DeepSearch: Simple and Effective Blackbox Fuzzing of Deep Neural Networks

Fuyuan Zhang
MPI-SWS, Germany
fuyuan@mpi-sws.org

Sankalan Pal Chowdhury
MPI-SWS, Germany
sankalan@mpi-sws.org

Maria Christakis
MPI-SWS, Germany
maria@mpi-sws.org

ABSTRACT

Although deep neural networks have been successful in image classification, they are prone to adversarial attacks. To generate misclassified inputs, there has emerged a wide variety of techniques, such as black- and whitebox testing of neural networks. In this paper, we present DeepSearch, a novel blackbox-fuzzing technique for image classifiers. Despite its simplicity, DeepSearch is shown to be more effective in finding adversarial examples than closely related black- and whitebox approaches. DeepSearch is additionally able to generate the most subtle adversarial examples in comparison to these approaches.

1 INTRODUCTION

Deep neural networks have been impressively successful in pattern recognition and image classification [30, 38, 41]. However, the more popular deep neural networks become for performing such tasks, the more incentives adversaries have for trying to trick them into making a wrong prediction, often with critical consequences. In fact, even very subtle perturbations of a correctly classified image, perhaps imperceptible to the human eye, may cause a deep neural network to change its prediction [68].

More formally, let $\mathbb{R}^n$ be the n-dimensional vector space for input images. We represent images as column vectors $\mathbf{x} = (x_1, ..., x_n)^T$, where $x_i \in \mathbb{R}$ $(1 \leq i \leq n)$ is the ith coordinate of $\mathbf{x}$. We also write $\mathbf{x}(i)$ to denote the ith coordinate x_i, i.e., $\mathbf{x}(i) = x_i$, and each such coordinate represents an image pixel. Now, let $C_m = \{l_1, ..., l_m\}$ be a set of labels for m classes, where l_i is the label for the ith class $(1 \leq i \leq m)$. A deep neural network that classifies images from $\mathbb{R}^n$ into m classes in C_m is essentially a function $\mathcal{N} : \mathbb{R}^n \to C_m$. For an input $\mathbf{x} \in \mathbb{R}^n$, $\mathcal{N}(\mathbf{x})$ is the label that the network assigns to $\mathbf{x}$.

Assume that input $\mathbf{x}$ is correctly classified, and $\mathbf{x}'$ is generated by applying subtle perturbations to $\mathbf{x}$. These perturbations are subtle when the distance between $\mathbf{x}$ and $\mathbf{x}'$ in $\mathbb{R}^n$ is sufficiently small according to a distance metric. When this is so and $\mathcal{N}(\mathbf{x}) \neq \mathcal{N}(\mathbf{x}')$, we say that $\mathbf{x}'$ is an *adversarial example* [68]. In other words, the network is tricked into classifying $\mathbf{x}'$ into a different class than $\mathbf{x}$ even though they are very similar.

In this paper, we use the L_∞ distance metric. The L_∞ distance between $\mathbf{x}$ and $\mathbf{x}'$ is defined as the maximum of their differences along any coordinate dimension i $(1 \leq i \leq n)$:

$$||\mathbf{x} - \mathbf{x}'||_{L_\infty} = \max_i(|x_i - x_i'|)$$

For $d \in \mathbb{R}$, we write $\mathcal{B}(\mathbf{x}, d)$ to denote the set of images within distance d from $\mathbf{x}$, i.e., $\mathcal{B}(\mathbf{x}, d) = \{\mathbf{x}' \mid ||\mathbf{x} - \mathbf{x}'||_{L_\infty} \leq d\}$, which is an n-dimensional cube.

Based on the above, a deep neural network $\mathcal{N}$ is *locally robust* for a correctly classified input $\mathbf{x}$ with respect to distance d if it assigns the same label to all images in $\mathcal{B}(\mathbf{x}, d)$. There have recently emerged numerous approaches for evaluating local robustness of a deep neural network. *Whitebox* techniques have full access to the network under evaluation (e.g., [12, 24, 39, 56]). On the other hand, *blackbox* techniques may only query the network for its predictions on particular inputs (e.g., [5, 6, 51]). In other words, such techniques may only observe the input-output behavior of the network and assume no knowledge about the network structure or its parameters.

Recently, Wicker et al. [74] proposed a blackbox technique for testing image classifiers. They first extract features from images and then encode the problem of finding adversarial examples as a two-player turn-based stochastic game. Although competitive with state-of-the-art whitebox techniques, this approach is very complicated. Inspired by the linear explanation of adversarial examples [24], we develop a simple, yet very effective, blackbox-fuzzing technique for deep neural networks.

Our approach. We present DeepSearch, a blackbox-fuzzing technique for testing the robustness of deep neural networks for image classification. DeepSearch is a novel blackbox approach that does not apply gradient-based methods. Starting from a correctly classified input image $\mathbf{x}$, DeepSearch strategically mutates its pixels to values that are more likely to lead to an adversarial example $\mathbf{x}'$. Once such an example is found, DeepSearch is able to significantly reduce its distance from $\mathbf{x}$, thereby finding subtle adversarial examples generated by only slightly perturbing pixels in $\mathbf{x}$.

Not only does DeepSearch use a fundamentally novel technique for testing neural networks, but it is also shown to be more effective in generating adversarial examples than state-of-the-art black- and whitebox approaches. Moreover, in comparison to these related approaches, DeepSearch is able to find the most subtle adversarial examples on two popular image datasets, namely MNIST [42] and CIFAR-10 [37].

Contributions. We make the following contributions:

(1) We present a simple, yet very effective, blackbox-fuzzing technique for deep neural networks.
(2) We perform an extensive evaluation showing that DeepSearch is more effective in finding adversarial examples than state-of-the-art black- and whitebox approaches.
(3) We show that a straightforward extension to our technique, which we call refinement, gives DeepSearch the advantage of finding the most subtle adversarial examples in comparison to related approaches.

Outline. The following section presents DeepSearch for finding adversarial examples in binary classifiers, that is, networks that classify inputs into two classes. Sect. 3 generalizes the technique to multiclass classifiers, which classify inputs into multiple classes. In

Sect. 4, we extend our technique further such that the generated adversarial examples are more subtle. We present our experimental evaluation in Sect. 5, discuss related work in Sect. 6, and conclude in Sect. 7.

2 FUZZING BINARY CLASSIFIERS

In this section, we present the technical details of how DeepSearch fuzzes (linear and non-linear) binary classifiers.

2.1 Linear Binary Classifiers

In the following, we review background on linear binary classifiers, give an intuition on how our technique fuzzes their inputs through an example, and present our methodology in detail.

Background. A *binary classifier* classifies inputs into two classes, denoted with labels $C_2 = \{l_1, l_2\}$, according to the definition below.

Definition 1 (Binary Classifier). Given a classification function $f : \mathbb{R}^n \to \mathbb{R}$, a *binary classifier* $\mathcal{N}_f : \mathbb{R}^n \to C_2$ is defined as follows:

$$\mathcal{N}_f(\mathbf{x}) = \begin{cases} l_1, & \text{if } f(\mathbf{x}) > 0 \\ l_2, & \text{if } f(\mathbf{x}) < 0 \end{cases}$$

If function f is linear, then $\mathcal{N}_f$ is a *linear* binary classifier, otherwise it is non-linear.

The set of values $\mathcal{D}_f = \{\mathbf{x} \mid f(\mathbf{x}) = 0\}$ constitute the *decision boundary* of $\mathcal{N}_f$, which classifies the domain $\mathbb{R}^n$ into the two classes in C_2. As an example, consider Fig. 1a, showing a linear binary classifier $\mathcal{N}_f : \mathbb{R}^2 \to C_2$. (The red dashed and dash-dotted lines should be ignored for now.) Observe that input $\mathbf{x}_0$ is classified in l_1 whereas $\mathbf{x}'_0$ is in l_2. Note that the decision boundary of a linear classifier $\mathcal{N}_f : \mathbb{R}^n \to C_2$ is a hyperplane; it is, therefore, a straight line in Fig. 1a.

Given a binary classifier $\mathcal{N}_f$ and a correctly classified input $\mathbf{x}$, there exists an adversarial example in the set of inputs within distance $d \in \mathbb{R}$ of $\mathbf{x}$, denoted $\mathcal{B}(\mathbf{x}, d)$, if and only if another input $\mathbf{x}' \in \mathcal{B}(\mathbf{x}, d)$ is classified differently than $\mathbf{x}$. In other words, $\mathbf{x}'$ is an adversarial example if $\mathcal{N}_f(\mathbf{x}) \neq \mathcal{N}_f(\mathbf{x}')$, which is equivalent to $f(\mathbf{x})f(\mathbf{x}') < 0$. For example, assume that input $\mathbf{x}_0 = (x_h, x_v)$ in Fig. 1a is correctly classified and that the dash-dotted square represents $\mathcal{B}(\mathbf{x}, d)$. Then, $\mathbf{x}'_0$ is an adversarial example.

Example. We give an intuition on how DeepSearch handles linear binary classifiers using the example of Fig. 1a. Recall that $\mathbf{x}_0$ is a correctly classified input for which $f(\mathbf{x}_0) > 0$. To find an adversarial example, DeepSearch fuzzes $\mathbf{x}_0$ with the goal of generating a new input $\mathbf{x}'_0$ such that $f(\mathbf{x}'_0) < 0$.

Fuzzing is performed as follows. Input $\mathbf{x}_0$ has two coordinates x_h and x_v, for the horizontal and vertical dimensions. DeepSearch independently mutates each of these coordinates to the minimum and maximum values that are possible within $\mathcal{B}(\mathbf{x}_0, d)$, with the intention to find the minimum value of f on $\mathcal{B}(\mathbf{x}_0, d)$. For instance, when mutating x_h, we obtain inputs $\mathbf{x}_0[l_h/x_h]$ and $\mathbf{x}_0[u_h/x_h]$ in the figure. Values l_h and u_h are, respectively, the minimum and maximum that x_h may take, and $\mathbf{x}_0[l_h/x_h]$ denotes substituting x_h with l_h (similarly for $\mathbf{x}_0[u_h/x_h]$). We then evaluate $f(\mathbf{x}_0[l_h/x_h])$ and $f(\mathbf{x}_0[u_h/x_h])$, and for x_h, we select the value (l_h or u_h) that causes function f to *decrease*. This is because, in our example, an adversarial input $\mathbf{x}'_0$ must make the value of f negative. Let us assume that $f(\mathbf{x}_0[u_h/x_h]) < f(\mathbf{x}_0[l_h/x_h])$; we, thus, select u_h for coordinate x_h.

DeepSearch mutates coordinate x_v in a similar way. It evaluates $f(\mathbf{x}_0[l_v/x_v])$ and $f(\mathbf{x}_0[u_v/x_v])$, and selects the value that causes f to decrease. Let us assume that $f(\mathbf{x}_0[u_v/x_v]) < f(\mathbf{x}_0[l_v/x_v])$; we, thus, select u_v for x_v.

Next, we generate input $\mathbf{x}'_0$ by substituting each coordinate in $\mathbf{x}_0$ with the boundary value that was previously selected. In other words, $\mathbf{x}'_0 = \mathbf{x}_0[u_h/x_h, u_v/x_v]$, and since $f(\mathbf{x}'_0) < 0$, DeepSearch has generated an adversarial example. Note that $f(\mathbf{x}'_0)$ is actually the minimum value of f on $\mathcal{B}(\mathbf{x}_0, d)$.

DeepSearch for linear binary classifiers. We now formalize how DeepSearch treats linear binary classifiers. Consider a linear classification function $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b = \sum_{i=1}^{n} w_i x_i + b$, where $\mathbf{w}^T = (w_1, ..., w_n)$ and $b \in \mathbb{R}$. Note that f is monotonic with respect to all of its variables $x_1, ..., x_n$. For instance, if $w_i > 0$, then f is monotonically increasing in x_i.

Recall that $\mathcal{B}(\mathbf{x}, d)$ denotes the set of inputs within distance $d \in \mathbb{R}$ of an input $\mathbf{x}$. $\mathcal{B}(\mathbf{x}, d)$ may be represented by an n-dimensional cube $\mathcal{I} = I_1 \times ... \times I_n$, where $I_i = [l_i, u_i]$ is a closed interval bounded by $l_i, u_i \in \mathbb{R}$ with $l_i \leq u_i$ for $1 \leq i \leq n$. Intuitively, value l_i (resp. u_i) is the lower (resp. upper) bound on the ith dimension of $\mathbf{x}$. An input $\mathbf{x}'$ is a *vertex* of $\mathcal{I}$ if each of its coordinates $\mathbf{x}'(i)$ is an endpoint of I_i for $1 \leq i \leq n$, i.e., $\mathbf{x}'(i) = u_i$ or l_i $(1 \leq i \leq n)$.

We write $f(\mathcal{I})$ for the values of f on $\mathcal{I}$: $f(\mathcal{I}) = \{f(\mathbf{x}) \mid \mathbf{x} \in \mathcal{I}\}$. Due to the monotonicity of f, the maximum and minimum values of f on $\mathcal{I}$ may be easily calculated by applying f to vertices of $\mathcal{I}$:

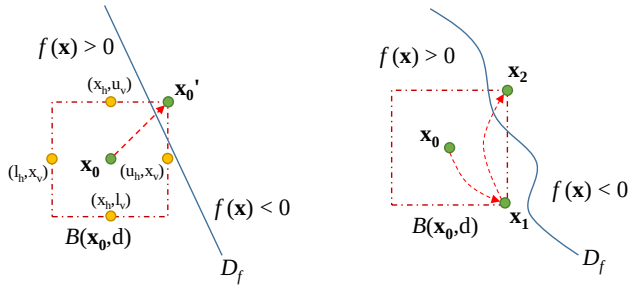
Theorem 1. Given a linear classification function $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$, where $\mathbf{w}^T = (w_1, ..., w_n)$ and $b \in \mathbb{R}$, an n-dimensional cube $\mathcal{I} = I_1 \times ... \times I_n$, where $I_i = [l_i, u_i]$ for $1 \leq i \leq n$, and an input $\mathbf{x} \in \mathcal{I}$, we have:

(1) $\min f(\mathcal{I}) = f(\mathbf{x}')$, where $\mathbf{x}'(i) = l_i$ (resp. $\mathbf{x}'(i) = u_i$) if $f(\mathbf{x}[u_i/x_i]) > f(\mathbf{x}[l_i/x_i])$ (resp. $f(\mathbf{x}[u_i/x_i]) \leq f(\mathbf{x}[l_i/x_i])$) for $1 \leq i \leq n$
(2) $\max f(\mathcal{I}) = f(\mathbf{x}')$, where $\mathbf{x}'(i) = u_i$ (resp. $\mathbf{x}'(i) = l_i$) if $f(\mathbf{x}[u_i/x_i]) > f(\mathbf{x}[l_i/x_i])$ (resp. $f(\mathbf{x}[u_i/x_i]) \leq f(\mathbf{x}[l_i/x_i])$) for $1 \leq i \leq n$

According to the above theorem, we can *precisely* calculate the minimum and maximum values of f in any n-dimensional cube. In particular, assume a correctly classified input $\mathbf{x}$ for which $f(\mathbf{x}) > 0$. For each dimension i $(1 \leq i \leq n)$ of $\mathbf{x}$, we first construct inputs $\mathbf{x}[l_i/x_i]$ and $\mathbf{x}[u_i/x_i]$. We then compare the values of $f(\mathbf{x}[l_i/x_i])$ and $f(\mathbf{x}[u_i/x_i])$. To generate a new input $\mathbf{x}'$, we select the value of its ith coordinate as follows:

$$\mathbf{x}'(i) = \begin{cases} l_i, & \text{if } f(\mathbf{x}[u_i/x_i]) > f(\mathbf{x}[l_i/x_i]) \\ u_i, & \text{if } f(\mathbf{x}[u_i/x_i]) \leq f(\mathbf{x}[l_i/x_i]) \end{cases}$$

As shown here, selecting a value for $\mathbf{x}'(i)$ requires evaluating function f twice, i.e., $f(\mathbf{x}[l_i/x_i])$ and $f(\mathbf{x}[u_i/x_i])$. Therefore, for n dimensions, f must be evaluated $2n$ times. In practice however, due to the monotonicity of f, evaluating it only once per dimension is sufficient. For instance, if $f(\mathbf{x}[u_i/x_i])$ already decreases (resp. increases) the value of f in comparison to $f(\mathbf{x})$, there is no need to evaluate $f(\mathbf{x}[l_i/x_i])$. Value u_i (resp. l_i) should be selected for the



(a) Linear classifier (b) Non-linear classifier

Figure 1: DeepSearch for binary classifiers.

ith coordinate. Hence, the minimum value of f can be computed by evaluating the function exactly n times. If, for the newly generated input $\mathbf{x}'$, the sign of f becomes negative, $\mathbf{x}'$ constitutes an adversarial example.

We treat the case where $f(\mathbf{x}) < 0$ for a correctly classified input $\mathbf{x}$ analogously. DeepSearch aims to generate a new input $\mathbf{x}'$ such that the sign of f becomes positive. We are, therefore, selecting coordinate values that cause f to *increase*.

2.2 Non-Linear Binary Classifiers

We generalize our technique to non-linear binary classifiers. In this setting, DeepSearch iteratively *approximates* the minimum and maximum values of f in $\mathcal{B}(\mathbf{x}, d)$.

Example. As an example, consider the non-linear classification function f shown in Fig. 1b. Since f is non-linear, the decision boundary $\mathcal{D}_f$ of the binary classifier is a curve.

Starting from correctly classified input $\mathbf{x}_0$, DeepSearch treats f as linear within $\mathcal{B}(\mathbf{x}_0, d)$ and generates $\mathbf{x}_1$, exactly as it would for a linear binary classifier. Observe that input $\mathbf{x}_1$ is not adversarial. Unlike for a linear binary classifier however, where the minimum value of f in $\mathcal{B}(\mathbf{x}_0, d)$ is precisely computed, the non-linear case is handled by iteratively approximating the minimum. In particular, after generating $\mathbf{x}_1$, DeepSearch iterates starting from $\mathbf{x}_1$, while again treating f as linear in $\mathcal{B}(\mathbf{x}_0, d)$. As a result, our technique generates input $\mathbf{x}_2$, which is adversarial.

DeepSearch for non-linear binary classifiers. Alg. 1 shows DeepSearch for binary classifiers. It uses iterative approximations to search for adversarial examples. Note that our technique is blackbox, and consequently, it cannot differentiate between linear and non-linear classifiers. Alg. 1 is, therefore, the general algorithm that DeepSearch applies to fuzz any binary classifier.

The main function in Alg. 1 is `DS-Binary`. Input $\mathbf{x}_{init}$ is the input from which we start the first iteration, e.g., it corresponds to $\mathbf{x}_0$ in Fig. 1. Input $\mathbf{x}$ is used to compute $\mathcal{B}(\mathbf{x}, d)$, and for now, assume that $\mathbf{x}$ is equal to $\mathbf{x}_{init}$. (We will discuss why $\mathbf{x}$ is needed in Sect. 4.) In addition to these inputs, the algorithm also takes a classification function f and the distance d. If it terminates, it returns an adversarial example $\mathbf{x}'$.

Function `DS-Binary` first assigns $\mathbf{x}_{init}$ to $\mathbf{x}_0$ and constructs n intervals $I_1, ..., I_n$ to represent $\mathcal{B}(\mathbf{x}_0, d)$ (lines 20–21). Then, based on the sign of $f(\mathbf{x}_0)$, our algorithm iteratively approximates the minimum (lines 23–26) or the maximum (lines 28–31) value of f in $\mathcal{B}(\mathbf{x}_0, d)$. Each iteration evaluates f $2n$ times (see Sect. 2.1).

Algorithm 1: DeepSearch for binary classifiers.

Input: input $\mathbf{x} \in \mathbb{R}^n$, initial input $\mathbf{x}_{init} \in \mathcal{B}(\mathbf{x}, d)$, function $f : \mathbb{R}^n \to \mathbb{R}$, distance $d \in \mathbb{R}$
Output: an adversarial input $\mathbf{x}'$

```
1  Function ApproxMax(x, f, (I_1, ..., I_n)) is
2      x' := (0, ..., 0)
3      foreach 1 ≤ i ≤ n do
4          if f(x[u_i/x_i]) > f(x[l_i/x_i]) then
5              x' := x'[u_i/x'_i]
6          else
7              x' := x'[l_i/x'_i]
8      return x'
9
10 Function ApproxMin(x, f, (I_1, ..., I_n)) is
11     x' := (0, ..., 0)
12     foreach 1 ≤ i ≤ n do
13         if f(x[u_i/x_i]) > f(x[l_i/x_i]) then
14             x' := x'[l_i/x'_i]
15         else
16             x' := x'[u_i/x'_i]
17     return x'
18
19 Function DS-Binary(x, x_init, f, d) is
20     construct intervals (I_1, ..., I_n) such that B(x, d) = I_1 × ... × I_n
21     initialize x_0 := x_init and k := 0
22     if f(x_0) > 0 then
23         repeat
24             x_{k+1} := ApproxMin (x_k, f, (I_1, ..., I_n))
25             k := k + 1
26         until x_k is an adversarial example
27     else
28         repeat
29             x_{k+1} := ApproxMax (x_k, f, (I_1, ..., I_n))
30             k := k + 1
31         until x_k is an adversarial example
32     return x_k
```

`DS-Binary` terminates when an adversarial example is found. If such an example is found in k iterations, we evaluate f $2nk$ times. Note that, in practice, we set a bound on the number of iterations, which we discuss in Sect. 5.

`ApproxMin` and `ApproxMax` implement Thm. 1 to calculate the minimum and maximum values of function f in the n-dimensional cube $I_1 \times ... \times I_n$. When f is linear, calling these functions on any input $\mathbf{x} \in I_1 \times ... \times I_n$ does not affect the computation. In other words, the minimum and maximum values are precisely computed for any $\mathbf{x}$. When f is non-linear, it is still assumed to be linear within the n-dimensional cube. Given that the size of the cube is designed to be small, this assumption does not introduce too much imprecision. As a consequence of this assumption however, different inputs in the n-dimensional cube lead to computing different minimum and maximum values of f. For instance, in Fig. 1b, calling `ApproxMin` on $\mathbf{x}_0$ returns $\mathbf{x}_1$, while calling it on $\mathbf{x}_1$ returns $\mathbf{x}_2$.

3 FUZZING MULTICLASS CLASSIFIERS

In this section, we extend our technique for blackbox fuzzing of binary classifiers to multiclass classifiers.

3.1 Linear Multiclass Classifiers

Background. A *multiclass classifier* classifies inputs in m classes according to the following definition.

Definition 2 (Multiclass Classifier). For classification function $f : \mathbb{R}^n \to \mathbb{R}^m$, which returns m values each corresponding to one class in $C_m = \{l_1, ..., l_m\}$, a *multiclass classifier* $\mathcal{N}_f : \mathbb{R}^n \to C_m$ is defined as

$$\mathcal{N}_f(\mathbf{x}) = l_j, \qquad \text{iff } j = \arg\max_i f_i(\mathbf{x}),$$

where $f_i : \mathbb{R}^n \to \mathbb{R}$ denotes the function derived by evaluating f for the ith class, i.e., $f(\mathbf{x}) = (f_1(\mathbf{x}), ..., f_m(\mathbf{x}))^T$.

In other words, a multiclass classifier $\mathcal{N}_f$ classifies an input $\mathbf{x}$ in l_j if $f_j(\mathbf{x})$ evaluates to the largest value in comparison to all other functions f_i.

Function f of a multiclass classifier $\mathcal{N}_f$ may be decomposed into multiple binary classifiers such that the original multiclass classifier may be reconstructed from these binary classifiers. First, to decompose a multiclass classifier into binary classifiers, for any pair of classes l_i and l_j ($1 \le i, j \le m$), we define a classification function $g_{ij} : \mathbb{R}^n \to \mathbb{R}$ as $g_{ij}(\mathbf{x}) = f_i(\mathbf{x}) - f_j(\mathbf{x})$. We then construct a binary classifier $\mathcal{N}_{g_{ij}} : \mathbb{R}^n \to \{l_i, l_j\}$ as follows:

$$\mathcal{N}_{g_{ij}}(\mathbf{x}) = \begin{cases} l_i, & \text{if } g_{ij}(\mathbf{x}) > 0 \\ l_j, & \text{if } g_{ij}(\mathbf{x}) < 0 \end{cases}$$

As usual, the set of values $\mathcal{D}_{g_{ij}} = \{\mathbf{x} \mid f_i(\mathbf{x}) - f_j(\mathbf{x}) = 0\}$ constitutes the pairwise decision boundary of binary classifier $\mathcal{N}_{g_{ij}}$, which classifies the domain $\mathbb{R}^n$ into the two classes $\{l_i, l_j\}$. As an example, consider Fig. 2a depicting a multiclass classifier $\mathcal{N}_f : \mathbb{R}^2 \to C_3$, where f is linear and $C_3 = \{l_1, l_2, l_3\}$. Assume that $\mathcal{N}_f$ correctly classifies input $\mathbf{x}$ in l_2. Based on the above, linear binary classifiers $\mathcal{N}_{g_{21}}$ and $\mathcal{N}_{g_{23}}$ also classify $\mathbf{x}$ in l_2, i.e., $g_{21}(\mathbf{x}) > 0$ and $g_{23}(\mathbf{x}) > 0$.

Second, a multiclass classifier may be composed from multiple binary classifiers as follows. An input $\mathbf{x}$ is classified in class l_i by multiclass classifier $\mathcal{N}_f$ if and only if it is classified in l_i by all $m-1$ binary classifiers $\mathcal{N}_{g_{ij}}$ for $1 \le j \le m, i \ne j$, where $l_i \in C_m$ and $g_{ij}(\mathbf{x}) = f_i(\mathbf{x}) - f_j(\mathbf{x})$:

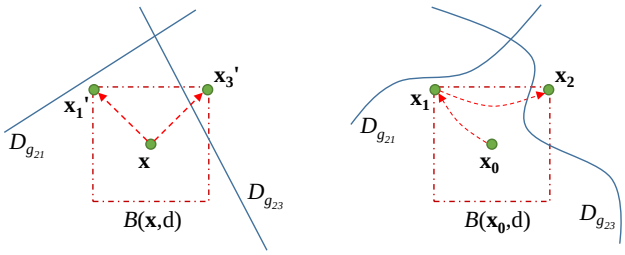
$$\mathcal{N}_f(\mathbf{x}) = l_i, \qquad \text{iff } \forall 1 \le j \le m, i \ne j : \mathcal{N}_{g_{ij}}(\mathbf{x}) = l_i$$

For instance, in Fig. 2a, if both $\mathcal{N}_{g_{21}}$ and $\mathcal{N}_{g_{23}}$ classify input $\mathbf{x}$ in class l_2, then the multiclass classifier also classifies it in l_2.

Based on the above, a multiclass classifier has an adversarial input if and only if this input is also adversarial for a constituent binary classifier:

Corollary 1. Let $\mathcal{N}_f$ be a multiclass classifier and $\mathbf{x} \in \mathbb{R}^n$ a correctly classified input, where $\mathcal{N}_f(\mathbf{x}) = l_i$ and $l_i \in C_m$. There exists an adversarial example $\mathbf{x}' \in \mathcal{B}(\mathbf{x}, d)$ for $\mathcal{N}_f$, where $d \in \mathbb{R}$, if and only if $\mathbf{x}'$ is an adversarial example for a binary classifier $\mathcal{N}_{g_{ij}}$ ($1 \le j \le m, i \ne j$), where $g_{ij}(\mathbf{x}) = f_i(\mathbf{x}) - f_j(\mathbf{x})$:

$$\mathcal{N}_f(\mathbf{x}') \ne l_i, \qquad \text{iff } \exists 1 \le j \le m, i \ne j : \mathcal{N}_{g_{ij}}(\mathbf{x}') \ne l_i$$



(a) Linear classifier **(b) Non-linear classifier**

Figure 2: DeepSearch for multiclass classifiers.

Example. This corollary is crucial in generalizing our technique to multiclass classifiers. Assume a correctly classified input $\mathbf{x}$, for which $\mathcal{N}_f(\mathbf{x}) = l_i$. According to the above corollary, the robustness of $\mathcal{N}_f$ in $\mathcal{B}(\mathbf{x}, d)$ reduces to the robustness of all $m-1$ binary classifiers $\{\mathcal{N}_{g_{ij}} \mid 1 \le j \le m, i \ne j\}$ in $\mathcal{B}(\mathbf{x}, d)$. We, therefore, use DeepSearch for binary classifiers to test each binary classifier in this set. If there exists an adversarial input $\mathbf{x}'$ for one of these classifiers, i.e., $\mathcal{N}_{g_{ij}}(\mathbf{x}') \ne l_i$ for some j, then $\mathbf{x}'$ is also an adversarial input for $\mathcal{N}_f$, i.e., $\mathcal{N}_f(\mathbf{x}') \ne l_i$.

Let us consider again the example of Fig. 2a. Recall that multiclass classifier $\mathcal{N}_f$ correctly classifies input $\mathbf{x}$ in class l_2, and so do binary classifiers $\mathcal{N}_{g_{21}}$ and $\mathcal{N}_{g_{23}}$, i.e., $g_{21}(\mathbf{x}) > 0$ and $g_{23}(\mathbf{x}) > 0$. As a result, DeepSearch tries to generate inputs that decrease the value of each of these functions in $\mathcal{B}(\mathbf{x}, d)$ in order to find adversarial examples. Function g_{21} evaluates to its minimum value in $\mathcal{B}(\mathbf{x}, d)$ for input $\mathbf{x}'_1$, and function g_{23} for input $\mathbf{x}'_3$. Observe that $\mathbf{x}'_3$ is an adversarial example for $\mathcal{N}_{g_{23}}$, and thus also for $\mathcal{N}_f$, whereas $\mathbf{x}'_1$ is not.

DeepSearch for linear multiclass classifiers. Let us assume a linear classification function $f(\mathbf{x}) = \mathbf{W}^T\mathbf{x} + \mathbf{b}$, where $\mathbf{W}^T = (\mathbf{w}_1^T, ..., \mathbf{w}_m^T)^T$, $\mathbf{w}_i \in \mathbb{R}^n$ ($1 \le i \le m$), and $\mathbf{b} = (b_1, ..., b_m)^T \in \mathbb{R}^m$. Then, f_i, which denotes the function derived by evaluating f for the ith class, is of the form $f_i(\mathbf{x}) = \mathbf{w}_i^T\mathbf{x} + b_i$ for $1 \le i \le m$. For any pair of class labels l_i and l_j, function g_{ij} is defined as $g_{ij}(\mathbf{x}) = f_i(\mathbf{x}) - f_j(\mathbf{x}) = (\mathbf{w}_i^T - \mathbf{w}_j^T)\mathbf{x} + (b_i - b_j)$. Hence, g_{ij} is also linear, and $\mathcal{N}_{g_{ij}}$ is a linear binary classifier.

Assume that classifier $\mathcal{N}_f$ correctly classifies input $\mathbf{x} \in \mathbb{R}^n$ in l_i, $\mathcal{N}_f(\mathbf{x}) = l_i$ ($l_i \in C_m$). According to Cor. 1, the robustness of $\mathcal{N}_f$ in $\mathcal{B}(\mathbf{x}, d)$ ($d \in \mathbb{R}$) reduces to the robustness of each binary classifier $\mathcal{N}_{g_{ij}}$ ($1 \le j \le m, i \ne j$) in $\mathcal{B}(\mathbf{x}, d)$. To find an adversarial example for a binary classifier $\mathcal{N}_{g_{ij}}$ in $\mathcal{B}(\mathbf{x}, d)$, DeepSearch must generate an input $\mathbf{x}' \in \mathcal{B}(\mathbf{x}, d)$ such that $g_{ij}(\mathbf{x}') < 0$. (Recall that by definition $g_{ij}(\mathbf{x}) > 0$.) Since all functions g_{ij} ($1 \le j \le m, i \ne j$) are linear, we easily find their minimum values in $\mathcal{B}(\mathbf{x}, d)$ as follows.

Let $I_1, ..., I_n$ be intervals such that $\mathcal{B}(\mathbf{x}, d) = I_1 \times ... \times I_n$, where $I_k = [l_k, u_k]$ for $1 \le k \le n$. As in Sect. 2.1, for each dimension k, DeepSearch evaluates function f twice to compare the values of $f(\mathbf{x}[u_k/x_k])$ and $f(\mathbf{x}[l_k/x_k])$. To generate a new input $\mathbf{x}'_j$ for which function g_{ij} evaluates to its minimum value, we select its kth coordinate as follows:

$$\mathbf{x}'_j(k) = \begin{cases} l_k, & \text{if } g_{ij}(\mathbf{x}[u_k/x_k]) > g_{ij}(\mathbf{x}[l_k/x_k]) \\ u_k, & \text{if } g_{ij}(\mathbf{x}[u_k/x_k]) \le g_{ij}(\mathbf{x}[l_k/x_k]). \end{cases}$$

Note that, although we calculate the minimum value of $m-1$ linear functions, we still evaluate f $2n$ times. This is because a

Algorithm 2: DeepSearch for multiclass classifiers.

Input: input $\mathbf{x} \in \mathbb{R}^n$, initial input $\mathbf{x}_{init} \in \mathcal{B}(\mathbf{x}, d)$,
function $f : \mathbb{R}^n \to \mathbb{R}^m$, distance $d \in \mathbb{R}$
Output: an adversarial input $\mathbf{x}'$

1 **Function** `DS-Multiclass`($\mathbf{x}$, $\mathbf{x}_{init}$, f, d) **is**
2 construct intervals $(I_1, \ldots, I_n)$ such that $\mathcal{B}(\mathbf{x}, d) = I_1 \times \ldots \times I_n$
3 $l_i := \mathcal{N}_f(\mathbf{x})$
4 define functions $\{g_{ij} \mid g_{ij}(\mathbf{x}) = f_i(\mathbf{x}) - f_j(\mathbf{x}), 1 \le j \le m, i \ne j\}$
5 initialize $\mathbf{x}_0 := \mathbf{x}_{init}$ and $k := 0$
6 **repeat**
7 $r := \arg\min_j g_{ij}(\mathbf{x}_k)$
8 $\mathbf{x}_{k+1} :=$ `ApproxMin`($\mathbf{x}_k$, g_{ir}, $(I_1, \ldots, I_n)$)
9 $k := k + 1$
10 **until** $\mathbf{x}_k$ is an adversarial example
11 **return** $\mathbf{x}_k$

function g_{ij} is defined as $g_{ij}(\mathbf{x}) = f_i(\mathbf{x}) - f_j(\mathbf{x})$, where $f_i(\mathbf{x})$ and $f_j(\mathbf{x})$ are the values of $f(\mathbf{x})$ for the ith and jth classes, respectively. If the sign of $g_{ij}(\mathbf{x}'_j)$ becomes negative for some j, then DeepSearch has found an adversarial example for $\mathcal{N}_f$ in $\mathcal{B}(\mathbf{x}, d)$.

3.2 Non-Linear Multiclass Classifiers

We now extend our technique to non-linear multiclass classifiers. Analogously to Sect. 2.2, DeepSearch iteratively approximates the minimum values of functions g_{ij} in $\mathcal{B}(\mathbf{x}, d)$.

Example. As an example, consider Fig. 2b depicting a multiclass classifier $\mathcal{N}_f : \mathbb{R}^2 \to C_3$, where f is non-linear and $C_3 = \{l_1, l_2, l_3\}$. Assume that $\mathcal{N}_f$ classifies input $\mathbf{x}_0$ in class l_2, and thus, so do non-linear binary classifiers $\mathcal{N}_{g_{21}}$ and $\mathcal{N}_{g_{23}}$.

Let us also assume that $g_{21}(\mathbf{x}_0) < g_{23}(\mathbf{x}_0)$. Since g_{21} evaluates to a smaller value than g_{23} for input $\mathbf{x}_0$, we consider it more likely to have an adversarial example. In other words, we first approximate the minimum value of g_{21} because it is closer to becoming negative for the initial input. DeepSearch treats g_{21} as linear within $\mathcal{B}(\mathbf{x}_0, d)$ and generates $\mathbf{x}_1$. Observe that input $\mathbf{x}_1$ is not adversarial. Now, assume that $g_{21}(\mathbf{x}_1) > g_{23}(\mathbf{x}_1)$. As a result, DeepSearch tries to find the minimum of function g_{23} in $\mathcal{B}(\mathbf{x}_0, d)$, also by treating it as linear. It generates input $\mathbf{x}_2$, which is an adversarial example for classifiers $\mathcal{N}_{g_{23}}$ and $\mathcal{N}_f$.

DeepSearch for non-linear multiclass classifiers. Alg. 2 is the general DeepSearch algorithm for multiclass classifiers. The inputs are the same as for Alg. 1. For now, assume that $\mathbf{x}$ is equal to $\mathbf{x}_{init}$. Again, if the algorithm terminates, it returns an adversarial example for $\mathcal{N}_f$ in $\mathcal{B}(\mathbf{x}, d)$.

Function `DS-Multiclass` assigns $\mathbf{x}_{init}$ to $\mathbf{x}_0$ and constructs n intervals $I_1, \ldots, I_n$ to represent $\mathcal{B}(\mathbf{x}_0, d)$. It also computes the class label l_i of $\mathbf{x}_0$, and defines functions g_{ij} $(1 \le j \le m, i \ne j)$ (lines 2–5). The rest of the algorithm uses `ApproxMin` from Alg. 1 to iteratively approximate the minimum of one function g_{ij} per iteration. This function is selected on line 7 such that its value for input $\mathbf{x}_k$ is smaller in comparison to all other constituent binary classification functions. This heuristic allows our algorithm to find an adversarial example faster than having to generate an input $\mathbf{x}_{k+1}$ for all $m - 1$ functions g_{ij} per iteration. `DS-Multiclass` terminates when an adversarial example is found. If such an example is found in k iterations, we evaluate f $2nk$ times.

4 FUZZING WITH ITERATIVE REFINEMENT

The closer the adversarial examples are to a correctly classified input, the more subtle they are. Such adversarial examples are said to have a low distortion rate. In this section, we extend DeepSearch with an iterative refinement approach for finding subtle adversarial examples. On a high level, given an input $\mathbf{x}$ and a distance d, for which we have already found an adversarial example $\mathbf{x}'$ in region $\mathcal{B}(\mathbf{x}, d)$, DeepSearch iteratively reduces distance d as long as the smaller region still contains an adversarial example. If none is found, the distance is not reduced further.

Background. Let $\mathcal{I} = I_1 \times \ldots \times I_n$ be an n-dimensional cube, where $I_i = [l_i, u_i]$ is a closed interval bounded by $l_i, u_i \in \mathbb{R}$ with $l_i \le u_i$ for $1 \le i \le n$. For an input $\mathbf{x}$ with an ith coordinate $\mathbf{x}(i) \in (-\infty, l_i] \cup [u_i, +\infty)$ for $1 \le i \le n$, we define a projection operator Proj that maps $\mathbf{x}$ to a vertex of $\mathcal{I}$ as follows

$$\textsc{Proj}(\mathcal{I}, \mathbf{x})(i) = \begin{cases} u_i, & \text{if } \mathbf{x}(i) \ge u_i \\ l_i, & \text{if } \mathbf{x}(i) \le l_i, \end{cases}$$

where $\textsc{Proj}(\mathcal{I}, \mathbf{x})(i)$ denotes the ith coordinate of $\textsc{Proj}(\mathcal{I}, \mathbf{x})$. As an example, consider Fig. 3a showing a linear multiclass classifier. Input $\mathbf{x}_2$ is a projection of $\mathbf{x}_1$.

Using the above projector operator, the minimum and maximum values of a linear classification function f may also be projected on $\mathcal{I}$. In particular:

Theorem 2. Let $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$ be a linear classification function, and $\mathcal{I}_1$, $\mathcal{I}_2$ two n-dimensional cubes such that $\mathcal{I}_1 \subseteq \mathcal{I}_2$. Assuming that $\mathbf{x}$ is an input for which f obtains its minimum or maximum value in $\mathcal{I}_2$, we have:

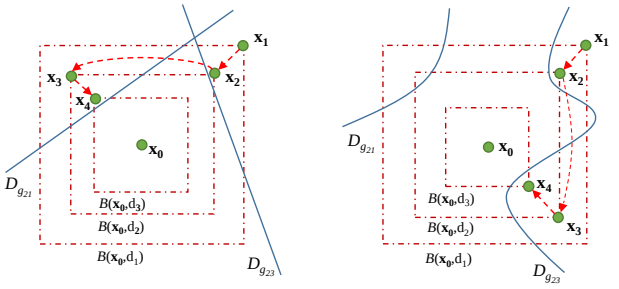
(1) if $\min f(\mathcal{I}_2) = f(\mathbf{x})$, then $\min f(\mathcal{I}_1) = f(\textsc{Proj}(\mathcal{I}_1, \mathbf{x}))$
(2) if $\max f(\mathcal{I}_2) = f(\mathbf{x})$, then $\max f(\mathcal{I}_1) = f(\textsc{Proj}(\mathcal{I}_1, \mathbf{x}))$

In Fig. 3a, assume that input $\mathbf{x}_0$ is correctly classified in class l_2. Then, in region $\mathcal{B}(\mathbf{x}_0, d_1)$, function g_{23} obtains its minimum value for input $\mathbf{x}_1$. When projecting $\mathbf{x}_1$ to vertex $\mathbf{x}_2$ of $\mathcal{B}(\mathbf{x}_0, d_2)$, notice that g_{23} evaluates to its minimum for input $\mathbf{x}_2$ in this smaller region.

Example. Fig. 3 shows two multiclass classifiers $\mathcal{N}_f : \mathbb{R}^2 \to C_3$, where $C_3 = \{l_1, l_2, l_3\}$. In Fig. 3a, function f is linear, whereas in Fig. 3b, it is non-linear. For correctly classified input $\mathbf{x}_0$, we assume that $\mathcal{N}_f(\mathbf{x}_0) = l_2$, and thus, $\mathcal{N}_{g_{21}}(\mathbf{x}_0) = l_2$ and $\mathcal{N}_{g_{23}}(\mathbf{x}_0) = l_2$.

In both subfigures, assume that $\mathbf{x}_1$ is an adversarial example found by `DS-Multiclass` (see Alg. 2) in $\mathcal{B}(\mathbf{x}_0, d_1)$. Once such an example is found, our technique with refinement uses bisect search to find the smallest distance d' such that the projection of $\mathbf{x}_1$ on $\mathcal{B}(\mathbf{x}_0, d')$ is still adversarial. In Fig. 3, this distance is d_2, and input $\mathbf{x}_2$ constitutes the projection of $\mathbf{x}_1$ on $\mathcal{B}(\mathbf{x}_0, d_2)$. So, $\mathbf{x}_2$ is closer to $\mathbf{x}_0$, which means that it has a lower distortion rate than $\mathbf{x}_1$. In fact, since we are using bisect search to determine distance d_2, $\mathbf{x}_2$ is the closest adversarial input to $\mathbf{x}_0$ that may be generated by projecting $\mathbf{x}_1$ on smaller regions.

However, in region $\mathcal{B}(\mathbf{x}_0, d_2)$, there may be other vertices that are adversarial and get us even closer to $\mathbf{x}_0$ with projection. To find such examples, we apply `DS-Multiclass` again, this time starting from input $\mathbf{x}_2$ and searching in region $\mathcal{B}(\mathbf{x}_0, d_2)$. As a result, we



(a) Linear classifier **(b) Non-linear classifier**

Figure 3: DeepSearch with iterative refinement.

generate adversarial input $\mathbf{x}_3$ in the subfigures. Now, by projecting $\mathbf{x}_3$ to the smallest possible region around $\mathbf{x}_0$, we compute $\mathbf{x}_4$, which is the adversarial example with the lowest distortion rate so far.

Assume that applying `DS-Multiclass` for a third time, starting from $\mathbf{x}_4$ and searching in $\mathcal{B}(\mathbf{x}_0, d_3)$, does not generate any other adversarial examples. In this case, our technique returns $\mathbf{x}_4$.

DeepSearch with iterative refinement. Alg. 3 describes our technique with iterative refinement. Each iteration consists of a refinement and a search step, which we explain next.

The refinement step (lines 3–4) first calculates the L_∞ distance d between $\mathbf{x}$ and $\mathbf{x}'$. In Fig. 3, input $\mathbf{x}$ of the algorithm is $\mathbf{x}_0$, and $\mathbf{x}'$ is $\mathbf{x}_1$. So, $\mathbf{x}'$ is an adversarial input that was generated by our technique, and consequently, a vertex of $\mathcal{B}(\mathbf{x}, d)$. On line 4, we use bisect search to find the minimum distance d' such that the input derived by projecting $\mathbf{x}'$ on $\mathcal{B}(\mathbf{x}, d')$ is still adversarial. In Fig. 3, this is distance d_2, and $\textsc{Proj}(\mathcal{B}(\mathbf{x}, d'), \mathbf{x}')$ of the algorithm corresponds to adversarial input $\mathbf{x}_2$ in the figure.

Note that this refinement is possible because of Thm. 2, which guarantees that a linear function f evaluates to its minimum in $\mathcal{B}(\mathbf{x}, d')$ for the input derived with projection. When f is non-linear, it might not evaluate to its minimum for adversarial input $\textsc{Proj}(\mathcal{B}(\mathbf{x}, d'), \mathbf{x}')$. However, it is still the case that this projected input is closer to $\mathbf{x}$, i.e., $||\mathbf{x} - \textsc{Proj}(\mathcal{B}(\mathbf{x}, d'), \mathbf{x}')||_{L_\infty} \leq ||\mathbf{x} - \mathbf{x}'||_{L_\infty}$, and thus, has a lower distortion rate.

The search step (lines 5–8) calls function `DS-Binary` (Alg. 1) or `DS-Multiclass` (Alg. 2), depending on whether f is a binary classification function. The goal is to search for another adversarial example (other than $\textsc{Proj}(\mathcal{B}(\mathbf{x}, d'), \mathbf{x}')$) in region $\mathcal{B}(\mathbf{x}, d')$. In Fig. 3, an adversarial input found by this step, when starting the search from $\mathbf{x}_2$, is $\mathbf{x}_3$, which is also a vertex of $\mathcal{B}(\mathbf{x}_0, d_2)$.

On lines 9–12, we essentially check whether the search step was successful in finding another adversarial input $\mathbf{x}''$. Recall that `DS-Binary` and `DS-Multiclass` only terminate when such an input is found, but in practice, we bound their number of iterations. If this bound is reached, they might not return an adversarial example. If they do, like input $\mathbf{x}_3$ in Fig. 3, the algorithm iterates (line 13). If not, like when starting the search from input $\mathbf{x}_4$ in the figure, which is a projection of $\mathbf{x}_3$ on $\mathcal{B}(\mathbf{x}_0, d_3)$, then we return the projected input and terminate.

5 EXPERIMENTAL EVALUATION

In this section, we evaluate DeepSearch by using it to test the robustness of deep neural networks trained for popular datasets. We also compare its effectiveness with state-of-the-art black- and whitebox approaches. Our experiments are designed around the following research questions:

RQ1: Is DeepSearch more effective than random fuzzing?
RQ2: Is DeepSearch more effective than state-of-the-art techniques for finding adversarial examples?
RQ3: Is DeepSearch with iterative refinement effective in finding adversarial examples with low distortion rates?
RQ4: How does reducing the number of queries to the network under test affect the effectiveness of DeepSearch?

In Sect. 5.3, we present experimental results providing answers to these questions. Note that the fourth question uses the term 'queries' to refer to the number of times we evaluate classification function f, e.g., $2nk$ times in Algs. 1 and 2. Sects. 5.1 and 5.2 discuss our experimental setup and the metrics we use to evaluate and compare the effectiveness of our technique.

Algorithm 3: DeepSearch with iterative refinement.

Input: input $\mathbf{x} \in \mathbb{R}^n$, adversarial input $\mathbf{x}' \in \mathcal{B}(\mathbf{x}, d)$ $(d \in \mathbb{R})$, function $f : \mathbb{R}^n \to \mathbb{R}^m$
Output: an adversarial input $\mathbf{x}'' \in \mathcal{B}(\mathbf{x}, d')$ $(d' \leq d)$

1 **Function** `DS-Refinement`$(\mathbf{x}, \mathbf{x}', f)$ **is**
2 **repeat**
3 $d := ||\mathbf{x} - \mathbf{x}'||_{L_\infty}$
4 apply bisect search to find the smallest distance $d' \leq d$ such that input $\textsc{Proj}(\mathcal{B}(\mathbf{x}, d'), \mathbf{x}')$ is an adversarial example
5 **if** f is binary **then**
6 $\mathbf{x}'' :=$ `DS-Binary`$(\mathbf{x}, \textsc{Proj}(\mathcal{B}(\mathbf{x}, d'), \mathbf{x}'), f, d')$
7 **else**
8 $\mathbf{x}'' :=$ `DS-Multiclass`$(\mathbf{x}, \textsc{Proj}(\mathcal{B}(\mathbf{x}, d'), \mathbf{x}'), f, d')$
9 **if** $\mathbf{x}''$ is an adversarial example **then**
10 $\mathbf{x}' := \mathbf{x}''$
11 **else**
12 $\mathbf{x}' := \textsc{Proj}(\mathcal{B}(\mathbf{x}, d'), \mathbf{x}')$
13 **until** $\mathbf{x}''$ is not an adversarial example
14 **return** $\mathbf{x}'$ and d'

5.1 Evaluation Setup

Datasets and network models. We evaluate our approach on deep neural networks trained for two popular datasets, namely MNIST [42] and CIFAR-10 [37]. For MNIST, we trained a LeNet-5 [41] network with 99.2% test accuracy, and for CIFAR-10, a ResNet-32 [28] network with 92.19% test accuracy.

For each dataset, we first randomly selected 1100 images from the test set and carried out our experiments only on the images that were correctly classified by the networks we trained. For MNIST, 1085 images were correctly classified by the LeNet-5 network, and for CIFAR-10, 1024 images by the ResNet-32 network.

Random fuzzing. Random fuzzing is a very simple technique for testing the robustness of a neural network, and in practice, it can prove effective in finding adversarial examples. In the following, we introduce two random-fuzzing approaches that we use as baselines in our experiments.

Basic random fuzzing For $\mathbf{x} \in \mathbb{R}^n$ and $d \in \mathbb{R}$, we generate 6000 images within $\mathcal{B}(\mathbf{x}, d)$ with random mutations.

Iterative random fuzzing For $\mathbf{x} \in \mathbb{R}^n$ and $d \in \mathbb{R}$, we perform iterative random fuzzing as follows.

(1) *Seed-pool initialization*: We first generate 1000 images within $\mathcal{B}(\mathbf{x}, d/5)$ with random mutations. From these, we select the 10 best images. In particular, for $f(\mathbf{x}) > 0$ (resp. $f(\mathbf{x}) < 0$), an image $\mathbf{x}'$ is selected if $f(\mathbf{x}')$ evaluates to one of the 10 smallest (resp. largest) values of f.
(2) *Iteration*: For each image $\mathbf{x}'$ in the pool, we generate 100 images within $\mathcal{B}(\mathbf{x}', d/5) \cap \mathcal{B}(\mathbf{x}, d)$ with random mutations. So, for all seed images, we derive a total of 1000 random images. From these, we then select the 10 best images and populate the pool with them for the next iteration. We iterate 9 times in our experiments.

Existing approaches. We compare DeepSearch with three state-of-the-art approaches for generating adversarial examples, which we describe below.

- The Fast Gradient Sign Method (FGSM) [24] is a popular whitebox technique, optimized for the L_∞ distance metric. For an input $\mathbf{x}$, it computes an adversarial example $\mathbf{x}'$ as $\mathbf{x}' = \mathbf{x} + \epsilon \cdot \text{sign}(\nabla_{\mathbf{x}} L_f(\mathbf{x}, l))$, where $L_f(\mathbf{x}, l)$ is the loss function, f the classification function, and l the label given to $\mathbf{x}$ by f.
- The Basic Iterative Method (BIM) [39], also called iterative FGSM, extends FGSM with iterations. Starting from an image $\mathbf{x}$, each iteration takes a small step α toward the gradient sign and clips the new image such that it is within $\mathcal{B}(\mathbf{x}, \epsilon)$. An adversarial example is computed by $\mathbf{x}'_0 = \mathbf{x}$ and $\mathbf{x}'_{i+1} = \mathbf{x}'_i + \text{clip}_\epsilon(\alpha \cdot \text{sign}(\nabla_{\mathbf{x}} L_f(\mathbf{x}, l)))$. In our experiments, we set $\alpha = 1$ and allow 120 (resp. 40) iterations for MNIST (resp. CIFAR-10).
- Gradient Estimation [5] is a blackbox technique that estimates gradients through queries to f. The effectiveness of single-step gradient estimation (sGE) is known to be comparable to FGSM. We use logits loss [5, 12] for sGE as it is more effective than cross-entropy loss [23] and set $\delta = 0.001$, where δ is a parameter that controls the estimation accuracy of this method.

Query reduction. Recall that for an n-dimensional input $\mathbf{x}$, our technique makes $2n$ queries per iteration. To reduce this number, we adapt random grouping [5] to our setting:

(1) We divide the n dimensions into $\lceil \frac{n}{k} \rceil$ ($0 < k \leq n$) sets $G_1, ..., G_{\lceil \frac{n}{k} \rceil}$, where each set G_i ($1 \leq i \leq \lceil \frac{n}{k} \rceil$) is derived by randomly selecting k indices from $\{1, ..., n\} / \{\cup_{j=1}^{i-1} G_j\}$. Note that there might only be $l < k$ remaining indices, in which case we select these.
(2) For each set G_i, our technique mutates all coordinates that correspond to indices in the set toward the same direction at the same time. Hence, DeepSearch makes only 2 queries per set, namely $f[u_{i_1}/x_{i_1}, ..., u_{i_l}/x_{i_l}]$ and $f[l_{i_1}/x_{i_1}, ..., l_{i_l}/x_{i_l}]$, where $i_1, ..., i_l \in G_i$ and $l = |G_i|$.

Consequently, the total number of queries per iteration reduces to $\lceil 2\frac{n}{k} \rceil$. In our experiments, we show how random grouping affects the effectiveness of DeepSearch.

DeepSearch termination criteria. In our experiments, Alg. 2 terminates when $f(\mathbf{x}_k)$ changes in the wrong direction 4 times. Alg. 3 does not terminate as long as the current iteration decreases the normalized L_∞ distance of adversarial examples by 0.001.

5.2 Metrics

To evaluate the effectiveness of DeepSearch, we use the following two metrics.

Misclassification rate. The misclassification rate measures the percentage of input images for which adversarial examples are found. The higher this rate, the more effective a given technique in finding adversarial examples. Assume we write $\mathit{findAdv}(\mathbf{x})$ to denote whether an adversarial example is found for input $\mathbf{x}$. If so, we have that $\mathit{findAdv}(\mathbf{x}) = 1$; Otherwise, we have $\mathit{findAdv}(\mathbf{x}) = 0$. For a set of images $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_k\}$, the misclassification rate of a given technique is:

$$\mathit{MisR}(\mathbf{X}) = \frac{1}{k} \sum_{i=1}^{k} \mathit{findAdv}(\mathbf{x}_i)$$

Average distortion rate. Let sets $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_k\}$ and $\mathbf{X}_{adv} = \{\mathbf{x}'_1, ..., \mathbf{x}'_k\}$ be input images and adversarial examples, respectively. The average distortion rate between $\mathbf{X}$ and $\mathbf{X}_{adv}$ with respect to the L_∞ distance is:

$$\mathit{AvgDR}_{L_\infty}(\mathbf{X}, \mathbf{X}_{adv}) = \frac{1}{k} \sum_{i=1}^{k} \frac{||\mathbf{x}_i - \mathbf{x}'_i||_{L_\infty}}{||\mathbf{x}_i||_{L_\infty}}$$

As shown here, the lower this rate, the more subtle the adversarial examples. For approaches that achieve similar misclassification rates, we use this metric to determine which approach finds more subtle perturbations.

Other metrics. We do not measure time in our experiments; our aim is to show that our blackbox technique is more effective than state-of-the-art whitebox approaches. These approaches are typically fast due to their whitebox nature—they have access to more information about the network under test. So, comparing time would disadvantage DeepSearch.

For comparing iterative approaches, we do not bound the number of iterations to the same fixed value. This is because the notion of an iteration is inherently different across approaches. We, instead, use the same distance metric (L_∞) for all approaches.

5.3 Experimental Results

In our experiments, we consider the following four variants of DeepSearch:

(1) Single-step DeepSearch (sD) implements Alg. 2 but executes lines 7–9 only once.
(2) Iterative DeepSearch (iD) implements Alg. 2 as is (using the termination criterion discussed in Sect. 5.1).
(3) DeepSearch with single-step refinement (DsR) implements Alg. 3 but executes lines 3–12 only once.
(4) DeepSearch with iterative refinement (DiR) implements Alg. 3 as is (using the termination criterion discussed in Sect. 5.1).

In the rest of this section, we use the following abbreviations and use both cross-entropy loss and logits loss for FGSM and BIM: bRF for basic random fuzzing, iRF for iterative random fuzzing,

FGSM-X (resp. FGSM-L) for FGSM with cross-entropy (resp. logits) loss, BIM-X (resp. BIM-L) for BIM with cross-entropy (resp. logits) loss.

Results on misclassification rate. Our experimental results on the misclassification rate of the different techniques are shown in Fig. 4.

Overall, DeepSearch is significantly more effective than the two random-fuzzing approaches (bRF and iRF). Observe that, for MNIST, random fuzzing is not as effective as for CIFAR-10. In fact, for CIFAR-10, random fuzzing is quite effective for a large L_∞ distance. As shown in the figure, the misclassification rate of bRF (resp. iRF) surpasses the three single-step approaches (sGE, sD, and FGSM) when the distance is greater than 24 (resp. 16).

When comparing single-step approaches, we observe that **single-step DeepSearch is more effective than sGE and whitebox FGSM for both MNIST and CIFAR-10**. For the MNIST model, the advantage of sD is more prominent when the L_∞ distance ranges from 0.15 to 0.4. For this distance range, the misclassification rate of sD is 10–20% higher than sGE and 15–28% higher than FGSM. For CIFAR-10, when the distance ranges from 4 to 16, the difference is again more noticeable. For this range, sD is 1–3% more effective than sGE and 4–7% more effective than FGSM. It is important to note here that **sD is more effective than sGE while performing half of the queries** to function f. Recall from Sect. 2.1 that, in practice, sD performs n queries, whereas sGE requires $2n$.

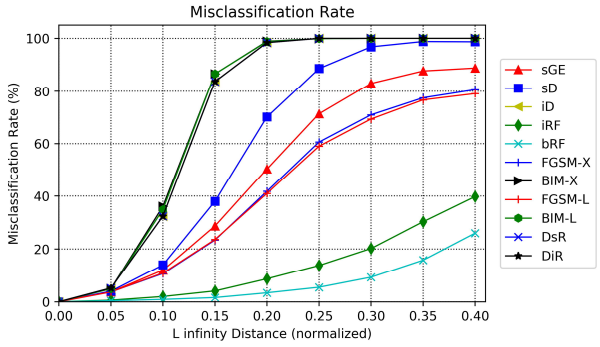
When comparing iterative approaches, **iterative DeepSearch closely matches the effectiveness of whitebox BIM for both models**. For the MNIST model, when the L_∞ distance is above 0.2, the misclassification rates of both iD and BIM are close to 100%. The results are similar for CIFAR-10 when the distance is above 4. Despite the comparable effectiveness of iD and BIM, DeepSearch with iterative refinement (DiR) achieves a lower distortion rate than BIM, as we discuss later in this section. We also mention our empirical observation that DeepSearch terminates in 3–4 iterations for the vast majority of input images (around 95%) for which DeepSearch finds adversarial examples.

For DeepSearch variants, we observe that iD is significantly more effective than sD. This confirms that the iterative version of DeepSearch (implemented in Alg. 2) is indeed effective in finding adversarial examples.

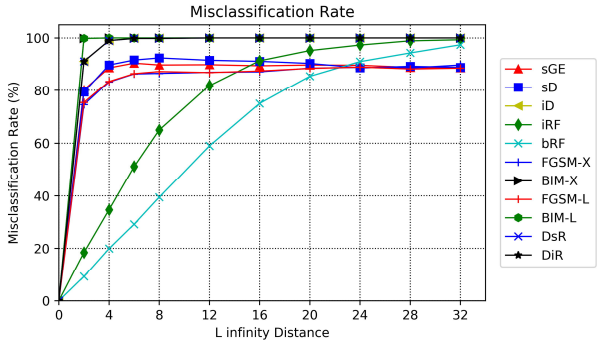
Results on average distortion rate. Our experimental results on the average distortion rate of the different techniques are shown in Fig. 5.

As shown in the figure, only DeepSearch with single-step and iterative refinement (DsR and DiR) as well as BIM with logits loss (BIM-L) are effective in finding subtle adversarial examples, even more so when the L_∞ distance is large. For both MNIST and CIFAR-10, **DeepSearch with iterative refinement achieves the lowest average distortion rate**.

For both models, BIM-L achieves a lower distortion rate than DsR. However, DiR is even better than BIM-L. Specifically, for MNIST, when the L_∞ distance is between 0.15 and 0.4, the distortion rate of DiR is 2–3% lower than BIM-L. For CIFAR-10, the distortion rates are similar but slightly better for DiR. Overall, **DeepSearch with iterative refinement is more effective in finding subtle adversarial examples than whitebox BIM**. This is a notable



(a) MNIST (LeNet-5)



(b) CIFAR-10 (ResNet-32)

Figure 4: Comparison of misclassification rates.

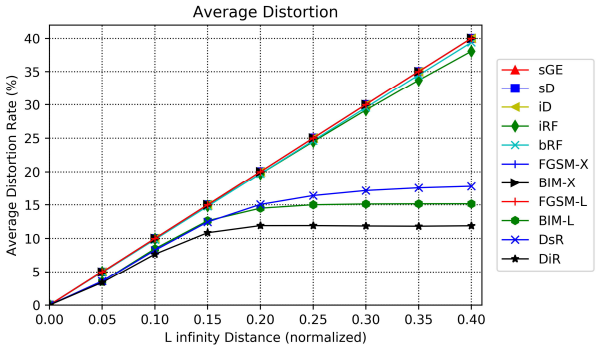
result given that DiR (or iD) also matches the misclassification rate of BIM-L despite its blackbox nature.

In comparison to iD, DsR reduces the distortion rate significantly. For MNIST, when the L_∞ distance ranges from 0.15 to 0.4, DsR reduces the distortion rate by 3–22%. For the CIFAR-10 model and a distance range between 12 and 32, DsR reduces the distortion rate by 4–12%. These percentages are even higher for DiR. In particular, for MNIST, DiR reduces the distortion rate by 4–28%, and for CIFAR-10, by 5–13%. These results confirm the effectiveness of refinement (Alg. 3) in generating subtle adversarial examples.

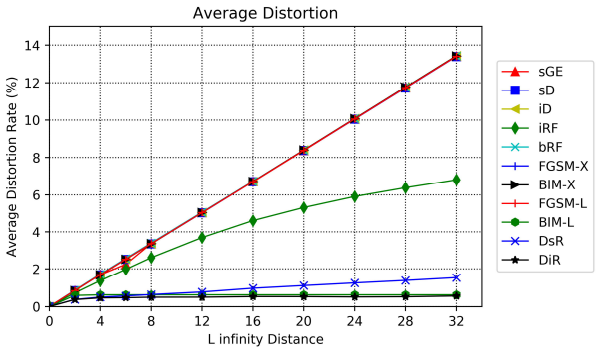
Results on query reduction for DeepSearch. We now show how query reduction based on random grouping affects DeepSearch with iterative refinement (DiR). Figs. 6 and 7 show how different values of k (see Sect. 5.1) impact the misclassification rate and average distortion rate of our technique. In our setting, k may range from 1 to 256.

As shown in Fig. 6, as k increases, the misclassification rate of DeepSearch decreases, and therefore, so does its effectiveness in finding adversarial examples. For example, for MNIST, when the L_∞ distance is between 0.1 and 0.4, the misclassification rate drops by 32–85% as k increases from 1 to 256. Similarly, for CIFAR-10, when the distance is between 4 to 32, the misclassification rate drops by 4–78%.

In Fig. 7, the average distortion rate of DeepSearch increases for larger values of k, and thus, its effectiveness in generating subtle

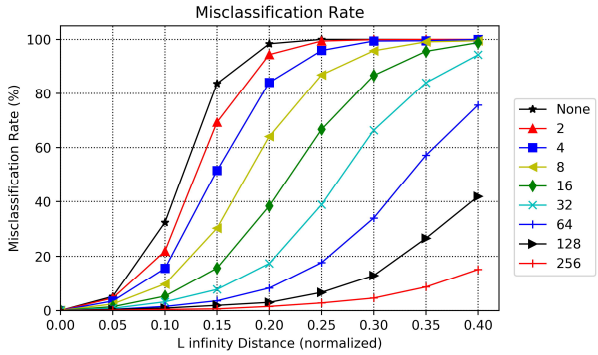


(a) MNIST (LeNet-5)

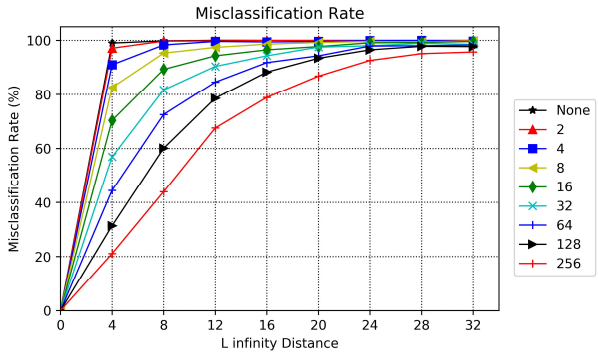


(b) CIFAR-10 (ResNet-32)

Figure 5: Comparison of average distortion rates.



(a) MNIST (LeNet-5)



(b) CIFAR-10 (ResNet-32)

Figure 6: Effect of random grouping on the misclassification rate of DeepSearch.

adversarial examples decreases. For example, for MNIST, when the L_∞ distance is between 0.2 and 0.4, there is a 4–20% increase in the distortion rate as k changes from 1 to 256. Similarly, for CIFAR-10, when the distance is between 4 and 32, the distortion rate increases by 1–4%.

Although the effectiveness of DeepSearch decreases as the value of k increases, the number of queries may still be reduced without compromising the advantage of DeepSearch over, say, whitebox FGSM. For example, for MNIST, when the distance is from 0.25 to 0.4 and $k \leq 16$, DeepSearch still achieves a higher misclassification rate and a lower distortion rate than FGSM. The same holds for CIFAR-10 when the distance is from 16 to 32 and $k \leq 128$.

5.4 Threats to Validity

We have identified the following three threats to the validity of our experiments.

Datasets and network models. Our experimental results may not generalize to other datasets or network models. However, we used two of the most popular datasets for image classification, MNIST and CIFAR-10. Moreover, our network models have very high test accuracy, 99.2% for LeNet-5 and 92.19% for ResNet-32.

Existing approaches. The second threat is related to the choice of existing approaches with which we compare. DeepSearch uses iterative linearization of non-linear networks and is tailored to the L_∞ distance metric. We, thus, compare with approaches that are similar in spirit.

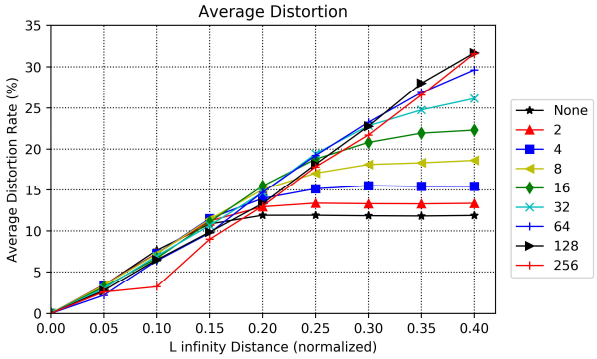
Both FGSM and BIM perform linear perturbations of non-linear models and are optimized for the L_∞ distance. DeepFool [50] also uses linearization of networks, but it is optimized for the L_2 distance, so we do not compare with it.

Blackbox gradient-estimation methods [5] are more effective than other query-based blackbox techniques. We selected sGE for comparison because it may be more effective than FGSM (see Fig. 4). On the other hand, we omitted iterative gradient estimation because, when step α (see Sect. 5.1) is sufficiently small, the effectiveness of the approach should be comparable to BIM, with which we compare.

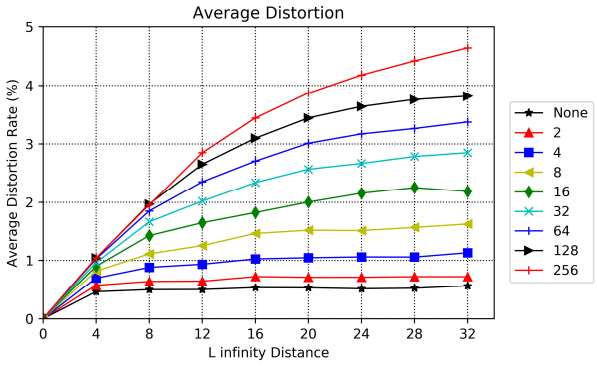
Fairness of comparison. The selection of parameters for each approach could affect the fairness of our comparison.

The number of iterations of BIM and iterative DeepSearch are not comparable; each iteration performs fundamentally different computations. We chose a bound for BIM that is sufficiently large for adversarial examples to reach the edge of the neighborhood around the original image. We set $\alpha = 1$ as suggested by the authors of BIM.

In sGE, δ affects the precision of the gradient estimation. We set $\delta = 0.001$, which is much more precise than the value chosen by the authors of sGE ($\delta = 0.01$).



(a) MNIST (LeNet-5)



(b) CIFAR-10 (ResNet-32)

Figure 7: Effect of random grouping on the average distortion rate of DeepSearch.

6 RELATED WORK

Adversarial robustness. Szegedy et al. [68] first discovered adversarial examples in neural networks and used box-constrained L-BFGS to find adversarial examples. Since then, multiple white-box adversarial attacks have been proposed: FGSM [24], BIM [39], DeepFool [50], JSMA [55], PGD [49] and C&W [12]. DeepSearch is closer to FGSM, BIM and DeepFool. Goodfellow et al. [24] first argued that the primary cause of adversarial examples is the linear nature of neural networks, and they proposed FGSM that allows fast generation of adversarial examples. BIM improved FGSM by extending it with iterative procedures. DeepFool [50] is another method that performs adversarial attacks through iterative linearization of neural networks. As a blackbox approach, DeepSearch differs with FGSM, BIM and DeepFool in its way to perform iterative search.

Blackbox adversarial attacks are more difficult than whitebox ones, and many blackbox attacks require a large number of queries. Papernot et al. [53, 54] explored blackbox attacks based on the phenomenon of transferability [53, 68]. Chen et al. [13] and Bhagoji et al. [5] proposed blackbox attacks based on gradient estimation [40, 65]. Uesato et al. [71] used SPSA [64], and Ilyas et al. [32] used NES [59]. Narodytska et al. [51] performed a local-searched based attack. The boundary attack [6] only requires access to the final decision of neural networks. DeepSearch is closer to the work of Bhagoji et al. [5] (which approximates FGSM and BIM), but DeepSearch does not rely on gradient estimation.

Although research on developing adversarial attacks is moving fast, research on defending neural networks against adversarial attacks is relatively slow [1, 2, 9–12, 15, 20, 29, 46, 62]. Many defense techniques are shown to be ineffective soon after they have been developed. We refer to the work of Carlini et al. [8] for a more detailed discussion on evaluating adversarial robustness.

Testing deep neural networks. Recently, significant progress has been made on testing neural networks. Several useful test coverage criteria have been proposed to guide test case generation: DeepXplore [56] proposed neuron coverage and the first whitebox testing framework for neural networks; DeepGauge[47] proposed a set of finer-grained test coverage criteria; DeepCT [48] further proposed combinatorial test coverage for neural networks; Sun et al. [66] proposed coverage criteria inspired by MC/DC; Kim et al. [35] proposed surprise adequacy for deep learning systems. Sekhon et al.[60] and Li et al. [43] pointed out the limitation of existing structural coverage criteria for neural networks. Li et al. [60] also discussed improvements for better coverage criteria.

Moreover, Sun et al. [67] proposed the first concolic testing [22, 61] approach for neural networks. DeepCheck [26] tests neural networks based on symbolic execution [14, 36]. TensorFuzz [52] proposed the first framework of coverage-guided fuzzing for neural networks. DeepHunter [78] considered various mutation strategies for their fuzzing framework. Wicker et al. [74] extracted features from images and computed adversarial examples using a two-player turned-based stochastic game. DLFuzz [27] proposed the first differential fuzzing framework for deep learning systems. DeepTest [69] and DeepRoad [80] proposed testing tools for autonomous driving systems based on deep neural networks. For more on testing neural networks, we refer to the work of Zhang et al. [79] that surveys testing of machine-learning systems.

Formal verification of deep neural networks. Verification of neural networks is more challenging than testing. Early work [57] used abstract interpretation [16] to verify small-sized neural networks. Recent work [25, 34, 63] used SMT [3] techniques and considered new abstract domains.

Liu et al. [44] classified recent work into five categories: Reachability-analysis based approaches include MaxSens [77], ExactReach [76], and AI^2[21]; NSVerify [45], MIPVerify [70] and ILP[4] are based on primal optimization; Duality [18], ConvDual [75] and Certify [58] use dual optimization; Fast-Lin & Fast-Lip [73], ReluVal [72] and DLV [31] combine reachability with search; Sherlock [17], Reluplex[33], Planet[19] and BaB[7] combine search with optimization. Lie et al. [44] provide a more detailed comparison and discussion on the above mentioned work.

7 CONCLUSION

We proposed and implemented DeepSearch, a novel blackbox-fuzzing technique for testing deep neural networks. DeepSearch is simple and effective in finding adversarial examples with low distortion. To the best of our knowledge, DeepSearch is among the early approaches advancing blackbox fuzzing for neural networks.

REFERENCES

[1] Anish Athalye and Nicholas Carlini. 2018. On the Robustness of the CVPR 2018 White-Box Adversarial Example Defenses. *CoRR* abs/1804.03286 (2018).

[2] Anish Athalye, Nicholas Carlini, and David A. Wagner. 2018. Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples. In *ICML (PMLR)*, Vol. 80. PMLR, 274–283.

[3] Clark W. Barrett and Cesare Tinelli. 2018. Satisfiability Modulo Theories. In *Handbook of Model Checking*. Springer, 305–343.

[4] Osbert Bastani, Yani Ioannou, Leonidas Lampropoulos, Dimitrios Vytiniotis, Aditya V. Nori, and Antonio Criminisi. 2016. Measuring Neural Net Robustness with Constraints. In *NIPS*. 2613–2621.

[5] Arjun Nitin Bhagoji, Warren He, Bo Li, and Dawn Song. 2018. Practical Black-Box Attacks on Deep Neural Networks Using Efficient Query Mechanisms. In *ECCV (LNCS)*, Vol. 11216. Springer, 158–174.

[6] Wieland Brendel, Jonas Rauber, and Matthias Bethge. 2018. Decision-Based Adversarial Attacks: Reliable Attacks Against Black-Box Machine Learning Models. In *ICLR*. OpenReview.net.

[7] Rudy Bunel, Ilker Turkaslan, Philip H. S. Torr, Pushmeet Kohli, and Pawan Kumar Mudigonda. 2018. A Unified View of Piecewise Linear Neural Network Verification. In *NeurIPS*. 4795–4804.

[8] Nicholas Carlini, Anish Athalye, Nicolas Papernot, Wieland Brendel, Jonas Rauber, Dimitris Tsipras, Ian J. Goodfellow, Aleksander Madry, and Alexey Kurakin. 2019. On Evaluating Adversarial Robustness. *CoRR* abs/1902.06705 (2019).

[9] Nicholas Carlini and David A. Wagner. 2016. Defensive Distillation is Not Robust to Adversarial Examples. *CoRR* abs/1607.04311 (2016).

[10] Nicholas Carlini and David A. Wagner. 2017. Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods. In *AISec@CCS*. ACM, 3–14.

[11] Nicholas Carlini and David A. Wagner. 2017. MagNet and "Efficient Defenses Against Adversarial Attacks" Are Not Robust to Adversarial Examples. *CoRR* abs/1711.08478 (2017).

[12] Nicholas Carlini and David A. Wagner. 2017. Towards Evaluating the Robustness of Neural Networks. In *S&P*. IEEE Computer Society, 39–57.

[13] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. 2017. ZOO: Zeroth Order Optimization Based Black-box Attacks to Deep Neural Networks Without Training Substitute Models. In *AISec@CCS*. ACM, 15–26.

[14] Lori A. Clarke. 1976. A System to Generate Test Data and Symbolically Execute Programs. *TSE* 2 (1976), 215–222. Issue 3.

[15] Cory Cornelius. 2019. The Efficacy of SHIELD under Different Threat Models. *CoRR* abs/1902.00541 (2019).

[16] Patrick Cousot and Radhia Cousot. 1977. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *POPL*. ACM, 238–252.

[17] Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. 2018. Output Range Analysis for Deep Feedforward Neural Networks. In *NFM (LNCS)*, Vol. 10811. Springer, 121–138.

[18] Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy A. Mann, and Pushmeet Kohli. 2018. A Dual Approach to Scalable Verification of Deep Networks. In *UAI*. AUAI Press, 550–559.

[19] Rüdiger Ehlers. 2017. Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks. In *ATVA (LNCS)*, Vol. 10482. Springer, 269–286.

[20] Logan Engstrom, Andrew Ilyas, and Anish Athalye. 2018. Evaluating and Understanding the Robustness of Adversarial Logit Pairing. *CoRR* abs/1807.10272 (2018).

[21] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. 2018. AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation. In *S&P*. IEEE Computer Society, 3–18.

[22] Patrice Godefroid, Nils Klarlund, and Koushik Sen. 2005. DART: Directed Automated Random Testing. In *PLDI*. ACM, 213–223.

[23] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press.

[24] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In *ICLR*.

[25] Divya Gopinath, Guy Katz, Corina S. Pasareanu, and Clark W. Barrett. 2018. DeepSafe: A Data-Driven Approach for Assessing Robustness of Neural Networks. In *ATVA (LNCS)*, Vol. 11138. Springer, 3–19.

[26] Divya Gopinath, Kaiyuan Wang, Mengshi Zhang, Corina S. Pasareanu, and Sarfraz Khurshid. 2018. Symbolic Execution for Deep Neural Networks. *CoRR* abs/1807.10439 (2018).

[27] Jianmin Guo, Yu Jiang, Yue Zhao, Quan Chen, and Jiaguang Sun. 2018. DLFuzz: Differential Fuzzing Testing of Deep Learning Systems. In *ESEC/FSE*. ACM, 739–743.

[28] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *CVPR*. IEEE Computer Society, 770–778.

[29] Warren He, James Wei, Xinyun Chen, Nicholas Carlini, and Dawn Song. 2017. Adversarial Example Defense: Ensembles of Weak Defenses Are Not Strong. In *WOOT*. USENIX.

[30] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara Sainath, and Brian Kingsbury. 2012. Deep Neural Networks for Acoustic Modeling in Speech Recognition. *Signal Process. Mag.* 29 (2012), 82–97. Issue 6.

[31] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. 2017. Safety Verification of Deep Neural Networks. In *CAV (LNCS)*, Vol. 10426. Springer, 3–29.

[32] Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. 2018. Black-Box Adversarial Attacks with Limited Queries and Information. In *ICML (PMLR)*, Vol. 80. PMLR, 2142–2151.

[33] Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. 2017. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *CAV (LNCS)*, Vol. 10426. Springer, 97–117.

[34] Guy Katz, Derek A. Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljic, David L. Dill, Mykel J. Kochenderfer, and Clark W. Barrett. 2019. The Marabou Framework for Verification and Analysis of Deep Neural Networks. In *CAV (LNCS)*, Vol. 11561. Springer, 443–452.

[35] Jinhan Kim, Robert Feldt, and Shin Yoo. 2019. Guiding Deep Learning System Testing Using Surprise Adequacy. In *ICSE*. IEEE Computer Society/ACM, 1039–1049.

[36] James C. King. 1976. Symbolic Execution and Program Testing. *CACM* 19 (1976), 385–394. Issue 7.

[37] Alex Krizhevsky. 2009. *Learning Multiple Layers of Features from Tiny Images*. Technical Report. University of Toronto.

[38] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. ImageNet Classification with Deep Convolutional Neural Networks. *CACM* 60 (2017), 84–90. Issue 6.

[39] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. 2017. Adversarial Examples in the Physical World. In *ICLR*. OpenReview.net.

[40] Peter D. Lax and Maria Shea Terrell. 2014. *Calculus with Applications*. Springer.

[41] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-Based Learning Applied to Document Recognition. In *Proc. IEEE*. IEEE Computer Society, 2278–2324.

[42] Yann LeCun and Corinna Cortes. 1998. The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist

[43] Zenan Li, Xiaoxing Ma, Chang Xu, and Chun Cao. 2019. Structural Coverage Criteria for Neural Networks Could Be Misleading. In *ICSE (NIER)*. IEEE Computer Society/ACM, 89–92.

[44] Changliu Liu, Tomer Arnon, Christopher Lazarus, Clark W. Barrett, and Mykel J. Kochenderfer. 2019. Algorithms for Verifying Deep Neural Networks. *CoRR* abs/1903.06758 (2019).

[45] Alessio Lomuscio and Lalit Maganti. 2017. An Approach to Reachability Analysis for Feed-Forward ReLU Neural Networks. *CoRR* abs/1706.07351 (2017).

[46] Pei-Hsuan Lu, Pin-Yu Chen, Kang-Cheng Chen, and Chia-Mu Yu. 2018. On the Limitation of MagNet Defense Against L1-Based Adversarial Examples. In *DSN Workshops*. IEEE Computer Society, 200–214.

[47] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, Jianjun Zhao, and Yadong Wang. 2018. DeepGauge: Multi-Granularity Testing Criteria for Deep Learning Systems. In *ASE*. ACM, 120–131.

[48] Lei Ma, Fuyuan Zhang, Minhui Xue, Bo Li, Yang Liu, Jianjun Zhao, and Yadong Wang. 2018. Combinatorial Testing for Deep Learning Systems. *CoRR* abs/1806.07723 (2018).

[49] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards Deep Learning Models Resistant to Adversarial Attacks. In *ICLR*. OpenReview.net.

[50] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. 2016. DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks. In *CVPR*. IEEE Computer Society, 2574–2582.

[51] Nina Narodytska and Shiva Prasad Kasiviswanathan. 2017. Simple Black-Box Adversarial Attacks on Deep Neural Networks. In *CVPR Workshops*. IEEE Computer Society, 1310–1318.

[52] Augustus Odena, Catherine Olsson, David Andersen, and Ian J. Goodfellow. 2019. TensorFuzz: Debugging Neural Networks with Coverage-Guided Fuzzing. In *ICML (PMLR)*, Vol. 97. PMLR, 4901–4911.

[53] Nicolas Papernot, Patrick D. McDaniel, and Ian J. Goodfellow. 2016. Transferability in Machine Learning: From Phenomena to Black-Box Attacks Using Adversarial Samples. *CoRR* abs/1605.07277 (2016).

[54] Nicolas Papernot, Patrick D. McDaniel, Ian J. Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. 2017. Practical Black-Box Attacks Against Machine Learning. In *AsiaCCS*. ACM, 506–519.

[55] Nicolas Papernot, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. 2016. The Limitations of Deep Learning in Adversarial Settings. In *EuroS&P*. IEEE Computer Society, 372–387.

[56] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. DeepXplore: Automated Whitebox Testing of Deep Learning Systems. In *SOSP*. ACM, 1–18.

[57] Luca Pulina and Armando Tacchella. 2010. An Abstraction-Refinement Approach to Verification of Artificial Neural Networks. In *CAV (LNCS)*, Vol. 6174. Springer, 243–257.
[58] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. 2018. Certified Defenses Against Adversarial Examples. In *ICLR*. OpenReview.net.
[59] Tim Salimans, Jonathan Ho, Xi Chen, and Ilya Sutskever. 2017. Evolution Strategies as a Scalable Alternative to Reinforcement Learning. *CoRR* abs/1703.03864 (2017).
[60] Jasmine Sekhon and Cody Fleming. 2019. Towards Improved Testing for Deep Learning. In *ICSE (NIER)*. IEEE Computer Society/ACM, 85–88.
[61] Koushik Sen, Darko Marinov, and Gul Agha. 2005. CUTE: A Concolic Unit Testing Engine for C. In *ESEC/FSE*. ACM, 263–272.
[62] Yash Sharma and Pin-Yu Chen. 2018. Bypassing Feature Squeezing by Increasing Adversary Strength. *CoRR* abs/1803.09868 (2018).
[63] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin T. Vechev. 2019. An Abstract Domain for Certifying Neural Networks. *PACMPL* 3 (2019), 41:1–41:30. Issue POPL.
[64] James C. Spall. 1992. Multivariate Stochastic Approximation Using a Simultaneous Perturbation Gradient Approximation. *TAC* 37 (1992), 332–341. Issue 3.
[65] James C. Spall. 2003. *Introduction to Stochastic Search and Optimization*. John Wiley and Sons.
[66] Youcheng Sun, Xiaowei Huang, and Daniel Kroening. 2018. Testing Deep Neural Networks. *CoRR* abs/1803.04792 (2018).
[67] Youcheng Sun, Min Wu, Wenjie Ruan, Xiaowei Huang, Marta Kwiatkowska, and Daniel Kroening. 2018. Concolic Testing for Deep Neural Networks. In *ASE*. ACM, 109–119.
[68] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. 2014. Intriguing Properties of Neural Networks. In *ICLR*.
[69] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. DeepTest: Automated Testing of Deep-Neural-Network-Driven Autonomous Cars. In *ICSE*. ACM, 303–314.
[70] Vincent Tjeng, Kai Y. Xiao, and Russ Tedrake. 2019. Evaluating Robustness of Neural Networks with Mixed Integer Programming. In *ICLR*. OpenReview.net.
[71] Jonathan Uesato, Brendan O'Donoghue, Pushmeet Kohli, and Aäron van den Oord. 2018. Adversarial Risk and the Dangers of Evaluating Against Weak Attacks. In *ICML (PMLR)*, Vol. 80. PMLR, 5032–5041.
[72] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. 2018. Formal Security Analysis of Neural Networks Using Symbolic Intervals. In *Security*. USENIX, 1599–1614.
[73] Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Luca Daniel, Duane S. Boning, and Inderjit S. Dhillon. 2018. Towards Fast Computation of Certified Robustness for ReLU Networks. In *ICML (PMLR)*, Vol. 80. PMLR, 5273–5282.
[74] Matthew Wicker, Xiaowei Huang, and Marta Kwiatkowska. 2018. Feature-Guided Black-Box Safety Testing of Deep Neural Networks. In *TACAS (LNCS)*, Vol. 10805. Springer, 408–426.
[75] Eric Wong and J. Zico Kolter. 2018. Provable Defenses Against Adversarial Examples via the Convex Outer Adversarial Polytope. In *ICML (PMLR)*, Vol. 80. PMLR, 5283–5292.
[76] Weiming Xiang, Hoang-Dung Tran, and Taylor T. Johnson. 2017. Reachable Set Computation and Safety Verification for Neural Networks with ReLU Activations. *CoRR* abs/1712.08163 (2017).
[77] Weiming Xiang, Hoang-Dung Tran, and Taylor T. Johnson. 2018. Output Reachable Set Estimation and Verification for Multilayer Neural Networks. *TNNLS* 29 (2018), 5777–5783. Issue 11.
[78] Xiaofei Xie, Lei Ma, Felix Juefei-Xu, Minhui Xue, Hongxu Chen, Yang Liu, Jianjun Zhao, Bo Li, Jianxiong Yin, and Simon See. 2019. DeepHunter: A Coverage-Guided Fuzz Testing Framework for Deep Neural Networks. In *ISSTA*. ACM, 146–157.
[79] Jie M. Zhang, Mark Harman, Lei Ma, and Yang Liu. 2019. Machine Learning Testing: Survey, Landscapes and Horizons. *CoRR* abs/1906.10742 (2019).
[80] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. 2018. DeepRoad: GAN-Based Metamorphic Testing and Input Validation Framework for Autonomous Driving Systems. In *ASE*. ACM, 132–142.