Tightening Robustness Verification of Convolutional Neural Networks with Fine-Grained Linear Approximation

Yiting Wu,[1] **Min Zhang**[1,2]

[1] Shanghai Key Laboratory for Trustworthy Computing, East China Normal University
[2] Shanghai Institute of Intelligent Science and Technology, Tongji University
51205902026@stu.ecnu.edu.cn, zhangmin@sei.ecnu.edu.cn

Abstract

The robustness of neural networks can be quantitatively indicated by a lower bound within which any perturbation does not alter the original input's classification result. A certified lower bound is also a criterion to evaluate the performance of robustness verification approaches. In this paper, we present a tighter linear approximation approach for the robustness verification of Convolutional Neural Networks (CNNs). By the tighter approximation, we can tighten the robustness verification of CNNs, i.e., proving they are robust within a larger perturbation distance. Furthermore, our approach is applicable to general sigmoid-like activation functions. We implement DeepCert, the resulting verification toolkit. We evaluate it with open-source benchmarks, including LeNet and the models trained on MNIST and CIFAR. Experimental results show that DeepCert outperforms other state-of-the-art robustness verification tools with at most 286.3% improvement to the certified lower bound and 1566.8 times speedup for the same neural networks.

Introduction

Robustness is becoming crucial to neural networks with the successful application of deep learning to safety-critical domains such as self-driving (Gopinath et al. 2018) and access control by face and voiceprint recognition (Goswami et al. 2018; Shen et al. 2018). AI systems powered by non-robust neural networks are vulnerable and fragile to the perturbation from the environment and adversarial attack (Moosavi-Dezfooli, Fawzi, and Frossard 2016; Fawzi et al. 2017). Therefore, the robustness must be certified before a neural network is deployed (Balunovic et al. 2019).

Formal methods, which have been proved successful in verifying traditional software and hardware (Woodcock et al. 2009), also show its capability to verify the robustness of neural networks (Huang et al. 2017; Katz et al. 2017). A comprehensive survey on robustness verification using formal methods can be referred to (Huang et al. 2020). Most existing robustness verification approaches fall into two classes. One is to check whether a given neural network is robust or not with respect to a concrete input and a perturbation distance. An adversarial example is usually generated as a witness in the non-robustness case. The other is to compute a lower bound for a neural network and an input. All the perturbed inputs around the original input within the computed lower bound have the same classification result as the original input. A certified lower bound quantitatively describes the robustness of a neural network.

Computing a certified lower bound for neural networks is a computationally expensive task, e.g., NP-complete even for ReLU-based networks (Katz et al. 2017). Thus, acceleration techniques based on approximation or abstraction are necessary to achieve scalability (Singh et al. 2019b; Wan et al. 2020). There is a trade-off between the precise of computed lower bound and the scalability (Wong et al. 2018). In general, the tighter an approximation is, the more precise lower bound it can compute while consuming more resources, i.e., time and memory (Lyu et al. 2020). Therefore, besides efficiency, certified lower bound is also a criterion to evaluate robustness verification approaches' performances.

This paper proposes a fine-grained linear approximation approach to the robustness verification of Convolutional Neural Networks (CNNs). The essence of linear approximation is to determine an upper-bound linear constraint and a lower-bound linear constraint to approximate a non-linear function $f(x)$ between an interval $[l, u]$. In our approach, to achieve a tighter approximation, we calculate the slopes of the two linear constraints according to the value of l, u, and the property of $f(x)$ between $[l, u]$, such as convexity and monotonicity of derivatives. Another advantage of our approach is that its computational complexity is constant time, and consequently the scalability of overall verification only depends on LP solving. By the tighter approximation, we can tighten the robustness verification of CNNs by proving neural networks are robust within a greater lower bound of perturbation. Furthermore, the approximation approach is general and applicable to sigmoid-like activation functions.

We implement the fine-grained approximation approach atop the state-of-the-art robustness verification framework CNN-Cert (Boopathy et al. 2019). We call this extended version DeepCert. We evaluate it with open-source benchmarks including LeNet models (LeCun et al. 1998) and neural networks trained on MNIST and CIFAR datasets. Experimental results show that DeepCert outperforms relevant tools such as CNN-Cert, CROWN (Zhang et al. 2018) and its extended version FROWN (Lyu et al. 2020) with 76.6%, 286.3%, 252.8% improvement to the certified lower bound, respec-

tively. Meanwhile, DeepCert has better scalability with up to 1.2, 3.7 and 1566.8 times speedup compared with CNN-Cert, CROWN, and FROWN, respectively.

In summary, this work follows the previous sequel of robustness verification approaches based on linear approximations and makes the following two major contributions:

1. A fine-grained linear approximation approach for computing a larger robust lower bound of CNNs with general activation functions.
2. A verification toolkit DeepCert which outperforms the state-of-the-art tools with at most 286.3% improvement to the certified lower bound and 1566.8 times speedup.

Related Work. There have been many efforts made towards the robustness verification of neural networks. Most are focused on ReLU-based fully-connected neural networks for the simplicity of network architecture and the piece-wise linearity of the ReLU activation function (Liu et al. 2019). It can be equivalently reduced to a mixed integer linear programming (MILP) problem (Tjeng et al. 2019), an satisfiability modulo theory (SMT) based problem (Katz et al. 2017), and some other abstracted models for better scalability (Weng et al. 2018; Botoeva et al. 2020).

There is some recent work on the verification of more complex networks such as CNN and those that contain more complex activation functions such as Sigmoid, Tanh, and Atan. The pioneering work (Szegedy et al. 2014) first proposed an approximation-based approach to generate adversarial examples for neural networks with general activation functions. CROWN (Zhang et al. 2018) is the first framework for efficiently certifying non-trivial robustness for general activation functions in neural networks. Recently, CROWN is extended to FROWN with a tighter approximation approach to compute a larger certified lower bound (Lyu et al. 2020). However, both the two tools are limited to fully-connected neural networks. CNN-Cert (Boopathy et al. 2019) is one of the state-of-the-art robustness verification frameworks which support both CNNs and non-ReLU activation functions. CNN-Cert's basic idea is to compute a certified lower bound by linear bounding techniques separately on the non-linear operations in neural networks.

Besides approximation, there are some other techniques employed for the robustness verification problem of general neural networks. For instance, the approaches (Mirman, Gehr, and Vechev 2018; Gehr et al. 2018; Singh et al. 2019a) based on abstract interpretation (Cousot 1996) are proposed by transforming neural networks into abstract domains that can be efficiently solved. Another approach uses the Lipschitz continuity feature of neural networks to analyze their reachability properties, including safety and robustness (Ruan, Huang, and Kwiatkowska 2018).

Preliminaries

This section briefly introduces some necessary preliminaries and notations to understand our approach. Specifically, we summarize the essence of the approximation-based approaches for computing certified lower bounds in the most relevant works CNN-Cert, CROWN and FROWN.

Notations. Let Φ denote an m-layer convolutional neural network. Its output of an input example x_0 is vector $\Phi(x_0)$. We want to certify the robustness of Φ near a selected input example x_0. $\forall t \in \{0, 1, ..., m\}$, we use $\phi^t(x)$ to represent the output of layer t, where $\phi^0(x) = x, \phi^m(x) = \Phi(x)$. In fully-connected layers, $\phi^t(x)$ is a vector; while in convolutional layers it is a tensor, which can be regarded as an array of matrices. We use σ to indicate the activation function used by Φ, W^t to indicate the connection weight (Weights) from layer $t-1$ to layer t, and b^t to indicate the bias of layer t. Both W^t and b^t are matrices in fully-connected layers, while in convolutional layers they are tensors.

Note that if layer t is a fully connected layer or a convolutional layer with an activation function, then $\phi^t(x)$ represents the output before the activation function is processed. It means that when layer t is a fully connected layer, then $\phi^t(x) = W^t\sigma(\phi^{t-1}(x)) + b^t$; when layer t is a convolutional layer, then $\phi^t(x) = W^t * \sigma(\phi^{t-1}(x)) + b^t$.

Let $\mathbb{B}_p(x_0, \epsilon)$ be an ϵ-bounded ℓ_p norm-ball, which is the set of all the vectors x such that $\|x - x_0\|_p \leq \epsilon$.

Definition 1 (Minimum adversarial distortion) *The minimum adversarial distortion $\epsilon_{\min}$ of a neural network Φ w.r.t. an input x_0 and ℓ_p norm-ball is the smallest perturbation $\|\delta\|_p$ such that* $\text{argmax}_i\Phi_i(x_0) \neq \text{argmax}_i\Phi_i(x_0 + \delta)$.

Apparently, for any $\epsilon \leq \epsilon_{min}$, all the inputs in $\mathbb{B}_p(x_0, \epsilon)$ are predicted by Φ to the same class as x_0 is. Computing ϵ_{min} is essentially an optimization problem, which is computationally expensive, e.g., NP-Complete even for the simple fully-connected ReLU-based neural networks (Katz et al. 2017). Therefore, it is more practical to compute a bound lower than ϵ_{min} and prove that all the inputs in the bounded norm-ball are classified to the same result as x_0 is.

Definition 2 (Certified lower bound) ϵ_{cert} *is called a certified lower bound of a neural network Φ w.r.t. an input x_0 and ℓ_p norm-ball if* $\text{argmax}_i\Phi_i(x) = \text{argmax}_i\Phi_i(x_0)$ *for all x in $\mathbb{B}_p(x_0, \epsilon_{\text{cert}})$.*

There are basically two ways of computing a non-trivial certified lower bound. One is to compute directly by reducing it to an optimization problem, which is a dual problem of computing ϵ_{min}. The other is to assume a concrete bound first and then check whether it satisfies the condition in Definition 2. Thus, it is reduced to a satisfiability problem. During the process of reduction, a linear approximation is usually employed to accelerate the decision procedure. The task of linear approximation during the encoding is to approximate the non-linear constraints using linear constraints, which can be solved in polynomial time. Let $\tilde{\Phi}$ denote a linear over-approximation of Φ. If the following formula holds

$$\forall x \in \mathbb{B}_p(x_0, \epsilon_{cert}).\text{argmax}_i\tilde{\Phi}_i(x) = \text{argmax}_i\tilde{\Phi}_i(x_0), \quad (1)$$

then, ϵ_{cert} is a certified lower bound for Φ w.r.t. x_0 and ℓ_p norm. Note that if the formula does not hold, we cannot conclude ϵ_{cert} is not a certified lower bound because a counterexample of the formula may not be a counterexample of the original formula in Definition 2 due to the approximation. In that case, one can try a smaller lower bound than ϵ_{cert}, or use refinement techniques (Wang et al. 2018a) for a tighter approximation based on the returned counterexample.

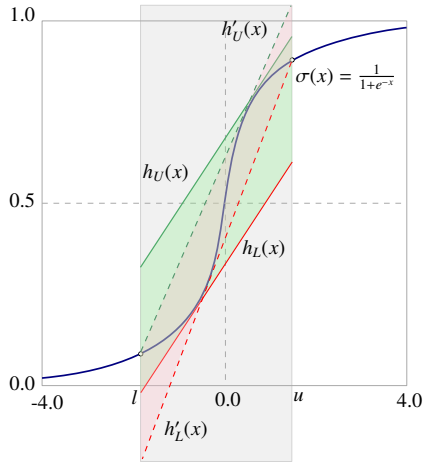
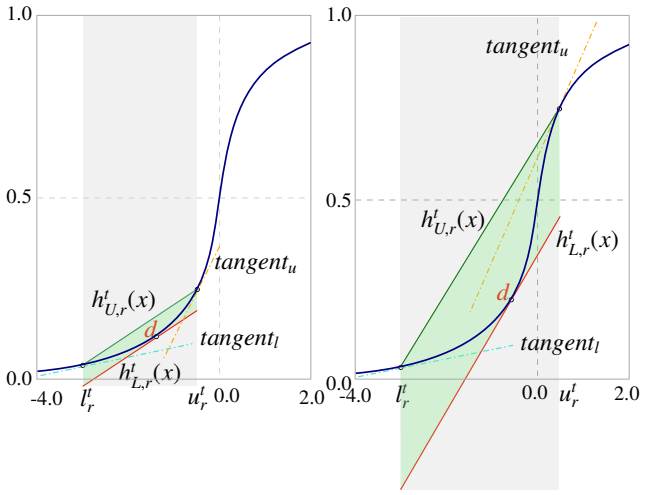


Figure 1: Linear approximation of $\sigma(x)$ between $[l, u]$



(a) $l_r^t < u_r^t \leq 0$ (b) $l_r^t < 0 < u_r^t$

Figure 2: Approximation when $\sigma'(l_r^t) < k$ and $\sigma'(u_r^t) > k$

Linear Approximation. The key idea of linear approximation is to determine an upper linear constraint and a lower linear constraint to enclose the non-linear constraint between an interval. Non-linear constraints arise from network architecture and activation functions. A CNN usually consists of convolutional layers, max-pooling layers, batch normalization layers and residual blocks, and general activation functions, all of which are non-linear. As for fully-connected neural networks, non-linearity only comes from activation functions. In our work, we make use of the linear approximation approach in CNN-Cert to deal with network layers and focus on the approximation of general activation functions. The readers who are interested in layer approximation can refer to the work (Boopathy et al. 2019) for the details.

Four types of activation functions are commonly used in neural networks, *i.e.,* ReLU, hyperbolic tangent, inverse tangent, and sigmoid. Except ReLU, we call other three functions sigmoid-like. We only focus on the linear approximation to them as ReLU has been well studied in literature.

Definition 3 (Linear upper/lower bounds) *Let $\sigma(x)$ be a non-linear function with $x \in [l, u]$, $\alpha_L, \alpha_U, \beta_L, \beta_U \in \mathbb{R}$, and*

$$h_U(x) = \alpha_U(x + \beta_U), \qquad h_L(x) = \alpha_L(x + \beta_L).$$

$h_U(x)$ and $h_L(x)$ are called linear upper and lower bounds of $\sigma(x)$ if for all x in $[l, u]$ there is $h_L(x) \leq \sigma(x) \leq h_U(x)$.

Figure 1 depicts an example of linear approximation of the sigmoid activation function $\sigma(x)$ with $x \in [l, u]$. We use $h_U(x)$ and $h_L(x)$ to represent the linear upper-bound and lower-bound. Note that $h_U(x)$ and $h_L(x)$ are not unique and do not necessarily have the same slope. For instance, $h'_U(x)$ and $h'_L(x)$ represented by the dashed green and red lines are two valid linear constraints for the approximation of $\sigma(x)$.

The central task of linear approximation is to determine $h_U(x)$ and $h_L(x)$, which directly affect the quality of certified lower bounds. The tighter the approximation is, the closer the computed lower bounded is to the optimal one. For instance, CNN-Cert takes $h'_U(x)$ and $h'_L(x)$ as the linear upper and lower bounds for the approximation in Figure 1, where $h'_U(x)$ and $h'_L(x)$ are two tangent lines of $\sigma(x)$ on the two points between l and u such that $h'_U(x)$ and $h'_L(x)$ cross the two endpoints $(l, \sigma(l))$ and $(u, \sigma(u))$, respectively. Apparently, the approximation using $h'_U(x)$ and $h'_L(x)$ is tighter than the one using $h_U(x)$ and $h_L(x)$.

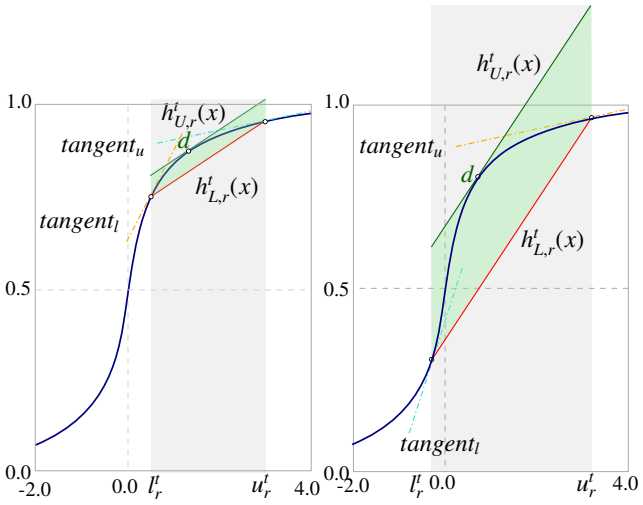
Fine-Grained Linear Approximation

Given a bounded norm-ball $\mathbb{B}_p(x_0, \epsilon_{\text{cert}})$, we assume that the upper and lower bounds of the output of layer t before activation are u_r^t and l_r^t, i.e., $l_r^t \leq \phi_r^t(x) \leq u_r^t$. Note that u_r^t and l_r^t are vectors consisting of constant scalars, which can be computed layer by layer using interval arithmetic (Moore, Kearfott, and Cloud 2009). To preserve variable dependence among layers, the intervals can be computed more precisely using symbolic interval propagation (Wang et al. 2018b).

We consider the linear approximation of $\sigma(\phi_r^t(x))$ between $[l_r^t, u_r^t]$, where σ is a sigmoid-like function. We compute the tangent lines $tangent_l$ and $tangent_u$ of $\sigma(\phi_r^t(x))$ at the two endpoints $(l_r^t, \sigma(l_r^t))$ and $(u_r^t, \sigma(u_r^t))$. Then, we check whether $tangent_l$ is above or below the point $(u_r^t, \sigma(u_r^t))$ and $tangent_u$ is above or below the point $(l_r^t, \sigma(l_r^t))$. It is equivalent to compare the slopes $\sigma'(l_r^t)$ and $\sigma'(u_r^t)$ with $k = \frac{\sigma(u_r^t) - \sigma(l_r^t)}{u_r^t - l_r^t}$.

Case 1. When $\sigma'(l_r^t) < k$ and $\sigma'(u_r^t) > k$, there are two sub-cases as shown in Figure 2, where $tangent_u$ (the orange dashed line) is below $(l_r^t, \sigma(l_r^t))$ and $tangent_l$ (the blue dashed line) is below $(u_r^t, \sigma(u_r^t))$. In the both two sub-cases, we choose the line in green that connects the starting point and the ending point as the linear upper bound, and the tangent line in red of $\sigma(x)$ at $x = d$ such that $\sigma'(d) = k$ as the linear lower bound. Namely, we have $h_{U,r}^t = k(x - l_r^t) + \sigma(l_r^t)$ and $h_{L,r}^t = k(x - d) + \sigma(d)$ with $\sigma'(d) = k$ and $l_r^t < d < u_r^t$.

Case 2. When $\sigma'(l_r^t) > k$ and $\sigma'(u_r^t) < k$, there are also two sub-cases as shown in Figure 3. Apparently, they are symmetries of the two sub-cases of Case 1. Similar to Case 1, we choose the line (in red) that connects the starting point and the ending point of $\sigma(x)$ in the interval as the linear lower bound, and the tangent line (in green) of $\sigma(x)$ at the point $(d, \sigma(d))$ such that $\sigma'(d)$ is equal to the slope of the lower bound. That is, we have $h_{U,r}^t = k(x - d) + \sigma(d)$ and $h_{L,r}^t = k(x - l_r^t) + \sigma(l_r^t)$ with $\sigma'(d) = k$ and $l_r^t < d < u_r^t$.



(a) $0 \le l^t_r < u^t_r$ (b) $l^t_r < 0 < u^t_r$

Figure 3: Approximation when $\sigma'(l^t_r) > k$ and $\sigma'(u^t_r) < k$

Case 3. When $\sigma'(l^t_r) < k$ and $\sigma'(u^t_r) < k$, there is only one possible case when $l^t_r < 0 < u^t_r$ and $\sigma(x)$ between the interval is divided into two parts by the line connecting the two endpoints. Figure 4 shows an example of the case. Under this situation, we take the tangent line of σ over $(l^t_r, \sigma(l^t_r))$ whose cut point is $d_1 > 0$ (the green one) as the linear upper bound, and the tangent line of σ over $(u^t_r, \sigma(u^t_r))$ whose cut point $d_2 < 0$ (the red one) as the linear lower bound. Namely, there are $h^t_{U,r} = \sigma'(d_1)(x - l^t_r) + \sigma(l^t_r)$ and $h^t_{L,r} = \sigma'(d_2)(x - u^t_r) + \sigma(u^t_r)$ where, d_1, d_2 meet the equations $\sigma'(d_1) = \frac{\sigma(d_1)-\sigma(l^t_r)}{d_1-l^t_r}$ and $\sigma'(d_2) = \frac{\sigma(d_2)-\sigma(u^t_r)}{d_2-u^t_r}$, respectively. Note that $\sigma'(d_1)$ and $\sigma'(d_2)$ are not necessarily the same, which depends on l^t_r and u^t_r.

Another case when $\sigma'(l^t_r) > k$ and $\sigma'(u^t_r) > k$ is impossible because it is never satisfiable for sigmoid-like functions. Thus, we do not need to consider approximation in this case.

Note that the approximation of ReLU can be considered a special case of our approach. In the cases of $l^r_t < u^r_t \le 0$ and $0 \le l^r_t < u^r_t$, we have ReLU(x)=0 and x, respectively. When $l^r_t < 0 < u^r_t$, it can be over-approximated like the case of Figure 2(b) by choosing the lower bound line to be kx.

We briefly discuss the difference of our approximation approach to sigmoid-like functions from those of CNN-Cert and CROWN. The major difference arises in the cases shown in Figures 2(b) and 3(b). In the two cases, CROWN takes the same approximation as the one in Case 3. We consider the case of Figure 2(b) as an example. The upper and lower bounds taken by CROWN are denote by the the dashed green and red lines in Figure 5. Note that the lower bound line is not fully shown in the figure due to space limit. Its lower endpoint is the intersection with the vertical line crossing $(l, \sigma(l))$. CNN-Cert chooses the same upper bound as ours, but it takes the same lower bound as CROWN does. Apparently, compared with the other two approximations, ours is the tightest in these two cases, by which we can compute a lower robustness bound that is closer to the optimal one.

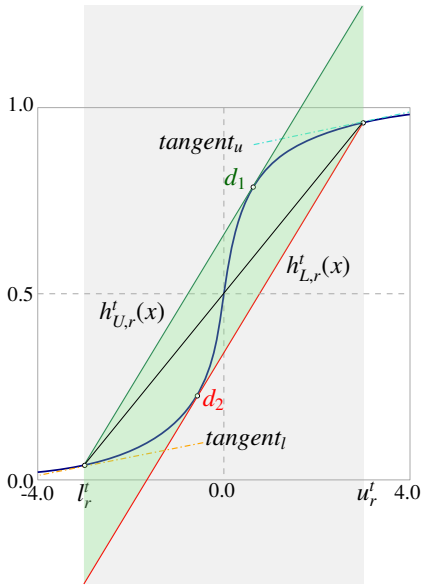


Figure 4: Approximation when $\sigma'(l^t_r) < k$ and $\sigma'(u^t_r) < k$

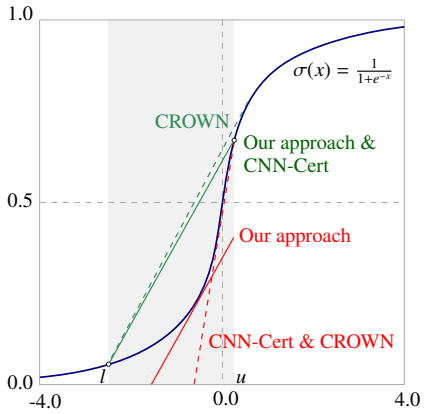


Figure 5: Comparison with existing approaches in case 1(b)

Lower Robustness Bound Computation

It consists of two steps to compute a lower robustness bound for a neural network i.e., computing upper and lower output bounds for all input under a perturbation threshold, and computing a lower bound such that all perturbed inputs have the same classification result as the original input.

Computing upper and lower bounds of neural network output. Given an m-layer CNN Φ, an input x_0, and a perturbation threshold ϵ with an ℓ_p norm, we estimate the upper and lower bounds of Φ for all input in the norm-ball $\mathbb{B}_p(x_0, \epsilon)$. First, we define linear upper and lower bounds that over-approximate $\Phi(x)$. Figure 6 depicts the classical layer-by-layer approximation process of $\Phi(x)$. According to Boopathy et al. (2019), each layer $\phi^t(x)$ for $0 < t \le m$ in a CNN can be over-approximated using a linear upper bound and a linear lower bound. The linear approximation of a layer consists of two steps. The first step is to approximate the activation functions in the previous layer, and the second step is to approximate matrix operations such as con-

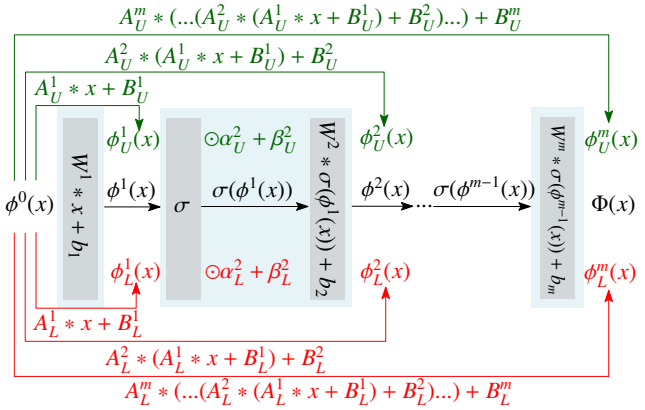


Figure 6: Linear approximation process of $\Phi(x)$

volution, batch normalization and pooling. We make use of our approach to the approximation of activation functions, and reuse the approaches in (Boopathy et al. 2019) for the approximation of matrix operations.

We assume that a layer $\phi^t(x)$ can be represented in the linear form of the output of its previous layer ϕ^{t-1}, i.e.,

$$A_L^t * \phi^{t-1}(x) + B_L^t \leq \phi^t(x) \leq A_U^t * \phi^{t-1}(x) + B_U^t. \quad (2)$$

Particularly, there are $\phi^0(x) = x$ and $\phi^m(x) = \Phi(x)$. Then, $\Phi(x)$ on the domain $\mathbb{B}_p(x_0, \epsilon)$ is over-approximated by the following linear lower and upper bounds:

$$A_L * x + B_L \leq \Phi(x) \leq A_U * x + B_U, \quad (3)$$

where,

$$A_L = \Pi_{i=m}^1 A_L^i, \quad B_L = B_L^m + \Sigma_{i=m-1}^1 (\Pi_{j=m}^{i+1} A_L^j) B_L^i,$$
$$A_U = \Pi_{i=m}^1 A_U^i, \quad B_U = B_U^m + \Sigma_{i=m-1}^1 (\Pi_{j=m}^{i+1} A_U^j) B_U^i.$$

Note that the operator Π means convolution multiplication.

By the inequality 3, we can compute the upper and lower bounds Φ_U and Φ_L of $\Phi(x)$, which are maximal and minimal vectors consisting of the classification probabilities for the inputs in $\mathbb{B}_p(x_0, \epsilon)$. Φ_U and Φ_L are defined as follows:

$$\Phi_U = \max_{x \in \mathbb{B}_p(x_0,\epsilon)} (A_U * x + B_U), \quad \Phi_L = \min_{x \in \mathbb{B}_p(x_0,\epsilon)} (A_U * x + B_U).$$

However, it is impractical to compute Φ_U and Φ_L by enumerating all the inputs in $\mathbb{B}_p(x_0, \epsilon)$ based on the above equations. We compute each element in the vectors separately using the following deduced equation. We consider the rth element in Φ_L for example and denote it by $\Phi_{L,r}$. We use $A_{L,r}$ to denote the rth convolution kernel of A_L, and $B_{L,r}$ to denote the rth element in B_L. By the definition of the dual norm, we have:

$$\begin{aligned}
\Phi_{L,r} &= \min_{x \in \mathbb{B}_p(x_0,\epsilon)} (A_{L,r} * x + B_{L,r}) \\
&= - \max_{x \in \mathbb{B}_p(x_0,\epsilon)} (A_{L,r} * x) + B_{L,r} \\
&= - \max_{y \in \mathbb{B}_p(0,1)} [A_{L,r} * (x_0 + \epsilon y)] + B_{L,r} \\
&= -\epsilon \max_{y \in \mathbb{B}_p(0,1)} (A_{L,r} * y) + A_{L,r} * x_0 + B_{L,r} \\
&= -\epsilon \max_{y' \in \mathbb{B}_p(0,1)} [vector(A_{L,r}) y'] + A_{L,r} * x_0 + B_{L,r} \\
&= -\epsilon \|vector(A_{L,r})\|_q + A_{L,r} * x_0 + B_{L,r}, \quad (4)
\end{aligned}$$

Algorithm 1: Binary search for lower robustness bound

input : CNN Φ, input x_0, norm $\|\cdot\|$
output: A certified lower robustness bound ϵ_{cert}

1 $Y_0 \leftarrow \arg\max_i \Phi_i(x_0)$; // Y_0 is the predicted class of x_0
2 $\epsilon_{\text{cert}} \leftarrow 0.05$, $\epsilon_{\min} \leftarrow 0$, $\epsilon_{\max} \leftarrow +\infty$; // Initialize variables
3 **while** $\epsilon_{cert} > 0$ **do**
4 $flag \leftarrow$ true ; // certification result flag
5 Compute Φ_U, Φ_L of $\Phi(x)$ with $x \in \mathbb{B}_p(x_0, \epsilon_{\text{cert}})$;
6 **for** $i = 1 : len(\Phi(x_0))$ **do**
7 **if** $i! = Y_0$ *and* $\Phi_{U,i} \geq \Phi_{L,Y_0}$ **then**
8 $flag \leftarrow$ false; // classified to i
9 break;
10 **if** *flag*==*false* **then** // decrease when failed to certify ϵ_{cert}
11 $\epsilon_{\max} \leftarrow \epsilon_{\text{cert}}$;
12 $\epsilon_{\text{cert}} \leftarrow \max\{\frac{\epsilon_{\text{cert}}}{2}, \frac{\epsilon_{\max}+\epsilon_{\min}}{2}\}$;
13 **else** // increase when ϵ_{cert} is certified
14 $\epsilon_{\min} \leftarrow \epsilon_{\text{cert}}$;
15 $\epsilon_{\text{cert}} \leftarrow \min\{2\epsilon_{\text{cert}}, \frac{\epsilon_{\max}+\epsilon_{\min}}{2}\}$;
16 **if** $\epsilon_{cert} - \epsilon_{min} < 0.00001$ **then** // terminate
17 **return** ϵ_{cert};

where, $vector(A_{L,r})$ is a vector obtained by expanding the rth convolution kernel of A_L, $\|\cdot\|_q$ is l_q norm with $q = p/(p-1)$. Likewise, we have $\Phi_{U,r} = \epsilon\|vector(A_{U,r})\|_q + A_{U,r} * x_0 + B_{U,r}$.

Computing lower robustness bound. Because the computational complexity of directly computing a non-trivial lower robustness bound is high, we iteratively pick up a perturbation bound and check whether all the inputs under it can be predicated to the same class as the original input is according to the lower and upper output bounds. We make use of binary search to accelerate the process.

Algorithm 1 sketches the binary search procedure. Let Y_0 be the classification result (true label) of input example x_0 predicted by Φ, i.e., $Y_0 = \arg\max_i \Phi_i(x_0)$. Then, we declare three variables ϵ_{cert}, $\epsilon_{\min}$ and $\epsilon_{\max}$, which store the present bound to certify, and the lower and upper bounds to search, respectively. The Boolean variable *flag* indicates the success (true) or failure (false) of the certification result.

We first determine the upper and lower output bounds Φ_U and Φ_L for the inputs in $\mathbb{B}_p(x_0, \epsilon_{\text{cert}})$ using the aforementioned approach (line 5). For each class i other than Y_0, we check whether the upper bound of i is greater than or equal to the lower bound of Y_0, i.e., $\Phi_{U,i} \geq \Phi_{L,Y_0}$ (line 7). If the evaluation result is true, it means that there exists some x in $\mathbb{B}_p(x_0, \epsilon_{\text{cert}})$ such that x is more likely to be classified to i than Y_0. Namely, Φ is not robust within the norm-ball under the bound ϵ_{cert}. In that case, we decrease the value of ϵ_{cert} (line 12). If there is no class i meeting the condition $\Phi_{U,i} \geq \Phi_{L,Y_0}$, it means ϵ_{cert} is certified to be a robustness bound. We increase it for a larger one (line 15). The algorithm terminates when the difference between ϵ_{cert} and $\epsilon_{\min}$ is reasonablly small, e.g., 0.00001 in our algorithm.

Implementation. We implement our approach atop CNN-Cert in Python as an extension named DeepCert. In the ex-

Network		Bound$_{\text{CNN-Cert}}$			Bound$_{\text{DeepCert}}$			Impr. (%)			Time$_{\text{CNN-Cert}}$			Time$_{\text{DeepCert}}$			STD$_{\text{CNN-Cert}}$			STD$_{\text{DeepCert}}$		
		l_∞	l_2	l_1	l_∞	l_2	l_1	l_∞	l_2	l_1	l_∞	l_2	l_1	l_∞	l_2	l_1	l_∞	l_2	l_1	l_∞	l_2	l_1
MNIST	Sig	0.066	0.353	0.779	0.068	0.378	0.835	3.2	7.1	7.2	1.1	1.1	1.1	1.1	1.1	1.1	0.015	0.059	0.121	0.017	0.073	0.148
4 layers	Tan	0.023	0.157	0.374	0.025	0.163	0.389	5.1	4.2	4.0	1.4	1.4	1.4	1.3	1.4	1.5	0.007	0.035	0.072	0.007	0.037	0.078
8680 nodes	Atan	0.024	0.149	0.358	0.025	0.160	0.386	4.6	7.6	7.9	1.1	1.1	1.2	1.1	1.2	1.2	0.007	0.033	0.074	0.025	0.038	0.081
MNIST	Sig	0.086	0.348	0.740	0.103	0.419	0.890	20.6	20.6	20.1	17.0	16.3	16.9	16.1	16.2	16.1	0.019	0.061	0.113	0.027	0.086	0.890
8 layers	Tan	0.013	0.076	0.176	0.014	0.082	0.190	9.4	8.9	8.1	17.1	17.0	17.2	16.7	16.8	16.9	0.003	0.076	0.029	0.004	0.018	0.190
14570 nodes	Atan	0.012	0.072	0.178	0.013	0.081	0.200	12.7	12.3	12.2	16.3	16.3	16.4	16.2	16.3	16.5	0.003	0.072	0.178	0.004	0.081	0.200
MNIST	Sig	0.013	0.064	0.168	0.013	0.064	0.169	0.8	0.6	0.5	12.7	12.9	12.8	12.7	13.6	12.7	0.002	0.010	0.025	0.002	0.010	0.025
LeNet-5	Tan	0.017	0.098	0.268	0.017	0.100	0.273	2.4	1.9	1.8	12.9	12.9	12.9	12.7	12.9	13.0	0.002	0.010	0.027	0.002	0.010	0.027
7 layers	Atan	0.015	0.091	0.251	0.016	0.094	0.257	2.6	2.8	2.3	12.7	12.7	12.8	12.8	12.8	12.8	0.002	0.011	0.029	0.002	0.011	0.029
MNIST	Sig	0.007	0.025	0.040	0.009	0.031	0.050	40.9	24.7	25.6	10.9	10.7	10.9	10.5	10.5	10.5	0.009	0.030	0.046	0.012	0.036	0.059
Resnet	Tan	0.006	0.020	0.036	0.007	0.024	0.041	25.5	17.9	13.9	10.9	10.8	10.8	10.4	10.5	10.5	0.007	0.025	0.047	0.008	0.028	0.047
3 blocks	Atan	0.005	0.015	0.027	0.006	0.019	0.034	18.8	22.7	22.7	10.5	10.4	10.4	10.4	10.5	10.4	0.008	0.026	0.046	0.010	0.032	0.056
MNIST	Sig	0.005	0.023	0.045	0.007	0.033	0.062	44.9	45.6	37.3	28.3	28.4	28.4	28.3	27.9	28.0	0.006	0.028	0.055	0.009	0.037	0.072
Resnet	Tan	0.006	0.019	0.033	0.007	0.023	0.041	19.0	20.1	24.6	28.7	28.5	28.6	28.2	27.9	28.2	0.008	0.027	0.044	0.009	0.032	0.055
4 blocks	Atan	0.009	0.028	0.048	0.011	0.035	0.059	21.6	24.5	20.9	28.1	27.8	28.0	28.1	27.9	28.0	0.011	0.033	0.057	0.013	0.040	0.068
MNIST	Sig	0.008	0.030	0.035	0.011	0.036	0.053	41.3	21.8	49.3	64.9	65.1	64.7	64.3	64.0	54.0	0.009	0.029	0.043	0.011	0.036	0.059
Resnet	Tan	0.008	0.022	0.037	0.011	0.036	0.065	36.0	63.8	76.6	65.8	64.9	64.9	64.6	64.5	54.3	0.005	0.015	0.025	0.008	0.025	0.045
5 blocks	Atan	0.004	0.016	0.028	0.005	0.019	0.033	12.2	17.9	18.5	64.6	63.6	64.2	64.9	54.0	64.1	0.005	0.019	0.033	0.006	0.023	0.039
CIFAR	Sig	0.021	0.205	0.684	0.022	0.216	0.722	2.4	5.2	5.6	6.1	6.1	6.1	6.0	6.1	6.0	0.012	0.068	0.194	0.013	0.078	0.225
5 layers	Tan	0.009	0.103	0.305	0.010	0.106	0.310	1.1	3.2	1.6	6.2	6.2	6.2	6.2	6.2	6.1	0.004	0.027	0.068	0.004	0.028	0.073
14680 nodes	Atan	0.008	0.094	0.302	0.008	0.095	0.308	1.3	1.7	1.9	5.7	5.7	5.7	5.6	6.0	5.6	0.004	0.030	0.085	0.004	0.031	0.089

Table 1: Experiment I: Averaged certified lower bound ϵ_{cert} and execution time by CNN-Cert and DeepCert

tension, we reuse the approximation approaches of CNN-Cert to the matrix operations in CNNs. It supports the CNNs consisting of various types of layers, including convolution, norm-batch, and max pooling. Moreover, it also supports the commonly-used activation functions such as ReLU and all the sigmoid-like functions.

Experimental Evaluation

This section aims to evaluate our approach's effectiveness by comparing it with three relevant tools CNN-Cert, CROWN, and FROWN. We conducted comprehensive experiments on open-source benchmarks including LeNet-5 models (LeCun et al. 1998) and the CNNs trained on MNIST and CIFAR-10. We compared the lower robustness bounds returned by the tools and the time cost on the verification. All the experiments were conducted on a workstation running an 8-core Intel Xeon CPU E5-2620 v4, 32 GB of RAM, and an NVIDIA Tesla K80 GPU.

Benchmarks and Metrics. We trained different types of CNNs, including two pure CNNs and three residual networks trained on MNIST, a network that contains average pooling layers with the same structure as LeNet-5, and a CNN trained on CIFAR-10. The two pure CNN models on MNIST only consist of convolutional layers and fully-connected layers. One has 4 layers, 5 filters and 8680 hidden nodes, while the other has 8 layers, 5 filters and 14570 hidden nodes. Both of them use 3×3 convolutions. LeNet-5 contains 3 convolutional layers, 2 average pooling layers and 2 fully-connected layers. The three residual networks contain 3, 4 and 5 residual blocks respectively with each block consisting of two convolutions. The network on CIFAR-10 is pure CNN with 5 layers, 5 filters and 14680 nodes. For each network architecture, we choose three commonly-used sigmoid-like activation functions *sigmoid*, *tanh*, *arctan* in the trained networks. Namely, we trained three variant neural networks using the three different activation functions for each architecture. It is worth mentioning that LeNet-5 takes only *tanh* as its activation function. We trained two variants of LeNet-5 by replacing only *tanh* with *sigmoid*, *arctan* in the same architecture. We also consider the cases with different types of norms including l_1, l_2 and l_∞ norms.

We compare DeepCert with the other three tools in terms of the lower robustness bound the tools to certify that. We first check whether the labels of test images given by the network are correct, and then select those with correct labels to compute the targeted certified lower bound. We use $(\epsilon'_{\text{cert}} - \epsilon_{\text{cert}})/\epsilon_{\text{cert}}$ to quantitatively represent the improvement, where ϵ'_{cert} and ϵ_{cert} indicate the lower bounds certified by DeepCert and other tools, respectively. Likewise, the efficiency is indicated by the average of execution time cost on the ten images, and we use t/t' to represent the speedup of our approach, where t' and t are the time cost by DeepCert and other tools, respectively.

Experiment I. In the first experiment, we compare DeepCert with CNN-Cert. We fix 100 input images and take the average of 100 certified lower bounds as the final result for each tool. Table 1 shows the experimental results on the 21 neural networks under the seven different network architectures, including the certified bounds, improvement, averaged execution time, and the standard deviation. It can be clearly seen that our tool almost always returns a larger lower bound than CNN-Cert, up to 76.6% improvement in all tests. Particularly, DeepCert has a much better performance on resident networks because back layers repeatedly use the output of front layers in resident networks. Consequently, the computed lower bound for resident networks is more sensitive to the precision of activation function approximation than the one for pure CNNs. For the networks that contain pure convolutional layers, the more activation functions they contain,

Network		ℓ_p	Certified Bounds ϵ_{cert}			Bound Improvement (%)		Certification Time (sec)			Time Speedup	
Tool			CROWN	FROWN	DeepCert	vs. CROWN	vs. FROWN	CROWN	FROWN	DeepCert	vs. CROWN	vs. FROWN
MNIST 4 layers 5 filters 8680 nodes	Sig	l_∞	0.061	0.067	0.067	10.4	1.0	1.9	1292.8	1.3	1.4	986.8
		l_2	0.217	0.335	0.372	72.0	11.1	2.3	1930.4	1.3	1.8	1473.6
		l_1	0.482	0.634	0.821	70.3	29.5	2.5	2052.5	1.3	1.9	1566.8
	Tanh	l_∞	0.012	0.015	0.023	96.6	60.3	1.1	1152.7	1.4	0.8	823.4
		l_2	0.066	0.066	0.158	**140.2**	**140.2**	1.0	1417.5	1.5	0.7	964.3
		l_1	0.150	0.145	0.378	**151.4**	**159.9**	1.0	1583.1	1.5	0.7	1048.4
	Atan	l_∞	0.013	-	0.022	77.8	-	0.8	-	1.1	0.7	-
		l_2	0.077	-	0.150	**94.9**	-	0.8	-	1.1	0.7	-
		l_1	0.178	-	0.365	**105.0**	-	0.8	-	1.2	0.7	-
MNIST 8 layers 5 filters 14570 nodes	Sig	l_∞	0.083	0.111	0.099	19.6	-11.1	4.0	5211.1	16.1	0.3	323.1
		l_2	0.338	0.428	0.407	20.3	-4.9	4.3	5508.8	16.2	0.3	340.1
		l_1	0.719	0.870	0.865	20.3	-0.6	3.9	5362.3	16.2	0.2	331.4
	Tanh	l_∞	0.005	0.006	0.014	**164.1**	**124.0**	4.6	3964.1	16.7	0.3	237.2
		l_2	0.022	0.023	0.079	**269.3**	**252.9**	3.6	3787.4	16.8	0.2	224.9
		l_1	0.048	0.053	0.185	**286.3**	**245.7**	4.0	3737.7	16.9	0.2	220.8
	Atan	l_∞	0.007	-	0.076	83.4	-	3.8	-	16.3	0.2	-
		l_2	0.036	-	0.076	**109.3**	-	4.0	-	16.4	0.3	-
		l_1	0.080	-	0.191	**137.9**	-	4.2	-	16.5	0.3	-

Table 2: Experiment II: Averaged certified lower bound ϵ_{cert} and computation time by CROWN, FROWN and DeepCert

the greater improvement our tool makes, as shown by the first two network architectures in the table. Another observation from the comparison is that DeepCert achieves a better improvement for the l_∞ norm. As for the time efficiency, DeepCert is slightly faster than CNN-Cert, and meanwhile, it costs almost the same time for the same network architecture with different activation functions.

Experiment II. We compare DeepCert with CROWN and FROWN in this experiment. Because CROWN and FROWN are designed for only DNNs, we transform CNNs with pure convolutional layers into DNNs by converting the convolutional layers into fully-connected layers. The converted 4-layer network has 8680 hidden nodes, and the 8-layer one has 14570 hidden nodes. In this experiment, we randomly choose 10 images as one input set instead of 100 images. There are two main reasons that we do not use 100 images. One is that FROWN takes too much time for each input image, e.g., 20 minutes. The other reason is that the average result of 100 images is similar to the one of 10 images, according to the observation result from (Boopathy et al. 2019).

Table 2 shows the results of the three tools. DeepCert can certify larger robustness bounds in all the six cases with up to 286.3% improvement than CROWN. It takes more time in most cases because CROWN simply takes the tangent line at the midpoint of the interval as the lower or upper bounding line. DeepCert achieves a better balance between the tightness of certified bound and the efficiency than CROWN.

Compared with FROWN, DeepCert can compute larger robustness bounds with up to 252.9% improvement in most of the cases, and meanwhile it has an up to 1566.8× speedup. However, in our experiment we found an exception where FROWN computed lager robustness bounds than DeepCert does for the 8-layer network with sigmoid activation function. The reason for the exception is that FROWN always performs a gradient-based search for optimal upper and lower bounds, which might be tighter than what DeepCert does in some cases. Tighter linear approximations lead to more precise certified bounds. However, the searching procedure is costly, leading to drastic increase of time consumption. Note that no result is provided for the networks with *arctan* because FROWN does not support *arctan*. The experimental results also reflect that the computation time of DeepCert mainly depends on the size of networks, while those of CROWN and FROWN depend on the types of activation functions and L-norms besides the size of networks.

In summary, the two experiments show that in general DeepCert can compute larger lower robustness bounds with significant improvement than the three state-of-the-art relevant tools, and meanwhile it has a comparable performance on efficiency. The improvement mainly arises from our fine-grained approximation to activation functions, which is the major difference of DeepCert from other three frameworks.

Concluding Remarks

In this paper we have proposed a fine-grained linear approximation approach to general activation functions for computing a larger robustness lower bound for convolutional neural networks. The approach is applicable to approximate sigmoid-like functions using tighter lower and upper linear bounds with a low computational complexity. We implemented the approach atop the state-of-the-art robustness lower bound certification tool CNN-Cert. The experiments on various neural networks with different activation functions have shown our approach's generality and effectiveness in tightening robustness certificates significantly at a lower time cost than other relevant tools. We believe that the balance of the trade-off between the preciseness and efficiency would make our approach applicable to real-world networks.

Like other existing approximation-based approaches, one issue of our approach is the incompleteness. The lack of completeness means that failing to compute a certified robustness bound does not imply the network is not robust within the bound (Liu et al. 2019). Refinement (Elboher, Gottschlich, and Katz 2020) is a useful technique to prove or disprove the robustness within the bound. One of our future work is to investigate a refinement approach for our approximation approach for further tightening the lower robustness bound without the loss of its scalability.

Acknowledgments

This work is partially supported by National Key Research and Development Program (2020AAA0107800), Huawei Technologies Co., Ltd., Joint Funding and AI Project (No. 20DZ1100300) of Shanghai Science and Technology Committee, and NSFC general projects (No. 61872146, 61872144). Min Zhang is the corresponding author.

References

Balunovic, M.; Baader, M.; Singh, G.; Gehr, T.; and Vechev, M. 2019. Certifying geometric robustness of neural networks. In *NeurIPS'19*, 15313–15323.

Boopathy, A.; Weng, T.-W.; Chen, P.-Y.; Liu, S.; and Daniel, L. 2019. CNN-Cert: An efficient framework for certifying robustness of convolutional neural networks. In *Proc. of AAAI'19*, volume 33, 3240–3247.

Botoeva, E.; Kouvaros, P.; Kronqvist, J.; Lomuscio, A.; and Misener, R. 2020. Efficient Verification of ReLU-Based Neural Networks via Dependency Analysis. In *AAAI'20*, 3291–3299. AAAI Press.

Cousot, P. 1996. Abstract interpretation. *ACM Computing Surveys* 28(2): 324–328.

Elboher, Y. Y.; Gottschlich, J.; and Katz, G. 2020. An Abstraction-Based Framework for Neural Network Verification. In *CAV'20*, 43–65.

Fawzi, A.; Moosavi-Dezfooli, S.-M.; Frossard, P.; et al. 2017. The robustness of deep networks: A geometrical perspective. *IEEE Signal Processing Magazine* 34(6): 50–62.

Gehr, T.; Mirman, M.; Drachsler-Cohen, D.; Tsankov, P.; Chaudhuri, S.; and Vechev, M. T. 2018. AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation. In *S&P'18*, 3–18. IEEE Computer Society.

Gopinath, D.; Katz, G.; Păsăreanu, C. S.; and Barrett, C. 2018. Deepsafe: A data-driven approach for assessing robustness of neural networks. In *ATVA'18*, 3–19. Springer.

Goswami, G.; Ratha, N.; Agarwal, A.; et al. 2018. Unravelling robustness of deep learning based face recognition against adversarial attacks. In *AAAI'18*, 6829–6836.

Huang, X.; Kroening, D.; Ruan, W.; Sharp, J.; Sun, Y.; Thamo, E.; Wu, M.; and Yi, X. 2020. A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. *Computer Science Review* 37: 100270.

Huang, X.; Kwiatkowska, M.; Wang, S.; and Wu, M. 2017. Safety verification of deep neural networks. In *CAV'17*, 3–29. Springer.

Katz, G.; Barrett, C.; Dill, D. L.; Julian, K.; and Kochenderfer, M. J. 2017. Reluplex: An efficient SMT solver for verifying deep neural networks. In *CAV'17*, 97–117. Springer.

LeCun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86(11): 2278–2324.

Liu, C.; Arnon, T.; Lazarus, C.; Barrett, C. W.; and Kochenderfer, M. J. 2019. Algorithms for Verifying Deep Neural Networks. *CoRR* abs/1903.06758.

Lyu, Z.; Ko, C.; Kong, Z.; et al. 2020. Fastened CROWN: Tightened Neural Network Robustness Certificates. In *AAAI'20*, volume 34, 5037–5044. AAAI Press.

Mirman, M.; Gehr, T.; and Vechev, M. T. 2018. Differentiable Abstract Interpretation for Provably Robust Neural Networks. In *ICML'18*, 3575–3583.

Moore, R. E.; Kearfott, R. B.; and Cloud, M. J. 2009. *Introduction to interval analysis*, volume 110. Siam.

Moosavi-Dezfooli, S.-M.; Fawzi, A.; and Frossard, P. 2016. Deepfool: a simple and accurate method to fool deep neural networks. In *CVPR'16*, 2574–2582.

Ruan, W.; Huang, X.; and Kwiatkowska, M. 2018. Reachability Analysis of Deep Neural Networks with Provable Guarantees. In *IJCAR'18*, 2651–2659.

Shen, H.; Wang, B.; Wang, J.; et al. 2018. Research on Robustness of Voiceprint Recognition Technology. In *ACAI'18*, 1–5. ACM.

Singh, G.; Gehr, T.; Püschel, M.; and Vechev, M. 2019a. An abstract domain for certifying neural networks. In *POPL'19*, volume 3, 41:1–30. ACM.

Singh, G.; Gehr, T.; Püschel, M.; and Vechev, M. T. 2019b. Boosting Robustness Certification of Neural Networks. In *ICLR'19*. OpenReview.net.

Szegedy, C.; Zaremba, W.; Sutskever, I.; et al. 2014. Intriguing properties of neural networks. In Bengio, Y.; and LeCun, Y., eds., *ICLR'14*.

Tjeng, V.; Xiao, K. Y.; Tedrake, R.; et al. 2019. Evaluating Robustness of Neural Networks with Mixed Integer Programming. In *ICLR'19*.

Wan, W.; Zhang, Z.; Zhu, Y.; Zhang, M.; and Song, F. 2020. Accelerating Robustness Verification of Deep Neural Networks Guided by Target Labels. *CoRR* abs/2007.08520.

Wang, S.; Pei, K.; Whitehouse, J.; Yang, J.; and Jana, S. 2018a. Efficient formal safety analysis of neural networks. In *NeurIPS'18*, 6367–6377.

Wang, S.; Pei, K.; Whitehouse, J.; Yang, J.; and Jana, S. 2018b. Formal Security Analysis of Neural Networks using Symbolic Intervals. In *27th USENIX Security Symposium*, 1599–1614.

Weng, T.; Zhang, H.; Chen, H.; Song, Z.; et al. 2018. Towards Fast Computation of Certified Robustness for ReLU Networks. In *ICML'18*, 5273–5282. PMLR.

Wong, E.; Schmidt, F. R.; Metzen, J. H.; and Kolter, J. Z. 2018. Scaling provable adversarial defenses. In *NeurIPS'18*, 8410–8419.

Woodcock, J.; Larsen, P. G.; Bicarregui, J.; and Fitzgerald, J. 2009. Formal methods: Practice and experience. *ACM Computing Surveys* 41(4): 1–36.

Zhang, H.; Weng, T.-W.; Chen, P.-Y.; et al. 2018. Efficient neural network robustness certification with general activation functions. In *NeurIPS'18*, 4939–4948.