Efficient Preimage Approximation for Neural Network Certification

Anton Björklund[a,*], **Mykola Zaitsev**[a] **and Marta Kwiatkowska**[a]

[a]Department of Computer Science, University of Oxford, United Kingdoms
ORCID (Anton Björklund): https://orcid.org/0000-0002-7749-2918, ORCID (Mykola Zaitsev):
https://orcid.org/0000-0002-7547-5284, ORCID (Marta Kwiatkowska): https://orcid.org/0000-0001-9022-7599

Abstract. The growing reliance on artificial intelligence in safety- and security-critical applications demands effective neural network certification. A challenging real-world use case is certification against "patch attacks", where adversarial patches or lighting conditions obscure parts of images, for example traffic signs. One approach to certification, which also gives quantitative coverage estimates, utilizes preimages of neural networks, i.e., the set of inputs that lead to a specified output. However, these preimage approximation methods, including the state-of-the-art PREMAP algorithm, struggle with scalability. This paper presents novel algorithmic improvements to PREMAP involving tighter bounds, adaptive Monte Carlo sampling, and improved branching heuristics. We demonstrate efficiency improvements of at least an order of magnitude on reinforcement learning control benchmarks, and show that our method scales to convolutional neural networks that were previously infeasible. Our results demonstrate the potential of preimage approximation methodology for reliability and robustness certification.

1 Introduction

The widespread deployment of AI technologies places significant demands on their trustworthiness and reliability, which are of particular concern in high-stakes applications, as recognized in the EU AI Act [9]. Much effort has been devoted to ensuring safety and security of AI applications involving neural networks (NNs) because of their instability to adversarial perturbations [12]. A particularly challenging real-world example are "patch attacks" on images, such as traffic signs. In Figure 1, we show two examples of physical patch attacks: real graffiti on a traffic sign [10] (left) and a patch caused by a sunbeam [5] (middle), which can result from a specific adversarial manipulation of the image or for natural reasons.

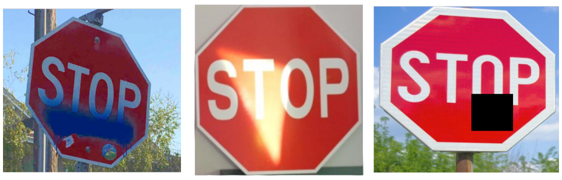

Figure 1. Examples of physical patch attacks. Left: real graffiti [10]. Middle: sunbeam [5]. Right: our abstraction of a patch attack.

The above examples highlight the importance of studying robustness of image recognition models to adversarial patch attacks for applications such as safe autonomous driving or biometric security software. To this end, a wide range of methodologies have been developed for neural networks to enhance their robustness, either via (certified) adversarial training [26, 20] or designing certifiable [33] or empirical [31] defences. An alternative method is to certify robustness by computing deterministic [13, 15] or probabilistic [30, 36] guarantees on the network outputs. A drawback of certifiable defences is their relatively high inference time, although there are viable trade-offs [33, 32], Similarly to formal verification, certification is more powerful than adversarial training or empirical defences because it can guarantee that the network output is correct for all inputs in a given region. Using the patch attack in Figure 1 (right) as an illustration, and assuming the patch is placed in a fixed position, certification aims to guarantee the prediction for all possible (RGB) colour values.

Adversarial robustness certification methods commonly work by bounding the outputs of the neural network for a given input subdomain, which may yield loose bounds for the resulting output over-approximation. A recently developed alternative approach focuses instead on bounding the preimage of a given output specification (typically a polyhedron), either through over-approximating the preimage [16, 39] or under-approximating it [38, 39]. Preimage under-approximation, in contrast to over-approximation, offers stronger compliance guarantees, since all inputs in the computed approximation satisfy the guarantee, and quantitative verification, i.e., estimating the proportion of inputs that meet the specification, which can directly serve as a reliability certificate. However, despite the positive experimental results reported in [38, 39], existing methods do not scale to high dimensions, such as commonly used convolutional neural networks, which is compounded by the prohibitive size of the branching computation tree.

In this paper, we aim to significantly improve the scalability and computational performance of the state-of-the-art preimage approximation algorithm PREMAP [38, 39]. This raises a number of challenges, which we resolve by careful analysis that allowed us to select effective algorithmic improvements such as bound tightening, optimized branching and refinement, as well as adaptive Monte Carlo sampling. We experimentally evaluate our method on a range of case studies, showing that our method is able to complete almost twice the number of cases solved using the original method. Now we are able to certify robustness to patch attacks for convolutional networks applied on GTSRB (traffic signs), SVHN (house numbers) and CIFAR (standard image benchmark), which were not feasible with the

[*] Corresponding Author. Email: anton.bjorklund@cs.ox.ac.uk.

original PREMAP. We also provide a comparison with the previous version of PREMAP on controller benchmarks, showing at least an order of magnitude improvements in running time. Additionally, we demonstrate how PREMAP can be used for interpretability, to investigate important regions of the input. Our results demonstrate the potential of preimage approximation methodology for exploitation in reliability and robustness certification.

2 Background

This section presents the relevant background. In Section 2.1, we define preliminary concepts and provide an overview of the PREiMage APproximation (PREMAP) method [38, 39], focusing on key aspects needed for our algorithmic improvements; for more detail of the techniques please refer to [39]. Related works are discussed in Section 2.2.

2.1 Preliminaries

Neural networks. We use $f : \mathbb{R}^d \to \mathbb{R}^m$ to denote a feed-forward neural network, where $f(\mathbf{x}) = \mathbf{y}$. For layer l, $\mathbf{W}^l$ denotes the weight matrix, $\mathbf{b}^l$ the bias, $\mathbf{z}^l$ the pre-activation neurons, and $h^l(\mathbf{z}^l)$ the activation function, such that $\mathbf{z}^l = \mathbf{W}^l h^{l-1}(\mathbf{z}^{l-1}) + \mathbf{b}^l$. We use under- and overlines to denote the lower and upper bounds, e.g., $\underline{\mathbf{x}}_i \leq \mathbf{x}_i \leq \overline{\mathbf{x}}_i$. In this paper we focus on ReLU neural networks, where $h^l(\mathbf{z}^l) = \text{ReLU}(\mathbf{z}^l) = \max(\mathbf{z}^l, 0)$ is applied element-wise, but our method can also be generalized to other activation functions that can be bounded by linear functions [37, 6].

Linear relaxation. Linear relaxation is used to transform non-convex neuron constraints into linear programs, which provide effective means to approximate a neural network to ensure tractability of analysis. As shown in Figure 2, given concrete lower and upper bounds $\underline{\mathbf{z}}^l \leq \mathbf{z}^l \leq \overline{\mathbf{z}}^l$ on the pre-activation values of layer l, there are three cases to consider. When $\overline{\mathbf{z}}_i^l \leq 0$ then $h^l(\mathbf{z}_i^l) = 0$ and the neuron is *inactive*. Similarly, when $\underline{\mathbf{z}}_i^l \geq 0$ then $h^l(\mathbf{z}_i^l) = \mathbf{z}_i^l$ and the neuron is *active*. Otherwise, the neuron is *unstable* and can be bounded by

$$\alpha \mathbf{z}_i^l \leq h^l(\mathbf{z}_i^l) \leq (\mathbf{z}_i^l - \underline{\mathbf{z}}_i^l)\overline{\mathbf{z}}_i^l / (\overline{\mathbf{z}}_i^l - \underline{\mathbf{z}}_i^l), \tag{1}$$

where α is any value in $[0, 1]$.

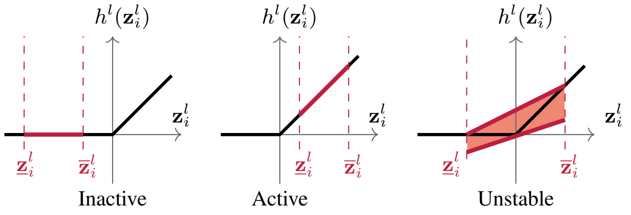
$h^l(\mathbf{z}_i^l)$
$\mathbf{z}_i^l$
$\underline{\mathbf{z}}_i^l$
$\overline{\mathbf{z}}_i^l$
Inactive
$h^l(\mathbf{z}_i^l)$
$\mathbf{z}_i^l$
$\underline{\mathbf{z}}_i^l$
$\overline{\mathbf{z}}_i^l$
Active
$h^l(\mathbf{z}_i^l)$
$\mathbf{z}_i^l$
$\underline{\mathbf{z}}_i^l$
$\overline{\mathbf{z}}_i^l$
Unstable

Figure 2. Linear bounds for inactive, active and unstable ReLU neurons.

For the neural network f, linear relaxation is used to compute linear lower and upper bounds for the output like

$$\underline{\mathbf{A}}\mathbf{x} + \underline{\mathbf{b}} \leq f(\mathbf{x}) \leq \overline{\mathbf{A}}\mathbf{x} + \overline{\mathbf{b}}, \tag{2}$$

for a given bounded input region. These methods are known as Linear Relaxation based Perturbation Analysis (LiRPA [34]) algorithms.

Preimage approximation. Given a subset $\mathbb{O} \subset \mathbb{R}^m$, the *preimage* of a function $f : \mathbb{R}^d \to \mathbb{R}^m$ is defined to be the set of all inputs $\mathbf{x} \in \mathbb{R}^d$ that are mapped by f to an element of $\mathbb{O}$. Since computing the exact preimage is intractable, we are interested in computing an *under-* or *over-approximation* of the preimage.

To apply LiRPA, we need to bound the input domain $\mathbb{I} \subset \mathbb{R}^d$, for example as an axis-aligned hyper-rectangle $\mathbb{I} \subseteq \{\mathbf{x} \in \mathbb{R}^d \mid \bigwedge_{i=1}^d \underline{\mathbf{x}}_i \leq \mathbf{x}_i \leq \overline{\mathbf{x}}_i\}$. We specify the output domain $\mathbb{O}$ as a set of linear inequalities (a *polytope*) on the output $\bigwedge_{j=1}^K (\mathbf{c}_j^\top \mathbf{y} + d_j \geq 0)$. Since the inequalities are linear, they can be implemented by adding a linear layer to the neural network $f_\mathbb{O}(\mathbf{x})_j = \mathbf{c}_j^\top f(\mathbf{x}) + d_j$. Then, by bounding $f_\mathbb{O}$ with LiRPA, such that $\underline{\mathbf{A}}\mathbf{x} + \underline{\mathbf{b}} \leq f_\mathbb{O}(\mathbf{x}) \leq \overline{\mathbf{A}}\mathbf{x} + \overline{\mathbf{b}}$, we obtain an under-approximation $\mathbf{P}$ of the preimage as

$$\mathbf{x} \in \mathbb{I} \wedge \underline{\mathbf{A}}\mathbf{x} + \underline{\mathbf{b}} \geq 0 \implies f(\mathbf{x}) \in \mathbb{O}$$

and an over-approximation $\mathbf{P}$ as

$$\mathbf{x} \in \mathbb{I} \wedge f(\mathbf{x}) \in \mathbb{O} \implies \overline{\mathbf{A}}\mathbf{x} + \overline{\mathbf{b}} \geq 0.$$

PREMAP. The anytime PREMAP algorithm [39, 38], implements both under- or over-approximation of the preimage of a given output specification $\mathbb{O}$ restricted to the input domain $\mathbb{I}$, in the form of a disjoint union of polytopes. However, in this paper we will focus on under-approximations because they offer stronger guarantees and enable quantitative verification. Algorithm 1 shows an overview of the method with our modification highlighted with green shading (discussed further in Section 3). PREMAP relies on linear relaxation with LiRPA to approximate the preimage (see line 3), which is further refined through splitting the problem into disjoint subregions (on lines 8 and 9), which can be approximated independently in parallel (see lines 10 and 11) with a divide-and-conquer approach. The algorithm terminates upon reaching a preset threshold for the preimage approximation, with exact rather than statistical quantitative guarantees.

Branching and refinement. Since the initial bound given by LiRPA might be very loose (Algorithm 1, line 3), PREMAP iteratively partitions a subdomain $\mathbb{I}_{\text{sub}} \subseteq \mathbb{I}$ into two smaller subdomains $\mathbb{I}_{\text{sub}}^-$ and $\mathbb{I}_{\text{sub}}^+$ (lines 5 to 12). Which subdomain to split is selected by the largest difference in estimated volume between the exact preimage and the approximation in the subdomain (lines 6). The split can be either on the input, which partitions the input into two subdomains along some feature, or based on the pre-activation value of an intermediate unstable neuron [39], called ReLU splitting. As this paper deals with larger input domains, for which input splitting performs less well, we will focus on the latter (line 8).

A key advantage of ReLU splitting is that it allows us to replace unstable neuron bounds with precise bounds. For an unstable ReLU neuron $h^l(\mathbf{z}_j^l) = \max(0, \mathbf{z}_j^l)$, we use linear relaxation to bound the post-activation value as in Equation (1). When a split is introduced, the neuron becomes stable in each subdomain, and the exact linear functions $h^l(\mathbf{z}_j^l) = \mathbf{z}_j^l$ and $h^l(\mathbf{z}_j^l) = 0$ can be used in place of its linear relaxation. This can typically tighten the approximation on each subdomain as the linear relaxation errors for this unstable neuron are removed and substituted with the exact symbolic function.

The bounds, $\underline{\mathbf{A}}\mathbf{x} + \underline{\mathbf{b}}$, for the new under-approximations are optimized via projected gradient descent (Algorithm 2, line 3) and the process of refining the subdomains continues until an appropriate termination criterion is met (Algorithm 1, line 7). The procedure can also be stopped anytime to produce a valid (but smaller) approximation. The (anytime) union of the preimage approximations $\mathbf{P}$ for the disjoint subdomains $\mathbb{I}_{\text{sub}}$ constitutes the preimage approximation of the output property.

Algorithm 1 PREMAP is an algorithm for finding preimage under approximations. The inputs are the neural network f, the input domain $\mathbb{I}$, the output specifications $\mathbb{O}$, the early stopping threshold v_{th}, and the number of samples n. Some functions are detailed in Algorithm 2. The lines with our modifications compared to [39] are highlighted with green shading and are discussed in detail in Sections 3.1 to 3.3.

```
function PREMAP($f, \mathbb{I}, \mathbb{O}, v_{\text{th}}, n$)
    $\mathbf{X} \leftarrow$ SAMPLE($\mathbb{I}, 5n$)                    $\rangle\rangle$ Sample extra when it is easy
    $\mathbf{P} \leftarrow$ PREIMAGEAPPROX($f_{\mathbb{O}}, \mathbb{I}, \mathbf{X}$)
    $\mathbb{D} \leftarrow \{(\mathbf{P}, \mathbb{I}, \mathbf{X})\}$              $\rangle\rangle$ Approximation, domain, samples
    while ESTIMATECOVERAGE($\mathbb{D}, f_{\mathbb{O}}$) $< v_{\text{th}}$
        $\mathbf{P}, \mathbb{I}_{\text{sub}}, \mathbf{X} \leftarrow$ POP($\mathbb{D}, \arg\max_j$(PRIORITY($\mathbb{D}_j, f_{\mathbb{O}}$)))
        $\mathbf{X} \leftarrow$ SAMPLE($\mathbb{I}_{\text{sub}}, n, \mathbf{X}$)                 $\rangle\rangle$ Top up samples
        $l, i \leftarrow$ SELECTNEURON($f_{\mathbb{O}}, \mathbb{I}_{\text{sub}}, \mathbf{X}$)
        $\mathbb{I}^-_{\text{sub}}, \mathbf{X}^-, \mathbb{I}^+_{\text{sub}}, \mathbf{X}^+ \leftarrow$ SPLITNEURON($\mathbb{I}_{\text{sub}}, \mathbf{X}, l, i$)
        $\mathbf{P}^- \leftarrow$ PREIMAGEAPPROX($f_{\mathbb{O}}, \mathbb{I}^-_{\text{sub}}, \mathbf{X}^-$)
        $\mathbf{P}^+ \leftarrow$ PREIMAGEAPPROX($f_{\mathbb{O}}, \mathbb{I}^+_{\text{sub}}, \mathbf{X}^+$)
        $\mathbb{D} \leftarrow \mathbb{D} \cup \{(\mathbf{P}^-, \mathbb{I}^-_{\text{sub}}, \mathbf{X}^-),\ (\mathbf{P}^+, \mathbb{I}^+_{\text{sub}}, \mathbf{X}^+)\}$
    return $\{\mathbf{P} \mid (\mathbf{P}, \cdot, \cdot) \in \mathbb{D}\}$
```

Algorithm 2 Additional functions for PREMAP, see Algorithm 1 for details. The LIRPAfunction produces linear bounds using the LiRPA library [34] with optimisable parameters $\boldsymbol{\alpha}$.

```
function PREIMAGEAPPROX($f_{\mathbb{O}}, \mathbb{I}_{\text{sub}}, \mathbf{X}$)
    $\underline{\mathbf{A}}^{(\boldsymbol{\alpha})}, \underline{\mathbf{b}}^{(\boldsymbol{\alpha})} \leftarrow$ LIRPA($f_{\mathbb{O}}, \mathbb{I}_{\text{sub}}$)
    $\boldsymbol{\alpha}, \beta \leftarrow \arg\max_{\boldsymbol{\alpha}} \sum_{\mathbf{x} \in \mathbf{X}} \sigma(-\log(\sum \exp(-\underline{\mathbf{A}}^{(\boldsymbol{\alpha})}\mathbf{x} - \underline{\mathbf{b}}^{(\boldsymbol{\alpha})})))$
    $\mathbf{P} \leftarrow \{\mathbf{x} \mid \mathbf{x} \in \mathbb{I}_{\text{sub}} \wedge \underline{\mathbf{A}}^{(\boldsymbol{\alpha})}\mathbf{x} + \underline{\mathbf{b}}^{(\boldsymbol{\alpha})} \geq 0\}$
    return $\mathbf{P}$                              $\rangle\rangle$ Preimage under approximation
function ESTIMATECOVERAGE($\mathbb{D}, f_{\mathbb{O}}$)
    $v_{\mathbf{P}} \leftarrow \sum_{(\mathbf{P}, \mathbb{I}_{\text{sub}}, \mathbf{X}) \in \mathbb{D}} \sum_{\mathbf{x} \in \mathbf{X}} [\mathbf{x} \in \mathbf{P}] / |\mathbf{X}| \cdot |\mathbb{I}_{\text{sub}}|$
    $v_{\mathbb{O}} \leftarrow \sum_{(\mathbf{P}, \mathbb{I}_{\text{sub}}, \mathbf{X}) \in \mathbb{D}} \sum_{\mathbf{x} \in \mathbf{X}} [f_{\mathbb{O}}(\mathbf{x}) \geq 0] / |\mathbf{X}| \cdot |\mathbb{I}_{\text{sub}}|$
    return $v_{\mathbf{P}} / v_{\mathbb{O}}$              $\rangle\rangle$ Approximated fraction of the preimage
function PRIORITY($\mathbf{P}, \mathbb{I}_{\text{sub}}, \mathbf{X}, f_{\mathbb{O}}$)
    $v_{\mathbb{O} \setminus \mathbf{P}} \leftarrow \sum_{\mathbf{x} \in \mathbf{X}} [f_{\mathbb{O}}(\mathbf{x}) \geq 0 \wedge \mathbf{x} \notin \mathbf{P}] / |\mathbf{X}| \cdot |\mathbb{I}_{\text{sub}}|$
    return $v_{\mathbb{O} \setminus \mathbf{P}}$            $\rangle\rangle$ Volume of preimage not approximated
```

2.2 *Related works*

Robustness certification methods focus on computing deterministic [13, 15] or probabilistic/statistical [30, 36] guarantees on the output of a neural network. The methods can be classified into complete methods, such as constraint solving [15], or sound though incomplete methods, e.g., convex relaxation [25], which can be strengthened to ensure completeness by employing branch-and-bound procedures [3, 4]. Building on [38, 39], our work employs convex (linear) relaxation and Monte Carlo sampling in a divide-and-conquer framework for approximating the preimage of the neural network with exact (not statistical) quantitative guarantees. We leverage LiRPA [34] as implemented in CROWN tools [37, 35, 29] to compute symbolic relaxation bounds, adapting to preimage approximation. We note that exact preimage computation is intractable for high dimensions, although a variant of this problem known as backward reachability has been studied in control [28] through exact computation or (guaranteed) over-approximation [24, 8]. The method of [16] is limited to preimage over-approximation and scales less well than PREMAP, and consequently our method. Compared to [3, 4], our divide-and-conquer approach focuses on minimizing the difference between the under-approximation and the preimage rather than maximizing a function value.

Methods of defending against patch attacks [31, 32] include active defences, such as small receptive fields or masking out all adversarial pixels from the input image, and passive defences, for example certifiable training using bound propagation. The most common of these, small receptive fields, extract features and predictions from small regions of the image, to avoid spatially localized attacks. The prediction involves robust aggregation such as majority voting. The smaller the receptive fields the better robustness but the worse clean performance, and [33] offers a computationally efficient trade-off.

Compared to active defences, which extend the model, our approach computes guarantees for the underlying neural network models. Certifiable training [19] involves estimating and propagating the bounds of neuron activations in each layer to bound the attackers influence on final predictions and is computationally expensive; in contrast, our method approximates the set of inputs for which the prediction is guaranteed for a given trained network.

3 Methods

The aim is to apply PREMAP [38, 39] on larger models with larger input spaces. This means we need to revisit some of the design decisions that worked well on smaller problems. In Section 3.1 we calculate tighter bounds for intermediate layers, in Section 3.2 we describe how we adaptively draw more Monte Carlo samples, and in Section 3.3 we design new heuristics for the selection of which neuron to split next.

3.1 *Bound tightening*

When a neuron is split, the linear relaxation error from that neuron towards all the following layers is reduced. However, β-CROWN [29], on which PREMAP is built, typically only updates the final layer because the optimization is computationally intensive [29]. Instead of using the full β-CROWN optimization, in this work, we propagate the bounds using backwards mode LiRPA [34]. These bounds are not as tight, but much faster to compute.

While LiRPA [34] only propagates bounds to the following layers in the network, a split also implies restrictions in previous layers. Hence, we derive how the bounds of previous layers *and the input* might tighten after a split. As with backwards mode LiRPA, these bounds are fast to compute but result in looser bounds.

After splitting the i:th neuron on layer l, we use LiRPA [34] as in Equation (2) to calculate the linear approximation bounds with respect to the preceding layers $m < l$ (including the input) such that

$$\underline{\mathbf{A}}^{lm}\mathbf{z}^m + \underline{\mathbf{b}}^{lm} \leq \mathbf{z}^l \leq \overline{\mathbf{A}}^{lm}\mathbf{z}^m + \overline{\mathbf{b}}^{lm}.$$

The two constraints defined by the split, $\mathbf{z}_i^l < 0$ and $\mathbf{z}_i^l \geq 0$, can be propagated backwards using the linear approximations:

$$[\underline{\mathbf{A}}^{lm}\mathbf{z}^m + \underline{\mathbf{b}}^{lm}]_i < 0 \ \text{ and } \ 0 \leq [\overline{\mathbf{A}}^{lm}\mathbf{z}^m + \overline{\mathbf{b}}^{lm}]_i. \tag{3}$$

From this we can update the bounds $\underline{\mathbf{z}}^m$ and $\overline{\mathbf{z}}^m$ so that:

$$\underline{\mathbf{z}}_j^m \leftarrow \max(\underline{\mathbf{z}}_j^m, -(c + \underline{\mathbf{b}}_i^{lm})/\underline{\mathbf{A}}_{ij}^{lm}) \mid \underline{\mathbf{A}}_{ij}^{lm} < 0$$

$$\overline{\mathbf{z}}_j^m \leftarrow \min(\overline{\mathbf{z}}_j^m, -(c + \underline{\mathbf{b}}_i^{lm})/\underline{\mathbf{A}}_{ij}^{lm}) \mid \underline{\mathbf{A}}_{ij}^{lm} > 0$$

for $\mathbf{z}_i^l < 0$, where $c = \sum_{k \neq j} \min(\underline{\mathbf{A}}_{ik}^{lm}\underline{\mathbf{z}}_k^m, \underline{\mathbf{A}}_{ik}^{lm}\overline{\mathbf{z}}_k^m)$ and

$$\underline{\mathbf{z}}_j^m \leftarrow \max(\underline{\mathbf{z}}_j^m, -(c + \overline{\mathbf{b}}_i^{lm})/\overline{\mathbf{A}}_{ij}^{lm}) \mid \overline{\mathbf{A}}_{ij}^{lm} > 0$$

$$\overline{\mathbf{z}}_j^m \leftarrow \min(\overline{\mathbf{z}}_j^m, -(c + \overline{\mathbf{b}}_i^{lm})/\overline{\mathbf{A}}_{ij}^{lm}) \mid \overline{\mathbf{A}}_{ij}^{lm} < 0$$

for $\mathbf{z}_i^l \geq 0$, where $c = \sum_{k \neq j} \max(\overline{\mathbf{A}}_{ik}^{lm} \underline{\mathbf{z}}_k^m, \overline{\mathbf{A}}_{ik}^{lm} \overline{\mathbf{z}}_k^m)$. Note that the division introduces additional numerical instability, which we mitigate with padding around zero, making the bounds slightly looser. After the "reverse" propagation of bounds we use LiRPA to update the bounds of subsequent layers, as mentioned above. This happens in Algorithm 1 at the end of the SPLITNEURON function on line 9.

3.2 *Sampling*

The samples are used in PREMAP for three things: Monte Carlo estimation of the volume of both the preimage and the approximation (in Algorithm 2, lines 7 to 11), the gradient-based optimization of the bounds in β-CROWN to maximise the preimage approximation volume [39] (on line 3 of Algorithm 2), and in the heuristics to select which neuron to split (in Algorithm 1, line 8).

We could, like the previous version of PREMAP [39], only draw a large, fixed set of initial samples. Each split defines two subsets where we can reuse the samples (see line 9 in Algorithm 1). However, the size of the subsets decrease exponentially with each split. Hence, in this work, when the subsets get too small we draw additional samples (line 7). At first, we use rejection sampling from uniform samples, $\mathbb{U}(\underline{\mathbf{x}}, \overline{\mathbf{x}})$. However, just as with subsets, the hit rate decreases exponentially.

The inequalities introduced in Equation (3) create a polytope in the input space. To sample from this polytope we use *hit-and-run* sampling [21, 7]. The idea behind *hit-and-run* sampling is to start from a point inside the polytope, conveniently provided by line 9 of Algorithm 1. Then we draw a random direction, calculate the distance to the edge of the polytope, and uniformly sample a new point on this line. Since consecutive steps are correlated, we take multiple steps between every recorded sample. This yields uniformly distributed samples [21] from the polytope and the procedure can be parallelized and GPU accelerated [7]. Note that Equation (3) is an outer bound, so we still need to do rejection sampling on the activations.

3.3 *Heuristics*

The heuristic for selecting which neuron to split is condensed in Algorithm 1 on line 8. What we do is calculate a score for each neuron i on layer l and select the neuron with the highest score. The original PREMAP [39] aims to keep the splits balanced in terms of samples with a score function of

$$1 - \left|2 \sum\nolimits_{i=1}^{n} [\mathbf{z}_{ij}^l \geq 0]/n - 1\right|,$$

where $\mathbf{z}_{ij}^l$ is the pre-activation for sample j (out of n) at neuron i on layer l. However, this heuristic offers no indication of how well LiRPA [34] might tighten the bounds. Therefore, we suggest replacing it with a combination of heuristics described below.

Since the layer l is likely to be in the middle of the network, after a split the negative side, $\mathbf{z}_j^l <= 0.0$, will be constant (by the definition of ReLU) while the positive side will be non-linear with respect to the output. Hence, as heuristics, we estimate how much the split would reduce the "uncertainty" on the negative part.

The heuristics are visualized in Figure 3. First, we calculate the linear bound with respect to the output, $f_{\mathbb{O}}(\mathbf{x}) \geq \underline{\mathbf{A}}^l \mathbf{z}^l + \underline{\mathbf{b}}^l$ (or $f_{\mathbb{O}}(\mathbf{x}) \leq \overline{\mathbf{A}}^l \mathbf{z}^l + \overline{\mathbf{b}}^l$ for an over-approximation). Then we calculate the *area* (yellow) that will be reduced,

$$\sum_k \left|\underline{\mathbf{A}}_i^{lk} \underline{\mathbf{z}}_i^l \underline{\mathbf{z}}_i^l\right| / \max_{l'i'} \left(\sum_k |\underline{\mathbf{A}}_{i'}^{l'k} \underline{\mathbf{z}}_{i'}^{l'} \underline{\mathbf{z}}_{i'}^{l'}| \right),$$

the maximum *under*-approximation (green),

$$\sum_k \left|\underline{\mathbf{A}}_i^{lk} \underline{\mathbf{z}}_i^l\right| / \max_{l'i'} \left(\sum_k |\underline{\mathbf{A}}_{i'}^{l'k} \underline{\mathbf{z}}_{i'}^{l'}| \right),$$

and the average *extra* uncertainty for the samples (blue),

$$\sum_k \left(\sum_{j=1}^{n} \left|\underline{\mathbf{A}}_i^{lk} \min(\mathbf{z}_{ij}^l, 0)\right| / \sum_{j=1}^{n} \left[\mathbf{z}_{ij}^l < 0\right] \right) / \max_{l'i'}(\ldots),$$

where k represents the, potentially, multiple outputs of $f_{\mathbb{O}}$. We combine these heuristics with a weighted sum that is empirically determined in Section 4.2. Before settling on these heuristics we also considered other options, which are documented in Appendix A.

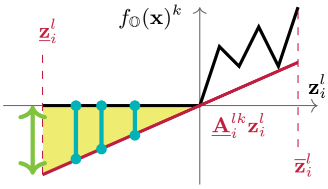
$f_{\mathbb{O}}(\mathbf{x})^k$
$\underline{\mathbf{z}}_i^l$
$\mathbf{z}_i^l$
$\underline{\mathbf{A}}_i^{lk} \mathbf{z}_i^l$
$\overline{\mathbf{z}}_i^l$

Figure 3. Visualising the three new neuron selection heuristics. They are the yellow area, the green distance, and the average blue length.

It is possible that the selected split assigns all samples to one branch. This means either that (a) the other side is impossible to reach with the current constraints but the bounds given by LiRPA are not tight enough to show that, or (b) the other side is reachable, but our current estimate for that volume is zero. Hence, we can avoid processing that branch (in Algorithm 1, line 10 or line 11) without significantly affecting the volume of the under-approximation.

3.4 *Parallelisation*

In the original PREMAP [39, 38] generating the under- or over-approximation after a split (see Algorithm 1, lines 10 and 11) is done in parallel. Furthermore, β-CROWN [29] processes the branches (the while loop on line 5) in batches to take advantage of GPU resources. For the pseudocode we opted not to show either for simplicity, but in practice we use both strategies.

Due to the early stopping criteria introduced by PREMAP (line 5 of Algorithm 1) batching has an additional advantage. If it is difficult to find tighter bounds for the primary branch selected by the heuristics (lines 6 and 8), the "beam search" introduced by the batching might still progress and stop after fewer splits.

4 Experiments

In this section we empirically evaluate our improvements and demonstrate use-cases for PREMAP. In Section 4.1 we describe the experimental setup and datasets and in Section 4.2 we choose the weights for the new heuristics. We compare to the previous version of PREMAP in Section 4.3 and different network architectures in Section 4.4. In Section 4.5 we perform an ablation study and in Section 4.6 we use PREMAP as an investigation tool to enhance interpretability.

4.1 *Experimental setup*

The experiments were run with four cores from an Intel Xeon 6252 CPU and one NVIDIA RTX 2080 Ti, except for the control tasks in Section 4.3 that were run on an Apple M4 Pro CPU. All preimage approximations were limited to 10 minutes with a batch size of 2 (except the previous version of PREMAP that only processes the splits in parallel). Unless otherwise mentioned, we use a stopping volume threshold of 0.9.

For most of the experiments we use the following image datasets: GTSRB [27] with traffic signs, SVHN [22] with house numbers, and CIFAR-10 [17] with general images. Details about the model architectures (fully connected and convolutional) and training is in Appendix B. For experiments with image datasets (32×32 pixels) we randomly sample images, evenly from every class, and patches with random sizes between 3×3 and 6×6 and random positions within the images. Additionally, in Section 4.3, we use five reinforcement learning control tasks further detailed in [39].

The code for the method and experiments will be released under an open source licence upon the acceptance of this paper.

4.2 *Parameters*

As described in Section 3.3 we now have three heuristics to combine. All of them are normalized to $[0, 1]$ to make a weighted sum easier. For this experiment we sample images and patches as described in Section 4.1 and uniformly sample weights for the heuristics. The weights are sampled and evaluated multiple times so that we can normalize the times (by dividing by the mean). We then fit a Gaussian process with an RBF kernel and a white kernel and evaluate it on a grid. The top results can be seen in Table 1, where the "under" heuristic is slightly less important than "area" and "extra".

Table 1. Using a Gaussian process to find weights for the heuristics. Showing the top results from a grid of $\{0.0, 0.25, 0.5, 0.75, 1.0\}$.

Time (s)	area	under	extra
0.715 ± 0.399	1.000,	0.750,	1.000
0.715 ± 0.400	1.000,	1.000,	1.000
0.718 ± 0.399	1.000,	0.500,	1.000
0.724 ± 0.399	1.000,	0.250,	1.000
0.726 ± 0.399	0.750,	1.000,	1.000

4.3 *Comparison*

Here we compare our version of PREMAP with the previous version [39]. In Table 2 we consider the five smaller reinforcement learning tasks from [39], but run them with ReLU splitting instead of input splitting. In all but *dubinsrejoin*, the improved splitting heuristic leads to finishing within time and with fewer splits. The *cartpole* and *dubinsrejoin* preimages are harder to approximate without input splitting [39], but our new version makes more progress and processes more splits in the same time.

Table 3 contains results from the two versions of PREMAP on the GTSRB dataset. We aggregate the results from ten random images from every class with randomly placed and sized patches and count the number of preimage approximations that finish within the time limit. Some preimage approximations are easy enough to finish without entering the loop (Algorithm 1, lines 5 to 12), which is also visible in Figure 4. If we filter out those results, our version is able to complete more than twice the number of harder cases, especially for the convolutional neural network.

Table 2. Comparing our (new) version of PREMAP to the previous (old) version on five reinforcement learning controllers, averaging ten different seeds. The time limit is ten minutes with a stopping threshold of 0.9 for the first two datasets and 0.75 for the last three. We measure the coverage of the under-approximation (larger is better), the number of subdomains required to reach that (smaller is better), and the time it took (smaller is better).

	Coverage		Domains		Time (s)	
Dataset	new	old	new	old	new	old
auto_park	0.918	0.916	2.0	3.0	0.05	0.39
vcas	0.931	0.905	17.0	21.1	0.16	5.57
lunarlander	0.750	0.574	813.8	2745.8	36.60	600.15
cartpole	0.750	0.424	1850.5	1601.7	118.05	600.14
dubinsrejoin	0.207	0.000	4742.7	1159.0	596.11	600.16

Table 3. Comparing our (new) version of PREMAP with the previous (old) version on the GTSRB dataset. Since some patch and image combinations reach the threshold of 0.9 in a single iteration, the second row is filtered to exclude those. We measure the average coverage of the under-approximation (larger is better), the number of instances reaching the threshold within 10 minutes (larger is better), and the average time required (smaller is better).

Dataset: GTSRB			Coverage		Completed		Time (s)	
Model	Subset	#	new	old	new	old	new	old
FC	All	500	0.850	0.714	408	351	120.55	188.37
FC	Hard	292	0.718	0.318	174	61	224.76	441.06
CNN	All	500	0.751	0.501	358	244	186.91	313.70
CNN	Hard	342	0.626	0.197	185	54	282.32	497.88

4.4 *Networks*

In Figure 4 we run PREMAP on the image datasets using both fully connected (FC) and convolutional (CNN) neural networks. We divide the results into two groups based on how close to the edge the patch was. Both GTSRB and SVHN show strong dependence on the location of the patch, which is to be expected since the traffic signs and digits are in the centre of the images. Some SVHN images have distracting digits in the surroundings [22] making the robustness at the edges unsurprising.

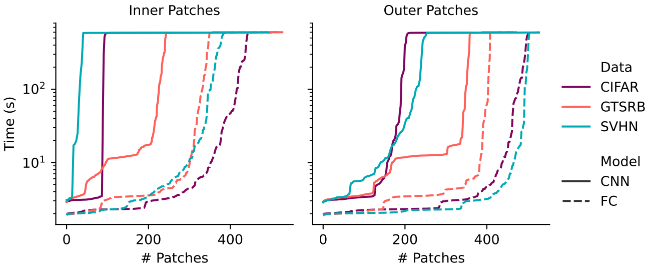

Figure 4. Time to calculate a preimage under-approximation with a 10 minute time limit and a 0.9 threshold. The patches are applied to random images (from every class) with random positions and sizes. We group the results based on if the centre of the patch is within 10 pixels of the centre of the image.

The LiRPA processing and the maximum number of splits is dependent on the number of unstable neurons. For CNNs, the number of activations scales with the size of the image. Here the first layer has $30 \times 30 \times 32 = 28\,800$ activations. As can be seen in Figure 4, this makes the CNNs slower and harder than the fully connected networks even though they have fewer parameters.

4.5 Ablation

In this section we perform an ablation study on a) the tightening of intermediate bounds and b) the new split selection heuristics. The result can be seen in Figure 5, where we measure the coverage of the under-approximation over time running on random patches, as before, and average the results. Tightening is better than using neither improvement, the heuristics are even better, and using both provides the best results. Note that the tightening of intermediate bounds introduces an additional computational overhead, which reduces the utility on a time versus coverage graph.

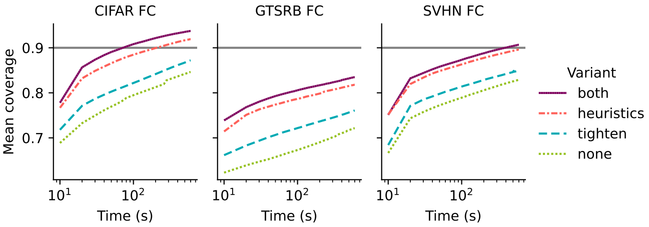

Figure 5. Comparing the average approximation coverage (higher is better) over time. The stopping threshold is 0.9.

Figure 5 also demonstrates the anytime property of the algorithm: if we stop early we still get an under-approximation, just a smaller one, and when given more time more approximations reach the 0.9 threshold.

4.6 Interpretability

A popular approach to explainable artificial intelligence (XAI) is to cover part of the image to see if the classification changes [14]. This reasoning is easy to understand, since if the prediction changes that area must have been important. However, this requires careful design to make a "neutral" replacement pattern [11], preferably such that the modified image is not be out-of-distribution [2]. With PREMAP we do not consider any singular replacement pattern, but rather the whole preimage of an area.

In Figure 6 we apply 4 × 4 patches on a grid covering the image on the left. In the second image from the left we show the estimate of the volume of the preimage (not the approximation) for every patch, which indicates that this model is robust against patches of this size on this image. However, in the third image we see that patches around the '5' require more splits for the approximation. This means that in these areas the model is highly non-linear. Furthermore, in the rightmost image we see that the preimage approximation of some patches do not reach a high coverage in the allotted time, notably around the areas where the '5' could be turned into an '8' with two patches. This is how PREMAP can be used to investigate models, which can aid model development and debugging [1].

5 Conclusions

This work introduced algorithmic improvements to the PREMAP algorithm for finding preimage approximations for neural networks. The main enhancements come from the improved heuristics for selecting splits, the calculation of tighter bounds after a split, and the adaptive Monte Carlo sampling. The improved performance is also empirically validated on both smaller control benchmarks and patch attack certification on images. We also demonstrate how to control the trade-off between time and approximation coverage and how PREMAP could be used for explainable AI. While we have focused our experiments on preimage under-approximation because of its inherent advantages for robustness and reliability certification, the methods also extend to over-approximations.

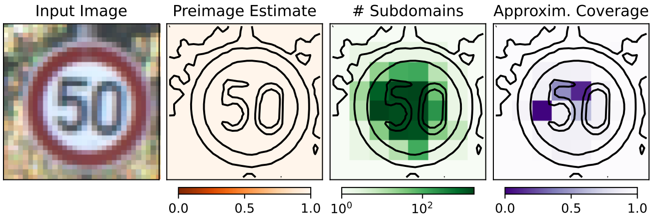

Figure 6. Using PREMAP to investigate which parts of the image are important for classification without the need to provide a user-defined replacement pattern by applying patches on a grid covering the image. The figures are, from left to right, the original image, the estimated preimage volume (higher means more robust), the number of subdomains required (lower means easier), and the coverage of the preimage under approximations (higher is better).

Future work includes applying PREMAP to other domains where certification is needed, for example biometric security or medical diagnosis. PREMAP could also be further enhanced to handle even larger and more complex networks, such as transformers. Finally, extending PREMAP to non-uniform input spaces would unlock additional applications, for example being able to prioritize dangerous conditions.

Acknowledgements

This project received funding from the ERC under the European Unions Horizon 2020 research and innovation programme (FUN2MODEL, grant agreement No. 834115) and ELSA: European Lighthouse on Secure and Safe AI project (grant agreement No. 101070617 under UK guarantee).

A Heuristics

To improve the heuristic for selecting which neuron to split we consider 12 alternative formulas for calculating the priority. These are detailed in Table 4. Note that not all are new, e.g., "balance" is the one used in the previous version of PREMAP [39, 38] and "lower" [3] and "gap" [23] are used in β-CROWN [29].

Some heuristics in Table 4 use the (linear) bounds produced by LiRPA [34, 29]. To minimize computational overhead we store them for reuse after each optimization. Some heuristics use the activation values of the samples. This requires computation on the scale of a forward pass through the network, which is not significant for the overall runtime.

To evaluate the heuristics we draw random samples (i.e., random images with random patches as in Section 4) and run PREMAP multiple times with random combinations of the heuristics and normalize the results by dividing by the mean times for that sample. The weights for the different heuristics are drawn such that roughly half are zero and the rest are uniform.

The results for individual heuristics can be seen in Figure 7, where we fit a linear model to more clearly show the trend. Three of the proposed heuristics have a promising slope (negative), with "balance" and "lower" being among the worst (enabling these correlates with longer times). However, some of these heuristics might be correlated, so we cannot choose based on individual plots.

In Table 5 we enumerate all combinations of the heuristics (only showing the best) and calculate the corresponding linear models. The

Table 4. Potential heuristics for selecting which neuron i (on layer l) to split. There are $j \in [1, \dots, n]$ samples and $k \in [1, \dots, K]$ outputs. All formulas not already limited to $[0, 1]$ are normalised by dividing by the maximum value and stable neurons with bounds $\underline{\mathbf{z}}_i^l \geq 0 \vee \overline{\mathbf{z}}_j^l \leq 0$ are skipped.

Heuristic	Formula
balance	$1 - \lvert 2\sum_j [\mathbf{z}_{ij}^l \geq 0]/n - 1 \rvert$
soft	$1 - \lvert 2\sum_j \sigma(\mathbf{z}_{ij}^l)/n - 1 \rvert$
lower	$\max(-\underline{\mathbf{z}}_i^l, 0)$
width	$\overline{\mathbf{z}}_i^l - \underline{\mathbf{z}}_i^l$
loose	$\overline{\mathbf{z}}_i^l - \underline{\mathbf{z}}_i^l - \left(\max_j(\mathbf{z}_{ij}^l) - \min_j(\mathbf{z}_{ij}^l)\right)$
bound	$1 - \left(\max_j(\mathbf{z}_{ij}^l) - \min_j(\mathbf{z}_{ij}^l)\right)/(\overline{\mathbf{z}}_i^l - \underline{\mathbf{z}}_i^l)$
gap	$(-\underline{\mathbf{z}}_i^l \cdot \overline{\mathbf{z}}_i^l)/(\overline{\mathbf{z}}_i^l - \underline{\mathbf{z}}_i^l)$
area	$\sum_k \lvert \underline{\mathbf{A}}^{lk} \underline{\mathbf{z}}_i^l \underline{\mathbf{z}}_i^l \rvert$
under	$\sum_k \lvert \underline{\mathbf{A}}^{lk} \underline{\mathbf{z}}_i^l \rvert$
extra	$\sum_k \sum_j \lvert \underline{\mathbf{A}}^{lk} \min(\mathbf{z}_{ij}^l, 0) \rvert / \sum_j [\mathbf{z}_{ij}^l < 0]$
stable	$\min_j(\mathbf{z}_{ij}^l) \geq 0 \vee \max_j(\mathbf{z}_{ij}^l) \leq 0$
pure	$2 \cdot \max\left(\left\lvert \sum_j \left[\mathbf{z}_{ij}^l \geq 0 \wedge f(\mathbf{x}_j) \in \mathbb{O}\right] / \sum_j \left[\mathbf{z}_{ij}^l \geq 0\right] - \frac{1}{2} \right\rvert, \left\lvert \sum_j \left[\mathbf{z}_{ij}^l < 0 \wedge f(\mathbf{x}_j) \in \mathbb{O}\right] / \sum_j \left[\mathbf{z}_{ij}^l < 0\right] - \frac{1}{2} \right\rvert \right)$

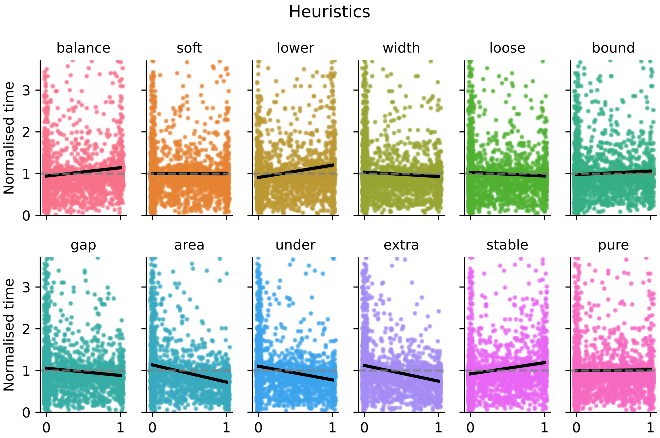

Figure 7. The effect off using various heuristics for the selection of which neuron to split. The black line is a fitted, single variable, linear regression. Lower values (y) with a non-zero weight (x) indicates that the heuristic helped reduce the number of splits required.

combination of "area", "under", and "extra" continues to impress, since anything else added would have a positive coefficient. This is the selection we use for the paper, further discussed in Sections 3.3 and 4.2.

B Neural networks

For the image datasets we use two types of neural networks: fully connected and convolutional. The architectures are described in Table 6. The networks were trained with cross-entropy loss using the AdamW [18] optimizer with brightness and hue variation until the accuracy on the test set stopped decreasing (usually around 50 epochs). For the reinforcement learning tasks we used the same networks as in [39].

References

[1] A. Barredo Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. Garcia, S. Gil-Lopez, D. Molina, R. Benjamins, R. Chatila, and F. Herrera. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, 58:82–115, June 2020. ISSN 1566-2535. doi:10.1016/j.inffus.2019.12.012.

Table 5. Slopes and coefficients of linear models fitted to subsets of the heuristics similar to Figure 7. Lower is better.

Slope	Coefficients
−0.224	area: −0.224
−0.215	extra: −0.215
−0.193	under: −0.193
−0.091	gap: −0.091
−0.308	area: −0.164, extra: −0.144
−0.303	area: −0.179, under: −0.124
−0.288	under: −0.124, extra: −0.164
−0.224	gap: 0.000, area: −0.225
−0.348	area: −0.143, under: −0.088, extra: −0.116
−0.287	gap: 0.045, area: −0.175, extra: −0.156
−0.286	gap: 0.037, area: −0.189, under: −0.133
−0.274	gap: 0.031, under: −0.132, extra: −0.173
−0.323	gap: 0.065, area: −0.157, under: −0.101, extra: −0.131
−0.303	loose: 0.134, area: −0.172, under: −0.121, extra: −0.144
−0.302	width: 0.130, area: −0.176, under: −0.119, extra: −0.137
−0.286	soft: 0.199, area: −0.191, under: −0.132, extra: −0.163

Table 6. Architectures for the neural networks on the image datasets.

Fully connected		Convolutional	
Layer	**Size**	**Layer**	**Size**
Input	32 × 32 × 3	Input	32 × 32 × 3
Flatten	3072	Convolutional	30 × 30 × 3
Linear	300	ReLU	30 × 30 × 32
ReLU	300	Average pooling	15 × 15 × 32
Linear	300	Convolutional	12 × 12 × 32
ReLU	300	ReLU	12 × 12 × 32
Linear	300	Average pooling	6 × 6 × 32
ReLU	300	Convolutional	4 × 4 × 64
Dropout	300	Average pooling	2 × 2 × 64
Linear	$\lvert\mathbf{y}\rvert$	ReLU	2 × 2 × 64
		Flatten	256
		Dropout	256
		Linear	$\lvert\mathbf{y}\rvert$

[2] A. Björklund, A. Henelius, E. Oikarinen, K. Kallonen, and K. Puolamäki. Explaining any black box model using real data. *Frontiers in Computer Science*, 5, Aug. 2023. ISSN 2624-9898. doi:10.3389/fcomp.2023.1143904.

[3] R. Bunel, I. Turkaslan, P. H. S. Torr, P. Kohli, and P. K. Mudigonda. A unified view of piecewise linear neural network verification. In *Advances in Neural Information Processing Systems 31, NeurIPS 208*, pages 4795–4804, 2018.

[4] R. Bunel, J. Lu, I. Turkaslan, P. H. Torr, P. Kohli, and M. P. Kumar. Branch and bound for piecewise linear neural network verification. *Journal of Machine Learning Research*, pages 1–39, 2020.

[5] H. Cao, L. Yuan, G. Xu, Z. He, Z. Fang, and Y. Fang. Secure Traffic Sign Recognition: An Attention-Enabled Universal Image Inpainting Mechanism against Light Patch Attacks. *arXiv:2409.04133*, 2024. doi:10.48550/arXiv.2409.04133.

[6] S. Chevalier, D. Starkenburg, and K. Dvijotham. Achieving the Tightest Relaxation of Sigmoids for Formal Verification. *arXiv:2408.10491*, Aug. 2024. doi:10.48550/arXiv.2408.10491.

[7] M. V. Corte and L. V. Montiel. Novel Matrix Hit and Run for Sampling Polytopes and Its GPU Implementation. *arXiv:2104.07097*, 2021. doi:10.48550/arXiv.2104.07097.

[8] S. Dathathri, S. Gao, and R. M. Murray. Inverse abstraction of neural networks using symbolic interpolation. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019*, pages 3437–3444. AAAI Press, 2019. doi:10.1609/aaai.v33i01.33013437.

[9] European Union. Regulation (EU) 2024/1689 of the European Parliament and of the Council Laying down Harmonised Rules on Artificial Intelligence and Amending Regulations (Artificial Intelligence Act), June 2024. URL https://eur-lex.europa.eu/eli/reg/2024/1689.

[10] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song. Robust physical-world attacks on deep learning models. In *CVPR*, 2018.

[11] R. C. Fong and A. Vedaldi. Interpretable Explanations of Black Boxes by Meaningful Perturbation. *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct. 2017. doi:10.1109/iccv.2017.371.

[12] I. Goodfellow, J. Shlens, and C. Szegedy. Explaining and Harnessing Adversarial Examples. In *International Conference on Learning Representations*, Dec. 2014. doi:10.48550/arXiv.1412.6572.

[13] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu. Safety verification of deep neural networks. In *Computer Aided Verification - 29th International Conference, CAV 2017, Proceedings, Part I*, volume 10426 of *Lecture Notes in Computer Science*, pages 3–29. Springer, 2017. doi:10.1007/978-3-319-63387-9_1.

[14] M. Ivanovs, R. Kadikis, and K. Ozols. Perturbation-based methods for explaining deep neural networks: A survey. *Pattern Recognition Letters*, 150:228–234, Oct. 2021. ISSN 0167-8655. doi:10.1016/j.patrec.2021.06.030.

[15] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *Computer Aided Verification: 29th International Conference, CAV 2017, Proceedings, Part I 30*, pages 97–117. Springer, 2017. doi:10.1007/s10703-021-00363-7.

[16] S. Kotha, C. Brix, J. Z. Kolter, K. Dvijotham, and H. Zhang. Provably Bounding Neural Network Preimages. *Advances in Neural Information Processing Systems*, 36:80270–80290, Dec. 2023. doi:10.48550/arXiv.2302.01404.

[17] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images, 2009.

[18] I. Loshchilov and F. Hutter. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations*, 2019. doi:10.48550/arXiv.1711.05101.

[19] Y. Mao, M. N. Mueller, M. Fischer, and M. Vechev. Understanding certified training with interval bound propagation. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=h05eQniJsQ.

[20] Y. Mao, M. N. Müller, M. Fischer, and M. T. Vechev. Understanding certified training with interval bound propagation. In *The Twelfth International Conference on Learning Representations, ICLR 2024*, 2024. doi:10.48550/arXiv.2306.10426.

[21] L. V. Montiel and J. E. Bickel. Approximating Joint Probability Distributions Given Partial Information. *Decision Analysis*, 10(1):26–41, Mar. 2013. ISSN 1545-8490. doi:10.1287/deca.1120.0261.

[22] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, A. Y. Ng, et al. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*. Granada, 2011. URL http://ufldl.stanford.edu/housenumbers.

[23] A. D. Palma, R. Bunel, A. Desmaison, K. Dvijotham, P. Kohli, P. H. S. Torr, and M. P. Kumar. Improved Branch and Bound for Neural Network Verification via Lagrangian Decomposition. *arXiv:2104.06718*, Apr. 2021. doi:10.48550/arXiv.2104.06718.

[24] N. Rober, S. M. Katz, C. Sidrane, E. Yel, M. Everett, M. J. Kochenderfer, and J. P. How. Backward Reachability Analysis of Neural Feedback Loops: Techniques for Linear and Nonlinear Systems. *IEEE Open Journal of Control Systems*, 2:108–124, 2023. ISSN 2694-085X. doi:10.1109/OJCSYS.2023.3265901.

[25] H. Salman, G. Yang, H. Zhang, C. Hsieh, and P. Zhang. A convex relaxation barrier to tight robustness verification of neural networks. In *Advances in Neural Information Processing Systems 32, NeurIPS 2019*, pages 9832–9842, 2019.

[26] Z. Shi, Y. Wang, H. Zhang, J. Yi, and C.-J. Hsieh. Fast Certified Robust Training with Short Warmup. In *Advances in Neural Information Processing Systems*, volume 34, pages 18335–18349. Curran Associates, Inc., 2021. doi:10.48550/arXiv.2103.17268.

[27] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, 32:323–332, 2012. ISSN 0893-6080. doi:10.1016/j.neunet.2012.02.016.

[28] J. A. Vincent and M. Schwager. Reachable Polyhedral Marching (RPM): A Safety Verification Algorithm for Robotic Systems with Deep Neural Network Components. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9029–9035, May 2021. doi:10.1109/ICRA48506.2021.9561956.

[29] S. Wang, H. Zhang, K. Xu, X. Lin, S. Jana, C.-J. Hsieh, and J. Z. Kolter. Beta-CROWN: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. In *Advances in Neural Information Processing Systems*, volume 34, pages 29909–29921. Curran Associates, Inc., Oct. 2021. doi:10.48550/arXiv.2103.06624.

[30] S. Webb, T. Rainforth, Y. W. Teh, and M. P. Kumar. A Statistical Approach to Assessing Neural Network Robustness. In *International Conference on Learning Representations*, Sept. 2018. doi:10.48550/arXiv.1811.07209.

[31] H. Wei, H. Tang, X. Jia, Z. Wang, H. Yu, Z. Li, S. Satoh, L. Van Gool, and Z. Wang. Physical Adversarial Attack Meets Computer Vision: A Decade Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(12):9797–9817, Dec. 2024. ISSN 1939-3539. doi:10.1109/TPAMI.2024.3430860.

[32] C. Xiang, C. Sitawarin, T. Wu, and P. Mittal. Short: Certifiably robust perception against adversarial patch attacks: A survey. In *1st Symposium on Vehicle Security and Privacy (VehicleSec)*, Mar. 2023.

[33] C. Xiang, T. Wu, S. Dai, J. Petit, S. Jana, and P. Mittal. PATCHCURE: Improving Certifiable Robustness, Model Utility, and Computation Efficiency of Adversarial Patch Defenses. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 3675–3692, 2024. ISBN 978-1-939133-44-1. doi:10.48550/arXiv.2310.13076.

[34] K. Xu, Z. Shi, H. Zhang, Y. Wang, K.-W. Chang, M. Huang, B. Kailkhura, X. Lin, and C.-J. Hsieh. Automatic Perturbation Analysis for Scalable Certified Robustness and Beyond. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

[35] K. Xu, H. Zhang, S. Wang, Y. Wang, S. Jana, X. Lin, and C.-J. Hsieh. Fast and Complete: Enabling Complete Neural Network Verification with Rapid and Massively Parallel Incomplete Verifiers. In *9th International Conference on Learning Representations*, Jan. 2021. doi:10.48550/arXiv.2011.13824.

[36] P. Yang, R. Li, J. Li, C.-C. Huang, J. Wang, J. Sun, B. Xue, and L. Zhang. Improving Neural Network Verification through Spurious Region Guided Refinement. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 389–408, 2021. ISBN 978-3-030-72016-2. doi:10.1007/978-3-030-72016-2_21.

[37] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel. Efficient Neural Network Robustness Certification with General Activation Functions. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. doi:10.48550/arXiv.1811.00866.

[38] X. Zhang, B. Wang, and M. Kwiatkowska. Provable Preimage Under-Approximation for Neural Networks. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 3–23. Springer Nature, 2024. ISBN 978-3-031-57256-2. doi:10.1007/978-3-031-57256-2_1.

[39] X. Zhang, B. Wang, M. Kwiatkowska, and H. Zhang. PREMAP: A Unifying PREiMage APproximation Framework for Neural Networks. *arXiv:2408.09262*, Aug. 2024. doi:10.48550/arXiv.2408.09262.