

# Robustness Certification with Generative Models

Matthew Mirman  
Department of Computer Science  
ETH Zurich  
Zürich, Switzerland  
matthew.mirman@inf.ethz.ch

Alexander Hägele  
Department of Computer Science  
ETH Zurich  
Zürich, Switzerland  
ahaegle@ethz.ch

Pavol Bielik  
Department of Computer Science  
ETH Zurich  
Zürich, Switzerland  
pavol.bielik@inf.ethz.ch

Timon Gehr  
Department of Computer Science  
ETH Zurich  
Zürich, Switzerland  
timon.gehr@inf.ethz.ch

Martin Vechev  
Department of Computer Science  
ETH Zurich  
Zürich, Switzerland  
martin.vechev@inf.ethz.ch

## Abstract

Generative neural networks are powerful models capable of learning a wide range of rich semantic image transformations such as altering person's age or head orientation. In this work, we advance the state-of-the-art in verification by bridging the gap between (i) the well studied but limited norm-based and geometric transformations, and (ii) the rich set of semantic transformations used in practice. This problem is especially hard since the generated images lie on a highly non-convex manifold, preventing the use of existing verifiers, which often rely on convex relaxations. We present a new verifier, called GENPROVE, capable of certifying the rich set of semantic transformations of generative models. GENPROVE provides both sound deterministic and probabilistic guarantees, by capturing non-convex sets of distributions over activation states, while scaling to realistic networks.

**CCS Concepts:** • Security and privacy;

**Keywords:** Verification, Deep Learning, Adversarial Attacks

## ACM Reference Format:

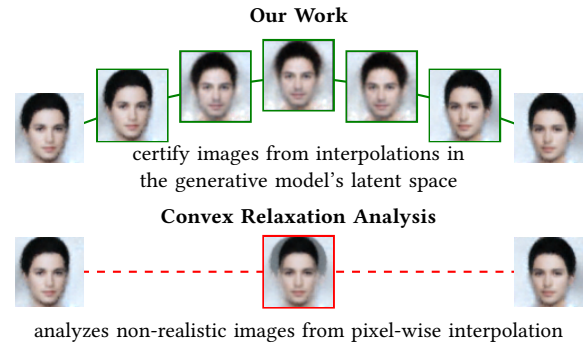
Matthew Mirman, Alexander Hägele, Pavol Bielik, Timon Gehr, and Martin Vechev. 2021. Robustness Certification with Generative Models. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI '21)*, June 20–25, 2021, Virtual Event, Canada. ACM, New York, NY, USA, 18 pages. <https://doi.org/10.1145/3453483.3454100>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org). PLDI '21, June 20–25, 2021, Virtual Event, Canada

© 2021 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-1-4503-8391-2/21/06...\$15.00

<https://doi.org/10.1145/3453483.3454100>

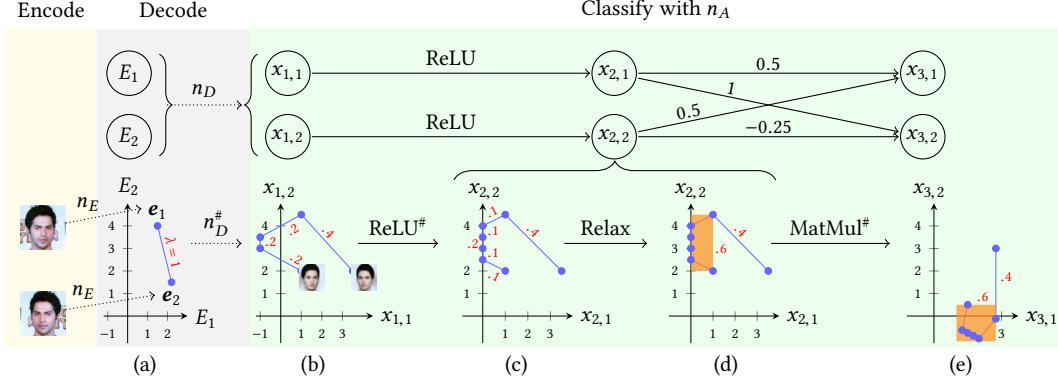


**Figure 1.** Example of generated latent space images (□) our work certifies compared to naive pixel-wise interpolation (□).

## 1 Introduction

While there has been much progress on certifying deep neural networks for norm-constrained pixel perturbations [10, 11, 14, 17, 19, 36, 37, 39, 44, 45, 48] and geometric transformations [1, 9, 28, 40], these works only capture a restricted subset of natural changes that can occur in practice. At the same time, to train state-of-the-art deep models, a wide range of rich semantic transformations are often being used to improve accuracy via data augmentation. As such transformations are hard to specify manually, they are often learned directly from data via generative networks [12, 13, 29, 35, 46, 49]. As a concrete example, a generative network can be trained so that interpolating between the encodings of the flipped head produces images of intermediate head orientations, as in Figure 1. While generative networks provide a compelling framework to express such semantic transformations, they have so far eluded certification due to the scale of non-convexity that they introduce.

**This work.** The goal of this work is to advance the state-of-the-art in verification by bridging the gap between the norm-based and geometric transformations supported by existing verifiers and the rich set of semantic transformations used in practice. In particular, our verifier can certify a number of rich semantic transformations such as: (i) robustness to addition or removal of image features (e.g., changing shoe color or adding mustache), (ii) baldness is robust to *all* head



**Figure 2.** Using GENPROVE to find probability bounds for latent space interpolation of flipped images. Blue chains represent activation distributions at each layer exactly. The orange boxes represent the relaxation that GENPROVE creates, obviating the need to keep track of the segments it covers. Each segment or box’s associated probabilities are shown in red. The inference shown here is faithful to the weights of the toy network in the top row. We provide pseudocode for GENPROVE in Appendix A.

orientations, and (iii) robustness to higher dimensional specifications that use norm-based perturbations but are applied over the latent space of generative models.

The key technical challenge we address is efficiently handling the non-convexity inherent to generative models, while producing accurate bounds *and* scaling to large networks. We address this by introducing tight approximations when necessary to otherwise exact bound computation. Further, we develop a technique for verifying probabilistic properties, allowing us to produce tight deterministic bounds on the probability of a *probabilistic* specification being satisfied. As we will see in our evaluation, this is a critical component for certifying complex generative specifications for which the equivalent deterministic specification holds only rarely.

**Main contributions.** Our key contributions are:

- A relaxation technique that handles non-convex behaviors (Section 3.1) which allows us to scale to large networks with  $\approx 200k$  neurons.
- The first application of probabilistic abstract interpretation [6] to neural networks, which allows us to produce tight deterministic bounds on the probability of a *probabilistic* specification being satisfied (Section 4).
- A verifier, GENPROVE, supporting rich semantic transformations including novel specifications using parametric curves (Section 4.2) and higher dimensional specifications (Section 5.3).
- A thorough evaluation that shows the practical usability and hardness of the problem – we adapted a number of existing verifiers (Zonotope [14], DeepZono [39], ExactLine [42]) but show they either do not scale to complex settings or their bounds are too imprecise.

## 2 Overview of GENPROVE

We start by describing the terminology used throughout our work. Let  $N: \mathbb{R}^m \rightarrow \mathbb{R}^n$  be a neural network which classifies an input  $x \in \mathbb{R}^m$  (in our case an image) to  $\arg \max_i N(x)_i$ .

**Specification.** A robustness specification is a pair  $(\mathbb{X}, \mathbb{Y})$  where  $\mathbb{X} \subseteq \mathbb{R}^m$  is a set of input activations and  $\mathbb{Y} \subseteq \mathbb{R}^n$  is a set of permissible outputs for those inputs.

**Deterministic robustness.** Given a specification  $(\mathbb{X}, \mathbb{Y})$ , a neural network  $N$  is said to be  $(\mathbb{X}, \mathbb{Y})$ -robust if  $\forall x \in \mathbb{X}$ , we have  $N(x) \in \mathbb{Y}$ . In the adversarial robustness literature,  $\mathbb{X}$  is usually an  $l_2$ - or  $l_\infty$ -ball, and  $\mathbb{Y}$  is a set of outputs corresponding to a specific classification. In our case,  $\mathbb{X}$  will be represented as a segment (or a parametric curve) connecting two encodings  $e_1e_2$  produced by a generative model.

**Probabilistic robustness.** A limitation of deterministic robustness properties is that the result is always binary – the property either fully holds or does not. While useful for certifying single images, when combined with complex generative models it becomes too restrictive. Instead, we would like to compute a lower bound on the robustness of complex transformations (e.g., a lower bound for different head orientations is 0.9) for cases when existing deterministic techniques only output that the specification does not hold.

Formally, given a distribution  $\mu$  over  $\mathbb{X}$  (e.g., uniform distribution), we call the bounds on the *robustness probability*  $\Pr_{x \sim \mu}[N(x) \in \mathbb{Y}]$  the *probabilistic [robustness] bounds*.

**GENPROVE overview.** In Figure 2 we illustrate how GENPROVE computes exact and probabilistic bounds for the robustness of a classifier based on a latent space image transformation. In this example, the goal is to verify that a classification network  $n_A$  is robust (i.e., does not change its prediction) when presented with images of a head from different angles, produced by interpolating encodings in the latent space of an autoencoder. To represent this specification, we first use the encoder,  $n_E$ , to produce encodings  $e_1$  and  $e_2$  from the original image and that image flipped horizontally. It is a common technique to use the decoder  $n_D$  to get a picture of a head at an intermediate angle, on an interpolated point  $e$ , taken from the segment  $e_1e_2 = \{e_1 + \alpha \cdot (e_2 - e_1) \mid \alpha \in [0, 1]\}$ .

Decodings for  $\mathbf{e}_1$  and  $\mathbf{e}_2$  can be seen in Figure 2(b). Our goal is to check the property for *all* possible encodings on  $\overline{\mathbf{e}_1\mathbf{e}_2}$  (not only points  $\mathbf{e}_1$  and  $\mathbf{e}_2$ ).

To accomplish this, we propagate lists of line segments and interval (box) constraints through the decoder and classifier, starting with  $\overline{\mathbf{e}_1\mathbf{e}_2}$ . At each layer, we adaptively relax this list by combining segments into interval constraints, in order to reduce the number of points that need to be managed in downstream layers. This relaxation is key, as without it, the number of tracked points could grow exponentially with the number of layers. While Sotoudeh and Thakur [42] demonstrated that this is not a significant concern when propagating through just classifiers, for generative models or decoders, the desired output region will be highly non-convex (with better models producing more segments). One may think of the number of segments produced by the model in such a case as the model’s “generative resolution.”

**Example of inference with overapproximation.** Consider the simple (instructive) two dimensional input classifier network shown in Figure 2, with inputs  $x_{1,1}$  and  $x_{1,2}$ . The possible inputs to this network we would like to consider are the points in the region described by the blue polygonal chain in Figure 2(b), whose axes are  $x_{1,1}$  and  $x_{1,2}$ . The chain has coordinates (1, 2), (−1, 3), (−1, 3.5), (1, 4.5), (3.5, 2) with (1, 2) representing  $n_D(\mathbf{e}_1)$  and (3.5, 2) representing  $n_D(\mathbf{e}_2)$ . The segments of the chain are annotated with weights  $\lambda = 0.2, 0.2, 0.2, 0.4$ . These weights are such that the distribution produced by picking segment  $j$  with probability  $\lambda^{(j)}$  and then picking a point uniformly on that segment is the same as the distribution of  $n_D(\mathbf{e})$  given  $\mathbf{e} \sim U(\overline{\mathbf{e}_1\mathbf{e}_2})$  where  $U(S)$  is the uniform distribution on  $S$ .

After applying the ReLU layer to this chain (marked with ReLU<sup>#</sup>), one can observe in Figure 2(c) that the first and third segment of this chain are split in half, resulting in 6 segments, which is 50% more than there were originally. As the segments represent uniform distributions, the weights of the new segments is the proportional weight of that part on the pre-ReLU segment. Here, each part of the new segment obtains half the pre-ReLU segment’s weight.

Because a 50% increase is significant, we now consolidate, moving from exact to approximate yet sound analysis. Here, we use a heuristic (labeled Relax), to choose segments to subsume that are small and close together. As they are quite close together, we pick the first 5 segments, replacing them by the (orange) box that has the smallest corner at (0, 2) and largest corner at (1, 4.5). This box, introduced in Figure 2(d), is assigned a weight equivalent to the sum 0.6, of the weights of all removed segments. Whereas each segment represents a uniform distribution, the new box represents a specific but unknown distribution with all its mass in the box. As a box is represented by two points (maximum and minimum), only four points are maintained, a significant reduction.

The last step performs matrix multiplication. As this operation is linear, segments can be transformed by transforming their nodes, without adding new points. Box constraints can be transformed using interval arithmetic, also without adding points. The weights of the regions are preserved, as the probability of selecting each region has not changed, only the regions themselves.

**Computing probabilistic bounds.** Let  $\mathbb{A}^{(j)}$  for  $j = 1 \dots k$  represent the regions (either the box and segment, or all 6 segments) shown in Figure 2(e), each with weight  $\lambda^{(j)}$ . Letting  $[H]$  be the indicator for predicate  $H$ , we bound the probability  $P_{t, \overline{\mathbf{e}_1\mathbf{e}_2}} = \Pr_{\mathbf{e} \in \overline{\mathbf{e}_1\mathbf{e}_2}}[\arg \max_i n_D(\mathbf{e})_i = t]$  of class  $t = 1$  being selected by classifier  $n_A$  as follows:

$$\begin{aligned} l &\leq P_{t, \overline{\mathbf{e}_1\mathbf{e}_2}} \leq u && \text{where} \\ l &= \sum_j [\forall x_3 \in \mathbb{A}^{(j)}. x_{3,1} > x_{3,2}] \lambda^{(j)} \\ u &= \sum_j [\exists x_3 \in \mathbb{A}^{(j)}. x_{3,1} > x_{3,2}] \lambda^{(j)} \end{aligned}$$

As an example, we compute the lower bound for the case where we used relaxations. Here, the entirety of the orange box lies within the region where  $x_{3,1} > x_{3,2}$ , so its indicator is 1 and we use its weight. On the other hand, the segment contains a point where  $x_{3,1} = 2.75$  and  $x_{3,2} = 3$  which violates this condition, so its indicator is 0 and its weight is not used. We can thus show a probabilistic lower bound of 0.6. Note that it is possible to provide an exact lower bound by computing the fraction of the segment that satisfies the condition (as described formally in Section 4). We now observe that all of the regions which would have been preserved using the exact procedure (in blue) would have contributed the same amount to the lower bound, as they all entirely satisfy the constraint. Exact inference would produce the same lower bound, but uses 50% more points.

### 3 Certification of Deterministic Properties

We review concepts from prior work [14] and define GEN-PROVE for deterministic properties. Our goal is to automatically show that images  $\mathbf{x}$  from a given set  $\mathbb{X}$  of valid inputs are mapped to safe outputs from a set  $\mathbb{Y}$ . We write this property as  $f[\mathbb{X}] \subseteq \mathbb{Y}$ . For example,  $f$  might be a decoder,  $\mathbb{X}$  a line segment in latent space and  $\mathbb{Y}$  the images for which a given classifier detects a desired attribute.

Such properties compose: If we want to show that  $h[\mathbb{X}] \subseteq \mathbb{Z}$  for  $h(\mathbf{x}) = g(f(\mathbf{x}))$ , it suffices to find a set  $\mathbb{Y}$  for which we can show  $f[\mathbb{X}] \subseteq \mathbb{Y}$  and  $g[\mathbb{Y}] \subseteq \mathbb{Z}$ . For example,  $f$  could be a decoder and  $g$  an attribute detector, where  $\mathbb{Z}$  describes the output activations that lead to an attribute being detected.

We assume that we can decompose our neural network  $f$  as a sequence of  $l$  layers:  $f = L_{l-1} \circ \dots \circ L_0$ . To show a property  $f[\mathbb{X}] \subseteq \mathbb{Y}$ , we will try to find sets  $\mathbb{A}_0, \dots, \mathbb{A}_l$  such that  $\mathbb{X} \subseteq \mathbb{A}_0$ ,  $L_i[\mathbb{A}_i] \subseteq \mathbb{A}_{i+1}$  for  $0 \leq i < l$  and  $\mathbb{A}_l \subseteq \mathbb{Y}$ .

We determine the sets in order: We pick  $\mathbb{A}_0$  based on  $\mathbb{X}$  such that  $\mathbb{X} \subseteq \mathbb{A}_0$  and then for each  $0 \leq i < l$ , we pick  $\mathbb{A}_{i+1}$  such that  $L_i[\mathbb{A}_i] \subseteq \mathbb{A}_{i+1}$ . At the end, we check if we have



$\mathbb{A}_l \subseteq \mathbb{Y}$ . If so, the verification succeeds and the property holds. Otherwise, our procedure fails to prove the property.

**Abstract interpretation.** We automate this analysis using abstract interpretation [5]: we choose the sets  $\mathbb{A}_0, \dots, \mathbb{A}_l$  such that they admit a simple symbolic representation in terms of real parameters. An *abstract domain* is a set of such symbolic representations. We write  $\mathcal{A}_n$  to denote an abstract domain where each element represents a member of  $\mathcal{P}(\mathbb{R}^n)$ . In our case, each abstract element  $a \in \mathcal{A}_n$  represents a set of vectors of  $n$  neural network activations. The concretization function  $\gamma_n: \mathcal{A}_n \rightarrow \mathcal{P}(\mathbb{R}^n)$ , which is specific to each abstract domain, maps a symbolic representation  $a \in \mathcal{A}_n$  to its concrete interpretation as a set  $\mathbb{A} \in \mathcal{P}(\mathbb{R}^n)$  of neural network activation vectors. We will sometimes drop subscripts indicating dimensionality when they are irrelevant or clear from context. Our procedure will compute abstract elements  $a_0, \dots, a_l$  such that  $\mathbb{A}_i = \gamma(a_i)$  for all  $0 \leq i \leq l$ .

An abstract transformer  $T_f^\#: \mathcal{A}_m \rightarrow \mathcal{A}_n$  transforms symbolic representations to symbolic representations, overapproximating the function  $f: \mathbb{R}^m \rightarrow \mathbb{R}^n$ , which means it has to satisfy the soundness property  $f[\gamma_m(a)] \subseteq \gamma_n(T_f^\#(a))$  for all  $a \in \mathcal{A}_m$ . We will compute  $a_{i+1} = T_{L_i}^\#(a_i)$ . The soundness property of the abstract transformer ensures that we have  $L_i[\mathbb{A}_i] \subseteq \mathbb{A}_{i+1}$ , as this is equivalent to  $L_i[\gamma(a_i)] \subseteq \gamma(T_{L_i}^\#(a_i))$ .

By composing abstract transformers for all layers  $L_i$  of the neural network  $f$  in this fashion, we obtain an abstract transformer  $T_f^\# = T_{L_{l-1}}^\# \circ \dots \circ T_{L_0}^\#$ . Abstract interpretation provides a sound, typically incomplete method to certify properties: To show that a neural network  $f: \mathbb{R}^m \rightarrow \mathbb{R}^n$  satisfies  $f[\mathbb{X}] \subseteq \mathbb{Y}$ , it suffices to show that  $\gamma_n(T_f^\#(a)) \subseteq \mathbb{Y}$ , for some abstract element  $a \in \mathcal{A}_m$  with  $\mathbb{X} \subseteq \gamma_m(a)$ .

**Box domain.** If we pick  $\mathbb{A}_0$  as a bounding box of  $\mathbb{X}$ , we can compute sets  $\mathbb{A}_i$  for  $1 \leq i \leq l$  by evaluating the layers  $L_i$  using interval arithmetic. The analysis computes a range of possible values for each network activation, i.e., the sets  $\mathbb{A}_i$  are boxes. At the end, we check if  $\mathbb{A}_l$ 's bounds place it inside  $\mathbb{Y}$ .

This interval analysis is an instance of abstract interpretation. An element of the *box domain*  $\mathcal{B}_n$  is a box: a pair of vectors  $(\mathbf{a}, \mathbf{b})$  where  $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$ . The concretization function is  $\gamma_n(\mathbf{a}, \mathbf{b}) = \prod_{l=1}^n [a_l, b_l]$ . Abstract transformers for the box domain propagate bounds using interval arithmetic. While fast, this is imprecise and often fails to prove true properties.

**Union domains.** A list  $a = (a^{(1)}, \dots, a^{(k)})$  of abstract elements (potentially from multiple different abstract domains) can be interpreted as a union with concretization  $\gamma(a) = \bigcup_{j=1}^k \gamma(a^{(j)})$ . Among other possibilities, we can obtain a union domain abstract transformer  $T_f^\#$  by propagating each element of the union independently using an abstract transformer for its abstract domain:

$$T_f^\#(a) = (T_f^\#(a^{(1)}), \dots, T_f^\#(a^{(k)})).$$

Soundness follows directly from soundness of the component abstract transformers.

For example, we can cover the set  $\mathbb{X}$  with boxes and then propagate them through the network independently using interval arithmetic for each box. In the end, we have to show that all resulting boxes are within  $\mathbb{Y}$ .

**Relaxation.** At any point in the analysis, we can choose to replace an abstract element  $a_i$  by an abstract element  $a'_i$  where  $\gamma(a_i) \subseteq \gamma(a'_i)$ . This can increase precision or reduce the number of parameters needed to represent  $a_i$ .

### 3.1 GENPROVE for Deterministic Properties

GENPROVE for deterministic properties is an analysis with an union domain where each component  $a^{(j)}$  represents either a box or a line segment. Let  $n_i$  denote the number of neurons in layer  $i$ . Formally,  $\gamma(a_i) = \bigcup_{j=1}^{k_i} \gamma(a_i^{(j)})$ , where for each  $j$  either  $\gamma(a_i^{(j)}) = \prod_{l=1}^{n_i} [a_l, b_l]$  is a box with given lower bounds  $\mathbf{a}$  and upper bounds  $\mathbf{b}$ , or  $\gamma(a_i^{(j)}) = \overline{\mathbf{x}_1 \mathbf{x}_2}$  is a segment connecting the given points  $\mathbf{x}_1$  and  $\mathbf{x}_2$  in  $\mathbb{R}^{n_i}$ . We represent  $a_i$  as a list of bounds of boxes and a list of pairs of endpoints of segments.

Like Sotoudeh and Thakur [42], we focus on the case where the set  $\mathbb{X}$  of input activations is a segment. The work of Sotoudeh and Thakur [42] discusses how to split a given segment into multiple segments that cover it, such that a given neural network is an affine function on each of the new segments. Essentially, it determines the points where the segment crosses decision boundaries of piecewise-linear activation functions and splits it at those points. In order to compute an abstract element  $a_{i+1}$  such that  $L_i[\gamma(a_i)] \subseteq \gamma(a_{i+1})$ , we first split all segments according to this strategy applied to only the current layer  $L_i$ . Then, we map the endpoints of the resulting segments to the next layer by applying  $L_i$  to all of them. This is valid and captures exactly the image of the segments under  $L_i$ , because due to the splits,  $L_i$ , restricted to any one of the segments, is always an affine function. Further, we propagate the boxes through  $L_i$  by applying interval arithmetic. Note that if we propagate a segment  $\overline{\mathbf{x}_1 \mathbf{x}_2}$  using this strategy alone for all layers, this analysis produces the exact image of  $\mathbb{X}$ , which is equivalent to performing the analysis using Sotoudeh and Thakur [42]'s method.

**Relaxation.** Before applying layer  $L_i$ , we may apply relaxation operators to turn  $a_i$  into  $a'_i$ , such that  $\gamma(a_i) \subseteq \gamma(a'_i)$ . We use two kinds of relaxation operators: *bounding box* operators remove a single segment  $\overline{\mathbf{cd}}$ . The removed segment is replaced by its bounding box  $\prod_{l=1}^{n_i} [\min(c_l, d_l), \max(c_l, d_l)]$ . *Merge* operators replace multiple boxes by their common bounding box. By carefully applying the relaxation operators, we can explore a rich tradeoff between pure instantiation of Sotoudeh and Thakur [42] and pure interval arithmetic. Our analysis generalizes both: if we never apply relaxation operators, the analysis reduces to Sotoudeh and Thakur [42]

and will be exact but potentially slow. If we relax the initial segment into its bounding box, the analysis reduces to interval arithmetic and will be imprecise but fast.

**Relaxation heuristic.** We define the following heuristic, applied before each convolutional layer. The heuristic is parameterized by a relaxation percentage  $p \in [0, 1]$  and a clustering parameter  $k \in \mathbb{N}$ . Each chain of connected segments with  $t > 1000$  nodes is traversed in order, and each segment is turned into its bounding box, until the chain ends, the total number of different segment endpoints visited exceeds  $t/k$  or we find a segment whose length is strictly above the  $p$ -th percentile, computed over all segment lengths in the chain prior to applying the heuristic. All bounding boxes generated in one such step (from adjacent segments) are then merged, the next segment (if any) is skipped, and the traversal is restarted on the remaining segments of the chain.

## 4 Certification of Probabilistic Properties

We now define GENPROVE for the probabilistic case. This setting is particularly useful when it is not possible to prove the property deterministically (or it does not hold). Our goal here is to automatically show that images  $\mathbf{x}$  drawn from a given input distribution  $\mu$  map to desirable outputs  $\mathbb{D}$  with a probability in some interval  $[l, u]$ . We can write this property as  $\Pr_{\mathbf{x} \sim \mu}[d(\mathbf{x}) \in \mathbb{D}] \in [l, u]$ . For example, we can choose  $d$  as a decoder,  $\mu$  as the uniform distribution on a line segment in its latent space,  $\mathbb{D}$  as the set of images for which a given classifier detects a desired attribute and  $[l, u] = [0.95, 1]$ . The property then states that for at least a fraction 0.95 of the interpolated points, the classifier detects the desired attribute.

Unlike with the deterministic setting, such probabilistic properties do not compose naturally. We therefore reformulate them by defining sets  $\mathbb{X}$  and  $\mathbb{Y}$  of *probability distributions* and a distribution transformer  $f$ , in analogy to deterministic properties. Let  $d_*$  be the *pushforward* of  $d$ , formally defined below (intuitively, the pushforward allows a distribution to be mapped through a deterministic function). We let  $\mathbb{X} = \{\mu\}$ ,  $\mathbb{Y} = \{\nu \mid \Pr_{\mathbf{y} \sim \nu}[\mathbf{y} \in \mathbb{D}] \in [l, u]\}$  and  $f = d_*$ . That is,  $f$  maps a distribution of inputs to  $d$  to the corresponding distribution of outputs of  $d$ . Our property again reads  $f[\mathbb{X}] \subseteq \mathbb{Y}$  and can be decomposed into properties  $(L_i)_*[\mathbb{A}_i] \subseteq \mathbb{A}_{i+1}$  talking about each individual layer. We again overapproximate  $\mathbb{X}$  with  $\mathbb{A}_0$  such that  $\mathbb{X} \subseteq \mathbb{A}_0$  and push it through each network layer, computing sets  $\mathbb{A}_1, \dots, \mathbb{A}_L$ . The sets  $\mathbb{A}_i$  now contain *distributions* over activation vectors. We automate this analysis using probabilistic abstract interpretation.

**Probabilistic abstract interpretation.** We denote as  $\mathbb{D}_n$  the set of probability measures over  $\mathbb{R}^n$ . Probabilistic abstract interpretation is a variant of abstract interpretation where instead of deterministic points from  $\mathbb{R}^n$ , we abstract probability measures from  $\mathbb{D}_n$ . That is, a probabilistic abstract domain [6] is a set of symbolic representations of sets of

measures over program states. We again use subscript notation to determine the number of activations: a probabilistic abstract domain  $\mathcal{A}_n$  has elements that each represent an element of  $\mathcal{P}(\mathbb{D}_n)$ . The probabilistic concretization function  $\gamma_n: \mathcal{A}_n \rightarrow \mathcal{P}(\mathbb{D}_n)$  maps each abstract element to the set of measures it represents.

For a measurable function  $d: \mathbb{R}^m \rightarrow \mathbb{R}^n$ , the corresponding pushforward  $d_*: \mathbb{D}_m \rightarrow \mathbb{D}_n$  maps each measure  $\mu \in \mathbb{D}_m$  to a measure  $\nu \in \mathbb{D}_n$ , given by

$$\nu(\mathbb{Y}) = \Pr_{\mathbf{x} \sim \mu}[d(\mathbf{x}) \in \mathbb{Y}] = \mu(d^{-1}(\mathbb{Y})),$$

where  $\mathbb{Y}$  ranges over measurable subsets of  $\mathbb{R}^n$ .

A probabilistic abstract transformer  $T_f^\# : \mathcal{A}_m \rightarrow \mathcal{A}_n$  abstracts the pushforward  $f_*$  in the standard way: it satisfies  $f_*[\gamma_m(a)] \subseteq \gamma_n(T_f^\#(a))$  for all  $a \in \mathcal{A}_m$ , analogous to the deterministic setting.

Probabilistic abstract interpretation gives a sound method to compute bounds on robustness probabilities. Namely, to show that  $\Pr_{\mathbf{x} \sim \mu}[N(\mathbf{x}) \in \mathbb{Y}] \in [l, u]$ , it suffices to show that  $\nu(\mathbb{Y}) \in [l, u]$  for each  $\nu \in \gamma_n(T_N^\#(a))$  for some  $a$  with  $\mu \in \gamma_m(a)$ .

**Lifting.** Note that we can reuse a deterministic abstract domain directly as a probabilistic abstract domain, by ignoring probabilities. More concretely, consider a deterministic abstract domain  $\mathcal{A}_n$  with deterministic concretization function  $\gamma_n: \mathcal{A}_n \rightarrow \mathcal{P}(\mathbb{R}^n)$ . We can interpret  $\mathcal{A}_n$  as a probabilistic abstract domain by simply defining a probabilistic concretization function  $\gamma'_n: \mathcal{A}_n \rightarrow \mathcal{P}(\mathbb{D}_n)$ . Namely, for some abstract element  $a \in \mathcal{A}_n$  representing a set  $\mathbb{A} = \gamma_n(a)$  of (deterministic) activation vectors, we let  $\gamma'_n(a)$  be the set of all probability measures whose support is a subset of  $\mathbb{A}$ .<sup>1</sup>

The analysis then just propagates abstract elements with the same abstract transformers it would use in the deterministic setting. For example, if we run probabilistic analysis with the box domain,  $\gamma(a_i)$  is the set of all probability measures on the box  $a_i$ , and the analysis propagates the box constraints using interval arithmetic. Of course, such an analysis is rather limited, as it can at most prove properties with  $l = 0$  or  $u = 1$ . For example, it would be impossible to prove that a probability is between 0.6 and 0.8 using only this kind of lifted analysis. However, interval arithmetic, lifted in this fashion, is a powerful component of GENPROVE for probabilistic properties, detailed below.

**Convex combination domains.** A formal convex combination  $a = \sum_{j=1}^k \lambda^{(j)} \cdot a^{(j)}$  of abstract elements (potentially from multiple different abstract domains) can be interpreted as an abstract element whose concretization  $\gamma(a)$  contains all probability measures of the form  $\sum_{j=1}^{k_i} \lambda_i^{(j)} \cdot \mu^{(j)}$ , where each  $\mu^{(j)}$  is some probability measure chosen from the corresponding  $\gamma(a^{(j)})$ . For example, if the abstract elements  $a^{(j)}$  represent disjoint boxes, then  $a$  represents all probability

<sup>1</sup>This is subject to some technical constraints: For example, all deterministic concretizations have to be measurable sets.

measures for which the probability of each box is the corresponding weight  $\lambda^{(j)}$ . In general, we can think of  $\gamma(a)$  as the set of distributions generated by a set of random processes: Each process first randomly selects an index  $j$  according to the probabilities  $\lambda^{(j)}$  and then samples from some fixed probability measure  $\mu^{(j)} \in \gamma(a^{(j)})$ . For each  $1 \leq j \leq k$ , this measure is fixed in advance for each of the random processes.

Similar to unions, we can apply abstract transformers to each abstract element  $a^{(j)}$  independently and to form the convex combination of the results using the same weights:

$$T_f^\#(a) = \sum_{j=1}^k \lambda^{(j)} \cdot T_f^\#(a^{(j)}).$$

This is sound because pushforwards are linear functions.

#### 4.1 GENPROVE for Probabilistic Properties

Probabilistic GENPROVE is an analysis with a convex combination domain where each component  $a_i^{(j)}$  represents either a lifted box or a single probability measure on a segment. Formally, this means

$$\gamma(a_i) = \left\{ \sum_{j=1}^{k_i} \lambda_i^{(j)} \cdot \mu^{(j)} \mid \mu^{(1)} \in \gamma(a_i^{(1)}), \dots, \mu^{(k_i)} \in \gamma(a_i^{(k_i)}) \right\},$$

where for each  $j$ , the concretization  $\gamma(a_i^{(j)})$  is either the set of probability measures supported at most on a box  $\prod_{l=1}^{n_i} [a_l, b_l]$  with lower bounds  $\mathbf{a}$  and upper bounds  $\mathbf{b}$ , or  $\gamma(a_i^{(j)}) = \{\nu\}$ , where  $\nu$  is a distribution on a segment  $\overline{\mathbf{x}_1 \mathbf{x}_2}$  with endpoints  $\mathbf{x}_1$  and  $\mathbf{x}_2$  in  $\mathbb{R}^{n_i}$ . To automate analysis, we represent  $\gamma(a_i)$  as a list of bounds of boxes with associated weights  $\lambda_i^{(j)}$ , and a list of segments with associated distributions and weights  $\lambda_i^{(j)}$ . If, as in our evaluation, we consider a restricted case, where distributions on segments are uniform, it suffices to associate a weight to each segment. The weights should be non-negative and sum up to 1.

The element  $a_i$  can be propagated through layer  $L_i$  to obtain  $a_{i+1}$  in a similar fashion as in deterministic analysis. However, when splitting a segment, we now also need to split the distribution associated to it. For example, if we want to split the segment  $\mathbb{L} = \overline{\mathbf{c}\mathbf{d}}$  with distribution  $\nu$  and weight  $\lambda$  into two segments  $\mathbb{L}' = \overline{\mathbf{c}\mathbf{e}}$  and  $\mathbb{L}'' = \overline{\mathbf{e}\mathbf{d}}$  with  $\mathbb{L}' \cup \mathbb{L}'' = \mathbb{L}$ , we have to form distributions  $\nu', \nu''$  and weights  $\lambda', \lambda''$  where  $\lambda' = \lambda \cdot \Pr_{\mathbf{x} \sim \nu}[\mathbf{x} \in \mathbb{L}']$ ,  $\lambda'' = \lambda \cdot \Pr_{\mathbf{x} \sim \nu}[\mathbf{x} \in \mathbb{L}'' \setminus \mathbb{L}']$ ,  $\nu'$  is  $\nu$  conditioned on the event  $\mathbb{L}'$  and  $\nu''$  is  $\nu$  conditioned on the event  $\mathbb{L}''$ . If distributions on segments are uniform, this would result in the weight being split according to the relative lengths of the new segments. To propagate a lifted box, we apply interval arithmetic, preserving the box's weight. In practice, this is the same computation used for the deterministic propagation of a box.

We focus on the case where we want to propagate a singleton set containing the uniform distribution on a segment  $\mathbb{L} = \overline{\mathbf{x}_1 \mathbf{x}_2}$  through the neural network. In this case, each distribution on a propagated segment will remain uniform, and it suffices to store a segment's weight without an explicit

representation for the corresponding distribution, as noted above. As in the deterministic case, if we apply the analysis to the uniform distribution on a segment without relaxation, the analysis will compute an exact representation of the output distribution. I.e.,  $\mathbb{A}_l$  will contain only the distribution of outputs obtained when the neural network is applied to inputs distributed uniformly at random on  $\mathbb{L}$ .

**Relaxation.** As this does not scale, we again apply relaxation operators. Similar to the deterministic setting, we can replace a probabilistic abstract element  $a_i$  by another probabilistic abstract element  $a'_i$  with  $\gamma(a_i) \subseteq \gamma(a'_i)$ .

**Relaxation heuristic.** Here, we use the same heuristic described for the deterministic setting. When replacing a segment by its bounding box, we preserve its weight. When merging multiple boxes, their weights are added to give the weight for the resulting box.

**Computing bounds.** Given the abstract element,  $a_l$ , describing a superset of the possible output distributions of the network, we compute bounds on the robustness probabilities  $\mathbb{P} = \{\Pr_{\mathbf{y} \sim \nu}[\mathbf{y} \in \mathbb{D}] \mid \nu \in \gamma(a_l)\}$ . The part of the distribution tracked using segments has all its probability mass in determined locations, while the probability mass in a box can be located anywhere within it. We compute bounds as:

$$(l, u) = (\min \mathbb{P}, \max \mathbb{P}) = \left( e + \sum_{j \in \mathbb{L}} \lambda_l^{(j)}, e + \sum_{j \in \mathbb{U}} \lambda_l^{(j)} \right),$$

where  $e$  is the probability of the output being on a segment. If  $\mathbb{D}$  is given as a set of linear constraints, we compute  $e$  by splitting the segments to not cross the constraints and summing up all weights of resulting segments contained in  $\mathbb{D}$ .  $\mathbb{L}$  is the set of indices of lifted boxes contained in  $\mathbb{D}$  and  $\mathbb{U}$  is the set of indices of lifted boxes that intersect with  $\mathbb{D}$ .

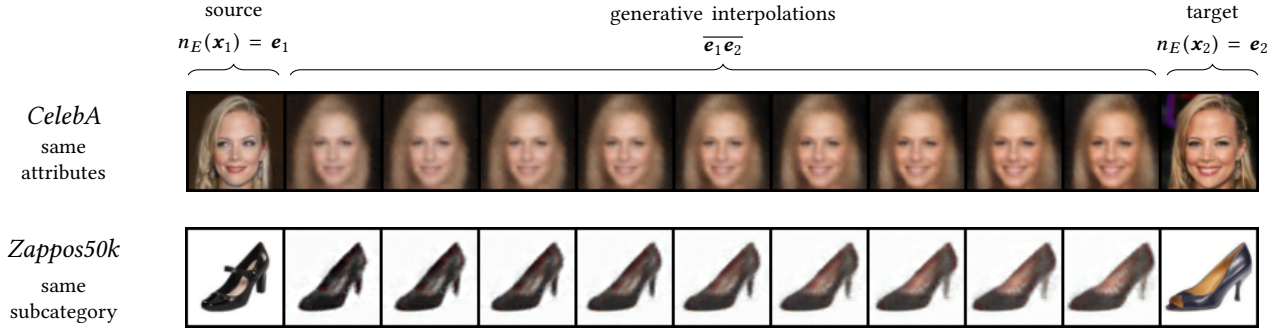
#### 4.2 Generalization to Parametric Curves

The approaches presented so far relied on a number of operations on line segments: We needed to form their image under affine transformations, we had to determine splitting points based on ReLU decision boundaries, and we had to be able to split a line segment into multiple segments whose union is the original segment. For the probabilistic case, we additionally tracked probability measures on those segments.

Therefore, we develop GENPROVECURVE which generalizes our analysis to handle other one-dimensional shapes for which these operations can be supported. Let  $\gamma: [l, u] \rightarrow \mathbb{R}^n$  be a continuous function given by

$$\gamma(t) = \mathbf{a}^{(0)} + \sum_{i=1}^k \mathbf{a}^{(i)} \cdot \eta^{(i)}(t),$$

for one-dimensional continuous functions  $\eta^{(i)}: [l, u] \rightarrow \mathbb{R}$  and vectors  $\mathbf{a}^{(i)} \in \mathbb{R}^n$ . The function  $\gamma$  represents the curve  $\gamma[[l, u]]$  in  $\mathbb{R}^n$ . For the probabilistic case, we additionally consider a probability measure  $\mu$  on  $[l, u]$  describing the



**Figure 3.** Example of a generative specification used in our work. Here,  $\mathbf{x}_1$  and  $\mathbf{x}_2$  are original images with corresponding embeddings  $\mathbf{e}_1$  and  $\mathbf{e}_2$ , respectively. For this specification, the images are chosen such that they contain the same attributes (for CelebA) or belong to the same subcategory (for Zappos50k). The goal is to verify, deterministically or probabilistically, that the network under test does not change its prediction when presented with the interpolated images  $\bar{\mathbf{e}_1 \mathbf{e}_2}$ .

distribution of the curve parameter. (The probability measure in  $\mathbb{R}^n$  describing our probabilistic curve is then implicitly given by  $\nu(X) = \mu(\gamma^{-1}(X))$ .)

We can symbolically form the image of the shape  $\gamma[[l, u]]$  under an affine transformation  $f(\mathbf{x}) = A \cdot \mathbf{x} + \mathbf{b}$  as the set  $f[\gamma[[l, u]]] = (f \circ \gamma)[[l, u]]$ , where  $f \circ \gamma$  is given by

$$f(\gamma(t)) = A \cdot \gamma(t) + \mathbf{b} = (A \cdot \mathbf{a}^{(0)} + \mathbf{b}) + \sum_{i=1}^k (A \cdot \mathbf{a}^{(i)} \cdot \eta^{(i)}(t)).$$

I.e., to apply an affine transformation to our curve, it suffices to transform the coefficient vectors  $\mathbf{a}^{(i)}$ .

To find splitting points for ReLU decision boundaries, we need to solve the equation  $\gamma(t)_j = 0$  for  $t$  for each component  $j \in \{1, \dots, n\}$ . For example, if we consider quadratic parametric curves  $\gamma: [l, u] \rightarrow \mathbb{R}^n$  of the form

$$\gamma(t) = \mathbf{a}^{(0)} + \mathbf{a}^{(1)} \cdot t + \mathbf{a}^{(2)} \cdot t^2,$$

we have to solve a quadratic equation for each component, yielding at most two splitting points per component, where we ignore solutions outside  $[l, u]$ . We can split the curve at those points by restricting it to segments between subsequent splitting points in sorted order. In the probabilistic case, we further restrict the measure to the same segments and associate the resulting measures to the new curves.

## 5 Evaluation

In this section, we demonstrate the benefits of GENPROVE and the techniques presented in our work. In particular, our goal is to answer the following three research questions:

- RQ1** Is probabilistic abstract interpretation necessary for handling complex generative specifications (as compared to traditional abstract interpretation)?
- RQ2** Does GENPROVE produce tight bounds and scale to realistic large networks (unlike existing methods)?
- RQ3** What novel specifications can be verified using GENPROVE (beyond what is currently possible)?

We first answer RQ1 by showing that our application of probabilistic abstract interpretation is key for analysing generative specifications. Concretely, we demonstrate that: (i) using deterministic verification is a limiting factor for *all* non-trivial benchmarks and networks, (ii) probabilistic interpretation proposed in our work improves the fraction of samples for which tight bounds can be computed from 0.5% to up to 76.2%, and (iii) our combination of probabilistic analysis with relaxations can produce non-trivial verified bounds for 100% of samples.

We then answer RQ2 by demonstrating that GENPROVE scales to realistically large networks with 200k neurons while producing bounds that are very tight and close to zero (e.g.,  $5.7 \cdot 10^{-5}$ ). In contrast, we show that all prior methods fail – either because they are imprecise and produce extremely loose bounds close to 1, or they exhaust the ample GPU memory and crash. Note that, as we will show later in this section, only increasing the GPU memory is not a scalable solution and a fundamentally different approach, like the one proposed in our work, is needed.

To answer RQ3, we show the versatility of GENPROVE in certifying the novel class of generative specifications in five ways: (i) we show how GENPROVE can be used to certify and specify the higher dimensional specification where a generative network defines a continuous set of images that a classifier should categorize correctly under any possible  $L_\infty$  attack (ii) certifying robustness to different head orientations, (iii) certifying attribute independence via adding previously absent attributes (e.g., changing the hair color as illustrated in Figure 3), (iv) certifying attribute independence over input regions that curve through areas with previously absent attributes. and finally (v) certifying out of distribution detection with non-uniform specifications.

**Experimental setup.** We certify robustness of generative models using a variety of different approaches:



**Table 1.** Comparing deterministic analysis with probabilistic analysis. The number is the percentage of 100 samples evaluated on average consistency  $\hat{C}$  that did not return the full interval  $l = 0$  and  $u = 1$ . We note that the poor performance of BASELINE on ConvMed is due to out of memory errors, where the full interval is returned.

DATASET	NETWORK	% of samples w/ non-trivial verified bounds					
		Exact Verification			Verification with Relaxations		
		(Deterministic) BASELINE	(Probabilistic) GENPROVE <sup>0</sup>		(Deterministic) GENPROVE <sup>DET0.02</sup> <sub>100</sub>	(Probabilistic) GENPROVE <sup>0.02</sup> <sub>100</sub>	
CelebA	ConvSmall	91.2%	+8.8% →	100%	78.6%	+21.4% →	100%
	ConvMed	9.5%	+0.5% →	10%	23.8%	+76.2% →	100%
Zappos50k	ConvSmall	56%	+44% →	100%	56%	+44% →	100%
	ConvMed	8%	+3% →	11%	58%	+42% →	100%
Prior Work				Our Work			

- GENPROVE<sub>k</sub><sup>p</sup> which implements both the probabilistic and deterministic (denoted as GENPROVE<sup>DET</sup>) verifier proposed in our work. Here,  $p$  is the relaxation percentage and  $k$  is the clustering parameter. Note that setting the relaxation percentage to zero (denoted as GENPROVE<sup>0</sup>) instantiates our approach without relaxations, thus producing exact results.
- BASELINE is the deterministic approach proposed by Sotoudeh and Thakur [42], producing exact results. We note that we use our own, more scalable and GPU-optimized, implementation of this approach. The original implementation supports only computations on CPUs and takes prohibitively large amounts of time when run on the large networks we use for evaluation.
- A wide range of existing convex abstract domains for neural networks: Box [14], Zonotope [14], DeepZono [39], and HybridZono [33]. We adapted all of them to certification of generative models by representing the initial segment  $\overline{e_1 e_2}$ . Note that for every domain but Box, this step is exact and does not lose precision.

We implement GENPROVE in the DiffAI [33] framework, taking advantage of the GPU parallelization provided by PyTorch [34]. Our implementation will be made available on GitHub along with all the models used for testing.

For a fair comparison of the runtime and scalability, all the verifiers used are also implemented with GPU support. All of our experiments are performed on a machine with a Titan RTX GPU with 24 GB of GPU memory.

**Generative models.** We use 3 datasets of increasing complexity – MNIST [24], Zappos50k [50, 51], and CelebA [31]. For each dataset, we trained a VAE [22] autoencoder with the architectures and training schemes described in full detail in Appendix B. For all datasets, the decoder and generator have each 74 128 neurons, unless otherwise specified.

**Target networks.** For each dataset, we trained a variety of attribute detectors or classifiers:

- *CelebA*: We trained attribute detectors on the scaled  $64 \times 64$  images with three different architectures – ConvSmall, ConvMed, and ConvLarge, with 24 676, 63 804 and 123 180 neurons respectively. The attribute detectors are trained to recognize the 40 attributes provided by CelebA (e.g., bald, bangs) [31]. Here, an attribute  $i$  is detected in the image if the  $i$ -th component of the network output is strictly positive.
- *Zappos50k*: We trained classifiers on the  $64 \times 64$  images with the same three architectures as for CelebA. The classifiers are trained to recognize the 21 subcategories (e.g. heels, boots) from the Zappos50k dataset [50, 51].
- *MNIST*: We used a classifier with 175 816 neurons trained with three different techniques to recognize digits [24]. Specifically, we used the publicly available ConvBiggest architecture from [33] and trained it using standard training, using Box with DiffAI, as well as FGSM [16] with  $\epsilon = 0.1$ .

**Evaluation metrics.** We show the precision of our system by verifying that a given model (denoted as  $n_A$  for an attribute detector, and  $n_C$  for a classifier) is robust to the transformations learned by the generative model.

We formalize this concept with a metric called *consistency*: for a point picked uniformly between the encodings  $e_1$  and  $e_2$  of ground truth inputs, we determine the probability that its decoding (computed by a decoder  $n_D$ ) will have (or not) the same attribute. As a concrete example, Figure 3 shows generative interpolations for both CelebA and Zappos50k. Formally, consistency is defined for attribute detectors as

$$C_{i, n_A, n_D}(e_1, e_2) = \Pr_{e \sim U(\overline{e_1 e_2})} [\text{sign } n_A(n_D(e))_i = t],$$



**Table 2.** Scalability and precision of our method compared to wide range of existing convex abstract domains. For a fair comparison, all methods are lifted probabilistically. We report average consistency  $\hat{C}$  bound widths (lower is better).

		average consistency $\hat{C}$ bound width ( $u - l$ )					
		( $\approx 25k$ neurons)	( $\approx 64k$ neurons)	( $\approx 123k$ neurons)			
DATASET	DOMAIN	ConvSmall	ConvMed	ConvLarge	Precise	Scalable	
CelebA	Prior Work	Box [14]	0.98	0.98	0.98	-	✓
		HybridZono [33]	0.97	0.97	0.97	-	✓
		DeepZono [39]	1.0	1.0	1.0	-	-
		Zonotope [14]	1.0	1.0	1.0	-	-
	Our Work	GENPROVE <sup>0</sup>	<b>0.0</b>	0.9	0.95	✓	-
		GENPROVE <sup>0.02</sup> <sub>100</sub>	<b><math>1.8 \cdot 10^{-5}</math></b>	<b><math>1.1 \cdot 10^{-4}</math></b>	<b><math>1.6 \cdot 10^{-4}</math></b>	✓	✓
Zappos50k	Prior Work	Box [14]	1.0	1.0	1.0	-	✓
		HybridZono [33]	1.0	1.0	1.0	-	✓
		DeepZono [39]	1.0	1.0	1.0	-	-
		Zonotope [14]	1.0	1.0	1.0	-	-
	Our Work	GENPROVE <sup>0</sup>	<b>0.0</b>	0.89	0.99	✓	-
		GENPROVE <sup>0.02</sup> <sub>100</sub>	<b><math>3.3 \cdot 10^{-5}</math></b>	<b><math>4.5 \cdot 10^{-5}</math></b>	<b><math>5.7 \cdot 10^{-5}</math></b>	✓	✓

and for classifiers as

$$C_{n_C, n_D}(\mathbf{e}_1, \mathbf{e}_2) = \Pr_{\mathbf{e} \sim U(\mathbf{e}_1, \mathbf{e}_2)} [\arg \max_i n_C(n_D(\mathbf{e}))_i = t].$$

Suppose  $P$  is a set of pairs  $\{\mathbf{a}, \mathbf{b}\}$  from the data and it holds that  $\text{sign } \mathbf{a}_{A,i} = \text{sign } \mathbf{b}_{A,i}$  for every attribute, where  $\mathbf{a}_{A,i}$  is the label of attribute  $i$  for  $\mathbf{a}$ . We compute bounds on the *average consistency* as  $\hat{C}_P = \text{mean}_{\mathbf{a}, \mathbf{b} \in P, i} C_{i, n_A, n_D}(n_E(\mathbf{a}), n_E(\mathbf{b}))$  for attribute detectors, and  $\hat{C}_P = \text{mean}_{\mathbf{a}, \mathbf{b} \in P} C_{n_C, n_D}(n_E(\mathbf{a}), n_E(\mathbf{b}))$  for classifiers, where  $n_E$  is the encoding network.

We compute a probabilistic bound,  $[l, u]$ , for each method in our evaluation such that  $l \leq \hat{C} \leq u$ . We call  $u - l$  its width.

### 5.1 RQ1 - Probabilistic Abstract Interpretation

We start by addressing RQ1 and demonstrate that while traditional deterministic verification methods may be precise for deterministic specifications, they are of limited utility when tasked with verifying probabilistic specifications. To understand why, recall that for the deterministic domains, there are only three possible outputs for the lower and upper bounds:  $[0, 0]$  meaning that none of the specification was correct,  $[1, 1]$  meaning that the specification was entirely correct, or least usefully, the full interval  $[0, 1]$  implying that the technique was unable to verify one way or the other how much of the specification was correct. This severely limits the usefulness of the verification, especially for cases with imperfect networks and imperfect specifications.

Table 1 demonstrates the limited applicability of deterministic domains. Here, we report the fraction of specifications where the bounds were strictly tighter than  $[0, 1]$ . Based on the results, we can immediately see that GENPROVE provides

useful bounds for 100% of the specifications for every network and dataset. At the same time, the deterministic methods BASELINE and GENPROVE<sup>DET</sup> rarely return useful bounds for the consistency specification. In particular, BASELINE proves at best 91.2%, and at worst 8% of the specifications. This is because even on the large network, GENPROVE<sup>0</sup> and BASELINE run out of memory. One should also observe that GENPROVE<sup>0</sup> performs better than BASELINE and when it does not run out of memory, performs better than GENPROVE<sup>DET</sup>. We note that GENPROVE not only always provides useful results, but is also precise and achieved an average width ( $u - l$ ) of at worst 0.0001 for CelebA (ConvMed). In comparison, the next-best domain (BASELINE on ConvSmall) produced a width of at best 0.1748 (not shown in Table 1), which is a full four orders of magnitude worse on a smaller network.

### 5.2 RQ2 - Precision and Scalability

Next, we address RQ2 by comparing the precision and scalability of probabilistic GENPROVE to a variety of existing convex abstract domains, as well as sampling. To study the scalability of all domains, we certify the robustness of three networks of increasing complexity – ConvSmall with  $\approx 25k$  neurons, ConvMed with  $\approx 64k$  neurons and ConvLarge with  $\approx 123k$  neurons. Further, to provide variety, we certify robustness using two datasets: CelebA and Zappos50k.

**Precision.** Table 2 shows the result of running the verifiers using the same  $|P| = 100$  pairs of images with either matching attributes (for CelebA) or the same class (for Zappos50k). We report the bound  $[0, 1]$  if memory is exhausted.

**Table 3.** Comparison of the memory usage and runtime of our GENPROVE with and without relaxations.

DATASET	DOMAIN	peak GPU memory in GB / OOM (%)			runtime in seconds		
		ConvSmall	ConvMed	ConvLarge	ConvSmall	ConvMed	ConvLarge
<i>CelebA</i>	GENPROVE <sup>0</sup>	7 GB / 0%	22.7 GB / 90%	23.1 GB / 95%	11 sec	OOM	OOM
	GENPROVE <sup>0.02</sup> <sub>100</sub>	3.5 GB / 0%	6.8 GB / 0%	9.4 GB / 0%	13 sec	25 sec	41 sec
<i>Zappos50k</i>	GENPROVE <sup>0</sup>	6.5 GB / 0%	22.7 GB / 89%	23.6 GB / 99%	11 sec	OOM	OOM
	GENPROVE <sup>0.02</sup> <sub>100</sub>	6.4 GB / 0%	6.6 GB / 0%	7.1 GB / 0%	15 sec	25 sec	32 sec

**Table 4.** Comparison of the precision of our method to sampling. Our method not only provides results that are guaranteed to be sound, but also leads to tighter bounds. Note that the runtime of both methods (not shown) is the same.

	DOMAIN	bound width ( $u - l$ )	
		<i>CelebA</i>	<i>Zappos50k</i>
<i>Verified Correctness</i>	GENPROVE <sup>0.02</sup> <sub>100</sub>	$1.6 \cdot 10^{-4}$	$5.7 \cdot 10^{-5}$
<i>99.999% Confidence</i>	Sampling	$2.1 \cdot 10^{-4}$	$1.5 \cdot 10^{-3}$

One can first observe that Box, HybridZono, Zonotope, and DeepZono are unable to certify any samples as they almost always produce a probabilistic interval with width 1. We posit that this is due to non-convexity being highly important for these kinds of specifications. Zonotope and DeepZono run out of memory for all the samples, even for the smallest network ConvSmall.

While GENPROVE<sup>0</sup> is theoretically complete, it also predominantly failed to provide useful bounds as it frequently ran out of GPU memory. However, we can see that for the small network ConvSmall, where it does scale, it does produce exact results: the width of all bounds is 0.

GENPROVE<sup>0.02</sup><sub>100</sub> is the only approach that is both scalable and precise. The bounds are tight even for the largest network ConvLarge:  $5.7 \cdot 10^{-5}$  for Zappos50k for example. Significantly, the bounds remain tight as the network size increases. For example, when the network size increased by 500% (from ConvMed to ConvLarge), the bound width increased from  $3.3 \cdot 10^{-5}$  only to  $5.7 \cdot 10^{-5}$ .

While the speed of each method can be seen in Table 8 in the appendix, these numbers can be misleading: despite Box and HybridZono’s apparent speed, they fail to provide any useful information for any specifications due to aforementioned imprecision. Similarly, Zonotope, DeepZono and GENPROVE<sup>0</sup> also appear very fast while failing to provide useful information. These fail however due to running out of GPU memory. In contrast, GENPROVE takes 41.4 seconds on the most complicated specification and network here, but produces extremely tight and useful bounds in every case.

**Scalability.** Table 3 shows the average runtime, peak GPU memory, and the fraction of samples that resulted in out-of-memory (OOM) errors. For the CelebA dataset, while the network size increased 5×, the memory usage of GENPROVE<sup>0.02</sup><sub>100</sub> increased only 2.7×. The improvement in memory usage is even better for the Zappos50k dataset, where increasing the size 2× leads to an increase in memory usage of only 1.08×. This shows that the relaxation technique proposed in our work successfully reduces the memory usage while still achieving tight bounds. The results for runtime are similar: runtime increases sublinearly depending on network size. Overall, the verification is fast and takes on average  $\approx 40$  seconds for ConvLarge and  $\approx 11$  seconds for ConvSmall.


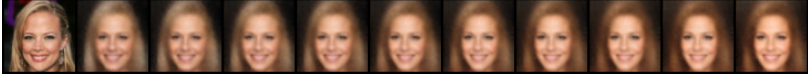

The results in Table 3 also detail why GENPROVE<sup>0</sup> does not scale: the needed GPU memory increases significantly with network size. For complex specifications, the number of segments that must be tracked often increases exponentially.

To improve the scalability of both GENPROVE<sup>0</sup> and GENPROVE<sup>0.02</sup><sub>100</sub> further, it is possible to split the specification into smaller parts. In our case, this corresponds to partitioning the initial segment (or other one-dimensional shapes) into multiple smaller segments that are verified sequentially and then merged together. However, while useful for avoiding the memory limitations, it comes at the cost of increased runtime. Given that the number of segments can grow exponentially with the network size, we believe that developing and incorporating techniques like the relaxation proposed in our work is critical for scaling to state-of-the-art networks.

**Comparison to sampling.** In our next experiment, shown in Table 4, we compare to a sampling method, where samples are drawn from the uniform distribution over the initial segment. We report the Clopper-Pearson interval with a confidence of 99.999%. Notably, the probabilistic bound returned by sampling is only guaranteed to be correct 99.999% of the time (in cases it reaches the desired confidence), whereas for other analyses it is guaranteed to always be correct.

The results show that the sampling does scale and also produces bound widths with a reasonable precision. However, not only does GENPROVE produce bounds that are guaranteed to be correct, it also produces bounds that are up to two orders of magnitude tighter (i.e., by up to  $\approx 2500\%$ ). These results were

**Table 5.** An illustration of the various specifications we support and what we are able to certify about them. This is in addition to the specifications shown in Figure 3 and certification of adversarial regions around a generative specification (not shown).

SPECIFICATION TYPE		ILLUSTRATION	CERTIFIED RESULT
(a)	head orientation		bound width ( $u - l$ ) $1.0 \cdot 10^{-4}$
(b)	adding "brown hair"		robust attributes 32/40
(c)	quadratic curve		robust attributes 29/40

consistent across all the networks and datasets we evaluated, the full version of which is included in Appendix D.

### 5.3 RQ3 - Verifying Novel Generative Specifications

So far, we have demonstrated that GENPROVE can verify generative specifications like those shown in Figure 3, which interpolate between two images with identical attributes. We now address RQ3 and demonstrate how our method applies to five additional generative specifications.

**Certifying robustness to head orientation.** As shown by Dumoulin et al. [7], VAEs can generate images of intermediate poses from flipped images. An example of this transformation is shown in Table 5 (a). We evaluated line specifications between encodings of horizontally flipped images. For a head, ideally the intermediate reconstructions will be of the intermediate 3D orientations. As pose is not a provided CelebA attribute, the attribute detector should recognize the same attributes for all interpolations. We evaluated GENPROVE<sub>100</sub><sup>0.02</sup> on images from CelebA dataset and successfully produced tight bounds for all evaluated images. The average bound width was only  $1.0 \cdot 10^{-4}$ , with average lower bound  $l = 0.8433$  and average upper bound  $u = 0.8434$ . That is, we verified that on average, the target network is robust to 84% of the generated interpolations. The results for other method follow the results shown in Table 2, that is, they either do not scale or provide bound width close to 1.

**Certifying attribute independence for CelebA.** We use GENPROVE to demonstrate that attribute detection for one feature is invariant to transformation of an independent feature. Specifically, we verify for a single image the effect of adding a different hair color, as shown in Table 5 (b). To achieve this, we find the attribute vector  $m$  for "BrownHair" using the 80k training images in the manner described by [23], and compute probabilistic bounds for  $C_j(n_E(o), n_E(o) + 3m, o_{A,j})$  for  $j \neq 11$  and the image  $o$ . Here, we used the ConvMed attribute detector. Using GENPROVE we are able to

prove that 32 out of the 40 attributes are entirely robust to brown hair addition, and 8 of them were not robust. Among the attributes which can be proven to be robust was  $i = 39$  for "young" for example. We are able to find that attribute  $i = 9$  for "BlondHair" is not entirely robust to the addition of the BrownHair vector, which is expected. Here, our approach is able to find tight lower and upper bounds on the robustness probability of  $[0.6038, 0.6039]$  for that attribute. The average interval width for all attributes was  $7.87 \cdot 10^{-6}$ . One can observe in Table 5 (b) that this matches visually what the interpolated images show: the first 6 or so reconstructions appear to have blond hair, whereas the rest have brown hair.

**Certifying curved specifications.** We demonstrate the first exact analysis (both deterministic and probabilistic) of a non-convex smooth input for neural networks. Given three encoding vectors,  $e_0, e_1, e_1$ , we create the following quadratic curve that passes through them at  $t = 0, 0.5, 1$  respectively:

$$\gamma(t) = e_0 + (4e_1 - e_2 - 3e_0) \cdot t + 2(e_2 + e_0 - 2e_1) \cdot t^2.$$

We use the encoding of an image of a head for  $e_0$ , the encoding of the flipped head for  $e_2$ , and the midpoint of these two encodings perturbed by a scaled moustache attribute vector,  $m$  (found as described earlier for the BrownHair attribute vector, but for attribute 22) for  $e_1 := 0.5(e_0 + e_2) + 4m$ . We visualize this specification in Table 5 (c).

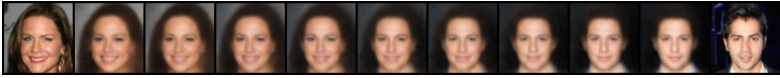

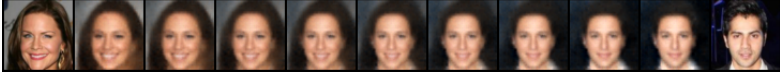
We used GENPROVECURVE to demonstrate attribute independence for 29 out of the 40 different attributes. As GENPROVECURVE is exact, it produced a bound width of 0. The average probability of attribute consistency is 0.85. Here, we used a smaller generator architecture, DecoderSmall with only 41 160 neurons, the usual ConvSmall attribute detector. Even though it is exact, GENPROVECURVE was able to verify the non-linear specification in only 12.6 seconds.

**Certifying adversarial regions around generative output.** Unlike any other pre-existing methods, GENPROVE can

**Table 6.** Average number of fully verified images for interpolations of images in the same MNIST class using adversarial region width  $\epsilon = 0.1$  for ConvBiggest with 175 816 neurons trained in three ways.

TRAINING SCHEME	verification of adversarial generative interpolations			
	standard accuracy	adversarial accuracy (PGD [32])	provable accuracy (Box)	bound width ( $u - l$ )
Standard training	99.2%	54.5%	0.0%	0.9999
Adversarial with FGSM [16]	99.5%	97.1%	0.0%	1.0
Adversarial with DiffAI [33]	99.1%	97.7%	92.5%	0.0990

**Table 7.** Using an interpolation specification with an arcsin distribution between unrelated images to compare the realism of images produced by various generative models. We compute an upper bound on the probability that the out-of-distribution detector (in this case a GAN discriminator) successfully determines that the generated image is fake.

	MODEL	INTERPOLATION	UPPER BOUND	BOUND WIDTH
(a)	VAE		0.4528	0
(b)	FactorVAE		0.32	$3.5 \times 10^{-5}$
(c)	ACAI		0.29	$8.8 \times 10^{-5}$

be easily applied to higher-dimensional specifications. Specifically, we use generative models to construct a base specification, which we additionally want to be adversarially robust. We define the adversarial consistency  $C_{\epsilon, l, n_A, n_D}^{\text{adv}}(\mathbf{e}_1, \mathbf{e}_2)$  as:

$$\Pr_{\mathbf{e} \sim U(\overline{\mathbf{e}_1, \mathbf{e}_2})} [\forall a \in B_{\infty, \epsilon}(n_D(\mathbf{e})). \arg \max_i n_A(a)_i = t].$$

Here,  $B_{\infty, \epsilon}(n_D(\mathbf{e}))$  refers to the  $L_\infty$  adversarial region of size  $\epsilon$  around the output of the generator. To handle this we propagate the interval using GENPROVE through the decoder  $n_D$  to produce a list of segments and boxes. We compute a box around each segment, and then enlarge each box in the entire list, in every dimension, by  $\epsilon$ . We then propagate the boxes through  $n_A$ . Crucially, these operations all fall under the framework developed in our work, and so this specification can be seen as an instance of GENPROVE.

Because no other method is both capable of handling generative specifications without adversarial regions, or easily extensible to handle this specification, we only use GENPROVE to demonstrate the benefit of DiffAI training. Table 6 shows the result of applying GENPROVE to solve this specification on regularly trained networks, FGSM-trained networks [16], and DiffAI-trained networks [33] for the MNIST dataset. We report, in addition to the standard accuracy, the accuracy against the PGD adversary [32] with 5 iterations, and the provability using Box. We finally report the bound

width on the generative adversarial specification using GENPROVE. One can see that on a DiffAI-trained network, we are able to provide tight bounds on the adversarial consistency.

**Certifying complex specifications.** Finally, we demonstrate the full capabilities of GENPROVE for certifying non-uniform specifications involving naive out-of-distribution detection using a GAN discriminator, and autoencoders specifically trained for disentanglement and interpolation as shown in Table 7. Here, we trained two more VAEs on CelebA: (i) ACAI [2] which is designed specifically to produce realistic encoding interpolations, and (ii) FactorVAE [20] which is designed to learn a latent encoding where each dimension represents an independent disentangled feature. For out-of-distribution detection, we used the discriminator from a vanilla GAN [15]. Each has been modified to use MSE as their reconstruction loss to avoid sigmoids. We use Decoder for all decoders, Encoder for the encoders and the ACAI critic, and EncoderSmall for the GAN discriminator. Further, FactorVAE uses a small feedforward network 5 layers deep (each layer has 100 neurons) as its factorization critic. Each network was trained for 100 epochs, with a batch size of 64. The autoencoders used 64 latent dimensions while the GAN used 128. All other hyperparameters are as in the respective papers.



To demonstrate non-uniform distributions, we use the arc-sine distribution over the interpolation specification. Table 7 compares the upper bound of an interpolation specification between two unrelated images. A small number means that the discriminator was fooled by the generator in question. We can see that the most successful generator is ACAI, which is specifically trained to produce realistic interpolations.

## 6 Related work

Next, we review work most closely related to ours.

**Certifying generative models.** Dvijotham et al. [9] verifies lower bounds on a probabilistic property for all inputs in a specification for variational autoencoders with a latent random variable using a dual approach. In contrast, GENPROVE finds tight upper *and* lower bounds on the probability that a property is satisfied given a distribution over a specification. Further, our approach scales to networks that are orders of magnitude larger – we successfully certify CelebA networks with nearly 200k neurons compared to a network with 3 hidden layers of 64 units each used in Dvijotham et al. [9].

**Convex relaxations.** PROVEN [47] proposes a technique to infer confidence intervals on the probability of misclassification from preexisting convex relaxation methods that find linear constraints on outputs. In our evaluation, we show that for interpolations of generative models, convex relaxation methods are unable to prove meaningful bounds. This implies that the linear lower bound function used by PROVEN would be bounded above by 0, and thus because  $F_{g_t}^L(0.5) \geq F_0^L(0.5)$  and  $F_0^L(0.5) = 1$ , the lower bound,  $\gamma_L$ , that their system should derive would be  $\gamma_L = 0$ . This is because even the most precise convex relaxation over the generated images might include many images that are not realistic. For example, the convex hull includes the pixel-wise average of the generated endpoint images, as in Figure 1.

**Adversarial defenses.** Another line of work, smoothing, provides a defense with statistical guarantees [3, 4, 25, 26, 30]. In our evaluation, we compared to a variant of this technique, sampling, and demonstrated that GENPROVE computes two orders of magnitude tighter bounds across all the datasets and models. Further, our work provides provable guarantees compared to sampling whose bounds are correct only with some probability (e.g., with 99% confidence).

At the same time, a number of recent adversarial defences started to incorporate generative models as core component or their approach [27, 38, 41]. For all of those, incorporating techniques presented in our work is a natural next step required to certify that the defence is provably correct.

**Certifying line segments.** Of particular note, the work of Sotoudeh and Thakur [42] also restricts the network inputs to line segments. They used this method to certify non

norm-based properties [18] and to improve Integrated Gradients [43]. In our work we build on the results of Sotoudeh and Thakur [42] and extend them in several major aspects by: (i) computing tight deterministic bounds on the probability of a probabilistic specification, (ii) introducing relaxations that enable scaling to large networks, (iii) ensuring the correctness of the probabilistic guarantees in the presence of these relaxations, and (iv) exploring novel specifications including parametric curves and higher dimensional specifications.

## 7 Discussion

In this work, we developed GENPROVE and demonstrated its use to certify transformations given by generative models. While generative models are intended to represent the underlying data distribution, physical limitations imply they actually generate slightly different distributions. Unfortunately, it is usually not possible to deterministically certify how much a given generative model differs from the ground truth. This is because for most real-world applications, the ground truth is only approximated from data. In such cases, our domain is useful for either verifying that the generative model satisfies some property given by a trusted downstream classifier, or verifying that the downstream classifier obeys a property specified by a trusted generator. In this work, we predominantly consider verifying classifiers based on a trusted generator. The experiment shown in Table 7 is an example where one might consider the converse case: we can evaluate the generator against a trusted classifier that judges whether the produced images appear to be real.

## 8 Conclusion

We presented GENPROVE, a scalable non-convex relaxation approach to certify neural network properties when subjected to transformations learned by generative models. Our method supports both deterministic and probabilistic certification and is able to verify, for the first time, interesting visual transformation properties based on latent space interpolation, beyond the reach of prior work.

## References

- [1] Mislav Balunovic, Maximilian Baader, Gagandeep Singh, Timon Gehr, and Martin Vechev. 2019. Certifying Geometric Robustness of Neural Networks. In *NeurIPS*.
- [2] David Berthelot, Colin Raffel, Aurko Roy, and Ian Goodfellow. 2018. Understanding and Improving Interpolation in Autoencoders via an Adversarial Regularizer. In *ICLR*.
- [3] Xiaoyu Cao and Neil Zhenqiang Gong. 2017. Mitigating evasion attacks to deep neural networks via region-based classification. In *ACSAC*.
- [4] Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. 2019. Certified adversarial robustness via randomized smoothing. In *ICML*.
- [5] Patrick Cousot and Radhia Cousot. 1977. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*.
- [6] Patrick Cousot and Michael Monerau. 2012. Probabilistic Abstract Interpretation. In *Programming Languages and Systems*, Helmut Seidl (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 169–193.

- [7] Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Olivier Mastropietro, Alex Lamb, Martin Arjovsky, and Aaron Courville. 2017. Adversarially learned inference. In *ICLR*.
- [8] Vincent Dumoulin and Francesco Visin. 2016. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285* (2016).
- [9] Krishnamurthy Dvijotham, Marta Garnelo, Alhussein Fawzi, and Pushmeet Kohli. 2018. Verification of deep probabilistic models. *arXiv preprint arXiv:1812.02795* (2018).
- [10] Krishnamurthy Dvijotham, Sven Gowal, Robert Stanforth, Relja Arandjelovic, Brendan O'Donoghue, Jonathan Uesato, and Pushmeet Kohli. 2018. Training verified learners with learned verifiers. *arXiv preprint arXiv:1805.10265* (2018).
- [11] Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy A Mann, and Pushmeet Kohli. 2018. A Dual Approach to Scalable Verification of Deep Networks.. In *UAI*.
- [12] Akshat Gautam, Muhammed Sit, and Ibrahim Demir. 2020. Realistic River Image Synthesis using Deep Generative Adversarial Networks. *arXiv preprint arXiv:2003.00826* (2020).
- [13] Yixiao Ge, Zhuowan Li, Haiyu Zhao, Guojun Yin, Shuai Yi, Xiaogang Wang, et al. 2018. Fd-gan: Pose-guided feature distilling gan for robust person re-identification. In *NeurIPS*.
- [14] Timon Gehr, Matthew Mirman, Petar Tsankov, Dana Drachler Cohen, Martin Vechev, and Swarat Chaudhuri. 2018. AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation. In *S&P*.
- [15] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *NeurIPS*.
- [16] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and harnessing adversarial examples. In *ICLR*.
- [17] Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Timothy Mann, and Pushmeet Kohli. 2018. On the Effectiveness of Interval Bound Propagation for Training Verifiably Robust Models. *arXiv preprint arXiv:1810.12715* (2018).
- [18] Kyle D Julian, Mykel J Kochenderfer, and Michael P Owen. 2018. Deep neural network compression for aircraft collision avoidance systems. *Journal of Guidance, Control, and Dynamics* 42, 3 (2018), 598–608.
- [19] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. 2017. Reluplex: An efficient SMT solver for verifying deep neural networks. In *CAV*.
- [20] Hyunjik Kim and Andriy Mnih. 2018. Disentangling by factorising. In *ICML*.
- [21] Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *ICLR*.
- [22] Diederik P Kingma and Max Welling. 2013. Auto-encoding variational bayes. In *ICLR*.
- [23] Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, Hugo Larochelle, and Ole Winther. 2016. Autoencoding beyond pixels using a learned similarity metric. In *ICML*.
- [24] Yann LeCun, Corinna Cortes, and CJ Burges. 2010. MNIST handwritten digit database. *ATT Labs [Online]*. Available: <http://yann.lecun.com/exdb/mnist> 2 (2010).
- [25] Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. 2019. Certified robustness to adversarial examples with differential privacy. In *S&P*.
- [26] Bai Li, Changyou Chen, Wenlin Wang, and Lawrence Carin. 2018. Second-order adversarial attack and certifiable robustness. *arXiv preprint arXiv:1809.03113* (2018).
- [27] Yingzhen Li, John Bradshaw, and Yash Sharma. 2019. Are Generative Classifiers More Robust to Adversarial Attacks?. In *ICML*.
- [28] Chen Liu, Ryota Tomioka, and Volkan Cevher. 2019. On Certifying Non-uniform Bound against Adversarial Attacks. In *ICML*.
- [29] Jinxian Liu, Bingbing Ni, Yichao Yan, Peng Zhou, Shuo Cheng, and Jianguo Hu. 2018. Pose transferrable person re-identification. In *CVPR*.
- [30] Xuanqing Liu, Minhao Cheng, Huan Zhang, and Cho-Jui Hsieh. 2018. Towards robust neural networks via random self-ensemble. In *ECCV*.
- [31] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. 2015. Deep Learning Face Attributes in the Wild. In *ICCV*.
- [32] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards deep learning models resistant to adversarial attacks. In *ICLR*.
- [33] Matthew Mirman, Timon Gehr, and Martin Vechev. 2018. Differentiable Abstract Interpretation for Provably Robust Neural Networks. In *ICML*.
- [34] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. (2017).
- [35] Xuelin Qian, Yanwei Fu, Tao Xiang, Wenxuan Wang, Jie Qiu, Yang Wu, Yu-Gang Jiang, and Xiangyang Xue. 2018. Pose-normalized image generation for person re-identification. In *ECCV*.
- [36] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. 2018. Certified Defenses against Adversarial Examples. In *ICLR*.
- [37] Hadi Salman, Greg Yang, Huan Zhang, Cho-Jui Hsieh, and Pengchuan Zhang. 2019. A convex relaxation barrier to tight robustness verification of neural networks. In *NeurIPS*.
- [38] Pouya Samangouei, Maya Kabkab, and Rama Chellappa. 2018. Defense-GAN: Protecting Classifiers Against Adversarial Attacks Using Generative Models. In *ICLR*.
- [39] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. 2018. Fast and effective robustness certification. In *NeurIPS*.
- [40] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. 2019. An abstract domain for certifying neural networks. In *POPL*.
- [41] Yang Song, Taesup Kim, Sebastian Nowozin, Stefano Ermon, and Nate Kushman. 2018. PixelDefend: Leveraging Generative Models to Understand and Defend against Adversarial Examples. In *ICLR*.
- [42] Matthew Sotoudeh and Aditya V Thakur. 2019. Computing Linear Restrictions of Neural Networks. *arXiv preprint arXiv:1908.06214* (2019).
- [43] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic attribution for deep networks. In *ICML*.
- [44] Vincent Tjeng, Kai Xiao, and Russ Tedrake. 2019. Evaluating Robustness of Neural Networks with Mixed Integer Programming. In *ICLR*.
- [45] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. 2018. Efficient formal safety analysis of neural networks. In *NeurIPS*.
- [46] Xinlong Wang, Zhipeng Man, Mingyu You, and Chunhua Shen. 2017. Adversarial generation of training examples: applications to moving vehicle license plate recognition. *arXiv preprint arXiv:1707.03124* (2017).
- [47] Tsui-Wei Weng, Pin-Yu Chen, Lam M Nguyen, Mark S Squillante, Ivan Oseledets, and Luca Daniel. 2019. PROVEN: Certifying Robustness of Neural Networks with a Probabilistic Approach. In *ICML*.
- [48] Eric Wong, Frank Schmidt, Jan Hendrik Metzen, and J Zico Kolter. 2018. Scaling provable adversarial defenses. *NeurIPS*.
- [49] Xin Yi, Ekta Walia, and Paul Babyn. 2019. Generative adversarial network in medical imaging: A review. *Medical image analysis* (2019).
- [50] A. Yu and K. Grauman. 2014. Fine-Grained Visual Comparisons with Local Learning. In *CVPR*.
- [51] A. Yu and K. Grauman. 2017. Semantic Jitter: Dense Supervision for Visual Comparisons via Synthetic Images. In *ICCV*.

## A GENPROVE Propagation Pseudocode And Example

Here we show the pseudocode for the full propagation algorithm for GENPROVE, and provide an example of propagation using it. Here, we only show linear probabilistic computation, and do not demonstrate how the final output is verified against a constraint.

We will walk through Algorithm 1 using an example beginning with a line segment in two dimensions  $a = (1, 0)$  and  $b = (0, 1)$ , and a 1 layer neural network with the following weights and biases:

$$M_1 = \begin{pmatrix} 2 & 2 & 3 \\ -1 & 1 & 0 \end{pmatrix} \quad B_1 = (-1, 0, 1)$$

The algorithm first constructs a list  $D$  containing the single line segment from  $a$  to  $b$  with weight one. In the first iteration of the loop, there is only one iteration of first inner loop, where  $i = 1$ , and thus  $D_1$  is a segment so we proceed there. We create new start and end nodes for this segment,  $a := D_{i,3}M_1 + B_1$  and  $b := D_{i,4}M_1 + B_1$ . Specifically,

$$a = (1, 0) \begin{pmatrix} 2 & 2 & 3 \\ -1 & 1 & 0 \end{pmatrix} + (-1, 0, 1) = (1, 2, 4)$$

$$b = (0, 1) \begin{pmatrix} 2 & 2 & 3 \\ -1 & 1 & 0 \end{pmatrix} + (-1, 0, 1) = (-1, 1, 1).$$

We then fill a  $T$  with sorted zero-axis intersection times, starting with 0 and 1. Specifically, for each dimension  $d$  we calculate the time  $t_d$  such that  $(b_d - a_d)t_d + a_d = 0$ . We can compute this as  $t_d = -a_d / (b_d - a_d)$ . We only include this time if  $t_d$  is strictly between 0 and 1. In the example, we compute  $T = [0, 0.5, 1]$  as the intersection times for dimension  $d = 2$  and  $d = 3$  fall outside of 0 and 1.

For each time in this list, we compute the start,  $\tilde{a}$ , and end nodes,  $\tilde{b}$  for a new segment, and the probability  $p$  corresponding to that segment. The nodes of the segments are computed by interpolating between  $a$  and  $b$  using the times in  $T$  whereas the probability for each segment is the difference between the times of the nodes, multiplied by the probability of the original segment between  $a$  and  $b$ . As  $T$  has three nodes we calculate two segments. The first from  $(1, 2, 4)$  to  $(0, 1.5, 2.5)$  with  $p = 0.5$  and the second from  $(0, 1.5, 2.5)$  to  $(-1, 1, 1)$  with  $p = 0.5$ . We apply ReLU to each dimension of the nodes of the segments, and add these to a currently empty list, or domain element,  $\tilde{D}$ , of segments produced at that layer:

$$\tilde{D} = [(\text{Segment}, 0.5, (1, 2, 4), (0, 1.5, 2.5)), \\ (\text{Segment}, 0.5, (0, 1.5, 2.5), (0, 1, 1))].$$

Next, the algorithm determines which segments to merge using some Relax heuristic, which returns a list of relaxed boxes. Pedagogically, assume this returns a single box which contains both segments entirely. This is a box that goes from a minimal point  $(0, 1, 1)$  to a maximal point  $(1, 2, 4)$ , or as we

work with in the algorithm, has a center point  $(0.5, 1.5, 2.5)$  and radius  $(0.5, 0.5, 1.5)$ . Finally, we delete the segments that are contained within this box. We add this box to the list  $\tilde{D}$  with associated probability equivalent to the sum of the deleted elements probabilities. We note that this process also applies to merging boxes that are already part of  $\tilde{D}$ .

---

### Algorithm 1 Pseudocode for inference with GENPROVE

---

**Input:**  $k$  network layers with weights and biases  $M_i, B_i$ , and a line segment  $a, b$  in the input space.

**Output:**  $D$  a list of boxes and segments describing the probabilities of possible regions of the output space.

$D = [(\text{Segment}, 1, a, b)]$ .

**for**  $l = 1$  **to**  $k - 1$  **do**

$\tilde{D} = []$

**for**  $i = 1$  **to**  $|D|$  **do**

**if**  $D_i == \text{Segment}$  **then**

$a = D_{i,3}M_l + B_l$

$b = D_{i,4}M_l + B_l$

$T = [0, 1]$

**for**  $d = 1$  **to**  $|b|$  **do**

$t_d = \frac{-a_d}{(b_d - a_d)}$

**if**  $0 < t_d < 1$  **then**

$T.\text{push}(t_d)$

**end if**

**end for**

$T.\text{sort}()$

**for**  $t = 2$  **to**  $|T|$  **do**

$p = T_t - T_{t-1}$

$\tilde{a} = (b - a) * T_{t-1} + a$

$\tilde{b} = (b - a) * T_t + a$

$\tilde{D}.\text{push}((\text{Segment}, D_{i,2} * p, \text{ReLU}(\tilde{a}), \text{ReLU}(\tilde{b})))$

**end for**

**else**

$c = D_{i,3}M_l + b_l$

$r = D_{i,4}M_l|_p$

$\tilde{c} = \text{ReLU}(c + r) + \text{ReLU}(c - r)$

$\tilde{r} = \text{ReLU}(c + r) - \text{ReLU}(c - r)$

$\tilde{D}.\text{push}((\text{Box}, D_{i,2}, 0.5 * \tilde{c}, 0.5 * \tilde{r}))$

**end if**

**end for**

$\tilde{P} = \text{Relax}(\tilde{D})$

**for**  $p = 1$  **to**  $|\tilde{P}|$  **do**

**for**  $i = 1$  **to**  $|\tilde{D}|$  **do**

**if**  $\gamma(\tilde{D}_i) \subseteq \gamma(\tilde{P}_p)$  **then**

$\tilde{P}_{p,2} = \tilde{P}_{p,2} + \tilde{D}_{i,2}$

**delete**  $\tilde{D}_i$

**end if**

**end for**

**end for**

$D = \tilde{D} + \tilde{P}$

**end for**

---

## B Network Architectures

Our experiments use two different encoder architectures (Encoder and EncoderSmall), two decoder architectures (Decoder and DecoderSmall), and four different classifier/attribute detector architectures (ConvSmall, ConvMed, ConvLarge, ConvBiggest). These are described in detail here.

Here we use  $\text{Conv}_s C \times W \times H$  to denote a convolution which produces  $C$  channels, with a kernel width of  $W$  pixels and height of  $H$ , with a stride of  $s$  and padding of 1. FC  $n$  is a fully connected layer which outputs  $n$  neurons.  $\text{ConvT}_{s,p} C \times W \times H$  is a transposed convolutional layer [8] with a kernel width and height of  $W$  and  $H$  respectively and a stride of  $s$  and padding of 1 and out-padding of  $p$ , which produces  $C$  output channels.  $l$  refers to the number of latent dimensions, and  $o$  refers to either the number of attributes or number of classes. For CelebA and Zappos50k use 64 latent dimensions, while the VAE for MNIST uses 50 latent dimensions.

- *EncoderSmall* is a standard convolutional neural network with 74128 neurons. It is used for encoding MNIST and Zappos50k. It is trained with Adam [21] with a learning rate of 0.001 and a batch size of 128. The network was trained for 300 epochs for Zappos50k and 20 epochs for MNIST.

$$\begin{aligned} x &\rightarrow \text{Conv}_2 16 \times 4 \times 4 \rightarrow \text{ReLU} \\ &\rightarrow \text{Conv}_2 32 \times 4 \times 4 \rightarrow \text{ReLU} \\ &\rightarrow \text{FC } 100 \\ &\rightarrow l. \end{aligned}$$

- *Encoder* is also a standard convolutional neural network, but significantly larger with 246784 neurons, used only for encoding CelebA. It is trained with Adam with a learning rate of 0.0001 and a batch size of 100 for 20 epochs.

$$\begin{aligned} x &\rightarrow \text{Conv}_1 32 \times 3 \times 3 \rightarrow \text{ReLU} \\ &\rightarrow \text{Conv}_2 32 \times 4 \times 4 \rightarrow \text{ReLU} \\ &\rightarrow \text{Conv}_1 64 \times 3 \times 3 \rightarrow \text{ReLU} \\ &\rightarrow \text{Conv}_2 64 \times 4 \times 4 \rightarrow \text{ReLU} \\ &\rightarrow \text{FC } 512 \rightarrow \text{ReLU} \\ &\rightarrow \text{FC } 512 \\ &\rightarrow l. \end{aligned}$$

- *Decoder* is a transposed convolutional network which has 74128 neurons used for decoding every dataset in nearly every experiment, unless otherwise specified. Of course, the training parameters are the same as the respective encoders.

$$\begin{aligned} l &\rightarrow \text{FC } 400 \rightarrow \text{ReLU} \\ &\rightarrow \text{FC } 2048 \rightarrow \text{ReLU} \\ &\rightarrow \text{ConvT}_{2,1} 16 \times 3 \times 3 \rightarrow \text{ReLU} \\ &\rightarrow \text{ConvT}_{1,0} 3 \times 3 \times 3 \\ &\rightarrow x. \end{aligned}$$

- *DecoderSmall* is a smaller transposed convolutional network which has 41160 neurons used for decoding

CelebA for testing GENPROVECURVE. The training parameters are the same as the respective encoders.

$$\begin{aligned} l &\rightarrow \text{FC } 200 \rightarrow \text{ReLU} \\ &\rightarrow \text{FC } 2048 \rightarrow \text{ReLU} \\ &\rightarrow \text{ConvT}_{2,1} 8 \times 3 \times 3 \rightarrow \text{ReLU} \\ &\rightarrow \text{ConvT}_{1,0} 3 \times 3 \times 3 \\ &\rightarrow x. \end{aligned}$$

- *ConvSmall* is a convolutional network which has 24676 neurons. The convolutions use a padding of 1. It is only used for a toy parameter comparison on CelebA. It was trained for 300 epochs with a batch size of 100 using Adam with a learning rate of 0.0001. It had a test-set accuracy of 89.87%.

$$\begin{aligned} x &\rightarrow \text{Conv}_2 16 \times 4 \times 4 \rightarrow \text{ReLU} \\ &\rightarrow \text{Conv}_2 32 \times 4 \times 4 \rightarrow \text{ReLU} \\ &\rightarrow \text{FC } 100 \\ &\rightarrow o. \end{aligned}$$

- *ConvMed* is a convolutional network which has 63804 neurons. Here, the convolutions use a padding of 1. This is used as a classifier and attribute detector for Zappos50k and CelebA experiments. For both experiments it was trained with a batch size of 128 using Adam with a learning rate of 0.001. For Zappos50k it was trained for 5 epochs and achieved a test-set accuracy of 79.40%. For CelebA it was trained for 10 epochs and achieved a test-set accuracy of 89.87%.

$$\begin{aligned} x &\rightarrow \text{Conv}_1 12 \times 4 \times 4 \rightarrow \text{ReLU} \\ &\rightarrow \text{Conv}_2 16 \times 4 \times 4 \rightarrow \text{ReLU} \\ &\rightarrow \text{FC } 500 \\ &\rightarrow \text{FC } 200 \\ &\rightarrow \text{FC } 100 \\ &\rightarrow o. \end{aligned}$$

- *ConvLarge* is a convolutional network which has 123180 neurons. Here, the convolutions use a padding of 1. This is used as a classifier and attribute detector for Zappos50k and CelebA experiments. For both experiments it was trained with a batch size of 128 using Adam with a learning rate of 0.001. For Zappos50k it was trained for 5 epochs and achieved a test-set accuracy of 82.20%. For CelebA it was trained for 10 epochs and achieved a test-set accuracy of 89.86%.

$$\begin{aligned} x &\rightarrow \text{Conv}_1 16 \times 3 \times 3 \rightarrow \text{ReLU} \\ &\rightarrow \text{Conv}_2 16 \times 4 \times 4 \rightarrow \text{ReLU} \\ &\rightarrow \text{Conv}_1 32 \times 3 \times 3 \rightarrow \text{ReLU} \\ &\rightarrow \text{Conv}_2 32 \times 4 \times 4 \rightarrow \text{ReLU} \\ &\rightarrow \text{FC } 200 \\ &\rightarrow \text{FC } 100 \\ &\rightarrow o. \end{aligned}$$

- *ConvBiggest* is a convolutional network which has 175816 neurons. Here, the convolutions use a padding of 1. This is used as a classifier detector for MNIST experiments. When trained with DiffAI the training



schedule suggested by Goyal et al. [17] is used. Each method trains it for 30 epochs.

$$\begin{aligned}
 x &\rightarrow \text{Conv}_1 64 \times 3 \times 3 && \rightarrow \text{ReLU} \\
 &\rightarrow \text{Conv}_1 64 \times 3 \times 3 && \rightarrow \text{ReLU} \\
 &\rightarrow \text{Conv}_2 128 \times 3 \times 3 && \rightarrow \text{ReLU} \\
 &\rightarrow \text{Conv}_1 128 \times 3 \times 3 && \rightarrow \text{ReLU} \\
 &\rightarrow \text{Conv}_1 128 \times 3 \times 3 && \rightarrow \text{ReLU} \\
 &\rightarrow \text{FC } 200 \\
 &\rightarrow o.
 \end{aligned}$$

### C GENPROVE Refinement Schedule

While many refinement schemes start with an imprecise approximation and progressively tighten it, we observe that

being only occasionally memory limited and rarely time limited, it conserves more time to start with the most precise approximation we have determined usually works, and progressively try less precise approximations as we determine that more precise ones can not fit into GPU memory. Thus, we start searching for a probabilistic robustness bound with  $\text{GENPROVE}_N^p$  and if we run out of memory, try  $\text{GENPROVE}_{\max(0.95N, 5)}^{\min(1.5p, 1)}$  for schedule A, and  $\text{GENPROVE}_{\max(0.95N, 5)}^{\min(3p, 1)}$  for schedule B. This procedure is repeated until a solution is found, or time has run out.

### D Detailed Experimental Output

**Table 8.** Average consistency  $\hat{C}$  bound widths, runtime, and memory usage. For these metrics, lower values are better. Additionally, the percentage of runs which ran out of memory is reported as OOM. Unacceptably large widths, which mean the analysis failed to provide useful bounds, are written in red.

Dataset	Network	Neurons	Domain	Width ( $u - l$ )	Seconds	GPU Memory		
						OOM (%)	Peak (GB)	
CelebA	ConvSmall	24676	Prior Work	Box	0.98	0.0031	0	0.07
				HybridZono	0.9703	0.0042	0	0.07
				DeepZono	1.0	1.3485	100	23.62
				Zonotope	1.0	1.3345	100	23.62
			Our Work	GENPROVE <sup>0</sup>	0.0	10.7657	0	7.05
				GENPROVE <sup>0.02</sup> <sub>100</sub>	$1.78 \times 10^{-5}$	12.7403	0	3.51
			99.999% Confidence	Sampling	$3.73 \times 10^{-4}$	14.7877	0	0.62
	ConvMed	63804	Prior Work	Box	0.98	0.0046	0	0.16
				HybridZono	0.97	0.0059	0	0.16
				DeepZono	1.0	1.6848	100	23.62
				Zonotope	1.0	1.5015	100	23.62
			Our Work	GENPROVE <sup>0</sup>	0.9	1.2914	90	22.77
				GENPROVE <sup>0.02</sup> <sub>100</sub>	$1.10 \times 10^{-4}$	25.3728	0	6.83
			99.999% Confidence	Sampling	$3.11 \times 10^{-4}$	26.4949	0	0.70
	ConvLarge	123180	Prior Work	Box	0.98	0.0040	0	0.08
				HybridZono	0.97	0.0202	0	0.08
				DeepZono	1.0	0.9907	100	23.62
				Zonotope	1.0	0.9560	100	23.62
			Our Work	GENPROVE <sup>0</sup>	0.95	0.7649	95	23.14
				GENPROVE <sup>0.02</sup> <sub>100</sub>	$1.61 \times 10^{-4}$	41.3746	0	9.38
			99.999% Confidence	Sampling	$2.06 \times 10^{-4}$	42.2111	0	0.71
Zappos50k	ConvSmall	24676	Prior Work	Box	1.0	0.0039	0	0.04
				HybridZono	1.0	0.0048	0	0.04
				DeepZono	1.0	1.3525	100	23.62
				Zonotope	1.0	1.3428	100	23.62
			Our Work	GENPROVE <sup>0</sup>	0.0	11.1377	0	6.54
				GENPROVE <sup>0.02</sup> <sub>100</sub>	$3.26 \times 10^{-5}$	14.8033	0	6.39
			99.999% Confidence	Sampling	$1.59 \times 10^{-3}$	15.2826	0	0.59
	ConvMed	63804	Prior Work	Box	1.0	0.0053	0	0.13
				HybridZono	1.0	0.0380	0	0.13
				DeepZono	1.0	1.3513	100	23.62
				Zonotope	1.0	1.3402	100	23.62
			Our Work	GENPROVE <sup>0</sup>	0.89	3.4067	89	22.67
				GENPROVE <sup>0.02</sup> <sub>100</sub>	$4.53 \times 10^{-5}$	25.1927	0	6.61
			99.999% Confidence	Sampling	$1.13 \times 10^{-3}$	27.0847	0	0.67
	ConvLarge	123180	Prior Work	Box	1.0	0.005	0	0.046
				HybridZono	1.0	0.038	0	0.046
				DeepZono	1.0	1.359	100	23.623
				Zonotope	1.0	1.350	100	23.623
			Our Work	GENPROVE <sup>0</sup>	0.99	0.414	99	23.581
				GENPROVE <sup>0.02</sup> <sub>100</sub>	$5.7 \times 10^{-5}$	32.058	0	7.160
			99.999% Confidence	Sampling	$1.53 \times 10^{-3}$	32.124	0	0.679