A. Zeljić, A. Irfan, K. Julian, D. Gopinath, S. Fouladi, G. Katz, C. Păsăreanu, and C. Barrett. Parallelization Techniques for Verifying Neural Networks. In *Proc. 20th Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD)*, pages 128–137, 2020.

[85] H. Wu, A. Zeljić, G. Katz, and C. Barrett. Efficient Neural Network Analysis with Sum-of-Infeasibilities. In *Proc. 28th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 143–163, 2022.

[86] W. Xiang, H.-D. Tran, and T. Johnson. Output Reachable Set Estimation and Verification for Multilayer Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)*, 99:1–7, 2018.

[87] H. Xu, Y. Ma, H. Liu, D. Deb, H. Liu, J. Tang, and A. Jain. Adversarial Attacks and Defenses in Images, Graphs and Text: A Review, 2019. Technical Report. http://arxiv.org/abs/1909.08072.

[88] K. Xu, Z. Shi, H. Zhang, Y. Wang, K.-W. Chang, M. Huang, B. Kailkhura, X. Lin, and C.-J. Hsieh. Automatic Perturbation Analysis for Scalable Certified Robustness and Beyond. In *Proc. Advances in Neural Information Processing Systems*, pages 1129–1141, 2020.

[89] P. Yang, R. Li, J. Li, C.-C. Huang, J. Wang, J. Sun, B. Xue, and L. Zhang. Improving Neural Network Verification through Spurious Region Guided Refinement, 2020. Technical Report. http://arxiv.org/abs/2010.07722.

[90] T. Zelazny, H. Wu, C. Barrett, and G. Katz. On Reducing Over-Approximation Errors for Neural Network Verification. In *Proc. 22nd Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD)*, pages 17–26, 2022.

[91] C. Zhang, O. Vinyals, R. Munos, and S. Bengio. A Study on Overfitting in Deep Reinforcement Learning, 2018. Technical Report. http://arxiv.org/abs/1804.06893.

[92] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel. Efficient Neural Network Robustness Certification with General Activation Functions, 2018. Technical Report. http://arxiv.org/abs/1811.00866.

[93] Z. Zhao, Y. Zhang, G. Chen, F. Song, T. Chen, and J. Liu. CLEVEREST: Accelerating CEGAR-based Neural Network Verification via Adversarial Attacks. In *Proc. 29th Static Analysis Symposium (SAS)*, 2022.