Certifying Geometric Robustness of Neural Networks

Mislav Balunović, Maximilian Baader, Gagandeep Singh, Timon Gehr, Martin Vechev
Department of Computer Science
ETH Zurich
`{mislav.balunovic, mbaader, gsingh, timon.gehr, martin.vechev}@inf.ethz.ch`

Abstract

The use of neural networks in safety-critical computer vision systems calls for their robustness *certification* against natural geometric transformations (e.g., rotation, scaling). However, current certification methods target mostly norm-based pixel perturbations and cannot certify robustness against geometric transformations. In this work, we propose a new method to compute sound and asymptotically optimal linear relaxations for any composition of transformations. Our method is based on a novel combination of sampling and optimization. We implemented the method in a system called DEEPG and demonstrated that it certifies significantly more complex geometric transformations than existing methods on both defended and undefended networks while scaling to large architectures.

1 Introduction

Robustness against geometric transformations is a critical property that neural networks deployed in computer vision systems should satisfy. However, recent work [1, 2, 3] has shown that by using natural transformations (e.g., rotations), one can generate *adversarial examples* [4, 5] that cause the network to misclassify the image, posing a safety threat to the entire system. To address this issue, one would ideally like to *prove* that a given network is free of such geometric adversarial examples. While there has been substantial work on certifying robustness to changes in pixel intensity (e.g., [6, 7, 8]), only the recent work of [9] proposed a method to certify robustness to geometric transformations. Its key idea is summarized in Fig. 1: Here, the goal is to prove that any image obtained by translating the original image by some $\delta_x, \delta_y \in [-4, 4]$ is classified to label 3. To accomplish this task, [9] propagates the image and the parameters δ_x, δ_y through every component of the transformation using interval bound propagation. The output region I is a convex shape capturing all images that can be obtained by translating the original image between -4 and 4 pixels. Finally, I is fed to a standard neural network verifier which tries to prove that all images in I classify to 3. This method can also be improved using tighter relaxation based on Polyhedra [10]. Unfortunately, as we show later, bound propagation is not satisfactory. The core issue is that any approach based on bound propagation inherently accumulates loss for every intermediate result, often producing regions that are too coarse to allow the neural network verifier to succeed. Instead, we propose a new method based on sampling and optimization which computes a convex relaxation for the *entire composition* of transformations.

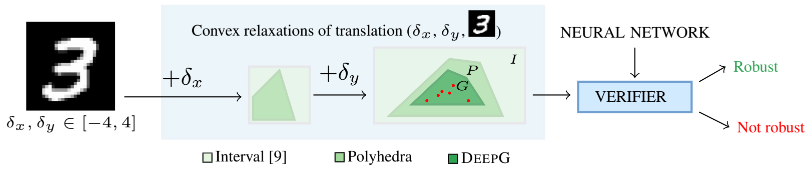


Figure 1: End-to-end certification of geometric robustness using different convex relaxations.

The key idea of our method is to sample the parameters of the transformation (e.g., δ_x, δ_y), obtaining sampled points at the output (red dots in Fig. 1), and to then compute sound and asymptotically optimal linear constraints around these points (shape G). We implemented our method in a system called DEEPG and showed that it is significantly more precise than bound propagation (using Interval or Polyhedra relaxation) on a wide range of geometric transformations. To the best of our knowledge, DEEPG is currently the state-of-the-art system for certifying geometric robustness of neural networks.

Main contributions Our main contributions are:

- A novel method, combining sampling and optimization, to compute asymptotically optimal linear constraints bounding the set of geometrically transformed images. We demonstrate that these constraints enable significantly more precise certification compared to prior work.
- A complete implementation of our certification in a system called DEEPG. Our results show substantial benefits over the state-of-the-art across a range of geometric transformations. We make DEEPG publicly available at `https://github.com/eth-sri/deepg/`.

2 Related work

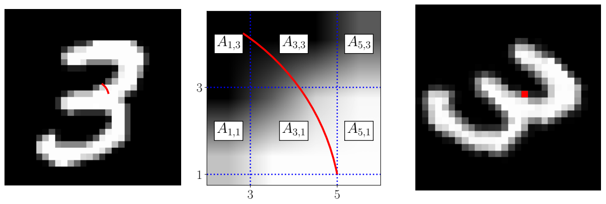
We now discuss some of the closely related work in certification of the neural networks and their robustness to geometric transformations.

Certification of neural networks Recently, a wide range of methods have been proposed to certify robustness of neural networks against adversarial examples. Those methods are typically based on abstract interpretation [6, 7], linear relaxation [8, 11, 12], duality [13], SMT solving [14, 15, 16], mixed integer programming [17], symbolic intervals [18], Lipschitz optimization [19], semi-definite relaxations [20] and combining approximations with solvers [21, 22]. Certification procedures can also be extended into end-to-end training of neural networks to be provably robust against adversarial examples [23, 24]. Recent line of work [25, 26, 27] proposes to construct a classifier based on the smoothed neural network which comes with probabilistic guarantees on the robustness against L_2 perturbations. None of these works except [9] consider geometric transformations, while [9] only verifies robustness against rotation. The work of [28] also generates linear relaxations of non-linear specifications, but they do not handle geometric transformations. We remark that [1] considers a much more restricted (discrete) setting leading to a finite number of images. This means that certification can be performed by brute-force enumeration of this finite set of transformed images. In our setting, as we will see, this is not possible, as we are dealing with an uncountable set of transformed images.

Neural networks and geometric transformations There has been considerable research in empirical quantification of geometric robustness of neural networks [2, 3, 29, 30, 31, 32]. Another line of work focuses on the design of architectures which possess an inherent ability to learn to be more robust against such transformations [33, 34]. However, all of these approaches offer only empirical evidence of robustness. Instead, our focus is to provide formal guarantees.

Certification of geometric transformations Prior work [9] introduced a method for analyzing rotations using the interval propagation and performed certification using the state-of-the-art verifier DEEPPOLY. It is straightforward to generalize their interval approach to handle more complex transformations beyond rotation (we provide details in Appendix A.4). However, as we show experimentally, interval propagation loses precision which is why certification often does not succeed.

To capture relationships between pixel values and transformations, one would ideally use the Polyhedra relaxation [10] instead of intervals. While Polyhedra offers higher precision, its worst-case running time is exponential in the number of variables [35]. Hence, it does not scale to geometric transformations, where every pixel introduces a new variable. Thus, we extended the recent DeepPoly relaxation [9] (a restricted Polyhedra) with custom approximations for the operations used in several geometric transformations (e.g., translation, scaling). Our experimental results show that even though this approach significantly improves over intervals, it is not precise enough to certify robustness of most images in our dataset. In turn, this motivates the method introduced in this paper.



(a) Original image (b) Interpolation (c) Rotated image

Figure 2: Image rotated by $-\frac{\pi}{4}$ degrees. Here, (a) shows the original image, while (b) shows part of (a) with a focus on relevant interpolation regions. Finally, (c) shows the resulting rotated image.

3 Background

Our goal is to certify the robustness of a neural network against adversarial examples generated using parameterized geometric transformations. In this section we formulate this problem statement, introduce the notation of transformations and provide a running example which we use throughout the paper to illustrate key concepts.

Geometric image transformations A geometric image transformation consists of a parameterized spatial transformation $\mathcal{T}_{\boldsymbol{\mu}}$, an interpolation I which ensures the result can be represented on a discrete pixel grid, and parameterized changes in brightness and contrast $\mathcal{P}_{\alpha,\beta}$. We assume $\mathcal{T}_{\boldsymbol{\mu}}$ is a composition of bijective transformations such as rotation, translation, shearing and scaling (full descriptions of all transformations are in Appendix A.1). While our approach also applies to other interpolation methods, in this work we focus on the case where I is the commonly-used bilinear interpolation.

To ease presentation, we assume the image (with integer coordinates) consists of an even number of rows and columns, is centered around $(0, 0)$, and its coordinates are odd integers. We note that all results hold in the general case (without the assumption).

Interpolation The bilinear interpolation $I\colon \mathbb{R}^2 \to [0, 1]$ evaluated on a coordinate $(x, y) \in \mathbb{R}^2$ is a polynomial of degree 2 given by

$$I(x,y) := \frac{1}{4} \sum_{\delta_i,\delta_j \in \{0,2\}} p_{i+\delta_i,j+\delta_j}(2 - |i + \delta_i - x|)(2 - |j + \delta_j - y|). \tag{1}$$

Here, (i, j) is the lower-left corner of an *interpolation region* $A_{i,j} := [i, i + 2] \times [j, j + 2]$ which contains pixel (x, y). Matrix p consists of pixel values at corresponding coordinates in the original image. The function I is continuous on $\mathbb{R}^2$ and smooth on the interior of every interpolation region. These interpolation regions are depicted with the blue horizontal and vertical dotted lines in Fig. 2b.

The pixel value $\tilde{p}_{x,y}$ of the transformed image can be obtained by (i) calculating the preimage of the coordinate (x, y) under $\mathcal{T}_{\boldsymbol{\mu}}$, (ii) interpolating the resulting coordinate using I to obtain a value ξ, and (iii) applying the changes in contrast and brightness via $P_{\alpha,\beta}(\xi) = \alpha\xi + \beta$, to obtain the final pixel value $\tilde{p}_{x,y} = \mathcal{I}_{\alpha,\beta,\boldsymbol{\mu}}(x, y)$. These three steps are captured by

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

Running example To illustrate key concepts introduced throughout the paper, we use the running example of an MNIST image [36] shown in Fig. 2. On this image, we will apply a rotation R_ϕ with an angle ϕ. For our running example, we set $\mathcal{P}_{\alpha,\beta}$ to be the identity.

Consider the pixel $\tilde{p}_{5,1}$ in the transformed image shown in Fig. 2c (the pixel is marked with a red dot). The transformed image is obtained by rotating the original image in Fig. 2a by an angle $\phi = -\frac{\pi}{4}$. This results in the pixel value

$$\tilde{p}_{5,1} = I \circ R^{-1}_{-\frac{\pi}{4}}(5,1) = I(2\sqrt{2}, 3\sqrt{2}) \approx 0.30$$

Here, the preimage of point $(5, 1)$ under $R_{-\frac{\pi}{4}}$ produces the point $(2\sqrt{2}, 3\sqrt{2})$ with non-integer coordinates. This point belongs to the interpolation region $A_{1,3}$ and by applying $I(2\sqrt{2}, 3\sqrt{2})$ to the original image in Fig. 2a, we obtain the final pixel value ≈ 0.30 for pixel $(5, 1)$ in the rotated image.

Neural network certification To certify robustness of a neural network with respect to a geometric transformation, we rely on the state-of-the-art verifier DeepPoly [9]. For complex properties such as geometric transformations, the verifier needs to receive a convex relaxation of all possible inputs to the network. If this relaxation is too coarse, the verifier will not be able to certify the property.

Problem statement To guarantee robustness, our goal is to compute a convex relaxation of all possible images obtainable via the transformation $\mathcal{I}_{\alpha,\beta,\boldsymbol{\mu}}$. This relaxation can then be provided as an input to a neural network verifier (e.g., DeepPoly). If the verifier proves that the neural network classification is correct for all images in this relaxation, then geometric robustness is proven.

4 Asymptotically optimal linear constraints via optimization and sampling

We now present our method for computing the optimal linear approximation (in terms of volume).

Motivation As mentioned earlier, designing custom transformers for every operation incurs precision loss at every step in the sequence of transformations. Our key insight is to define an optimization problem in a way where its solution is the *optimal* (in terms of volume) lower and upper linear constraint for the entire sequence. To solve this optimization problem, we propose a method based on sampling and linear programming. Our method produces, for every pixel, asymptotically optimal lower and upper linear constraints for the *entire composition* of transformations (including interpolation). Such an optimization problem is generally difficult to solve, however, we find that with geometric transformations, our approach is scalable and contributes only a small portion to the entire end-to-end certification running time.

Optimization problem To compute linear constraints for every pixel value, we split the hyperrectangle h representing the set of possible parameters into s splits $\{h_k\}_{k\in[s]}$. Our goal will be to compute *sound* lower and upper linear constraints for the pixel value $\mathcal{I}_{\boldsymbol{\kappa}}(x,y)$ for a given pixel (x,y). Both of these constraints will be linear in the parameters $\boldsymbol{\kappa} = (\alpha,\beta,\boldsymbol{\mu}) \in h_k$. We define *optimal* and *sound* linear (lower and upper) constraints for $\mathcal{I}_{\boldsymbol{\kappa}}(x,y)$ to be a pair of hyperplanes fulfilling

$$\boldsymbol{w}_l^T\boldsymbol{\kappa} + b_l \leq \mathcal{I}_{\boldsymbol{\kappa}}(x,y) \qquad \forall \boldsymbol{\kappa} \in h_k \tag{3}$$

$$\boldsymbol{w}_u^T\boldsymbol{\kappa} + b_u \geq \mathcal{I}_{\boldsymbol{\kappa}}(x,y) \qquad \forall \boldsymbol{\kappa} \in h_k, \tag{4}$$

while minimizing

$$L(\boldsymbol{w}_l, b_l) = \frac{1}{V}\int_{\boldsymbol{\kappa}\in h_k} \big(\mathcal{I}_{\boldsymbol{\kappa}}(x,y) - (b_l + \boldsymbol{w}_l^T\boldsymbol{\kappa})\big)\mathrm{d}\boldsymbol{\kappa} \tag{5}$$

$$U(\boldsymbol{w}_u, b_u) = \frac{1}{V}\int_{\boldsymbol{\kappa}\in h_k} \big((b_u + \boldsymbol{w}_u^T\boldsymbol{\kappa}) - \mathcal{I}_{\boldsymbol{\kappa}}(x,y)\big)\,\mathrm{d}\boldsymbol{\kappa}. \tag{6}$$

Here V denotes the normalization constant equal to the volume of h_k. Intuitively, the optimal constraints should result in a convex relaxation of minimum volume. This formulation also allows independent computation for every pixel, facilitating parallelization across pixels. Next, we describe how we obtain lower constraints (upper constraints are computed analogously).

Step 1: Compute a potentially unsound constraint To generate a reasonable but a potentially unsound linear constraint, we sample parameters $\boldsymbol{\kappa}_1, \ldots, \boldsymbol{\kappa}_N$ from h_k, approximate the integral in Eq. (5) by its Monte Carlo estimate L_N and enforce the constraints only at the sampled points:

$$\begin{aligned} &\min_{(\boldsymbol{w}_l,b_l)\in W} L_N(\boldsymbol{w}_l, b_l) = \min_{(\boldsymbol{w}_l,b_l)\in W} \frac{1}{N}\sum_{i=1}^{N} \big(\mathcal{I}_{\boldsymbol{\kappa}_i}(x,y) - (b_l + \boldsymbol{w}_l^T\boldsymbol{\kappa}_i)\big), \\ &b_l + \boldsymbol{w}_l^T\boldsymbol{\kappa}_i \leq \mathcal{I}_{\boldsymbol{\kappa}_i}(x,y) \qquad \forall i \in [N]. \end{aligned} \tag{7}$$

This problem can be solved exactly using linear programming (LP). The solution is a potentially unsound constraint $b_l' + \boldsymbol{w}_l'^T\boldsymbol{\kappa}$ (it may violate the constraint at non-sampled points). For our running example, the region bounded by these potentially unsound lower and upper linear constraints is shown as orange in Fig. 3.

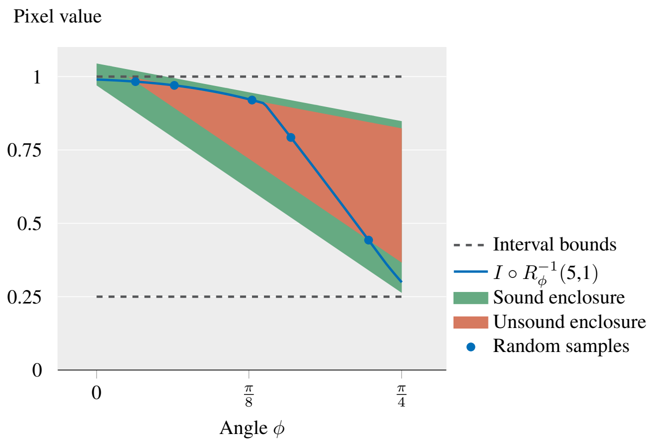


Figure 3: Unsound (Step 1) and sound (Step 3) enclosures for $I \circ R_\phi^{-1}(5,1)$, with respect to random sampling from $\phi \in [0, \frac{\pi}{4}]$, in comparison to the interval bounds from prior work [9]. Note that $I \circ R_\phi^{-1}(5,1)$ is not piecewise linear, because bilinear interpolation is a polynomial of degree 2.

Step 2: Bounding the maximum violation Our next step is to compute an upper bound on the violation of Eq. (3) induced by our potentially unsound constraint from Step 1. This violation is equal to the maximum of the function $f(\boldsymbol{\kappa}) = b_l' + \boldsymbol{w}_l'^T \boldsymbol{\kappa} - \mathcal{I}_{\boldsymbol{\kappa}}(x, y)$ over the hyperrectangle h_k. It can be shown that the function f is Lipschitz continuous which enables application of standard global optimization techniques with guarantees [37]. We remark that such methods have already been applied for optimization over inputs to neural network [38, 19].

We show a high level description of this optimization procedure in Algorithm 1. Throughout the optimization, we maintain a partition of the domain of function f into hyperrectangles h. The hyperrectangles are stored in a priority queue q sorted by an upper bound f_h^{bound} of the maximum value the function can take inside of the hyperrectangle. At every step, shown in Line 6, the hyperrectangle with the highest upper bound is further refined into smaller hyperrectangles $h_1', \dots, h_k'$ and their upper bounds are recomputed. This procedure finishes when the difference between every upper bound and the maximal value at one of the hyperrectangle centers is at most ϵ. Finally, maximum upper bound of the elements in the queue is returned as a result of the optimization. We provide more details on the optimization algorithm in Appendix A.5.

Algorithm 1 Lipschitz Optimization with Bound Refinement

```
1:  **Input:** $f, h, k \geq 2$
2:  $f_{\max} := f(\text{center}(h))$
3:  $f_h^{\text{bound}} := f^{\text{bound}}(h, \nabla f)$
4:  $q := [(h, f_h^{\text{bound}})]$
5:  **repeat**
6:      pop $(h', f_{h'}^{\text{bound}})$ from $q$ with maximum $f_{h'}^{\text{bound}}$
7:      $h_1', \dots, h_k' := \text{partition}(h', \nabla f)$
8:      **for** $i := 1$ **to** $k$ **do**
9:          $f_{\max} := \max(f(\text{center}(h_i')), f_{\max})$
10:         $f_{h_i'}^{\text{bound}} := f^{\text{bound}}(h_i', \nabla f)$
11:         **if** $f_{h_i'}^{\text{bound}} > f_{\max} + \epsilon$ **then**
12:             add $(h_i', f_{h_i'}^{\text{bound}})$ to $q$
13:         **end if**
14:     **end for**
15: **until** a maximal $f_{h'}^{\text{bound}}$ in $q$ is lower than $f_{\max} + \epsilon$
16: **return** $f_{\max} + \epsilon$
```

The two most important aspects of the algorithm, which determine the speed of convergence, are (i) computation of an upper bound, and (ii) choosing an edge along which to refine the hyperrectangle. To compute an upper bound inside of a hyperrectangle spanned between points $\boldsymbol{h}_l$ and $\boldsymbol{h}_u$, we use:

$$f(\boldsymbol{\kappa}) \leq f(\tfrac{\boldsymbol{h}^u + \boldsymbol{h}^l}{2}) + \frac{1}{2} \left|\nabla^{\boldsymbol{h}} f\right|^T (\boldsymbol{h}^u - \boldsymbol{h}^l). \tag{8}$$

Here $\left|\nabla^{\boldsymbol{h}} f\right|$ can be any upper bound on the true gradient which satisfies $|\partial_i f(\boldsymbol{\kappa})| \leq \left|\nabla^h f\right|_i$ for every dimension i. To compute such a bound, we perform reverse-mode automatic differentiation of the function f using interval propagation (this is explained in more details in Appendix A.2). As an added benefit, results of our analysis can be used for pruning of hyperrectangles. We reduce a hyperrectangle to one of its lower-dimensional faces along dimensions for which analysis on gradients proves that the respective partial derivative has a constant sign within the entire hyperrectangle. We also improve on standard refinement heuristics — instead of refining along the largest edge, we additionally weight edge length by an upper bound on the partial derivative of that dimension. In our experiments, we find that these insights speed up convergence compared to simply applying the method out of the box.

Let v_l be the result of the above Lipschitz optimization algorithm. It is then guaranteed that

$$v_l \leq \max_{\boldsymbol{\kappa} \in h_k} \left(b'_l + \boldsymbol{w}'^T_l \boldsymbol{\kappa} - \mathcal{I}_{\boldsymbol{\kappa}}(x, y)\right) \leq v_l + \epsilon.$$

Step 3: Compute a sound linear constraint In the previous step we obtained a bound on the maximum violation of Eq. (3). Using this bound, in this step, we update our linear constraints $b_l = b'_l - v_l - \epsilon$ and $\boldsymbol{w}_l = \boldsymbol{w}'_l$ to obtain a sound lower linear constraint (it satisfies Eq. (3)). The region bounded by the sound lower and upper linear constraints is shown as green in Fig. 3. It is easy to check that our constraint is sound:

$$b_l + \boldsymbol{w}_l^T \boldsymbol{\kappa} = b'_l - v_l - \epsilon + \boldsymbol{w}'^T_l \boldsymbol{\kappa} \leq \mathcal{I}_{\boldsymbol{\kappa}}(x, y) \qquad \forall \boldsymbol{\kappa} \in h_k.$$

Running example As in Section 3, we focus on the pixel $(5, 1)$, choose $s = 2$ splits for $[0, \frac{\pi}{2}]$, and focus our attention on the split $[0, \frac{\pi}{4}]$. In Step 1, we sample random points $\{0.1,\ 0.2,\ 0.4,\ 0.5,\ 0.7\}$ for our parameter $\phi \in [0, \frac{\pi}{4}]$ and evaluate $I \circ R_\phi^{-1}(5, 1)$ on these points, obtaining $\{0.98, 0.97, 0.92, 0.79, 0.44\}$.

These points correspond to the blue dots in Fig. 3. Solving the LP in Eq. (7) yields $b'_l = 1.07$ and $w'_l = -0.90$. Similarly, we compute a potentially unsound upper constraint. Together, these two constraints form the orange enclosure shown in Fig. 3. This enclosure is in fact unsound, as some points on the blue line (those not sampled above) are not included in the region.

In Step 2, using Lipschitz optimization for the function $1.07 - 0.9\phi - I \circ R_\phi^{-1}$ with $\epsilon = 0.02$ over $\phi \in [0, \frac{\pi}{4}]$ we obtain $v_l = 0.08$ resulting in the sound linear lower constraint $0.97 - 0.9\phi$. This, together with the similarly obtained sound upper constraint, forms the sound (green) enclosure in Fig. 3. In the figure we also show the black dashed lines corresponding to the interval bounds from prior work [9] which enclose much larger volume than our linear constraints.

Asymptotically optimal constraints While our constraints may not be optimal, one can show they are asymptotically optimal as we increase the number of samples:

Theorem 1. *Let N be the number of points sampled in our algorithm and ϵ the tolerance used in the Lipschitz optimization. Let $(\boldsymbol{w}_l, b_l)$ be our lower constraint and let $(\boldsymbol{w}^*, b^*)$ be the minimum of L. For every $\delta > 0$ there exists N_δ such that $|L(\boldsymbol{w}_l, b_l) - L(\boldsymbol{w}^*, b^*)| < \delta + \epsilon$ for every $N > N_\delta$, with high probability. Analogous result holds for upper constraint $(\boldsymbol{w}_u, b_u)$ and function U.*

We provide a proof of Theorem 1 in Appendix A.3. Essentially, as we increase the number of sampled points in our linear program, we approach the optimal constraints. The ϵ tolerance in Lipschitz optimization could be further decreased towards 0 to obtain an offset as small as desired. In our experiments, we also show empirically that close-to-optimal bounds can be obtained with a relatively small number of samples.

Table 1: Comparison of DEEPG which uses linear constraints with the baseline based on interval bound propagation. Here, R(ϕ) corresponds to rotations with angles between $\pm\phi$; T(x, y), to translations between $\pm x$ pixels horizontally and between $\pm y$ pixels vertically; Sc(p), to scaling the image between $\pm p\%$; Sh(m), to shearing with a shearing factor between $\pm m\%$; and B(α, β), to changes in contrast between $\pm\alpha\%$ and brightness between $\pm\beta$.

		Accuracy (%)	Attacked (%)	Certified (%)	
				Interval [9]	DEEPG
MNIST	R(30)	99.1	0.0	7.1	**87.8**
	T(2, 2)	99.1	1.0	0.0	**77.0**
	Sc(5), R(5), B(5, 0.01)	99.3	0.0	0.0	**34.0**
	Sh(2), R(2), Sc(2), B(2, 0.001)	99.2	0.0	1.0	**72.0**
Fashion-MNIST	Sc(20)	91.4	11.2	19.1	**70.8**
	R(10), B(2, 0.01)	87.7	3.6	0.0	**71.4**
	Sc(3), R(3), Sh(2)	87.2	3.5	3.5	**56.6**
CIFAR-10	R(10)	71.2	10.8	28.4	**87.8**
	R(2), Sh(2)	68.5	5.6	0.0	**54.2**
	Sc(1), R(1), B(1, 0.001)	73.2	3.8	0.0	**54.4**

5 Experimental evaluation

We implemented our certification method in a system called DEEPG. First, we demonstrate that DEEPG can certify robustness to significantly more complex transformations than both prior work and traditional bound propagation approaches based on relational abstractions. Second, we experimentally show that our method requires relatively small number of samples to converge to the optimal linear constraints. Third, we investigate the effectiveness of a variety of training methods to train a network provably robust to geometric transformations. Finally, we demonstrate that DEEPG is scalable and can certify geometric robustness for large networks. We provide networks and code to reproduce the experiments in this paper at `https://github.com/eth-sri/deepg/`.

Experimental setup We evaluate on image recognition datasets: MNIST [36], Fashion-MNIST [39] and CIFAR-10 [40]. For each dataset, we randomly select 100 images from the test set to certify. Among these 100 images, we discard all images that are misclassified even without any transformation. In all experiments for MNIST and Fashion-MNIST we evaluate a 3-layer convolutional neural network with 9 618 neurons, while for the more challenging CIFAR-10 dataset we consider a 4-layer convolutional network with 45 216 neurons. Details of these architectures are provided in Appendix B.2. We certify robustness to composition of transformations such as rotation, translation, scaling, shearing and changes in brightness and contrast. These transformations are formally defined in Appendix A.1. All experiments except the one with large networks were performed on a desktop PC with 2 GeForce RTX 2080 Ti GPU-s and 16-core Intel(R) Core(TM) i9-9900K CPU @ 3.60GHz.

Comparison with prior work In the first set of experiments we certify robustness to geometric transformations and compare our results to prior work [9]. While they considered only rotations, we implemented their approach for other transformations and their compositions. This generalization is described in detail in Appendix A.4 and shown as Interval in Table 1. For each dataset and geometric transformation, we train a neural network using data augmentation with the transformation that we are certifying. Additionally, we use PGD adversarial training to obtain a network robust to noise which we later show significantly increases verifiability of the network. We provide runtime analysis of the experiments and all hyperparameters used for certification in Appendix B.2.

We first measure the success of a randomized attack which samples 100 transformed images uniformly at random [2]. Then, we generate linear constraints using DEEPG, as described in Section 4. Constraints produced by both our method and the interval baseline are given as an input to the state-of-the-art neural network verifier DeepPoly [9]. We invoke both methods for every split separately, with the same set of splits. In order to make results fully comparable, both methods are parallelized over pixels in the same way and the refinement parameter k of interval propagation is chosen such that its runtime is roughly equal to the one of DEEPG. Table 1 shows the results of our evaluation.

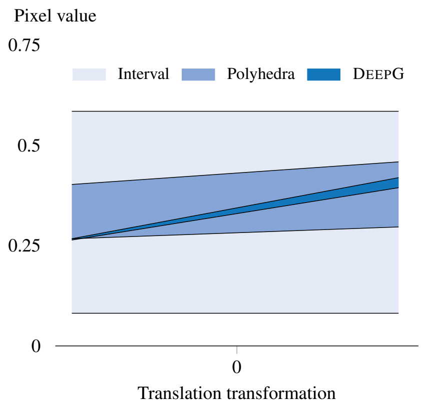


Figure 4: Translation transformation approximated using interval propagation, polyhedra and DEEPG, for a representative pixel.

	Images certified (%)		
	Interval	Polyhedra	DeepG
T(0.25)	0	14	**90**
Sc(4)	0	23	**75**
Sh(10)	0	12	**38**

Table 2: Certification success rates of interval propagation [9], Polyhedra and DeepG (for translation, shearing and scaling).

While interval propagation used in prior work can prove robustness for simpler transformations, it fails for more complex geometric transformations. For example, already for translation which has two parameters it does not succeed at certifying a single image. This shows that, in order to certify complex transformations, one has to capture relationships between pixel values and transformation parameters using a relational abstraction. Linear constraints computed by DEEPG provide a significant increase in certification precision, justifying the more involved method to compute the constraints.

Comparison with custom transformers To understand the benefits of our method further, we decided to construct a more advanced baseline than the interval propagation. In particular, we crafted specialized transformers for DeepPoly [9], which is a restriction of Polyhedra, to handle the operations used in geometric transformations. These kind of transformers have brought significant benefits over the interval propagation in certifying robustness to noise perturbations, and thus, we wanted to see what the benefits would be in our setting of geometric transformations. Concretely, we designed Polyhedra transformers for addition and multiplication which enables handling of geometric operations. These transformers are non-trivial and are listed in Appendix B.1. Fig. 4 shows that relaxation with these transformers is significantly tighter than intervals. This also translates to higher certification rates compared to intervals, shown in Table 2. However, this method still fails to certify many images on which DEEPG succeeds. This experiment shows that generating constraints for the entire composition of transformations as in DEEPG is (expectedly) more effective than crafting transformers for individual operations of the transformation.

Convergence towards optimal bounds While Theorem 1 shows that DEEPG obtains optimal linear constraints in the limit, we also experimentally check how quickly our method converges in practice. For this experiment, we consider rotation between $-2°$ and $2°$, composed with scaling between -5% and 5%. We run DEEPG while varying the number of samples used for the LP solver (n) and tolerance in Lipschitz optimization (ϵ). In Table 3 we show the approximation error (average distance between lower and upper linear constraint), certification rate and time taken to compute the constraints. For instance, even with only 100 samples and $\epsilon = 0.01$ DEEPG can certify almost every image in 1.2 seconds. While higher number of samples and smaller tolerance are necessary to obtain more precise bounds, they do not bring significant increase in certification rates.

Table 3: Speed of convergence of DEEPG towards optimal linear bounds.

n	ϵ	Approximation error	Certified(%)	Runtime(s)
100	0.1	0.032	54.8	1.1
100	0.01	0.010	96.5	1.2
1000	0.001	0.006	97.8	4.9
10000	0.00001	0.005	98.2	46.1

Table 4: Certification using DEEPG for neural networks trained using different training techniques.

		Accuracy (%)	Attack success (%)	Certified (%)	
				Interval [9]	DEEPG
MNIST	Standard	98.7	52.0	0.0	12.0
	Augmented	99.0	4.0	0.0	46.5
	L_∞-PGD	98.9	45.5	0.0	20.2
	L_∞-DIFFAI	98.4	51.0	1.0	17.0
	L_∞-PGD + Augmented	**99.1**	**1.0**	0.0	**77.0**
	L_∞-DIFFAI + Augmented	98.0	6.0	**42.0**	66.0

Comparison of different training methods Naturally, we would like to know how to train a neural network which is certifiably robust against geometric transformations. In this experiment, we evaluate effectiveness of a wide range of training methods to train a network certifiably robust to the translation up to 2 pixels in both x and y direction. While [2] train with adversarial examples and show that this leads to lower empirical success of an adversary, we are interested in a different question: can we train neural networks to be provably robust against geometric transformations?

We first train the same MNIST network as before, in a standard fashion, without any kind of defense. As expected, the resulting network shown in the first row of Table 4 is not robust at all – random attack can find many translations which cause misclassification of the network. To alleviate this problem, we incorporate data augmentation into training by randomly translating every image in a batch between -4 and 4 pixels before passing it through the network. As a result, the network is significantly more robust against the attack, however there are still many images that we fail to certify. To make the network more amenable to certification, we consider two additional techniques. They are both based on the observation that convex relaxation of geometric transformations can be viewed as noise in the pixel values. To train a network which is robust to noise we consider adversarial training with projected gradient descent (PGD) [41] and provable defense based on DIFFAI [23]. We also consider combination of these techniques with data augmentation.

Based on the results shown in Table 4, we conclude that training with PGD coupled with data augmentation achieves both highest accuracy and highest number of certified images. Training with DIFFAI significantly increases certification rate for interval bound propagation, but has the drawback of significantly lower accuracy than other methods.

Evaluation on large networks We evaluated whether DEEPG can certify robustness of large CIFAR-10 networks with residual connections. We certified ResNet-Tiny and ResNet-18 from [42] with 312k and 558k neurons, respectively. As certifying these networks is challenging, we consider relatively small rotation between -2 and 2 degrees. As before, we generated constraints using both DEEPG and interval bound propagation. This experiment was performed on a Tesla V100 GPU.

ResNet-Tiny was trained using PGD adversarial training and has standard accuracy 83.8%. Using the constraints from DEEPG, we certify 91.1% of images while interval constraints allow us to certify only 1.3%. Average time for the verifier to certify or report failure is 528 seconds per image.

ResNet-18 was trained using DIFFAI [42] and has standard accuracy 40.2%. In this case, using constraints from DEEPG, the verifier certifies 82.2% of images. Similarly as before (see Table 4), training the network with DIFFAI also enables a high certification rate of 77.8% even using interval constraints. However, the drawback is that this network has low accuracy of only 40.2% compared to ResNet-Tiny trained with PGD which has 83.8% accuracy. On average, the verifier takes 1652 seconds per image. Here, generating the constraints for both networks took 25 seconds on average.

6 Conclusion

We introduced a new method for computing (optimal at the limit) linear constraints on geometric image transformations combining Lipschitz optimization and linear programming. We implemented the method in a system called DEEPG and showed that it leads to significantly better certification precision in proving robustness against geometric perturbations than prior work, on both defended and undefended networks.

Acknowledgments

We would like to thank anonymous reviewers for their feedback and Christoph Müller for his help with enabling certification of large residual networks using DeepPoly.

References

[1] K. Pei, Y. Cao, J. Yang, and S. Jana, "Towards practical verification of machine learning: The case of computer vision systems," *arXiv preprint arXiv:1712.01785*, 2017.

[2] L. Engstrom, B. Tran, D. Tsipras, L. Schmidt, and A. Madry, "Exploring the landscape of spatial robustness," in *Proceedings of the 36th International Conference on Machine Learning*, 2019.

[3] C. Kanbak, S. M. Moosavi Dezfooli, and P. Frossard, "Geometric robustness of deep networks: analysis and improvement," *Proceedings of IEEE CVPR*, 2018.

[4] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[5] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli, "Evasion attacks against machine learning at test time," in *Joint European conference on machine learning and knowledge discovery in databases*, 2013.

[6] 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 *Security and Privacy (SP), 2018 IEEE Symposium on*, 2018.

[7] G. Singh, T. Gehr, M. Mirman, M. Püschel, and M. Vechev, "Fast and effective robustness certification," in *Advances in Neural Information Processing Systems 31*, 2018.

[8] 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 31*, 2018.

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

[10] P. Cousot and N. Halbwachs, "Automatic discovery of linear restraints among variables of a program," in *Proceedings of the 5th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, 1978.

[11] T.-W. Weng, H. Zhang, H. Chen, Z. Song, C.-J. Hsieh, D. Boning, and I. S. D. A. Daniel, "Towards fast computation of certified robustness for relu networks," in *International Conference on Machine Learning (ICML)*, 2018.

[12] H. Salman, G. Yang, H. Zhang, C. Hsieh, and P. Zhang, "A convex relaxation barrier to tight robustness verification of neural networks," *CoRR*, 2019.

[13] K. Dvijotham, R. Stanforth, S. Gowal, T. Mann, and P. Kohli, "A dual approach to scalable verification of deep networks," in *Proceedings of the Thirty-Fourth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-18)*, 2018.

[14] R. Bunel, I. Turkaslan, P. H. Torr, P. Kohli, and M. P. Kumar, "A unified view of piecewise linear neural network verification," in *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[15] R. Ehlers, "Formal verification of piece-wise linear feed-forward neural networks," in *Automated Technology for Verification and Analysis (ATVA)*, 2017.

[16] G. Katz, C. W. 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, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*, 2017.

[17] V. Tjeng, K. Y. Xiao, and R. Tedrake, "Evaluating robustness of neural networks with mixed integer programming," 2019.

[18] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Formal security analysis of neural networks using symbolic intervals," in *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018.*, 2018.

[19] W. Ruan, X. Huang, and M. Kwiatkowska, "Reachability analysis of deep neural networks with provable guarantees," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden.*, 2018.

[20] A. Raghunathan, J. Steinhardt, and P. S. Liang, "Semidefinite relaxations for certifying robustness to adversarial examples," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[21] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Efficient formal safety analysis of neural networks," in *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[22] G. Singh, T. Gehr, M. Püschel, and M. Vechev, "Boosting robustness certification of neural networks," in *International Conference on Learning Representations*, 2019.

[23] M. Mirman, T. Gehr, and M. Vechev, "Differentiable abstract interpretation for provably robust neural networks," in *International Conference on Machine Learning (ICML)*, 2018.

[24] E. Wong and Z. Kolter, "Provable defenses against adversarial examples via the convex outer adversarial polytope," in *Proceedings of the 35th International Conference on Machine Learning*, 2018.

[25] M. Lecuyer, V. Atlidakis, R. Geambasu, D. Hsu, and S. Jana, "Certified robustness to adversarial examples with differential privacy," *arXiv preprint arXiv:1802.03471*, 2018.

[26] B. Li, C. Chen, W. Wang, and L. Carin, "Second-order adversarial attack and certifiable robustness," *arXiv preprint arXiv:1809.03113*, 2018.

[27] J. Cohen, E. Rosenfeld, and Z. Kolter, "Certified adversarial robustness via randomized smoothing," in *Proceedings of the 36th International Conference on Machine Learning*, 2019.

[28] C. Qin, K. D. Dvijotham, B. O'Donoghue, R. Bunel, R. Stanforth, S. Gowal, J. Uesato, G. Swirszcz, and P. Kohli, "Verification of non-linear specifications for neural networks," in *International Conference on Learning Representations*, 2019.

[29] I. Goodfellow, H. Lee, Q. V. Le, A. Saxe, and A. Y. Ng, "Measuring invariances in deep networks," in *Advances in neural information processing systems*, 2009.

[30] A. Fawzi, S.-M. Moosavi-Dezfooli, and P. Frossard, "The robustness of deep networks: A geometrical perspective," *IEEE Signal Processing Magazine*, 2017.

[31] R. Alaifari, G. S. Alberti, and T. Gauksson, "ADef: an iterative algorithm to construct adversarial deformations," in *International Conference on Learning Representations*, 2019.

[32] C. Xiao, J.-Y. Zhu, B. Li, W. He, M. Liu, and D. Song, "Spatially transformed adversarial examples," in *International Conference on Learning Representations*, 2018.

[33] G. E. Hinton, A. Krizhevsky, and S. D. Wang, "Transforming auto-encoders," in *Artificial Neural Networks and Machine Learning – ICANN 2011*, 2011.

[34] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu, "Spatial transformer networks," in *Advances in Neural Information Processing Systems 28*, 2015.

[35] G. Singh, M. Püschel, and M. Vechev, "Fast polyhedra abstract domain," in *ACM SIGPLAN Notices*, 2017.

[36] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database," *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2010.

[37] P. Hansen and B. Jaumard, "Lipschitz optimization," in *Handbook of global optimization*, 1995.

[38] E. de Weerdt, Q. P. Chu, and J. A. Mulder, "Neural network output optimization using interval analysis," *IEEE Transactions on Neural Networks*, 2009.

[39] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," 2017.

[40] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009.

[41] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *International Conference on Learning Representations*, 2018.

[42] M. Mirman, G. Singh, and M. Vechev, "A provable defense for deep residual networks," *arXiv preprint arXiv:1903.12519*, 2019.

A Additional details of the method

A.1 Image transformations and their gradients

In this section we give an overview over the transformations used in this paper. For each of the transformations, we list parametrized form of the transformation, then Jacobian of the transformation function (with respect to both inputs and parameters) and finally the inverse function of the transformation.

Spatial transformations Here we list the equations for spatial transformations.

Rotation:

$$\begin{aligned}
R_\phi(x,y) &= \begin{pmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \\
\partial_{x,y} R_\phi(x,y) &= \begin{pmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{pmatrix} \\
\partial_\phi R_\phi(x,y) &= \begin{pmatrix} -x\sin\phi - y\cos\phi \\ x\cos\phi - y\sin\phi \end{pmatrix} \\
R_\phi^{-1}(x,y) &= \begin{pmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}
\end{aligned}$$

Translation:

$$\begin{aligned}
T_{v_1,v_2}(x,y) &= \begin{pmatrix} x+v_1 \\ y+v_2 \end{pmatrix} \\
\partial_{x,y} T_{v_1,v_2}(x,y) &= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \\
\partial_{v_1,v_2} T_{v_1,v_2}(x,y) &= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \\
T_{v_1,v_2}^{-1}(x,y) &= \begin{pmatrix} x-v_1 \\ y-v_2 \end{pmatrix}
\end{aligned}$$

Scaling:

$$\begin{aligned}
\mathsf{Sc}_{\lambda_1,\lambda_2}(x,y) &= \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \\
\partial_{x,y} \mathsf{Sc}_{\lambda_1,\lambda_2}(x,y) &= \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} \\
\partial_{\lambda_1,\lambda_2} \mathsf{Sc}_{\lambda_1,\lambda_2}(x,y) &= \begin{pmatrix} x & 0 \\ 0 & y \end{pmatrix} \\
\mathsf{Sc}_{\lambda_1,\lambda_2}^{-1}(x,y) &= \begin{pmatrix} \frac{1}{\lambda_1} & 0 \\ 0 & \frac{1}{\lambda_2} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}
\end{aligned}$$

Shearing:

$$\begin{aligned}
\mathsf{Sh}_m &= \begin{pmatrix} 1 & m \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \\
\partial_{x,y} \mathsf{Sh}_m(x,y) &= \begin{pmatrix} 1 & m \\ 0 & 1 \end{pmatrix} \\
\partial_m \mathsf{Sh}_m(x,y) &= \begin{pmatrix} y \\ 0 \end{pmatrix} \\
\mathsf{Sh}_m^{-1} &= \begin{pmatrix} 1 & -m \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}
\end{aligned}$$

Interpolation

$$I^{i,j}(x,y) = \frac{1}{4} \sum_{\substack{v \in \{i,i+2\} \\ w \in \{j,j+2\}}} p_{v,w}(2 - |v-x|)(2-|w-y|)$$

$$I(x,y) = \begin{cases} I^{i,j}(x,y) \text{ if } (x,y) \in D_{i,j}. \end{cases}$$

$$\partial_x I^{i,j}(x,y) = \frac{p_{i+2,j} - p_{i,j}}{2} + \frac{(y-\hat{y})}{4} A_{i,j}$$
$$\partial_y I^{i,j}(x,y) = \frac{p_{i,j+2} - p_{i,j}}{2} + \frac{(x-\hat{x})}{4} A_{i,j}$$

where $A_{i,j} = p_{i,j} - p_{i,j+2} - p_{i+2,j} + p_{i+2,j+2}$. Furthermore, the bilinear interpolation I is a polynomial of degree 2, which can be seen by performing the rewrite:

$$\begin{aligned} I^{i,j}(x,y) =& \tfrac{p_{i,j}}{4}(i+2-x)(j+2-y) + \tfrac{p_{i+2,j}}{4}(x-i)(j+2-y) \\ &+ \tfrac{p_{i+2,j}}{4}(i+2-x)(y-j) + \tfrac{p_{i+2,j+2}}{4}(x-i)(y-j) \end{aligned}$$

Brightness and contrast

$$\mathcal{P}_{\alpha,\beta}(v) = \alpha v + \beta$$
$$\partial_v \mathcal{P}_{\alpha,\beta}(v) = \alpha$$
$$\nabla_{\alpha,\beta} \mathcal{P}_{\alpha,\beta}(v) = (v, 0)^T$$

A.2 Computing an upper bound on the gradient

Here we explain how to compute an upper bound on the true gradient of the transformation. For the Lipschitz optimization in our running example, we want to calculate the upper bound on the gradient for $I \circ R_\phi(5,1) - a'_l - c'_l\phi$ on the interval $\phi \in [0, \frac{\pi}{4}]$. Calculating this upper bound amounts to interval propagation on the analytic gradient:

$$(\partial_1 I, \partial_2 I)|_D \cdot (\partial_\phi R_\phi)|_{\phi'} - c'_l$$

The abstraction for the area D is

$$D = R_{[0,\frac{\pi}{4}]}(5,1) = \begin{pmatrix} [2\sqrt{2}, 5] \\ [\frac{1}{\sqrt{2}}, 1 + \frac{5}{\sqrt{2}}] \end{pmatrix}.$$

We need to calculate the interval of partial derivatives of I for every $I^{i,j}$ separately and join the intervals in the end. The area D has non empty intersection with $\{A_{i,j}\}$ where $(i,j) \in \{1,3\} \times \{-1,1,3\}$. The estimate on $M_{3,1} = D \cap A_{3,1}$ using the concrete pixel values (Fig. 5e), the incline of the linear constraint $w'_l = -0.90$ (§4) and $\phi \in [0, \frac{\pi}{4}]$ we get

$$\begin{aligned} &([0.23, 0.47], [-0.83, -0.24]) \cdot \left(\begin{smallmatrix} [-4.53, -0.71] \\ [2.82, 5] \end{smallmatrix}\right) - 0.90 \\ &= [-7.18, -1.74]. \end{aligned}$$

Similar computation can be performed for the other interpolation areas. The resulting intervals from all interpolation areas are joined to make up the final result.

A.3 Proof of theorem 1

Proof. Notice that L_n is essentially a pointwise Monte Carlo estimate of the integral in L. Thus, For a fixed $(\boldsymbol{w}, b) \in W$ from the law of large numbers we know that for $\epsilon > 0$, $\Pr(|L_N(\boldsymbol{w}, b) - L(\boldsymbol{w}, b)| < \epsilon)$ tends to 0 as N tends to infinity.

Next, consider a finite number of pairs $(\boldsymbol{w_1}, b_1), \ldots, (\boldsymbol{w_k}, b_k)$. One could apply union bound on the above observation and show that $\Pr(\exists i; |L_N(\boldsymbol{w}_i, b_i) - L(\boldsymbol{w}_i, b_i)| > \epsilon)$ tends to 0 as N tends to infinity.

Finally, one could choose equidistant subdivision of W such that along each dimension of W we make cuts of width δ. Using this subdivision we obtain finite number of $(\boldsymbol{w_1}, b_1), ..., (\boldsymbol{w_k}, b_k)$. Note that for each $(\boldsymbol{w}, b) \in W$ there exists i such that $|\boldsymbol{w} - \boldsymbol{w_i}| < \delta$ and $|b - b_i| < \delta$. Then, from the definition of L_n we can obtain, using triangle and Cauchy-Schwarz inequality:

$$\begin{aligned}|L_n(\boldsymbol{w}, b) - L_n(\boldsymbol{w}_i, b_i)| &= |(b_i - b) - (\boldsymbol{w}_i - \boldsymbol{w})^T \frac{1}{N}\sum_{j=1}^{N} \boldsymbol{\kappa}_j| \\ &< |(b_i - b)| + |(\boldsymbol{w}_i - \boldsymbol{w})|_\infty |\frac{1}{N}\sum_{j=1}^{N} \boldsymbol{\kappa}_j|_1 \\ &< \delta + \delta \cdot R = \delta(1 + R)\end{aligned}$$

Here we denote by R maximum L_1 norm of the element in the parameter space (which exists because parameter space is bounded). Analogously, one can obtain that: $|L(\boldsymbol{w}, b) - L(\boldsymbol{w_i}, b_i)| < \delta(1 + R)$. One could choose width δ small enough such that $\delta(1 + R) < \epsilon$, i.e., $\delta < \frac{\epsilon}{1+R}$.

First, notice that L_N is a pointwise Monte Carlo estimate of the integral in L. Thus, due to the law of large numbers for a fixed $(\boldsymbol{w}, b) \in W$ and any $\epsilon > 0$, $\Pr(|L_N(\boldsymbol{w}, b) - L(\boldsymbol{w}, b)| \geq \epsilon)$ tends to 0 as N tends to infinity. Next, consider a finite number of weight-bias pairs $(\boldsymbol{w_1}, b_1), \ldots, (\boldsymbol{w_k}, b_k) \in W$. Using union bound on the above probability we know that:

$$\Pr(\exists i \in \{1, 2, ..., k\}, |L_N(\boldsymbol{w}_i, b_i|) - L(\boldsymbol{w}_i, b_i)| > \epsilon) \leq \sum_{j=1}^{k} \Pr(|L_N(\boldsymbol{w}_j, b_j|) - L(\boldsymbol{w}_j, b_j)| > \epsilon).$$

As each summand on the right goes to 0 as we increase N to the infinity, we can conclude that with high probability our estimate L_N is ϵ-close to true function L on all of the k weight-bias pairs.

Finally, we want to prove that our estimate $L_N(\boldsymbol{w}, b)$ is ϵ-close to the function $L(\boldsymbol{w}, b)$, with high probability (so far we have proved this only for finitely many $(\boldsymbol{w}, b)$). In order to show this, we choose equidistant subdivision of W such that along each dimension of W we make cuts of width δ. Using this subdivision we obtain finite number of $(\boldsymbol{w_1}, b_1), ..., (\boldsymbol{w_k}, b_k)$. Note that for each $(\boldsymbol{w}, b) \in W$ there exists i such that $|\boldsymbol{w} - \boldsymbol{w_i}| < \delta$ and $|b - b_i| < \delta$. Then, from the definition of L_N we can obtain, using triangle and Cauchy-Schwarz inequality:

$$\begin{aligned}|L_N(\boldsymbol{w}, b) - L_N(\boldsymbol{w}_i, b_i)| &= |(b_i - b) - (\boldsymbol{w}_i - \boldsymbol{w})^T \frac{1}{N}\sum_{j=1}^{N} \boldsymbol{\kappa}_j| \\ &< |(b_i - b)| + |(\boldsymbol{w}_i - \boldsymbol{w})|_\infty |\frac{1}{N}\sum_{j=1}^{N} \boldsymbol{\kappa}_j|_1 \\ &< \delta + \delta \cdot R \\ &= \delta(1 + R)\end{aligned}$$

Here R denotes the maximum L_1 norm of the element in the parameter space (which exists because parameter space is bounded). Analogously, one can obtain that: $|L(\boldsymbol{w}, b) - L(\boldsymbol{w_i}, b_i)| < \delta(1 + R)$. We can choose width δ small enough such that $\delta(1 + R) < \epsilon$, i.e., $\delta < \frac{\epsilon}{1+R}$.

Then:

$$\begin{aligned}&|L_N(\boldsymbol{w}, b) - L(\boldsymbol{w}, b)| \\ &\leq |L_N(\boldsymbol{w}, b) - L_n(\boldsymbol{w_i}, b_i)| + |L_N(\boldsymbol{w_i}, b_i) - L(\boldsymbol{w_i}, b_i)| + |L(\boldsymbol{w_i}, b_i) - L(\boldsymbol{w}, b)| \\ &< \epsilon + \epsilon + \epsilon \\ &< 3\epsilon.\end{aligned}$$

Now, we know that for finite subdivision of weights our estimate L_n is ϵ-close to L, with high probability. From the above, we get that if our subdivision is fine enough (δ small enough) then L_n is ϵ-close to L on the entire weights space W. This way we obtain that:

$$\Pr(\exists (w, b) \in W; |L_N(\boldsymbol{w}, b) - L(\boldsymbol{w}, b)| > \epsilon) \overset{N \to \infty}{\to} 0.$$

Now we obtained that for N large enough L_N is ϵ-close to L pointwise, with high probability. Finally, let $(\boldsymbol{w}', b')$ and $(\boldsymbol{w}^*, b^*)$ be minimums of L_N and L, respectively. We find that:

$$\begin{aligned}&L(\boldsymbol{w}', b') - L(\boldsymbol{w}^*, b^*)\\&= (L(\boldsymbol{w}', b') - L_N(\boldsymbol{w}', b')) + (L_N(\boldsymbol{w}', b') - L_N(\boldsymbol{w}^*, b^*)) + (L_N(\boldsymbol{w}^*, b^*) - L(\boldsymbol{w}^*, b^*))\\&< \epsilon + 0 + \epsilon\\&= 2\epsilon.\end{aligned}$$

□

A.4 Computation of interval pixel bounds

This section generalizes [9] to arbitrary combinations of simple geometric transformations. To compute interval bounds for each pixel in the transformed image with respect to a hyperrectangle of parameters

$$(\alpha, \beta, \boldsymbol{\mu}) \in h := [a_1, b_1] \times \cdots \times [a_n, b_n] \subset \mathbb{R}^n,$$

the algorithm first partitions the parameter space into s *splits* $\{h_k\}_{k\in[s]}$. Then, for each split h_k, it computes the interval bounds for every pixel value $\tilde{p}_{x,y}$ as follows ((x, y) fixed):

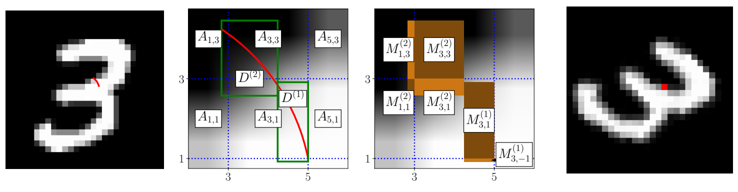
Step 1 Partition the current split h_k into r *refined splits* $\{h_{kl}\}_{l\in[r]}$ so to increase accuracy of approximation.

Step 2 For every $l \in [r]$, calculate a pair of intervals $D^{(l)} = [d_1, d_1'] \times [d_2, d_2'] \subset \mathbb{R}^2$ which approximate the reachable coordinates induced by $\{\mathcal{T}_{\boldsymbol{\mu}}^{-1}(x, y) \mid \forall(\alpha, \beta, \boldsymbol{\mu}) \in h_{kl}\}$. To accomplish this, we use the fact we can easily obtain the closed form of the inverse of the transformations we consider (Appendix A.1). For example, for rotations, $R_{-\phi} = R_{\phi}^{-1}$. We can then propagate the interval bounds through this inverse.

Step 3 Calculate the interval approximation ι_{kl} for every $D^{(l)}$. This is done by propagating the pair of intervals $D^{(l)}$ through $\mathcal{P}_{[a_1,b_1],[a_2,b_2]} \circ I$, that is

$$\begin{aligned}\iota_{kl} &= \mathcal{P}_{[a_1,b_1],[a_2,b_2]} \circ I(D^{(l)})\\&= \mathcal{P}_{[a_1,b_1],[a_2,b_2]}\left(\bigcup_{i,j} I^{i,j}(M_{i,j}^{(l)})\right).\end{aligned}$$

Here we have that $M_{i,j}^{(l)} := D^{(l)} \cap A_{i,j}$, that is, we intersect the approximation of the reachable coordinates $D^{(l)}$ with the interpolation region $A_{i,j}$.



(a) Original image (b) Interpolation (c) Intersected regions (d) Rotated image

5	0	0	0.35
3	0.36	0.83	0.97
1	0.99	0.99	0.99
-1	0.99	0.99	0.99
	1	3	5

(e) Original pixel values

Figure 5: Image rotation by $-\frac{\pi}{4}$ degrees. Here, (a) shows the original image, while (b) and (c) show an excerpt of (a) focusing on particular interpolation regions. They also include regions computed by the interval analysis. Finally, (d) shows the transformed rotated image.

Step 4 Finally, we combine (join) all interval approximations ι_{kl} into a single interval $\iota_k = \cup_{l\in[r]}\iota_{kl}$, capturing the possible values $\tilde{p}_{x,y} = \mathcal{I}_{\alpha,\beta,\boldsymbol{\mu}}(x,y)$ that pixel (x,y) can take on for the split h_k.

As we show later, the interval range ι_k computed for each of the resulting splits h_k can then be used for certification or for adversarial example generation.

Running example To illustrate the above steps on our example, consider again pixel $(5,1)$ and the angle range $\phi \in [-\frac{\pi}{2}, 0]$. This range corresponds, using $R_{-\phi} = R_{\phi}^{-1}$, to $\phi \in [0, \frac{\pi}{2}]$. For $s = 2$, we obtain splits $[0, \frac{\pi}{4}]$ (the small red arch in Fig. 2a which is zoomed in in Fig. 2b) and $[\frac{\pi}{4}, \frac{\pi}{2}]$. As discussed above, the interval approximations are produced independently for both splits. Let us consider the split $[0, \frac{\pi}{4}]$ and refine it by splitting it again in half ($r = 2$). The approximation $D^{(1)}$ for the first refined split $h_{11} := [0, \frac{\pi}{8}]$ is

$$\begin{aligned} D^{(1)} = R_{[0,\pi/8]}(5,1) &= \begin{pmatrix} \cos(h_{11}) & -\sin(h_{11}) \\ \sin(h_{11}) & \cos(h_{11}) \end{pmatrix}\begin{pmatrix} 5 \\ 1 \end{pmatrix} \\ &= \begin{pmatrix} [\cos\frac{\pi}{8},1] & [-\sin\frac{\pi}{8},0] \\ [0,\sin\frac{\pi}{8}] & [\cos\frac{\pi}{8},1] \end{pmatrix}\begin{pmatrix} 5 \\ 1 \end{pmatrix} = \begin{pmatrix} \left[5\cos\frac{\pi}{8}-\sin\frac{\pi}{8},5\right] \\ \left[\cos\frac{\pi}{8},1+5\sin\frac{\pi}{8}\right] \end{pmatrix}. \end{aligned}$$

Similarly, one obtains $D^{(2)}$. The green rectangles shown in Fig. 2b capture the values of $D^{(1)}$ and $D^{(2)}$. Here, $D^{(1)}$ intersects the interpolation regions $A_{3,-1}$ and $A_{3,1}$, shown in Fig. 5c. The interval approximations for the values on $M^{(1)}_{3,-1} = D^{(1)} \cap A_{3,-1}$ using Eq. (1) are

$$\begin{aligned} I^{3,-1}\left(M^{(1)}_{3,-1}\right) &= I([5\cos\frac{\pi}{8} - \sin\frac{\pi}{8}, 5], [\cos\frac{\pi}{8}, 1]) \\ &\approx [0.58, 1], \end{aligned}$$

where the upper bound of the interval was cut back to 1. Similarly, we obtain $I^{3,1}\left(M^{(1)}_{3,1}\right) \approx [0.02, 1]$. Thus, the interval approximation for pixel $(5,1)$ on the first refinement split $\phi \in [0, \frac{\pi}{8}]$ is $[0.58, 1] \cup [0.02, 1] = [0.02, 1]$.

The approximation of the second refined split $D^{(2)}$ intersects four interpolation regions: $A_{1,1}$, $A_{1,3}$, $A_{3,1}$ and $A_{3,3}$. This intersection is also shown in Fig. 5c. Via similar calculations as above we obtain

$$\begin{aligned} I^{1,1}\left(M^{(2)}_{1,1}\right) &\approx [0.61, 1]\,, & I^{1,3}\left(M^{(2)}_{1,3}\right) &\approx [0.20, 0.87]\,, \\ I^{3,1}\left(M^{(2)}_{3,1}\right) &\approx [0.25, 1], & I^{3,3}\left(M^{(2)}_{3,3}\right) &\approx [0.08, 1]\,. \end{aligned}$$

Thus, the interval approximation for the second refinement split $\phi \in [\frac{\pi}{8}, \frac{\pi}{4}]$ is $[0.08, 1]$.

In the final step 4, the two intervals $[0.08, 1]$ and $[0.02, 1]$ are combined to obtain the range $[0.02, 1]$ for the split $[0, \frac{\pi}{4}]$ corresponding to $0.02 \leq \mathcal{I}_{[-\frac{\pi}{4},0]}(5,1) \leq 1$.

In the next section, we show how to improve the precision of the interval overapproximation $[c_l, c_u]$, corresponding to $c_l \leq \mathcal{I}_{\alpha,\beta,\boldsymbol{\mu}}(x,y) \leq c_u$, by using linear upper and lower constraints to capture the correlation between pixel values and parameters $(\alpha, \beta, \boldsymbol{\mu})$.

A.5 Lipschitz optimization with bound refinement

Here we provide more details of our algorithm for Lipschitz optimization. In order to obtain a sound linear constraint, we need to find sound overapproximation of maximum of the function $f(\boldsymbol{\kappa}) = b'_l + \boldsymbol{w}'^T_l \boldsymbol{\kappa} - \mathcal{I}_{\boldsymbol{\kappa}}(x,y)$ on a hyperrectangle domain. The result is an interval $[f_{\text{max}}, f_{\text{max}} + \epsilon]$ which contains true maximal value $f^\star$ of function f.

Lipschitz continuity A function $f\colon D \subset \mathbb{R}^n \to \mathbb{R}$ is Lipschitz continuous with a Lipschitz constant $L \in \mathbb{R}_{\geq 0}$ if $\|f(\boldsymbol{y}) - f(\boldsymbol{x})\| \leq L\|\boldsymbol{y} - \boldsymbol{x}\|$ for all $\boldsymbol{x}, \boldsymbol{y} \in D$.

If D is compact, the Lipschitz constant of a smooth function f is equal to $\max_{\boldsymbol{x}\in D}\|\nabla f|_{\boldsymbol{x}}\|$. We also use this equation for the (non-smooth) bilinear interpolation I. As mentioned in Section 3, I is smooth on every interior of $A_{i,j}$. Further, $I|_{A_{i,j}}$ can be extended smoothly at the boundary of $A_{i,j}$ by $I^{i,j}$. This allows us to approximate the Lipschitz constant L of I over some compact region $D \subset \mathbb{R}^2$, using the approximated gradient $\nabla^D I = \bigcup_{i,j} \nabla I^{i,j}(D \cap A_{i,j})$, by $\|\nabla^D I\|$, where the approximation can be computed using any overapproximation based analysis (we use intervals, but one can plug in other approximations such as zonotope).

Lipschitz optimization with iteratively refined bounds The objective we aim to solve is finding the ϵ optimal value $f_{\max}$ of the Lipschitz continuous function f over a hyperrectangle $h = \prod_{i=1}^{n}[a_i, b_i] \subset \mathbb{R}^n$.

To address this problem, we adapt a branch-and-bound algorithm for solving constrained (multidimensional) optimization tasks [38] to piecewise differentiable Lipschitz continuous functions. In particular, we show how to leverage any overapproximation based analysis (e.g., interval analysis) to soundly approximate local gradients (Lipschitz constants). This process is repeated at every step, thus obtaining more refined bounds for the local gradients at every iteration. Overall, this method leads to significantly fewer iterations of the optimization algorithm, improves convergence speed at flat areas, and removes the need to know the global Lipschitz constant in advance.

The inputs to the algorithm (Algorithm 1) are a Lipschitz continuous function f, the hyperrectangle h over which we optimize and the number of cuts $k \in \mathbb{N}_{\geq 2}$ in which a hyperrectangle will be split to gain precision. The idea is to constrain the range of the true maximum value $f^\star$ by increasing its lower bound $f_{\max}$ (lower bound on all of h) while decreasing all of its upper bounds $f_{h'}^{\text{bound}}$ on the subrectangles h' that need to be considered.

We use a priority queue q to store pairs of the form $(h', f_{h'}^{\text{bound}})$, where h' corresponds to a subrectangle of h, potentially containing the true maximal value $f^\star$, and $f_{h'}^{\text{bound}}$ corresponds to an upper bound (Eq. (9) or Eq. (10)) of f in h'. The queue is ordered by the upper bounds. To improve our bounds on $f^\star$, we pop the top of the queue $(h', f_{h'}^{\text{bound}})$ and split h' into k splits $h'_1, \ldots, h'_k$ according to Eq. (11) or Eq. (12). The lower bound $f_{\max}$ is updated to f evaluated at a center of h'_i, if this would increase $f_{\max}$. The new pairs $(h'_i, f_{h'_i}^{\text{bound}})$ are added to the queue as long as the new bound $f_{h'_i}^{\text{bound}}$ is larger than $f_{\max} + \epsilon$, and thus may help to increase $f_{\max}$ by more than ϵ. The algorithm terminates if the largest upper bound $f_{h'_i}^{\text{bound}}$ and the lower bound $f_{\max}$ are ϵ-close.

Bounds The function f on the hyperrectangle $h' = [a'_1, b'_1] \times \cdots \times [a'_n, b'_n]$ can be bounded using the (commonly used) Cauchy-Schwarz bound or the triangle bound

$$f_h^{\text{cs-bound}}(h, \nabla f) = f(c_h) + \frac{L}{2}\sqrt{\sum_{i=1}^{n}(b_i - a_i)^2} \tag{9}$$

$$f_h^{\text{t-bound}}(h, \nabla f) = f(c_h) + \frac{1}{2}\sum_{i=1}^{n} v_i(b_i - a_i) \tag{10}$$

where we compute the approximation (e.g., interval or zonotope) of the gradient $\nabla^h f$ on h to calculate L as the upper bound of the approximation $\|\nabla^h f\|$ and v_i is the upper bound of the approximation $\|\partial_i^h f\|$. We use the triangle bound as it is tighter.

Triangle bound Let $f\colon D \to \mathbb{R}$ be a smooth function and D be a convex set. Using the mean value theorem, we know that for all $\boldsymbol{x}, \boldsymbol{y} \in D$ there exists a $\boldsymbol{z} \in D$ such that

$$\begin{aligned} f(\boldsymbol{y}) - f(\boldsymbol{x}) = \nabla f(\boldsymbol{z})^T(\boldsymbol{x} - \boldsymbol{y}) &= \sum_i \partial_i f(z)(\boldsymbol{x}^{(i)} - \boldsymbol{y}^{(i)}) \\ &\leq \sum_i \max_{\boldsymbol{z} \in D} |\partial_i f(z)||\boldsymbol{x}^{(i)} - \boldsymbol{y}^{(i)}| \end{aligned}$$

which results into

$$f(\boldsymbol{y}) \leq f(\boldsymbol{x}) + \sum_i \max_{\boldsymbol{z} \in D} |\partial_i f(z)||\boldsymbol{x}^{(i)} - \boldsymbol{y}^{(i)}|.$$

We note, that in our case $f = \mathcal{I}$ as in Section 3, the inequality still holds: The points where f is not differentiable can be handled by applying the inequality piecewise.

Table 5: Hyperparameters used in our experiments. Splits denotes number of domain splits per dimension (same for Interval and DEEPG). k is number of refinements per dimension used for Interval baseline. Last two columns show number of samples n used to solve LP and ϵ in Lipschitz optimization in DEEPG.

		Splits	k (Interval)	n (DEEPG)	ϵ (DEEPG)
MNIST	R(30°)	10	10 000	1 000	0.0001
	T(2,2)	11	150	2 000	0.00001
	Sh(2), R(2°), Sc(2), B(2, 0.001)	1	12	1 000	0.006
FashionMNIST	Sc(20)	10	10 000	1 000	0.0001
	R(10), B(2, 0.01)	4	25	2 000	0.002
	Sc(3), R(3), Sh(2)	2	35	1 000	0.001
CIFAR-10	R(10°)	20	10 000	1 000	0.00001
	R(2), Sh(2)	2	50	1 000	0.0001
	Sc(1), R(1°), B(1, 0.001)	2	12	1 000	0.001

Partitioning The hyperrectangle h can be split into k hyperrectangles of equal size by cutting one of its edges l into k equal parts. The common choice for l is to use the longest edge. We refine this by weighing the edge length with gradient information

$$l = \arg\max_{l}\,(b'_l - a'_l) \qquad \text{largest edge} \tag{11}$$

$$l = \arg\max_{l}\, v_l(b'_l - a'_l) \qquad \text{largest weighted edge} \tag{12}$$

B Additional details for experimental evaluation

In this section we list additional details which are necessary to reproduce our experimental results.

B.1 Polyhedra transformers

Here we describe Polyhedra transformers used in our experiments. Note that all geometric transformations are linear which is why polyhedra approximation for the transformation itself does not lose precision. After the transformation is applied, we obtain exact linear expression for new position of every pixel. Next operation in the sequence is bilinear interpolation which is polynomial of degree 2 and involves multiplication between two linear expressions (addition which is part of the polynomial is again exact in polyhedra). Multiplication cannot be captured exactly and we need to decide how to lose precision. We choose standard solution of concretizing one of the polyhedra expressions to interval (we concretize one which produces interval with smaller width).

Finally, we need to join polyhedra constraints over all interpolation regions. This amounts to computing lower and upper convex hull of polyhedra constraints. This is challenging problem, however for the case of 1-dimensional transformations there is relatively easy solution. If parameter of the transformation is in the interval $[l, u]$ we intersect polyhedra constraints with lines $y = l$ and $y = u$ and choose smallest (largest) intercept with each of the vertical lines and join the intercepts, thus forming a new linear lower bound.

B.2 Parameter configurations for experiments

Table 7 shows architectures used for our experiments on MNIST, Fashion-MNIST and CIFAR-10 datasets. In Table 5 we show parameter configurations for all of our experiments. As noted before, parameters are chosen in such a way that runtime of both DEEPG and interval analysis are roughly the same. We did not observe a significant difference in performance when changing the parameters. We report runtime for our experiments in Table 6.

MNIST In the experiments on MNIST, in all of the training methods we consider, the network is trained for 100 epochs using batch size 128. We use Adam optimizer and with initial learning rate 0.001 which we decay by 0.5 every 10 epochs. For the experiments which use PGD training, we

Table 6: For every experiment, breakdown of runtime between time taken to compute the constraints using DEEPG and time taken to certify the network using DeepPoly.

		Runtime (seconds)	
		Constraint generation (DEEPG)	Certification (DeepPoly)
MNIST	R(30)	10	25
	T(2, 2)	206	57
	Sc(5), R(5), B(5, 0.01)	162	14
	Sh(2), R(2), Sc(2), B(2, 0.001)	50	1
Fashion-MNIST	Sc(20)	9	6
	R(10), B(2, 0.01)	174	47
	Sc(3), R(3), Sh(2)	95	6
CIFAR-10	R(10)	49	68
	R(2), Sh(2)	13	14
	Sc(1), R(1), B(1, 0.001)	179	60

Table 7: Architectures used in experimental evaluation.

MNIST	FashionMNIST	CIFAR-10
CONV 32 4×4 + 2	CONV 32 4×4 + 1	CONV 32 4×4 + 1
CONV 64 4×4 + 2	CONV 32 4×4 + 2	CONV 32 4×4 + 2
FC 200	CONV 64 4×4 + 2	CONV 64 4×4 + 2
FC 10	FC 150	FC 150
	FC 10	FC 10

train with $\epsilon = 0.1$ and perform 40 steps with the step size 0.005 in every iteration. We also use L_1 regularization factor of 0.00005. For the experiments with FashionMNIST we use exactly the same setup.

CIFAR-10 In the experiments on CIFAR-10, in all of the training methods we consider the network is trained for 100 epochs using batch size 128. We use SGD optimizer and with initial learning rate 0.01 which we decay by 0.5 every 10 epochs. For the experiments which use PGD training, we train with $\epsilon = 0.005$ and perform 7 steps with the step size 0.002 in every iteration. We also use L_1 regularization factor of 0.00001, except for the experiment with rotation and shearing where we use 0.00005.