Summary Slides

Certifying Geometric Robustness of Neural Networks

Presented by
Zhong Yuyi

Perturbation Type

Mislav B, etc. Certifying Geometric Robustness of Neural Networks. NeurIPS 2019.

- A wide range of methods have been proposed to certify robustness of neural networks <u>against adversarial examples</u>
- None of these works consider geometric transformations, mostly focus on L_∞ perturbation (<u>change pixel intensity</u>)
- There has been considerable research in empirical quantification of geometric robustness of neural networks
- Offer only empirical evidence of robustness. Instead, our focus is to provide formal guarantees

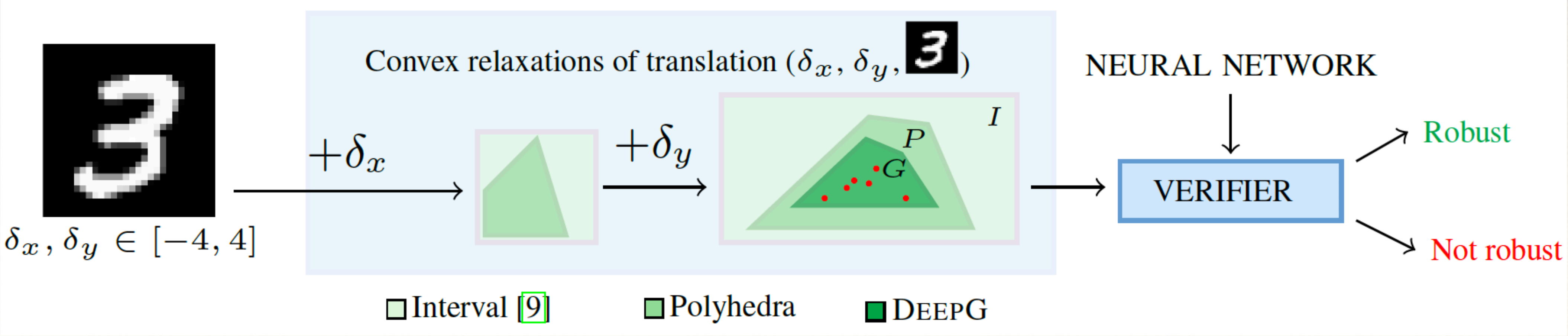
Contributions

Mislav B, etc. Certifying Geometric Robustness of Neural Networks. NeurIPS 2019.

1. They propose a new method to compute sound and asymptotically optimal linear relaxations for <u>any composition</u> of transformations
2. Method is based on a novel combination of sampling and optimization.
3. It certifies significantly more complex geometric transformations than existing methods on both <u>defended and undefended</u> networks while <u>scaling</u> to large architectures

Geometric Robustness

- Recent work has shown that by using natural transformations (e.g., rotations), one can generate adversarial examples that cause the network to misclassify the image.
- To address this issue, one would ideally like to prove that a given network is free of such geometric adversarial examples.
- Eg: any image obtained by translating the original image by some $\delta_x, \delta_y \in [-4,4]$ is classified to label 3.
- Key technique: I is a convex approximation of the perturbed images produced after geometric transformations



Convex Approximation of Adversarial Region

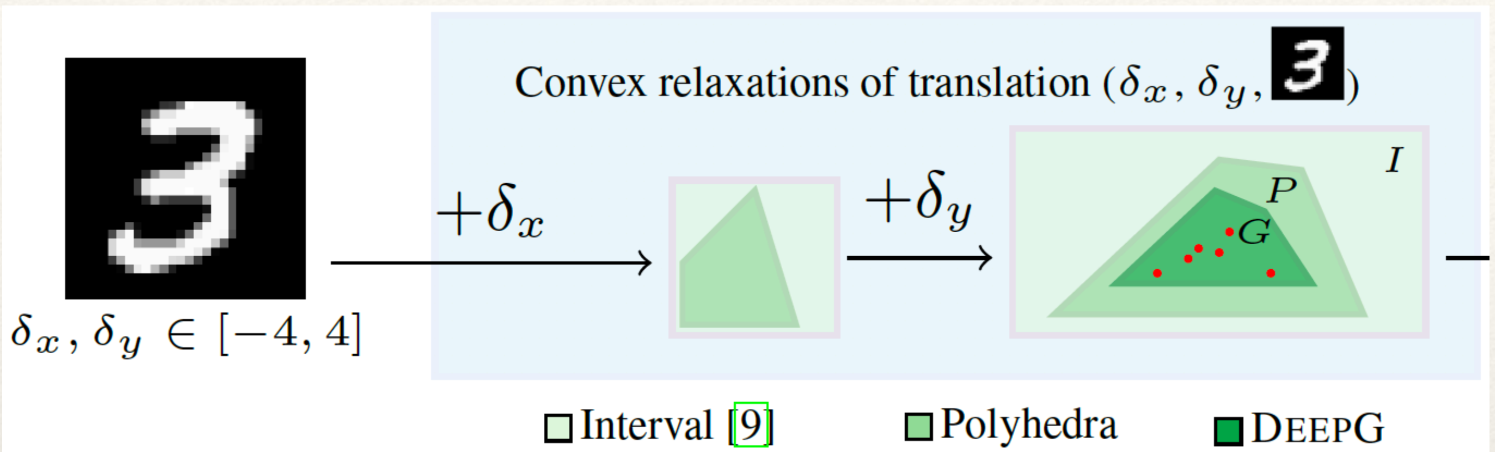


Fig. A

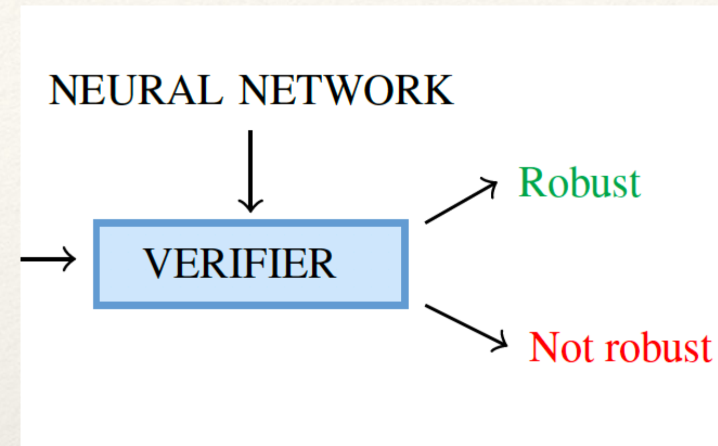


Fig. B

The whole verification as two processes: apply perturbation and verify robustness on adversarial region

- Fig. A: process of finding the convex shape capturing all perturbed image (the convex is the representation of adversarial region)
- Fig. B: feed the convex shape into the verifier, execute the abstract transformer layer through layer, determine the robustness based on the abstract execution result.

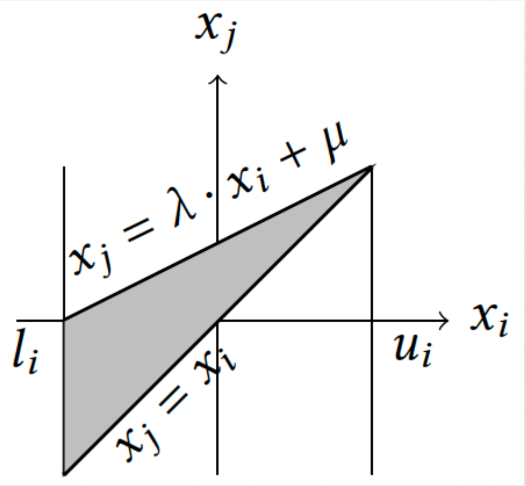
Convex Approximation of Adversarial Region

The whole verification as two processes: find the adversarial region given perturbation and verify robustness on adversarial region.

DeepPoly: polyhedral abstract domain, L_∞ perturbation

Step 1: find the convex shape, given L_∞

- $-\epsilon \leq x_i - x_i' \leq \epsilon$, each pixel has intensity within range of $[x_i - \epsilon, x_i + \epsilon]$
- Trivial to get the polyhedra from the interval
- $[-1,1]$ to $l_i = -1$, $u_i = 1$, $a_i^{\leq} = -1$, $a_i^{\geq} = 1$



Step 2: feed adversarial region to the polyhedral abstract transformer

- Propagate the polyhedral shape layer through layer
- Get the abstract value of output neuron
- See if the robustness holds

Challenge for geometric perturbation is in step 1.

Geometric perturbation could be a composition of rotation, translation, shearing and scaling transformations.

It is non-trivial to get the convex shape that captures such complex transformations.

Convex Approximation

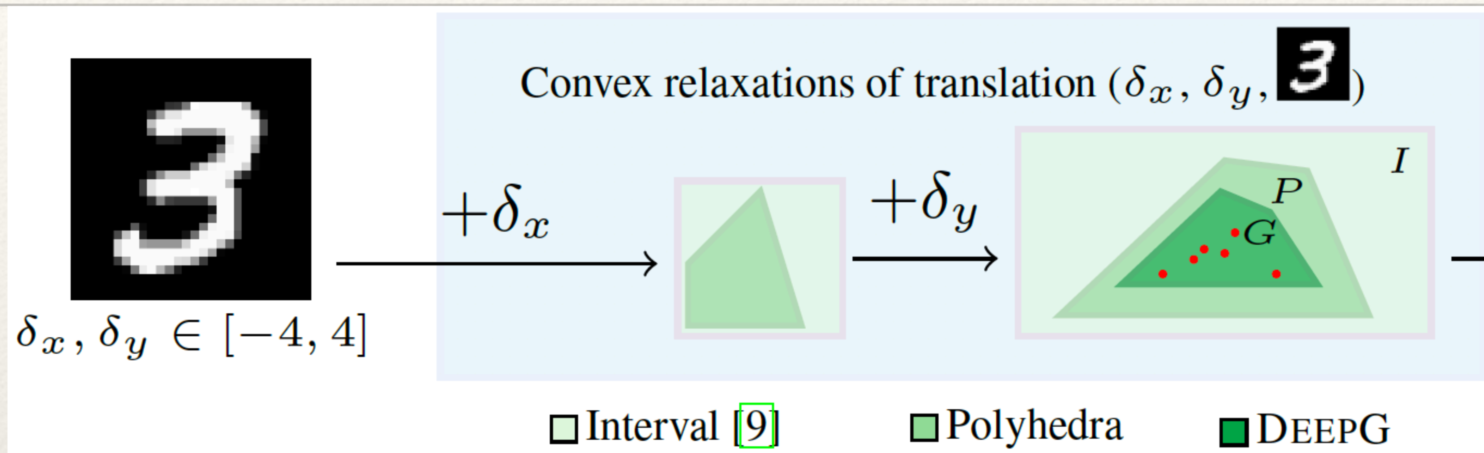


Fig. A

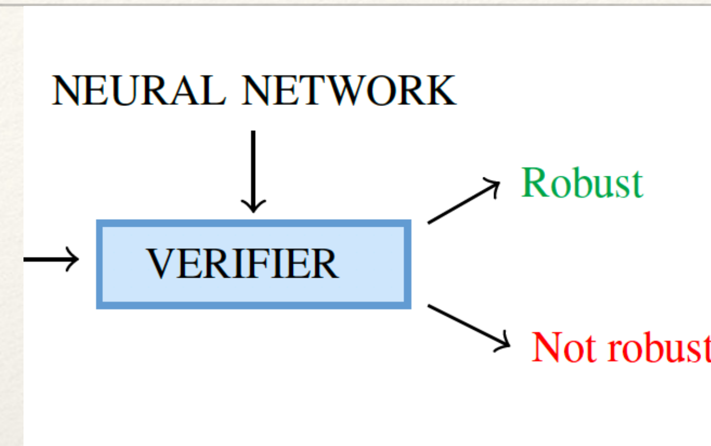


Fig. B

The main contribution of the paper is in Fig. A.

- Propagates the image through every component of the transformation (δ_x, δ_y) using interval bound propagation (or tighter relaxation based on Polyhedra)
- Bound propagation accumulates loss for every intermediate result, often producing adversarial regions that are too coarse to allow the neural network verifier to succeed
- A new method based on sampling and optimization which computes a convex approximation for the entire composition of transformations
- The key idea: to sample the parameters of the transformation, 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).

Geometric Image Transformations

- A geometric image transformation consists of a parameterized spatial transformation T_μ, an interpolation I which ensures the result can be represented on a discrete pixel grid, and parameterized changes in brightness and contrast $P_{\alpha,\beta}$
- T_μ is a composition of rotation, translation, shearing and scaling transformations (affine transformation)
- T_μ Geometric example: transformation function (w.r.t. inputs and parameters) and the inverse function

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

$$T_{v_1,v_2}(x,y) = \begin{pmatrix} x+v_1 \\ y+v_2 \end{pmatrix}$$

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

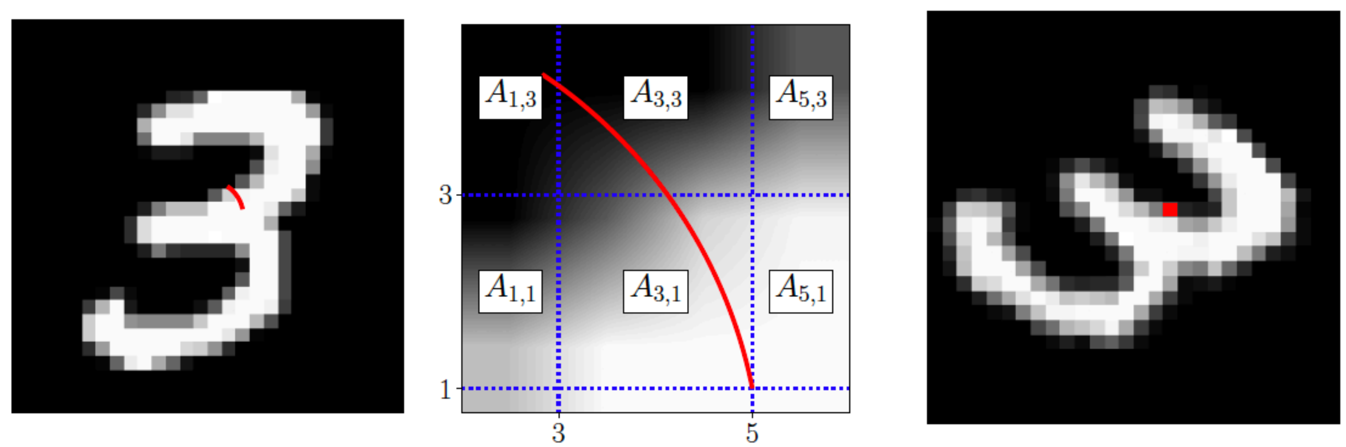
$$T_{v_1,v_2}^{-1}(x,y) = \begin{pmatrix} x-v_1 \\ y-v_2 \end{pmatrix}$$

Geometric Image Transformations

- Changes in brightness and contrast $P_{\alpha,\beta}(v) = \alpha v + \beta$ (Brightness increases in β and contrast increases α times
- What is an interpolation I? (Put aside first)
- The pixel value $\tilde{p}_{x,y}$ of the transformed image can be obtained by
  - (i) calculating the preimage of (x, y) under T_μ, using inverse function
  - (ii) interpolating the resulting coordinate using I to obtain a intermediate pixel value v
  - (iii) applying the changes in contrast and brightness via $P_{\alpha,\beta}(v) = \alpha v + \beta$ to obtain the final pixel value $\tilde{p}_{x,y} = I_{\alpha,\beta,\mu}(x, y)$
- The three steps are captured by $I_{\alpha,\beta,\mu}(x, y) = P_{\alpha,\beta} \circ I \circ T_\mu^{-1}(x, y)$
- I takes coordinates (x, y) as input, outputs pixel value v

Interpolation

- To ease presentation, we assume the image consists of an even number of rows and columns; Pixel coordinates are odd integers (all results hold in general case).
- Pixel value computation $I_{\alpha,\beta,\mu}(x,y) = P_{\alpha,\beta} \circ I \circ T_{\mu}^{-1}(x,y)$, suppose T_{μ} is rotating the original image by an angle $\phi = -\dfrac{\pi}{4}$, $P_{\alpha,\beta}(v)$ is identity
- $\tilde{p}_{5,1} = I \circ T_{-\frac{\pi}{4}}^{-1}(5,1)$



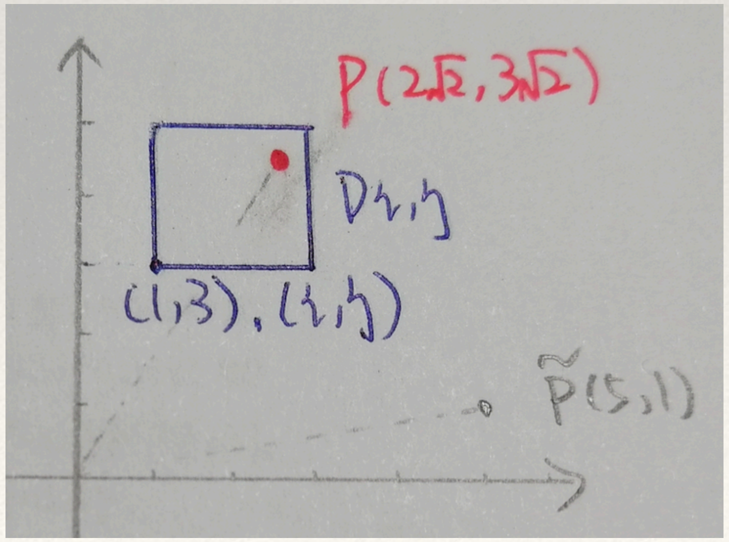
(a) Original image (b) Interpolation (c) Rotated image

Interpolation

- $\tilde{p}_{5,1} = I \circ T^{-1}_{-\frac{\pi}{4}}(5,1)$
- The preimage of point (5,1) produces the point $(2\sqrt{2}, 3\sqrt{2})$ with non-integer coordinates.
- $\tilde{p}_{5,1} = I \circ T^{-1}_{-\frac{\pi}{4}}(5,1) = I(2\sqrt{2}, 3\sqrt{2})$
- Interpolation function $I : R^2 \to [0,1]$ evaluated on a coordinate $(x, y) \in R^2$
- An interpolation region $A_{i,j} := [i, i+2] \times [j, j+2]$ which contains pixel (x, y)
- A region has four corners
- The pixel value is the normalised, weighted sum of pixel value of the four corners
- $\tilde{p}_{5,1} = I(2\sqrt{2}, 3\sqrt{2}) \approx 0.30$

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

$$I(x,y) = \left\{ I^{i,j}(x,y) \text{ if } (x,y) \in D_{i,j}. \right.$$



Certification Framework

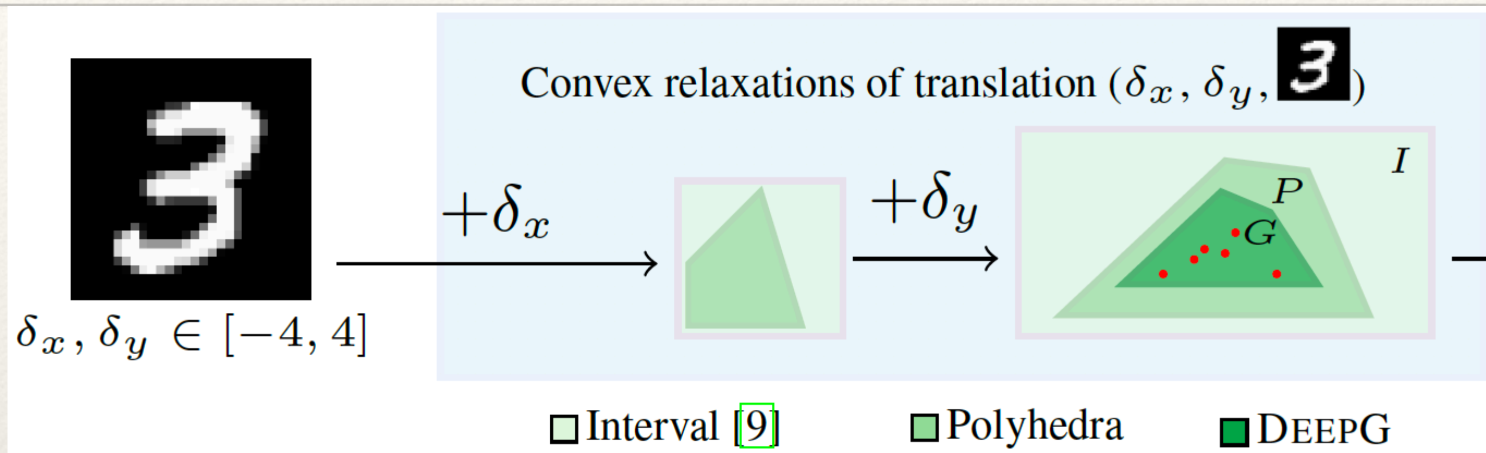


Fig. A

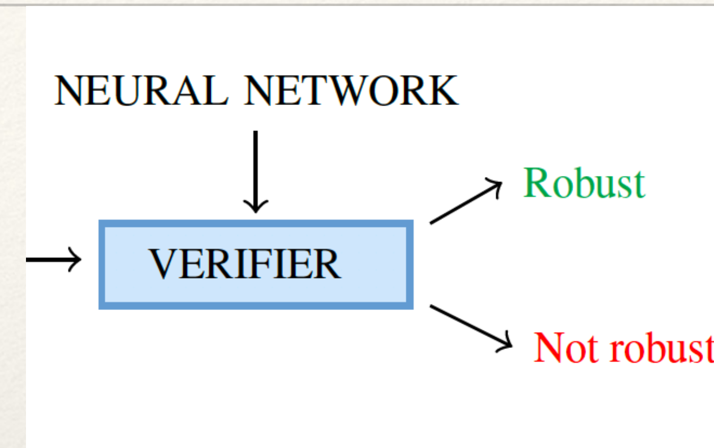
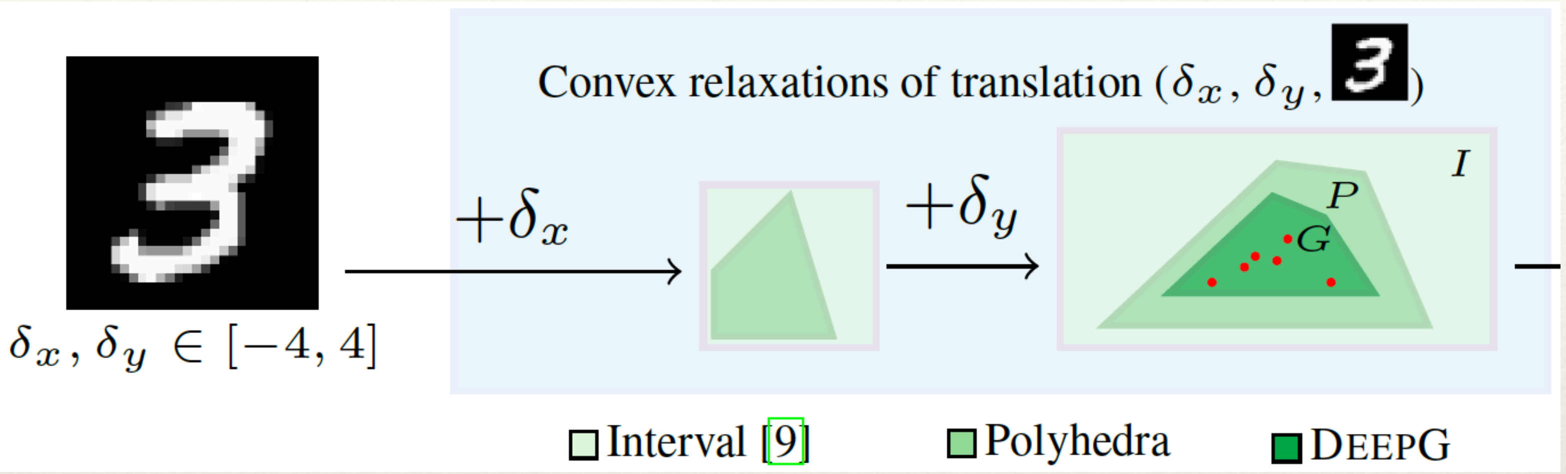


Fig. B

- The verifier needs to receive a convex shape of all possible inputs to the network
- The goal is to compute a convex relaxation of all possible images obtainable via the transformation $I_{\alpha,\beta,\mu}$
- Then this relaxation can then be provided as an input to a neural network verifier DeepPoly.

Optimal linear Constraints



- Geometric transformation is a composition/sequence of several operations. Propagating the image through every operation will accumulate loss at each step.
- Key insight: define an optimization problem where its solution is the optimal lower and upper linear constraint for the entire sequence.
- To solve optimization problem, they leverage sampling and linear programming
- The method produces, for every pixel, asymptotically optimal lower and upper linear constraints for the entire composition of transformations

Split the Optimization Problem

- To compute linear constraints for every pixel value
- Split the hyper-rectangle h representing the set of possible parameters (α, β, μ) into s splits $\{h_k\}_{k \in [s]}$.
- Eg: rotation with angle $\phi \in [0, \frac{\pi}{2}]$, choose $s = 2$, getting splits $[0, \frac{\pi}{4}]$ and $[\frac{\pi}{4}, \frac{\pi}{2}]$
- Compute sound lower and upper linear constraints for the pixel value $I_\kappa(x, y)$ for a given pixel (x, y), where these constraints will be linear in the parameters $\kappa = (\alpha, \beta, \mu) \in h_k$
- Eg: the constraint will be linear in parameter ϕ
- Optimal and sound linear (lower and upper) constraints for $I_\kappa(x, y)$ to be a pair of hyperplanes fulfilling

$$
\begin{aligned}
\boldsymbol{w}_l^T \boldsymbol{\kappa} + b_l \leq \mathcal{I}_{\boldsymbol{\kappa}}(x, y) \qquad \forall \boldsymbol{\kappa} \in h_k \\
\boldsymbol{w}_u^T \boldsymbol{\kappa} + b_u \geq \mathcal{I}_{\boldsymbol{\kappa}}(x, y) \qquad \forall \boldsymbol{\kappa} \in h_k,
\end{aligned}
$$

Split the Optimization Problem

- Optimal and sound linear (lower and upper) constraints for $I_\kappa(x, y)$ 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

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

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

- 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 to obtain lower constraints (upper constraints are computed analogously)

Compute Lower Bound

1. Compute a potentially unsound constraint using sampling and LP
2. Bounding the maximum violation
3. Compute a sound linear constraint

- Step 1: To generate a reasonable but a potentially unsound linear constraint, sample parameters $\kappa_1, \ldots, \kappa_N$ from h_k, and compute the constraint using

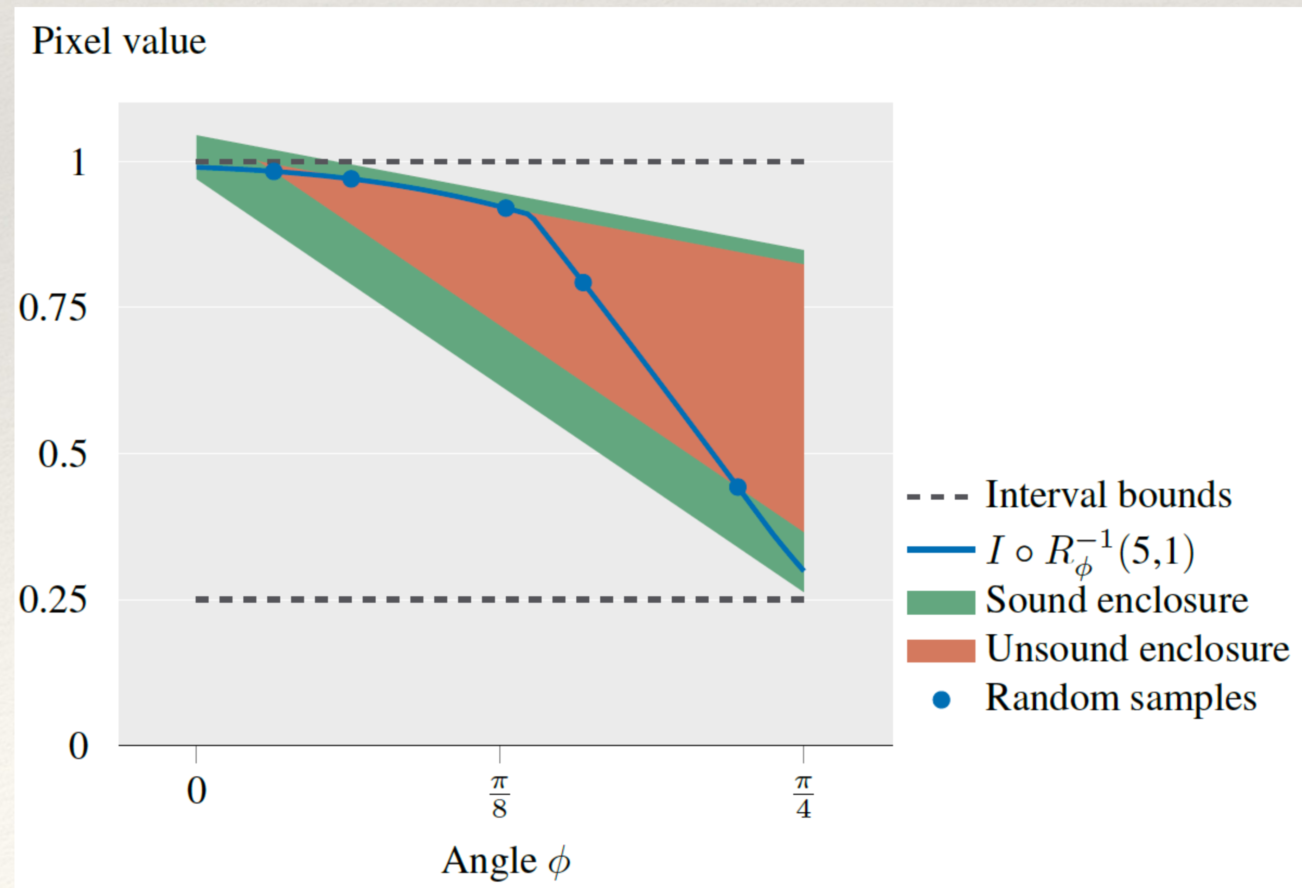
$$b_l + \boldsymbol{w}_l^T \boldsymbol{\kappa}_i \leq \mathcal{I}_{\boldsymbol{\kappa}_i}(x, y) \quad \forall i \in [N].$$

$$\min_{(\boldsymbol{w}_l, b_l) \in W} \frac{1}{N} \sum_{i=1}^{N} \left(\mathcal{I}_{\boldsymbol{\kappa}_i}(x, y) - (b_l + \boldsymbol{w}_l^T \boldsymbol{\kappa}_i) \right)$$

- The same as (3) and (5) (in the continuous h_k domain)
- This problem can be solved exactly using linear programming (LP).
- Lead to a potentially unsound constraint $b_l' + w_l'^T \kappa$ (it may violate the constraint at non-sampled points)

Compute Unsound Constraint

- Step 1: To generate a <u>reasonable but a potentially unsound</u> linear constraint, sample parameters $\kappa_1, \ldots, \kappa_N$ from h_k, and compute the constraint using
- Eg: Consider split $\phi \in [0, \frac{\pi}{4}]$, we sample random points {0.1, 0.2, 0.4, 0.5, 0.7} from $[0, \frac{\pi}{4}]$
- Evaluate $I \circ T_\phi^{-1}(5,1)$ on these points, obtaining pixel value $\{0.98, 0.97, 0.92, 0.79, 0.44\}$
- These sampled nodes correspond to the blue dots. Solving LP yields $b_l' = 1.07, w_l' = -0.90$
- Similarly compute upper bound
- Get the orange unsound enclosure
- Some points on the blue line (those not sampled above) are not included in the region



Bounding the Maximum Violation

1. Compute a potentially unsound constraint using sampling and LP
2. Bounding the maximum violation
3. Compute a sound linear constraint

- Step 2: compute an upper bound on the violation of Eq. 3

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

- is equal to the maximum of the function $f(\kappa) = b_l' + w_l'^T \kappa - I_\kappa(x, y)$ over parameter space h_k
- It can be shown that the function f is Lipschitz continuous which enables application of standard global optimization technique
- Function $f : D \subset R^n \rightarrow R$ is Lipschitz continuous if $\|f(y) - f(x)\| \leq C\|y - x\|$ for all $x, y \in D$
- Meaning the absolute value of the function derivative is bounded by constant C

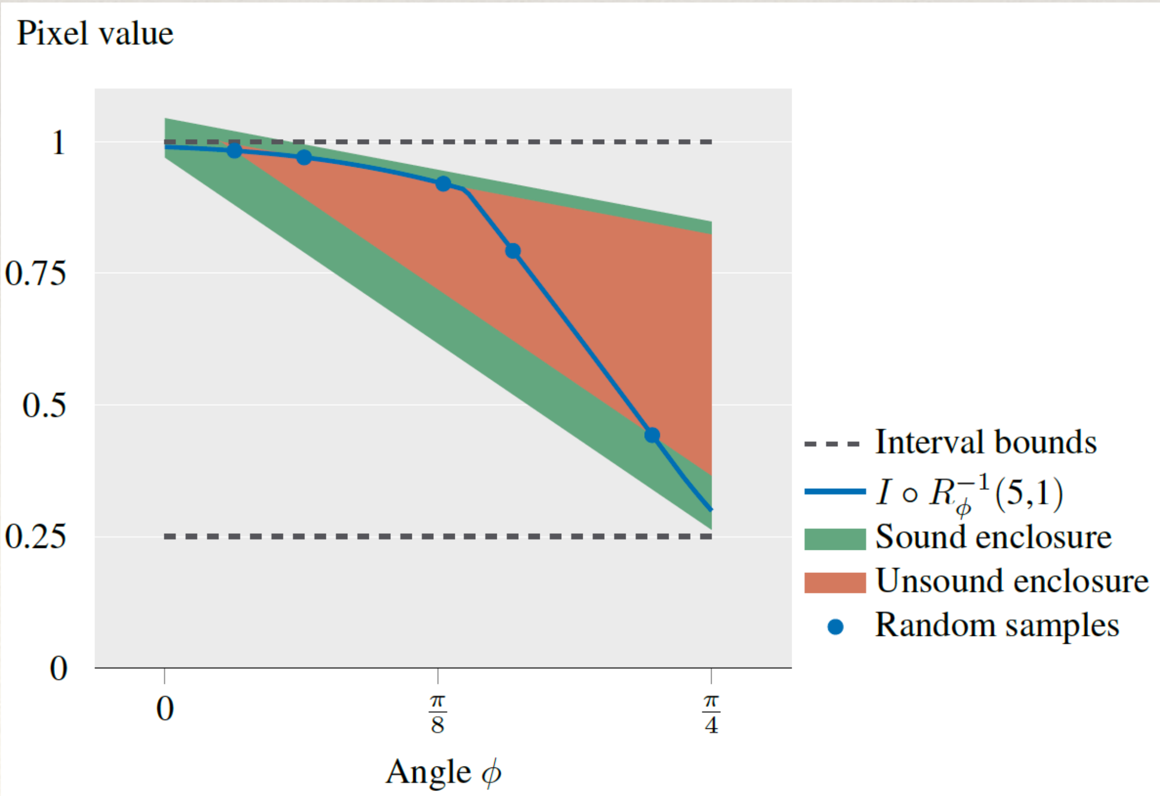
Bounding the Maximum Violation

- Can be solved using Lipschitz optimization algorithm
- Set the tolerance parameter ϵ
- The returned maximum would be v_l. The true maximum f^* would be within $[v_l\ , v_l + \epsilon]$
- It is then guaranteed that $v_l \leq \max_{\kappa \in h_k}(b_l' + w_l'^T \kappa - I_\kappa(x, y)) \leq v_l + \epsilon$
- Eg: Solve the Lipschitz optimization for function $f = 1.07 - 0.9\phi - I \circ R_\phi^{-1}$ with $\epsilon = 0.02$
- Obtain $v_L = 0.08$

Compute Sound Constraint

1. Compute a potentially unsound constraint using sampling and LP
2. Bounding the maximum violation
3. Compute a sound linear constraint

- Step 3: In the previous step we obtained a bound v_l. Using this bound, we update our linear constraints $b_l = b_l' - v_l - \epsilon, w_l = w_l'$ to obtain a sound lower linear constraint
- Now the new constraint satisfies $b_l' - v_l - \epsilon + \boldsymbol{w}_l'^T \boldsymbol{\kappa} \leq \mathcal{I}_{\boldsymbol{\kappa}}(x, y) \quad \forall \boldsymbol{\kappa} \in h_k$
- Eg: $v_l = 0.08$, $\epsilon = 0.02$, the unsound bound $1.07 - 0.9\phi$ is refined to $0.97 - 0.9\phi$
- Together with the similarly obtained upper bound, we form the green sound enclosure



Asymptotically Optimal Constraints

- While our constraints may not be optimal, one can show they are asymptotically optimal as we increase the number of samples
- In our experiments, we also show empirically that close-to-optimal bounds can be obtained with a relatively small number of samples

Experiment

		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**

- DeepG can certify robustness to significantly more complex transformations
- Comparison of DeepG with the baseline based on interval bound propagation
- MNIST and Fashion-MNIST: a 3-layer convolutional neural network with 9 618 neurons
- CIFAR-10: a 4- layer convolutional network with 45 216 neurons

Convergence Towards Optimal 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

- We consider rotation between $[-2,2]$ degrees, composed with scaling between $[-5\%, 5\%]$.
- Varying the number of samples used for the LP solver (n) and tolerance in Lipschitz optimization (ϵ)
- Higher number of samples and smaller tolerance are necessary to obtain more precise bounds
- Even with only 100 samples and $\epsilon = 0.01$ DeepG can certify almost every image in 1.2 seconds.

Comparison of Different Training Methods

		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

- Robust to the translation up to 2 pixels in both x and y direction
- Data augmentation: train the network with translated images
- Adversarial training: with projected gradient descent (PGD)
- Provable defense: DiffAI