Verification of Geometric Robustness of Neural Networks via Piecewise Linear Approximation and Lipschitz Optimisation

Ben Batten, Yang Zheng, Alessandro De Palma, Panagiotis Kouvaros, Alessio Lomuscio

Verification of Geometric Robustness of Neural Networks via Piecewise Linear Approximation and Lipschitz Optimisation

Ben Batten[a,*], **Yang Zheng**[b], **Alessandro De Palma**[c], **Panagiotis Kouvaros**[e,d] **and Alessio Lomuscio**[a,e]

[a]Imperial College London, UK
[b]University of California San Diego, USA
[c]Inria, École Normale Supérieure, PSL University, CNRS, France
[d]Department of Information Technologies, University of Limassol, Cyprus
[e]Safe Intelligence, UK

Abstract. We address the problem of verifying neural networks against geometric transformations of the input image, including rotation, scaling, shearing, and translation. The proposed method computes provably sound piecewise linear constraints for the pixel values by using sampling and linear approximations in combination with branch-and-bound Lipschitz optimisation. The method obtains provably tighter over-approximations of the perturbation region than the present state-of-the-art. We report results from experiments on a comprehensive set of verification benchmarks on MNIST and CIFAR10. We show that our proposed implementation resolves up to 32% more verification cases than present approaches.

1 Introduction

Neural networks as used in mainstream applications, including computer vision, are known to be fragile and susceptible to adversarial attacks [18]. The area of *formal verification of neural networks* is concerned with the development of methods to establish whether a neural network is *robust*, with respect to its classification output, to variations of the image. A large body of literature has so far focused on norm-bounded input perturbations, aiming to demonstrate that imperceptible adversarial alterations of the pixels cannot alter the classifier's classification (ℓ_p robustness). In safety-critical applications such as autonomous driving, however, resistance to norm-bounded perturbations is inadequate to guarantee safe deployment. In fact, image classifiers need to be robust against a number of variations of the image, including contrast, luminosity, hue, and beyond. A particularly important class of specifications concerns robustness to geometric perturbations of the input image [1, 23, 28, 33]. These may include translation, shearing, scaling, and rotation.

Owing to the highly nonlinear variations of the pixels in geometric transformations, verifying robustness to these perturbations is intrinsically a much harder problem than ℓ_p robustness. Previous work over-approximates these variations through hyper-rectangles [33] or pairs of linear bounds over the pixel values [1], hence failing to capture most of the complexity of the perturbation region. Developing more precise methods for verifying geometric robustness remains an open challenge. In this paper we work towards this end. Specifically, we make three contributions:

1. We present a piecewise linear relaxation method to approximate the set of images generated by geometric transformations, including rotation, translation, scaling, and shearing. This construction can incorporate previous approaches [1, 33] as special cases while supporting additional constraints, allowing significantly tighter over-approximations of the perturbation region.
2. We show that sound piecewise linear constraints, the building blocks of the proposed relaxation, can be generated via suitable modifications of a previous approach [1] that generates linear constraints using sampling, linear and Lipschitz optimisation. We derive formal results as well as effective heuristics that enable us to improve the efficiency of the linear and Lipschitz optimisations in this context (cf. Propositions 1—3). As we demonstrate, the resulting piecewise constraints can be readily used within existing tight neural network verifiers.
3. We introduce an efficient implementation for the verification method above and discuss experimental results showing considerable gains in terms of verification accuracy on a comprehensive set of benchmark networks.

The rest of this paper is organized as follows: Section 2 discusses related work. In Section 3 we introduce the problem of verifying neural networks against geometric robustness properties. In Section 4 we present our novel piecewise linear approximation strategy via sampling, optimisation and shifting. In Section 5 we discuss the experimental results obtained and contrast the present method against the state-of-the-art on benchmark networks. We conclude in Section 6. Our code is publicly available on GitHub[1].

2 Related Work

We here briefly discuss related work from l_p-based neural network verification, geometric robustness and formal verification thereof.

[*] Corresponding author. Email: b.batten@imperial.ac.uk.

[1] https://github.com/benbatten/PWL-Geometric-Verification

ℓ_p robustness verification. There is a rich body of work on the verification of neural networks against ℓ_p-bounded perturbations: see, e.g., [27] for a survey. Neural network verifiers typically rely on Mixed-Integer Linear Programming (MILP) [3, 36], branch-and-bound [4, 5, 7, 13, 19, 39, 42, 46], or on abstract interpretation [17, 32, 34]. These methods cannot be used to certify geometric robustness out of the box, as ℓ_p balls are unable to accurately represent geometric transformations [23, 33].

Geometric robustness. The vulnerability of neural networks to geometric transformations has been observed in [10, 11]. A common theme among these works is their *quantitative* nature, whereby measures of invariance to geometric robustness are discussed [22] and methods to improve spatial robustness are developed. These are based on augmentation [16, 41], regularisation schemes [43], robust optimisation [10] and specialised, invariance-inducing network architectures [20]. Differently from the cited works, our key aim here is the *qualitative analysis* of networks towards establishing formal guarantees of geometric robustness.

Formal verification of geometric robustness. One of the earliest works [29] on this subject discretises the transformation domains, enabling robustness verification through the evaluation of the model at a finite number of discretised transformations. In contrast to Pei et al. [29], we here focus on continuous domains, which do not allow exhaustive evaluation. Previous work on continuous domains relies on over-approximations, whereby, for each pixel, the set of allowed values under the perturbation is replaced by a convex relaxation [1, 23, 33]. In particular, [23] and [33] use an l_∞ norm ball and intervals respectively, resulting in loose over-approximations. Balunović et al. [1] devise more precise convex relaxations by computing linear approximations with respect to the transformation parameters. In this work we further improve precision over Balunović et al. [1] by deriving piecewise linear approximations. While the above works consider the geometric transformation as a whole (see Section 3), Mohapatra et al. [28] decompose the transformation into network layers to be pre-pended to the network under analysis, resulting in looser approximations when using standard neural network verifiers. More recently, randomised smoothing techniques have been investigated for geometric robustness [14, 15, 26]: differently from our work, these only provide probabilistic certificates. Finally, Yang et al. [44] recently presented a method to train networks more amenable to geometric robustness verification. Our work is agnostic to the training scheme: we here focus on the more challenging general case.

3 Geometric robustness verification

Our main contribution is a new piecewise linear relaxation of geometric transformations to verify robustness of neural networks to geometric perturbations. We here introduce relevant notation in the verification problem and present the geometric attack model.

Notation. Given two vectors $a, b \in \mathbb{R}^n$, we use $a \geq b$ and $a \leq b$ to represent element-wise inequalities. Given a vector $a \in \mathbb{R}^m$ and a matrix $A \in \mathbb{R}^{m \times n}$, we denote their elements using $a[i]$ and $A[i, j]$, respectively.

Neural networks for classification. We consider a feedforward neural network with L hidden layers $f : \mathbb{R}^n \to \mathbb{R}^m$. Let $x_0 \in \mathbb{R}^n$ denote the input and x_i denotes the activation vectors at layer i. We use $\mathbb{L}_i$ to denote an affine map at layer i, *e.g.*, linear, convolutional, and average pooling operations. Let σ_i be an element-wise activation function, such as ReLU, sigmoid or tanh. The activation vectors are related by $x_{i+1} = \sigma_i\big(\mathbb{L}_i(x_i)\big), i = 0, 1, \ldots, L-1$. We are interested in neural networks for classification: the network output $f(x_0) = \mathbb{L}_L(x_L) \in \mathbb{R}^m$ represents the score of each class, and the label i^* assigned to the input x_0 is the class with highest score, *i.e.*, $i^* = \arg\max_{i=1,\ldots m} f(x_0)[i]$.

Robustness verification. Let $\mathcal{A}$ be a general attacker that takes a nominal input $\bar{x} \in \mathbb{R}^n$ and returns a perturbed input $\mathcal{A}(\bar{x}) \in \mathbb{R}^n$. We denote the attack space as $\Omega_\epsilon(\bar{x}) \subset \mathbb{R}^n$, *i.e.*, $\mathcal{A}(\bar{x}) \in \Omega_\epsilon(\bar{x})$, where $\epsilon > 0$ denotes the attack budget. Formally verifying that a classification neural network f is *robust* with respect to an input $\bar{x}$ and its attack space $\Omega_\epsilon(\bar{x})$ implies ensuring that all points in $\Omega_\epsilon(\bar{x})$ will share the same classification label of $\bar{x}$. This can be done by solving the following optimisation problem $\forall\ i \neq i^*$:

$$
\begin{aligned}
\gamma_i^* := \min_{x_0, x_1, \ldots x_L, y} \quad & y[i^*] - y[i] \\
\text{subject to} \quad & x_0 \in \Omega_\epsilon(\bar{x}), && \text{(1a)} \\
& x_{i+1} = \sigma_i\big(\mathbb{L}_i(x_i)\big), i \in [L] && \text{(1b)} \\
& y = \mathbb{L}_L(x_L), && \text{(1c)}
\end{aligned}
$$

with (1b) being neural network constraints, (1c) as the neural network output, (1a) denoting the attack model constraint, and $L := \{0, 1, \ldots, L-1\}$. If $\gamma_i^* > 0\ \forall\ i \in \{1, \ldots, m\}$, the network is certified to be robust.

Even when $\Omega_\epsilon(\bar{x})$ is a convex set, such as in the case of ℓ_p perturbations, for which $\Omega_\epsilon(\bar{x}) = \{x \in \mathbb{R}^n \mid \|x - \bar{x}\|_p \leq \epsilon\}$, the nonconvex neural network constraints (1b) make the verification problem (1) difficult to solve. However, in this setting, tractable lower bounds $\underline{\gamma}_i^* \leq \gamma_i^*$ on the solution can be obtained through a variety of techniques, including: linear relaxations [9, 31, 33, 35, 37, 45], semi-definite programming [2, 6, 12, 30] and Lagrangian duality [4, 7, 8, 40, 46]. These techniques lie at the core of the network verifiers described in Section 2. If $\underline{\gamma}_i^* > 0\ \forall\ i \in \{1, \ldots, m\}$, the network is robust, but a negative lower bound will leave the property undecided, pointing to the importance of tight lower bounds. When considering geometric transformations, the attack model constraint (1a) is highly nonconvex, making verification even more challenging.

Attack model via geometric transformation. A geometric transformation of an image is a composite function, consisting of a spatial transformation $\mathcal{T}_\mu$, a bilinear interpolation $\mathcal{I}(u, v)$, which handles pixels that are mapped to non-integer coordinates, and changes in brightness and contrast $\mathcal{P}_{\alpha,\beta}$. The spatial transformation $\mathcal{T}_\mu$ can be a composition of rotation, translation, shearing, and scaling; see *e.g.*, [1] for detailed descriptions. The pixel value $\hat{p}_{u,v}$ at position (u, v) of the transformed image is obtained as follows: (1) the pre-image of (u, v) is calculated under $\mathcal{T}_\mu$; (2) the resulting coordinate is interpolated via $\mathcal{I}$ to obtain a value ξ; (3) $\mathcal{P}_{\alpha,\beta}(\xi) = \alpha\xi + \beta$ is applied to compute the final pixel value $\hat{p}_{u,v}$. In other words, we have that $\hat{p}_{u,v} = \mathcal{G}_{u,v}(\alpha, \beta, \mu)$, where:

$$\mathcal{G}_{u,v}(\alpha, \beta, \mu) := \mathcal{P}_{\alpha,\beta} \circ \mathcal{I} \circ \mathcal{T}_\mu^{-1}(u, v). \tag{2}$$

We consider the following standard bilinear interpolation:

$$\mathcal{I}(u, v) = \sum_{\delta_i, \delta_j \in \{0,1\}} p_{i+\delta_i, j+\delta_j} (1 - |i + \delta_i - u|)(1 - |j + \delta_j - v|),$$

where (i, j) denotes the lower-left corner of the interpolation region $[i, i + 1] \times [j, j + 1]$ that contains pixel (u, v), and the matrix p denotes the pixel values of the original image. Note that the interpolation function $\mathcal{I}$ is continuous on $\mathbb{R}^2$ but can be nonsmooth on the

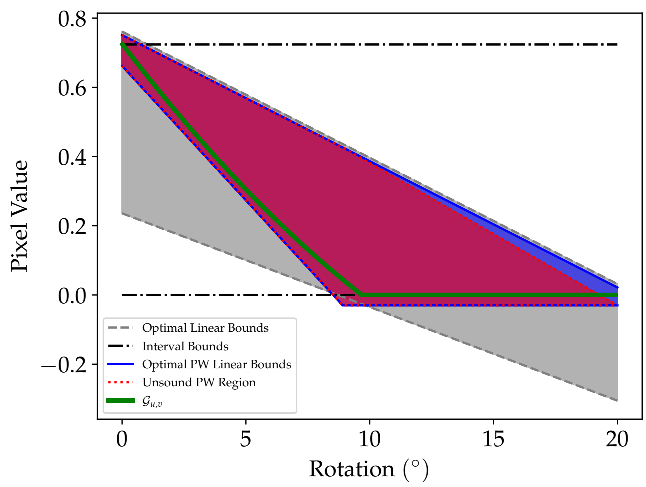

Figure 1. Comparison of sound and unsound piecewise (PW) linear domains (our work), sound linear domain (gray area) [33], and interval bounds (dashed line) [33]. The true pixel value function (the green curve) is marked for a rotation of 18°.

boundaries of interpolation regions. Thus, $\mathcal{G}_{u,v}(\alpha,\beta,\mu)$ is in general *nonsmooth* with respect to the spatial parameter μ (*e.g.*, rotation).

For simplicity, in the following we will denote the transformation parameter as $\kappa = (\alpha,\beta,\mu)$. The geometric attack model assumes interval constraints on (α,β,μ), denoted by $\mathcal{B} \subset \mathbb{R}^d$, where d is the dimension of κ. The attack space $\Omega_\epsilon(\bar{x})$ from (1a) is then defined as the set of all images resulting by the application of $\mathcal{G}_{u,v}(\kappa)$ on each pixel (u,v) of $\bar{x}$, for all $\kappa \in \mathcal{B}$. More formally, given $\mathtt{im}:\mathbb{R}^n \to \mathbb{R}^{h\times w}$, a mapping re-arranging images into their spatial dimensions, $\Omega_\epsilon(\bar{x}) = \{x' \in \mathbb{R}^n \mid \mathtt{im}(x')[u,v] \in \Omega_\epsilon(\bar{x})[u,v]\}$, with $\Omega_\epsilon(\bar{x})[u,v] = \{\mathcal{G}_{u,v}(\kappa) \mid \forall\, \kappa \in \mathcal{B}\}$.

Problem statement. The geometric attack model $\Omega_\epsilon(\bar{x})$ defines a highly nonconvex constraint on the admissible image inputs, which is not readily supported by bounding techniques designed for ℓ_p perturbations. As a result, previous work replaces it by over-approximations [1, 33], which allow verification through ℓ_p-based neural network verifiers. Nevertheless, as described in Section 2, their over-approximations are imprecise, resulting in loose lower bounds $\underline{\gamma}_i^*$. In this work, we aim to derive a tighter convex relaxation of the geometric attack model $\Omega_\epsilon(\bar{x})$ based on piecewise linear constraints. By relying on networks verifiers with support for these constraints, we will then show that our approach leads to effective verification bounds for (1).

4 Piecewise linear formulation

As mentioned above, the pixel value function $\mathcal{G}_{u,v}(\kappa)$ at location (u,v) is generally *nonlinear* and *nonsmooth* with respect to the transformation parameters κ. This is one source of difficulty for solving the verification problem (1). In this section, we introduce a new convex relaxation method to derive tight over-approximations of $\mathcal{G}_{u,v}(\kappa)$.

4.1 Piecewise linear bounds

Deriving an interval bound for each pixel (u,v), *i.e.*, $L_{u,v} \le \mathcal{G}_{u,v}(\kappa) \le U_{u,v}$, for all $\kappa \in \mathcal{B}$ and lower and upper bounds $L_{u,v}, U_{u,v} \in \mathbb{R}$, is arguably the simplest way to get a convex relaxation [23, 33]. However, even a small geometric transformation can lead to a large interval bound, making this approach too loose for effective verification.

This naive interval bound approach has been extended in [1], where linear lower and upper bounds were used for each pixel value, *i.e.*,

$$\underline{w}^{\mathsf{T}}\kappa + \underline{b} \le \mathcal{G}_{u,v}(\kappa) \le \overline{w}^{\mathsf{T}}\kappa + \bar{b}, \quad \forall \kappa \in \mathcal{B}. \tag{3}$$

The linear bounds (3), however, can be still too loose to approximate the nonlinear function $\mathcal{G}_{u,v}(\kappa)$ (see Figure 1 for illustration). Our key idea is to use piecewise linear bounds to approximate the pixel values:

$$\max_{j=1,\ldots,q}\{\underline{w}_j^{\mathsf{T}}\kappa + \underline{b}_j\} \le \mathcal{G}_{u,v}(\kappa) \le \min_{j=1,\ldots,q}\{\overline{w}_j^{\mathsf{T}}\kappa + \bar{b}_j\}, \tag{4}$$

$\forall \kappa \in \mathcal{B}$, where q is the number of piecewise segments, $\underline{w}_j \in \mathbb{R}^d, \underline{b}_j \in \mathbb{R}, j = 1,\ldots,q$ define the piecewise linear lower bound, and $\overline{w}_j \in \mathbb{R}^d, \bar{b}_j \in \mathbb{R}, j = 1,\ldots,q$ define the piecewise linear upper bound. We remark that the pixel values constrained by (4) form a convex set. Furthermore, our approach can include the strategies in [1, 33] as special cases. Employing the relative constraints among the piecewise segments will result in a tighter set.

For each pixel value, we would like to derive optimal and sound piecewise linear bounds by minimizing the approximation error. Specifically, we aim to compute the lower bound via

$$\begin{aligned}\min_{\underline{w}_j,\underline{b}_j, j=1,\ldots,q} \quad & \int_{\mathcal{B}} \left(\mathcal{G}_{u,v}(\kappa) - \big(\max_{j=1,\ldots,q}\{\underline{w}_j^{\mathsf{T}}\kappa + \underline{b}_j\}\big)\right) d\kappa \\ \text{s.t.} \quad & \max_{j=1,\ldots,q}\{\underline{w}_j^{\mathsf{T}}\kappa + \underline{b}_j\} \le \mathcal{G}_{u,v}(\kappa), \quad \forall \kappa \in \mathcal{B}.\end{aligned} \tag{5}$$

Computing the upper bound for (4) is similar. This optimisation problem (5) is highly nontrivial to solve since *the integral cost function is hard to evaluate due to the nonlinearity of* $\mathcal{G}_{u,v}(\kappa)$. Motivated by [1], we first sample the transformation parameter κ_i from $\mathcal{B}$ to obtain the sampled pixel values $\mathcal{G}_{u,v}(\kappa_i)$, and then solve a sampled version of (5). The resulting piecewise bound is guaranteed to be sound on the sampling points $\kappa_i \in \mathcal{B}$ but could be unsound on non-sampled points. To derive a final sound piecewise bounds for $\mathcal{G}_{u,v}(\kappa)$, we bound the maximum violation over the entire $\mathcal{B}$ using a branch-and-bound Lipschitz optimisation procedure.

4.2 Linear optimisation based on sampling points

Here, we first randomly select N transformation parameters $\kappa_i \in \mathcal{B}$, $i = 1,\ldots,N$, to obtain a sampled version of (5) as follows

$$\begin{aligned}&\min_{\underline{w}_j,\underline{b}_j, j=1,\ldots,q} \frac{1}{N}\sum_{i=1}^{N}\left(\mathcal{G}_{u,v}(\kappa_i) - \big(\max_{j=1,\ldots,q}\{\underline{w}_j^{\mathsf{T}}\kappa_i + \underline{b}_j\}\big)\right) \\ &\text{subject to} \max_{j=1,\ldots,q}\{\underline{w}_j^{\mathsf{T}}\kappa_i + \underline{b}_j\} \le \mathcal{G}_{u,v}(\kappa_i), i = 1,\ldots,N.\end{aligned} \tag{6}$$

We denote the optimal cost value of (6) as β^*. In (6), the number of piecewise linear segments q is fixed *a priori*. Still, problem (6) is nontrivial to solve jointly for all piecewise segments $\underline{w}_j, \underline{b}_j, j = 1,\ldots,q$ unless $q = 1$ (where (6) is reduced to a single linear program). One difficulty is to determine the effective domain of each piecewise linear segment.

To alleviate this, we propose to split the whole domain $\mathcal{B}$ into q sub-domains $\mathcal{B}_1,\ldots,\mathcal{B}_q$, and then optimize each piecewise linear segment over $\mathcal{B}_j$, $j = 1, \ldots, q$, individually. We then use the following q independent linear programs to approximate the solution to (6):

$$\begin{aligned} \beta_j^* := \min_{\underline{w}_j, \underline{b}_j} \quad & \frac{1}{N} \sum_{\kappa_i \in \mathcal{B}_j} \Big(\mathcal{G}_{u,v}(\kappa_i) - \big(\underline{w}_j^\mathsf{T} \kappa_i + \underline{b}_j\big) \Big) \\ \text{subject to} \quad & \underline{w}_j^\mathsf{T} \kappa_i + \underline{b}_j \leq \mathcal{G}_{u,v}(\kappa_i), \ i = 1, \ldots, N, \end{aligned} \tag{7}$$

for $j = 1, \ldots, q$. Note that in (7), we minimise the approximation error over only the sample points within a given domain $\mathcal{B}_j$; however, we force each segment to satisfy the constraints at every sample point $\kappa_i \in \mathcal{B}$ over the whole domain.

We have the following result for the quality of the solution from (7).

Proposition 1. *Given any subdomains $\mathcal{B}_j$, $j = 1, \ldots, q$, the optimal solutions $\underline{w}_j, \underline{b}_j$, $j = 1, \ldots, q$, to* (7) *are suboptimal to* (6)*, i.e., $\sum_{j=1}^q \beta_j^* \geq \beta^*$. There exists a set of subdomains $\mathcal{B}_j$, $j = 1, \ldots, q$, such that the optimal solutions to* (6) *and* (7) *are identical, i.e., $\sum_{j=1}^q \beta_j^* = \beta^*$.*

Proof. Consider the piecewise linear function in the objective function (6). Let $\mathcal{B}_j, j = 1, \ldots, q$ be the effective piecewise domain of the jth segment, *i.e.*,

$$\max_{j=1,\ldots,q} \{\underline{w}_j^\mathsf{T} \kappa_i + \underline{b}_j\} = \begin{cases} \underline{w}_1^\mathsf{T} \kappa_i + \underline{b}_1, & \text{if} \quad \kappa_i \in \mathcal{B}_1 \\ \vdots & \\ \underline{w}_q^\mathsf{T} \kappa_i + \underline{b}_q, & \text{if} \quad \kappa_i \in \mathcal{B}_q. \end{cases} \tag{8}$$

Then, the objective function (6) can be equivalently written into

$$\begin{aligned} & \frac{1}{N} \sum_{i=1}^{N} \Big(\mathcal{G}_{u,v}(\kappa_i) - \big(\max_{j=1,\ldots,q} \{\underline{w}_j^\mathsf{T} \kappa_i + \underline{b}_j\} \big) \Big) \\ = & \frac{1}{N} \sum_{j=1}^{q} \left(\sum_{\kappa_i \in \mathcal{B}_j} \Big(\mathcal{G}_{u,v}(\kappa_i) - \big(\underline{w}_j^\mathsf{T} \kappa_i + \underline{b}_j\big) \Big) \right) \end{aligned}$$

Therefore, (6) is equivalent to

$$\begin{aligned} & \min_{\underline{w}_j, \underline{b}_j, \mathcal{B}_j, j=1,\ldots,q} \\ & \qquad \sum_{j=1}^{q} \left(\frac{1}{N} \sum_{\kappa_i \in \mathcal{B}_j} \Big(\mathcal{G}_{u,v}(\kappa_i) - \big(\underline{w}_j^\mathsf{T} \kappa_i + \underline{b}_j\big) \Big) \right) \\ & \textbf{s.t.} \quad \underline{w}_j^\mathsf{T} \kappa_i + \underline{b}_j \leq \mathcal{G}_{u,v}(\kappa_i), \quad i = 1, \ldots, N, j = 1, \ldots q. \end{aligned} \tag{9}$$

Note that the piecewise domains $\mathcal{B}_j$ are determined by the linear segments $\underline{w}_j, \underline{b}_j, j = 1, \ldots, q$ implicitly in (8). We need to simultaneously optimize the choices of $\mathcal{B}_j$ in (10), making it computationally hard to solve.

A suboptimal solution for (10) is to *a priori* fix the effective domain $\mathcal{B}_j$ and optimize over $\underline{w}_j, \underline{b}_j, j = 1, \ldots, q$ only, i.e.,

$$\begin{aligned} & \hat{\beta} := \\ & \min_{\underline{w}_j, \underline{b}_j, j=1,\ldots,q} \sum_{j=1}^{q} \left(\frac{1}{N} \sum_{\kappa_i \in \mathcal{B}_j} \Big(\mathcal{G}_{u,v}(\kappa_i) - \big(\underline{w}_j^\mathsf{T} \kappa_i + \underline{b}_j\big) \Big) \right) \\ & \textbf{s.t.} \quad \underline{w}_j^\mathsf{T} \kappa_i + \underline{b}_j \leq \mathcal{G}_{u,v}(\kappa_i), \quad i = 1, \ldots, N, j = 1, \ldots q, \end{aligned} \tag{10}$$

which is decoupled into q individually linear programs, $j = 1, \ldots, q$

$$\begin{aligned} \beta_j^* := \min_{\underline{w}_j, \underline{b}_j} \quad & \frac{1}{N} \sum_{\kappa_i \in \mathcal{B}_j} \Big(\mathcal{G}_{u,v}(\kappa_i) - \big(\underline{w}_j^\mathsf{T} \kappa_i + \underline{b}_j\big) \Big) \\ \text{subject to} \quad & \underline{w}_j^\mathsf{T} \kappa_i + \underline{b}_j \leq \mathcal{G}_{u,v}(\kappa_i), \ i = 1, \ldots, N. \end{aligned} \tag{11}$$

Therefore, it is clear that $\hat{\beta} = \sum_{j=1}^q \beta_j^* \geq \beta^*$. On the other hand, suppose the optimal solution to (6) leads to the optimal effective domains $\mathcal{B}_j, j = 1, \ldots, q$ in (8). Then, using this set $\mathcal{B}_j, j = 1, \ldots, q$, the decoupled linear programs (11) are equivalent to (10) and (6). □

To obtain a good solution (6), choosing the subdomains $\mathcal{B}_j$ becomes essential. A uniform grid partition is one, naive choice. Another is to partition the subdomains based on the distribution of the sampling points $\mathcal{G}_{u,v}(\kappa_i)$. The details of the splitting procedure are provided in the appendix.

Remark 1. (Explicit input splitting vs. piecewise linear constraints) We note that one can perform explicit input splitting $\mathcal{B}_j, j = 1, \ldots, q$, and verify each of them by solving (1) separately in order to certify the original large domain $\mathcal{B}$. The main drawback of this explicit input splitting is that we need to call a verifier for each subdomain $\mathcal{B}_j$ which can be hugely time consuming and not scalable. On the contrary, it only requires to solve multiple small linear programs (7) to derive our piece-wise linear constraints. Then, we only need to call a verifier once to solve the verification problem (1) over $\mathcal{B}$. For tight verifiers, such as those mentioned in Section 2, this process is much more efficient than explicit input splitting.

4.3 Lipschitz optimisation for obtaining sound piecewise linear bounds

The piecewise linear constraints from (7) are valid for the sampling points $\kappa_i \in \mathcal{B}, i = 1, \ldots, N$. To make the constraints sound over all $\kappa \in \mathcal{B}$, we must shift them such that all points on the pixel value function, $\mathcal{G}_{u,v}(\kappa)$, satisfy the constraints in (4). For this, we define a new function that tracks the violation of a piecewise bound over the entire domain $\mathcal{B}$:

$$\xi_{u,v}^* := \max_{\kappa \in \mathcal{B}} \quad \underline{f}_{u,v}(\kappa), \tag{12}$$

where $\underline{f}_{u,v}(\kappa) = \max_{j=1,\ldots,q}\{\underline{w}_j^\mathsf{T} \kappa + \underline{b}_j\} - \mathcal{G}_{u,v}(\kappa)$. Then, we naturally have a sound piecewise linear lower bound as

$$\max_{j=1,\ldots,q} \{\underline{w}_j^\mathsf{T} \kappa + \underline{b}_j\} - \xi_{u,v}^* \leq \mathcal{G}_{u,v}(\kappa), \ \forall \kappa \in \mathcal{B}.$$

However, computing the exact maximum ξ^* is computationally hard due to the *nonconvexity, nonlinearity and nonsmoothness* of $\underline{f}_{u,v}(\kappa)$. Instead, given any $\epsilon > 0$, we can use a branch-and-bound Lipschitz optimisation procedure to find $\underline{\xi}^* \in \mathbb{R}$ satisfying $\underline{\xi}^* \leq \xi_{u,v}^* \leq \underline{\xi}^* + \epsilon$.

To establish the branch-and-bound Lipschitz optimisation procedure, we need to characterise the properties of the violation function $\underline{f}_{u,v}(\kappa)$.

Proposition 2. *The violation function $\underline{f}_{u,v}(\kappa) := \max_{j=1,\ldots,q}\{\underline{w}_j^\mathsf{T} \kappa + \underline{b}_j\} - \mathcal{G}_{u,v}(\kappa)$ is nonconvex, nonsmooth, and Lipschitz continuous over $\mathcal{B} \subset \mathbb{R}^d$. Furthermore, there exist $L_m > 0$, $m = 1, \ldots, d$, such that $\forall \kappa_1, \kappa_2 \in \mathcal{B}$*

$$|\underline{f}_{u,v}(\kappa_1) - \underline{f}_{u,v}(\kappa_2)| \leq \sum_{m=1}^{d} L_m |\kappa_1(m) - \kappa_2(m)|. \tag{13}$$

Proof. The pixel value function is given by $\mathcal{G}_{u,v}(\kappa) := \mathcal{P}_{\alpha,\beta} \circ \mathcal{I} \circ \mathcal{T}_\mu^{-1}(u,v)$. We know that the spatial transformation $\mathcal{T}_\mu(u,v)$ and $\mathcal{P}_{\alpha,\beta}$ are continuous and differentiable everywhere. The interpolation function $\mathcal{I}(u,v)$ is continuous everywhere, but it is only differentiable within each interpolation region and it can be nonsmooth on the boundary. Also, $\mathcal{T}_\mu(u,v)$ and $\mathcal{I}(u,v)$ are generally nonconvex.

In addition, the piecewise linear function $\max_{j=1,\dots,q}\{\underline{w}_j^\top \kappa + \underline{b}_j\}$ is continuous but not differentiable everywhere. Therefore, the violation function $\underline{f}_{u,v}(\kappa)$ is nonconvex and nonsmooth in general. Finally, all the functions $\mathcal{T}_\mu(u,v)$, $\mathcal{P}_{\alpha,\beta}$, $\mathcal{I}(u,v)$ and $\max_{j=1,\dots,q}\{\underline{w}_j^\top \kappa + \underline{b}_j\}$ are Lipschitz continuous, so is the violation function $\underline{f}_{u,v}(\kappa)$. Thus, there exist $L_m > 0$, $m = 1,\dots,d$, such that (13) holds. □

The properties of the violation function $\underline{f}_{u,v}(\kappa)$ in Proposition 2 are directly inherited from nonconvexity and nonsmoothness of the interpolation function $\mathcal{I}(u,v)$. The Lipschitz continuity is also from the interpolation function and the piecewise linear function.

With the information of L_m in (13), we are ready to get a lower and an upper bound for ξ^* upon evaluating the function at any point $\kappa_0 \in \mathcal{B}$:

$$\begin{aligned} \underline{f}_{u,v}(\kappa_0) &\le \xi^* \\ &= \max_{\kappa\in\mathcal{B}} \underline{f}_{u,v}(\kappa) \\ &\le \max_{\kappa\in\mathcal{B}} \underline{f}_{u,v}(\kappa_0) + \sum_{m=1}^{d} L_m|\kappa(m)-\kappa_0(m)| \\ &\le \underline{f}_{u,v}(\kappa_0) + \sum_{m=1}^{d} L_m h_m, \end{aligned} \tag{14}$$

where $h_m > 0$ denotes the difference of the lower and upper bound in each box constraint of $\mathcal{B}$. These lower and upper bounds (14) are useful in the branch-and-bound procedure.

Still, we need estimate the Lipschitz constant L_m in (13). In our work, we show how to estimate the constant L_m based on the gradient of $\underline{f}_{u,v}(\kappa)$ whenever it is differentiable (note that $\underline{f}_{u,v}(\kappa)$ is not differentiable everywhere)

Proposition 3. *Let* $\mathrm{Diff}(\mathcal{B})$ *be the subset of* $\mathcal{B}$ *where* $\underline{f}_{u,v}(\kappa)$ *is differentiable. Then, the Lipschitz constants in* (13) *can be chosen as* $L_m = \sup_{\kappa\in\mathrm{Diff}(\mathcal{B})} |\nabla \underline{f}_{u,v}^\top e_m|$, *where* $e_m \in \mathbb{R}^d$ *is a basis vector with only the* m*-th element being one and the rest being zero.*

Proof. This proof is motivated by [21]. In order to prove Proposition 3, we first state a useful result from [21, Lemma 3]. Let $f : \mathbb{R}^n \to \mathbb{R}$ be Lipschitz continuous over an open set $\Omega \subset \mathbb{R}^n$. We denote $\mathrm{Diff}(\Omega)$ as the subset of Ω where $f(x)$ is differentiable. We also let $\mathcal{D}$ be the set of $(x,v) \in \mathbb{R}^{2n}$ for which the directional derivative, $\nabla_v f(x)$, exists and $x \in \Omega$. Finally, we let $\mathcal{D}_v$ be the set $\mathcal{D}_v = \{x \in \mathbb{R}^n \mid (x,v) \in \mathcal{D}\}$. Then, we have the following inequality [21, Lemma 3]

$$\sup_{x\in\mathcal{D}_v} |\nabla_v f(x)| \le \sup_{x\in\mathrm{Diff}(\Omega)} |\nabla f(x)^\top v|. \tag{15}$$

We now proceed to prove Proposition 3. Fix any $\kappa_1, \kappa_2 \in \mathcal{B}$, and we define a function $h : \mathbb{R} \to \mathbb{R}$ as $h(t) = \underline{f}_{u,v}(\kappa_1 + t(\kappa_2 - \kappa_1))$. Since $\underline{f}_{u,v}(\kappa)$ is Lipschitz continuous in $\mathcal{B}$, it is clear that $h(t)$ is Lipschitz continuous on the interval $[0,1]$. Thus, by Rademacher's Theorem, $h(t)$ is differentiable everywhere except for a set of measure zero.

We can further define a Lebesgue integrable function $g(t)$ that equal to $h'(t)$ almost everywhere as follows

$$g(t) = \begin{cases} h'(t), & \text{if} \quad h'(t) \quad \text{exists} \\ \sup_{s\in[0,1]} |h'(s)|, & \text{otherwise} \end{cases}.$$

Note that if $\underline{f}_{u,v}(\kappa)$ is differentiable at some point, we have

$$h'(t) = \nabla \underline{f}_{u,v}(\kappa_1 + t(\kappa_2 - \kappa_1))^\top(\kappa_2 - \kappa_1).$$

Then we have the following inequalities

$$\begin{aligned} &|\underline{f}_{u,v}(\kappa_1) - \underline{f}_{u,v}(\kappa_2)| = |h(1) - h(0)| = \left|\int_0^1 g(t)dt\right| \\ &\le \int_0^1 |g(t)|dt \\ &\le \int_0^1 \sup_{s\in[0,1]} |h'(s)|dt = \sup_{s\in[0,1]} |h'(s)| \\ &\le \sup_{\kappa\in\mathcal{D}_{\kappa_2-\kappa_1}} |\nabla_{\kappa_2-\kappa_1} \underline{f}_{u,v}(\kappa)|. \end{aligned}$$

Furthermore, considering the inequality in (15) [21, Lemma 3], we have

$$\begin{aligned} &|\underline{f}_{u,v}(\kappa_1) - \underline{f}_{u,v}(\kappa_2)| \le \sup_{\kappa\in\mathrm{Diff}(\mathcal{B})} |\nabla \underline{f}_{u,v}(\kappa)^\top(\kappa_2 - \kappa_1)| \\ &\le \sum_{m=1}^{d} \sup_{\kappa\in\mathrm{Diff}(\mathcal{B})} |\nabla \underline{f}_{u,v}(\kappa)^\top e_m||\kappa_2(m) - \kappa_1(m)| \end{aligned}$$

where $e_m \in \mathbb{R}^d$ is a basis vector with only the m-th element being one and the rest being zero. Therefore, the Lipschitz constants in (13) can be chosen as $L_m = \sup_{\kappa\in\mathrm{Diff}(\mathcal{B})} |\nabla \underline{f}_{u,v}(\kappa)^\top e_m|$. □

Maximum directional gradient. To bound the maximum violation $\xi_{u,v}^*$ in (12) using (14), we need to estimate the constant L_m, and Proposition 3 requires us to calculate the maximum directional gradient $|\nabla \underline{f}_{u,v}^\top e_m|$. Each component of $\nabla \underline{f}_{u,v}$ varies independently with respect to any constituent of the transformation composition, $\mathcal{T}_\mu(\kappa_m)$, $m = 1,\dots,d$. Each L_m depends only on a transformation, $\mathcal{T}_\mu$, and interpolation, $\mathcal{I}_{u,v}$. The only component that is not differentiable everywhere in the parameter space $\kappa \in \mathcal{B}$, is interpolation $\mathcal{I}_{u,v}(x,y)$ - this due to it being disjoint across interpolation regions. We overcome this by calculating the interpolation gradient, $\nabla_{x,y}\mathcal{I}_{u,v}$ separately in each interpolation region, and taking the maximum interval of gradients from the union, $[\nabla_{x,y}I_{min}, \nabla_{x,y}I_{max}] = [\min(\cup_{k=1,\dots,n}\nabla_{x,y}I_k), \max(\cup_{k=1,\dots,n}\nabla_{x,y}I_k)]$, where I_k are the relevant interpolation regions, and $\cup_{k=1,\dots,n}I_k = \mathcal{R} \subset \mathcal{B}$. Computing a bound on L_m this way mirrors the IBP-based procedure outlined in [1]. With this we can calculate an upper bound on L_m to be applied in the Lipschitz algorithm.

Branch-and-bound Lipschitz optimisation procedure. Similar to [1], we use a branch-and-bound procedure (See Appendix) where $\underline{f}_{u,v}$ and $\mathcal{B}$ are given as inputs alongside the Lipschitz error, ϵ, and samples per subdomain, n. The procedure first samples the violation function $\underline{f}_{u,v}$, obtaining maximum value candidates, this is placed in a list of 3-tuples with the upper bound, $\underline{f}_{\mathrm{bound},i}$, and corresponding domain, $\mathcal{B}_i$. The key upper bound operation $\mathrm{bound}(\cdot)$ is obtained using (14). We then check whether each 3-tuple in our list meets the termination criteria, as parameterised by ϵ. If the requirement is satisfied for all elements then we terminate and return $\underline{\xi}^*$. Until the requirement is met for every list element we iteratively split unsatisfied subdomains. This process is repeated until a satisfactory maximum candidate is found, splitting κ in each iteration. We can ignore any sub-domain, κ_1^n, of κ_1 where the function bound $\underline{f}_{\text{bound}}$ in κ_1^n is smaller than a maximum value candidate $\underline{f}_{\text{max}}$ in any other subdomain. Deciding how to split subdomains is non-trivial for higher dimensional parameter spaces. In the case $\kappa \in \mathbb{R}^1$ we need only decide where to split on a single axis; for which we use the domain midpoint. The crux of our algorithm is approximating the gradient of $\underline{f}_{u,v}(\kappa)$ when it is differentiable, as stated in Proposition 3 (see appendix for further details on the branch-and-bound procedure). For bounding the violation of piecewise linear bounds we can consider the piecewise bound itself to be made of q linear sub-regions with each one bounded by the intersection with the neighbouring linear piece - or the lower and upper bounds on the transformation parameters. We can then bound the Lipschitz constant in the same way as for a single linear bound, instead starting with q sub-domains. Solving the Lipschitz bounding procedure for each linear segment over only its local domain in this way enables us to bound the Lipschitz constant of a piecewise linear bound in the same time as a linear bound takes.

5 Experimental Evaluation

In this section we present three sets of results: (i) a quantitative study directly comparing the model-agnostic bounds produced by our piecewise linear approach against the state-of-the-art linear bounds [1], (ii) an empirical evaluation of verification results obtained using linear and piecewise linear bounds, without input splitting and using the same neural network verifier [3], and (iii) a comparison of our results against the present state-of-the-art method [1].

5.1 *Experimental setup*

We consider the MNIST image recognition dataset [25] and CIFAR10 [24]. In line with the previous literature [1], we use two fully-connected ReLU networks, MLP2 and MLP6, and one convolutional ReLU network, CONV, from the first competition for neural network verification (VNN-COMP) [38]. The fully-connected networks comprise 2 and 6 layers respectively. Each layer of each of the networks has 256 ReLU nodes. The convolutional network comprises two layers. The first layer has 32 filters of size 5×5, a padding of 2 and strides of 2. The second layer has 64 filters of size of 4×4, a padding of 2 and strides of 1. Additionally, we employ a larger convolutional ReLU network from relevant previous work [1], composed of three layers: a convolutional layer with 32 filters of size 4×4 and strides of 2, a convolutional layer with 64 filters of size 4×4 and strides of 2, and a fully connected layer with 200 nodes. All experiments were carried out on an Intel Core i9-10940X (3.30GHz, 28 cores) equipped with 256GB RAM and running Linux kernel 5.12. DeepG experiment ANS: we do not use GPU in these experiments.

Once a convex over-approximation of the attack space $\Omega_\epsilon(\bar{x})$ is computed, (cf. Section 3) a neural network verifier is required to provide a lower bound on problem (1). Unless stated otherwise, the verification results reported in this work are obtained using VENUS, a complete MILP-based verification toolkit for feed-forward neural networks [3].

5.2 *Experimental results*

In the following, we will use “L" to denote the linear relaxation from equation (3), and “PWL" to denote the piecewise linear relaxation

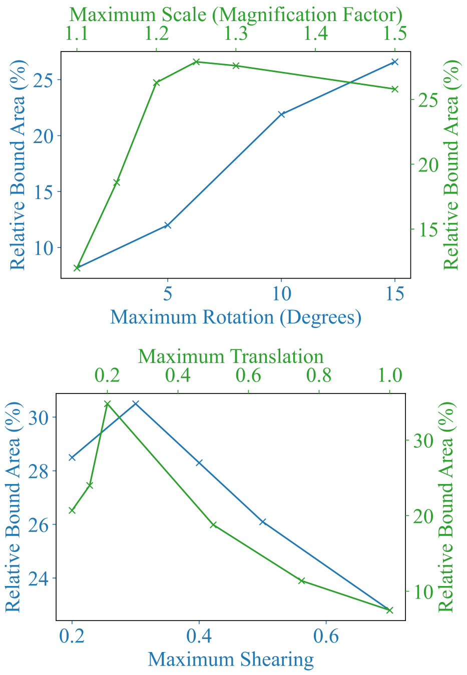

Figure 2. A comparison of area captured by piecewise linear and linear bounds as a function of transformation parameter. Relative bound area is defined as $1 - (V_{\text{PWL}}/V_{\text{L}})$.

Table 1. Comparison of verification results for piecewise linear constraints and linear constraints.

Model	Attack	Verified		Falsified		Time (s)	
		L	PWL	L	PWL	L	PWL
MLP2	R(5)	26	**28**	74	72	0.7	3.7
	Sh(0.2)	20	**26**	80	74	1.2	12
	Sc(1.1)	24	24	76	76	1.1	51
	T(0.1)	16	16	84	84	11	54
MLP6	R(15)	0	**2**	12	32	1602	1253
	Sh(0.5)	0	0	16	**68**	1591	778
	Sc(1.3)	0	0	24	**78**	1404	727
	T(0.2)	0	**2**	26	74	1397	648
CONV	R(10)	20	**48**	2	0	1447	1044
	Sh(0.2)	18	**50**	0	0	1548	1044
	Sc(1.3)	0	**10**	4	4	1750	1663
	T(0.15)	0	**32**	2	0	1767	1397

from equation (4).

PWL vs L: comparing areas. Figure 2 is a direct comparison of bound *tightness* between our piecewise linear bounds and the current state-of-the-art linear bounds [1]. For each image, linear and piecewise linear bounds are generated, each one capturing the reachable pixel values for a given transformation. We always use two piecewise segments ($q = 2$) and use a Lipschitz error of 0.01 to compute bounds. The area enclosed by each set of bounds is then calculated and averaged for every pixel over all images. In each case the piecewise linear bounds are guaranteed to be tighter (enclose a smaller area) than the linear bounds, as in Section 4. Figure 2 shows the relative area (specifically, $1 - (V_{\text{PWL}}/V_{\text{L}})$ with V_{PWL} and V_{L} being the volume enclosed by the piecewise linear and linear bounds, respectively) of the two bound types. In Figure 2, there is an initial increase in relative tightness for all transformations – this is a result of linear bounds being unable to efficiently capture the increasing nonlinearity in the pixel value curve, $\mathcal{G}_{u,v}(\kappa)$. After an initial increase,

Table 2. Comparison of L and PWL using VENUS, with verification results taken from DeepG [1].

Dataset	Transformation	Accuracy (%)	DeepG	Linear (Ours)		PWL (Ours)	
			Certified (%)	Certified (%)	Time (s)	Certified (%)	Time (s)
MNIST	R(30)	99.1	87.8	90.8	37.9	92.9	28.3
CIFAR	R(2)Sh(2)	68.5	54.2	65.0	239.5	66.0	204.9

the behaviour for different transformations diverges. For rotation, the relative advantage of the piecewise bounds continues to increase up to 15 degrees. For scaling, however, there is a peak at $1.25\times$ magnification, followed by a decrease in the relative tightness. This result is explained by a corresponding increase in the complexity of the pixel value curve. Notably, the piecewise bounds are best suited to non-monotonic pixel value curves with a single, sharp vertex. For curves with many vertices and large fluctuations, piecewise linear bounds become increasingly linear (the gradient of the pieces converge) to maintain convexity. Though this is the case for $q = 2$, as we study here, for larger numbers of piecewise segments the advantage over linear bounds will continue to hold, as the piecewise bounds approximate the convex hull of the pixel values for $q \to \infty$. The plots for shearing and translation show a similar pattern to scaling. Although the relative tightness may decrease for larger transformations, the total bounded area increases, making any proportional reduction in area more significant.

PWL vs L: verification results. Table 1 reports the experimental results obtained for verification queries using VENUS, on the VNN-COMP networks. For each type of input bound – piecewise linear and linear – the table shows the percentage of certified images (Verified column), the percentage of images for which a valid counter example was found (Falsified column), and the average verification time. We verify the robustness of each of the networks with respect to one of four transformations - rotation, scaling, shearing, or translation - on 50 randomly selected images from the MNIST test set. For each verification query we use a timeout of 30 minutes. We observe a considerable performance advantage using piecewise linear bounds for the convolution network, in every case, at least doubling the count of verifiable images. For the 6-layer MLP network, many of the transformations tried could not be verified, leading to numerous counter examples and time-outs. However, for every transformation the piecewise linear bounds were able to find more counter examples than linear bounds – this is a result of the improved tightness of piecewise linear bounds. For the 2-layer MLP, results across the bound types are very similar, in some cases they are equal. This is due to two factors, both of which stem from the network's small size. Firstly, the 2-layer network is the least robust of all three. Accordingly, our results are for very small transformations for which the pixel value curve is approximately linear. In these cases, linear bounds can capture the input set as well as piecewise linear bounds. Secondly, the advantage of piecewise linear bounds' tightness is compounded over each layer of a network – the 2-layer MLP is so small that this effect is minimal, further aligning the performance of the approaches. Finally, the use of piecewise linear constraints result in a reduction of average verification times on both the 6-layer MLP and the convolutional network: this is due to the fact that their relative tightness compensates for the additional cost of their encoding, leading the employed MILP-based verifier to positive lower bounds on the verification problem (1) in less time.

Comparison with literature results. In Table 2 we provide a comparison of verification results obtained using VENUS with both linear and piecewise linear constraints, with the DeepG [1] results, obtained using linear constraints and the DeepPoly [33] verifier, which relies on a relatively loose LP relaxation of (1). Further, we use a MILP-based verifier which enabled us to add the pixel domain constraints in addition to our transformation-based bounds. This, coupled with the tighter verifier, enables our linear bounds to out-perform those from DeepG. We consider MNIST and CIFAR10 benchmark presented in Balunović et al. [1]. The MNIST example consists of verifying a 30 degree rotation transformation by way of 10, 3-degree sub-problems. This is in contrast to Table 1, where each perturbation is represented by a single set of bounds and a single verifier call per image. Table 2 shows that, even under the small-perturbation setting, the use of tighter verification algorithms (L versus DeepG) increases the number of verified properties. Furthermore, we show that the method proposed in this work, PWL, leads to the tightest certification results. The CIFAR10 example comprises a composition of rotation and shearing of 2 degrees and 2% respectively. This query is solved via 4 sub-problems (with each transformation domain split in half). The results show a 12% improvement for the PWL bounds over the DeepG result. However, much of this gain comes from the verifier itself. The gap between the linear bounds and their piecewise counterpart is 1%. We attribute this smaller gap to the relatively small domain over which each sub-problem runs. Nevertheless, we point out that verifying perturbations through a series of sub-problems is extremely expensive, as it requires repeated calls to both neural network verifiers, and to the constraint-generation procedure (including the branch-and-bound-based Lipschitz optimisation). For this reason, we focus on verification the setting without transformation splitting, and aim to maximize certifications through the use of tight verifiers and over-approximations of the geometric transforms.

6 Conclusions

We have introduced a new piecewise linear approximation method for geometric robustness verification. Our approach can generate provably tighter convex relaxations for images obtained by geometric transformations than the state-of-the-art methods [1, 33]. Indeed, we have shown experimentally that the proposed method can provide better verification precision in certifying robustness against geometric transformations than prior work [1], while being more computational efficient.

Despite the positive results brought by our piecewise linear approximation method, further topics deserve further exploration. Firstly, it remains challenging to obtain the optimal piecewise linear constraints via (6). To get a good set of piecewise linear constraints, our current method (7) requires to obtain a good heuristic partition of the domain $\mathcal{B}_1, \ldots, \mathcal{B}_q$. It will be interesting to further investigate and quantify the suboptimality of the solution from (7). Second, the number of piecewise linear segment q is a hyperparameter in our framework. A larger value q leads to a better approximation of the pixel value function in theory; however, this also results in more linear constraints for the verification problem in practice. Future work will investigate how to choose a good value of q based on the curvature of of the pixel value function.

Acknowledgements

Ben Batten was funded by the UKRI Centre for Doctoral Training in Safe and Trusted Artificial Intelligence. Alessandro De Palma was supported by the "SAIF" project, funded by the "France 2030" government investment plan managed by the French National Research Agency, under the reference ANR-23-PEIA-0006. Alessio Lomuscio was supported by a Royal Academy of Engineering Chair in Emerging Technologies.

References

[1] M. Balunović, M. Baader, G. Singh, T. Gehr, and M. Vechev. Certifying geometric robustness of neural networks. *NeurIPS19*, 2019.

[2] B. Batten, P. Kouvaros, A. Lomuscio, and Y. Zheng. Efficient neural network verification via layer-based semidefinite relaxations and linear cuts. In *IJCAI21*, pages 2184–2190, 2021.

[3] E. Botoeva, P. Kouvaros, J. Kronqvist, A. Lomuscio, and R. Misener. Efficient verification of relu-based neural networks via dependency analysis. In *AAAI20*, volume 34, pages 3291–3299, 2020.

[4] R. Bunel, A. De Palma, A. Desmaison, K. Dvijotham, P. Kohli, P. H. Torr, and M. P. Kumar. Lagrangian decomposition for neural network verification. In *Conference on Uncertainty in Artificial Intelligence*, 2020.

[5] R. Bunel, I. Turkaslan, P. Torr, M. P. Kumar, J. Lu, and P. Kohli. Branch and bound for piecewise linear neural network verification. *Journal of Machine Learning Research*, 21, 2020.

[6] S. Dathathri, K. Dvijotham, A. Kurakin, A. Raghunathan, J. Uesato, R. Bunel, S. Shankar, J. Steinhardt, I. Goodfellow, P. Liang, and K. Pushmeet. Enabling certification of verification-agnostic networks via memory-efficient semidefinite programming. *NeurIPS20*, 2020.

[7] A. De Palma, H. S. Behl, R. Bunel, P. H. S. Torr, and M. P. Kumar. Scaling the convex barrier with active sets. In *International Conference on Learning Representations*, 2021.

[8] K. Dvijotham, R. Stanforth, S. Gowal, T. Mann, and P. Kohli. A dual approach to scalable verification of deep networks. In *Conference on Uncertainty in Artificial Intelligence*, 2018.

[9] R. Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *ATVA17*, volume 10482, pages 269–286. Springer, 2017.

[10] L. Engstrom, B. Tran, D. Tsipras, L. Schmidt, and A. Madry. Exploring the landscape of spatial robustness. In *ICML19*, volume 97, pages 1802–1811. PMLR, 2019.

[11] A. Fawzi, S. Moosavi-Dezfooli, and P. Frossard. The robustness of deep networks: A geometrical perspective. *IEEE Signal Processing Magazine*, 34(6):50–62, 2017. doi: 10.1109/MSP.2017.2740965.

[12] M. Fazlyab, M. Morari, and G. J. Pappas. Safety verification and robustness analysis of neural networks via quadratic constraints and semidefinite programming. *IEEE TACON20*, pages 1–1, 2020. doi: 10.1109/TAC.2020.3046193.

[13] C. Ferrari, M. N. Mueller, N. Jovanović, and M. Vechev. Complete verification via multi-neuron relaxation guided branch-and-bound. In *International Conference on Learning Representations*, 2022.

[14] M. Fischer, M. Baader, and M. Vechev. Certified defense to image transformations via randomized smoothing. *arXiv preprint arXiv:2002.12463*, 2020.

[15] M. Fischer, M. Baader, and M. Vechev. Scalable certified segmentation via randomized smoothing. In *ICML21*, pages 3340–3351. PMLR, 2021.

[16] X. Gao, R. K. Saha, M. R. Prasad, and A. Roychoudhury. Fuzz testing based data augmentation to improve robustness of deep neural networks. In *ICSE20*, pages 1147–1158, 2020.

[17] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*, 2018.

[18] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[19] P. Henriksen and A. Lomuscio. Deepsplit: An efficient splitting method for neural network verification via indirect effect analysis. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI21)*, 2021.

[20] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu. Spatial transformer networks. *arXiv preprint arXiv:1506.02025*, 2015.

[21] M. Jordan and A. G. Dimakis. Exactly computing the local lipschitz constant of relu networks. *arXiv preprint arXiv:2003.01219*, 2020.

[22] C. Kanbak, S. Moosavi-Dezfooli, and P. Frossard. Geometric robustness of deep networks: Analysis and improvement. In *CVPR18*, June 2018.

[23] P. Kouvaros and A. Lomuscio. Formal verification of cnn-based perception systems. *arXiv preprint arXiv:1811.11373*, 2018.

[24] A. Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[25] Y. LeCun, C. Cortes, and C. J. Burges. The mnist database of handwritten digits, 1998.

[26] L. Li, M. Weber, X. Xu, L. Rimanic, B. Kailkhura, T. Xie, C. Zhang, and B. Li. Tss: Transformation-specific smoothing for robustness certification, 2021.

[27] C. Liu, T. Arnon, C. Lazarus, C. Strong, C. Barrett, and M. Kochenderfe. Algorithms for verifying deep neural networks. *Foundations and Trends® in Optimization*, 3-4:244–404, 2020.

[28] J. Mohapatra, T. Weng, P. Chen, S. Liu, and L. Daniel. Towards verifying robustness of neural networks against a family of semantic perturbations. In *CVPR20*, pages 244–252, 2020.

[29] K. Pei, Y. Cao, S. Yang, and S. Jana. Towards practical verification of machine learning: The case of computer vision systems. *CoRR*, abs/1712.01785, 2017.

[30] A. Raghunathan, J. Steinhardt, and P. Liang. Semidefinite relaxations for certifying robustness to adversarial examples. In *NeurIPS18*, 2018.

[31] H. Salman, G. Yang, H. Zhang, C.-J. Hsieh, and P. Zhang. A convex relaxation barrier to tight robustness verification of neural networks. In *Neural Information Processing Systems*, 2019.

[32] G. Singh, T. Gehr, M. Mirman, M. Püschel, and M. Vechev. Fast and effective robustness certification. In *Advances in Neural Information Processing Systems*, 2018.

[33] G. Singh, T. Gehr, M. Püschel, and M. Vechev. An abstract domain for certifying neural networks. In *PACMPL19*, volume 3, pages 1–30, 2019.

[34] G. Singh, T. Gehr, M. Püschel, and M. Vechev. An abstract domain for certifying neural networks. *Proc. ACM Program. Lang.*, 2019.

[35] C. Tjandraatmadja, R. Anderson, J. Huchette, W. Ma, K. PATEL, and J. Vielma. The convex relaxation barrier, revisited: Tightened single-neuron relaxations for neural network verification. *NeurIPS20*, 2020.

[36] V. Tjeng, K. Xiao, and R. Tedrake. Evaluating robustness of neural networks with mixed integer programming. *arXiv preprint arXiv:1711.07356*, 2017.

[37] H. Tran, S. Bak, W. Xiang, and T. Johnson. Verification of deep convolutional neural networks using imagestars. In *International Conference on Computer Aided Verification*, pages 18–42. Springer, 2020.

[38] VNN-COMP. Vefication of neural networks competition. https://sites.google.com/view/vnn20/vnncomp, 2020.

[39] 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 complete and incomplete neural network verification. In *Neural Information Processing Systems*, 2021.

[40] E. Wong and J. Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *ICML18*, pages 5286–5295, 2018.

[41] C. Xiao, J. Zhu, B. Li, W. He, M. Liu, and D. Song. Spatially transformed adversarial examples. *arXiv preprint arXiv:1801.02612*, 2018.

[42] 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 *International Conference on Learning Representations*, 2021.

[43] F. Yang, Z. Wang, and C. Heinze-Deml. Invariance-inducing regularization using worst-case transformations suffices to boost accuracy and spatial robustness. *CoRR*, abs/1906.11235, 2019.

[44] R. Yang, J. Laurel, S. Misailovic, and G. Singh. Provable defense against geometric transformations. In *The Eleventh International Conference on Learning Representations*, 2023.

[45] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel. Efficient neural network robustness certification with general activation functions. In *Neural Information Processing Systems*, 2018.

[46] H. Zhang, S. Wang, K. Xu, L. Li, B. Li, S. Jana, C.-J. Hsieh, and J. Z. Kolter. General cutting planes for bound-propagation-based neural network verification. In *Neural Information Processing Systems*, 2022.

Appendix

A Linear optimisation over sub-domains

Our discussion below focuses on $q = 2$ in (6), and with this choice, we have already found promising improvements in our experiments (see the main text). With this constraint we can find suboptimal piecewise bounds by solving two independent linear optimisation problems, where each problem is applied over a subset of the piecewise domain, divided at a given sample point, n. We name the parameter sub-spaces divided by n, $\boldsymbol{\kappa}_1$, and $\boldsymbol{\kappa}_2$ where $\boldsymbol{\kappa}_1, \boldsymbol{\kappa}_2 \subset \mathcal{B}$. Expressing (6) in this way gives

$$\min_{\underline{w}_1, \underline{b}_1} \quad \frac{1}{N} \sum_{i=1}^{n} \Big(\mathcal{G}_{u,v}(\kappa_i) - \{ \underline{w}_1^{\mathsf{T}} \kappa_i + \underline{b}_1 \} \Big) \tag{16a}$$

$$\text{subject to} \quad \{ \underline{w}_1^{\mathsf{T}} \kappa_i + \underline{b}_1 \} \leq \mathcal{G}_{u,v}(\kappa_i), \quad i = 1, \ldots, N,$$

$$\min_{\underline{w}_2, \underline{b}_2} \quad \frac{1}{N} \sum_{i=n}^{N} \Big(\mathcal{G}_{u,v}(\kappa_i) - \{ \underline{w}_2^{\mathsf{T}} \kappa_i + \underline{b}_2 \} \Big) \tag{16b}$$

$$\text{subject to} \quad \{ \underline{w}_2^{\mathsf{T}} \kappa_i + \underline{b}_2 \} \leq \mathcal{G}_{u,v}(\kappa_i), \quad i = 1, \ldots, N.$$

In (16a) and (16b) we optimise the area over over only the sample points within a piece's domain, $\boldsymbol{\kappa}_1$, or $\boldsymbol{\kappa}_2$; however, we enforce the constraints at every sample point. By doing this we guarantee convexity of our piecewise constraints. We develop a heuristic to determine the sample point, n, at which we split $\mathcal{B}$ based on the error between sampled points and optimal *linear* bounds

$$\underline{n} = \max_{i=1,\ldots,N} \Big(\{ \overline{w}^{\mathsf{T}} \kappa_i + \overline{b} \} - \mathcal{G}_{u,v}(\kappa_i) \Big), \tag{17}$$

where $\underline{n}$ is the splitting point for the lower bound. We calculate $\overline{n}$ correspondingly using the lower linear bound. There exists a splitting point, n, that would produce optimal piecewise bounds, but finding it is infeasible. In practice, we first compute a single linear bound for lower and upper constraints and then use this bound to compute the splitting point from 17. Then, once the piecewise bound is obtained, half of the original linear bound is effectively discarded for the verification procedure. We compute the bounds in this way for two reasons: firstly, it enables us to apply our splitting heuristic in 17, and secondly, it is computationally efficient in our experimental setting where we require the linear bounds for comparison.

B Details of branch-and-bound procedure

With our unsound constraints, our method closely follows that of [1], with the important exception that we treat our single piecewise bound as two, separate linear bounds with domains, $\boldsymbol{\kappa}_1$, and $\boldsymbol{\kappa}_2$. We first define a function, $\underline{f}_{u,v}$, to track the violation of a bound by the pixel value function, $\mathcal{G}_{u,v}$. In the case that the lower bound is piecewise, we will maximise $\underline{f}_{u,v}$ twice over $\boldsymbol{\kappa}_1$, and $\boldsymbol{\kappa}_2$, and $\overline{f}_{u,v}$ once over $\mathcal{B}$. Maximisation of $\underline{f}_{u,v}$ is done via a branch-and-bound Lipschitz procedure. Algorithm 1 shows a simplified version of the implementation we use. For each instance of $\underline{f}_{u,v}(\kappa)$ where $\kappa \in \boldsymbol{\kappa}_1$, we first approximate the Lipshitz constant, L_i, and use it to bound $\underline{f}_{u,v}$

$$\underline{f}_{\text{bound}} = L_i \frac{\boldsymbol{\kappa}_1}{2} + \Big(\underline{f}_{u,v}(\kappa_i) \Big), \tag{18}$$

where κ_i is the midpoint of $\boldsymbol{\kappa}_1$. We find upper bound candidates by sampling the violation function$f_{u,v}(\kappa)$ at four, evenly spaced points in $\boldsymbol{\kappa}_1$; the largest valued obtained becomes the maximum value candidate, $\underline{f}_{\text{max}}$. We aim to find a maximum value-bound pair that satisfies $\underline{f}_{\text{bound}} - \underline{f}_{\text{max}} < \epsilon$, with ϵ given. This process is repeated until a satisfactory maximum candidate is found, splitting $\boldsymbol{\kappa}$ in each iteration. We can ignore any sub-domain, $\boldsymbol{\kappa}_1^n$, of $\boldsymbol{\kappa}_1$ where the function bound $\underline{f}_{\text{bound}}$ in $\boldsymbol{\kappa}_1^n$ is smaller than a maximum value candidate $\underline{f}_{\text{max}}$ in any other sub-domain. This is because we can guarantee that the maximum value, in this case, is not in the $\boldsymbol{\kappa}_1^n$ sub-domain. We deal only with 1-dimensional parameter spaces for which we split at the midpoint. The outline of this procedure is given in Algorithm 1.

Algorithm 1 Branch-and-bound Lipschitz Optimisation Procedure

```
Input: $\underline{f}_{u,v}$, $\mathcal{B}$, $\epsilon$, $n$, $N$
Output: $\xi^*$
$\underline{f}_{\text{max}} := \max_{l=1,\ldots,n} \underline{f}_{u,v}(\kappa_l)$, where $\kappa_l \in \mathcal{B}$.
$\underline{f}_{\text{bound}} := \text{bound}(\underline{f}_{u,v}, \nabla \underline{f}_{u,v}, \mathcal{B})$, where the operation $\text{bound}(\cdot)$ refers to (14).
$\mathcal{L} := [(\underline{f}_{\text{max}}, \underline{f}_{\text{bound}}, \mathcal{B})]$.
while $\underline{f}_{\text{bound},i=1,\ldots,N} - \underline{f}_{\text{max},i=1,\ldots,N} > \epsilon$ do
    for $i \leftarrow 1$ to $N$ do
        if $\underline{f}_{\text{bound},i} - \underline{f}_{\text{max},i} > \epsilon$ then
            $\mathcal{B}_{i,i+N} = \text{split}(\mathcal{B}_i)$.
            $\underline{f}_{\text{max},i} := \max_{l=1,\ldots,n} \underline{f}_{u,v}(\kappa_l)$, where $\kappa_l \in \mathcal{B}_i$.
            $\underline{f}_{\text{bound},i} := \text{bound}(\underline{f}_{u,v}, \nabla \underline{f}_{u,v}, \mathcal{B}_i)$.
            $\mathcal{L}_i := [(\underline{f}_{\text{max},i}, \underline{f}_{\text{bound},i}, \mathcal{B}_i)]$.
        end if
    end for
end while
return $\xi^* = \max_{i=1,\ldots,N} \underline{f}_{\text{max},i}$.
```