Global Robustness Evaluation of Deep Neural Networks with Provable Guarantees for L0 Norm

Wenjie Ruan[1], Min Wu[1], Youcheng Sun[1], Xiaowei Huang[2]
Daniel Kroening[1], and Marta Kwiatkowska[1]

[1]University of Oxford, UK
{wenjie.ruan; min.wu; youcheng.sun}@cs.ox.ac.uk
{daniel.kroening; marta.kwiatkowska}@cs.ox.ac.uk
[2]University of Liverpool, UK
xiaowei.huang@liverpool.ac.uk

Abstract. Deployment of deep neural networks (DNNs) in safety or security-critical systems demands provable guarantees on their correct behaviour. One example is the robustness of image classification decisions, defined as the invariance of the classification for a given input over a small neighbourhood of images around the input. Here we focus on the L_0 norm, and study the problem of quantifying global robustness of a trained DNN, where global robustness is defined as the expectation of the maximum safe radius over a testing dataset. We first show that the problem is NP-hard, and then propose an approach to iteratively generate lower and upper bounds on the network's robustness. The approach is *anytime*, i.e., it returns intermediate bounds and robustness estimates that are gradually, but strictly, improved as the computation proceeds; *tensor-based*, i.e., the computation is conducted over a set of inputs simultaneously, instead of one by one, to enable efficient GPU computation; and has *provable guarantees*, i.e., both the bounds and the robustness estimates can converge to their optimal values. Finally, we demonstrate the utility of the proposed approach in practice to compute tight bounds by applying and adapting the anytime algorithm to a set of challenging problems, including global robustness evaluation, guidance for the design of robust DNNs, competitive L_0 attacks, generation of saliency maps for model interpretability, and test generation for DNNs. We release the code of all case studies via Github[1].

1 Introduction

Deep neural networks (DNNs) have achieved significant breakthroughs in the past few years and are now being deployed in many applications. However, in safety-critical domains, where human lives are at stake, and security-critical applications, which often incur significant financial implications, concerns have been raised about the reliability of this technique. In established industries, e.g., avionics and automotive, such concerns can be partially addressed by certifying the product before its deployment. During the certification process, the manufacturer needs to demonstrate to a certification authority, e.g., the European Aviation Safety Agency, Vehicle Certification Agency, etc., that the

[1] https://github.com/L0-TRE/L0-TRE

product behaves correctly with respect to a set of pre-specified high-level requirements. For this purpose, it is necessary to develop techniques for both the discovery of critical requirements as well as the demonstration that these requirements are met by the product.

Safety certification for DNNs is challenging owing to the black-box nature of DNNs and the lack of rigorous foundations. An important low-level requirement for DNNs is the robustness to input perturbations. DNNs have been shown to suffer from poor robustness because of their susceptibility to *adversarial examples* [20]. These are small modifications to the input, sometimes imperceptible to humans, that make the network unstable. As a result, significant effort has been directed towards developing approaches for crafting adversarial examples or defending against them [1,3,14]. However, the cited approaches provide *no* formal guarantees, i.e., no conclusion can be made whether adversarial examples remain or how close crafted adversarial examples are to the optimal ones.

Recent efforts in the area of automated verification [5, 6] have instead focused on methods that generate adversarial examples, if they exist, and provide rigorous robustness proofs otherwise. These techniques rely on either a reduction to a constraint solving problem [6] by encoding the network as a set of constraints, or a layer-by-layer exhaustive search of the neighbourhood of an image [5]. Constraint-based approaches are limited to small networks. Exhaustive search, on the other hand, applies to large networks but suffers from the state-space explosion problem. To mitigate this, a Monte Carlo tree search has been employed [22].

This paper proposes a novel approach to quantify robustness of DNNs that offers a balance between the guaranteed accuracy of the method (thus, a feature so far exclusive to formal approaches) and the efficiency of algorithms that search for adversarial examples (without providing any guarantees). We consider the *global* robustness problem, which is a generalisation of the local, pointwise robustness problem. Specifically, we define a maximum safe radius for every input and then evaluate robustness over a given test dataset, i.e., a finite set of inputs. Global robustness is defined as the expected maximum safe radius over the test examples. We focus on the L_0 norm, which measures the distance between two matrices (e.g., two input images) by counting the number of elements (e.g., pixels) that are different.

The key idea of our approach is to generate sequences of lower and upper bounds for global robustness. Our method is anytime, tensor-based, and offers provable guarantees. First, the method is *anytime* in the sense that it can return intermediate results, including upper and lower bounds and robustness estimates. We prove that our approach can gradually, but strictly, improve these bounds and estimates as the computation proceeds. Second, it is *tensor-based*. As we are working with a set of inputs, a straightforward approach is to perform robustness evaluation for the inputs individually and to then merge the results. However, this is inefficient, particularly given that GPUs are widely available. To enable efficient computation on GPUs, our approach uses tensors. A tensor is a finite set of multi-dimensional arrays, of which each represents an input. A good tensor-based algorithm uses tensor operations whenever possible. Third, our approach offers *provable guarantees*. We show that the intermediate bounds and the robustness

estimates will converge to their optimal values in finite time, although this may be impractical for large networks.

We implement our approach in a software tool L0-TRE (*T*ensor-based Robustness Evaluation for L_0 Norm)[2], and conduct experiments on a set of challenging problems, including *i)* convergence of the global robustness evaluation on MNIST DNNs; *ii)* guidance for the design of robust MNIST DNN architectures; *iii)* competitive L_0 attack on MNIST and CIFAR-10 DNNs; *iv)* saliency map generation for model interpretability on five state-of-the-art ImageNet DNNs, including AlexNet, VGG-16, VGG-19, ResNet50 and ResNet101; and *v)* test case generation on MNIST and CIFAR-10 DNNs.

All the above applications require only simple adaptations of our method, e.g., slight modifications of the constraints or objective functions, or addition of extra constraints, which demonstrates that the technique is flexible enough to deliver a wide range of promising applications. The main contributions of this paper are as follows:

- We propose a novel method to quantify global robustness of DNNs w.r.t. the L_0-norm. This offers two key advantages, including *i)* theoretical lower and upper bounds to guarantee its convergence; and *ii)* explicit tensor-based parallelisation on GPUs with high computational efficiency.
- With simple adaptations, we show the utility of the proposed method on a broad range of practical applications, including *i)* anytime global robustness evaluation; *ii)* guidance for robust DNN design; *iii)* competitive L_0 adversarial attack; *iv)* saliency map generation for large-scale DNNs; and *v)* test case generation for testing of DNNs.
- We perform a rigorous theoretical analysis and extensive empirical case studies to support the claims above.

2 Problem Formulation

Let $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ be an N-layer neural network such that, for a given input $x \in \mathbb{R}^n$, $f(x) = (c_1(x), c_2(x), \ldots, c_m(x)) \in \mathbb{R}^m$ represents the confidence values for m classification labels. Specifically, we have

$$f(x) = f_N(f_{N-1}(\ldots f_1(x; W_1, b_1); W_2, b_2); \ldots); W_N, b_N) \tag{1}$$

where W_i and b_i for $i = 1, 2, \ldots, N$ are learnable parameters, and $f_i(z_{i-1}; W_{i-1}, b_{i-1})$ is the function that maps the output of layer $i-1$, *i.e.,* z_{i-1}, to the intput of layer i. Without loss of generality, we normalise the input to $x \in [0, 1]^n$. The output $f(x)$ is usually normalised to be in $[0, 1]^m$ with a softmax layer. We denote the classification label of input x by $cl(f, x) = \arg\max_{j=1,\ldots,m} c_j(x)$. Note that both f and cl can be generalised to work with a set $\mathtt{T}_0$ of inputs, i.e., $f(\mathtt{T}_0)$ and $cl(f, \mathtt{T}_0)$, in the standard way.

Definition 1 (Safe Norm Ball) *Given a network $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$, an input $x_0 \in \mathbb{R}^n$, a distance metric $||\cdot||_D$, and a real number $d \in \mathbb{R}$, a* norm ball $B(f, x_0, ||\cdot||_D, d)$ *is a*

[2] Available via Github `https://github.com/L0-TRE/L0-TRE`

subspace of $\mathbb{R}^n$ *such that*

$$B(f, x_0, ||\cdot||_D, d) = \{x \mid ||x_0 - x||_D \leq d\}. \tag{2}$$

The number d *is called the radius of* $B(f, x_0, ||\cdot||_D, d)$*. A norm ball* $B(f, x_0, ||\cdot||_D, d)$ *is* safe *if for all* $x \in B(f, x_0, ||\cdot||_D, d)$ *we have* $cl(f, x) = cl(f, x_0)$*.*

Intuitively, a norm ball $B(f, x_0, ||\cdot||_D, d)$ includes all inputs whose distance to x_0, measured by metric $||\cdot||_D$, is within d. We will discuss the definition of the distance metric $||\cdot||_D$ later.

Definition 2 (Maximum Radius of a Safe Norm Ball) *Let* d *be the radius of a safe norm ball* $B(f, x_0, ||\cdot||_D, d)$*. If for all* $d' > d$ *we have that* $B(f, x_0, ||\cdot||_D, d')$ *is not safe, then* d *is called the* maximum safe radius*, denoted by* $d_m(f, x_0, ||\cdot||_D)$*. Formally,*

$$d_m(f, x_0, ||\cdot||_D) = \arg\sup_d \{B(f, x_0, ||\cdot||_D, d) \text{ is safe} \mid d \in \mathbb{R}, d > 0\}. \tag{3}$$

We define the (*global*) robustness evaluation problem over a testing dataset $\mathtt{T}$, which is a set of i.i.d. inputs sampled from a distribution μ representing the problem f is working on. We use $|\mathtt{T}|$ to denote the number of inputs in $\mathtt{T}$. When $|\mathtt{T}| = 1$, we call it *local* robustness.

Definition 3 (Robustness Evaluation) *Given a network* f*, a finite set* $\mathtt{T}_0$ *of inputs, and a distance metric* $||\cdot||_D$*, the* robustness evaluation*, denoted as* $R(f, \mathtt{T}_0, ||\cdot||_D)$*, is an optimisation problem:*

$$\begin{aligned} &\min_{\mathtt{T}} ||\mathtt{T}_0 - \mathtt{T}||_D \\ \textit{s.t.} \quad &cl(f, x_i) \neq cl(f, x_{0,i}) \textit{ for } i = 1 \ldots |\mathtt{T}_0| \end{aligned} \tag{4}$$

where $\mathtt{T} = (x_i)_{i=1\ldots|\mathtt{T}_0|}$*,* $\mathtt{T}_0 = (x_{0,i})_{i=1\ldots|\mathtt{T}_0|}$*, and* $x_i, x_{0,i} \in [0, 1]^n$*.*

Intuitively, we aim to find a minimum distance between the original set $\mathtt{T}_0$ and a new, homogeneous set $\mathtt{T}$ of inputs such that all inputs in $\mathtt{T}_0$ are misclassified. The two sets $\mathtt{T}_0$ and $\mathtt{T}$ are homogeneous if they have the same number of elements and their corresponding elements are of the same type.

L_0 Norm The distance metric $||\cdot||_D$ can be any functional mapping $||\cdot||_D : \mathbb{R}^n \times \mathbb{R}^n \to [0, \infty]$ that satisfies the metric conditions. In this paper, we focus on the L_0 metric. For two inputs x_0 and x, their L_0 distance, denoted as $||x - x_0||_0$, is the number of elements that are different. When working with test datasets, we define

$$\begin{aligned} ||\mathtt{T} - \mathtt{T}_0||_0 &= \mathtt{E}_{x_0 \in \mathtt{T}_0}[||x - x_0||_0] \qquad \text{(our definition)} \\ &= \tfrac{1}{|\mathtt{T}_0|} \textstyle\sum_{x_0 \in \mathtt{T}_0} ||x - x_0||_0 \text{ (all inputs in } \mathtt{T}_0 \text{ are i.i.d.)} \end{aligned} \tag{5}$$

where $x \in \mathtt{T}$ is a homogeneous input of $x_0 \in \mathtt{T}_0$. While other norms such as L_1, L_2 and L_∞ have been widely applied for generating adversarial examples [8, 14], studies based on the L_0 norm are few and far between. In the Appendix, we justify why L_0 is an appropriate metric for our goals.

3 Anytime Robustness Evaluation

The accurate evaluation of robustness of Definition 3 is hard; we give the following result on the computational complexity in the size of the DNN.

Theorem 1 *Let f be a DNN that has convolutional and fully-connected layers with ReLU activation function. When $||\cdot||_D$ is the L_0 norm, $R(f, \mathtt{T}_0, ||\cdot||_D)$ is NP-hard, and there exists a deterministic algorithm that can compute $R(f, \mathtt{T}_0, ||\cdot||_D)$ in time complexity $O((n/2)!(1/\epsilon)^{n/2})$ when the error tolerance for each dimension is $\epsilon > 0$.*

In this paper, we propose to compute lower and upper bounds, and then gradually, but *strictly*, improve the bounds so that the gap between them can eventually be closed in finite time. Although the realistic running time can be long, this *anytime* approach provides pragmatic means to track progress. Experimental results in Section 5 show that our approach is able to achieve *tight* bounds *efficiently* in practice.

Definition 4 (Sequences of Bounds) *Given a robustness evaluation problem $R(f, \mathtt{T}_0, ||\cdot||_D)$, a sequence $\mathcal{L}(\mathtt{T}_0) = \{l_1, l_2, \ldots, l_k\} \in \mathbb{R}$ is an incremental lower bound sequence if, for all $1 \leq i < j \leq k$, we have $l_i \leq l_j \leq R(f, \mathtt{T}_0, ||\cdot||_D)$. The sequence is strict, denoted as $\mathcal{L}_s(\mathtt{T}_0)$, if for all $1 \leq i < j \leq k$, we have either $l_i < l_j$ or $l_i = l_j = R(f, \mathtt{T}_0, ||\cdot||_D)$. Similarly, we can define a decremental upper bound sequence $\mathcal{U}(\mathtt{T}_0)$ and a strict decremental upper bound sequence $\mathcal{U}_s(\mathtt{T}_0)$.*

We will, in Section 4, introduce our algorithms on computing these two sequences of lower and upper bounds. For now, assume they exist, then at a certain time $t > 0$, $l_t \leq R(f, \mathtt{T}_0, ||\cdot||_D) \leq u_t$ holds.

Definition 5 (Anytime Robustness Evaluation) *For a given range $[l_t, u_t]$, we define its centre and radius as follows.*

$$U_c(l_t, u_t) = \frac{1}{2}(l_t + u_t) \text{ and } U_r(l_t, u_t) = \frac{1}{2}(u_t - l_t). \tag{6}$$

The anytime evaluation of $R(f, \mathtt{T}_0, ||\cdot||_D)$ at time t, denoted as $R_t(f, \mathtt{T}_0, ||\cdot||_D)$, is the pair $(U_c(l_t, u_t), U_r(l_t, u_t))$.

The anytime evaluation will be returned whenever the computational procedure is interrupted. Intuitively, we use $U_c(l_t, u_t)$ to represent the current estimation, and $U_r(l_t, u_t))$ to represent its error bound. The following theorem provides a guarantee.

Theorem 2 (Guarantee of Anytime Robustness Evaluation) *Let f be a network and $||\cdot||_D$ a distance metric. At any time $t > 0$, the anytime evaluation $R_t(f, \mathtt{T}_0, ||\cdot||_D) = (U_c(l_t, u_t), U_r(l_t, u_t))$ such that*

$$U_c(l_t, u_t) - U_r(l_t, u_t) \leq R(f, \mathtt{T}_0, ||\cdot||_D) \leq U_c(l_t, u_t) + U_r(l_t, u_t). \tag{7}$$

4 Tensor-based Algorithms for Upper and Lower Bounds

We present our approach to generate the sequences of bounds.

Definition 6 (Complete Set of Subspaces for an Input) *Given an input $x_0 \in [0,1]^n$ and a set of t dimensions $T \subseteq \{1,...,n\}$ such that $|T| = t$, the subspace for x_0, denoted by $X_{x_0,T}$, is a set of inputs $x \in [0,1]^n$ such that $x(i) \in [0,1]$ for $i \in T$ and $x(i) = x_0(i)$ for $i \in \{1,...,n\} \setminus T$. Furthermore, given an input $x_0 \in [0,1]^n$ and a number $t \leq n$, we define*

$$\mathcal{X}(x_0, t) = \{X_{x_0,T} \mid T \subseteq \{1,...,n\}, |T| = t\} \tag{8}$$

as the complete set of subspaces for input x_0.

Intuitively, elements in $X_{x_0,T}$ share the same value with x_0 on the dimensions other than T, and may take any legal value for the dimensions in T. Moreover, $\mathcal{X}(x_0, t)$ includes all sets $X_{x_0,T}$ for any possible combination T with t dimensions.

Next, we define the subspace sensitivity for a subspace w.r.t. a network f, an input x_0, and a testing dataset $\mathtt{T}_0$. Recall that $f(x) = (c_1(x), c_2(x), ..., c_m(x))$.

Definition 7 (Subspace Sensitivity) *Given an input subspace $X \subseteq [0,1]^n$, an input $x_0 \in [0,1]^n$, and a label j, the subspace sensitivity w.r.t. X, x_0, and j is defined as*

$$S(X, x_0, j) = c_j(x_0) - \inf_{x \in X} c_j(x). \tag{9}$$

Let t be an integer. We define the subspace sensitivity for $\mathtt{T}_0$ and t as

$$S(\mathtt{T}_0, t) = (S(X_{x_0}, x_0, j_{x_0}))_{X_{x_0} \in \mathcal{X}(x_0,t), x_0 \in \mathtt{T}_0} \tag{10}$$

where $j_{x_0} = \arg\max_{i \in \{1,...,m\}} c_i(x_0)$ is the classification label of x_0 by network f.

Intuitively, $S(X, x_0, j)$ is the maximal decrease of confidence value of the output label j that can be witnessed from the set X, and $S(\mathtt{T}_0, t)$ is the two-dimensional array of the maximal decreases of confidence values of the classification labels for all subspaces in $\mathcal{X}(x_0, t)$ and all inputs in $\mathtt{T}_0$. It is not hard to see that $S(X, x_0, j) \geq 0$.

Given a testing dataset $\mathtt{T}_0$ and an integer $t > 0$, the number of elements in $S(\mathtt{T}_0, t)$ is in $O(|\mathtt{T}_0|n^t)$, i.e., polynomial in $|\mathtt{T}_0|$ and exponential in t. Note that, by Equation (9), every element in $S(\mathtt{T}_0, t)$ represents an optimisation problem. E.g., for $\mathtt{T}_0$, a set of 20 MNIST images, and $t = 1$, this would be $28 \times 28 \times 20 = 15,680$ one-dimensional optimisation problems. In the next section, we give a tensor-based formulation and an algorithm to solve this challenging problem via GPU parallization.

4.1 Tensor-based Parallization for Computing Subspace Sensitivity

A tensor $\mathcal{T} \in \mathbb{R}^{I_1 \times I_2 \times ... \times I_N}$ in an N-dimensional space is a mathematical object that has $\prod_{m=1}^{N} I_m$ components and obeys certain transformation rules. Intuitively, tensors are generalisations of vectors (*i.e.,* one index) and matrices (*i.e.,* two indices) to an arbitrary number of indices. Many state-of-the-art deep learning libraries, such as Tensorflow and Keras, are utilising the tensor format to parallelise the computation with

GPUs. However, it is nontrivial to write an algorithm working with tensors due to limited tensor-based operations. We will introduce tensor-based operations when needed.

The basic idea of our algorithm is to transform a set of nonlinear, noncovex optimisation problems as given in Equation (10) into a tensor formulation, and solve a set of optimisation problems via few DNN queries.

First, we introduce the following tensor operations used in our algorithm.

Definition 8 (Mode-n Unfolding and Folding) *Given a tensor $\mathcal{T} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$, the mode-n unfolding of $\mathcal{T}$ is a matrix $\mathbf{U}_{[n]}(\mathcal{T}) \in \mathbb{R}^{I_n \times I_M}$, such that $M = \prod_{k=1,k\neq n}^{N} I_k$ and $\mathbf{U}_{[n]}(\mathcal{T})$ is defined by the mapping from element $(i_1, i_2, \ldots, i_N)$ to (i_n, j), with*

$$j = \sum_{k=1,k\neq n}^{N} i_k \times \prod_{m=k+1,k\neq n}^{N} I_m.$$

Accordingly, tensor folding $\mathbf{F}$ is to fold an unfolded tensor back from matrix to a full tensor. Tensor unfolding and folding are dual operations linking tensors and matrices.

Given a neural network f, a number t, and a testing dataset $\mathtt{T}_0$, each $x_i \in \mathtt{T}_0$ generates a complete set $\mathcal{X}(x_i, t)$ of subspaces. Let $|\mathtt{T}_0| = p$ and $|\mathcal{X}(x_i, t)| = k$. It is noted that, for different x_i and x_j, we have $|\mathcal{X}(x_i, t)| = |\mathcal{X}(x_j, t)|$. Given an error tolerance $\epsilon > 0$, by applying grid search, we can recursively sample $\Delta = 1/\epsilon$ numbers in each dimension, and turn each subspace $X_{x_i} \in \mathcal{X}(x_i, t)$ into a two-dimensional grid $\mathcal{G}(X_{x_i}) \in \mathbb{R}^{n \times \Delta^t}$. We can formulate the following tensor

$$\mathcal{T}(\mathtt{T}_0, t) = \text{Tensor}((\mathcal{G}(X_{x_i}))_{x_i \in \mathtt{T}_0, X_{x_i} \in \mathcal{X}(x_i,t)}) \in \mathbb{R}^{n \times \Delta^t \times p \times k} \tag{11}$$

Theorem 6 in Appendix A.1 provides the error bound for reaching the global minimum by grid search.

Then, we apply the mode-1 tensor unfolding operation to have $\mathbf{T}_{[1]}(\mathcal{T}(\mathtt{T}_0, t)) \in \mathbb{R}^{n \times M}$ such that $M = \Delta^t * p * k$. Then this tensor can be fed into the DNN f to obtain

$$Y(\mathtt{T}_0, t) = f(\mathbf{T}_{[1]}(\mathcal{T}(\mathtt{T}_0, t))) \in \mathbb{R}^M. \tag{12}$$

After computing $Y(\mathtt{T}_0, t)$, we apply a tensor folding operation to obtain

$$\mathcal{Y}(\mathtt{T}_0, t) = \mathbf{F}(Y(\mathtt{T}_0, t)) \in \mathbb{R}^{\Delta^t \times p \times k}. \tag{13}$$

Here, we should note the difference between $\mathbb{R}^{\Delta^t * p * k}$ and $\mathbb{R}^{\Delta^t \times p \times k}$, with the former being a one-dimensional array and the latter a tensor. On $\mathcal{Y}(\mathtt{T}_0, t)$, we search the minimum values along the first dimension to obtain[3]

$$V(\mathtt{T}_0, t)_{min} = \min(\mathcal{Y}(\mathtt{T}_0, t), 1) \in \mathbb{R}^{p \times k} \tag{14}$$

Until now, we have solved all $p \times k$ optimisation problems. Then we construct the tensor

$$V(\mathtt{T}_0, t) = (\overbrace{c_{j_{x_i}}(x_i), ..., c_{j_{x_i}}(x_i)}^{k})_{x_i \in \mathtt{T}_0} \in \mathbb{R}^{p \times k} \tag{15}$$

[3] Here we use a Matlab notation $\min(Y, k)$, which finds minimum values over the k-th dimension for a multi-dimensional array Y. Other notations to appear later are similar.

from the set $\mathtt{T}_0$. Recall that $j_{x_i} = \arg\max_{k\in\{1,\ldots,m\}} c_k(x_i)$. Intuitively, $V(\mathtt{T}_0, t)$ is the tensor that contains the starting points of the optimisation problems and $V(\mathtt{T}_0, t)_{min}$ the resulting optimal values. The following theorem shows the correctness of our computation, where $S(\mathtt{T}_0, t)$ has been defined in Definition 7.

Theorem 3 *Let $\mathtt{T}_0$ be a testing dataset and t an integer. We have $S(\mathtt{T}_0, t) = V(\mathtt{T}_0, t) - V(\mathtt{T}_0, t)_{min}$.*

For the above computation, we need only one DNN query in Equation (12).

4.2 Tensor-based Parallelization for Computing Lower and Upper Bounds

Let $\mathcal{S}(\mathtt{T}_0, t) \in \mathbb{R}^{n\times p\times k}$ be the tensor obtained by replacing every element in $S(\mathtt{T}_0, t)$ with their corresponding inputs which, according to the computation of $V(\mathtt{T}_0, t)_{min}$, cause the largest decreases on the confidence values of the classification labels. We call $\mathcal{S}(\mathtt{T}_0, t)$ the solution tensor of $S(\mathtt{T}_0, t)$. We remark that the computation of $\mathcal{S}(\mathtt{T}_0, t)$ can be done using few tensor operations over $\mathcal{T}(\mathtt{T}_0, t)$ and $\mathcal{Y}(\mathtt{T}_0, t)$, which have been given in Section 4.1. For the simplification of the technicalities, we omit the details.

Lower Bounds We reorder $S(\mathtt{T}_0, t)$ and $\mathcal{S}(\mathtt{T}_0, t)$ w.r.t. the decreased values in $S(\mathtt{T}_0, t)$. Then, we retrieve the first row of the third dimension in tensor $\mathcal{S}(\mathtt{T}_0, t)$, *i.e.,* $\mathcal{S}(\mathtt{T}_0, t)[:, :, 1] \in \mathbb{R}^{n\times p}$, and check whether $cl(f, \mathcal{S}(\mathtt{T}_0, t)[:, :, 1]) = cl(f, \mathtt{T}_0)$. The result is an array of values in $\{True, False\}$, each of which is associated with an input $x_i \in \mathtt{T}_0$. If any element associated with x_i in the resulting array is $False$, we conclude that $d_m(f, x_i, ||\cdot||_D) = t - 1$, i.e., the maximum safe radius has been obtained and the computation for x_i has converged. On the other hand, if the element associated with x_i is $True$, we updated the lower bound for x_i into t.

Note that, after $\mathcal{S}(\mathtt{T}_0, t)$, no additional DNN query is needed for the lower bounds.

Upper Bounds Upper bounds are computed by iteratively applying perturbations based on the matrix $\mathcal{S}(\mathtt{T}_0, t)$ for every input in $\mathtt{T}_0$ until a misclassification occurs. However, doing this sequentially for all the inputs would be inefficient, since we need to query the network f after every perturbation on each image.

We design an efficient tensor-based algorithm to enable GPU parallelization. The key idea relies on the fact that we construct a new tensor $\mathcal{N} \in \mathbb{R}^{n\times p\times k}$ to maintain all the cumulated perturbations over the original inputs $\mathtt{T}$.

- Inilization: $\mathcal{N}[:, :, 1] = \mathcal{S}(\mathtt{T}_0, t)[:, :, 1]$.
- Iteratively construct the i-th row until $i = k$:

$$\begin{aligned}\mathcal{N}[:, :, i] = \{\mathcal{N}[:, :, i-1] \boxminus \{\mathcal{N}[:, :, i-1] \Cap \mathcal{S}(\mathtt{T}_0, t)[:, :, i]\}\} \\ \Cup\{\mathcal{S}(\mathtt{T}_0, t)[:, :, i] \boxminus \{\mathcal{N}[:, :, i-1] \Cap \mathcal{S}(\mathtt{T}_0, t)[:, :, i]\}\}\end{aligned} \tag{16}$$

where $\boxminus$, $\Cap$, and $\Cup$ are tensor operations such that $\mathcal{N}_1 \boxminus \mathcal{N}_2$ is to remove the corresponding non-zero elements in $\mathcal{N}_2$ from $\mathcal{N}_1$, $\mathcal{N}_1 \Cap \mathcal{N}_2$ is to maintain those elements that have the same values and make other elements have value 0, and $\mathcal{N}_1 \Cup \mathcal{N}_2$ is to merge the non-zero elements from two tensors. For all these operations, the operands

need to have the same shape. Intutively, $\mathcal{N}[:,:,i]$ represents the result of applying the first i perturbations recorded in $\mathcal{S}(\mathtt{T}_0, t)[:,:,1:i]$.

Then, we perform the mode-1 tensor unfolding on $\mathcal{N}$ to have $\mathbf{U}_{[1]}(\mathcal{N}) \in \mathbb{R}^{n\times(p*k)}$, which are fed into the DNN f and get the classification labels: $Y(\mathbf{U}_{[1]}(\mathcal{N})) \in \{1..m\}^{p*k}$. After that, a tensor folding operation is applied to have $\mathcal{Y}(\mathbf{U}_{[1]}(\mathcal{N})) \in \{1..m\}^{p\times k}$. Finally, we can get the minimum column index along each row such that misclassification happens, written as $\{m_1, m_2, ..., m_p\}$ such that $1 \leq m_i \leq k$. Then we let

$$\mathtt{T} = \{\mathcal{N}_{:,i,m_i} \in \mathbb{R}^{n\times p} \mid x_i \in \mathtt{T}_0\} \tag{17}$$

which are the optimal set of inputs as required in Definition 3.

We can see that, after $\mathcal{S}(\mathtt{T}_0, t)$, the computation of all upper bounds for a testing dataset $\mathtt{T}_0$ uses one additional DNN query.

Upper Bounds Tightening After obtaining $\mathtt{T}$, we notice that there may be redundancies in $\mathtt{T}-\mathtt{T}_0$, i.e., not all the changes in $\mathtt{T}-\mathtt{T}_0$ are necessary for the misclassifications. Therefore, we need an approach to reduce the redundancies and tighten the upper bounds. In this paper, we handle the tightening problem as an optimisation problem similar to that of Definition 3, so that we can reuse the tensor-based algorithms developed earlier.

Assume that $x_{0,i}$ and x_i are two corresponding inputs in $\mathtt{T}$ and $\mathtt{T}_0$, respectively, for $i \in \{1, ..., |\mathtt{T}_0|\}$. By abusing the notations, we let $z_{0,i} = x_{0,i} - x_i$ be the part of $x_{0,i}$ on which $x_{0,i}$ and x_i are different, and $l_{0,i} = x_i \Cap x_{0,i}$ be the part of $x_{0,i}$ on which $x_{0,1}$ and x_i are the same. Therefore, $x_{0,i} = z_{0,i} \Cup l_{0,i}$.

Definition 9 (Tightening Upper Bounds) *Given a network f, a finite testing dataset $\mathtt{T}_0$ with their upper bounds $\mathtt{T}$, and a distance metric $||\cdot||_D$, the tightening problem is an optimisation problem:*

$$\begin{aligned} &\min_{\mathtt{L}_1} ||\mathtt{L}_0 - \mathtt{L}_1||_D \\ &\textit{s.t.} \quad cl(f, z_{0,i} \Cup l_{1,i}) \neq cl(f, z_{0,i} \Cup l_{0,i}) \textit{ for } i = 1...|\mathtt{L}_0| \end{aligned} \tag{18}$$

where $\mathtt{L}_0 = (l_{0,i})_{i=1...|\mathtt{T}_0|}$, $\mathtt{L}_1 = (l_{1,i})_{i=1...|\mathtt{L}_0|}$, $l_{1,i}, l_{0,i} \in [0,1]^{|\mathtt{L}_0|}$ have the same shape.

Therefore, we can re-use the tensor-based algorithm for computing lower bounds, with the differences in terms of querying the DNN. Before querying DNN, we need to use the $\Cup$ operation to merge with $z_{0,i}$, as indicated in Equation (18).

4.3 Convergence Analysis

We perform convergence analysis of the proposed methods. For simplicity, in the proofs we consider the case of a single input x_0. The convergence guarantee can be easily generalised to the testing dataset $\mathtt{T}_0$ by following the procedure described in Section 4.1 and 4.2.

Theorem 4 (Guarantee for Lower Bounds) *Given a DNN f, and an input $x_0 \in [0,1]^n$, if our method generates a lower bound $l(f, x_0)$ then, for all x such that $||x - x_0||_0 \leq l(f, x_0)$, we have that $cl(f, x) = cl(f, x_0)$,* i.e., *f is guaranteed to be safe for any pixel perturbations with at most $l(f, x_0)$ pixels.*

Theorem 4 (proof in Appendix A.2) shows that the lower bounds generated by our algorithm are the lower bounds of $d_m(f, x_0, ||\cdot||_D)$. In our method, we gradually increase $t = l(f, x_0)$ and run the lower bound generation algorithm. Because the number of dimensions in an input is finite, the distance to an adversarial example is also finite. Therefore, the lower bound generation algorithm will eventually converge in finite time.

Theorem 5 (Guarantee for Upper Bounds) *Given a DNN f and an input $x_0 \in [0,1]^n$, we let our method generate an upper bound $u_i(f, x_0)$ for every time $i > 0$. Then we have $u_{i+1}(f, x_0) \leq u_i(f, x_0)$ for all $i > 0$, and $\lim_{i \mapsto \infty} u_i(f, x_0) = d_m(f, x_0, ||\cdot||_D)$.*

The three key ingredients in the proof of the monotonic decrease of upper bounds are: *i)* the complete subspaces generated at $t = i$ are always included in the complete subspaces at $t = i + 1$; *ii)* the pixel perturbation from subspace with higher priority always results in a larger confidence decrease than those with lower priority; and *iii)* the tightening strategy is able to exclude all the redundant pixel perturbations. The details of the proof are in Appendix A.3. Finally, we can show that radius of $[l_i, u_i]$ will converge to 0 deterministically (see Appendix A.4).

5 Case Studies

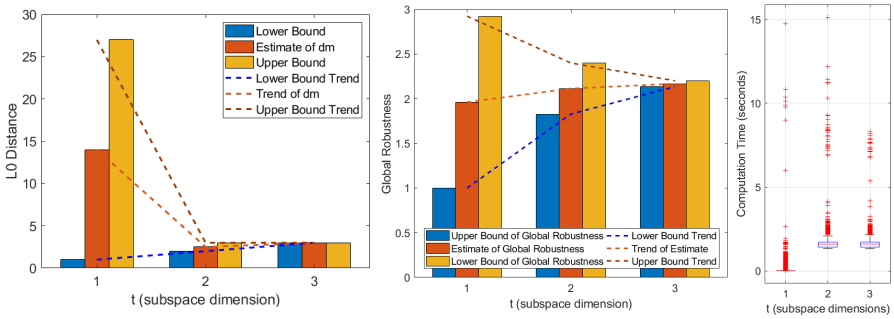
We validate our tool on a set of case studies. Some case studies involve simple adaptations to the optimisation problem in Definition 3, by changing or adding constraints. No significant adaptation is needed for our algorithm to work with these variants, demonstrating the flexibility of our method in working with a set of challenging problems. The data for our case studies is released via Github[4].

5.1 Case Study One: Convergence Analysis and Global Robustness Evaluation

We study the convergence and running time of our anytime global robustness evaluation algorithm on several DNNs in terms of the L_0-norm. To the best of our knowledge, no baseline method exists for this case study. The L_0-norm based algorithms, which we compared against in Section 5.3, cannot perform robustness evaluation based on both lower and upper bounds.

We train two DNNs on the MNIST dataset, with DNN-0 being trained on the original images of size 28×28 and sDNN on images resized into 14×14. The model structures are given in Appendix D. For DNN-0, we work with a set of 2,400 images randomly sampled from the dataset, and for sDNN, we work with a set of 5,300 images.

[4] `https://github.com/L0-TRE/L0-TRE`



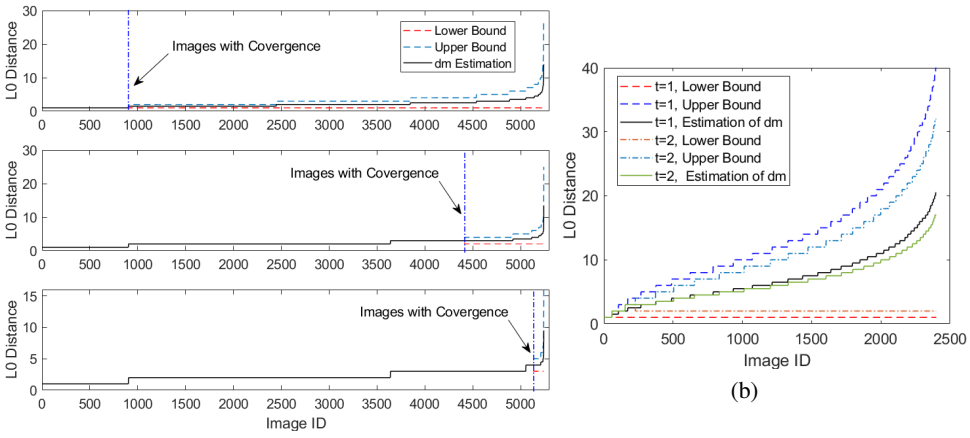
(a) (b) (c)

Fig. 1: (a) Convergence of lower bound, upper bound, and estimation of d_m for one image; (b) Convergence of lower bound, upper bound, and estimation of global robustness; (c) The boxplots of computational time at $t = 1, 2, 3$.

sDNN: Convergence Analysis and Robustness Evaluation Fig. 1 (a) shows the convergence of lower and upper bounds as well as the estimation of d_m (i.e., the maximum safe radius) for an image with a large initial upper bound for the L_0 distance being 27. This image is chosen to conduct the *worst case* analysis of our approach. Working with a single image (i.e., local robustness) is the special case of our optimisation problem where $|\mathtt{T}| = 1$. We observe that, when transitioning from $t = 1$ to $t = 2$, the uncertainty radius $U_r(l_t, u_t)$ of d_m is significantly reduced from 26 to 1, which demonstrates the effectiveness of our upper bound algorithm. Fig. 1 (b) shows the convergence of global robustness evaluation on the testing dataset. The trend clearly shows that our method can achieve tight upper and lower bounds efficiently, and that the anytime global robustness converges. Surprisingly, at $t = 1$, we have that $U_c(l_t, u_t) = 1.97$ and the converged global robustness is 2.1, i.e., the relative error of the estimation of global robustness at $t = 1$ is less than 7%. Because it takes polynomial time to compute the case $t = 1$, this observation suggests that, for this challenging NP-hard problem, our approach provides a good approximation with reasonable error but requiring significantly fewer computational resources.

Fig. 1 (c) depicts the boxplots of computational time at different iterations (*i.e.,* subspace dimensions t). Our tensor-based algorithm is efficient, especially at $t = 1$ where it takes less than 0.1 s to deal with one image, revealing its *potential for run-time evaluation*. In Fig. 2 (a), we plot the upper and lower bounds as well as the estimation of d_m for all images in the testing dataset. Images are ordered with respect to their upper bounds at $t = 1$. The dashed blue line indicates that all images before this line have converged. The charts show a clear overall trend for convergence. When we increase t by only a few steps, for most of the images our algorithm can efficiently find the true d_m.

DNN-0: Global Robustness Evaluation Fig. 2 (b) shows the overall convergence trends for all 2,400 images. As we can see, even for such a DNN with tens of thousands hidden neurons, L0-TRE can still achieve tight estimates of d_m for most images. Fig. 3 shows



(a) (b)

Fig. 2: (a) sDNN: Upper bounds, lower bounds, and estimations of d_m for all sampled images for $t = 1, 2, 3$ ordered from the top to bottom. (b) DNN-0: Upper bounds, lower bounds, and estimations of d_m for all sampled input images at $t = 1, 2$.

the results of anytime global robustness evaluation at $t = 1$ and $t = 2$. They demonstrate the feasibility of applying our approach to run-time global robustness evaluation of safety-critical systems: as shown in the boxplot, given a limited time/computation budget (such as in sub-second level for a real-time system, or only with low-computational capacity GPU for an embedded system), our method is able to evaluate global robustness. Fig. 11 in Appendix D gives some adversarial images returned by our upper bound algorithm.

5.2 Case Study Two: Guidance for the Design of Robust DNN Architecture

We trained 7 DNNs, named DNN-i for $i \in \{1, \ldots, 7\}$, on the MNIST dataset with the same hardware and software platform, and identical training parameters. These DNNs have different architectures, in terms of layer numbers and layer types, and different testing accuracies (training accuracies are all 100%). Details of the models are in Appendix E. Fig. 4 (a) gives details of the global robustness estimations, and their upper and lower bounds at $t = 1$ and $t = 2$ for all seven DNNs. Fig. 5 and Fig. 12 (Appendix E) illustrate the means and standard derivations of d_m estimations and uncertainty radius of all 1,000 sampled testing images, respectively. We observe clear convergence for all DNNs.

More importantly, we make the following observations by analysing their architectures: *i)* number of layers: a very deep DNN (*i.e.,* too many layers relative to the size of the training dataset) would be less robust, such as DNN-7; *ii)* convolutional layers: it is possible that a DNN with more convolutional layers can be less robust, e.g., compared with DNN-5, DNN-6 has one additional convolutional layer, but is significantly less robust; *iii)* batch-normalisation layer: adding a batch-normalisation layer may improve robustness, e.g., DNN-3 is more robust than DNN-2; *iv)* testing accuracy: a DNN

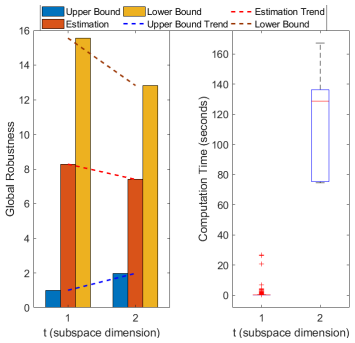


Fig. 3: Anytime robustness evaluation of DNN-0 at $t = 1, 2$ and box-plots of computation time.

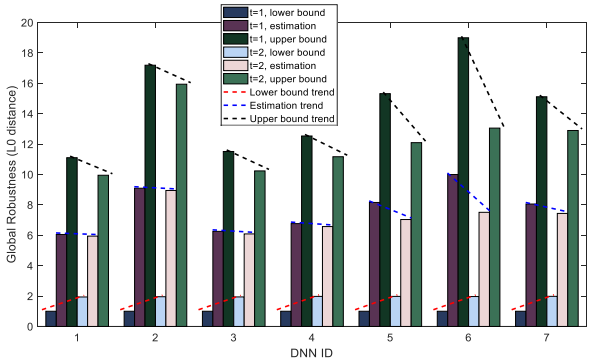


Fig. 4: Lower bounds, upper bounds, and global robustness estimations at $t = 1, 2$ for seven DNN models.

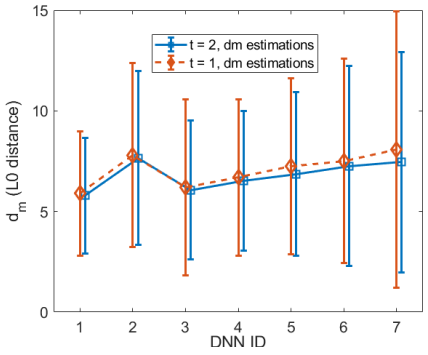


Fig. 5: Means and standard derivations of d_m estimations for tested images of DNN models.

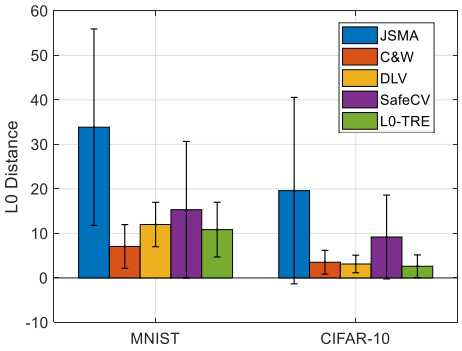


Fig. 6: Means and standard deviations of the adversarial L_0 distance.

with higher testing accuracy is not necessarily more robust, e.g., DNN-1 and DNN-3 are more robust than DNN-6 and DNN-7, which have higher testing accuracies. DNNs may need a balance between robustness and their ability to generalise (represented as testing accuracy). DNN-4 seems to be a good example, among our limited model choices.

Therefore, instead of completely relying on the designer's experience and intuition in selecting model structures, our method can provide an efficient robustness evaluation of the details of the architecture of a DNN.

5.3 Case Study Three: Competitive L_0 Attack

Our upper bound generation method can serve as a competitive L_0 attack by constraining `T` as a singleton set. We use a substantial set of benchmarks from MNIST and CIFAR-10 DNNs and compare with four state-of-the-art L_0-based attacking techniques,

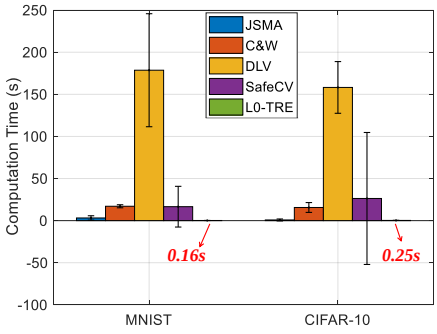


Fig. 7: Means and standard deviations of computational time of all methods.

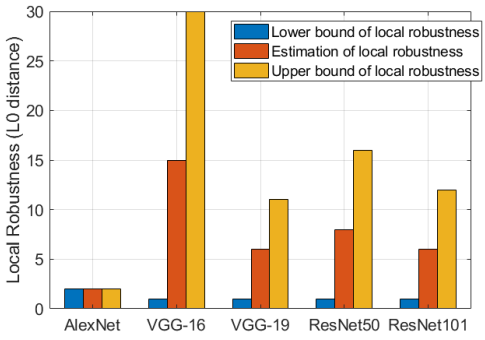


Fig. 8: Upper bound, lower bound and estimation of local robustness for ImageNet DNNs.

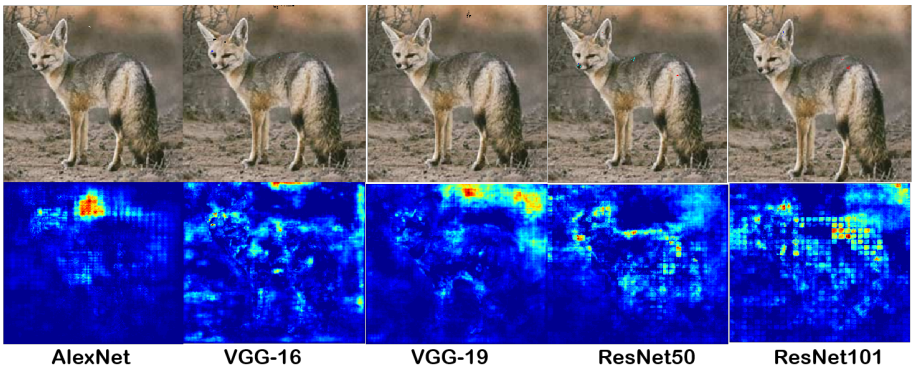


Fig. 9: Adversarial examples on upper boundaries (top row), saliency maps (bottom row).

including JSMA [14], C&W [1], DLV [5], and SafeCV [22], on 1,000 testing images. Details of the experimental settings are given in Appendix F. Note that adversarial attacks is not the primary goal of our method. Nevertheless, the empirical results indeed show the superiority of our method in terms of the effectiveness and, especially, efficiency.

Adversarial L_0 Distance: Fig. 6 depicts the average and standard deviations of L_0 distances of the adversarial images produced by the five methods. A smaller L_0 distance indicates the method applied can find an adversarial example closer to the original image, thus the classification should be invariant. For MNIST, the performance of our method is better than JSMA, DLV and SafeCV, and comparable to C&W. For CIFAR-10, the bar-chart reveals that the proposed L0-TRE method achieves the smallest L_0 distance (*i.e.,* by modifying 2.62 pixels on average) among all methods. For this experiment, we set $t = 1$, without taking further optimisation steps. On the other hand, as

shown in Fig. 7, C&W is around 62 to 100 times slower than L0-TRE since it implements sophisticated optimisation techniques as well as parameter tuning. . .

Computational Efficiency: Fig. 7 compares the time efficiency of all methods running on the same hardware platform. Our tensor-based parallelisation method enables extremely efficient attacks. For example, for MNIST, our method is $18\times$, $100\times$, $1050\times$, and $357\times$ faster than JSMA, C&W, DLV, and SafeCV, respectively. Applying this highly efficient algorithm to real-time systems is promising future work. Appendix F provides some adversarial examples found by all five methods. As we can see, by modifying 1 to 3 pixels, L_0 attacks can fool a well-trained neural network, leading to a misclassification.

5.4 Case Study Four: Saliency Map and Local Robustness Evaluation for Large-scale ImageNet DNN Models

Local Robustness Evaluation for ImageNet Models We apply our method to five state-of-the-art ImageNet DNN models, including AlexNet (8 layers), VGG-16 (16 layers), VGG-19 (19 layers), ResNet50 (50 layers), and ResNet101 (101 layers). We set $t = 1$ and generate the lower/upper bounds and estimates of local robustness on a certain input image. Fig. 8 shows local robustness evaluations and their bounds for these networks. The adversarial images on the upper boundaries are shown in the top row of Fig. 9. For AlexNet, on this specific image, L0-TRE is able to find its ground-truth adversarial example (local robustness converges at L_0 = 2). We also observe that, for this image, the most robust model is VGG-16 (local robustness = 15) and the most vulnerable one is AlexNet (local robustness = 2). Fig. 8 also reveals that, for similar network structures such as VGG-16 and VGG-19, ResNet50 and ResNet101, a model with deeper layers is less robust to adversarial perturbation. This observation is consistent with our conclusion in Case Study Three.

Saliency Map Generation Model interpretability (or explainability) has become a hot topic due to the black-box nature of DNNs and their inability to provide explanations of the decisions. In recent advances, such as [10], this problem is addressed by considering the contributions of each input dimension to the output decision. Our computation of subspace sensitivity $S(\mathtt{T}, t)$ can be re-used to compute such a contribution for each pixel of an image. As shown in Fig. 9 (see more examples in Appendix F.5), a brighter area means it is more vulnerable, which actually explains why VGG-16 is the most robust model while AlexNet is not. For AlexNet, there exists a very bright area, where a slight perturbation can lead to a misclassification. On the contrary, there are no obvious vulnerable spots in VGG-16 (for this image). It is worth mentioning that the optimisation constraints of Definition 3 can be adapted to generate saliency maps for hidden neurons, which can potentially be a contribution to the explainability of DNNs, as in [13].

5.5 Case Study Five: DNNs Test Case Generation

We apply our optimisation method to generate test cases for DNNs, following the neuron coverage criterion [16]. The neuron coverage requests that every hidden neuron in

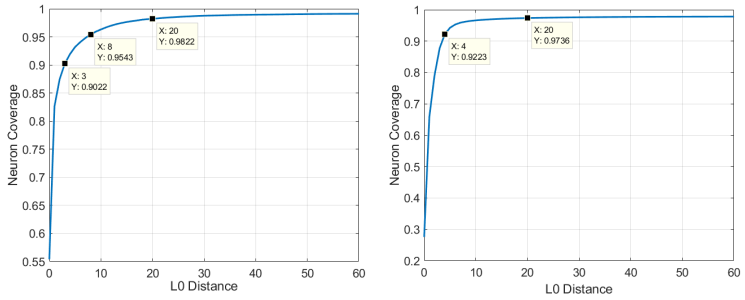


Fig. 10: Neuron coverage by robustness evaluation on MNIST (left) and CIFAR-10 (right).

the network must be activated (i.e., having activation value greater than some threshold) at least once. We use ne to range over hidden neurons, and $V(ne, x)$ to denote the activation value of ne, when the input is x. Then $V(ne, x) > 0$ means that the test requirement for ne is covered by x.

The generation of test cases is non-trivial. In [16], a gradient based technique is combined with the neuron coverage goal as a dual optimisation problem in order to generate test cases. This is ad-hoc. On the contrary, our optimisation method can be straightforwardly applied to coverage based test case generation, by slightly adapting the optimisation problem in Definition 3. Specifically, given any neuron ne that is not activated by inputs in $\mathtt{T}_0$, to find the input with minimised distance to $\mathtt{T}_0$ that activates ne corresponds to replacing the constraint $cl(f, x_i) \neq cl(f, x_{0,i})$ in Equation (4) with

$$V(ne, x_i) \leq 0 \wedge V(ne, x_{0,i}) > 0. \tag{19}$$

That is, instead of considering the change of classification label, now the optimisation problem finds new inputs that are able to make the particular neuron activated, with the objective of minimised distance from the original inputs $\mathtt{T}_0$. Subsequently, in the experiment of test case generation for DNNs, an arbitrary input image is first selected, then for every neuron in the DNN that is not activated, our optimisation method is applied to find the input (test case) that activates this neuron.

The neuron coverage level by test suites generated for two DNNs on MNIST and CIFAR-10 are presented in Fig. 10. The horizontal axis measures the L_0 distance and the vertical axis records the coverage percentage. We can see that our optimisation approach is very effective for neuron coverage. Most neurons can be covered by only modifying a small number of pixels. Fig. 19 in Appendix H further depicts the percentage of adversarial examples found, within corresponding distances, for the two cases. A significant portion of adversarial examples can be found in the relatively small L_0 distance end of the curve. Overall, we find that our method offers a convenient approach for criteria guided testing of DNNs.

6 Related Work

In the following, we review further related work. Owing to space limitations, this review is by no means complete.

Adversarial Example Generation Existing algorithms actually compute an upper bound of the maximum safety radius. However, they cannot guarantee to reach the maximum safety radius, while our method is able to produce both lower and upper bound with provable guarantee for convergence to maximum safety radius. Most of the algorithms proceed by first computing a gradient (either a cost gradient or a forward gradient) and then perturbing in different ways the input along the most promising direction. *FGSM* (Fast Gradient Sign Method) [3] is for the L_∞ norm. It computes the gradient $\nabla_X J(\theta, x, f(x))$ and then lets the perturbed input x' be $x + \epsilon \text{sign}(\nabla_X J(X, y_{true}))$, where ϵ is the multitude of change along the $sign$ direction of the gradient. *JSMA* (Jacobian Saliency Map based Attack) [14] is for the L_0 norm. It calculates the Jacobian matrix of a DNN's output (in the logit layer) with respect to the input. Then it iteratively modifies one or two pixels until the misclassification occurs. The *C&W Attack* (Carlini and Wagner) [1] works for the L_0, L_2, and L_∞ norms. It formulates the search for an adversarial example as an image distance minimisation problem. The basic idea is to introduce a new optimisation variable to avoid box constraint (image pixel needs to lie within $[0, 1]$). *DeepFool* [12] works for the L_2 norm. It iteratively linearises the network around the input x and moves across the boundary by a minimal step until reaching a misclassification. *VAT* (Visual Adversarial Training) [11] defines a KL-divergence at an input based on the model's robustness to the local perturbation of the input, and then perturbs the input according to the KL-divergence. We have shown that, for perturbations based on the L_0 norm, our approach performs better than the existing approaches in not only finding tighter upper bound but also running more efficiently.

Safety Verification and Reachability Analysis The approaches aim to not only find an upper bound but also provide guarantees on the obtained bound. There are two ways of achieving safety verification for DNNs. The first is to reduce the problem to a constraint solving problem. Notable works include, e.g., [6,17]. However, they can only work with small networks that have hundreds of hidden neurons. The second is to discretise the vector spaces of the input or hidden layers, and then apply exhaustive search algorithms or Monte-Carlo tree search algorithm on the discretised spaces. The guarantees are achieved by establishing local assumptions such as minimality of manipulations in [5] and minimum confidence gap for Lipschitz networks in [22]. Moreover, [9] considers determining if an output value of a DNN is reachable from a given input subspace, and reduces the problem to a MILP problem; and [2] considers the range of output values from a given input subspace. Both approaches can only work with small networks. We also mention [15], which computes a lower bound of local robustness for the L_2 norm by propagating relations between layers backward from the output. It is incomparable with ours because of the different distance metrics. The bound is loose and cannot be improved (i.e., no convergence).

7 Conclusions

In this paper, to evaluate global robustness of a DNN over a testing dataset, we present an approach to iteratively generate its lower and upper bounds. We show that the bounds are gradually, and strictly, improved and eventually converge to the optimal value. The method is anytime, tensor-based, and offers provable guarantees. We conduct experiments on a set of challenging problems to validate our approach.

Acknowledgements. WR and MK are supported by the EPSRC Programme Grant on Mobile Autonomy (EP/M019918/1).

References

1. Carlini, N., Wagner, D.: Towards evaluating the robustness of neural networks. In: Security and Privacy (SP), 2017 IEEE Symposium on. pp. 39–57. IEEE (2017)
2. Dutta, S., Jha, S., Sanakaranarayanan, S., Tiwari, A.: Output range analysis for deep neural networks. arXiv preprint arXiv:1709.09130 (2017)
3. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572 (2014)
4. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
5. Huang, X., Kwiatkowska, M., Wang, S., Wu, M.: Safety verification of deep neural networks. In: CAV 2017. pp. 3–29 (2017)
6. Katz, G., Barrett, C., Dill, D., Julian, K., Kochenderfer, M.: Reluplex: An efficient SMT solver for verifying deep neural networks. In: CAV 2017 (2017)
7. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. pp. 1097–1105 (2012)
8. Kurakin, A., Goodfellow, I., Bengio, S.: Adversarial examples in the physical world. arXiv preprint arXiv:1607.02533 (2016)
9. Lomuscio, A., Maganti, L.: An approach to reachability analysis for feed-forward relu neural networks. CoRR **abs/1706.07351** (2017), `http://arxiv.org/abs/1706.07351`
10. Lundberg, S., Lee, S.: A unified approach to interpreting model predictions. CoRR **abs/1705.07874** (2017), `http://arxiv.org/abs/1705.07874`
11. Miyato, T., Maeda, S.i., Koyama, M., Nakae, K., Ishii, S.: Distributional smoothing with virtual adversarial training. arXiv preprint arXiv:1507.00677 (2015)
12. Moosavi-Dezfooli, S.M., Fawzi, A., Frossard, P.: Deepfool: a simple and accurate method to fool deep neural networks. In: CVPR 2016. pp. 2574–2582 (2016)
13. Olah, C., Satyanarayan, A., Johnson, I., Carter, S., Schubert, L., Ye, K., Mordvintsev, A.: The building blocks of interpretability. Distill (2018). https://doi.org/10.23915/distill.00010, https://distill.pub/2018/building-blocks
14. Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z.B., Swami, A.: The limitations of deep learning in adversarial settings. In: EuroS&P 2016. pp. 372–387. IEEE (2016)
15. Pec, J., Roels, J., Goossens, B., Saeys, Y.: Lower bounds on the robustness to adversarial perturbations. In: NIPS (2017)
16. Pei, K., Cao, Y., Yang, J., Jana, S.: DeepXplore: Automated whitebox testing of deep learning systems. In: SOSP 2017. pp. 1–18. ACM (2017)
17. Pulina, L., Tacchella, A.: An abstraction-refinement approach to verification of artificial neural networks. In: CAV 2010. pp. 243–257 (2010)

18. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
19. Su, J., Vargas, D.V., Sakurai, K.: One pixel attack for fooling deep neural networks. CoRR **abs/1710.08864** (2017), http://arxiv.org/abs/1710.08864
20. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks. In: In ICLR. Citeseer (2014)
21. Tian, Y., Pei, K., Jana, S., Ray, B.: DeepTest: Automated testing of deep-neural-network-driven autonomous cars. arXiv preprint arXiv:1708.08559 (2017)
22. Wicker, M., Huang, X., Kwiatkowska, M.: Feature-guided black-box safety testing of deep neural networks. In: TACAS 2018 (2018)

A Appendix: Proofs of Convergence

A.1 Guarantee of the Global Minimum by Grid Search

We prove the following theorem that shows that grid search with ϵ can guarantee to find the global optimum given a certain error bound assuming that the neural network satisfies the Lipschitz condition, as adopted in [15, 22].

Theorem 6 (Guarantee of the global minimum of grid search) *Assume a neural network $f(x) : [0,1]^n \rightarrow \mathbb{R}^m$ is Lipschitz continuous w.r.t. a norm metric $||*||_D$ and its Lipschitz constant is K. By recursively sampling $\Delta = 1/\epsilon$ in each dimension, denoted as $\mathcal{X} = \{x_1, ..., x_{\Delta^n}\}$, the following relation holds:*

$$||f_{opt}(x^*) - min_{x \in \mathcal{X}} f(x)||_D \leq K||\frac{\epsilon}{2}\mathbf{I}_n||_D$$

where $f_{opt}(x*)$ represents the global minimum value, $min_{x \in \mathcal{X}} f(x)$ denotes the minimum value returned by grid search, and $\mathbf{I}_n \in \mathbb{R}^{n \times n}$ is an all-ones matrix.

Proof 1 *Based on the Lipschitz continuity assumption of f, we have*

$$||f(x_1) - f(x_2)||_D \leq K||x_1 - x_2||_D$$

Based on the grid search with ϵ, we then have $\forall \tilde{x} \in [0,1]^n, \exists x \in \mathcal{X}$ such that $||x^* - x||_D \leq ||\frac{\epsilon}{2}\mathbf{I}_n||_D$, denoted as $\mathcal{X}(\tilde{x})$. Thus the theorem holds given the fact that we can always find $\mathcal{X}(x^*)$ from the sampled set $\mathcal{X}$ for the global minimum x^*.

A.2 Proof of Theorem: Guarantee of Lower Bounds

Proof 2 *Our proof proceeds by contradiction. Let $l = l(f, x_0)$. Assume that there is another adversarial example x_0' such that $t' = ||x_0' - x_0||_0 \leq l$ where t' represents the number of perturbed pixels. By the definition of adversarial examples, there exists a subspace $X_k \in \mathbb{R}^{t'}$ such that $cl(f, \mathcal{S}(x_0, t')[:, k]) \neq cl(f, x_0)$. By $t' \leq l$, we can find a subspace $Y_q \in \mathbb{R}^l$ such that $X_k \subset Y_q$. Thus we have $S(Y_q, l) \geq S(X_k, t')$. Moreover, by $S(Y_q, l) \leq S(Y_1, l)$, we have $cl(f, \mathcal{S}(x_0, l)[:, 1]) \neq cl(f, x_0)$ since $cl(f, \mathcal{S}(x_0, l)[:, p]) \neq cl(f, x_0)$. However, this conflicts with $cl(f, \mathcal{S}(x_0, l)[:, 1]) = cl(f, x_0)$, which can be obtained by the algorithm for computing lower bounds in Section 4.2.*

A.3 Proof of Theorem: Guarantee of Upper Bounds

Proof 3 (Monotonic Decrease Property of Upper Bounds) *We use mathematical induction to prove that upper bounds monotonically decrease.*

Base Case $m = 1$*: Based on the algorithm in* Upper Bounds *of Section 4.2, we assume that, after $m = 1$ subspace perturbations, we find the adversarial example x' such that $cl(f, x') \neq cl(f, x_0)$.*

We know that, at $t = i$, based on the algorithm, we get $S(x_0, i)$ and $\mathcal{S}(x_0, i)$, the ordered subspace sensitivities and their corresponding subspaces. Assume that the ordered subspace list is $\{\phi_1, \phi_2, ..., \phi_m\}$. Then, from the assumption, we have $cl(f, x(\phi_1)) \neq$

$cl(f, x_0)$ where $x(\phi_1)$ denotes the input of the neural network corresponding to subspace ϕ_1.

Then, at $t = i{+}1$, based on the algorithm, we calculate $S(x_0, i{+}1)$ and $\mathcal{S}(x_0, i{+}1)$. Similarly, we assume the ordered subspace list is $\{\theta_1, \theta_2, ..., \theta_n\}$. Thus we can find a subspace θ_q in $\{\theta_1, \theta_2, ..., \theta_n\}$ such that $\phi_1 \subset \theta_q$. As a result, we know that $S(\phi_1) \leq S(\theta_q)$, thus $cl(f, x(\theta_q)) \neq cl(f, x_0)$. After exhaustive tightening process, we can at least find its subset ϕ_1, since $cl(f, \phi_1) \neq cl(f, x_0)$ still holds after removing the pixels $x = \theta_q - \phi_1$. So we know $||x(\theta_q) - x_0||_0 \leq ||x(\phi_1) - x_0||_0$. However, θ_q will not necessarily be found at $t = i + 1$ in our upper bound algorithm, depending on its location in $\{\theta_1, \theta_2, ..., \theta_n\}$:

1. If it is in the front position such as $\{\theta_q, \theta_2, ..., \theta_m\}$, then we know that $||x(\theta_q) - x_0||_0 \leq ||x(\phi_1) - x_0||_0$ based on the above analysis, i.e., *$u_i(f, x_0) \geq u_{i+1}(f, x_0)$ holds.*

2. If it is in the behind position such as $\{\theta_1, \theta_2, \theta_q, ..., \theta_m\}$, we know that $S(\theta_1) \geq S(\theta_q) \geq S(\phi_1)$, which means that subspace θ_1 leads to a larger network's confidence decrease than subspace θ_q, thus $||x(\theta_1) - x_0||_0 \leq ||x(\theta_q) - x_0||_0$. We already know $||x(\theta_q) - x_0||_0 \leq ||x(\phi_1) - x_0||_0$ thus $u_i(f, x_0) \geq u_{i+1}(f, x_0)$ also holds.

***Inductive Case** $m = k$: Assume that at $t = i$, after going through $m = k$ subspace perturbations,* i.e., *$\Phi_k = \{\phi_1 \cup \phi_2 \cup ... \cup \phi_k\}$, we find an adversarial example x' and $u_i(f, x_0) \geq u_{i+1}(f, x_0)$ holds. We need to show that after going through $m = k + 1$ subspace perturbations,* i.e., *$\Phi_{k+1} = \{\phi_1 \cup \phi_2 \cup ... \cup \phi_k \cup \phi_{k+1}\}$, the relation $u_i(f, x_0) \geq u_{i+1}(f, x_0)$ also holds.*

Similarly, at $t = i + 1$, we can find a k subspace set $\Theta_k = \{\theta_{q_1} \cup \theta_{q_2} \cup ... \cup \theta_{q_k}\}$ such that $\Phi_k \subset \Theta_k$, and we know that $u_{i+1}(f, x_0) = ||x(\Phi_k) - x_0||_0 \leq ||x(\Theta_k) - x_0||_0 = u_i(f, x_0)$ holds. Then, for the new subspace ϕ_{k+1} at $t = i$, we can also find $\theta_{q_{k+1}}$ at $t = i + 1$ such that $\phi_{k+1} \subset \theta_{q_{k+1}}$. We get $\Theta_{k+1} = \Theta_k \cup \theta_{q_{k+1}}$. As a result, we still have $\Phi_{k+1} \subset \Theta_{k+1}$, obviously, $cl(x(\Theta_{k+1}), f) \neq cl(x(\Phi_{k+1}), f)$, which means we can definitely find an adversarial example after all perturbations in Θ_{k+1}. After exhaustive tightening process, we can at least find its subset Φ_{k+1}, since $cl(f, x(\Phi_{k+1})) \neq cl(f, x_0)$ still holds after removing those pixels $x = \Theta_k - \phi_k$. So we know $||x(\Theta_{k+1}) - x_0||_0 \leq ||x(\Phi_{k+1}) - x_0||_0$, i.e., *$u_i(f, x_0) \geq u_{i+1}(f, x_0)$ holds.*

A.4 Proof of Theorem: Uncertainty Radius Convergence to Zero

Proof 4 (Uncertainty Radius Convergence to Zero: $\lim_{t \to n} U_r(l_i, u_i) = 0$) *Based on the definition of L_0-norm distance (*i.e., *$0 \leq l_i \leq d_m \leq u_i \leq n$), we know that $t \to n \implies l_n \to n \implies U_r(l_i, u_i) = 1/2(u_i - l_i) = n - n = 0$.*

B Appendix: L_0 Norm

The distance metric $||\cdot||_D$ can be any functional mapping $||\cdot||_D : \mathbb{R}^n \times \mathbb{R}^n \to [0, \infty]$ that satisfies the metric conditions. In this paper, we focus on the L_0 metric. For two inputs x_0 and x, their L_0 distance, denoted as $||x - x_0||_0$, is the number of elements

that are different. When working with testing datasets, we define

$$\begin{aligned} ||\mathtt{T} - \mathtt{T}_0||_0 &= \mathtt{E}_{x_0 \in \mathtt{T}_0}[||x - x_0||_0] \qquad \text{(our definition)} \\ &= \tfrac{1}{|\mathtt{T}_0|} \textstyle\sum_{x_0 \in \mathtt{T}_0} ||x - x_0||_0 \text{ (all inputs in } \mathtt{T}_0 \text{ are i.i.d.)} \end{aligned} \tag{20}$$

where x is a homogeneous input of x_0 in $\mathtt{T}$. While other norms such as L_1, L_2 and L_∞ have been widely applied, see e.g., [8, 14], in generating adversarial examples, the studies based on the L_0 norm are few and far between. We justify on the basis of several aspects that the L_0 norm is worthy of being considered.

Technical Reason The reason why norms other than the L_0 norm are widely used is *mainly technical*: the existing adversarial example generation algorithms [1, 14] proceed by first computing the gradient $\nabla_x J(\theta, x, f(x))$, where $J(\theta, x, f(x))$ is a loss or cost function and θ are the learnable parameters of the network f, and then adapting (in different ways for different algorithms) the input x into x' along the gradient descent direction. To enable this computation, the change to the input x needs to be *continuous and differentiable*. It is not hard to see that, while the L_1, L_2 and L_∞ norms are continuous and differentiable, the L_0 norm is not. Nevertheless, the L_0 norm is an effective and efficient method to quantify a range of adversarial perturbations and should be studied.

Tolerance of Human Perception to L_0 Norm Recently, [14] demonstrates through in-situ human experiments that the L_0 norm is good at approximating human visual perception. Specifically, 349 human participants were recruited for a study of visual perception on L_0 image distortions, with the result concluding that nearly all participants can correctly recognise L_0 perturbed images when the rate of distortion pixels is less than 5.61% (i.e., 44 pixels for MNIST and 57 pixels for CIFAR-10) and 90% of them can still recognise them when the distortion rate is less than 14.29% (i.e., 112 pixels for MNIST and 146 pixels for CIFAR-10). This experiment essentially demonstrates that human perception is tolerant of perturbations with only a few pixels changed, and shows the necessity of robustness evaluation based on the L_0 norm.

Usefulness of Approaches without Gradient From the security point of view, an attacker to a network may not be able to access its architecture and parameters, to say nothing of the gradient $\nabla_x J(\theta, x, f(x))$. Therefore, to evaluate the robustness of a network, we need to consider *black-box* attackers, which can only query the network for classification. For a black-box attacker, an attack (or a perturbation) based on the L_0 norm is to change several pixels, which is arguably easier to initiate than attacks based on other norms, which often require modifications to nearly all the pixels.

Effectiveness of Pixel-based Perturbations Perturbations by minimising the number of pixels to be changed have been shown to be effective. For example, [19, 22] show that manipulating a single pixel is sufficient for the classification to be changed for several networks trained on the CIFAR10 dataset and the Nexar traffic light challenge. Our approach can beat the state-of-the-art pixel based perturbation algorithms by finding tighter upper bounds to the maximum safety radius. As far as we know, this is the first work on finding lower bounds to the maximum safety radius.

C Appendix: Discussion of Application Scenarios

We now summarise possible application scenarios for the method proposed in this paper.

Safety Verification Safety verification [5] is to determine, for a given network f, an input x_0, a distance metric $||\cdot||_D$, and a number d, whether the norm ball $X(f, x_0, ||\cdot||_D, d)$ is safe. Our approach will compute a sequence of lower bounds $\mathcal{L}(x_0)$ and upper bounds $\mathcal{U}(x_0)$ for the maximum safe radius $d_m(x_0)$. For every round $i > 0$, we can claim one of the following cases:

- the norm ball $X(f, x_0, ||\cdot||_D, d)$ is safe when $d \leq \mathcal{L}(x_0)_i$
- the norm ball $X(f, x_0, ||\cdot||_D, d)$ is unsafe when $d \geq \mathcal{U}(x_0)_i$
- the safety of $X(f, x_0, ||\cdot||_D, d)$ is unknown when $\mathcal{L}(x_0)_i < d < \mathcal{U}(x_0)_i$.

As a byproduct, our method can return at least one adversarial image for the second case.

Competitive L_0 Attack We have shown that the upper bounds in $\mathcal{U}(x_0)$ are monotonically decreasing. As a result, the generation of upper bounds can serve as a competitive L_0 attack method.

Global Robustness Evaluation Our method can have an asymptotic convergence to the true global robustness with provable guarantees. As a result, for two neural networks f_1 and f_2 that are trained for the same task (*e.g.,* MNIST, CIFAR-10 or ImageNet) but with different parameters or architectures (*e.g.,* different layer types, layer numbers or hidden neuron numbers), if $R(f_1, ||\cdot||_0) > R(f_2, ||\cdot||_0)$ then we can claim that network f_1 is more robust than f_2 in terms of its resistance to L_0-norm adversarial attacks.

Test Case Generation Recently software coverage testing techniques have been applied to DNNs and several test criteria have been proposed, see e.g., [16, 21]. Each criterion defines a set of requirements that have to be tested for a DNN. Given a test suite, the coverage level of the set of requirements indicates the adequacy level for testing the DNN. The technique in this paper can be conveniently used for coverage-based testing of DNNs.

Real-time Robustness Evaluation By replacing the exhaustive search in the algorithm with random sampling and formulating the subspace as a high-dimensional tensor (to enable parallel computation with GPUs), our method becomes *real-time* (*e.g.,* for a MNIST network, it takes around $0.1s$ to generate an adversarial example). A real-time evaluation can be useful for major safety-critical applications, including self-driving cars, robotic navigation, *etc.*. Moreover, our method can display in real-time a *saliency map*, visualizing how classification decisions of the network are influenced by pixel-level sensitivities.

Table 1: sDNN

Layer Type	Size
Input layer	$14 \times 14 \times 1$
Convolution layer	$2 \times 2 \times 8$
Batch-Normalization layer	8 channels
ReLU activation	
Convolution layer	$2 \times 2 \times 16$
Batch-Normalization layer	16 channels
ReLU activation	
Convolution layer	$2 \times 2 \times 32$
Batch-Normalization layer	32 channels
ReLU activation	
Fully Connected	10
softmax + Class output	

D Appendix: Experimental Settings for Case Study One

D.1 Model Structures of sDNN

D.2 Parameter Settings of sDNN

Model Training Setup

- Hardware: Notebook PC with I7-7700HQ, 16GB RAM, GTX 1050 GPU
- Software: Matlab 2018a, Neural Network Toolbox, Image Processing Toolbox, Parallel Computing Toolbox
- Parameter Optimization Settings: SGDM, Max Epochs = 20, Mini-Batch Size = 128
- Training Dataset: MNIST training dataset with 50,000 images
- Training Accuracy: 99.5%
- Testing Dataset: MNIST testing dataset with 10,000 images
- Testing Accuracy: 98.73%

Algorithm Setup

- $\epsilon = 0.25$
- Maximum $t = 3$
- Tested Images: 5,300 images sampled from MNIST testing dataset

D.3 Model Structures of DNN-0

D.4 Parameter Settings of DNN-0

Model Training Setup

- Hardware: Notebook PC with I7-7700HQ, 16GB RAM, GTX 1050 GPU

Table 2: DNN-0

Layer Type	Size
Input layer	$28 \times 28 \times 1$
Convolution layer	$3 \times 3 \times 32$
ReLU activation	
Convolution layer	$3 \times 3 \times 64$
ReLU activation	
Maxpooling layer	2×2
Dropout layer	0.25
Fully Connected layer	128
ReLU activation	
Dropout layer	0.5
Fully Connected layer	10
Softmax + Class output	

- Software: Matlab 2018a, Neural Network Toolbox, Image Processing Toolbox, Parallel Computing Toolbox
- Parameter Optimization Settings: SGDM, Max Epochs = 30, Mini-Batch Size = 128
- Training Dataset: MNIST training dataset with 50,000 images
- Training Accuracy: 100%
- Testing Dataset: MNIST testing dataset with 10,000 images
- Testing Accuracy: 99.16%

Algorithm Setup

- $\epsilon = 0.25$
- Maximum $t = 2$
- Tested Images: 2,400 images sampled from MNIST testing dataset

D.5 Ground-Truth Adversarial Images

Fig. 11 displays some adversarial images returned by our upper bound algorithm. For each digital image, from the left to right, the first is original image, the second is the adversarial image returned at $t = 1$, and the third is the adversarial example at the boundary of a safe norm ball.

E Appendix: Experimental Settings for Case Study Two

E.1 Model Structures of DNN-1 to DNN-7

The model structures of DNN-1 to DNN-7 are described in respective tables.

Fig. 11: Ground truth adversarial examples when converging to d_m.

Model Training Setup

- Hardware: Notebook PC with I7-7700HQ, 16GB RAM, GTX 1050 GPU
- Software: Matlab 2018a, Neural Network Toolbox, Image Processing Toolbox, Parallel Computing Toolbox
- Parameter Optimization Settings: SGDM, Max Epochs = 30, Mini-Batch Size = 128
- Training Dataset: MNIST training dataset with 50,000 images
- Training Accuracy: All 7 models reach 100%
- Testing Dataset: MNIST testing dataset with 10,000 images
- Testing Accuracy: DNN-1 = 97.75%; DNN-2 = 97.95%; DNN-3 = 98.38%; DNN-4 = 99.06%; DNN-5 = 99.16%; DNN-6 = 99.13%; DNN-7 = 99.41%

Algorithm Setup

- $\epsilon = 0.3$
- Maximum $t = 2$
- Tested Images: 1,000 images sampled from MNIST testing dataset

Table 3: DNN-1

Layer Type	Size
Input layer	$28 \times 28 \times 1$
Convolution layer	$3 \times 3 \times 32$
ReLU activation	
Fully Connected layer	10
Softmax + Class output	

Table 4: DNN-2

Layer Type	Size
Input layer	$28 \times 28 \times 1$
Convolution layer	$3 \times 3 \times 32$
ReLU activation	
Convolution layer	$3 \times 3 \times 64$
ReLU activation	
Fully Connected layer	10
Softmax + Class output	

Table 5: DNN-3

Layer Type	Size
Input layer	$28 \times 28 \times 1$
Convolution layer	$3 \times 3 \times 32$
ReLU activation	
Convolution layer	$3 \times 3 \times 64$
Batch-Normalization layer	
ReLU activation	
Fully Connected layer	10
Softmax + Class output	

Table 6: DNN-4

Layer Type	Size
Input layer	$28 \times 28 \times 1$
Convolution layer	$3 \times 3 \times 32$
ReLU activation	
Convolution layer	$3 \times 3 \times 64$
Batch-Normalization layer	
ReLU activation	
Fully Connected layer	128
ReLU activation	
Fully Connected layer	10
Softmax + Class output	

F Appendix: Experimental Settings for Case Study Three

F.1 Model Structures for L_0 Attack

The architectures for the MNIST and CIFAR-10 models used in L_0 attack are illustrated in Table 10.

F.2 Model Training Setups

- Parameter Optimization Option: Batch Size = 128, Epochs = 50, Loss Function = tf.nn.softmax_cross_entropy_with_logits, Optimizer = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
- Training Accuracy:
  - MNIST (99.99% on 60,000 images)
  - CIFAR-10 (99.83% on 50,000 images)
- Testing Accuracy:
  - MNIST (99.36% on 10,000 images)
  - CIFAR-10 (78.30% on 10,000 images)

Table 7: DNN-5

Layer Type	Size
Input layer	$28 \times 28 \times 1$
Convolution layer	$3 \times 3 \times 32$
ReLU activation	
Convolution layer	$3 \times 3 \times 64$
Batch-Normalization layer	
ReLU activation	
Dropout layer	0.5
Fully Connected layer	128
ReLU activation	
Fully Connected layer	10
Softmax + Class output	

Table 8: DNN-6

Layer Type	Size
Input layer	$28 \times 28 \times 1$
Convolution layer	$3 \times 3 \times 32$
ReLU activation	
Convolution layer	$3 \times 3 \times 64$
ReLU activation	
Convolution layer	$3 \times 3 \times 128$
Batch-Normalization layer	
ReLU activation	
Dropout layer	0.5
Fully Connected layer	128
ReLU activation	
Fully Connected layer	10
Softmax + Class output	

Table 9: DNN-7

Layer Type	Size
Input layer	$28 \times 28 \times 1$
Convolution layer	$3 \times 3 \times 16$
ReLU activation	
Convolution layer	$3 \times 3 \times 32$
Batch-Normalization layer	
ReLU activation	
Convolution layer	$3 \times 3 \times 64$
Batch-Normalization layer	
ReLU activation	
Convolution layer	$3 \times 3 \times 128$
Batch-Normalization layer	
ReLU activation	
Dropout layer	0.5
Fully Connected layer	256
ReLU activation	
Dropout layer	0.5
Fully Connected layer	10
Softmax + Class output	

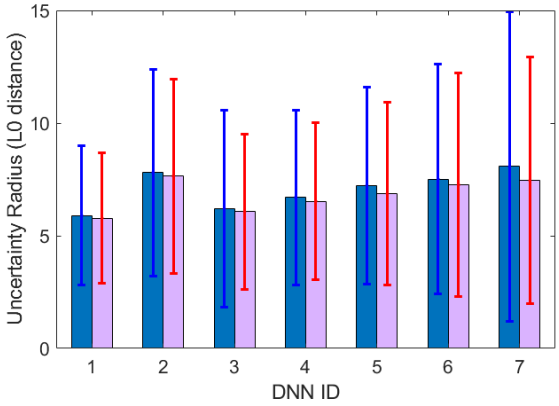


Fig. 12: The means and standard derivations of d_m uncertainty radiuses for 1,000 tested images at $t = 1, 2$.

Table 10: Model architectures for the MNIST and CIFAR-10 models.

Layer Type	MNIST	CIFAR-10
Convolution + ReLU	$3 \times 3 \times 32$	$3 \times 3 \times 64$
Convolution + ReLU	$3 \times 3 \times 32$	$3 \times 3 \times 64$
Max Pooling	2×2	2×2
Convolution + ReLU	$3 \times 3 \times 64$	$3 \times 3 \times 128$
Convolution + ReLU	$3 \times 3 \times 64$	$3 \times 3 \times 128$
Max Pooling	2×2	2×2
Flatten		
Fully Connected + ReLU	200	256
Dropout	0.5	0.5
Fully Connected + ReLU	200	256
Fully Connected	10	10

F.3 Experimental Setting for Competitive L0 Attack Comparison

Baseline Methods We choose four well-established baseline methods that can perform state-of-the-art L_0 adversarial attacks. Their code is available on Github.

- JSMA[5]: is a targeted attack based on the L_0-norm, here used so that adversarial examples are misclassified into all classes except the correct one.
- C&W[6]: is a state-of-the-art adversarial attack method, which models the attack problem as an unconstrained optimization problem that is solvable by the Adam optimizer in Tensorflow.

[5] https://github.com/tensorflow/cleverhans/blob/master/cleverhans_tutorials/mnist_tutorial_jsma.py

[6] https://github.com/carlini/nn_robust_attacks

– DLV[7]: in an untargeted DNN verification method based on exhaustive search and Monte Carlo tree search (MCTS).
– SafeCV[8]: is a feature-guided black-box safety verification and attack method based on the Scale Invariant Feature Transform (SIFT) for feature extraction, game theory, and MCTS.

Dataset We perform comparison on two datasets - MNIST and CIFAR-10. They are standard benchmark datasets for adversarial attack of DNNs, and are widely adopted by all these baseline methods.

– MNIST dataset[9]: is an image dataset of handwritten digits, which contains a training set of 60,000 examples and a test set of 10,000 examples. The digits have been size-normalized and centered in a fixed-size image.
– CIFAR-10 dataset[10]: is an image dataset of 10 mutually exclusive classes, i.e., 'airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck'. It consists of 60,000 32x32 colour images - 50,000 for training, and 10,000 for testing.

Platforms

– Hardware Platform:
  • NVIDIA GeForce GTX TITAN Black
  • Intel(R) Core(TM) i5-4690S CPU @ 3.20GHz $\times$ 4
– Software Platform:
  • Ubuntu 14.04.3 LTS
  • Fedora 26 (64-bit)
  • Anaconda, PyCharm

F.4 Algorithm Settings

MNIST and CIFAR-10 use the same settings, unless separately specified.

– JSMA:
  • bounds = (0, 1)
  • predicts = 'logits'
– C&W:
  • targeted = False
  • learning_rate = 0.1
  • max_iteration = 100
– DLV:
  • mcts_mode = "sift_twoPlayer"

[7] https://github.com/VeriDeep/DLV
[8] https://github.com/matthewwicker/SafeCV
[9] http://yann.lecun.com/exdb/mnist/
[10] https://www.cs.toronto.edu/~kriz/cifar.html

- startLayer, maxLayer = -1
- numOfFeatures = 150
- featureDims = 1
- MCTS_level_maximal_time = 30
- MCTS_all_maximal_time = 120
- MCTS_multi_samples = 5 (MNIST), 3 (CIFAR-10)

– SafeCV:
- MANIP = max_manip (MNIST), white_manipulation (CIFAR-10)
- VISIT_CONSTANT = 1
- backtracking_constant = 1
- simulation_cutoff = 100

– Ours:
- EPSILON = 0.5
- L0_UPPER_BOUND = 100

F.5 Adversarial Images

Fig. 13 and 14 present a few adversarial examples generated on the MNIST and CIFAR-10 datasets by our approach L0-TRE, together with results for four other tools, *i.e.,* C&W, JSMA, DLV, and SafeCV.

G Appendix: Experimental Settings for Case Study Four

G.1 State-of-the-art ImageNet DNN Models

– AlexNet [7] : a convolutional neural network, which was originally designed in the ImageNet Large Scale Visual Recognition Challenge in 2012. The network achieved a top-5 error of 15.3%, more than 10.8 percentage points ahead of the runner up.
– VGG-16 and VGG-19 [18] : were released in 2014 by the Visual Geometry Group at the University of Oxford. This family of architectures achieved second place in the 2014 ImageNet Classification competition, achieving 92.6% top-five accuracy on the ImageNet 2012 competition dataset.
– ResNet50 and ResNet101 [4] : are designed based on residual nets with a depth of 50 and 101 layers. An ensemble of these networks achieved 3.57% testing error on ImageNet and won the 1st place on the ILSVRC 2015 classification task. Their variants also won 1st places on the tasks of ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation.

G.2 Experimental Settings

Platforms

– Hardware: Notebook PC with I7-7700HQ, 16GB RAM, GTX 1050 GPU
– Software: Matlab 2018a, Neural Network Toolbox, Image Processing Toolbox, Parallel Computing Toolbox, and AlexNet, VGG-16, VGG-19, ResNet50 and ResNet101 Pretrained DNN Models

Algorithm Setup

- $\epsilon = 0.3$
- Maximum $t = 1$
- Tested Images: 20 ImageNet images, randomly chosen

G.3 Adversarial Images and Saliency Maps

Fig. 16 gives more examples of adversarial images and saliency maps.

H Appendix: Experimental Settings for Case Study Five

The experimental settings of this case study can be found in Appendix F. Fig. 19 depicts the percentage of adversarial examples found, within corresponding distances, for the two cases. A significant portion of adversarial examples can be found in the relatively small L_0 distance end of the curve.

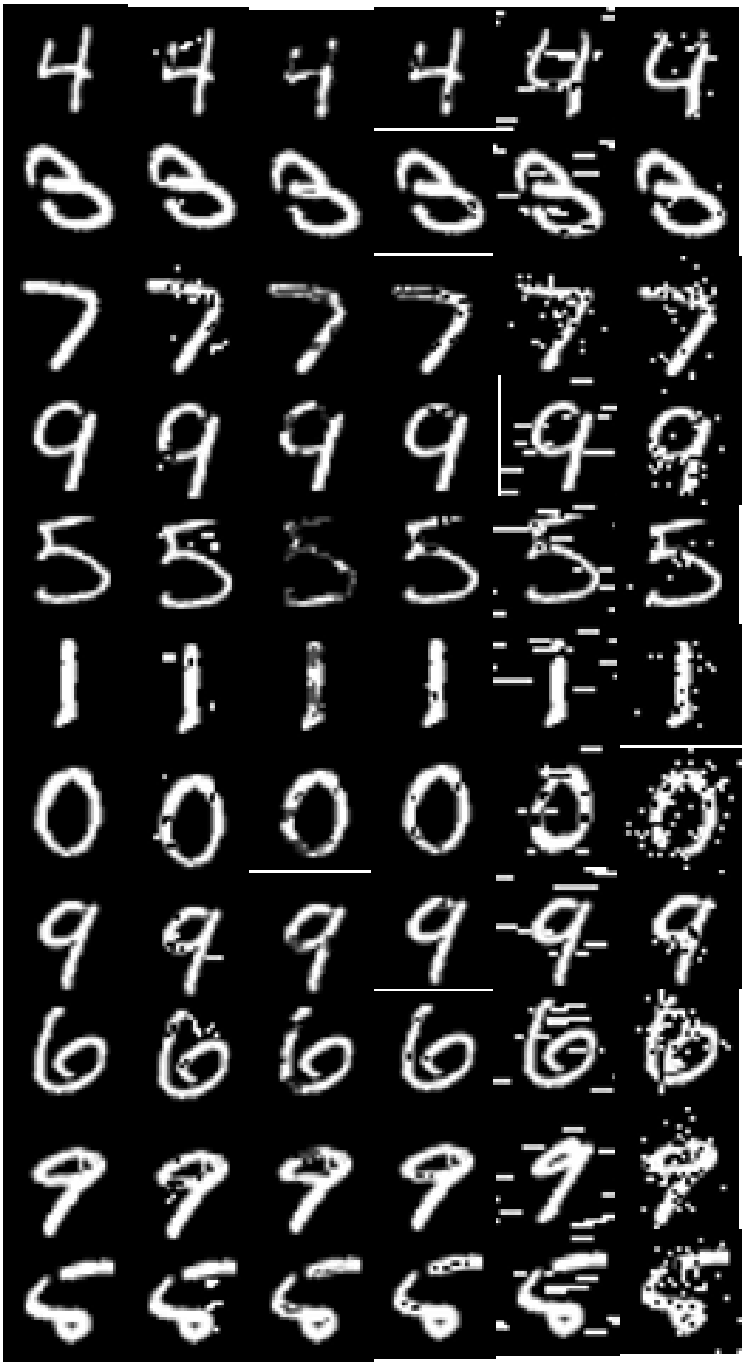


Fig. 13: Adversarial images generated by the L_0 attack methods on the MNIST dataset. From left to right: original image, our approach, C&W, JSMA, DLV, and SafeCV.

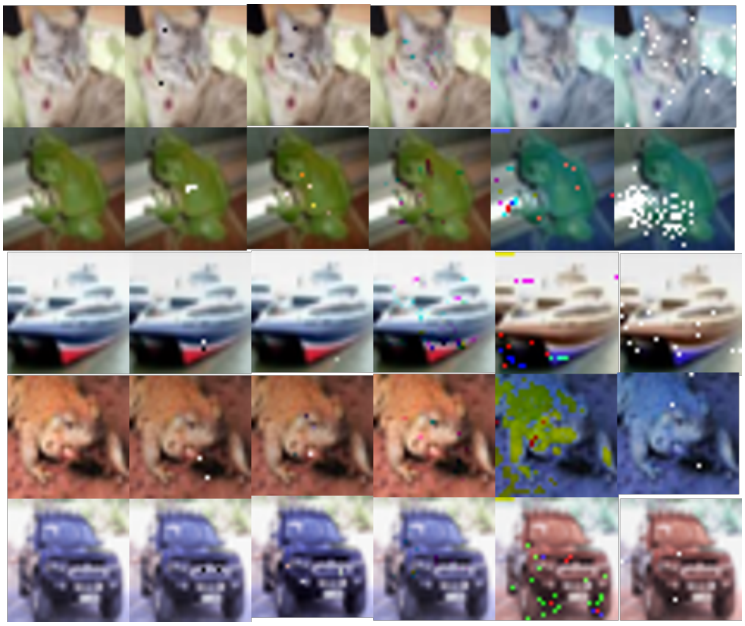


Fig. 14: Adversarial images generated by the L_0 attack methods on the CIFAR-10 dataset. From left to right: original image, our approach, C&W, JSMA, DLV, and SafeCV.

Fig. 15: Ground-truth adversarial examples (right column) generated by L0-TRE at $t = 1$, and the saliency maps of the original images (left column).

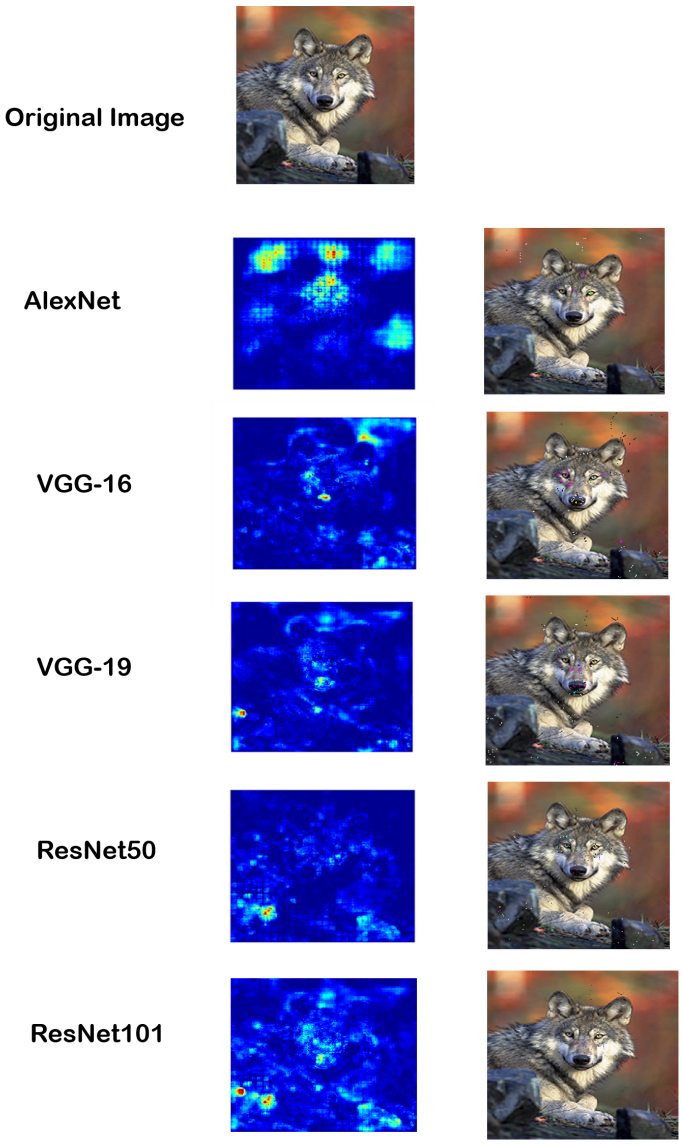


Fig. 16: Adversarial examples on upper boundaries returned by our tool L0-TRE (right column), and saliency maps for each ImageNet DNN model (left column).

Fig. 17: Adversarial examples on upper boundaries (right column) and their saliency maps (left column) for ImageNet AlexNet DNNs. Note that all adversarial images with $L_0 = 1, 2$ are also the ground-truth L_0-norm adversarial images since their upper bounds and lower bounds local robustness have converged.

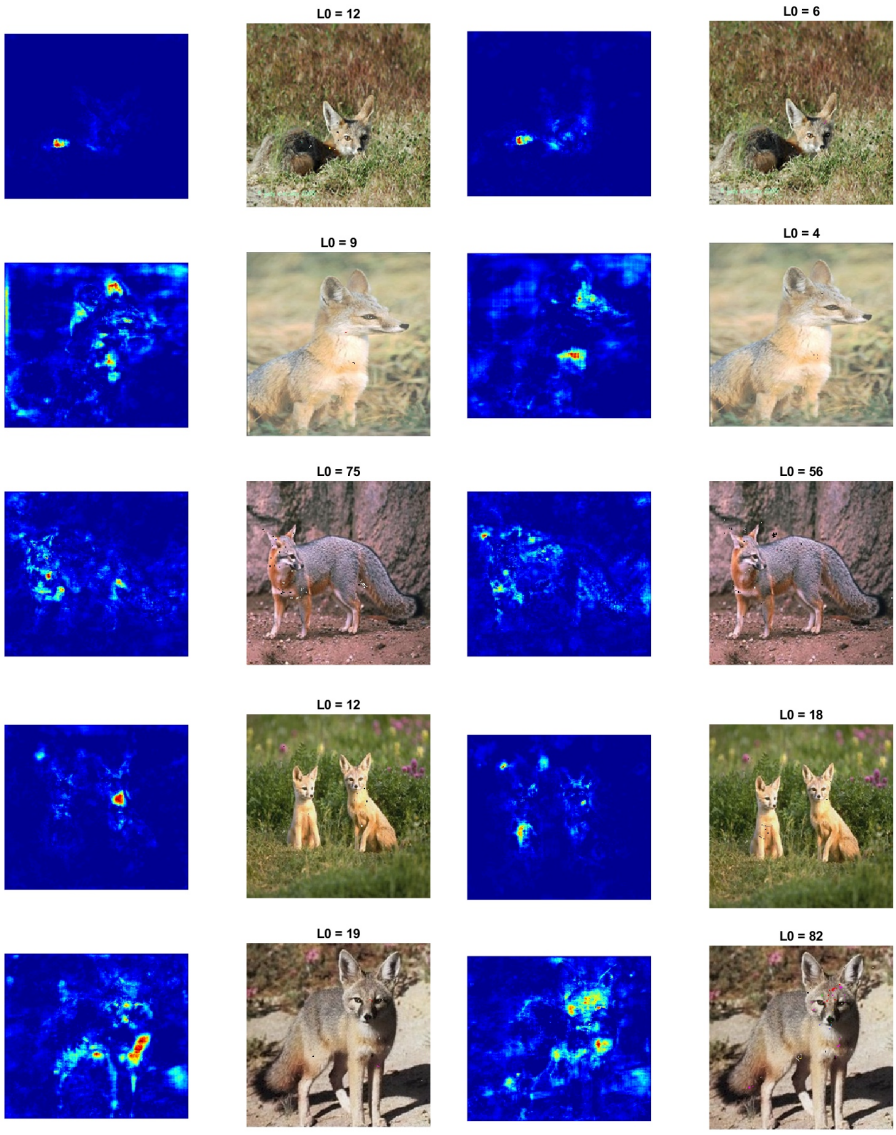


Fig. 18: Adversarial examples on upper boundaries (right column) returned by L0-TRE, and their saliency maps (left column) for VGG-16 and VGG-19. The first and second columns are for VGG-16; the third and fourth columns are for VGG-19.

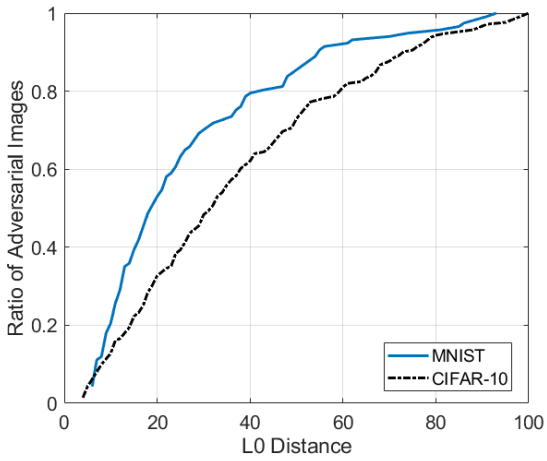


Fig. 19: Neuron coverage: the percentage of adversarial examples within each distance.