Towards Robust Saliency Maps

Nham Le[*] NV3LE@UWATERLOO.CA
University of Waterloo, Canada
Chuqin Geng[*] CHUQIN.GENG@MAIL.MCGILL.CA
McGill University, Canada
Xujie Si SIX@CS.TORONTO.EDU
University of Toronto, Canada
CIFAR AI Chair, Mila
Arie Gurfinkel ARIE.GURFINKEL@UWATERLOO.CA
University of Waterloo, Canada

Editors: Vu Nguyen and Hsuan-Tien Lin

Abstract

Saliency maps are one of the most popular tools to interpret the operation of a neural network: they compute input features deemed relevant to the final prediction, which are often subsets of pixels that are easily understandable by a human being. However, it is known that relying solely on human assessment to judge a saliency map method can be misleading. In this work, we propose a new neural network verification specification called *saliency-robustness*, which aims to use formal methods to prove a relationship between Vanilla Gradient (VG) – a simple yet surprisingly effective saliency map method – and the network's prediction: given a network, if an input x emits a certain VG saliency map, it is mathematically proven (or disproven) that the network must classify x in a certain way. We then introduce a novel method that combines both Marabou and Crown/LiRPA– two state-of-the-art neural network verifiers, to solve the proposed specification. Experiments on our synthetic dataset and MNIST show that Vanilla Gradient is surprisingly effective as a certification for the predicted output.

1. Introduction

As deep neural networks (DNNs) continue to advance in complexity and impact, the demand for explanation methods and tools to interpret key aspects of these models also grows. The ability to explain how a model operates can be crucial in meeting regulatory requirements (Goodman and Flaxman, 2017) and assisting developers in debugging the model (Koh and Liang, 2017). Among the various explanation methods available, one category that stands out is saliency maps (Simonyan et al., 2014b; Selvaraju et al., 2016; Kim et al., 2017), primarily due to their interpretability. Saliency maps identify input features that are considered relevant to the final prediction, often highlighting specific pixels that can be easily understood by humans. Fig. 1 visualizes an image of a dog and how some different saliency map methods highlight pixels that are deemed important. However, the abundance of different saliency map methods raises a methodological question for practitioners: how does one choose between these numerous options?

*. Main contributors

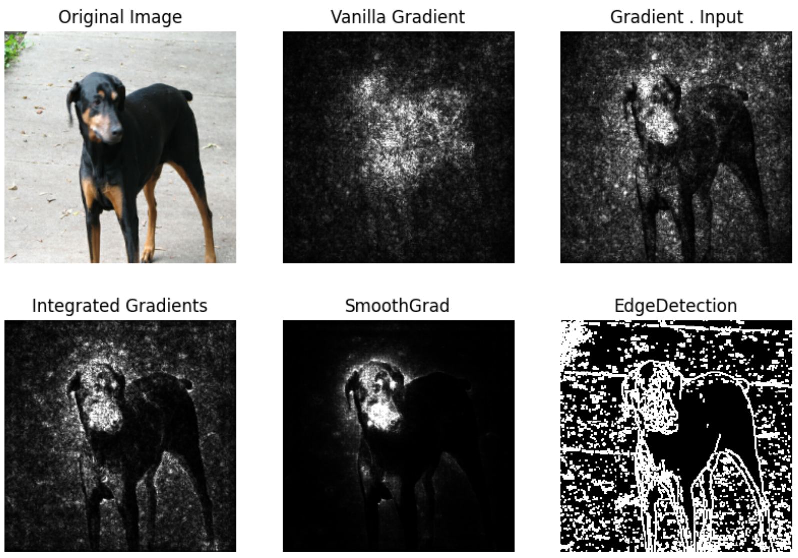

Figure 1: Different saliency map methods of a dog, together with the result of an edge detection algorithm (Canny, 1986) that does not take the model into account at all.

Thus far, the evaluation of most saliency map methods has heavily relied on subjective human judgment. The assessment typically follows the approach of "the saliency map is considered good if it appears visually appealing to me". While it is necessary to discard obviously inadequate methods through a sanity check (Adebayo et al., 2018), users of saliency maps are left to select the appropriate method based on their own visual assessment. As pointed out in (Adebayo et al., 2018), while we want an explanation to take into account *both* the input and the network, human judgment tends to be biased toward the input. This presents a problem as humans may inadvertently focus on explaining the input itself rather than understanding the relationship between the input and the model. The issue is highlighted in Fig. 1, where the Canny edge detector algorithm (Canny, 1986) (which does not consider the DNN at all and should not be used to explain any DNN) produces a visually convincing map that bears a resemblance to those generated by some other saliency methods.

In this paper, we propose to use *formal methods* to mathematically prove or disprove a relationship between a saliency map and the prediction of the network. As the first work in this direction, we apply our method with Vanilla Gradient (VG) (Baehrens et al., 2010) – an elegant yet surprisingly effective formulation of saliency maps: it passes all of the sanity checks proposed by (Adebayo et al., 2018) while several more modern methods (Smilkov et al., 2017; Sundararajan et al., 2017; Shrikumar et al., 2017) do not.

Our key insight is our novel concept of *saliency-robustness*, which states that if the Vanilla Gradient saliency map $E_{\mathcal{N}}$ for the network $\mathcal{N}$ is reliable, then two images generating similar saliency maps should be classified in the same manner by $\mathcal{N}$. This property is essential because it ensures that $E_{\mathcal{N}}$ can genuinely explain why an image belongs to the label "dog" rather than the label "cat". Conversely, if this property does not hold, then $E_{\mathcal{N}}$ may not provide accurate explanations. To the best of our knowledge, this is the first time in which a *mathematically proven* relationship between a saliency map function and a prediction is attempted.

To solve our proposed saliency-robustness property, we make another insight: computing the saliency map of a ReLU-activated neural network can be done by solving a system of linear constraints. Thus, the whole property can be encoded as a set of linear constraints and solved using an off-the-shelf SMT solver. While this is acceptable as a proof of concept, it is widely known that off-the-shelf SMT solvers do not scale to solving neural networks of interesting sizes (Katz et al., 2017). To overcome that challenge, we propose a novel method to solve the saliency-robustness property more effectively, by combining state-of-the-art techniques in neural network verification. To sum up, we make the following contributions:

- We propose a novel safety property for a neural network and its Vanilla Gradient saliency map, called *saliency-robustness*.
- We show that the proposed property can be verified by solving a constraint satisfiability problem over linear real arithmetic (LRA).
- We propose a novel method to solve the saliency-robustness problem more effectively, by combining two state-of-the-art techniques in neural network verification, namely constraint-based solving (Katz et al., 2019) and Jacobian bounding (Zhang et al., 2019; Wang et al., 2021; Shi et al., 2022).
- We conduct experiments on our synthetic benchmarks and dataset and a neural network from VNNCOMP23 (Müller et al., 2023), the annual neural network verification competition. We find that Vanilla Gradient, despite being the earliest form of the saliency maps, is a surprisingly good explanation for the tested network.

The rest of the paper is as follows: Section 2 goes over preliminaries and related work, Section 3 provides a concrete example as well as describes our synthetic dataset, Section 4 goes into details our proposed saliency-robustness property and how to solve it, Section 5 presents our experiments and results, and finally Section 6 summarizes our contributions, outlines the current limitations of the method, and discusses open problems for future work.

2. Background and Related Work

2.1. ReLU activated neural networks for classification tasks

A neural network classifier $\mathcal{N}$ of L layers is a set $\{(\boldsymbol{W}^i, \boldsymbol{b}^i) \mid 1 \leq i \leq L\}$, where $\boldsymbol{W}^i$ and $\boldsymbol{b}^i$ are the weight matrix and the bias for layer i, respectively. A neural network $\mathcal{N}$ defines a function $F_N : \mathbb{R}^{d_0} \to \mathbb{R}^{d_L}$ (in which d_0 and d_L are the input and output dimension, respectively). We define $F_{\mathcal{N}}(x) = z^L(x)$, where

$$h^0(x) = x \tag{1}$$

$$\forall i \in [1, L] \cdot z^i(x) = \boldsymbol{W}^i h^{i-1}(x) + \boldsymbol{b}^i \tag{2}$$

$$\forall i \in [1, L-1] \cdot h^i(x) = \sigma(z^i(x)) \tag{3}$$

in which σ is the activation function. Neurons are indexed linearly by $v_0, v_1, \ldots$. In this work, we focus only on the ReLU activation function, i.e., $\sigma(x) = \max(x, 0)$ element-wise, but the idea and techniques can be generalized for different activation functions and architectures as well. Note that convolutional neural nets (CNNs) also fit into this formulation, since CNN layers are also linear.

We denote the i^{th} element of a vector v as $v[i]$. The prediction vector $F_{\mathcal{N}}(x)[i]$ represents the score or likelihood for the i^{th} label, and the one with the highest score ($\arg\max_i F_{\mathcal{N}}(x)[i]$) is often considered as the predicted label of the network $\mathcal{N}$. We denote this output label as $\mathcal{O}_{\mathcal{N}}(x)$. When the context is clear, we omit the subscript $\mathcal{N}$ and the input x for simplicity.

2.2. Saliency maps

A saliency map function $E_{\mathcal{N}} : \mathbb{R}^{d_0} \to \mathbb{R}^{d_0}$ highlights features in the input x that E deems important to the classification of x by $\mathcal{N}$. In its original form (Baehrens et al., 2010), called *Vanilla Gradient*, the map computes the *gradient explanation* $E_{\mathcal{N}}(x) = \frac{\partial F_{\mathcal{N}}(x)}{\partial x}$, which is a matrix of size $d_0 \times d_L$, quantifying how much a change in each input dimension would change the score of a label in a *small* neighborhood around the input.

Vanilla Gradient suffers from a problem called "gradient saturation" (Shrikumar et al., 2017): when a pre-ReLU value goes below zero, then the activation is capped at zero and does not change anymore. From the Vanilla Gradient's perspective, this value can be -10 or -1, it does not make any difference. To overcome the gradient saturation problem, later saliency map functions propose different formulations for E, some notable are:

- **Gradient $\odot$ Input** (Shrikumar et al., 2017) computes $E_{\mathcal{N}}(x) = x \odot \frac{\partial F_{\mathcal{N}}(x)}{\partial x}$
- **Integrated Gradients** (Sundararajan et al., 2017) sums over scaled versions of the input by computing $E_{\mathcal{N}}(x) = (x - \overline{x}) \times \int_0^1 \frac{\partial F_{\mathcal{N}}(\overline{x}+\alpha(x-\overline{x}))}{\partial x} d\alpha$ in which $\overline{x}$ is a baseline input that omits a feature in the original input x
- **SmoothGrad** (Smilkov et al., 2017) averages Vanilla Gradients of noisy copies of an input, i.e computing $E_{\mathcal{N}}(x) = \frac{1}{N} \sum_i^N \frac{\partial F_{\mathcal{N}}(x+g_i)}{\partial x}$, in which g_i are noise vectors drawn from a normal distribution
- **Grad-CAM** (Selvaraju et al., 2016) computes the gradient with respect to the feature map of the last convolutional layer instead of to the input.

We refer curious readers to (Molnar, 2022) for a more comprehensive survey. In this work, we focus on Vanilla Gradient. While being the earliest form of saliency maps, Vanilla Gradient passes all the sanity checks proposed by (Adebayo et al., 2018) while many later methods do not. We leave extending our method to other saliency maps for future work. From this point on, unless specified otherwise, we use saliency map and Vanilla Gradient interchangeably.

2.3. Adversarial attacks against neural networks and the robustness verification problem

Given a neural network $\mathcal{N}$, the aim of adversarial attacks is to find a perturbation v of a target input $\hat{x}$, such that $\hat{x}$ and $\hat{x} + v$ are "similar" according to some domain knowledge, yet $\mathcal{O}(\hat{x}) \neq \mathcal{O}(\hat{x} + v)$. In this paper, we use the common formulation of "similarity" in the field: two inputs are similar if the L_∞ norm of υ is small. Under this formulation, finding an adversarial example can be defined as solving the following optimization problem:

$$\min ||\upsilon||_\infty \ s.t. \ \mathcal{O}(\hat{x}) \neq \mathcal{O}(\hat{x}+\upsilon) \tag{4}$$

In practice, it is very hard to formally define "similar": should an image and a crop of it be "similar"? Should two sentences differ by one synonym be the same? We refer curious readers to the survey (Xu et al., 2020) for a comprehensive review of different formulations.

One natural defense against adversarial attacks, called *robustness verification*, is to prove that $\min ||\upsilon||_\infty$ must be greater than some user-specified threshold ϵ. Formally, given that $\mathcal{O}(\hat{x}) = \ell$, we verify

$$\forall x' \in B(\hat{x}, \epsilon) \cdot \mathcal{O}(x') = \ell \tag{5}$$

where $B(\hat{x}, \epsilon)$ is a L_∞ norm-ball of radius ϵ centered at x: $B(\hat{x}, \epsilon) = \{x' \mid ||\hat{x} - x'||_\infty \leq \epsilon\}$. If Eq. (5) holds, we say that $\hat{x}$ is ϵ-robust.

2.4. Jacobian bounding

The Vanilla Gradient and the robustness verification problem are connected by the problem of bounding the Jacobian matrix $\frac{\partial F_\mathcal{N}(x)}{\partial x}$: the Vanilla Gradient can be viewed as bounding the Jacobian with $\epsilon = 0$, while in the context of robustness verification, once a local Jacobian bound is computed, one can know the radius of a guaranteed safe perturbation area in the input space (Hein and Andriushchenko, 2017; Weng et al., 2018). Efficiently computing a tight bound for Jacobian (or gradient) is still an open problem for deep neural networks. Sampling-based approaches (Weng et al., 2018) only estimate an under-estimation, and exact methods using MIP solver like (Jordan and Dimakis, 2020) are often too costly to scale to non-trivial networks. Recent advances in computing the bounds for each layer in the forward computation of a neural network have opened a new research direction into the problem: (Shi et al., 2022) shows that by viewing the backward computation as a part of a general computation graph, bounding the Jacobian can be done in the same manner as bounding the forward layers.

2.5. Neural networks verifiers

As we discuss in Section 4, the robustness verification problem can be encoded as a constraint satisfiability problem in linear real arithmetic, thus, in theory, can be solved using any off-the-shelf SMT solver such as Z3 (de Moura and Bjørner, 2008). However, this naive approach doesn't scale beyond tiny networks and researchers have invented specialized tools to verify the robustness of bigger networks perturbed by bigger epsilons. For a more comprehensive survey of existing verification algorithms and tools for neural networks, we refer curious readers to (Albarghouthi, 2021). Here, we briefly survey two major classes of neural network verifiers: constraint-based and abstraction-based.

Constraint-based verifiers SMT solvers usually support multiple theories (e.g. string, bitvector, etc.) as well as a combination of them, while neural network verifiers only need to reason about Quantifier-free Linear Real Arithmetic. Thus, dedicated neural network verifiers can exploit heuristics and architectures that may not be applicable to other theories.

The major solver in this direction is Reluplex (Katz et al., 2017) and its successor Marabou (Katz et al., 2019). Their main insights are that case-splitting of ReLU can be handled lazily, and bound-tightening procedures can help fix a ReLU to either its positive or negative side, thus reducing the number of needed splits.

Abstraction-based verifiers While Constraint-based verifiers such as Marabou can solve the encoded neural network verification precisely, their scalability remains an issue. One approach to scalability is to make the problem easier by abstracting (over-approximating) the semantics of a DNN, in the hope of claiming UNSAT faster in exchange for being imprecise: when the solver answers SAT, the found assignment may not be a valid one. This approach is often known as Abstract Interpretation (Cousot and Cousot, 1977). Some of the most prominent ones are ERAN (Singh et al., 2019, 2018) which uses polyhedrons to approximate ReLU as well as other activation functions, Crown/LiRPA (Wang et al., 2021) which uses lines abstraction, and NNV (Tran et al., 2021) which uses StarSet abstraction.

3. A motivating example

In this section, we provide a concrete example to illustrate our idea. We consider a multi-arm bandit machine with 5 arms, each capable of generating a specific reward by manipulating its complete state. However, unlike digital arms, these arms are analog and can be pulled at varying levels of intensity, ranging from 0% to 100%. For instance, if an arm has a reward value of 300, pulling it at 10% intensity will result in a reward of 30. The rewards for the five arms are as follows: 100, 100, 300, 100, 300.

To obtain the total reward, the player must pull each arm to an arbitrary level. We can represent the machine configuration with a five-element vector. For instance, if the machine has only the first two arms pulled to 50% level, this configuration can be represented by the vector $[0.5, 0.5, 0., 0., 0.]$. The total reward is simply the sum of rewards obtained from each arm. We consider the total reward greater than 300 to be a high reward, and anything less to be a low reward. However, this information is not revealed to the players. Suppose a player records several configurations of complete arm states, such as $[1.0, 0., 0., 1., 0.]$. The player has a dataset of 20 configurations. They can then train a simple FCN model to predict the corresponding reward outcome. By analyzing the saliency map, the player realizes that the third and fifth arms have the highest absolute value of gradients. They can take advantage of this information to improve their strategy. But they have a burning question: is the saliency map truly an explanation for the predicted outcome or a mere correlation?

The user looks at the saliency map of the input $[1, 1, 0, 1, 1]$ with respect to the "high reward" label, which is $M = [0.03, 0.23, 2.97, 0.05, 2.5]$. If the saliency map is an explanation for the prediction, then for all input in $[0, 1]^5$, a saliency map similar to M must imply that the output is classified as "high reward", the user figures. They look at M and see that the gap between the high and low values is about 2.5, thus they expect that for all saliency maps that have L_∞ distance to M of less than 1.25 (i.e saliency maps with the same two arms being highlighted), they should all guarantee the prediction of "high reward". Using our method, they verify that it is indeed the case, as shown by Table 1.

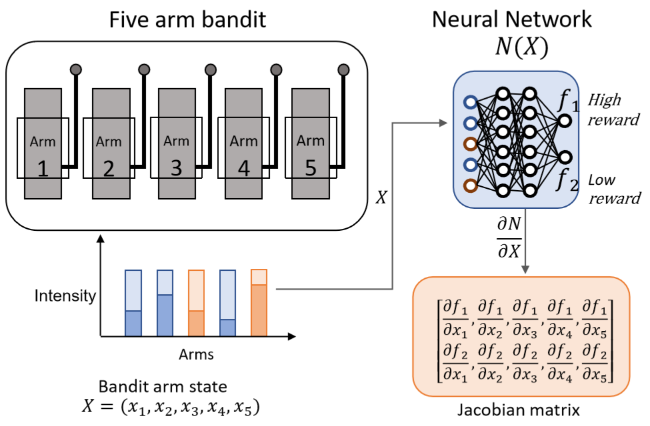

Figure 2: Five arm bandit.

	Z3	Ours
$\delta = 0.5$	3.22s UNSAT	0.8s UNSAT
$\delta = 0.75$	2.8s UNSAT	1.225s UNSAT
$\delta = 1$	2.9s UNSAT	1.258s UNSAT
$\delta = 1.25$	3.36s SAT	2.212s SAT
$\delta = 1.5$	2.78s SAT	0.8s SAT

Table 1: Verifying saliency-robustness for BanditNet

4. Methodology

In this section, we introduce our new verification problem, called *the saliency-robustness problem*, how solving it can be seen as solving a constraint satisfiability problem over LRA, and how to effectively solve it by combining state-of-the-art techniques in neural network verification.

4.1. The saliency-robustness problem

Given the aim of a saliency map E, we ask the question: if $E(x)$ and $E(x')$ are "similar" (according to some metrics or human judgment), must $\mathcal{O}(x) = \mathcal{O}(x')$? If that's not the case, then E is hardly a good explanation: if the same set of pixels are important for both recognizing digit 0 and 1, then that set of features cannot be used to explain why an image is of the label 0 but not 1.

We formalize this question by the following verification problem

$$\forall x, x' \in \mathbb{R}^{d_0} \cdot \mathcal{O}(x) = \ell \land E(x') \approx E(x) \implies \mathcal{O}(x') = \ell \tag{6}$$

in which $E(x) \approx E(x')$ indicate that they are similar. There are many different ways of defining similarity, and we leave exploring different formulations for future work. In this paper, we use the same notion of similarity that is commonly used in robustness verification (Albarghouthi, 2021): two saliency maps are similar if they are close in the L_∞ norm.

The quantifier in Eq. (6) reflects the ideal scenario in which the property can be verified in the whole input domain. In practice, this is rarely the case given the scalability of existing tools. Thus, we aim to solve a ϵ-relaxed problem and aim to push the parameter ϵ higher in future work. Concretely, we verify inputs in the *epsilon* vicinity of datapoints (similar to the robustness problem): given a target input $\hat{x}$, we check the *saliency-robustness* property

$$\forall x' \in B(\hat{x}, \epsilon) \cdot \mathcal{O}(\hat{x}) = \ell \wedge ||E(x') - E(\hat{x})||_\infty \leq \delta \tag{7}$$
$$\implies \mathcal{O}(x') = \ell \tag{8}$$

If Eq. (7) holds, we say that E is (ϵ, δ)-robust at $\hat{x}$. In our motivating example, the user wants to check queries ranging from $(1, 0.5)$- to $(1, 1.5)$-robustness of at $\hat{x} = [1, 1, 0, 1, 1]$.

Note that per our definition in Section 2.2, E is a 2D matrix computing the gradient of each input with respect to each label. In many saliency map methods (Selvaraju et al., 2016; Simonyan et al., 2014a; Kim et al., 2017), it is common to focus on only the gradient with respect to the label with the highest score $(\mathcal{O}(x))$. From this point on, unless specified otherwise, we consider E as a gradient *vector* with respect to the predicted label.

4.2. The saliency-robustness as a constraint satisfiability problem

We show that for any neural network consisting of only linear layers and piecewise-linear activation functions, Eq. (7) can be encoded into a satisfiability problem over linear real arithmetic (LRA). First, given a target input $\hat{x}$ and a neural network $\mathcal{N}$ that predicts $\mathcal{O}_\mathcal{N}(\hat{x}) = \ell$, we define the following first-order logic formula

$$f = \phi_F \wedge \phi_G \wedge \phi_\approx \wedge \phi_P$$

in which

$$\begin{aligned}\phi_F &= \text{Constraints for forward computation} \\ &\quad \text{(encoding } \mathcal{O}_\mathcal{N}(x)) \\ \phi_G &= \text{Constraints for computing gradient} \\ &\quad \text{(encoding } E_\mathcal{N}(x)) \\ \phi_\approx &= \text{Constraints for gradient similarity} \\ \phi_P &= \text{Constraints for } \epsilon\text{-robustness}\end{aligned}$$

The forward computation can be encoded using the same encoding used by Marabou (Katz et al., 2019): for a linear layer $z^i = \boldsymbol{W}^i h^{i-1} + \boldsymbol{b}^i$, we have the constraint $z^i[j] = \sum_{k=1}^{|h_{i-1}|} \boldsymbol{W}^i[j][k] + \boldsymbol{b}^i[j]$ for each entry in the resulting layer z^i; and for a ReLU activation layer $h^i = ReLU(z^i)$, each entry in the result vector can be encoded using two implications: $z^i[j] > 0 \implies h^i[j] = z^i[j]$ and $z^i[j] \leq 0 \implies h^i[j] = 0$.

The backward computation can be encoded recursively as follows. For each layer h^i and z^i, we denote ∂h^i and ∂z^i their gradient vectors. At the last layer z^L, we set $\partial z^L[\ell] = 1$, and $\partial z^L[j \neq \ell] = 0$.

For the linear layer $z^i = \boldsymbol{W}^i h^{i-1} + \boldsymbol{b}^i$, the j^{th} entry in the gradient of ∂h^{i-1} is computed by

$$\partial h^{i-1}[j] = \sum_{k=1}^{|z^i|} \boldsymbol{W}^i[k][j] \partial z^i[k] \tag{9}$$

The backward computation for the convolutional layer (which is a specialized version of the linear layer) can be encoded in a similar manner.

For the ReLU layer $h^i = ReLU(z^i)$, the j^{th} entry in the gradient of ∂z^i is encoded by two implications

$$z^i[j] > 0 \implies \partial z^i[j] = \partial h^i[j] \tag{10}$$
$$z^i[j] \leq 0 \implies \partial z^i[j] = 0 \tag{11}$$

The gradient similarity is encoded as bounds on each entry in the vector ∂h^0. Given the precomputed $E(\hat{x})$ (which can be computed using any off-the-shelf auto-gradient tools like Pytorch or Tensorflow), we set

$$\forall j \in [1, d_0] \cdot E(\hat{x})[j] - \delta \leq \partial h^0[j] \leq E(\hat{x})[j] + \delta \tag{12}$$

The robustness constraints are encoded as bounds on the input and *negation* of conditions on the output:

$$\forall j \in [1, d_0] \cdot \hat{x}[j] - \epsilon \leq x[j] \leq \hat{x}[j] + \epsilon \tag{13}$$
$$\forall j \neq \ell \in [1, d_L] \cdot \bigvee z^L[j] > z^L[\ell] \tag{14}$$

Theorem 1 *If f is UNSAT, then E is (ϵ, δ)-robust at $\hat{x}$*

Its correctness can be easily derived from Eq. (7) and the construction of f.

4.3. Solving the saliency-robustness problem by combining constraint-based NN verifiers with Jacobian bounding methods

In this part, we introduce a first cut to effectively verifying the saliency-robustness problem by combining two state-of-the-art neural network verification techniques – constraint-based neural network verifier and Jacobian bounding.

Given that the saliency-robustness problem can be encoded as a satisfiability problem over LRA, it could be solved by any off-the-shelf SMT solver such as Z3 (de Moura and Bjørner, 2008). However, like the robustness problem, which can also be encoded as a satisfiability problem over LRA, solving the encoding using an off-the-shelf SMT solver hardly scale to any network of interesting size (Katz et al., 2017).

Adapting constraint-based NN verifiers for solving the saliency-robustness problem In this paper, we use Marabou (Katz et al., 2019), a dedicated state-of-the-art constraint-based NN verifier as the core solver. Marabou extends the Simplex (Nelder and Mead, 1965) algorithm used in linear programming with special mechanisms to handle ReLU activation function. Like Simplex, at each iteration, Marabou tries to fix a variable so that it doesn't violate its constraints. If in Simplex, a violation can only happen when a variable becomes out-of-bound, in Marabou a violation can also happen when a variable doesn't satisfy its activation constraints, thus Marabou extends Simplex's pivot rules with a *PivotForRelu* rule and introduces *splitting* into the solving loop. Most importantly, Marabou supports *disjunctions*, thus allowing it to express and solve more complicated verification specifications, compared to other tools like Crown/LiRPA that only verifies the robustness property.

Out of the box, Marabou can solve the satisfiability of $\phi_F \wedge \phi_P$ (which is exactly the robustness property). Since Marabou only supports disjunctions but not implications, and doesn't have strict inequalities, to encode ϕ_G we model Eq. (10) and Eq. (11) using disjunction as follows:

$$z^i[j] \leq 0 \vee \partial z^i[j] = \partial h^i[j] \tag{15}$$

$$z^i[j] \geq 10^{-6} \vee \partial z^i[j] = 0 \tag{16}$$

Note that in Eq. (16) we use a small number to model strict inequality. This is a standard technique and is also recommended by Marabou's developer [1]. Encoding $\phi_{\approx}$ in Marabou is similar to encoding ϕ_P: we simply set the bounds for each of the entries of ∂x.

Precomputing Jacobian bounds It is not enough to encode the saliency-robustness problem into the form that Marabou accepts. The core solving loop of Marabou requires that every variable in the input has to be bounded. Thus, one of the first preprocessing steps in Marabou is to derive bounds for all variables. Unfortunately, while Marabou implements many procedures to derive and tighten bounds during the preprocessing phase, those procedures cannot compute bounds over disjunctions. In practice, that means we must find a way to effectively bound ∂h^i and ∂z^i and set them in Marabou manually.

Bounding Jacobian is a hard and open question (Zhang et al., 2019; Shi et al., 2022; Jordan and Dimakis, 2020), and we do not attempt to solve the problem in this paper. Instead, we use Crown/LiRPA (Shi et al., 2022) – a state-of-the-art recursive algorithm to precompute the Jacobian bounds to use with Marabou. To optimize memory usage, Crown/LiRPA does not maintain the intermediate Jacobian bounds for all layers. We work around this issues by calling Crown/LiRPA L times, each time marking one layer as the last layer in the computation graph, thus allowing us to collect the Jacobian bounds for all L layers in the network.[2]

5. Evaluation

In this section, we evaluate the saliency-robustness of the Vanilla Gradient, as well as compare the performance between Z3 – a state-of-the-art SMT solver, and our proposed method. We

1. https://github.com/NeuralNetworkVerification/Marabou/issues/496
2. This is currently the recommended solution suggested by the developers, see https://github.com/Verified-Intelligence/auto_LiRPA/issues/46

also conduct an experiment showing the relationship between the quality of the Jacobian bounds and the solving performance.

	Z3	Marabou + Crown/LiRPA		
Region	$\delta = 0.0001$	$\delta = 0.0001$	$\delta = 0.0005$	$\delta = 0.001$
$B(x_1, 0.05)$	TIMEOUT	3m46s UNSAT	TIMEOUT	TIMEOUT
$B(x_2, 0.03)$	TIMEOUT	1m34s UNSAT	1m22s UNSAT	5m22s UNSAT
$B(x_2, 0.05)$	TIMEOUT	TIMEOUT	TIMEOUT	TIMEOUT
$B(x_3, 0.03)$	TIMEOUT	ERROR	ERROR	ERROR
$B(x_3, 0.05)$	TIMEOUT	TIMEOUT	TIMEOUT	TIMEOUT
$B(x_4, 0.03)$	TIMEOUT	ERROR	ERROR	ERROR
$B(x_4, 0.05)$	TIMEOUT	1m19s UNSAT	1m28s UNSAT	3m23s UNSAT
$B(x_5, 0.05)$	TIMEOUT	ERROR	ERROR	ERROR
$B(x_6, 0.05)$	TIMEOUT	1m44s UNSAT	TIMEOUT	1m50s UNSAT
$B(x_7, 0.05)$	TIMEOUT	TIMEOUT	TIMEOUT	TIMEOUT

Table 2: Verifying the saliency-robustness property using Z3 and our method at different δs.

5.1. Experiment setup

Our experiments are based on our synthetic dataset for the five-arm bandit problem and benchmark from VNNCOMP23 (Müller et al., 2023) – the annual neural network verification competition. We use the MNIST dataset and the pre-trained model `mnistfc_256x2`, a 2-layers fully connected network with 256 neurons for each layer. Due to the scalability of both Crown/LiRPA in computing Jacobian bounds and Marabou in solving, experiments with CNNs or bigger fully connected networks with bigger δs all result in TIMEOUT [3]. Note that by adding extra variables to represent gradients into Marabou, every network is *double* in size, i.e., a query verifying the saliency-robustness of a 4-layer network has the same number of variables and constraints as verifying the robustness property of an 8-layer network.

Experiments are run on a `c5a.16xlarge` EC2 instances with 64 cores and 124GB of RAM. On all benchmarks and in both Z3 and Marabou+Crown/LiRPA, we allow the solver to use up to 30 cores. The timeout for each query is set to 10 minutes. Unless specified otherwise, we use the Crown-Optimized method in Crown/LiRPA, and set the number of refinement iterations to 200 instead of the default value of 20. We call this the Reference config.

5.2. The saliency-robustness for the five-arm bandits over the whole input domain

We train BanditNet, a 2 layers Fully-Connected Network with 6 neurons each, on our synthetic benchmark. For BanditNet, we verify the saliency-robustness for the whole input domain ($\epsilon = 1$), at different values of δ ranging from 0.5 to 1.5. We run each query using both Z3 and Marabou+Crown/LiRPA, and results are summarized in Table 1. Z3 performs well on

3. Crown/LiRPA needs at least 40GB of GPU memory to bound the Jacobian of a 2-layer CNN network with only 8 channels and kernels of size 3×3

this small network, but even here, we observe a significant difference in performance between our method and Z3, across all δs, for both SAT and UNSAT queries. Interestingly, we also observe that the query becomes hardest near the δ border 1.25.

5.3. The saliency-robustness for `mnistfc_256x2` in known unsafe regions

To verify the usefulness of Vanilla Gradient as an explanation for the prediction of `mnistfc_256x2`, we look at inputs in the benchmarks that are known to have adversarial examples in their vicinity. If checking the saliency-robustness in the same vicinity returns UNSAT, we can claim that the Vanilla Gradient is a useful tool to explain the prediction in that region.

In the VNNCOMP23 benchmarks, Marabou can find adversarial examples in 10 regions centered at 7 inputs at 2 different epsilon values (Table 2). As expected, Z3 does not scale to this network, while our method can verify 4 out of 10 regions at $\delta = 0.0001$. We also observe that as we increase δ, our queries become increasingly harder, resulting in more TIMEOUTs. There is an outlier at region $B(x_6, 0.05)$ in which at $\delta = 0.0005$ the query is timed out but at a harder $\delta = 0.001$ it can be solved again. It is interesting that other than returning TIMEOUT or SAT/UNSAT, we also observe cases where our method crashes the solver. Given the limited amount of time, we do not have a clear idea of the root cause of the crashes and we leave investigating those issues for future work.

5.4. The effect of bound's tightness on performance

	Reference config	No optimization		20 refinement iterations	
Region	Result	Result	avg $\times$	Result	avg $\times$
$B(x_1, 0.05)$	3m46s UNSAT	TIMEOUT	2.131	5m21s UNSAT	1
$B(x_2, 0.03)$	1m34s UNSAT	1m11s UNSAT	1.11	1m6s UNSAT	1.014
$B(x_2, 0.05)$	TIMEOUT	TIMEOUT	1.06	TIMEOUT	1
$B(x_3, 0.03)$	ERROR	ERROR	-	ERROR	-
$B(x_3, 0.05)$	TIMEOUT	TIMEOUT	1.05	TIMEOUT	1.002
$B(x_4, 0.03)$	ERROR	ERROR	1.69	ERROR	1
$B(x_4, 0.05)$	1m19s UNSAT	1m44s UNSAT	1.04	1m12s UNSAT	1
$B(x_5, 0.05)$	ERROR	ERROR	1.13	ERROR	1
$B(x_6, 0.05)$	1m44s UNSAT	TIMEOUT	1.04	1m44s UNSAT	1.004
$B(x_7, 0.05)$	TIMEOUT	TIMEOUT	1.07	TIMEOUT	1.004

Table 3: The effect of Jacobian bounds on solving the saliency-robustness problem. We use $\delta = 0.0001$ for this experiment. We show the average increase in Jacobian bounds in the two "avg $\times$" columns. The "avg $\times$" values for $B(x_3, 0.03)$ are missing since this query crashes Crown/LiRPA, thus no bounds were computed.

As a new area of research, the quality of Jacobian bounds is improved rapidly, and in some cases, newer methods like Crown/LiRPA can produce bounds orders of magnitudes smaller than older methods (Shi et al., 2022). To get an idea of how big of a difference different Jacobian bounds can make on our method, we conduct an experiment in which we turn off all optimizations in Crown/LiRPA, and another experiment in which we keep the same set of optimizations but use the default number of refinement iterations (20) to obtain looser bounds, then set them in Marabou. Table 3 shows the solving result as well as the average increase in the size of the obtained bounds. Without any optimization, we can see that the obtained Jacobian bounds are quite loose (more than 2 times bigger compared with the optimized bounds), resulting in more TIMEOUTs. In general, running Crown-Optimized for 20 iterations gives us relatively tight bounds, and the solving performance stays quite consistent between using 20 iterations and 200 iterations.

6. Conclusion

In this paper, we propose a novel verification problem, called *saliency-robustness*, which aims to verify whether a Vanilla Gradient saliency map can serve as an explanation or certification for a prediction. We model the problem as a constraint satisfiability problem over linear real arithmetic and show that for small networks, our formulation can be solved by off-the-shelf SMT solvers like Z3. Furthermore, when Z3 doesn't scale to networks of bigger sizes, we propose a method combining constraint-based neural network verifier with Jacobian bounding to solve it more effectively. Experiments show that our method outperforms Z3 and scales to the `mnistfc_256x2` pre-trained network used in VNNCOMP23.

Limitations and Future Work There are several limitations in this paper and open problems for future work. First, one thread to validity is the soundness of floating point arithmetic, which is a known issue in neural network verification in general Katz et al. (2017). Second, our proposed method is limited in scalability by both components: the Jacobian bounding algorithm of Crown/LiRPA does not scale to deeper neural networks and the quality of the bounds degrades significantly as we go deeper Shi et al. (2022), and Marabou (or any constraint-based NN verifiers for that matter) is inherently slower than abstraction-based methods in exchange for being precise. Finally, our current method does not handle non-linear formulations of saliency maps, which are used in many works such as Integrated Gradients or SmoothGrad. Extending our work to support such saliency map functions is a challenge for future work.

As the first work at verifying the saliency-robustness property, our proposed method serves as a proof of concept, and we humbly hope we interest other researchers to build upon it toward more robust saliency maps.

Acknowledgments

We thank the anonymous reviewers for their insightful comments. This work was supported, in part, by Individual Discovery Grants from the Natural Sciences and Engineering Research Council of Canada and the Canada CIFAR AI Chair Program.

References

Julius Adebayo, Justin Gilmer, Michael Muelly, Ian Goodfellow, Moritz Hardt, and Been Kim. Sanity checks for saliency maps. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 9525–9536, Red Hook, NY, USA, 2018. Curran Associates Inc.

Aws Albarghouthi. Introduction to neural network verification. *CoRR*, abs/2109.10317, 2021. URL https://arxiv.org/abs/2109.10317.

David Baehrens, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert Müller. How to explain individual classification decisions. *J. Mach. Learn. Res.*, 11:1803–1831, aug 2010. ISSN 1532-4435.

John Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-8:679 – 698, 12 1986. doi: 10.1109/TPAMI.1986.4767851.

Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 238–252, 1977.

Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-78800-3.

Bryce Goodman and Seth Flaxman. European union regulations on algorithmic decision-making and a "right to explanation". *AI Magazine*, 38(3):50–57, oct 2017. doi: 10.1609/aimag.v38i3.2741. URL https://doi.org/10.1609%2Faimag.v38i3.2741.

Matthias Hein and Maksym Andriushchenko. Formal guarantees on the robustness of a classifier against adversarial manipulation. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/e077e1a544eec4f0307cf5c3c721d944-Paper.pdf.

Matt Jordan and Alexandros G Dimakis. Exactly computing the local lipschitz constant of relu networks. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 7344–7353. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/5227fa9a19dce7ba113f50a405dcaf09-Paper.pdf.

Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In Rupak Majumdar and Viktor Kunčak, editors, *Computer Aided Verification*, pages 97–117, Cham, 2017. Springer International Publishing. ISBN 978-3-319-63387-9.

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. The marabou framework for verification and analysis of deep neural networks. In *CAV (1)*, volume 11561 of *Lecture Notes in Computer Science*, pages 443–452. Springer, 2019.

Been Kim, Martin Wattenberg, Justin Gilmer, Carrie Cai, James Wexler, Fernanda Viegas, and Rory Sayres. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). 2017. doi: 10.48550/ARXIV.1711.11279. URL https://arxiv.org/abs/1711.11279.

Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 1885–1894. JMLR.org, 2017.

Christoph Molnar. *Interpretable Machine Learning*. 2 edition, 2022. URL https://christophm.github.io/interpretable-ml-book.

Mark Niklas Müller, Christopher Brix, Stanley Bak, Changliu Liu, and Taylor T. Johnson. The third international verification of neural networks competition (vnn-comp 2022): Summary and results, 2023.

John A. Nelder and Roger Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

Ramprasaath R. Selvaraju, Abhishek Das, Ramakrishna Vedantam, Michael Cogswell, Devi Parikh, and Dhruv Batra. Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization. *CoRR*, abs/1610.02391, 2016. URL http://arxiv.org/abs/1610.02391.

Zhouxing Shi, Yihan Wang, Huan Zhang, J. Zico Kolter, and Cho-Jui Hsieh. Efficiently computing local lipschitz constants of neural networks via bound propagation. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 2350–2364. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/0ff54b4ec4f70b3ae12c8621ca8a49f4-Paper-Conference.pdf.

Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 3145–3153. JMLR.org, 2017.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Workshop Track Proceedings*, 2014a. URL http://arxiv.org/abs/1312.6034.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Workshop Track Proceedings*, 2014b. URL http://arxiv.org/abs/1312.6034.

Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. Fast and effective robustness certification. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman,

N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/f2f446980d8e971ef3da97af089481c3-Paper.pdf.

Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proc. ACM Program. Lang.*, 3(POPL), jan 2019. doi: 10.1145/3290354. URL https://doi.org/10.1145/3290354.

Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda B. Viégas, and Martin Wattenberg. Smoothgrad: removing noise by adding noise. *CoRR*, abs/1706.03825, 2017. URL http://arxiv.org/abs/1706.03825.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. *CoRR*, abs/1703.01365, 2017. URL http://arxiv.org/abs/1703.01365.

Hoang-Dung Tran, Neelanjana Pal, Patrick Musau, Diego Manzanas Lopez, Nathaniel Hamilton, Xiaodong Yang, Stanley Bak, and Taylor T. Johnson. Robustness verification of semantic segmentation neural networks using relaxed reachability. In *Computer Aided Verification: 33rd International Conference, CAV 2021, Virtual Event, July 20–23, 2021, Proceedings, Part I*, page 263–286, Berlin, Heidelberg, 2021. Springer-Verlag. ISBN 978-3-030-81684-1. doi: 10.1007/978-3-030-81685-8_12. URL https://doi.org/10.1007/978-3-030-81685-8_12.

Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. Beta-CROWN: Efficient bound propagation with per-neuron split constraints for complete and incomplete neural network verification. *Advances in Neural Information Processing Systems*, 34, 2021.

Tsui-Wei Weng, Huan Zhang, Pin-Yu Chen, Jinfeng Yi, Dong Su, Yupeng Gao, Cho-Jui Hsieh, and Luca Daniel. Evaluating the robustness of neural networks: An extreme value theory approach. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=BkUHlMZ0b.

Han Xu, Yao Ma, Haochen Liu, Debayan Deb, Hui Liu, Jiliang Tang, and Anil K. Jain. Adversarial attacks and defenses in images, graphs and text: A review. *International Journal of Automation and Computing*, 17:151–178, 2020.

Huan Zhang, Pengchuan Zhang, and Cho-Jui Hsieh. Recurjac: An efficient recursive algorithm for bounding jacobian matrix of neural networks and its applications. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'19/IAAI'19/EAAI'19. AAAI Press, 2019. ISBN 978-1-57735-809-1. doi: 10.1609/aaai.v33i01.33015757. URL https://doi.org/10.1609/aaai.v33i01.33015757.