Refutation-based Adversarial Robustness Verification of Deep Neural Networks

Joshua Smith[1], Jarom Allen[1], Viswanathan Swaminathan[2], and Zhen Zhang[1]

[1] Utah State University
[joshua.smith,jarom.allen,zhen.zhang]@usu.edu
[2] Adobe Research
vishy@adobe.com

Abstract. Adversarial robustness is a desirable property of neural networks, defined as having consistent and correct output for a region surrounding a known input. The problem of proving the adversarial robustness property is extremely challenging due to high dimensionality of the input spaces and large and complex modern neural networks. We present a highly versatile refutation-based adversarial robustness verification framework that performs abstraction and partitioning of discretized hyperrectangular regions in the input space of classification networks. A novel technique is presented for reliable adversarial example generation with improved variance. Coupled with a gradient-based dimension ranking heuristic, it enables the framework to place priority on refuting the adversarial robustness property to efficiently eliminate unsafe regions due to discovered adversarial examples and provide incremental feedback for retraining and analysis purposes. This work evaluates the efficiency of several dimension ranking heuristics on adversarial example generation and partitioning strategies of the proposed framework. We also analyze the utility of the generated set of adversarial examples in retraining to improve robustness across multiple network architectures.

1 Introduction

DNNs are compositions of functions trained with representative data to separate complex manifolds in very high dimensional space into distinct regions. Modern DNNs and linear classifiers are susceptible to adversarial input perturbations [1, 3, 8]. Adversarial perturbations are small changes to an input that result in large and/or unexpected changes of the output. With increased deployment of DNNs in safety-critical systems, it is critical to verify their adversarial robustness properties to avoid undesirable and even catastrophic behaviors.

Proving adversarial robustness of classification neural networks is challenging due to high dimensionality of both the input space and the networks themselves. It has been shown, however, that refuting the *absence* of adversarial examples claim is relatively efficient through well-known techniques [2–4, 6, 10]. Leveraging this observation, determining a region in the input space to be unsafe can be made more efficient by first generating and testing potential adversarial examples and then falling back to robust verification when adversarial generation fails. Iteratively applying this process to partitions of a region results in the following positive outcomes: a covering set of adversarial examples that can be used in retraining to increase adversarial robustness; safe regions when small enough to formally verify; and unsafe regions used for dataset augmentation. We propose a framework that implements this idea while providing for easily interchangeable strategies for abstraction, partitioning, and verification. Evaluation is performed on several proposed abstraction strategies, including a novel improved *randomized fast gradient sign method* (RFGSM), that increases verification region coverage by generating adversarial examples with greater variance. These examples are experimentally shown to significantly increase a network's local robustness through adversarial retraining. In fact, a network retrained using this set reported a 99.95% reduction in number of adversarial examples found. We also found that networks of simple architectures shared properties with respect to the generated set of adversarial examples that could be exploited in the training process. We evaluate the effect of different partitioning strategies of the framework on three different datasets and associated deep neural networks: MNIST, GTSRB, and CIFAR-10.

2 Related Work

Satisfiability Modulo Theory (SMT) solving has been adopted for DNN safety verification. Reluplex [5] implements an optimized SMT solver for networks with ReLU activations. DLV [4] models adversarial input perturbations by repeatedly applying discrete changes to a given input. Based on a dimension ranking heuristic, it limits the number of dimensions modified in the initial search for adversarial examples. Smith *et al.* [7] improve this dimension ranking heuristic by leveraging the training data to guide the heuristic, leading to faster adversarial example detection. These tools require that the neural networks have certain properties, such as ReLU or simple piecewise linear activations. Our proposed method does not place restrictions on the network architecture or operations other than those required by stochastic gradient descent.

We have found that adversarial examples are easily generated and well distributed throughout the input space. Many verification methods report either the absence of or the first found adversarial example. With an extremely low likelihood of a region free of adversarial examples, the common result is a *single* counterexample. *The proposed framework refutes local robustness by generating as many adversarial examples as possible.* FGSM [3] produces adversarial examples with the gradient of the loss function obtained from network training. R+FGSM [9] first randomly manipulates an input to escape a region of sharp gradients before performing FGSM. Both are constant time algorithms with high success rates (around 80%) but suffer from a low output variance when applied iteratively. Their adversarial examples lie along the direction dictated by the gradient of the loss function. *This work presents an improved FGSM to maintain its success rate but increase the variance of repeatedly generated adversarial examples. It significantly reduces the need for expensive formal verification techniques* *by rapidly refuting local robustness. The resulting set of covering adversarial examples were shown to increase its robustness through adversarial retraining.*

3 Preliminaries

A *region* η in the input space is a high-dimensional polytope. For a point $\vec{a} \in \mathbb{R}^n$ in the input space, assume its correct classification is known. The function f of a neural network transforms the input into a classification output y. Denote $\eta_{\vec{a}}$ as the region containing $\vec{a}$ over which the robustness property is to be verified.

Definition 1 (Adversarial example). *Let f be a classification neural network and $\vec{a}$ be a point in the input space clearly pertaining to class y_a, $\vec{a}$ is an adversarial example iff $f(\vec{a}) \neq y_a$.*

Definition 2 (Local adversarial robustness). *For a point $\vec{a} \in \mathbb{R}^n$ in the input space with correct classification $f(\vec{a})$ and a surrounding region $\eta_{\vec{a}}$ containing $\vec{a}$, an arbitrary point $\vec{x} \in \mathbb{R}^n$ within $\eta_{\vec{a}}$ always results in the same output as $\vec{a}$: $\forall \vec{x} \in \eta_{\vec{a}}: \quad f(\vec{x}) = f(\vec{a})$.*

Definition 3 (Domain Range). *The* domain range *is an upper and lower bound on each input dimension. This range enforces that the search space is limited to values in that domain. In the case of eight-bit images, the domain range for each dimension should be* $[0, 255]$.

Regions and sub-regions described in this paper have bounded range in each dimension. When data is converted to a digital format, it inherits the precision features of the storage medium. In the case of eight-bit images, the original data may be continuous but the storage medium forces the data into a discrete range. As almost all implementations of neural networks operate in a digital environment, it is reasonable to assume the existence of a nonzero granularity vector that enforces the discreteness of the input space.

Definition 4 (Granularity). *Given an input dimension i and its discrete domain range $D_i = [0, m]$, where $(m \in \mathbb{N})$, granularity g_i is the minimal distance between any two valid discrete values within D_i.*

Definition 5 (Valid point). *A point $\vec{a} \in \mathbb{N}^n$ in the discrete input space is valid if for every dimension i of $\vec{a} = [a_1, \ldots, a_i, \ldots, a_n]$, it satisfies the following conditions: the distance between a_i and x_i is an integer multiple of granularity: $|a_i - x_i| = k \cdot g_i$, where $a_i \neq x_i$, x_i is a valid discrete value, and $k \in \mathbb{N}$.*

An abstraction strategy maps a region to a finite set of valid testable points. Verification of an abstraction then boils down to testing these representative points to determine its safety.

Definition 6 (Safe and unsafe regions). *Given a valid point $\vec{a}$, its correct output classification $f(\vec{a})$, and a surrounding region $\eta_{\vec{a}}$, if there exists another valid point $\vec{x} \in \eta_{\vec{a}}$, such that $f(\vec{x}) \neq f(\vec{a})$, then $\eta_{\vec{a}}$ is unsafe. If for all valid points $\vec{x} \in \eta_{\vec{a}}$, $f(\vec{x}) = f(\vec{a})$, then $\eta_{\vec{a}}$ is safe. A region is described as unknown when it has not been proven to be either safe or unsafe.*

Definition 7 (Fast Gradient Sign Method (FGSM) [3]). *FGSM generates an adversarial input* $\vec{x_{adv}} \in \mathbb{R}^n$ *from input* $\vec{x} \in \mathbb{R}^n$ *using the following equation:* $\vec{x_{adv}} = \vec{x} + \epsilon sign(\nabla_{\vec{x}} J(\theta, \vec{x}, y))$. *The gradient of the neural network's cost function (J) with respect to the input* ($\nabla_{\vec{x}}$) *determines how to modify* $\vec{x}$ *toward a class change (increase cost function). A cost function is used in a network's training and evaluation to compare the its output with the correct or target value.* θ *is the network's internal parameters and* y *is the associated label for input* $\vec{x}$.

4 Framework Description

The framework operates on three basic principles: verification, partitioning, and abstraction. During verification, a region is found to be safe (free of adversarial examples), unsafe (contains at least one adversarial example), or unknown (cannot be verified rapidly). The verification strategy should scale in algorithmic complexity directly proportional to the size of the region and should determine if a solution (i.e., safe or unsafe) can be found in a reasonable amount of time. If the verification strategy cannot determine a solution quickly on the current region size, it returns unknown. Because verification can be quite expensive, the framework employs a novel abstraction technique to efficiently refute the property if possible before reverting to verification.

When a region is too large to be verified, it is partitioned into smaller subregions to allow for efficient verification. Verification of a region is first performed on representative testable points determined by a given abstraction strategy. On discovering an unsafe region, it waits to be partitioned in subsequent iterations to pinpoint the exact unsafe subregion(s). Abstraction, specifically the novel RFGSM technique presented in this paper, achieves an optimization that reduces the frequency of invoking the expensive verification process by rapidly eliminating unsafe regions through adversarial example generation.

Initialization and Main Procedure. We first define the initial hyperrectangular region I, domain range, granularity, and initial activation. A valid point must lie inside the domain range and obey the granularity defined by the application and the initial activation. The initial activation is the point around which safety is tested and is, by declaration, a valid point. A point is unsafe if the classification of the initial activation and that of the point disagree. Once configuration has completed, the initial region I is placed into the set of unverified regions P and the framework begins execution.

Starting with unverified regions in P, a region r is removed from either P or U (line 4). If no valid points exist in r or the only valid point in r is an adversarial example, the region is discarded and execution continues from the top-most loop (line 2). If the region r has not been verified, it is sent to the verification strategy, *verify*, which can return unknown, unsafe, or safe (line 7). In the safe case (line 8), r is placed in the safe region set S and execution then returns to the top-most loop. For an unsafe region (line 10), it returns a counterexample $\vec{a}$. Region r is then partitioned into a set of regions $R = \{r_1, \ldots, r_n\}$, where n is a tunable parameter of the partitioning strategy representing the number of subregions to generate. The region r_i containing $\vec{a}$ is removed from R (line 12) and placed into U, along with $\vec{a}$, also stored in set E as a record of found adversarial examples. If the outcome is *unknown* (line 14), the region is too large to be verified. A verification strategy compatible with this framework must be able to estimate the amount of work needed to formally verify the robustness property. The current verification strategy does this by calculating the number of valid points (Definition 5) in the region and comparing it to an experimentally chosen threshold. For a region too expensive to verify, it is partitioned into a set of subregions $R = \{r_1, \ldots, r_n\}$. Each subregion r_i is then abstracted to a set of testable points $\{\vec{a_1}, \ldots, \vec{a_m}\}$ to be checked for misclassification. On discovering an unsafe abstraction point $\vec{a_j}$, its enclosing region r_i is removed from R and placed in U (line 21) along with the counterexample $\vec{a_j}$. No action is taken on safe abstraction points. After all subregions in R have been abstracted and tested, all remaining regions are added to P to be verified in subsequent iterations. Any adversarial example found is stored in E. If at any time the program receives a user interrupt, an incremental report is produced.

Dimension Ranking. In both the abstraction and partitioning steps, a heuristic ranking of dimensions can improve on base performance. We propose the following four methods for performing this operation: random ordering, largest-first, Intellifeatures [7], and gradient-based ordering. Random ordering is used as a control to measure the baseline performance of a dimension ranking partitioning strategy. Largest-first selects the dimensions with the largest ranges and is valuable in maintaining close relative range magnitudes during partitioning. Intellifeatures uses knowledge about the classification regions represented in the training dataset to select dimensions that predominantly define the difference between the current point and the next closest distinct classification region. The gradient-based approach uses the magnitude of the gradient of the cost function with respect to the current input to order dimensions. Algorithm 2 describes the gradient-based method. These strategies are compared in Section 5 by their effect on the performance of both abstraction and partitioning.

Abstraction Strategies. An ideal abstraction strategy should limit calls to the expensive verification procedure by refuting adversarial robustness. It maps a region to a set of points that are also likely adversarial examples. Implemented in the framework are three different abstraction strategies: central point, random points, and RFGSM.

Adversarial example generation algorithms have many of the same traits as the ideal abstraction strategy. They can accurately generate adversarial examples with high probability. FGSM is a constant time algorithm but has limited output variance when generating multiple adversarial examples from the same base point $\vec{x}$. When $\vec{x}$ is constant across multiple invocations of FGSM, the only source of variance in the output comes from the parameter ϵ. The generated adversarial examples are then only scaled along the direction defined by the sign of the gradient and *cover* only a tiny portion of the abstracted region. For small regions relative to the granularity, FGSM can only generate few valid abstractions.

Algorithm 1: Abstraction Partitioning Algorithm

```
Input: $I$: Initial region
Output: Incremental Report $(|P|, E, U, |U|, S, |S|)$
1  $P = \{I\}$; $U = \emptyset$;$E = \emptyset$; $S = \emptyset$
2  while $(P \neq \emptyset \vee U \neq \emptyset) \wedge \neg SIGINT$ do
3      if $P \neq \emptyset$ then                                ▷ First iterate over regions in $P$
4          Choose region $r \in P$;  $P = P - \{r\}$
5          if $|r| = 0$ then
6              continue                                        ▷ Region is empty - loop again
7          $\{ver\_result, \vec{a}\}$ = verify($r$)
8          if $ver\_result$ = SAFE then
9              $S = S \cup \{r\}$                              ▷ Add to the set of safe regions
10         else if $ver\_result$ = UNSAFE then
11             Partition $r$ into subregions: $R = \{r_1, \dots, r_n\}$
12             Remove the region $r_i$ from $R$ s.t. $\vec{a} \in r_i$: $R = R - \{r_i\}$
13             $P = P \cup R$; $U = U \cup \{(r_i, \vec{a})\}$; $E = E \cup \{\vec{a}\}$
14         else if $ver\_result$ = UNKNOWN then                ▷ Region is too large.
15             Partition $r$ into $R = \{r_1, \dots, r_n\}$
16             for $i = 1; i \leqslant n; i = i + 1$ do
17                 Abstract $r_i$ to a set of testable points $\{\vec{a_1}, \dots, \vec{a_m}\}$ for
                    $j = 1; j \leqslant m; j = j + 1$ do
18                     if test_point_safety($\vec{a_j}$)=true then
19                         continue                            ▷ Check the next testable point
20                     $U = U \cup \{(r_i, \vec{a_j})\}$;$E = E \cup \{\vec{a_j}\}$
21                     $R = R - \{r_i\}$  ▷ Ensure unsafe region is not appended to $P$
22                     break                     ▷ Region $r_i$ is unsafe. Move on to $r_{i+1}$.
23             $P = P \cup R$     ▷ Add unknown regions back to P to be reevaluated.
24     else if $U \neq \emptyset$ then
25         Remove an element $(r, \vec{a})$ from $U$: $U = U - \{(r, a)\}$
26         if $|r| \leq 1$ then
27             continue                   ▷ $\vec{a} \in E$, and $r$ cannot be further subdivided
28         Partition $r$ into $R = \{r_1, \dots, r_n\}$
29         Remove the region $r_i$ from $R$ s.t. $\vec{a} \in r_i$: $R = R - \{r_i\}$
30         $P = P \cup R$; $U = U \cup \{(r_i, \vec{a})\}$
```

In this work, we propose an improved adversarial example generation algorithm, RFGSM, where dimensions are selected according to some heuristic to either follow FGSM or vary randomly. It is a promising abstraction strategy due to its high success rate, constant time complexity, and high output variance. Equation 1 shows the definition of RFGSM and the steps taken to generate a single adversarial example. Point $\vec{x} \in \mathbb{R}^n$ is in the input space. $M \in \{0, 1\}^n$ is a binary mask used to select dimensions to be manipulated by FGSM or random values in $R \in \{-1, 1\}^n$. Dimension ranking heuristics set M such that dimensions contributing most to the classification output of the neural network are varied by FGSM. Dimensions deemed to be less critical by the dimension ranking heuristic are permitted to vary randomly.

Algorithm 2: Gradient-based Ranking Heuristic

```
Input: $r$: Region under examination
       $g(x, y)$: Gradient of cost function with respect to input $x$
       $y$: Label/ground truth of desired class of verification region
       $n$: Number of dimensions in $r$ and $g$
Output: Dims: Ordered set of dimension indices
1 $c$ = centralPoint($r$); gradient = $g(c, y)$; $t$ = [];
2 for $j = 1;\ j \leq n;\ j = j + 1$ do
3     $t$.append((|gradient[$j$]|, $j$))        ▷ Pair (magnitude of gradient, dim index)
4 Sort $t$ in descending order by the |gradient[$j$]| element
5 Dims = [$t[1][2], t[2][2], \ldots, t[n][2]$];        ▷ List of indices in order by heuristic
6 return Dims
```

When dimension ranges become smaller than the granularity of that dimension, RFGSM tends to produce adversarial examples that lie outside the region. These counterexamples are still reported but no longer function as abstractions of that region. In these cases, the RFGSM algorithm falls back to an abstraction strategy such as random or central point. The success of RFGSM relies on solving the following major issues: adversarial examples produced must be valid according to the granularity and domain range of the application; ineffective FGSM as partitioned subregions become smaller; and choice of dimension ranking heuristics to best maintain success rate while increasing output variance.

Partitioning Strategies. Extremely aggressive partitioning strategies can rapidly increase the number of unverified regions and, thereby, the memory requirements in running the framework. Implemented in the framework is a dimension ranking partitioning strategy that is not aggressive and attempts to greedily select dimensions to subdivide based on a dimension ranking heuristic.

Verification Strategies. The framework is built to augment and optimize verification strategies by adding to them the partitioning and abstraction techniques mentioned above. In order to best take advantage of the benefits provided by the framework, the verification strategy should (1) be able to rapidly provide an estimate of the amount of work or time needed to verify a region and (2) scale in algorithmic complexity with region size. These criteria are not essential but do allow the framework to better decide whether to continue with full verification or perform abstraction and partitioning. The default verification technique implemented in the framework is a simple discrete exhaustive search with a user-configurable threshold and granularity that determine whether the *verify* function (Algorithm 1, Line 7) returns *unknown* or performs full verification. Our current efforts to improve the performance of this framework are focused on developing and testing more sophisticated verification strategies.

$$\vec{x_{adv}} = \vec{x} + \vec{\epsilon} \odot [\mathrm{sign}(\nabla_x J(\theta, \vec{x}, y)) \odot M + R \odot (1 - M)] \tag{1}$$

5 Results

We evaluated the performance of the framework using different abstraction and partitioning strategies. First, we show how well different dimension ranking heuristics maintain successful adversarial example generation in RFGSM. Then we demonstrate the utility of generated adversarial examples in adversarial retraining. Results were generated on a machine with an AMD Ryzen Threadripper 12-Core 3.5 GHz Processor and 132 GB memory. All timed tests were run using 10 threads processing regions concurrently. These tests were conducted on networks trained on the following datasets: Modified National Institute of Standards and Technology (*MNIST*), Canadian Institute for Advanced Research 10 (*CIFAR-10*), and German Traffic Sign Recognition Benchmark (*GTSRB*).

RFGSM: Dimension Ranking Evaluation. While all three abstraction strategies mentioned above are implemented in the framework, the results below show RFGSM with a random point fallback. In our experiments, the central point and random point abstraction performed significantly worse than RFGSM, leading us to focus on variants of RFGSM as more viable abstraction strategies.

Figure 1 charts the success rate of RFGSM with different dimension ranking heuristics as the *balance factor* decreases. The balance factor is the ratio of dimensions to be manipulated using FGSM. A balance factor equal to 1 reduces to pure FGSM whereas a 0 value reduces to pure random manipulation. Dimension ranking heuristics that maintain success rate at lower balance factors offer greater output variance. Note the *verification radius* (Figure 1), which is half of each dimension's range of the verification region. A verification radius of 0.1 means that generated adversarial examples may lie in the following region: $\forall i \in I,\ [i-0.1, i+0.1]$ where I is the value of the initial activation (central point of the verification region) and i is the value of a dimension of I.

Across the three different datasets, the gradient-based dimension ranking heuristic maintains highest success rate as balance factor decreases (i.e., increase in randomization). In general, IntelliFeature and random dimension ranking perform similarly. The gradient-based dimension ranking heuristic demonstrates superiority over the others in maximizing output variance while maintaining success rate. Adversarial examples generated using RFGSM with it is more representative of the regions they abstract, because they are not limited to merely the direction defined by the gradient. Note that the tests for Figure 1 were conducted without the framework to better isolate the RFGSM abstraction strategy.

Table 1 shows the incremental results of the entire framework when running with these three abstraction strategies for five minutes. RFGSM with the gradient-based dimension ranking heuristic consistently generates the most adversarial examples, maintains a smaller number of unverified regions, and functions well even at low balance factors and verification radii. The largest-first dimension ranking heuristic was not tested in RFGSM abstraction as the size of the range of a dimension does not have any apparent connection with its contribution to the classification output.

Adversarial Training using Incremental Results. The incremental report produced by the framework includes a covering set of discovered adversarial

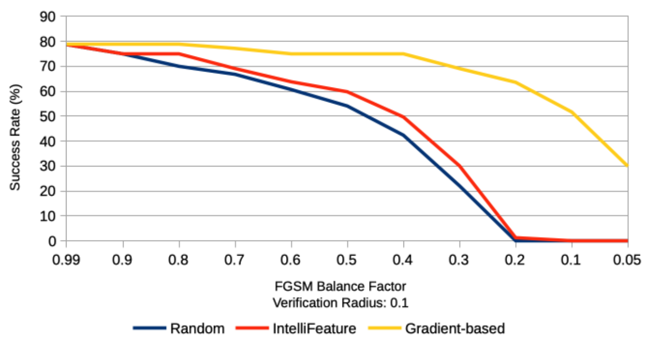

Fig. 1: RFGSM success rate with different dimension ranking heuristics.

Table 1: Results for using three dimension ranking heuristics for RFGSM.

	MNIST			GTSRB			CIFAR-10		
Balance Factor	0.5			0.4			0.3		
Verification Radius	0.3			0.1			0.012		
	$\lvert P\rvert$	$\lvert U\rvert$	$\lvert E\rvert$	$\lvert P\rvert$	$\lvert U\rvert$	$\lvert E\rvert$	$\lvert P\rvert$	$\lvert U\rvert$	$\lvert E\rvert$
Random	81624	48	70	13824	1249	1366	36679	3	12
Gradient-based	80225	538	1077	1	15579	63958	13545	19240	98989
IntelliFeature	81154	71	117	12085	2604	2969	36182	2	5

Table 2: Framework results on networks with differing adversarial training data.

Network	**Adv. Train**	**Train Arch.**	**Unsafe Reg.**	**Safe Reg.**	**Adv. Examples**
no_adv	None	None	8040	16362	31572
fcnn_adv1	FCNN	10x10x10	376	47158	612
fcnn_adv2	FCNN	32x32x32	247	46713	652
cnn_adv	CNN	CNN	53	55351	59

examples and safe and unsafe regions. We experimentally show that adversarial retraining using these discovered adversarial examples significantly increases local adversarial robustness. We evaluate its utility by training MNIST classification networks with adversarial examples found. All experiments were done with a base image of a handwritten digit "1".

We applied adversarial training to the same convolutional network for evaluation, which consists of two convolutional and two max pooling layers followed by two fully-connected layers. Adversarial training consists of a set of 3,000 adversarial examples, generated by our framework, in addition to the standard MNIST dataset. As shown in Table 2, the network no_adv received no adversarial training; adversarial training for fcnn_adv1 and fcnn_adv2 was from a three-layer fully-connected network with 10 neurons per layer and that with 32 neurons per layer, respectively; and cnn_adv received adversarial training from a network with identical architecture. After adversarial training, we ran the framework on these networks for an hour. This table shows significantly reduced adversarial examples and unsafe regions on networks that received adversarial training, leading to significant robustness improvement.

Improved robustness is further demonstrated by the increased intensity and frequency of input perturbations. We calculated the magnitude of the pixel-wise difference between each adversarial example and the base image, and then averaged these differences for each set of adversarial examples as shown in Figure 2. A brighter pixel corresponds to a larger average perturbed magnitude. For no_adv with no adversarial training, the perturbations are centered around the base image, but they are more dispersed across the entire image in networks that received adversarial training. Perturbations in these networks are also lighter and more frequent, indicating that more effort the framework has to perturb the base image in order to introduce adversarial examples.

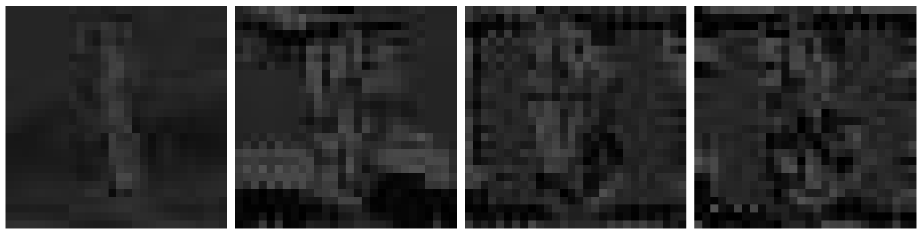

Fig. 2: Average of pixel-wise differences of adversarial examples for networks in Table 2. From left to right, they are no_adv, fcnn_adv1, fcnn_adv2, and cnn_adv.

6 Conclusion

Verifying the adversarial robustness property for deep neural networks and providing ample counterexamples are very valuable in dataset augmentation, retraining for robustness, and safety analysis. The proposed framework utilizes a novel adversarial example generation algorithm and dimension ranking heuristic to rapidly refute the robustness property then partitions the problem to allow for more efficient formal verification on smaller subregions. The result is an incremental report of discovered adversarial examples, associated known unsafe regions, and verified safe regions. This report and the covering set of adversarial examples can be used to significantly increase robustness through adversarial retraining even on networks with differing architectures.

7 Acknowledgement

The authors would like to thank Adobe Systems Incorporated for their support and feedback on this research work.

References

1. Amodei, D., Olah, C., Steinhardt, J., Christiano, P.F., Schulman, J., Mané, D.: Concrete problems in AI safety. CoRR **abs/1606.06565** (2016), `http://arxiv.org/abs/1606.06565`
2. Dunn, I., Melham, T., Kroening, D.: Generating realistic unrestricted adversarial inputs using dual-objective GAN training. CoRR **abs/1905.02463** (2019), `http://arxiv.org/abs/1905.02463`
3. Goodfellow, I., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. In: International Conference on Learning Representations (2015), `http://arxiv.org/abs/1412.6572`
4. Huang, X., Kwiatkowska, M., Wang, S., Wu, M.: Safety verification of deep neural networks. In: Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I. pp. 3–29 (2017). https://doi.org/10.1007/978-3-319-63387-9_1, `https://doi.org/10.1007/978-3-319-63387-9_1`
5. Katz, G., Barrett, C.W., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient SMT solver for verifying deep neural networks. In: Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I. pp. 97–117 (2017). https://doi.org/10.1007/978-3-319-63387-9_5, `https://doi.org/10.1007/978-3-319-63387-9_5`
6. Papernot, N., McDaniel, P.D., Jha, S., Fredrikson, M., Celik, Z.B., Swami, A.: The limitations of deep learning in adversarial settings. In: IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016. pp. 372–387 (2016). https://doi.org/10.1109/EuroSP.2016.36, `https://doi.org/10.1109/EuroSP.2016.36`
7. Smith, J., Huang, X., Swaminathan, V., Zhang, Z.: Improving deep neural network verification using specification-guided search. In: 2nd Workshop on Formal Methods for ML-Enabled Autonomous Systems (2019)
8. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks. In: International Conference on Learning Representations (2014), `http://arxiv.org/abs/1312.6199`
9. Tramèr, F., Kurakin, A., Papernot, N., Goodfellow, I., Boneh, D., McDaniel, P.: Ensemble adversarial training: Attacks and defenses (2017)

10. Wicker, M., Huang, X., Kwiatkowska, M.: Feature-guided black-box safety testing of deep neural networks. In: Tools and Algorithms for the Construction and Analysis of Systems - 24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Part I. pp. 408–426 (2018). https://doi.org/10.1007/978-3-319-89960-2_22, https://doi.org/10.1007/978-3-319-89960-2_22