Eager Falsification for Accelerating Robustness Verification of Deep Neural Networks

Xingwu Guo*, Wenjie Wan*†, Zhaodi Zhang*, Min Zhang*
*Shanghai Key Laboratory for Trustworthy Computing,
East China Normal University, Shanghai, China
†ByteDance Privacy and Security, Shanghai, China
zhangmin@sei.ecnu.edu.cn

Fu Song‡§, Xuejun Wen¶
‡ShanghaiTech University, Shanghai, China
§Shanghai Engineering Research Center of Intelligent
Vision and Imaging, Shanghai, China
¶Huawei International, Singapore

***Abstract*—Formal robustness verification of deep neural networks (DNNs) is a promising approach for achieving a provable reliability guarantee to AI-enabled software systems. Limited scalability is one of the main obstacles to the verification problem. In this paper, we propose *eager falsification* to accelerate the robustness verification of DNNs. It divides the verification problem into a set of independent subproblems and solves them in descending order of their falsification probabilities. Once a subproblem is falsified, the verification terminates with a conclusion that the network is not robust. We introduce a notion of *label affinity* to measure the falsification probability and present an approach to computing the probability based on symbolic interval propagation. Our approach is orthogonal to existing verification techniques. We integrate it into four state-of-the-art verification tools, *i.e.*, `MIPVerify`, `Neurify`, `DeepZ`, and `DeepPoly`, and conduct extensive experiments on 8 benchmark datasets. The experimental results show that our approach can significantly improve these tools by up to 200x speedup when the perturbation distance is in a reasonable range.**

***Index Terms*—Deep neural network, robustness verification, adversarial example, scalability**

I. Introduction

The significant progress of deep learning makes it applicable to safety-critical domains such as autonomous driving [1]–[3] and medical diagnostics [4]–[6]. Because applications in these domains demand high-reliability guarantees, it is necessary to formally certify the related properties of deep neural networks (DNNs). Among them, robustness is one of the most important properties, which requires DNNs to make the same classification for all perturbed inputs in a reasonable perturbation range. However, DNNs are found to be vulnerable against robustness and may suffer from adversarial attacks and environmental perturbations [7]–[12]. Formal robustness verification has been proposed and intensively investigated as a promising technique for certifying the robustness of neural networks [13]–[27].

Although many efforts have been made to the robustness verification DNNs, limited scalability is still one of the most challenging obstacles due to the intrinsic high computational complexity of the verification problem. It has been proved that the robustness verification problem of DNNs with the simplest activation function, Rectified Linear Unit (ReLU), is even NP-complete [14]. Therefore, heuristic verification strategies are necessary to mitigate the situation.

In this work, we propose a general and effective approach, called *eager falsification*, to accelerate the robustness verification. Specifically, we divide a verification problem into a set of independent subproblems and solve them in descending order of the probability of falsifying the robustness. The number of subproblems is equal to the one of potential misclassified labels. Once one of them is falsified, we can conclude that the deep neural network is not robust and terminate the verification. If all the subproblems are satisfied, we can conclude that the deep neural network is robust. The falsification is called *eager* in that in each iteration the subproblem that is the most likely to falsify is solved first.

Eager falsification relies on the way of identifying the subproblem where there most likely exists a perturbed input that is misclassified by the neural network. Given an original unperturbed input, we can easily know to which label the input is most likely misclassified according to its probability. Intuitively, we can take the sub-problem corresponding to that label as the most likely falsified case. However, we find that this is not always true. Instead of computing the falsification probability directly, we introduce a notion of *label affinity*, which reflects a relative probability in which a subproblem is falsified. We propose a symbolic interval propagation-based approach to computing affinities among classification labels for both feedforward neural networks (FNNs) and convolutional neural networks (CNNs).

Our eager falsification approach is not competing with other state-of-the-art robustness verification approaches. Instead, it is orthogonal to and compatible with many existing verification approaches [21], [22], [24], [26]. The properties *i.e. soundness* and *completeness* (if satisfied) of the original verification approach will be preserved when our eager falsification approach is applied. That is because it only affects the order of target labels for searching for adversarial examples, but does not alter the solution space of the original problem. To demonstrate the efficacy of our eager falsification approach, we integrate it into four recent promising verification tools, *i.e.* `MIPVerify` [22], `Neurify` [26], `DeepZ` [21], and DeepPoly [24] [1]. We conduct experiments to measure efficiency improvement by applying the

[1]The four extended tools are available at https://github.com/MakiseGuo/Verifast.

2332-6549/21/$31.00 ©2021 IEEE
DOI 10.1109/ISSRE52982.2021.00044

original tools and extended ones to their built-in benchmarks, respectively. The experimental results show that the proposed eager falsification approach can accelerate these tools with up to 218× speedup when the perturbation threshold is in reasonable ranges.

In summary, this work is a sequel of previous works on the robustness verification of DNNs and makes the following main contributions:

- A general and effective approach for accelerating the robustness verification of DNNs using eager falsification.
- A new approach for measuring the order of falsification probabilities of subproblems by label affinity.
- Extensions of four state-of-the-art verification tools with up to 218x efficiency speedup on extensive benchmarks.

The rest of this paper is organized as follows. Section II presents some preliminaries that are necessary to understand our approach. Section III describes the details of our eager falsification approach and the approach for sorting subproblems. Section IV presents extensive experiments by integrating our approach into four state-of-the-art verification tools and evaluates their performance improvements. Section V discusses related work and Section VI finally concludes the paper.

II. Preliminaries

In this section, we recap some preliminaries such as feed-forward deep neural networks, interval analysis, symbolic interval propagation, and linear relaxation that are necessary to understand our approach.

A. Deep Neural Networks (DNNs)

An l-layer ($l \geq 2$) DNN is a function $f : I \rightarrow O$, mapping the set of vectors I to the set of vectors O, where f is recursively defined as follows:

$$\begin{aligned} \bar{x}^0 &= \bar{x}, \\ \bar{x}^{k+1} &= \phi(W^k \bar{x}^k + \bar{b}^k) \quad \text{for } k = 0, ..., l-1, \\ f(\bar{x}) &= W^l \bar{x}^l + \bar{b}^l, \end{aligned} \tag{1}$$

$\bar{x}^0 = \bar{x} \in I$ is the input vector of real numbers, W^k and $\bar{b}^k$ respectively are the weight matrix and bias vector of the k-th layer, and $\phi(\cdot)$ (e.g., `ReLU`, `sigmoid`, `tanh` *etc.*) is an activation function applied to the input vector in a coordinate-wise way. The activation function `ReLU`, defined by $\texttt{Relu}(x) \equiv \max(0, x)$, is one of the most popular activation functions in the modern state-of-the-art DNN architectures [28]–[30]. In this work, we are focused on DNNs that only take `ReLU` as the activation function. For a given input $\bar{x}$, the *label* of $\bar{x}$ is determined by the function $\mathcal{L}$, defined as follows,

$$\mathcal{L}(f(\bar{x})) = \arg\max_j f(\bar{x})[j],$$

where, $f(\bar{x})[j]$ denotes the j-th element in the output vector $f(\bar{x})$ which is the confidence that $\bar{x}$ is classified to the label j. By applying the *softmax* function to the output $f(\bar{x})$, we will get the probabilities of the labels to which the input $\bar{x}$ is classified. For this reason, we in what follows may say $f(\bar{x})[j]$ is the probability that the input $\bar{x}$ is classified to the label j. For simplicity, we also use the indices j to represent the classification labels. $\mathcal{L}(f(\bar{x}))$ returns the label whose corresponding probability is the largest among all the labels. We call it the *true label* of the input $\bar{x}$. In the case that the last step is not well defined, namely, there are more than one maximal element in $f(\bar{x})$, the index of the first maximal element is regarded as the true label of the input $\bar{x}$.

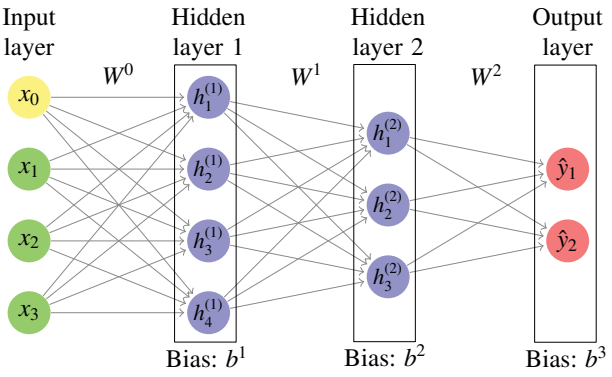


Fig. 1. A fully connected feedforward deep neural network $f : I \rightarrow O$

B. Robustness Verification of DNNs

Intuitively, a DNN is robust if and only if it can always return the same classification result for a slightly perturbed input as the true label of the original input. The perturbation range of an input is usually represented as an L-norm distance threshold. There are three widely used L-norms: L_0, L_2, and L_∞ norms. In this work, we only consider L_∞ norm, that is: for each pair of vectors $\bar{x}, \bar{x}'$ with the same size,

$$\left\|\bar{x} - \bar{x}'\right\|_\infty \equiv \max\{\left|\bar{x}[i] - \bar{x}'[i]\right| \ : \ i \text{ is an index of } \bar{x}\}.$$

An input $\bar{x}$ and a distance threshold ϵ form an input region, which contains all the inputs $\bar{x}'$ such that $\|\bar{x} - \bar{x}'\|_\infty \leq \epsilon$. A DNN is (local) robust *w.r.t.* the input $\bar{x}$ and distance threshold ϵ if and only if for all inputs x' such that $\|\bar{x} - \bar{x}'\|_\infty \leq \epsilon$, the DNN always makes the same predication result. In the literature, a DNN is called global robust *w.r.t.* a test dataset if it is local robust for each input of the given test dataset [31]. In this work, we focus on the local robustness.

Definition 1 (Robustness)**.** *Given a DNN $f : I \rightarrow O$, an input $\bar{x} \in I$, and a L_∞ distance threshold ϵ, the DNN f is* robust *w.r.t. the input $\bar{x}$ and distance threshold ϵ if*

$$\mathcal{L}(f(\bar{x})) = \mathcal{L}(f(\bar{x}'))$$

for all inputs $\bar{x}' \in I$ such that $\|\bar{x} - \bar{x}'\|_\infty \leq \epsilon$.

If there exists some input $\bar{x}' \in I$ such that $\|\bar{x} - \bar{x}'\|_\infty \leq \epsilon$ and $\mathcal{L}(f(\bar{x})) \neq \mathcal{L}(f(\bar{x}'))$, $\bar{x}'$ is called an *adversarial example* of $\bar{x}$. Therefore, the essence of the robustness verification is to check the satisfiability of the conjunction of the two constraints $\|\bar{x} - \bar{x}'\|_\infty \leq \epsilon$ and $\mathcal{L}(f(\bar{x})) \neq \mathcal{L}(f(\bar{x}'))$. Due to the non-linearity or semi-linearity, checking satisfiability is

computationally expensive, *e.g.*, NP-complete for the FNNs that contain the simplest ReLU activation function [14].

The robustness notion can be refined *w.r.t.* a specific label, which we call labeled robustness.

Definition 2 (Labeled robustness)**.** *Given a DNN $f : I \to O$, an input $\bar{x} \in I$, and a L_∞ distance threshold ϵ, let j be a label such that $j \neq \mathcal{L}(f(\bar{x}))$. The DNN f is called* j-robust *w.r.t. the input $\bar{x}$ and distance threshold ϵ, if for all inputs $\bar{x}' \in I$ such that $\|\bar{x} - \bar{x}'\|_\infty \leq \epsilon$, $\mathcal{L}(f(\bar{x}')) \neq j$.*

The next proposition states that the robustness verification problem can be equivalently reduced to a series of labeled robustness problems, which up to our knowledge has never been stated in the literature though straightforward.

Theorem 1. *Given a DNN $f : I \to O$, an input $\bar{x} \in I$, and a L_∞ distance threshold ϵ, suppose J is the set of all the possible labels of f. Then, f is robust w.r.t. $\bar{x}$ and ϵ if and only if f is j-robust w.r.t. $\bar{x}$ and ϵ, for all $j \in J \setminus \{\mathcal{L}(f(\bar{x}))\}$.*

The proof is straightforward and we omit the details of the proof due to the space limit.

To verify whether a DNN f is robust for an input $\bar{x}$ and a distance threshold ϵ, by Theorem 1, it suffices to verify that the DNN f is j-robust for every possible classification label j except for the true label $\mathcal{L}(f(\bar{x}))$. In other words, to falsify the robustness of a DNN f for the input $\bar{x}$ and distance threshold ϵ, it suffices to find a label j such that the DNN f is not j-robust. Theorem 1 provides a theoretical foundation for dividing the robustness verification problem into subproblems to conquer in efficient ways such as eager falsification in our approach.

Another application of labeled robustness is that given an input to a DNN f, we can verify whether the DNN f is robust or not on the input with respect to some specific classification labels. In some cases, perturbations are allowed if they do not cause a DNN to misclassify perturbed inputs to some specific labels that are completely different from the true label. For instance, consider a DNN supporting a self-driving car, the predication labels of the DNN are fed into a controller that drives the car. In this scenario, what matters is the consequences of incorrect predictions on the final driving behavior produced, rather than the incorrect predictions made by the DNN. It may be acceptable if the DNN misclassifies a dog as a cat, but not acceptable if the DNN incorrectly classifies a dog as a car.

C. Linear Approximation

Linear approximation is a technique for over-approximating non-linear constraints using linear constraints [19]. Linear relaxation of a unary non-linear interval function $g(X)$ with $X = [l, u]$ is a function $h(g, X)$ such that

$$h(g, X) = [lc_1, lc_2],$$

where lc_1, lc_2 are lower and upper linear bounds of $g(X)$. Linear approximation ensures that $lc_1(x) \leq g(x) \leq lc_2(x)$ for any $x \in [l, u]$. However, in general, lc_1 and lc_2 are not unique. The principle of defining lc_1 and lc_2 is that they should be as tight as possible.

As we focus on DNNs with only the `ReLU` activation function, we show the linear relaxation for `ReLU`. The linear relaxation function for `ReLU` on $X = [l, u]$ is defined as follows:

$$h(\texttt{ReLU}, X) = \begin{cases} X, & \text{if } l \geq 0; \\ [0, 0], & \text{if } u \leq 0; \\ [ax, \frac{u}{u-l}(x - l)], & \text{if } l < 0 < u; \end{cases}$$

where $a \in [0, 1]$ is adaptively chosen for the lower bound.

Fig. 2 shows three different cases on the linear relaxation of `ReLU` with different intervals.

- When $l = -5$ and $u = 1$ (Fig. 2(a)), the upper bound is $\frac{1}{6}(x + 5)$ and the lower bound is 0, *i.e.* $a = 0$.
- When $l = -2$ and $u = 4$ (Fig. 2(b)), the upper bound is $\frac{2}{3}(x + 2)$ and the lower bound is x, *i.e.* $a = 1$.
- When $l = -3$ and $u = 3$ (Fig. 2(c)), the upper bound is $\frac{1}{2}(x+3)$ and the lower bound is $\frac{1}{2}x$, forming a parallelogram to approximate ReLU with $[-3, 3]$.

Linear approximation can achieve remarkable improvement to efficiency by transforming non-linear problems into linear ones. That is because non-linear problems are usually hard to solve, while linear problems can be solved in polynomial time. However, linear-approximation causes overestimation of output range and consequently causes false positives, i.e., a returned adversarial example might not be feasible due to over-approximation. Recently, many efforts are made to define tight linear approximations to reduce the overestimation [32]–[34].

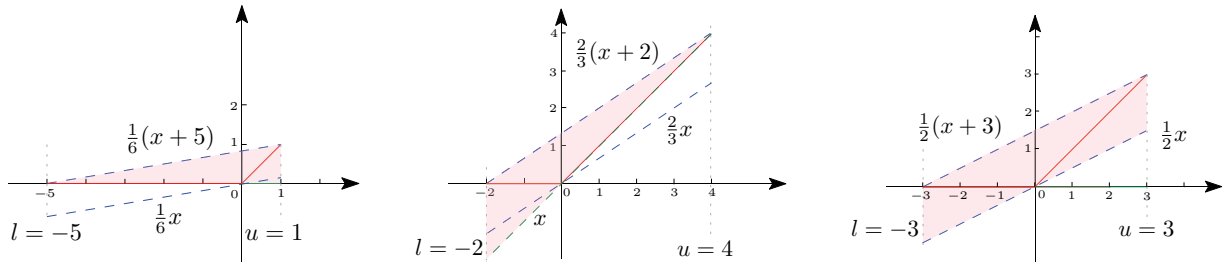
D. Symbolic Interval Propagation

To sort subproblems for each given DNN $f : I \to O$ with an input $\bar{x} \in I$ and a distance threshold ϵ, we will propagate the interval from the input layer to the output layer via interval propagation. However, naively computing the output interval of the DNN in this way suffers from high errors as it computes extremely loose bounds due to the dependency problem. In particular, it may produce a very conservative estimation of the output, which is not tight enough to be useful.

Consider a 3-layer DNN given in Fig. 3(a), where the weights are associated to the edges and all elements of the bias vectors are 0. Suppose the input of the first layer are the intervals $[1, 3]$ and $[2, 4]$. By performing the scalar multiplications and additions over intervals layer-by-layer, we get the output $[-5, 7]$. This output interval contains several concrete values that are introduced by overestimation, but are infeasible in practice. For instance, -5 can occur only when the neuron n_{21} outputs 10 and the neuron n_{22} outputs 5. To output 10 for the neuron n_{21}, the neurons n_{11} and n_{12} should output 3 and 4 simultaneously. But, to output 5 for the neuron n_{22}, the neurons n_{11} and n_{12} should output 1 and 2 simultaneously. This effect is known as the *dependency problem* [35].

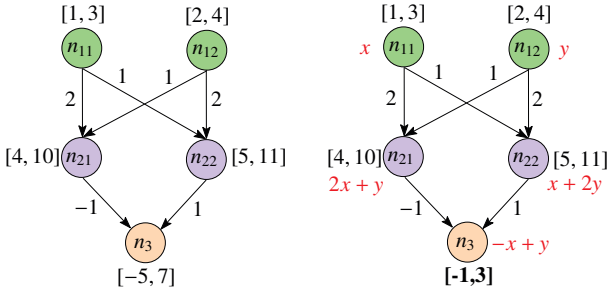
Symbolic interval propagation [36] is an effective solution to minimize overestimation of outputs by preserving the dependency information during propagating the intervals layer-by-layer. A *symbolic interval* is a pair of linear expressions $[e, e']$ such that e and e' are defined over the input variables.

Let us consider the same example using symbolic interval propagation as shown in Fig. 3(b). Suppose x and y are the



(a) $|l| > |u|$ (b) $|l| < |u|$ (c) $|l| = |u|$

Fig. 2. Examples of choosing different a for the linear relaxation of ReLU with different intervals



(a) Naive interval propagation (b) Symbolic interval propagation

Fig. 3. Naive interval propagation vs. symbolic interval propagation [36].

input variables of the neurons n_{11} and n_{12}. By applying the linear transformation of the first layer, the values of the neurons n_{21} and n_{22} are $2x+y$ and $x+2y$, respectively. Since $x \in [1,3]$ and $y \in [2,4]$, we have: $2x+y > 0$ and $x+2y > 0$. Therefore, the output symbolic intervals of the neurons n_{21} and n_{22} are $[2x+y, 2x+y]$ and $[x+2y, x+2y]$, respectively. By applying the linear transformation of the second layer, the value of the neuron n_3 is $-x+y$. Thus, the output of the DNN will be $[-x+y, -x+y]$. From $x \in [1,3]$ and $y \in [2,4]$, we can conclude that the output interval of the DNN is $[-1,3]$, which is strictly tighter than the interval $[-5,7]$ produced by directly performing interval propagation.

This exemplifies how symbolic interval propagation characterizes each neuron result in terms of the symbolic intervals and related activation functions. As the symbolic intervals keep the inter-dependence between variables, symbolic interval propagation significantly reduces the overestimation.

III. The Eager Falsification Verification Framework

In this section, we introduce our eager falsification verification framework for accelerating the robustness verification of DNNs. At its core, it divides the robustness verification problem into a series of independent labeled robustness verification problems and solves them in the order of their falsification probabilities to be falsified, which we call *eager falsification*.

A. Eager Falsification

To the best of our knowledge, most existing approaches reduce the robustness verification problem to the problem of proving that all the inputs in a specified region can be correctly classified to the true label. Equivalently, the robustness is falsified if an input is found to be classified to a different label. Thus, the verification time basically depends upon the size of search space.

Instead of considering all the labels, we propose to sort labels and verify DNNs label by label in order to search for adversarial examples in small and independent spaces specific to concrete labels. According to Theorem 1, the robustness verification problem can be reduced into a series of labeled robustness problem. In each labeled robustness problem, say j-robustness, we check whether there is an adversarial example in the specified region that can be classified to the label j. If there exists such an adversarial example, we can conclude that the DNN is non-robust, therefore the verification costs of remaining labeled robustness problems can be avoided.

Algorithm 1 shows our verification framework. It takes a DNN f, an input $\bar{x}$ and a perturbation threshold ϵ, and outputs one of the following results, depending on the underlying labeled robustness verification engine used in the algorithm, i.e., `Verifier` in Algorithm 1:

- *Robust*, meaning that the robustness of the DNN f is proved.
- *Non-robust*, meaning that the DNN f is falsified. An adversarial example is returned as a witness to the violation.
- *Unknown*, meaning that the robustness of the DNN f is neither proved nor falsified. Note that this case only occurs when the underlying verification engine is not complete.

In detail, Algorithm 1 divides the robustness verification problem into a list of labeled robustness problems. It sorts the labeled robustness problems in descending order of label affinity (to be detailed in Section III-B), represented by a sorted list of labels J' (lines 1–2). Note that the true label $\mathcal{L}(f(\bar{x}))$ is excluded from J'. Then, Algorithm 1 iteratively verifies the j-robustness of the DNN f for each label $j \in J'$ (while-loop) until J' becomes empty or an adversarial example is found.

During each iteration of the loop, Algorithm 1 fetches the head label j from the list J', and checks whether the DNN f is j-robust or not by invoking a back-end DNN verification engine `Verifier`. The engine `Verifier` takes the DNN f, the input $\bar{x}$, the distance threshold ϵ and the target label j as inputs,

Algorithm 1: Verification with Eager Falsification

Input : A DNN f, an input vector $\bar{x}$, a distance threshold ϵ

Output : {Robust, Non-robust with an adversarial example, Unknown}

```
1  J := Labels(f)/{L(f(x̄))} // J: the labels to certify
2  J′:=sort(f, x̄, ϵ, J); // Sort labels
3  flag:=false; // to indicate the unkown case.
4  while J′ ≠ nil do
5      j := head(J′); // Take the head label.
6      Result:=Verifier(f, x̄, ϵ, j); // Verify on j
7      switch Result do
8          case true do
9              J′ := tail(J′); // Delete the head
10             continue;
11         case false do
12             x̄′:=getAdvExample(f, x̄, ϵ, j); return x̄′ ;
13         case unkown do
14             flag:=true; // Try next label
15             J′ := tail(J′); // Remove the head of J′
16             continue;
17 if flag then
18     return unknown; // Some subproblem fails.
19 else
20     return robust; // All labels are certified.
```

and may output *True*, *False* or *Unknown*. Our implementation makes use of off-the-shelf DNN verification engines as they are state of the art.

If `Verifier` returns *true* (*i.e.*, the DNN f is j-robust), Algorithm 1 proceeds to verify the remaining labels. If it returns *false*, we extract and return an adversarial example. It may return *unknown*, if the DNN verification engine is not complete. In that case, we set a *flag* to record this failure and skip this label. After all the labels have been verified and not adversarial examples are found, Algorithm 1 returns *robust* if *flag* is not true, and *unknown* otherwise.

We remark that the soundness and completeness of Algorithm 1 rely on the back-end engine `Verifier`. We assume that the engine (*e.g.*, `DeepZ` and `Neurify`) is sound, which is reasonable according to the survey [37]. Then, Algorithm 1 is also sound, *i.e.*, if it returns *Robust*, f must be robust *w.r.t.* x and ϵ. Likewise, Algorithm 1 is complete if the back-end engine is complete.

B. Sorting Labeled Robustness Problems by Label Affinity

A premise order of the labeled robustness problems is that an instance that has a larger probability to be falsified should be falsified earlier. However, the real probability that a label robustness problem can be falsified is hard to calculate in theory, because if we know the exact probability, we would not need to verify it further. We observe that what matters is the order of probabilities rather than the exact probabilities. Therefore, to address this challenge, we introduce a new notion of *label affinity*, which is a real number representing the distance between an adversarial label to the correct classification label. For an input $\bar{x}$, a label that has a larger affinity to the true label of $\bar{x}$ is more likely to be falsified than those that have smaller affinities.

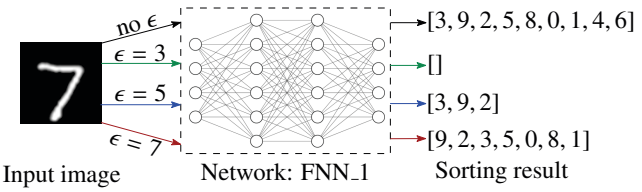


Fig. 4. Sorting results under different perturbation ranges

A naïve approach for measuring the affinities between two labels l_1 and l_2 for an input $\bar{x}$ is to compute the difference of the probabilities of classifying $\bar{x}$ to l_1 and l_2. Let us consider an example shown in Fig. 4. We suppose that the probabilities of classifying the image "7" by a network to 7, 3 and 9 are 72%, 15% and 6%, respectively. Then, the affinity between 3 and 7 is -0.57, and the one between 9 and 7 is -0.66. Therefore, 3 has a larger affinity with 7 than 9. The top list in Fig. 4 is the sorting result of labeled robustness problems.

A drawback of the above approach is that the information of perturbation threshold ϵ cannot be reflected by the affinity. Under different perturbation thresholds, the most likely misclassified labels may be different. For example, the image of "7" in Fig. 4 cannot be misclassified to other labels when ϵ is 3. When ϵ is 5, the most likely misclassified label is 3. In the case of $\epsilon = 9$, the most likely misclassified label becomes 9. Thus, label affinity should take perturbation threshold into account. To address this technical challenge, we propose a novel affinity definition that takes the distance threshold ϵ into account.

Given an input $\bar{x}$ and a distance threshold ϵ, we assume that a neural network f outputs an interval of probabilities for each label. This assumption is reasonable because the interval can be estimated using interval analysis technique. Let ℓ_r be the true label of $\bar{x}$ classified by f without any perturbations. We use $[l_r, u_r]$ to denote the probability interval of ℓ_r for all perturbed inputs of $\bar{x}$ under ϵ. Let ℓ_s be a different label from ℓ_r and $[l_s, u_s]$ be the probability interval of ℓ_s. Then, we define the affinity between ℓ_s and ℓ_r as follows:

$$\mathcal{A}(\ell_s, \ell_r) = \begin{cases} u_s - l_r & \text{if } u_s \geq l_r \\ -\infty & \text{otherwise} \end{cases} \tag{2}$$

Being $-\infty$ means that there is no affinity between ℓ_s and ℓ_r. If that is the case, we can safely exclude the verification of ℓ_s-robustness because the input $\bar{x}$ is theoretically impossible to be classified to ℓ_s no matter how $\bar{x}$ is perturbed under ϵ. The bigger $\mathcal{A}(\ell_s, \ell_r)$ is, the more overlap there is between the output probability intervals of ℓ_s and ℓ_r. It means a higher probability that there exists an adversarial example of $\bar{x}$ under ϵ such that the adversarial example is classified to ℓ_r.

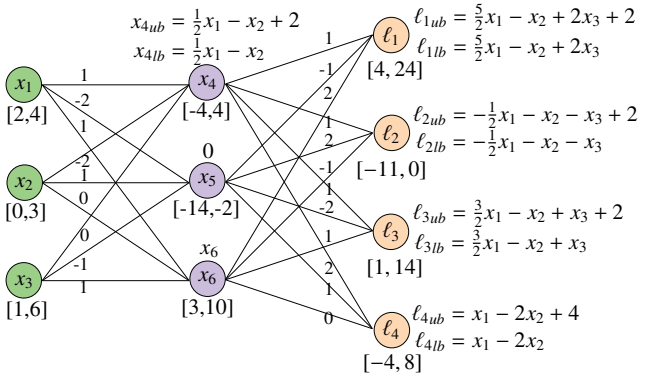


Fig. 5. An example of computing label affinities

We use the example in Fig. 5 to explain the process of computing label affinities. We assume that there are three input intervals, which are calculated by a concrete input and a distance perturbation. They are propagated layer by layer and finally an output interval is computed for each label. This can be achieved by symbolic interval propagation as we introduced in Section II-D. We assume that ℓ_3 is the true label of the input if there is no perturbation. Then, the label affinities between ℓ_1, ℓ_2, ℓ_4 and ℓ_3 are 23, $-\infty$, and 7, respectively. The verification of ℓ_2-robustness can be excluded because there are no adversarial examples that have higher probabilities of being misclassified to ℓ_2 than to ℓ_3. The verification of ℓ_1-robustness has a higher priority to falsify than the one of ℓ_4-robustness because it has a higher affinity than ℓ_3-robustness.

The example in Fig. 4 depicts the affection of perturbation distance to sorting result. Under different ϵ, *e.g.*, 3, 5, and 7, the sorted lists of labels which shall be falsified in Algorithm 1 are different. They are also different from the sorting result when perturbation is not considered in affinity definition.

- When $\epsilon = 3$, our sorting approach returns an empty list. It means that the original image cannot be misclassified to any other labels under the threshold distance $\epsilon = 3$, thereby the robustness of the DNN is proved.
- When $\epsilon = 5$, the sorting result is $[3, 9, 2]$. It means that only the three subproblems labeled by $3, 9, 2$ may have adversarial examples. Furthermore, the one labeled by 3 has the highest probability to be falsified.
- When $\epsilon = 7$, the sorting result is $[9, 2, 3, 5, 0, 8, 1]$, and the subproblem with label 9 has the highest label affinity and thereby shall be the first to falsify.

Our sorting approach has two advantages. First, the information of perturbation distance is considered, which produces a more precise order than the naïve one. Second, the labels that do not have adversarial examples are excluded from the list, and consequently their labeled robustness problems are omitted. These advantages allow us to efficiently find an adversarial example when there exists one. In the case that there are no adversarial examples, the improvement is more significant because all the labeled robustness problems are safely excluded before feeding them into the back-end verification engine.

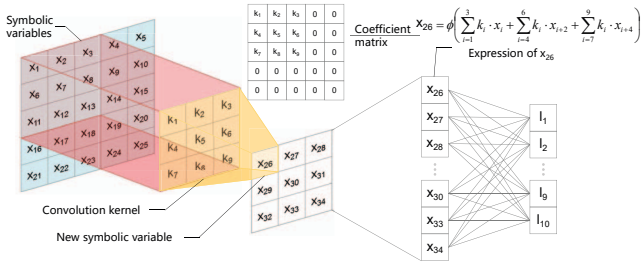


Fig. 6. A symbolic interval propagation example of CNN

C. Extending to CNNs

In this subsection, we discuss how to extend the aforementioned label sorting approach to the CNNs. Recall that the key feature of CNNs is convolutional layers. As the pooling layers can be handled straightforwardly, we only discuss how to handle convolutional layers using symbolic interval propagation.

Consider a convolutional layer with a group of t filters $F^{p,q} = (F_1^{p,q}, \cdots, F_t^{p,q})$, where for each $1 \le i \le t$, the filter $F_i^{p,q}$ has type $\mathbb{R}^{m \times n \times r} \to \mathbb{R}^{(m-p+1)\times(n-q+1)}$. During symbolic interval propagation, the input $\bar{x}$ to each filter $F_i^{p,q}$ becomes a three-dimensional array of symbolic intervals. After the computation of the filter $F_i^{p,q}$, the entry of $F_i^{p,q}(\bar{x})$ at the coordinate (i, j) also becomes a symbolic interval. By stacking the outputs produced by each filter, the output of the convolutional layer is a three-dimensional array of symbolic intervals, where the activation function `ReLU` is approximated as described in Section II-C.

Fig. 6 shows an example of the symbolic interval propagation for convolutional layers. In this example, for the sake of simplifying the presentation, the input to the convolutional layer and weights of the filter are given as two-dimensional arrays. Indeed, we can regard each entry of these arrays as a vector of symbolic intervals. Then, the symbolic interval of the variable x_{26} can be computed as follows:

$$x_{26} = \phi(\Sigma_{i=1}^{3} k_i \cdot x_i + \Sigma_{i=4}^{6} k_i \cdot x_{i+2} + \Sigma_{i=7}^{9} k_i \cdot x_{i+4}),$$

where ϕ denotes the activation function.

In our implementation, in contrast to `Neurify` which performs symbolic interval propagation via SMT solving, we implemented a data structure to store and propagate the coefficients of symbolic expressions instead of directly propagating symbolic expressions. Indeed, we found that the approach of `Neurify` becomes inefficient with the increase of the size of DNNs.

To further improve efficiency, inspired by the implementations of convolution computations in current popular deep learning libraries such as Caffe [38] and MXNet [39], we use image-to-column (img2col) and generalized matrix multiplication methods to implement symbolic interval propagation for convolutional layers. By combining these implementation-level accelerations, the efficiency of our approach is significantly improved using the BLAS library [40] and Multi-Core CPU. Thus, the overhead of label sorting is very limited and can be even ignored, compared with the time cost of verification.

IV. Implementation and Evaluation

In this section, we evaluate the effectiveness of our labeled robustness problem sorting approach to the acceleration of robustness verification of DNNs. We integrate the approach into four existing efficient verification tools and compare their time cost, respectively.

A. Implementation

We have implemented Algorithm 1 using Julia programming language [41]. To evaluate its performance, we leverage the following four most recent promising DNN verification tools as back-end labeled robustness verification engines. To demonstrate the orthogonality of our approach, we choose the tools that are based on different verification techniques including constraint solving, interval analysis and abstract interpretation.

- `MIPVerify` [22] formulates the robustness verification program as an MILP program. It improves existing MILP-based approaches via a tighter formulation for non-linearities and a novel presolve algorithm. `MIPVerify` is both sound and complete.
- `Neurify` [26] is one of the most efficient approximation-based DNN verification tools. It introduces symbolic interval analysis and linear relaxation to compute tighter bounds of outputs. Although `Neurify` is theoretically complete, its refinement process might take too much time in practice and thereby a threshold is usually needed to force termination.
- `DeepZ` [21] is based on abstract interpretation. It introduces a specialized abstract domain, coupled with abstract transformers for handling affine transforms, activation functions, and the max pooling operation. `DeepZ` supports both feed-forward and convolutional networks. It is sound but not complete.
- `DeepPoly` [24] is also based on abstract interpretation but with a new abstract domain. It is one of the state-of-the-art DNN verification tools with remarkable scalability and precision. It is intentionally designed to be incomplete for better scalability. Both `DeepZ` and `DeepPoly` are included in the toolkit ERAN.

We leverage the above four efficient tools as back-end verification engines. Note that these tools do not directly support labeled robustness verification and therefore we revise their implementations when they are used as our engines. To reduce the overhead caused by sorting, we propagate the coefficients of symbolic expressions instead of propagating the expressions directly. We make use of image-to-column (img2col) and generalized matrix multiplication methods in the BLAS library [40] to deal with convolutional layers.

a) Benchmarks: We choose eight neural networks that are provided by ERAN and `Neurify`. They have different architectures, including three FNNs (denoted by FNN_i with $i = 1, 2, 3$) and five CNNs (denoted by CNN_i with $i = 1...5$). Among them, CNN_3, CNN_4 and CNN_5 are adversarially trained with DiffAI [42], SPAD (ϵ=0.1) [43], and PGD (ϵ=0.1) [44], respectively. Network information is given in Table I. Column 2 gives the number of activation functions. Column 3 gives the numbers of layers and neurons in each layer.

TABLE I
Details of the DNNs used in our experiments

Model	ReLUs	Network Architecture	Source	Defense
FNN_1	48	$\langle$784,24,24,10$\rangle^{\#}$	`Neurify`	None
FNN_2	200	$\langle$784,100,100,10$\rangle^{\#}$	ERAN	None
FNN_3	500	$\langle$784,100,100,100,100,100,10$\rangle^{\#}$	ERAN	None
CNN_1	3604	$\langle$784,*k*:16*4*4 *s*:2 *p*:0, *k*:32*4*4 *s*:0 *p*:1,100,10$\rangle^{+}$	ERAN	None
CNN_2	5704	$\langle$784,*k*:16*4*4 s:2 *p*:1, *k*:32*4*4 s:2 *p*:1,1000,10$\rangle^{+}$	ERAN	None
CNN_3	48064	$\langle$784,*k*:32*3*3 s:1 *p*:1, *k*:32*4*4 s:2 *p*:1, *k*:64*3*3 s:1 *p*:1, *k*:64*4*4 s:2 *p*:1,100,10$\rangle^{+}$	ERAN	DiffAI
CNN_4	4804	$\langle$784,*k*:16*4*4 s:2 *p*:1, *k*:32*4*4 *s*:2 *p*:1,100,10$\rangle^{+}$	`Neurify`	SPAD ϵ=0.1
CNN_5	4804	$\langle$784,*k*:16*4*4 *s*:2 *p*:1, *k*:32*4*4 *s*:2 *p*:1,100,10$\rangle^{+}$	`Neurify`	PGD ϵ=0.1

#: $\langle x, y, ... \rangle$ denotes a DNN with x number of neurons in first layer, y number of neurons in second layer, etc.
+: k:c*h*w denotes the output channel (c), kernel height (h), kernel width (w), s and p denote stride (s) and padding (p).

b) Experimental settings: For each DNN, we evaluate the performance of the tools under different perturbation thresholds on the same set of inputs. We selected the first 100 images from the test set of MNIST [45] as original inputs and verify the robustness of target DNNs on them. Remark that we may evaluate some tools on subset of the selected DNNs because the back-end tools do not support some neural network architectures or input formats. The timeout threshold is set to three hours per execution. All the experiments were conducted on a workstation running Ubuntu 18.04 with a 32-core AMD Ryzen Threadripper 3970X CPU @ 3.7GHz and 128 GB of RAM.

B. Evaluation of Labeled Robustness Problem Sorting

We evaluate the effectiveness of our labeled robustness problem sorting approach, *i.e.*, whether a labeled robustness problem with a higher label affinity is more likely to be falsified than those with lower affinities. We conduct experiments on six DNNs including three FNNs and three CNNs. For each DNN, we selected 100 images and vary perturbation thresholds. Then, we obtain a list of sorted labeled robustness problems for each image. Let m be the number of images on which the target DNN is not robust, and n_j be the number of cases where a real adversarial example is found for the label j. Then, n_j/m denotes the falsification probability of the j-robustness problem. We chose `MIPVerify` as the back-end labeled robustness verification engine because of its completeness. However, `MIPVerify` cannot scale to large DNNs. If `MIPVerify` fails due to out of time or memory, we use `DeepPoly` as alternative. Because `DeepPoly` is practically incomplete, we only consider the cases where `DeepPoly` returns real counterexamples to guarantee the validity of the results.

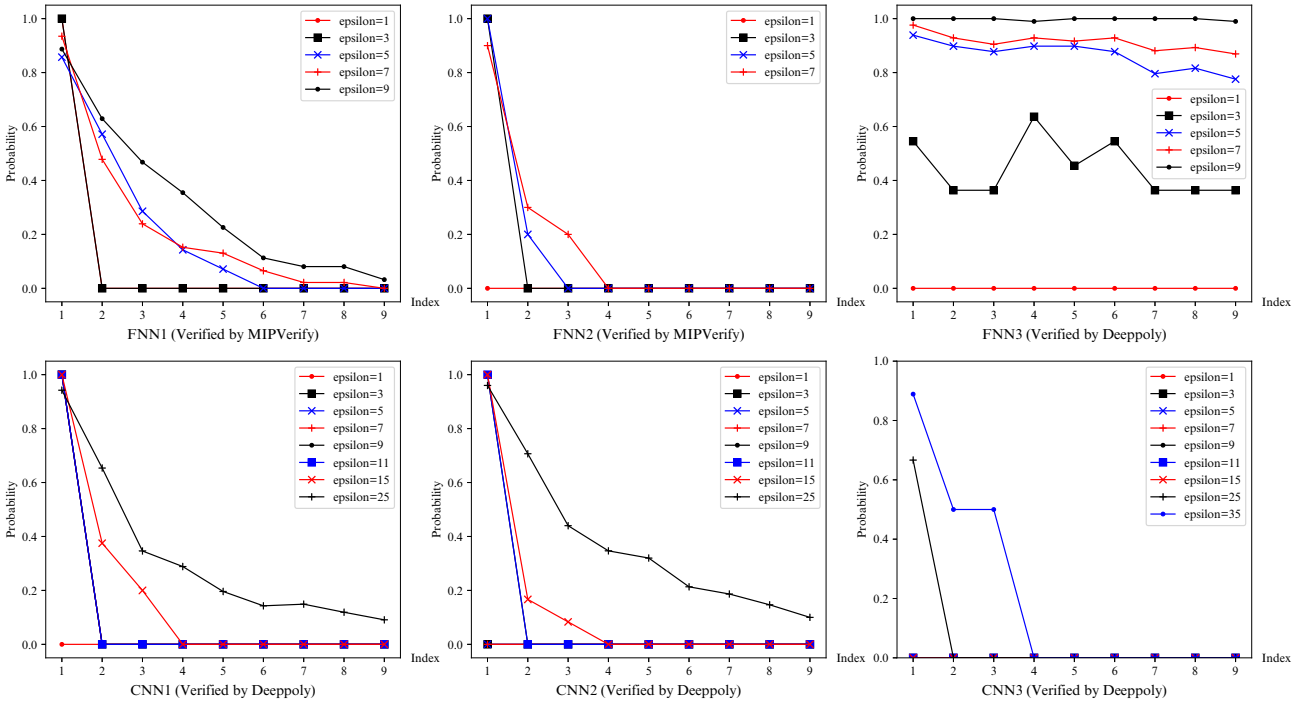


Fig. 7. The probabilities of falsifying sorted subproblems under different perturbation threshold and networks

Fig. 7 shows the falsification probabilities of all the labeled robustness problems under different perturbation thresholds. The horizontal axis in each figure represents the index of each label in the sorted list J' in Algorithm 1, namely, the descending order of the labeled robustness problems. The vertical axis represents corresponding falsification probability.

We can observe that in most cases, the falsification probabilities are monotonic decreasing, which demonstrates the effectiveness of our labeled robustness problem sorting approach using label affinity. There is an exception case for the network FNN3 when $\epsilon = 3$. We found that the number of the 4-th labeled robustness problems that are falsified is even more than that of the first one. This is due to the over-approximation during symbolic interval propagation, which incurs the overestimation of output range. A fine-grained approximation approach has been recently proposed to compute tighter output range [34]. It could be used to reduce the overestimation and affect the order of the subproblems to solve. We would investigate this in our future work.

Note that for clarity we do not show the cases where all the labeled robustness problems in a list can be falsified. This occurs when the perturbation threshold is large, *e.g.*, 35 on CNN3. It is also possible that the falsification probabilities of all the labeled robustness problems are 0, *e.g.*, the cases of FNN2, FNN3, CNN1 and CNN3 with $\epsilon = 1$, namely, these networks are robust on all the selected 100 images with $\epsilon = 1$.

C. Evaluation of Efficiency Improvement

Fig. 8 depicts the accelerations that our eager falsification brings to the four tools. The acceleration is quantitatively measured by:

$$\frac{T - (T^* + T_{sort})}{T^* + T_{sort}}, \qquad \text{(Acceleration Rate)}$$

where, T denotes the verification time of an original tool, T_{sort} denotes the sorting time of the labeled robustness problems, and T^* denotes the verification time by the extended tool after sorting. The total verification time of extended tools is $T^* + T_{sort}$.

a) Performance on `MIPVerify`*:* We tested our approach when `MIPVerify` is used as the back-end tool on the three FNNs and two CNNs which are CNN_4 and CNN_5. The other three CNNs are not supported by `MIPVerify` because the input of `MIPVerify` needs to be normalized between 0 and 1.

Fig. 8(a) shows the results on five DNNs. The maximal acceleration is up to 36.63x for FNN_1 when $\epsilon = 1$. We observe that the acceleration on smaller perturbations is more significant than that on larger perturbations. This is because the smaller the perturbation value is, the more the labeled robustness problems can be safely discarded without solving. Another observation is that the acceleration on CNNs is more significant than that on FNNs. The reason is that convolution layers can enhance feature extraction. Because the two CNNs have been adversarially trained with defense methods such as SPAD [43] and PGD [44], many robust cases are discarded before solving.

b) Performance on `Neurify`*:* We evaluated the performance of our approach on `Neurify` using the same DNNs as on `MIPVerify`. Because `Neurify` discards those labels that are impossible to be classified to, the efficiency improvement are purely benefited from our sorting approach.

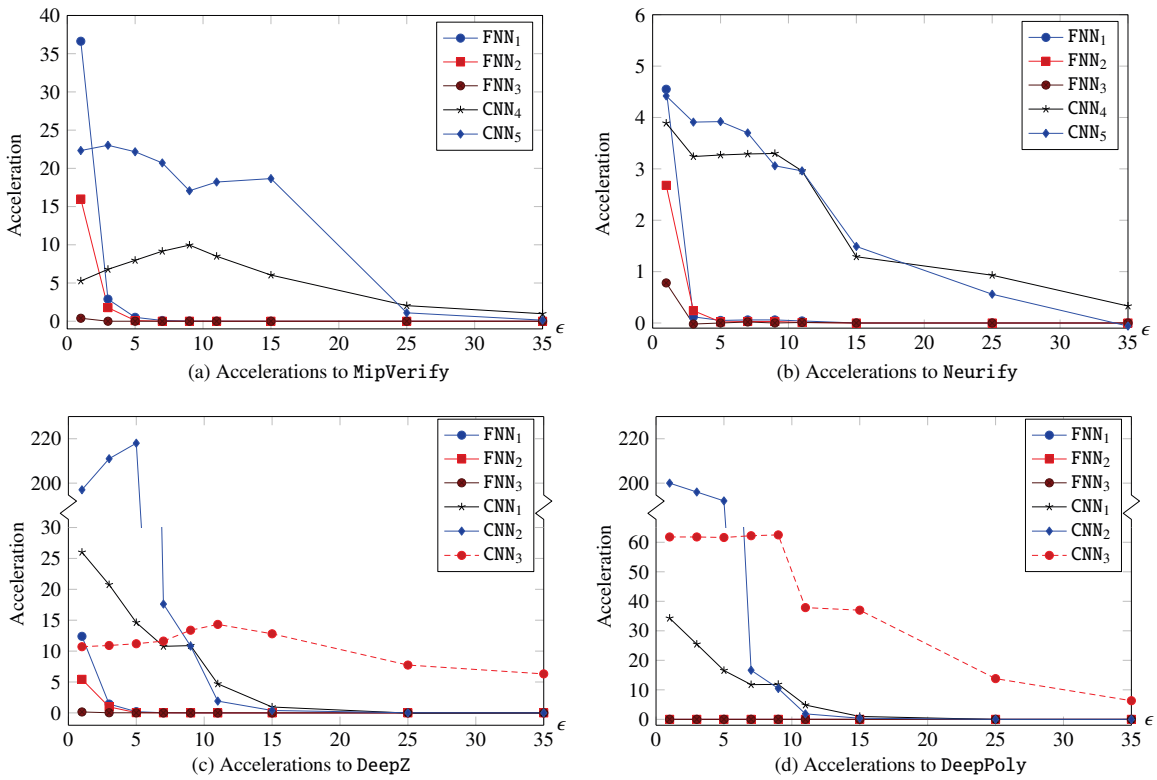


Fig. 8. Accelerations of the four extended tools on various neural networks

Fig. 8(b) depicts the acceleration results. It can be observed that our approach can achieve better performance on CNNs with smaller perturbation distances than FNNs or with larger perturbation distances. The result reveals that our sorting approach can help `Neurify` to achieve up to 4.55x speedup.

c) Performance on `DeepZ`*:* We evaluated the performance of our approach on `DeepZ` using three FNNs and three CNNs, i.e., FNN_1, FNN_2, FNN_3, CNN_1, CNN_2 and CNN_3. Fig. 8(c) shows the acceleration results. For better understanding the improvement, we also give the concrete experimental results in Table II as an example. In the case of FNNs, the effectiveness of our approach on `DeepZ` has similar trend than the one on `MIPVerify` with the increase of perturbation distance. Our approach achieves up to 12.39× speedup. But in the case of CNNs, our approach is able to significantly accelerate `DeepZ` with up to 218x speedup. This improvement benefits from discarding those unsatisfiable subproblems before solving them. The results on CNN_3 show the effectiveness of our approach on large CNNs even when the perturbation distance is big, *e.g.*, 6.31x speedup for $\epsilon = 35$.

In terms of precision, both tools proved and falsified the same number of robust properties of inputs under different distance thresholds. Compared with `MipVerify` and `MipVerify`*, both `DeepZ` and `DeepZ`* are able to verify more robust properties of FNN_2 and FNN_3 on which `MipVerify` and `MipVerify`* run out of time (cf. Tables II(b) and II(c)). This is because `DeepZ` and `DeepZ`* are based on abstract interpretation technique, which is more scalable than the MILP-based approach of `MipVerify` and `MipVerify`* for abstraction. However, `DeepZ` and `DeepZ`* are not complete for the same reason.

d) Performance on `DeepPoly`*:* We evaluated our approach on `DeepPoly` with the same DNNs as on `DeepZ`. Fig. 8(d) shows that our approach has a little improvement to `DeepPoly` for FNNs. Because compared with `DeepZ`, `DeepPoly` has achieved remarkable efficiency improvement to small-scale DNNs [24]. Nevertheless, the accelerations on `DeepPoly` for CNNs are similar to the one on `DeepZ`. In particular, the acceleration on `DeepPoly` for CNN_3 is higher than that of `DeepZ`, because without acceleration `DeepPoly` spends more time than `DeepZ`.

From the above experimental results, we can observe that our acceleration approach is effective to scale up all the four state-of-the-art tools, which are based on three different verification techniques. It demonstrates the generality and orthogonality of our approach to the existing tools. The results also show that the time cost of our subproblem sorting approach is usually small and can be omitted, compared with the verification time of the original tools.

D. Threats to Validity

There are some threats to the validity of the eager falsification approach and its performance. We discuss some threats below.

Large perturbation threshold would be one major threat. Our approach has a better performance when the distance threshold

TABLE II
PERFORMANCE COMPARISON BETWEEN DeepZ AND ITS EXTENDED VARIANT DeepZ* ACCELERATED BY EAGER FALSIFICATION

(a) The result of FNN_1

	FNN_1, valid input: 99/100					
ϵ	t_{sort}	DeepZ	DeepZ*	ACC	S/U	S*/U*
1	0.05	19.22	1.38	12.39	96/3	96/3
3	0.05	19.42	7.92	1.43	90/9	90/9
5	0.05	19.50	16.16	0.20	54/45	54/45
7	0.05	19.57	19.16	0.02	24/75	24/75
9	0.05	19.63	19.65	0.00	6/93	6/93
11	0.06	19.81	19.80	0.00	0/99	0/99

(b) The result of FNN_2

	FNN_2, valid input: 98/100					
ϵ	t_{sort}	DeepZ	DeepZ*	ACC	S/U	S*/U*
1	0.19	99.09	15.21	5.43	97/1	97/1
3	0.20	101.67	51.27	0.97	89/9	89/9
5	0.22	104.89	100.71	0.04	56/42	56/42
7	0.33	107.52	106.10	0.01	13/85	13/85
9	0.34	110.09	109.93	0.00	2/96	2/96
11	0.36	112.11	112.36	-0.01	0/98	0/98

(c) The result of FNN_3

	FNN_3, valid input: 99/100					
ϵ	t_{sort}	DeepZ	DeepZ*	ACC	S/U	S*/U*
1	0.54	179.57	156.28	0.15	99/0	99/0
3	0.67	197.18	192.40	0.02	81/18	81/18
5	0.70	228.52	228.77	0.00	33/66	33/66
7	0.71	254.76	254.77	0.00	3/96	3/96
9	0.72	271.70	271.70	0.00	0/99	0/99
11	0.73	284.41	283.52	0.00	0/99	0/99

(d) The result of CNN_1

	CNN_1, valid input: 100/100					
ϵ	t_{sort}	DeepZ	DeepZ*	ACC	S/U	S*/U*
1	4.82	136.49	0.23	26.00	100/0	100/0
3	4.84	140.46	1.60	20.75	99/1	99/1
5	4.72	143.61	4.47	14.62	99/1	99/1
7	4.87	147.40	7.65	10.77	98/2	98/2
9	4.86	150.88	7.82	10.89	97/3	97/3
11	5.05	155.13	22.13	4.70	95/5	95/5
15	4.84	176.46	85.63	0.95	89/11	89/11
25	5.11	203.17	203.46	-0.03	29/71	29/71
35	5.18	435.67	435.77	-0.01	0/100	0/100

(e) The result of CNN_2

	CNN_2, valid input: 100/100					
ϵ	t_{sort}	DeepZ	DeepZ*	ACC	S/U	S*/U*
1	10.03	2039.40	0.22	197	100/0	100/0
3	10.33	2245.56	0.22	211	100/0	100/0
5	10.54	2362.05	0.22	218	100/0	100/0
7	10.38	2501.49	108.99	17.61	100/0	100/0
9	10.36	2984.07	177.15	10.81	98/2	98/2
11	10.33	3489.56	750.19	1.91	95/5	95/5
15	10.42	4244.63	1626.08	0.39	82/18	82/18
25	10.46	5686.27	2375.54	0.02	8/92	8/92
35	10.43	TO	TO	–	–	–

(f) The result of CNN_3

	CNN_3, valid input: 95/100					
ϵ	t_{sort}	DeepZ	DeepZ*	ACC	S/U	S*/U*
1	104.37	1225.32	0.22	10.71	95/0	95/0
3	104.53	1247.42	0.22	10.91	95/0	95/0
5	105.23	1285.31	0.22	11.19	95/0	95/0
7	104.90	1327.19	0.22	11.62	95/0	95/0
9	104.76	1509.20	0.21	13.38	95/0	95/0
11	104.58	1801.22	12.96	14.32	95/0	95/0
15	104.84	1687.58	17.48	12.80	94/1	94/1
25	106.98	1868.05	106.83	7.74	92/3	92/3
35	106.82	2901.56	290.05	6.31	86/9	86/9

is in a reasonable small range than the case of large threshold. It can significantly improve the efficiency of the original tools when the distance threshold is relatively small and even solve the problems that the original tool fails. When the distance threshold is relatively large, the improvement may not be as significant as the small case. As we mentioned previously, when the distance threshold becomes larger, more labels have adversarial examples on a valid perturbed input. In the worst case, back-end tools can find adversarial examples on any label except the true one. Then, eager falsification may not save verification time. In contrast, it may cause overhead due to the extra time cost on sorting. Nevertheless, the tools extended with our approach are almost comparable with the original tools when the distance threshold is large. Besides, our assumption is pragmatic in practice that the distance threshold to be verified is relatively small.

Another threat is the quality of the neural networks to be verified. If a neural network is not robust and easy to perturb, eager falsification may not improve the verification efficiency. The reason is similar to the case when the perturbation threshold is too large. Our experimental results also reflect that the proposed approach has a better performance on the DNNs that are adversarially trained. Usually, a DNN becomes more robust after it is adversarially trained. For a relatively robust DNN, our approach discards the cases which do not contain adversarial examples and therefore the time spent on solving these subproblems by the original tools can be saved.

V. RELATED WORK

We discuss existing formal verification techniques for DNNs (cf. [37], [46] for a survey). DNN testing (e.g., [10]–[12], [47]–[56] to cite a few) is excluded, which is computationally less expensive and can work on large DNNs, but at the cost of losing theoretic guarantees.

Existing formal verification techniques can be broadly classified as either complete or incomplete ones. Complete techniques are based on constraint solvers such as SMT and MILP solving [14], [23], [23], [57], [58]. Such approaches essentially reduce the robustness verification to the problem of solving a collection of linear programming problems. The number grows exponentially with the number of neurons in networks, which limits the scalability of these approaches. For example, the verification of an FNN with 5 inputs, 5 outputs, and 300 total hidden neurons on a single input takes Reluplex a few hours [14]. Another solver-based verification system is Planet [23], which resorts to satisfiability (SAT) solvers. Although complete techniques produce neither false positives nor false negatives, their scalability is always an obstacle that prevents them from being applied to relatively large DNNs.

In contrast, incomplete techniques usually rely on approximation and abstraction for better scalability, but they may produce false positives. Existing incomplete techniques mainly include duality [17], layer-by-layer approximations of the adversarial polytope [59], discretizing the search space [60], abstract interpretation [20], [21], [24], linear approximations [18], bounding the local Lipschitz constant [18], or bounding the activation of the ReLU activation function with linear functions [18]. Recently, two novel abstraction-based frameworks have been proposed [61], [62] to transform complex DNNs into small ones by merging neurons and abstracting the transformation of neurons, respectively. Li *et al*. proposed a symbolic propagation technique for propagating values layer by layer on abstract domains [63]. The common feature of these approaches is that they do not intend to solve the verification task directly. Instead, they tune the verification problem into a classical linear programming problem for efficient solving. Although approximation and abstract can significantly improve the efficiency, they have to rely on iterative refinement when false positives are produced.

Our approach is orthogonal to these approaches and can be integrated with existing tools. Although both the symbolic interval analysis and linear relaxation techniques have been used in existing works, to our knowledge, they are the first time used for sorting labeled robustness verification problems.

Furthermore, our eager falsification verification methodology that reduces to the robustness verification problem of a DNN to the independent labeled robustness verification problems of the DNN is new to our knowledge.

VI. Conclusion and Future Work

We have proposed a general and effective approach to accelerate the robustness verification of DNNs and extended four state-of-the-art tools by our approach. As the experimental results demonstrated, our approach is orthogonal to and compatible with the tools, bringing them up to 218x speedup. We believe that the acceleration makes it possible and pragmatical to verify the DNNs in real-world systems using more powerful hardware such as GPU. Furthermore, all the labeled robustness problems are independent, thus could be verified in parallel from the implementation point of view.

As for the future work, we would extend our approach to the neural networks that take non-ReLU activation functions using the fine-grained approximation approach to compute output ranges for the labeled robustness problem sorting [34]. Besides, it is also worth investigating other metrics for the subproblem sorting, e.g., the percentage of overlapping sections. We believe this work would inspire new algorithmically efficient approaches. For instance, by dividing the verification problem into independent subproblems, it is possible to improve scalability and efficiency using parallel computing techniques.

Acknowledgements

The authors would like to thank the referees for their valuable comments. This work is partially supported by National Key Research and Development Program (2020AAA0107800), NSFC general projects (No. 61872146), Open Project Fund from Shenzhen Institute of Artificial Intelligence and Robotics for Society, and Joint Funding and AI Project (No. 20DZ1100300) of Shanghai Science and Technology Committee. Min Zhang is the corresponding author.

References

[1] P. Holley, "Texas becomes the latest state to get a self-driving car service," https://shorturl.at/bktzG, May 2018.

[2] Apollo, "An open, reliable and secure software platform for autonomous driving systems," http://apollo.auto, 2018.

[3] Waymo, "A self-driving technology development company," https://waymo.com/, 2009.

[4] D. C. Ciresan, A. Giusti, L. M. Gambardella, and J. Schmidhuber, "Deep neural networks segment neuronal membranes in electron microscopy images," in *Proceedings of the 26th Annual Conference on Neural Information Processing Systems.*, 2012, pp. 2852–2860.

[5] D. Shen, G. Wu, , and H.-I. Suk, "Deep learning in medical image analysis," *Annual Review of Biomedical Engineering*, vol. 19, pp. 221–248, 2017.

[6] T. Parag, D. C. Ciresan, and A. Giusti, "Efficient classifier training to minimize false merges in electron microscopy segmentation," in *Proceedings of 2015 IEEE International Conference on Computer Vision*, 2015, pp. 657–665.

[7] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *ICLP'14*, 2014.

[8] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[9] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: a simple and accurate method to fool deep neural networks," in *CVPR'16*, 2016, pp. 2574–2582.

[10] Y. Lei, S. Chen, L. Fan, F. Song, and Y. Liu, "Advanced evasion attacks and mitigations on practical ML-based phishing website classifiers," *CoRR*, vol. abs/2004.06954, 2020.

[11] G. Chen, S. Chen, L. Fan, X. Du, Z. Zhao, F. Song, and Y. Liu, "Who is real Bob? adversarial attacks on speaker recognition systems," *CoRR*, vol. abs/1911.01840, 2019.

[12] Y. Duan, Z. Zhao, L. Bu, and F. Song, "Things you may not know about adversarial example: A black-box adversarial image attack," *CoRR*, vol. abs/1905.07672, 2019.

[13] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 39–57.

[14] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient smt solver for verifying deep neural networks," in *CAV'17*. Springer, 2017, pp. 97–117.

[15] J. Peck, J. Roels, B. Goossens, and Y. Saeys, "Lower bounds on the robustness to adversarial perturbations," in *Advances in Neural Information Processing Systems*, 2017, pp. 804–813.

[16] M. Hein and M. Andriushchenko, "Formal guarantees on the robustness of a classifier against adversarial manipulation," in *Advances in Neural Information Processing Systems*, 2017, pp. 2266–2276.

[17] K. Dvijotham, R. Stanforth, S. Gowal, T. A. Mann, and P. Kohli, "A dual approach to scalable verification of deep networks." in *UAI'18*, 2018, pp. 550–559.

[18] T.-W. Weng, H. Zhang, H. Chen, Z. Song, C.-J. Hsieh, D. Boning, I. S. Dhillon, and L. Daniel, "Towards fast computation of certified robustness for relu networks," *arXiv preprint arXiv:1804.09699*, 2018.

[19] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel, "Efficient neural network robustness certification with general activation functions," in *NeurIPS'18*, 2018, pp. 4939–4948.

[20] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev, "AI^2: Safety and robustness certification of neural networks with abstract interpretation," in *S&P'18*. IEEE, 2018, pp. 3–18.

[21] G. Singh, T. Gehr, M. Mirman, M. Püschel, and M. Vechev, "Fast and effective robustness certification," in *NeurIPS'18*, 2018, pp. 10 802–10 813.

[22] V. Tjeng, K. Y. Xiao, R. Tedrake *et al.*, "Evaluating robustness of neural networks with mixed integer programming," in *ICLR'19*, 2019.

[23] R. Ehlers, "Formal verification of piece-wise linear feed-forward neural networks," in *ATVA'17*, 2017, pp. 269–286.

[24] G. Singh, T. Gehr, M. Püschel, and M. Vechev, "An abstract domain for certifying neural networks," in *POPL'19*, vol. 3. ACM, 2019, p. 41.

[25] G. Singh, T. Gehr, M. Püschel, and M. T. Vechev, "Boosting robustness certification of neural networks," in *7th International Conference on Learning Representations (ICLR)*, 2019.

[26] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Efficient formal safety analysis of neural networks," in *NeurIPS'18*, 2018, pp. 6367–6377.

[27] Y. Zhang, Z. Zhao, G. Chen, F. Song, and T. Chen, "BDD4BNN: A BDD-based quantitative analysis framework for binarized neural networks," *CoRR*, vol. abs/2103.07224, 2021.

[28] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[29] G. Huang, Z. Liu, L. Van Der Maaten *et al.*, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[30] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[31] W. Ruan, M. Wu, Y. Sun, X. Huang, D. Kroening, and M. Kwiatkowska, "Global robustness evaluation of deep neural networks with provable guarantees for the hamming distance," in *IJCAI'19*, 2019, pp. 5944–5952.

[32] H. Salman, G. Yang, H. Zhang, C. Hsieh, and P. Zhang, "A convex relaxation barrier to tight robustness verification of neural networks," in *32nd NeurIPS*, 2019, pp. 9832–9842.

[33] P. Henriksen and A. R. Lomuscio, "Efficient neural network verification via adaptive refinement and adversarial search," in *24th ECAI*, vol. 325, 2020, pp. 2513–2520.

[34] Y. Wu and M. Zhang, "Tightening robustness verification of convolutional neural networks with fine-grained linear approximation," in *AAAI'21*, 2021, pp. 11 674–11 681.

[35] R. E. Moore, R. B. Kearfott, and M. J. Cloud, *Introduction to interval analysis*. Siam, 2009, vol. 110.

[36] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Formal security analysis of neural networks using symbolic intervals," in *USENIX Security Symposium'18*, 2018, pp. 1599–1614.

[37] C. Liu, T. Arnon, C. Lazarus, C. W. Barrett, and M. J. Kochenderfer, "Algorithms for verifying deep neural networks," *CoRR*, vol. abs/1903.06758, 2019.

[38] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*, 2014, pp. 675–678.

[39] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," *arXiv preprint arXiv:1512.01274*, 2015.

[40] BLAS, "BLAS Library," http://www.netlib.org/blas/, 2018.

[41] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM review*, vol. 59, no. 1, pp. 65–98, 2017.

[42] M. Mirman, T. Gehr, and M. T. Vechev, "Differentiable abstract interpretation for provably robust neural networks," in *ICML'18*, 2018, pp. 3575–3583.

[43] E. Wong, F. R. Schmidt, J. H. Metzen, and J. Z. Kolter, "Scaling provable adversarial defenses," in *NeurIPS'18*, 2018, pp. 8410–8419.

[44] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *ICLR'18*, 2018.

[45] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[46] X. Huang, D. Kroening, W. Ruan, J. Sharp, Y. Sun, E. Thamo, M. Wu, and X. Yi, "Safety and trustworthiness of deep neural networks: A survey," *CoRR*, vol. abs/1812.08342v4, 2019. [Online]. Available: http://arxiv.org/abs/1812.08342v4

[47] K. Pei, Y. Cao, J. Yang, and S. Jana, "Deepxplore: Automated whitebox testing of deep learning systems," in *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP)*, 2017, pp. 1–18.

[48] L. Ma, F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, Y. Liu, J. Zhao, and Y. Wang, "Deepgauge: multi-granularity testing criteria for deep learning systems," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE)*, 2018, pp. 120–131.

[49] N. Papernot, P. D. McDaniel, I. J. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the ACM on Asia Conference on Computer and Communications Security (AsiaCCS)*, 2017, pp. 506–519.

[50] A. Ilyas, L. Engstrom, A. Athalye, and J. Lin, "Black-box adversarial attacks with limited queries and information," in *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018, pp. 2142–2151.

[51] M. Cheng, T. Le, P. Chen, H. Zhang, J. Yi, and C. Hsieh, "Query-efficient hard-label black-box attack: An optimization-based approach," in *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, 2019.

[52] C. Tu, P. Ting, P. Chen, S. Liu, H. Zhang, J. Yi, C. Hsieh, and S. Cheng, "Autozoom: Autoencoder-based zeroth order optimization method for attacking black-box neural networks," in *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI)*, 2019, pp. 742–749.

[53] A. N. Bhagoji, W. He, B. Li, and D. Song, "Exploring the space of black-box attacks on deep neural networks," *CoRR*, vol. abs/1712.09491, 2017.

[54] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[55] W. Brendel, J. Rauber, and M. Bethge, "Decision-based adversarial attacks: Reliable attacks against black-box machine learning models," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

[56] Z. Zhao, G. Chen, J. Wang, Y. Yang, F. Song, and J. Sun, "Attack as defense: Characterizing adversarial examples using robustness," *CoRR*, vol. abs/2103.07633, 2021.

[57] V. Tjeng and R. Tedrake, "Verifying neural networks with mixed integer programming," *arXiv preprint arXiv: 1711.07356*, pp. 945–950, 2017.

[58] W. Liu, F. Song, T. Zhang, and J. Wang, "Verifying ReLU neural networks from a model checking perspective," *Journal of Computer Science and Technology*, vol. 35, no. 6, pp. 1365–1381, 2020.

[59] W. Xiang, H.-D. Tran, and T. T. Johnson, "Output reachable set estimation and verification for multilayer neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 11, pp. 5777–5783, 2018.

[60] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks," in *CAV'17*. Springer, 2017, pp. 3–29.

[61] Y. Y. Elboher, J. Gottschlich, and G. Katz, "An abstraction-based framework for neural network verification," in *CAV'20*, 2020, pp. 43–65.

[62] P. Ashok, V. Hashemi, J. Kretínský, and S. Mohr, "Deepabstract: Neural network abstraction for accelerating verification," in *ATVA'20*, 2020.

[63] J. Li, J. Liu, P. Yang, L. Chen, X. Huang, and L. Zhang, "Analyzing deep neural networks with symbolic propagation: Towards higher precision and faster verification," in *ISSTA'19*, 2019, pp. 296–319.