Shared Certificates for Neural Network Verification

Anonymous Authors

No Institute Given

Abstract. Existing neural network verifiers compute a proof that each input is handled correctly under a given perturbation by propagating a symbolic abstraction of reachable values at each layer. This process is repeated from scratch independently for each input (e.g., image) and perturbation (e.g., rotation), leading to an expensive overall proof effort when handling an entire dataset. In this work we introduce a new method for reducing this verification cost without losing precision based on a key insight that abstractions obtained at intermediate layers for different inputs and perturbations can overlap or contain each other. Leveraging our insight, we introduce the general concept of shared certificates, enabling proof effort reuse across multiple inputs to reduce overall verification costs. We perform an extensive experimental evaluation to demonstrate the effectiveness of shared certificates in reducing the verification cost on a range of datasets and attack specifications on image classifiers including the popular patch and geometric perturbations.

1 Introduction

The success of neural networks across a wide range of application domains [1, 2] has lead to their widespread application and study. Despite this success, neural networks remain vulnerable against adversarial attacks [3, 4] which raises concerns over their trustworthiness in safety-critical settings such as autonomous driving and medical devices. To overcome this barrier, formal verification of neural networks has been proposed as a key technology in the literature [5]. As a result, recent years have witnessed growing interest in verifying critical safety properties of neural networks (e.g., fairness, robustness) [6–27] specified using pre and post conditions over network inputs and outputs respectively. Conceptually, existing verifiers work by propagating sets of inputs in the precondition captured in symbolic form (e.g., convex sets) through the network, an expensive process which produces over-approximations of all possible activations at intermediate layers. The final abstraction of the output can then be used to check postconditions. The key technical challenge all existing verifiers aim to address is speeding up and scaling the certification process, i.e, faster and more efficient propagation of symbolic shapes, while reducing the overapproximation error.

This work: accelerating certification via proof sharing. In this work we propose a new, complementary method for accelerating neural network verification based

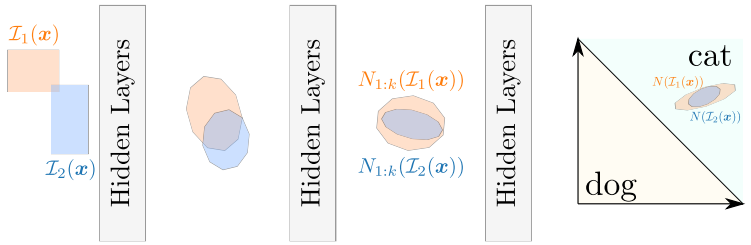


Fig. 1: Visualization of neural network verification. The input regions $\mathcal{I}_1(\boldsymbol{x}), \mathcal{I}_2(\boldsymbol{x})$ are propagated layer by layer through a neural network N. The high-dimensional convex shapes are visualized in 2d. While initially $\mathcal{I}_1(\boldsymbol{x})$ and $\mathcal{I}_2(\boldsymbol{x})$ only slightly overlap, at layer k, $N_{1:k}(\mathcal{I}_2(\boldsymbol{x}))$ is fully contained in $N_{1:k}(\mathcal{I}_1(\boldsymbol{x}))$.

on the key observation that instead of treating each certification attempt in isolation as existing verifiers do, we can reuse proof effort among multiple such attempts, thus obtaining significant overall speed-ups without losing precision. Fig. 1 illustrates both, standard verification and the concept of proof sharing.

In standard verification an input region $\mathcal{I}_1(\boldsymbol{x})$ (orange square) is propagated from left to right, obtaining intermediate shapes at each intermediate layer (here the goal is to verify all points in the input region are classified as "cat"). We observe that the abstraction obtained for a new region $\mathcal{I}_2(\boldsymbol{x})$ (e.g., blue shapes) can be contained inside existing abstractions from $\mathcal{I}_1(\boldsymbol{x})$, an effect we term *proof subsumption*. This effect can be observed both between abstractions obtained from different specifications (e.g., ℓ_∞ and adversarial patches) for the same data point and between proofs for the same property but different, yet semantically similar inputs. Building on this observation, we introduce the notion of proof sharing via templates. Proof sharing works in two steps: first we leverage abstractions from existing proofs in order to create templates, and second, we augment the verifier with these templates, stopping the expensive propagation at an intermediate layer as soon as the newly generated abstraction is included inside an existing template. Key technical ingredients to the effectiveness of our approach are fast template generation and inclusion checking techniques. We experimentally demonstrate that proof sharing can achieve significant speed-ups in challenging scenarios including proving robustness to adversarial patches [28] and geometric perturbations [29] across different neural network architectures.

Main Contributions Our key contributions are:

- An introduction and formalization of the concept of proof sharing in neural network verification: the idea that some proofs capture others (§3).
- A framework leveraging the above concept, enabling proof effort reuse via proof templates (§4).
- A thorough experimental evaluation involving verification of robustness to adversarial patch and geometric perturbations, demonstrating templates with proof match rates of up 95% as well as non-trivial end-to-end certification speed-ups (§5).

2 Background

Here we formally introduce the necessary background for proof sharing.

Neural Network A neural network N is a function $N : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}}$, commonly build from individual layers $N = N_L \circ N_{L-1} \circ \cdots \circ N_1$. Each individual layer N_i is made up of an affine transformation $N_i(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}$ (often called linear) and, in all but the last layer N_L, an elementwise non-linearity (called activation). The outputs of an layer are called activations. A common choice of non-linearity is the rectified linear unit (ReLU). For example, a linear layer with ReLU activation is specified by the equation $N_i(\boldsymbol{x}) = \max(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}, 0)$, where the maximum is applied elementwise. Networks that alternatively use linear layers and activation layers are called feed-forward neural networks. Throughout this text we consider feed-forward neural networks with ReLU activations. Our approach naturally extends to other types of neural networks, if verifiers exist for these architectures. While, as is common in the neural network verification literature, we use image classification as a proxy task, many other applications work analogously. A neural network classifying inputs into c classes outputs $d_{\text{out}} := c$ scores, one for each class, and assigns the class with the highest score.

In the following, for $k < L$, we let $N_{1:k}$ denote the first k layers and $N_{k+1:L}$ denote the last $L - k$ layers.

(Local) Neural Network Verification The goal in neural network verification is to prove a postcondition ψ over the output of the neural network.

In this work prove we local verification: proving a property ψ holds for a given region $\mathcal{I}(\boldsymbol{x})$ formed around the input $\boldsymbol{x}$. The problem is formally stated as follows:

Problem 1 (Local neural network verification). For a specification $\mathcal{I}(\boldsymbol{x})$, neural network N, and postcondition ψ verify that $\forall \boldsymbol{z} \in \mathcal{I}(\boldsymbol{x}).\ N(\boldsymbol{z}) \vdash \psi$.

In this work we focus on verifiers, based on abstract interpretation [16, 30], propagate $\mathcal{I}(\boldsymbol{x})$ through the network N, resulting in an abstraction at each layer and finally check if the abstracted output implies ψ. The lifting of a function from concrete inputs to abstractions is called an abstract transformer. This is showcased in Fig. 1, where the shapes $\mathcal{I}_1(\boldsymbol{x})$ and $\mathcal{I}_2(\boldsymbol{x})$ (input specifications or input regions) are propagated layer-by-layer through the neural network N.

For a verifier V, we let $V(\mathcal{I}(\boldsymbol{x}), N)$ denote the propagation of $\mathcal{I}(\boldsymbol{x})$ through the network N. We aim to declutter notation by but overload N slightly and writing $N(\mathcal{I}(\boldsymbol{x}))$ for the same if V is clear form context, i.e., $\{N(\boldsymbol{z}) \mid \boldsymbol{z} \in \mathcal{I}(\boldsymbol{x})\} \subseteq V(\mathcal{I}(\boldsymbol{x}), N) = N(\mathcal{I}(\boldsymbol{x}))$.

In the case of robustness verification, where one tries to prove classification invariance to adversarial examples [31, 32], a common input region is the ℓ_∞-bounded additive noise, defined as $\mathcal{I}_\epsilon(\boldsymbol{x}) := \{\boldsymbol{z} \mid \|\boldsymbol{x} - \boldsymbol{z}\|_\infty \leq \epsilon\}$. Here, ϵ defines the size of the maximal perturbation to $\boldsymbol{x}$, while ψ denotes classification to the same class as $\boldsymbol{x}$. Throughout this paper, we consider different instantiations for $\mathcal{I}(\boldsymbol{x})$ but assume that ψ denotes classification invariance (although other choices

Table 1: Proof subsumption on a robust MNIST classifier with 94 % accuracy. Verif. acc. denotes the percentage of verifiable inputs from the test set for ℓ_∞-perturbations ($\mathcal{I}_\epsilon$).

ϵ	verif. acc. [%]	$\mathcal{I}_p(\boldsymbol{x}) \subseteq \mathcal{I}_\epsilon(\boldsymbol{x})$ at layer k [%]				
		1	2	3	4	5
0.1	89.74	61.40	72.85	77.65	81.75	82.70
0.2	81.40	62.85	77.05	82.40	86.05	86.60

Fig. 3: The abstraction obtained for $\mathcal{I}_\epsilon(\boldsymbol{x})$ (blue) contains that for $\mathcal{I}^{i,j}_{p\times p}(\boldsymbol{x})$ (orange) (projected to $d = 2$).

would work analogously). For example, in Fig. 1, we can verify that all point contained in $N(\mathcal{I}_1(\boldsymbol{x}))$ are classified as "cat".

3 Proof Sharing with Templates

Before introducing our framework for proof sharing, we further expand the motivation example discussed in Fig. 1.

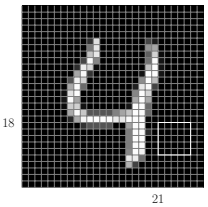


Fig. 2: Example of an MNIST image. $\mathcal{I}^{18,21}_{5\times5}(\boldsymbol{x})$ signifies arbitrary change in the outlined area.

3.1 Motivation: Proof Subsumption

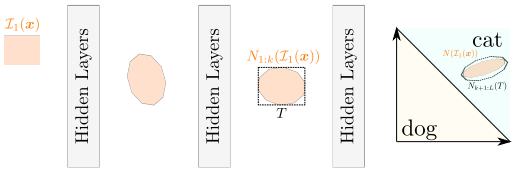
As stated earlier, we empirically observed that for many input regions $\mathcal{I}_i(\boldsymbol{x})$ and $\mathcal{I}_j(\boldsymbol{x})$, one abstraction contains another at some intermediate layer k, i.e., $N_{1:k}(\mathcal{I}_i(\boldsymbol{x})) \subseteq N_{1:k}(\mathcal{I}_j(\boldsymbol{x}))$, even though $\mathcal{I}_i(\boldsymbol{x}) \not\subseteq \mathcal{I}_j(\boldsymbol{x})$. We refer to this property as proof subsumption. Proof subsumption is showcased in Fig. 1, where $\mathcal{I}_1(\boldsymbol{x})$ and $\mathcal{I}_2(\boldsymbol{x})$ have only small overlap, yet $N_{1:k}(\mathcal{I}_2(\boldsymbol{x})) \subseteq N_{1:k}(\mathcal{I}_1(\boldsymbol{x}))$.

Consider N as a hand-written digit classifier for the MNIST dataset [33] (example shown in Fig. 2) and the following two specifications:

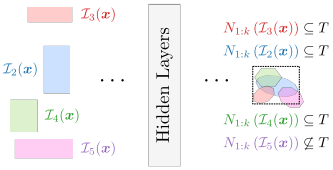
- ℓ_∞-bound perturbations: a small change to all the pixels in an input image $\mathcal{I}_\epsilon(\boldsymbol{x}) := \{\boldsymbol{z} \mid \|\boldsymbol{x} - \boldsymbol{z}\|_\infty \leq \epsilon\}$,
- and adversarial patches [28]. We let $\mathcal{I}^{i,j}_{p\times p}$ denote an $p \times p$ patch, placed on an image at coordinates (i, j). We showcase this in Fig. 2 and will formally define it later in this paper.

Evaluating such a classifier (technical details in App. A.1) in practice we indeed observe this effect. Table 1 shows the accuracy, the rate of correct predictions on the unperturbed test data, as well as the certified accuracy for $\mathcal{I}_\epsilon$ with $\epsilon = 0.1$ and 0.2. Further it show the percentage of $\mathcal{I}^{i,j}_{p\times p}(\boldsymbol{x})$ contained in $\mathcal{I}_\epsilon(\boldsymbol{x})$ at layer k .

We observe that regions with larger ϵ yield more inclusions. Further we observe that the containment increases as k increases. These two observations give



(a) We generate a (verifiable) template T from the abstraction obtained by propagating the orange input.



(b) We shortcut the verification if intermediate abstraction are contained in the T.

Fig. 4: Conceptualization of proof sharing with templates. In (a) we create a verifiable template T (black-dashed border) from specification $N_{1:k}(\mathcal{I}_1(\boldsymbol{x}))$. When verifying new specifications $\mathcal{I}_2 \dots \mathcal{I}_5$, shown in (b) we can shortcut the verification of all but $\mathcal{I}_5$ by subsuming them in T.

an intuition as to why we observe proof subsumption. As specifications pass through the neural network, in each layer the abstractions become more imprecise. While this fundamentally limits verification, it makes subsumption of abstractions more probable. Further, while passing through the layers of a neural network, we observed that semantically similar yet distinct image inputs, e.g., two similar looking handwritten digits, have activation vectors that grow closer in ℓ_2 norm as they pass through the layers of the neural network [1, 31]. We conjecture that these two effects are driving the observed proof subsumption. As larger ϵ also increases the size of the obtained abstractions for an ℓ_∞-perturbation, they also increase the rate of other abstractions matching them.

Fig. 3 shows a patch specification $\mathcal{I}^{i,j}_{p\times p}(\boldsymbol{x})$ (in orange) contained in the ℓ_∞ specification $\mathcal{I}_\epsilon$ (in blue) projected to 2 dimensions.

3.2 Proof Sharing with Templates

Leveraging this insight, we introduce the idea of proof sharing via templates, showcased in Fig. 4. There we use an abstraction obtained from a robustness proof $N_{1:k}(\mathcal{I}_1(\boldsymbol{x}))$ at layer k to create a template T. After ensuring that T is verifiable, it can be used to shortcut the evaluation of further proofs, e.g., of $\mathcal{I}_2(\boldsymbol{x}), \dots, \mathcal{I}_5(\boldsymbol{x})$.

Formally we decompose this into two sub-problems: the generation of proof templates and the matching of abstractions corresponding to other properties to these templates.

Our goal in proof sharing is to construct a suitable template T at layer k by using the abstractions produced by the verification procedure, so that T captures further abstractions at layer k obtained from propagating unseen $\mathcal{I}(\boldsymbol{x})$, yet still ensure that T verifies. As it can be challenging to capture all input regions with a single T, we allow a set of templates $\mathcal{T}$. Finding a set of templates is formally stated as:

Problem 2 (Template Generation). For a given neural network N, input $\boldsymbol{x}$ and set of specifications $\mathcal{I}_1, \dots \mathcal{I}_r$, layer k and a postcondition ψ, find a set of tem-

plates $\mathcal{T}$ with $|\mathcal{T}| \leq m$ such that:

$$\max_{\mathcal{T}} \sum_{i=1}^{r} \left[\bigvee_{T \in \mathcal{T}} N_{1:k}(\mathcal{I}_r(\boldsymbol{x})) \subseteq T \right] \quad (1)$$
$$\text{s.t. } \forall\, T \in \mathcal{T}.\, N_{k+1:L}(T) \vdash \psi$$

There are infinite number of template sets $\mathcal{T}$ satisfying Problem 2, therefore solving Problem 2 exactly is computationally infeasible. Therefore we compute an approximate solution to Eq. (1). In general, Problem 2 does not necessarily require that the templates T are created from previous proofs. However, building on proof subsumption as discussed in §3.1, in §4 we will infer the templates from previously obtained abstractions.

To leverage proof sharing once the templates $\mathcal{T}$ are obtained, we need to be able to match an abstraction $S = N_{1:k}(\mathcal{I}(\boldsymbol{x}))$ to a template in $\mathcal{T}$:

Problem 3 (Template Matching). Given a set of templates $\mathcal{T}$ at layer k of a neural network N, and a new input region $\mathcal{I}(\boldsymbol{x})$, determine whether there exists a $T \in \mathcal{T}$ such that $S \subseteq T$, where $S = N_{1:k}(\mathcal{I}(\boldsymbol{x}))$.

Together, Problems 2 and 3 outline a general framework for proof sharing, permitting many instantiations. We note that Problems 2 and 3 present an inherent precision vs. speed trade-off: Problem 3 can be solved most efficiently for small values of $m = |\mathcal{T}|$ and simpler representations of T (allowing faster checking of $S \subseteq T$). Alternatively, Eq. (1) can be maximized by large m and T represented by complex abstractions, thus attaining high precision.

Next, in §4 we use this framework to construct templates in an online setting. That is, upon receiving $\boldsymbol{x}$ and $\mathcal{I}_1, \ldots \mathcal{I}_r$ we first create a set of templates and then proceed to verify $\mathcal{I}_1, \ldots \mathcal{I}_r$.

Beyond proof sharing on the same input In this section focused on proof sharing for different specifications on the same input $\boldsymbol{x}$. However, we observed that proof sharing is even possible between different inputs $\boldsymbol{x}$ and $\boldsymbol{x}'$. To this end Eq. (1) in Problem 2 can be changed to maximize the expected template matches over a input distribution. We provide an investigation along these lines in App. C.

4 Efficient Verification via Proof Sharing

We now consider an instantiation of proof sharing, where we are given an input $\boldsymbol{x}$ and properties $\mathcal{I}_1, \ldots \mathcal{I}_r$ to verify. Our general approach, based on Problems 2 and 3 is shown in Algorithm 1. In this section we first discuss Algorithm 1 in general, then describe the possible choices of abstract domains and their implications on the algorithm. Finally, we will discuss template generation for two different specific problems.

In Algorithm 1, we first create the set of templates $\mathcal{T}$ (Line 1, discussed shortly) and subsequently verify $\mathcal{I}_1, \ldots \mathcal{I}_r$ using $\mathcal{T}$. For each $\mathcal{I}_i$ we propagate

Algorithm 1: Neural Network Verification Utilizing Proof Templates

Input: $\boldsymbol{x}, \mathcal{I}_1, \ldots, \mathcal{I}_r, k, \psi$, verifiers V_S, V_T
Result: $v_1, \ldots, v_r$ indicating $v_i := (N(\mathcal{I}_i(\boldsymbol{x})) \vdash \psi)$

```
1  T ← GENERATE_TEMPLATES(x, N, k, ψ)
2  v_1, ..., v_r ← False
3  for i ← 1 to r do
4  |  S ← V_S(I_i(x), N_{1:k})
5  |  for T_j ∈ T do
6  |  |  if S ⊆ T_j then
7  |  |  |  v_i ← True
8  |  |  |  break
9  |  |  end
10 |  end
11 |  if ¬v_i then
12 |  |  v_i ← (V_S(S, N_{k+1:L}) ⊢ ψ)
13 |  end
14 end
15 return v_1, ..., v_r
```

it up to a chosen layer k (Line 4) to obtain $S = N_{1:k}(\mathcal{I}_i(\boldsymbol{x}))$ and check if we can match it to a template $T_j \in \mathcal{T}$ (Line 6) using inclusion check. If a match is found, then we conclude that $N(\mathcal{I}_i(\boldsymbol{x})) \vdash \psi$ and set the verification output v_i to True. If this is not the case (Line 11) we check $N(\mathcal{I}_i(\boldsymbol{x})) \vdash \psi$ directly by checking $V_S(S, N_{k+1:L}) \vdash \psi$. Should the template generation fail, we revert to verifying $\mathcal{I}_i$ by applying V_S in the usual way.

Soundness As long as the templates T are sound, this procedure is sound, i.e Algorithm 1 only returns $v_i = \text{True}$ if truly $\forall \boldsymbol{z} \in \mathcal{I}_i(\boldsymbol{x}).\ N(z) \vdash \psi$. Formally:

Theorem 1. *Algorithm 1 is sound if* $\forall\ T \in \mathcal{T},\ z \in T.\ N_{k+1:L}(z) \vdash \psi$ *and* V_S *is sound.*

Proof. For a given $\boldsymbol{x}$ and $\mathcal{I}_i$, Algorithm 1 only claims $v_i = \text{True}$ if either the check in (i) Line 6 or (ii) Line 11 succeeds. Since V_S is sound, we know that $\forall \boldsymbol{z} \in \mathcal{I}_i(\boldsymbol{x}).\ N_{1:k}(z) \in S$. Therefore in case (i) by our requirement on T as well as $S \subseteq T$ it follows that $\forall \boldsymbol{z} \in \mathcal{I}_i(\boldsymbol{x}).\ N(z) \vdash \psi$. In case (ii) the same property holds due to the soundness of V_S.

An important takeaway of Theorem 1 is that the generation process of $\mathcal{T}$ does not affect the overall soundness as long as it can be ensured that the final set of templates fulfills the condition in Theorem 1. In particular that means that when solving Problem 2, it suffices to show the side condition $(\forall\ T \in \mathcal{T}.\ N_{k+1:L}(T) \vdash \psi)$ holds, while heuristically approximating the actual optimization criteria. We let V_T denote the verifier used to ensure this property in the GENERATE_TEMPLATES procedure.

Precision We say a verifier V_1 is more precise than another verifier V_2 on N if out of a set of specifications it can verify more than V_2.

Theorem 2. *If* $V_S(V_S(\mathcal{I}_i(\boldsymbol{x}), N_{1:k}), N_{k+1:L}) = V_S(\mathcal{I}_i(\boldsymbol{x}), N)$*, then Algorithm 1 is at least as precise as* V_S*.*

Proof. Since, even if the inclusion check in Line 6 fails, due to Line 12 we output $v_i = V_S(V_S(\mathcal{I}_i(\boldsymbol{x}), N_{1:k}), N_{k+1:L}) \vdash \psi$ (Line 12), which by our requirement equals $v_i = V_S(\mathcal{I}_i(\boldsymbol{x}), N) \vdash \psi$. Therefore we have at least the precision of V_S.

For verifiers V_S that do not fulfill the required property, potential losses in precision can be remedied (at the cost of runtime) by using $V_S(\mathcal{I}_i(\boldsymbol{x}), N_{1:L})$ in Line 12. Interestingly, it is even possible to increase the precision of Algorithm 1 over V_S by creating templates T that are verified with a more precise verifier V_T. However, in this discussion we restrict ourself to speed gains. We believe that obtaining precision gains requires instantiating our framework with a significantly different approach than that taken for improving speed which is the main focus of our work. We leave this as an interesting item for future work.

Speedup When solving Problem 2, the quality of the optimization as well as the run-time are key factors affecting the obtained speedup. Assuming, for simplicity, that $|\mathcal{T}| = 1$ and that all the verification of all $\mathcal{I}_1, \ldots, \mathcal{I}_r$ takes similar time, the expected run-time of Algorithm 1 is

$$t_{\mathcal{T}} + r(t_S + t_{\subseteq} + (1-p)t_\psi) \tag{2}$$

where $t_{\mathcal{T}}$ is the time required to generate the templates at Line 1, r is the number of specifications to be verified, t_S is the time required to compute S (Line 4) and $t_{\subseteq}$ is the time to check $S \subseteq T$ (Line 6), $p \in [0, 1]$ is the rate at which inclusion checks succeed and t_ψ is the time required to check ψ on S (Line 12).

Assuming, for simplicity, that $r(t_S + t_\psi)$ approximates the runtime of verifying the specifications directly with V_S (which we also empirically find to hold) we can calculate the speedup gained by *Algorithm* 1 by comparing it with Eq. (2). We note that the rt_S term appears in both expression and thus can't be improved upon. That leaves us $t_T + rt_{\subseteq} + r(1-p)t_\psi$ to offset rt_ψ and gain a speed-up. Other than by decreasing $t_{\subseteq}$, this speedup can be increased by speeding up template generation t_T or improving the quality of the templates, measured by p – two competing goals, making it important to correctly tune the template generation algorithm. Similarly, the parameter k, the layer at which the templates are generated, is of of critical importance. We have seen in §3.1 that in later layers more proof subsumption occurs. However, placing the templates earlier in the network minimizes the ratio $\frac{t_S}{t_\psi}$ and thereby maximizes the potential for speedups.

To showcase how we can achieve good templates as well as fast matching, we next discuss the choice of abstract domain to be used in the propagation and representation of the templates. Then we discuss the template generation procedure, and instantiate it for the verification of robustness to adversarial patches and geometric perturbations.

4.1 Choice of Abstract Domain

In order to solve Problems 2 and 3 in a way that minimizes the expected runtime and maximizes the overall precision the choice of abstract domain is crucial. Here we briefly review common choices of abstract domains for neural network verification and how they are suited to our problem. Geometrically these domains can be thought of as a convex abstraction of the set of vectors representing reachable values at each step of the neural network. We say that an abstraction a_1 is more precise than another abstraction a_2, if and only if $a_1 \subseteq a_2$, i.e, all points in a_1 occur in a_2. Similarly, we say that a domain is more precise than another, if it enables the use of more precise abstractions.

The Box (or Interval) domain [16, 34, 35] abstracts sets in d dimensions as

$$B = \{\boldsymbol{a} + \text{diag}(\boldsymbol{d})\boldsymbol{e} \mid \boldsymbol{e} \in [-1, 1]^d\},$$

parametrized by a center $\boldsymbol{a} \in \mathbb{R}^d$ and width $\boldsymbol{d} \in \mathbb{R}^d_{\geq 0}$.

The Zonotope domain [15, 16, 34, 36, 37] uses relaxations Z of the form

$$Z = \{\boldsymbol{a} + \boldsymbol{A}\boldsymbol{e} \mid \boldsymbol{e} \in [-1, 1]^p\}, \tag{3}$$

parametrized with $\boldsymbol{a} \in \mathbb{R}^d$ and $\boldsymbol{A} \in \mathbb{R}^{d \times p}$.

A third common choice are domains utilizing (restricted) convex Polyhedra P [19, 20, 38]. For our discussion here we consider P to be in the DeepPoly (DP) encoding proposed by by Singh et al. [19] (which is equivalent to that of Zhang et al. [20]).

While not a formal hierarchy, generally Boxes are less precise than Zonotopes or Polyhedra, which permits the certification of more specifications.

		T			
		B	Z	$\alpha(Z)$	P
S	B	✓	✗	✓	(✓)
	Z	✓	✗	✓	✗
	P	✓	✗	✓	(✓)

Table 2: Feasibility of $S \subseteq T$ for Box B, Zonotope Z (with order reduction) and DP Polyhedra P.

For efficient proof sharing, we require a fast inclusion check $S \subseteq T$, which is challenging in our context due to the high dimensionality d. While we point the interested reader to [39] for a detailed discussion, we summarize the key results in Table 2. There, ✓ denotes feasibility, i.e. low polynomial runtime (usually $2d$ comparisons, sometimes with an additional matrix multiplication), ✗ denotes infeasibility, e.g. exponential run time. If T is a Box all checks are simple as it suffices to compute the outer bounding box of S and compare the $2d$ constraints. If T is a DP Polyhedra these checks require a linear program (LP) to be solved. While the size of this LP permits a low theoretical time complexity (in case S is a Box or DP Polyhedra), in practice we consider calling an LP solver too expensive. For Zonotopes these checks are generally infeasible, as they require enumeration of the faces or corners, which is computationally expensive for large d and p. While Zonotopes can be encoded as Polyhedra (but not necessarily DP Polyhedra), the same LP inclusion check as for P could be used, however, the resulting LP would require exponentially many variables due

to the previously mentioned enumeration. However, by placing constraints on the matrix $\boldsymbol{A}$ in Eq. (3) these inclusion checks can be performed efficiently. The mapping of a Zonotope to such a restricted Zonotope is called order reduction via outer-approximation [39, 40].

In particular, for a Zonotope Z we consider the order reduction α_{Box} to its bounding box (where $\boldsymbol{A}$ is diagonal) and note that other choices of α are possible (e.g. the reduction to affine transformations of a hyperbox).

For a general Zonotope Z its tight bounding box $Z' = \alpha_{\text{Box}}(Z)$ can be easily obtained. The center of Z' is $\boldsymbol{a}$, the center of Z. The width $\boldsymbol{d} \in \mathbb{R}^d_{\geq 0}$ is given as $\boldsymbol{d}_i = \sum_{j=1}^{p} |A_{i,j}|$. Z' represented as either a Box or a Zonotope (with $\boldsymbol{A} = \text{diag}(\boldsymbol{d})$). To check $S \subseteq Z'$ for a general Zontope S it suffices to check $\alpha_{\text{Box}}(S) \subseteq Z'$ which reduces to the simple inclusion check for boxes: To check $B_1 \subseteq B_2$ for two boxes, we can simply check the inclusion of the lower and upper bound in each dimension:

$$\bigwedge_{i=1}^{d} a_i^1 \pm d_i^1 \subseteq a_i^2 \pm d_i^2 = \bigwedge_{i=1}^{d} a_i^2 - d_i^2 \leq a_i^1 - d_i^1 \wedge a_i^1 + d_i^1 \leq a_i^2 + d_i^2. \tag{4}$$

Based on this discussion we will use the Zonotope domain to represent all abstractions, and use verifiers $V_S = V_T$ that propagate these zonotopes using the so-called DeepZ transformers [37]. To permit efficient inclusion checks we apply α_{Box} on the resulting zonotopes to obtain the Box templates T, which we treat as a special case of Zonotopes.

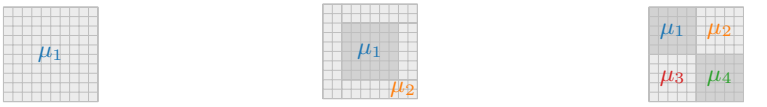
4.2 Template Generation

We now discuss instantiations for GENERATE_TEMPLATES in Algorithm 1. Building on the idea of proof subsumption, we know that abstractions for some specification contain abstractions for other specifications, we relax the Problem 2 in order to create m templates T_j from intermediate abstractions $N_{1:k}(\hat{\mathcal{I}}_i(\boldsymbol{x}))$ for some $\hat{\mathcal{I}}_1, \ldots, \hat{\mathcal{I}}_m$. $\hat{\mathcal{I}}_j$ are not necessarily directly related to $\mathcal{I}_1, \ldots \mathcal{I}_r$.

Then, for a chosen layer k, input $\boldsymbol{x}$, number of templates m and verifiers V_S and V_T we optimize

$$\begin{aligned} &\underset{\hat{\mathcal{I}}_1,\ldots,\hat{\mathcal{I}}_m}{\arg\max} \sum_{i=1}^{r} \left[\bigvee_{j=1}^{m} V_S(\mathcal{I}_i(\boldsymbol{x}), N_{1:k}) \subseteq T_j \right] \\ &\text{where } T_j = \alpha_{\text{Box}}(V_T(\hat{\mathcal{I}}_j(\boldsymbol{x}), N_{1:k})) \\ &\text{s.t. } V_T(T_j, N_{k+1:L}) \vdash \psi \text{ for } j \in 1, \ldots, m. \end{aligned} \tag{5}$$

This problem is solved by selecting $\hat{\mathcal{I}}_1, \ldots, \hat{\mathcal{I}}_m$ such that the resulting templates cover as much as possible of the given abstractions. To this end, we will next discuss application-specific parametric $\hat{\mathcal{I}}_i$ and solve Eq. (5) by optimizing over their parameters.



(a) ℓ_∞ (b) Center + Border (c) 2x2 Grid

Fig. 5: Example template splits μ on a 10×10 pixel grid.

4.3 Robustness to Adversarial Patches

We now instantiate the above scheme in order to verify the robustness of image classifiers against adversarial patches [28]. Consider an attacker that is allowed to arbitrarily change any $p \times p$ patch of the image, as showcased earlier in Fig. 2. For such a patch over pixel positions $([i, i+p-1] \times [j, j+p-1])$, the corresponding perturbation is

$$\mathcal{I}^{i,j}_{p\times p}(\boldsymbol{x}) := \{\boldsymbol{z} \in [0,1]^{h\times w} \mid \boldsymbol{z}_{\pi^C_{i,j}} = \boldsymbol{x}_{\pi^C_{i,j}}\}$$
$$\text{where } \pi_{i,j} = \left\{(k,l) \mid \begin{smallmatrix} k\in i,\ldots,i+p-1 \\ l\in j,\ldots,j+p-1 \end{smallmatrix}\right\}$$

where h and w denote the height and width of the input $\boldsymbol{x}$. Here $\pi_{i,j}$ denotes the parts of the image affected by the patch, and $\pi^C_{i,j}$ its complement, i.e., the unaffected part of the image. To prove robustness for an arbitrarily placed $p \times p$ patch, however, one must consider the perturbation set $\mathcal{I}_{p\times p}(\boldsymbol{x}) := \cup_{i,j}\mathcal{I}^{i,j}_{p\times p}(\boldsymbol{x})$.

To prove robustness for $\mathcal{I}_{p\times p}$, Chiang et al. [28] separately verifies $\mathcal{I}^{i,j}_{p\times p}(\boldsymbol{x})$ for all $i \in \{1,\ldots,h-p+1\}, j \in \{1,\ldots,w-p+1\}$. For example, with $p = 2$ and a 28×28 MNIST image, this approach requires 729 individual proofs. Because the different proofs for $\mathcal{I}_{p\times p}$ share similarities, one can apply proof sharing. We can utilize Algorithm 1 and check $\wedge_i v_i$ at the end to speed up this process. For template generation, we solve Eq. (5) for m templates with an input perturbation $\hat{\mathcal{I}}_i$ per template.

We empirically found that (recall Table 1) setting $\hat{\mathcal{I}}_i$ to be an ℓ_∞ region $\mathcal{I}_{\epsilon_i}$ to work particularly well to capture a majority of patch perturbations $\mathcal{I}^{i,j}_{p\times p}$ at intermediate layers. In particular we found that setting ϵ_i to the maximally verifiable value for this input to work particularly well.

To further improve upon this, use m template perturbations of the form

$$\hat{\mathcal{I}}_i(\boldsymbol{x}) := \{\boldsymbol{z} \mid \|\boldsymbol{x}_{\mu_i} - \boldsymbol{z}_{\mu_i}\|_\infty \leq \epsilon_i \wedge \boldsymbol{x}_{\mu^C_i} = \boldsymbol{z}_{\mu^C_i}\},$$

where μ_i denotes a subset of pixels of the input image and μ^C_i its complement. In particular, we consider $\mu_1,\ldots,\mu_m$, such that they partition the set of pixels in the image (e.g., in Fig. 5).

As noted earlier, this generation procedure needs to be fast, yet obtain $\mathcal{T}$ to which many abstractions match in order to obtain speedups. Thus, we consider small m, and fixed patterns $\mu_1,\ldots,\mu_m$. For each $\hat{\mathcal{I}}_i$ we aim to find the largest ϵ_i in order to maximize the number of matches, which can still be verified. Note

Algorithm 2: Online Template Generation for Patch Specifications

Input: $\boldsymbol{x}, N, \mu_1, \ldots, \mu_m, k_1, \ldots k_n, \psi, V_T$
Result: $\mathcal{T}^{k_1}, \ldots, \mathcal{T}^{k_n}$

1 $\mathcal{T}^{k_1}, \ldots \mathcal{T}^{k_n} \leftarrow \{\}, \ldots, \{\}$
2 **for** $i \leftarrow 1$ **to** m **do**
3 $\quad \hat{\mathcal{I}}_i(\boldsymbol{x}, \epsilon) := \{\boldsymbol{z} \mid \|\boldsymbol{x}_{\mu_i} - \boldsymbol{z}_{\mu_i}\| \leq \epsilon \wedge \boldsymbol{x}_{\mu_i^C} = \boldsymbol{z}_{\mu_i^C}\}$
4 $\quad f(\epsilon) := V_T(\hat{\mathcal{I}}_i(\boldsymbol{x}, \epsilon), N) \vdash \psi$
5 $\quad \epsilon_i \leftarrow \text{bin\_search}(\epsilon, f(\epsilon))$
6 $\quad$ **for** $j \leftarrow 1$ **to** n **do**
7 $\quad\quad T_{k_j} \leftarrow \alpha_{\text{Box}}(V_T(\hat{\mathcal{I}}_i(\boldsymbol{x}, \epsilon_i), N_{1:k_j}))$
8 $\quad\quad g(\beta) := V_T(\beta T_{k_j}, N_{k_j+1:L}) \vdash \psi$
9 $\quad\quad \beta_{k_j} \leftarrow \text{bin\_search}(\beta, g(\beta))$
10 $\quad\quad \mathcal{T}^{k_j} \leftarrow \mathcal{T}^{k_j} \cup \{T(\beta_{k_j})\}$
11 $\quad$ **end**
12 **end**
13 **return** $\mathcal{T}^{k_1}, \ldots, \mathcal{T}^{k_n}$

that for $m = 1$ this is equivalent to the ℓ_∞ input perturbation $\mathcal{I}_\epsilon$ with the maximal verifiable ϵ for the given image.

Concretely, we can perform binary search over ϵ_i in order find a large ϵ_i, still satisfying $N_{k+1:L}(\alpha_{\text{Box}}(N_{1:k}(\hat{\mathcal{I}}_i))) \vdash \psi$.

Templates at multiple layers This approach can be slightly extended, yielding templates at multiple layers without a large increase in computational cost.

With templates at layers k and k' (with $k \leq k'$) we first try to match the propagated shape against templates at layer k and if this fails, we propagate it further to layer k' and again try to match the proof. In Algorithm 1, this means repeating the block from line 4 to line 10 for each value of k before going on to the check on line 11.

The full template generation procedure is given in Algorithm 2. First, we perform a binary search over ϵ_i (Line 5). To find the largest ϵ_i, for which the specification is verifiable. Formally, verification with our chosen DeepZ Zonotopes is not monotonous in ϵ_i due to the transformers used for non-linearities (e.g., ReLU). This renders the application of binary search a best-effort approximation. Yet, as we don't require a formal maximum but rather aim to solve a surrogate for Problem 2, this still works well in practice. Further, as the lower bound of the binary search needs to be verified when it returns, we know that the found ϵ_i yields a template verifiable by V_T. Then for each layer k_j at which we are creating templates we obtain a box T_{k_j} from the Zonotope. As this T_{k_j} may not verifiable, due to the imprecision added in α_{Box}, we then perform another binary search for the largest scaling factor β, which is applied to the matrix $\boldsymbol{A}$ in Eq. (3). For a zonotope T we denote this operation as βT.

While we don't utilize verification using box propagation in this paper, we note that for it α_{Box} can be replaced by the identity and the second search over β be skipped.

4.4 Geometric Robustness

Geometric robustness verification [19, 25, 29, 41–43] aims to verify robustness to geometric transformations such as image rotations or image translations. These transformations typically include an interpolation operation. We demonstrate this on the example of a rotation R_γ by $\gamma \in \Gamma$ degrees for an interval Γ (e.g., $\gamma \in [-5, 5]$), for which we consider the perturbation $\mathcal{I}_\Gamma(\boldsymbol{x}) := \{R_\gamma(\boldsymbol{x}) \mid \gamma \in \Gamma\}$. We note that, unlike ℓ_∞ and patch verification, the input regions for geometric transformations are non-linear and have no closed-form solutions. Thus, an overapproximation of the input region must be obtained [29]. For large Γ, the approximate input region $\mathcal{I}_\Gamma(\boldsymbol{x})$, can be too coarse resulting in imprecise verification. Hence, in order to assert ψ on $\mathcal{I}_\Gamma$, existing state-of-the-art approaches, e.g., Balunovic et al. [29], split Γ into r smaller ranges $\Gamma_1, \ldots, \Gamma_r$ and then verify $(\mathcal{I}_{\Gamma_i}, \psi)$ for $i \in 1, \ldots, r$. These smaller perturbations share similarities facilitating proof sharing. We instantiate our approach similar to §4.3. A key difference to §4.3 is that while there are $\boldsymbol{x} \in \mathcal{I}^{i,j}_{p\times p}(\boldsymbol{x})$ for all i, j, here in general $\boldsymbol{x} \notin \mathcal{I}_{\Gamma_i}(\boldsymbol{x})$ for most i. Therefore, the individual perturbations $\mathcal{I}_i(\boldsymbol{x})$ do not overlap. To account for this, we consider m templates and split Γ into m equally sized chunks (unrelated to the r splits) obtaining the angles $\gamma_1, \ldots, \gamma_m$ at the center of each chunk. For m templates we then consider the perturbations $\hat{\mathcal{I}}_i := \mathcal{I}_{\epsilon_i}(R_{\gamma_i}(\boldsymbol{x}))$, denoting the ℓ_∞ perturbation of size ϵ_i around the γ_i degree rotated $\boldsymbol{x}$. To find the template we employ a procedure analogous to Algorithm 2.

4.5 Requirements for Proof Sharing

Here we discuss when proof sharing can be expected to be successful and which templates work well. To simplify things we discuss simple per-dimension box bounds here. However, similar arguments can be made for more complex relational abstractions such as Zonotopes or Polyhedra.

In order for an abstraction S to match to a template T the inclusion check in Eq. (4) needs to show containment for each dimension. For a particular dimension i this can occur in two ways: (i) when both S and T are just a point in that dimension and these points coincide, e.g., $a_i^S = a_i^T$, or (ii) when $a_i^S \pm d_i^S \subseteq a_i^T \pm d_i^T$. While particularly in ReLU networks, the first case can occur sometimes after a ReLU layer sets values to zero, it is rare and we focus our analysis on the second case. In this case the width of T in that dimension d_i^T must be sufficient to cover S. Ignoring case (i) and letting $\text{supp}(T)$ denote the dimensions in which $d_i^T > 0$, we can pose that $\text{supp}(S) \subseteq \text{supp}(T)$ as a necessary condition for inclusion. While it is in general hard to argue about the magnitudes of these values, this approach still provides an intuition. When starting from input specifications $\text{supp}(\mathcal{I}) \not\subseteq \text{supp}(\hat{\mathcal{I}})$, $\text{supp}(S) \subseteq \text{supp}(T)$ can only occur if during propagation

through the neural network $N_{1:k}$ the mass in $\text{supp}(\hat{\mathcal{I}})$ can "spread out" sufficiently to cover $\text{supp}(S)$. In the fully connected neural networks that we discuss here, the matrices of linear layers provide this possibility. However, in networks that only read part of the input at a time such as recurrent neural networks, or convolutional neural networks in which only locally neighboring inputs feed into the respective output in the next layer, these connections do not necessarily exist. This makes proof sharing hard until layers later in the neural network, that regionally or globally pool information. As this increases the depth of the layer k at which proof transfer can be applied, this also decreases the potential speed-up of proof transfer. Carefully choosing the template specifications can help to circumvent this.

5 Experimental Evaluation

We now experimentally evaluate the effectiveness of our algorithms from §4. We consider the verification of robustness to adversarial patch attacks and geometric transformations in §5.1 and §5.2, respectively.

We define specifications on 100 images each from the MNIST [33] and the CIFAR dataset [44]. We use Zonotope propagation as the baseline verifier as well as for V_S and V_T [37]. For both datasets we consider a 7 layer neural network with 200 neurons per layer, 7x200 as well as 9x500 which has 9 layers of 500 neurons. We provide further details in App. A. We implemented our algorithms in PyTorch [45] and ran our experiments on Ubuntu 18.04 with an Intel Core i9-9900K CPU and 64 GB RAM. For all timing results we provide the mean over three runs and their standard deviation in App. B.

5.1 Robustness against adversarial patches

For the MNIST dataset containing 28×28 images, as outlined in §4.3, in order to verify inputs to be robust against 2×2 patch perturbations, 729 individual perturbations must be verified. Only if all of these are verified, the overall property can be verified for a given image. Similarly, for CIFAR, containing 32×32 color images, there are 961 perturbations to be verified. (The patch is applied over all color channels.)

We now investigate the two main parameters of Algorithm 2: the masks $\mu_1, \ldots, \mu_m$ and the layers $k_1, \ldots k_n$.

We first study the impact of the layer k used for creating the template. To this end we consider the 7x200 networks, use $m = 1$ (covering the whole image; equivalent to $\mathcal{I}_\epsilon$). Table 3 shows the corresponding template matching rates, the overall percentage of individual patches that can be verified "patches verif.". (The overall percentage of images for which $\mathcal{I}_{2\times2}$ is true is reported as "verf." in Table 6.) Table 4 shows the corresponding verification times (including the template generation). We observe that many template matches can already be made at the second or third layer. We find that in terms of verification time, creating templates simultaneously at the second and third layer, using the

Table 3: Rate of $\mathcal{I}^{i,j}_{2\times2}$ matched to templates p for $\mathcal{I}_{2\times2}$ patch verification for different combinations of template layers k, 7x200 networks,using $m = 1$ template.

	template at layer							patch
	k	1	2	3	4	5	6	7	verif. [%]
MNIST		18.6	85.6	94.1	95.2	95.5	95.7	95.7	97.0
CIFAR		0.1	27.1	33.7	34.4	34.2	34.2	34.3	42.2

Table 4: Average verification time in seconds per image for $\mathcal{I}_{2\times2}$ patchs for different combinations of template layers k, 7x200 networks,using $m = 1$ template.

		Proof Sharing, template layer k							
	Baseline	1	2	3	4	1+3	2+3	2+4	2+3+4
MNIST	2.10	1.94	1.15	1.22	1.41	1.27	**1.09**	1.10	1.14
CIFAR	3.27	2.98	2.53	2.32	2.47	2.35	2.49	**2.42**	2.55

method outlined in §4.3, works best overall. Thus, for further experiments we use these two layers.

Next, we investigate the impact of the pixel masks $\mu_1, \ldots, \mu_m$. To this end we consider three different settings, as showcased in Fig. 5 earlier: (i) the full image (ℓ_∞-mask as before; $m = 1$), (ii) "center + border" ($m = 2$), where we consider the 6×6 center pixel as one group and all others as another, and (iii) the 2×2 grid ($m = 4$) where we split the image into equally sized quarters.

As we can see in Table 5, for higher m more patches can be matched to the templates, indicating that our optimization procedure is a good approximation to Problem 2. However, as $m > 1$ methods are only marginally better than the ℓ_∞-mask, verification time gained by these additional matches does not offset additional template creation time. Thus, $m = 1$ results in a better trade-off. Based on this investigation we now, in Table 6, evaluate all networks and datasets using $m = 1$ and template generation at layers 2 and 3. In all cases, we obtain a speed up between 1.2 to 2×.

Optimal Speedup Finally, we want to determine the maximal possible speedup with proof sharing and see how much of this potential is realized by our method. To this end we investigate the same setting as in Table 3. We let t^{BL} and t^{PS} denote the runtime of the of the base verifier on of verification with proof sharing respectively. Similar to the discussion in §4 we can break down t^{PS} into t_T (template generation time), t_S (time to propagate one input to layer k), $t_\subseteq$ (time to perform template matching) and t_ψ (time to verify S if no match). Table 7 shows different ratios of these quantities. For all we assume a perfect matching rate at layer k and calculate the achievable speedup for patch verification on MNIST. Comparing the optimal and realized results, we see that at layers 3

Table 5: $\mathcal{I}_{2\times2}$ patch verification with templates at the 2nd & 3rd layer of the 7x200 networks for different masks.

Method/Mask	m	patch matched [%]	t [s]
Baseline	-	-	2.14
L-infinity	1	94.1	**1.11**
Center + Border	2	94.6	1.41
2x2 Grid	4	**95.0**	3.49

Table 6: $\mathcal{I}_{2\times2}$ patch verification with templates generated on the second and third layer using the ℓ_∞-mask. Verification times are given for the baseline t^{BL} and for applying proof sharing t^{PS} in seconds per image.

Dataset	Net	verif. acc. [%]	t^{BL}	t^{PS}	patch mat. [%]	patch verif. [%]
MNIST	7x200	81.0	2.10	**1.10**	94.1	97.0
	9x500	96.0	2.70	**1.32**	93.0	95.3
CIFAR	7x200	29.0	3.28	**2.45**	33.7	42.2
	9x500	28.0	5.48	**4.48**	34.2	60.9

Table 7: Speedups achievable in the setting of Table 3. t^{BL} denotes time by baseline.

		speedup at layer k			
Layer k		1	2	3	4
realized	t^{PS}/t^{BL}	1.08	1.83	1.72	1.49
optimal	$(t_T + rt_S + rt_\subseteq)/t^{BL}$	3.75	2.51	1.92	1.56
optimal, no match.	$(t_T + rt_S)/t^{BL}$	4.02	2.68	2.01	1.62
optimal, no gen., no match.	rt_S/t^{BL}	4.57	2.90	2.13	1.69

and 4 our template generation algorithm, despite only approximately solving Problem 2 achieves near optimal speedup. By removing the time for template matching and template generation we can see that, at deeper layers, further speedups for the matching and generation algorithm achieve only diminishing returns.

5.2 Robustness against geometric perturbations

For the verification of geometric perturbations we take 100 images from the MNIST dataset and the 7x200 neural network from §5.1.

In Table 8, we consider an input specification with $\pm2°$ rotation followed by a $\pm10\%$ contrast and $\pm1\%$ brightness change. To verify this region, similar to Balunovic et al. [29], we split the rotation into r regions, which we represent

Table 8: ±2° rotation, ±10% contrast and ±1% brightness change split into r perturbations on 100 MNIST images. Verification rate, rate of splits matched and verified along with the run time of Zonotope t^{BL} and proof sharing t^{PS}.

r	verif. [%]	splits verif. [%]	splits matched [%]	t^{BL}	t^{PT}
4	73.0	87.3	73.1	3.06	**1.87**
6	91.0	94.8	91.0	9.29	**3.81**
8	93.0	95.9	94.2	20.64	**7.48**
10	95.0	96.5	94.9	38.50	**13.38**

Table 9: ±40° rotation split into 200 perturbations evaluated on 100 MNIST images. The verification rate is just 15 %, but 82.1 % of individual splits can be verified.

Method	m	splits matched [%]	t [s]
Baseline	-	-	11.79
Proof Sharing	1	38.0	9.15
	2	41.1	9.21
	3	58.5	**8.34**

as Box abstractions. As before we use Zonotopes, but initialize them to this Box specification. Here we use $m = 1$, a single template, with optimized ϵ. We observe that as we increase r, verification rate increases, but also the speed ups between 1.6 to 2.9× obtained from proof sharing.

Finally, we investigate the impact of the number of templates m. For this we consider a ±40° rotation generated input region with $r = 200$. In Table 9 we evaluate this for m templates obtained from the ℓ_∞ input perturbation around m equally spaced rotations, where we apply binary search to find ϵ. Again we observe that while $m > 1$ allows more templates matches. However, here the relative increase is much larger, thus making $m = 3$ faster than $m = 1$.

5.3 Discussion

We have shown that proof sharing can achieve speed-ups over conventional execution. While the speed-up analysis in Table 7 put a ceiling on what is achievable in this particular setting, we are optimistic that proof sharing can be an important tool for neural network robustness analysis. In particular as the size of certifiable neural networks continues to grow the potential for gains via proof sharing is equally growing. Furthermore, the idea of proof effort reuse may enable the use of larger disjunctive specifications such as the patch or geometric examples considered here.

6 Related Work

Here, we briefly discuss conceptually related work: Ashok et al. [46] simplify neural networks before applying verification algorithms to achieve speed-ups. This simplification is conducted in a way that the obtained proof can be adapted to the original neural network. Similarly, Zhong et al. [47] replace blocks neural networks with summaries in order to speed up verification. Paulsen et al. [48, 49] analyze the difference between two closely related neural networks, exploring their structural similarity in the analysis. Cheng and Yan [50] reuse previous analysis results after a neural network is slightly modified, e.g. fine-tuned. In a temporal setting, Wei and Liu [51] utilize verification results from previous time-steps to seed the verification at the next time-step, allowing faster adaptation to changes in inputs and the neural network.

In contrast to these works, we do not modify the neural network, but achieve speed-ups rather by only considering the convex relaxations obtained in the proofs.

7 Conclusion

We introduced the novel concept of proof sharing and proof effort reuse in the context of neural network verification. We showed how to instantiate this concept, achieving speedups of up to 2 to 3 x for patch verification and geometric verification. We believe that the ideas introduced in this work can serve as a solid foundation for exploring methods that effectively share proofs in neural network verification.

Bibliography

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, 2012.

[2] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, 2017.

[3] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *Proc. of ICLR*, 2018.

[4] T. B. B. an, "Adversarial patch," *ArXiv preprint*, vol. abs/1712.09665, 2017.

[5] J. M. Wing, "Trustworthy ai," *Commun. ACM*, vol. 64, no. 10, p. 64–71, 2021.

[6] G. Katz, C. W. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient SMT solver for verifying deep neural networks," in *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*, vol. 10426, 2017.

[7] R. Ehlers, "Formal verification of piece-wise linear feed-forward neural networks," in *Automated Technology for Verification and Analysis - 15th International Symposium, ATVA 2017, Pune, India, October 3-6, 2017, Proceedings*, vol. 10482, 2017.

[8] R. Bunel, I. Turkaslan, P. H. S. Torr, P. Kohli, and P. K. Mudigonda, "A unified view of piecewise linear neural network verification," in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, 2018.

[9] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljić, D. L. Dill, M. J. Kochenderfer, and C. Barrett, "The marabou framework for verification and analysis of deep neural networks," in *Proc. Computer Aided Verification (CAV)*, 2019.

[10] V. Tjeng, K. Y. Xiao, and R. Tedrake, "Evaluating robustness of neural networks with mixed integer programming," in *Proc. of ICLR*, 2019.

[11] R. Anderson, J. Huchette, C. Tjandraatmadja, and J. P. Vielma, "Strong mixed-integer programming formulations for trained neural networks," in *Proc. Integer Programming and Combinatorial Optimization (IPCO)*, vol. 11480, 2019.

[12] J. Lu and M. P. Kumar, "Neural network branching for neural network verification," in *Proc. of ICLR*, 2020.

[13] E. Botoeva, P. Kouvaros, J. Kronqvist, A. Lomuscio, and R. Misener, "Efficient verification of relu-based neural networks via dependency analysis,"

in *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, 2020.

[14] K. Dvijotham, R. Stanforth, S. Gowal, T. A. Mann, and P. Kohli, "A dual approach to scalable verification of deep networks," in *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI 2018, Monterey, California, USA, August 6-10, 2018*, 2018.

[15] E. Wong and J. Z. Kolter, "Provable defenses against adversarial examples via the convex outer adversarial polytope," in *Proc. of ICML*, vol. 80, 2018.

[16] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. T. Vechev, "AI2: safety and robustness certification of neural networks with abstract interpretation," in *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, 2018.

[17] G. Singh, T. Gehr, M. Mirman, M. Püschel, and M. T. Vechev, "Fast and effective robustness certification," in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, 2018.

[18] A. Raghunathan, J. Steinhardt, and P. Liang, "Semidefinite relaxations for certifying robustness to adversarial examples," in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, 2018.

[19] G. Singh, T. Gehr, M. Püschel, and M. T. Vechev, "An abstract domain for certifying neural networks," *PACMPL*, vol. 3, no. POPL, 2019.

[20] H. Zhang, T. Weng, P. Chen, C. Hsieh, and L. Daniel, "Efficient neural network robustness certification with general activation functions," in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, 2018.

[21] K. D. Dvijotham, R. Stanforth, S. Gowal, C. Qin, S. De, and P. Kohli, "Efficient neural network verification with exactness characterization," in *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2019, Tel Aviv, Israel, July 22-25, 2019*, vol. 115, 2019.

[22] T. Weng, H. Zhang, H. Chen, Z. Song, C. Hsieh, L. Daniel, D. S. Boning, and I. S. Dhillon, "Towards fast computation of certified robustness for relu networks," in *Proc. of ICML*, vol. 80, 2018.

[23] H. Salman, G. Yang, H. Zhang, C. Hsieh, and P. Zhang, "A convex relaxation barrier to tight robustness verification of neural networks," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, 2019.

[24] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Efficient formal safety analysis of neural networks," in *Advances in Neural Information Pro-*

cessing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada, 2018.

[25] J. Mohapatra, T. Weng, P. Chen, S. Liu, and L. Daniel, "Towards verifying robustness of neural networks against A family of semantic perturbations," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, 2020.

[26] A. Ruoss, M. Balunovic, M. Fischer, and M. T. Vechev, "Learning certified individually fair representations," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[27] C. Urban, M. Christakis, V. Wüstholz, and F. Zhang, "Perfectly parallel fairness certification of neural networks," *Proc. ACM Program. Lang.*, vol. 4, no. OOPSLA, pp. 185:1–185:30, 2020. [Online]. Available: https://doi.org/10.1145/3428253

[28] P. Chiang, R. Ni, A. Abdelkader, C. Zhu, C. Studer, and T. Goldstein, "Certified defenses for adversarial patches," in *Proc. of ICLR*, 2020.

[29] M. Balunovic, M. Baader, G. Singh, T. Gehr, and M. T. Vechev, "Certifying geometric robustness of neural networks," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, 2019.

[30] P. Cousot and R. Cousot, "Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints," in *Proc. Symposium on Principles of Programming Languages (POPL)*, 1977.

[31] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *Proc. of ICLR*, 2014.

[32] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Srndic, P. Laskov, G. Giacinto, and F. Roli, "Evasion attacks against machine learning at test time," in *ECML/PKDD (3)*, vol. 8190, 2013.

[33] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network," in *Advances in Neural Information Processing Systems 2, [NIPS Conference, Denver, Colorado, USA, November 27-30, 1989]*, 1989.

[34] M. Mirman, T. Gehr, and M. T. Vechev, "Differentiable abstract interpretation for provably robust neural networks," in *Proc. of ICML*, vol. 80, 2018.

[35] S. G. an, "On the effectiveness of interval bound propagation for training verifiabl," *ArXiv preprint*, vol. abs/1810.12715, 2018.

[36] E. Goubault and S. Putot, "A zonotopic framework for functional abstractions," *Formal Methods Syst. Des.*, vol. 47, no. 3, 2015.

[37] G. Singh, T. Gehr, M. Mirman, M. Püschel, and M. T. Vechev, "Fast and effective robustness certification," in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, 2018.

[38] P. Cousot and N. Halbwachs, "Automatic discovery of linear restraints among variables of a program," in *Proc. Symposium on Principles of Programming Languages (POPL)*, 1978.

[39] S. Sadraddini and R. Tedrake, "Linear encodings for polytope containment problems," in *58th IEEE Conference on Decision and Control, CDC 2019, Nice, France, December 11-13, 2019*, 2019.

[40] A. Kopetzki, B. Schürmann, and M. Althoff, "Methods for order reduction of zonotopes," in *56th IEEE Annual Conference on Decision and Control, CDC 2017, Melbourne, Australia, December 12-15, 2017*, 2017.

[41] K. Pei, Y. Cao, J. Yang, and S. Jana, "Towards practical verification of machine learning: Th," *ArXiv preprint*, vol. abs/1712.01785, 2017.

[42] M. Fischer, M. Baader, and M. T. Vechev, "Certified defense to image transformations via randomized smoothing," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[43] L. Li, M. Weber, X. Xu, L. Rimanic, B. Kailkhura, T. Xie, C. Zhang, and B. Li, "Tss: Transformation-specific smoothing for robustness certification," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '21, New York, NY, USA, 2021.

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

[45] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, 2019.

[46] P. Ashok, V. Hashemi, J. Kretínský, and S. Mohr, "Deepabstract: Neural network abstraction for accelerating verification," in *Automated Technology for Verification and Analysis - 18th International Symposium, ATVA 2020, Hanoi, Vietnam, October 19-23, 2020, Proceedings*, vol. 12302, 2020.

[47] Y. Zhong, Q. Ta, T. Luo, F. Zhang, and S. Khoo, "Scalable and modular robustness analysis of deep neural networks," in *Programming Languages and Systems - 19th Asian Symposium, APLAS 2021, Chicago, IL, USA, October 17-18, 2021, Proceedings*, 2021.

[48] B. Paulsen, J. Wang, and C. Wang, "Reludiff: differential verification of deep neural networks," in *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*, 2020.

[49] B. Paulsen, J. Wang, J. Wang, and C. Wang, "NEURODIFF: scalable differential verification of neural networks using fine-grained approximation," in *35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020, Melbourne, Australia, September 21-25, 2020*, 2020.

[50] C. Cheng and R. Yan, "Continuous safety verification of neural networks," in *Design, Automation & Test in Europe Conference & Exhibition, DATE 2021, Grenoble, France, February 1-5, 2021*, 2021.

[51] T. Wei and C. Liu, "Online verification of deep neural networks under domain or weigh," *ArXiv preprint*, vol. abs/2106.12732, 2021.

[52] V. Roth, J. Laub, M. Kawanabe, and J. M. Buhmann, "Optimal cluster preserving embedding of nonmetric proximity data," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 25, no. 12, 2003.

[53] H.-D. Tran, D. M. Lopez, P. Musau, X. Yang, L. V. Nguyen, W. Xiang, and T. T. Johnson, "Star-based reachability analysis of deep neural networks," in *International Symposium on Formal Methods*. Springer, 2019.

[54] S. Bak and P. S. Duggirala, "Simulation-equivalent reachability of large linear systems with inputs," in *International Conference on Computer Aided Verification*. Springer, 2017.

Supplementary Material for Shared Certificates for Neural Network Verification

A Experimental Details

A.1 Details for §3.1

We use a feed foward neural network with five linear layers of size 100 and ReLU activations, trained with DiffAI [34]. For verification we utilize the box domain.

Fig. 3 uses a PCA fit to the data to project down to 2 dimensions. For the evaluation of $\mathcal{I}_\epsilon$ we use the whole test set for the subsumption check we used 1000 randomly sampled $\boldsymbol{x}$, for which $\mathcal{I}_\epsilon(\boldsymbol{x})$ is verifiable, and then sample 2 random (i, j) for this image. Inclusion checks are performed after ReLU activations.

A.2 Details for §5

The neural networks was trained using robust training [3] for patch attacks, as outlined in [28]. The 7x200 and 9x500 networks were trained for 600 epochs to achieve accuracies on MNIST [33]of 98.3% and 95.3% respectively. These same values for the CIFAR [44] dataset a 48.8% and 48.1% respectively.

B Additional Results

B.1 Standard deviations for Timings

In Tables 10–14 we report the standard deviation for the timings in §5.

Table 10: Extended version of Table 4. Average verification time in seconds per image for $\mathcal{I}_{2\times2}$ patches for different combinations of template layers k, 7x200 networks,using $m = 1$ template.

		template at layer			
	Baseline	1	2	3	4
MNIST	2.10 ± 0.01	1.94 ± 0.01	1.15 ± 0.04	1.22 ± 0.01	1.41 ± 0.01
CIFAR	3.27 ± 0.24	2.98 ± 0.31	2.53 ± 0.09	2.32 ± 0.01	2.47 ± 0.16

	template at layer			
	1+3	2+3	2+4	2+3+4
MNIST	1.27 ± 0.05	$\mathbf{1.09} \pm 0.01$	1.10 ± 0.04	1.14 ± 0.03
CIFAR	2.35 ± 0.02	2.49 ± 0.14	$\mathbf{2.42} \pm 0.04$	2.55 ± 0.21

Table 11: Extended version of Table 5. $\mathcal{I}_{2\times2}$ patch verification with templates at the 2nd & 3rd layer of the 7x200 networks for different masks.

Method/Mask	m	patch matched [%]	t [s]
Baseline	-	-	2.14 ± 0.10
L-infinity	1	94.1	$\mathbf{1.11} \pm 0.05$
Center + Border	2	94.6	1.41 ± 0.01
2x2 Grid	4	**95.0**	3.49 ± 0.06

Table 12: Extended version of Table 6. $\mathcal{I}_{2\times2}$ patch verification with templates generated on the second and third layer using the ℓ_∞-mask. Verification times are given for the baseline t^{BL} and for applying proof sharing t^{PS} in seconds per image.

Dataset	Net	verif. acc. [%]	t^{BL}	t^{PS}	patch mat. [%]	patch verif. [%]
MNIST	7x200	81.0	2.10 ± 0.01	$\mathbf{1.10} \pm 0.03$	94.1	97.0
	9x500	96.0	2.70 ± 0.01	$\mathbf{1.32} \pm 0.00$	93.0	95.3
CIFAR	7x200	29.0	3.28 ± 0.07	$\mathbf{2.45} \pm 0.06$	33.7	42.2
	9x500	28.0	5.48 ± 0.03	$\mathbf{4.48} \pm 0.03$	34.2	60.9

Table 13: Extended version of Table 8. $\pm2°$ rotation, $\pm10\%$ contrast and $\pm1\%$ brightness change split into r perturbations on 100 MNIST images. The verification rate, rate of splits matched and verified along with the run time of Zonotope t^{BL} and proof sharing t^{PS}.

r	verif. [%]	splits verif. [%]	splits mat. [%]	t^{BL}	t^{PS}
4	73.0	87.3	73.1	3.06 ± 0.01	$\mathbf{1.87} \pm 0.01$
6	91.0	94.8	91.0	9.29 ± 0.01	$\mathbf{3.81} \pm 0.01$
8	93.0	95.9	94.2	20.64 ± 0.05	$\mathbf{7.48} \pm 0.01$
10	95.0	96.5	94.9	38.50 ± 0.05	$\mathbf{13.38} \pm 0.02$

Table 14: Extended version of Table 9. $\pm40°$ rotation split into 200 perturbations evaluated on 100 MNIST. While the verification rate is just 15 %, but 82.1 % of individual splits can be verified.

Method	m	splits matched [%]	t [s]
Baseline	-	-	11.79 ± 0.05
Proof Sharing	1	38.0	9.15 ± 0.01
	2	41.1	9.21 ± 0.01
	3	58.5	$\mathbf{8.34} \pm 0.01$

C Offline Proof Sharing

In contrast to the online setting in the main paper, here we discuss an offline setting. Therefore, we consider a version of Problem 2. Instead of Eq. (1), we optimize

$$\max_{\mathcal{T}} \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[\bigvee_{T\in\mathcal{T}} N_{1:k}(\mathcal{I}(\boldsymbol{x})) \subseteq T\right] \quad \text{s.t. } \forall\, T \in \mathcal{T}.\, N_{k+1:L}(T) \vdash \psi, \tag{6}$$

now with a fixed $\mathcal{I}$ and but rather than a single fixed $\boldsymbol{x}$, we consider $\boldsymbol{x} \sim \mathcal{D}$ is drawn from the data distribution. As standard in machine learning, we assume that we have access to a large sample from $\mathcal{D}$, the training set T_{train}.

We create $\mathcal{T}$ offline from the training set, and then verify new inputs $\boldsymbol{x}$ by utilizing $\mathcal{T}$. An important challenge here is that we require the templates $\mathcal{T}$ to be general enough to generalize to unseen inputs.

C.1 Template Generation on Training Data

We first outline the template generation process in general and then show an instantiation for ℓ_∞-robustness verification in App. C.2.

Template Generation In order to optimize Eq. (6) under our constraints, we attempt to find the set of templates $\mathcal{T}$ with $|\mathcal{T}| \leq m$ that contains the most abstraction at layer k obtained from T_{train}. While this is generally computationally hard, we approximate this with a clustering-based approach. To this end, we first compute the abstractions $N_{1:k}(\mathcal{I}(\boldsymbol{x}))$ for $\boldsymbol{x} \in T_{\text{train}}$ and then cluster and merge them. Subsequently, we further merge the obtained templates until we obtain a set $\mathcal{T}$ of m templates. We formalize the template generation procedure in Algorithm 3 and showcase it in Fig. 6.

We first (line 1, Fig. 6a) compute the set $\mathcal{V}$ of abstraction at layer k, that can be verified. Then, we cluster the abstractions in $\mathcal{V}$ into n groups $\mathcal{G}_i$ of similar abstractions using the function CLUSTER_SHAPES (which we will instantiate in App. C.2), showcased by different colors in Fig. 6b. For each group $\mathcal{G}_i$, we compute its convex hull T via the join operator $\bigsqcup$. The join returns a (usually the tightest) expressible shape in the chosen abstraction that includes all its inputs. If we verify this T, then we add the tuple $(T, \mathcal{G}_i)$, the template along with all abstractions it covers, to the set $\mathcal{H}$, shown in Fig. 6c. As depicted in Fig. 6d, we attempt to merge pairs of these templates (line 12). To this end, we traverse them in order of their distance d. Here, we use the Euclidean distance between the centers of T_1 and T_2 for $d(T_1, T_2)$. In order to merge $(T_1, \mathcal{G}_1)$ and $(T_2, \mathcal{G}_2)$, we first compute the set of their contained shapes $\mathcal{G}' = \mathcal{G}_1 \cup \mathcal{G}_2$ (line 13) and then again compute the join $T' = \bigsqcup(\mathcal{G}')$ (line 14). If T' can be verified, we replace $(T_1, \mathcal{G}_1)$ and $(T_2, \mathcal{G}_2)$ by $(T', \mathcal{G}')$ in $\mathcal{H}$ (lines 15 to 21). This procedure is repeated until no templates can be merged. Finally, $\mathcal{T}$ is obtained as the set of the m templates with the most associated abstractions (e.g., the largest $|\mathcal{G}|$) in $\mathcal{H}$.

Algorithm 3: Template generation over T_{train}

Input: training set T_{train}, specification $\mathcal{I}$, verifiers V_S, V_T
Result: Set $\mathcal{T}$ of templates, $|\mathcal{T}| \leq m$

1 $\mathcal{V} \leftarrow \{V_S(\mathcal{I}(\boldsymbol{x}), N_{1:k}) \mid \boldsymbol{x} \in T_{\text{train}}, N(\mathcal{I}(\boldsymbol{x})) \vdash \psi\}$
2 $\mathcal{G}_1, \ldots, \mathcal{G}_n \leftarrow$ CLUSTER_SHAPES($\mathcal{V}$)
3 $\mathcal{H} \leftarrow \emptyset$
4 **for** $i \leftarrow 1$ **to** n **do**
5 $\quad T \leftarrow \bigsqcup_D(\mathcal{G}_i)$
6 $\quad$ **if** $V_T(T, N_{k+1:L}) \vdash \psi$ **then**
7 $\quad\quad \mathcal{H} \leftarrow \mathcal{H} \cup \{(T, \mathcal{G}_i)\}$
8 $\quad$ **end**
9 **end**
10 pairs $\leftarrow$ all pairs $(T_1, \mathcal{G}_1)$, $(T_2, \mathcal{G}_2)$ in $\mathcal{H}$
11 $Q \leftarrow$ priority queue over pairs, ordered by $d(T_1, T_2)$
12 **foreach** *pair* $(T_1, \mathcal{G}_1)$, $(T_1, \mathcal{G}_2)$ *in* Q **do**
13 $\quad \mathcal{G}' = \mathcal{G}_1 \cup \mathcal{G}_2$
14 $\quad T' = \bigsqcup_D(\mathcal{G}')$
15 $\quad$ **if** $(V_T(T', N_{k+1:L}) \vdash \psi$ **then**
16 $\quad\quad \mathcal{H} \leftarrow \mathcal{H} \setminus \{(T_1, \mathcal{G}_1), (T_2, \mathcal{G}_2)\}$
17 $\quad\quad$ remove elements containing $(T_1, \mathcal{G}_1)$ from Q
18 $\quad\quad$ remove elements containing $(T_2, \mathcal{G}_2)$ from Q
19 $\quad\quad$ add all pairs $(T, \mathcal{G})$, $(T', \mathcal{G}')$ for $(T, \mathcal{G}) \in \mathcal{H}$ to Q
20 $\quad\quad \mathcal{H} \leftarrow \mathcal{H} \cup \{(T', \mathcal{G}')\}$
21 $\quad$ **end**
22 **end**
23 $\mathcal{T} \leftarrow T$ for $(T, \mathcal{G}) \in \mathcal{H}$ for the m largest $|\mathcal{G}|$
24 **return** $\mathcal{T}$

C.2 Dataset templates for ℓ_∞ robustness

We now instantiate the template generation algorithm for ℓ_∞-robustness verification $\mathcal{I}_\epsilon$. As in §4, we propagate specifications as Zonotopes (via V_S) and represent templates as Boxes. For the verification of the templates (lines 6 and 15) we perform exact verification via Mixed-Integer Linear Programming (MILP) [10] via verifier V_T. The box-encoded templates can be directly verified by the exact verifier. We note that since exact verification is strictly more precise than Zonotope propagation, the use of templates can potentially allow for *higher* certification rates than directly employing Zonotope propagation. While we did not observe this experimentally, it presents an interesting target for further investigation.

We now briefly outline the properties of exact verification, as we require these in the following discussion. The framework from Tjeng et al. [10], proves classification to the correct label l by maximizing the error $e = \max_{i \neq l} \boldsymbol{n}_i - \boldsymbol{n}_t$ (asserting that $e < 0$), where $\boldsymbol{n}$ denotes the output of the neural network (e.g. its logits) over the considered input region. If no counterexample to that assertion can be found, it certifies the specification, else it returns a set of counterexamples

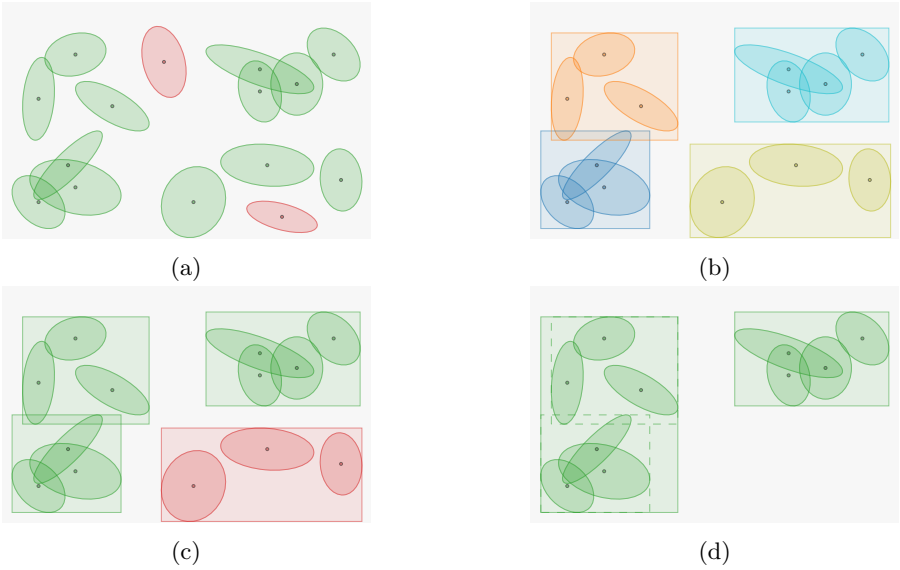


Fig. 6: Visualization of Algorithm 3. First, in (a) the abstractions at layer k for input regions in the training set are obtained, and restricted to the verifiable ones (green). These are then clustered and their convex hulls in domain D are obtained. (b) shows different clusters in different colors. The convex hulls are then verified (c), and restricted to the verifiable ones (green). Finally in (d), these regions are further merged, if possible, to obtain the set of templates.

$\{\boldsymbol{z}_{V,i}\}$ (concrete points in the input region), utilized later, for which this error is maximal. In both cases we can access value e of the error function.

Next, we instantiate CLUSTER_SHAPES with k-means clustering for which we provide a similarity matrix computed as follows: for each pair $P_1, P_2 \in \mathcal{V}$ we compute $B = P_1 \sqcup_B P_2$, where $\sqcup_B$ denotes their Box join, and use $\exp(e)$ as their distance, where $e \in \mathbb{R}$ is the error obtained from exact verification when attempting to verify ψ for B. To obtain a similarity matrix from these distances, we apply a constant shift embedding [52]. As invoking exact verification for each pair (P_1, P_2) is expensive, we only consider the t closest neighbors (in ℓ_2 distance between centers) and set all others to a maximal distance.

Half-space constraints To allow for larger templates T, e.g., those that allow to optimize Eq. (6) further by containing more abstractions, we extend the template representation from Boxes to Boxes with additional half-space constraints. The resulting convex polyhedra, formally called Stars [53, 54], allow for larger boxes while the half-space constraints cut-off potentially non-verifiable regions, yet, still allow efficient containment checks $S \subseteq T$.

Formally a Star B^S over a Box B is denoted as:

$$B^S(\boldsymbol{C}, \boldsymbol{c}) := \{\boldsymbol{z} \in \mathbb{R}^d \mid \boldsymbol{z} \in B \wedge C\boldsymbol{z} \leq \boldsymbol{c}\} \quad (7)$$
$$\text{with } \boldsymbol{C} \in \mathbb{R}^{c \times d}, \boldsymbol{c} \in \mathbb{R}^c.$$

Here each half-space constraint is described by a hyperplane parameterized by $\boldsymbol{C}_{i,\cdot}$ and $\boldsymbol{c}_i$.

The containment check $S \subseteq T^S$ for an abstraction S, e.g. a zonotope, and star T^S consists of: (i) a containment for the underlying box $S \subseteq T$, and (ii) checking if for each constraint $C_i \cdot z \leq c_i$, maximizing the linear expression $C_i \cdot z$ with respect to S yields an objective $\leq c_i$. For a zonotope S as given in Eq. (3), we have shown the computation of step (i) in App. C.2 and can efficiently check step (ii) via $\boldsymbol{C}\boldsymbol{a} + \sum_{j=1}^{p} |\boldsymbol{C}\boldsymbol{A}|_j \leq \boldsymbol{c}$.

A star encoded as in Eq. (7) can be directly verified using exact verification (MILP) by adding the half-space constraints as further LP constraints.

Obtaining half-space contraints In the template generation process we utilize Boxes as before. However, whenever we fail to verify a template (e.g., lines 6 and 15 in Algorithm 3), we attempt to add a half-space constraint. We repeat this up to n_{hs} times resulting in as many constraints. We leverage the exact verifier for obtaining half-space constraints. Recall, that it either verifies a region or provides a set of counterexamples $\{\boldsymbol{z}_{V,i}\}$. Since we only add additional half-space constraints, if the verification fails we utilize these counterexamples. In the following we assume a single $\boldsymbol{z}_V$, which in practice is usually the case, and derive a hyperplane that separates $\boldsymbol{z}_V$ from the abstraction we are trying to verify. If there are multiple $\boldsymbol{z}_{V,i}$, we iterate over them and perform the described procedure for each $\boldsymbol{z}_{V,i}$, that is not already cut by the hyperplane found for a previous counterexample. These hyperplanes directly yield the new constraints.

We showcase this in Fig. 7, where the green shaded area T shows the Box join over three abstraction $T = \bigsqcup(\{P_1, P_2, P_3\})$ The individual P_i, shown in blue, are zonotopes, that can be verified individually. .

The verification of the green area fails, with the counterexample $\boldsymbol{z}_V$ (red dot) shown in the top right corner. To find a hyperplane that separates $\boldsymbol{z}_V$ from the rest of T, we consider the line from the center $\boldsymbol{a}$ (green dot) of T to the point $\boldsymbol{z}_V$ and a hyperplane orthogonal to it (shown as the dashed line). Thus, adding a row i to the matrix $\boldsymbol{C}$ of the star: $\boldsymbol{C}_i = (\boldsymbol{z}_V - \boldsymbol{a})^T$. To find offset $\boldsymbol{h} = \lambda\boldsymbol{a} + (1-\lambda)\boldsymbol{z}_V$ (for $\lambda \in [0, 1]$) along this line to, we consider the value attained for $\boldsymbol{C}_i\boldsymbol{x}$ for $\boldsymbol{x}$ in the verified area (P_1, P_2, P_3):

$$c_p := max_{\substack{\boldsymbol{x} \in P \\ P \in \{P_1, P_2, P_3\}}} \boldsymbol{C}_i\boldsymbol{x}.$$

Then, the constant $\boldsymbol{c}_i$ of the new hyperplane is given by $\boldsymbol{c}_i = \kappa\boldsymbol{c}_p + (1-\kappa)\boldsymbol{c}_z$ for a hyper-parameter κ and $\boldsymbol{z}_z := \boldsymbol{C}_i\boldsymbol{z}_V$.

A high κ puts the hyperplane close to $\boldsymbol{z}_V$, removes only little volume from the template, while low κ puts it closer to $\boldsymbol{a}$. Since $\boldsymbol{z}_V$ is only the counterexample

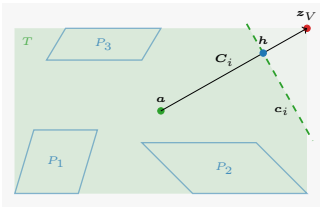


Fig. 7: The algorithm used to find half-space constraints, by cutting counterexample $\boldsymbol{z}_V$ from the template T with a hyperplane $(\boldsymbol{C}_i, \boldsymbol{c}_i)$. The normal of the hyperplane (specified by $\boldsymbol{C}_i$) is given by the vector between $\boldsymbol{a}$ (the center of T) and $\boldsymbol{z}_V$. The threshold $\boldsymbol{c}_i$ is chosen such that the hyperplane remove $\boldsymbol{z}_V$ but does not intersect any relaxations P_1, P_2, P_3 the template T was created from.

with the largest violation, but not necessarily the whole region preventing certification, the half-space constraint obtained from a high κ might not be sufficient to separate this region from T. Thus, in a subsequent iteration, another half-space constraint for the same region may be added. For low κ, fewer constraints are required, but more verifiable volume of T is lost.

C.3 Template Expansion

Algorithm 4: Template expansion

Input: $\mathcal{T} = \{T_i\}_{i=1}^m$, scaling matrix $\boldsymbol{D}$
Result: Set $\mathcal{T}$ of expanded templates, $|\mathcal{T}| = m$
for $i \leftarrow 1$ **to** k **do**
  $T \leftarrow T_i$
  while $N_{k+1:L}(T) \vdash \psi$ **do**
    $T_i \leftarrow T$
    scale all dimensions of T according to $\boldsymbol{D}$
  end
end
return $\mathcal{T} = \{T_i\}_{i=1}^m$

To further improve the generalization from the training set to the test set, we introduce an operation called *template expansion*, outlined in Algorithm 4. For the scaling operation we utilize a diagonal matrix $\boldsymbol{D} := \text{diag}(f_1, \ldots, f_d)$, where $f_i \geq 1$ is the scaling factor for the i-th dimension, e.g. $\boldsymbol{D} = 1.1\boldsymbol{I}$ would scale all dimensions by a factor 1.1.

Half-space Constraints When we use the Box abstraction with half-space constraints (Stars) to encode templates, the expansion only modifies the underlying

box, but not the half-space constraints, since these have already been selected to be close to the decision boundary. However, we attempt to add new constraints during the expansion process if the verification fails.

C.4 Experimental Evaluation

Here we instantiate the presented algorithm App. C.2 for the MNIST dataset. We first consider templates expressed in the Box abstraction, and investigate Stars with additional half-space later. We generate the set of templates $\mathcal{T}$ individually for each label.

We consider a 5x100 network, apply template generation individually per label and create templates at the third and fourth layer. For technical details see the end of this section. We create templates in the box as well as in the star abstraction and allow up to $m = 25$ templates and use template expansion (App. C.3). For $\epsilon = 0.05$ we found between 6 and 24 templates per label and layer.

Table 15 shows the results. We provide the fraction of input regions that could be successfully matched to templates as well as the overall verification time. If an input cannot be matched with any of the templates, then we propagate the standard Zonotope abstraction through the rest of the network to verify it.

We observe that templates can subsume up to 60.6% of input regions in the test set with $\epsilon = 0.05$, and up to 49.2% for higher $\epsilon = 0.1$ when expressed in the Box abstraction. Combining template at multiple layers gives more matched templates as many inputs can be matched in the third layer, while unmatched ones can again be considered at the fourth layer. We observe that improvements in matching rate directly lead to speed ups over standard verification. Additionally allowing half-space constraints, that is use Stars instead of Boxes as templates, allows us to increase the matching rate up to 65.2 % and 54.5 % for the two ϵ respectively. However, as checking matches for Stars is computationally more expensive the verification time does not improve further.

Adding half-space constraints to boxes allows them to have larger bounds and still be verifiable, with the half-space constraints truncating counterexamples from the templates. These stars can better approximate the decision boundaries, thereby increasing the number of matches on average from 35.2% to 65.1%. Unfortunately, this gain cannot be converted to a speed-up, due to the larger cost of the containment check for stars and the late template layer.

These results highlight, that with the algorithm outlined in App. C.2, a set of templates $\mathcal{T}$ can be obtained that generalize remarkably well to new input regions (e.g., up to 65.2 % containment). That is, while more precise abstractions such as stars allow templates that capture a far higher rate of containment for new input regions, their added cost of the containment check makes the obtained speedups smaller.

Finally, we see that Template Expansion (App. C.3) uniformly leads to a higher matching rate and speed-ups.

Table 15: Template matching rate and verification time of the whole MNIST test set t in seconds for the 5x100 using up to m templates per label and layer pair. The baseline verification 292.13 ± 1.77 and 291.80 ± 2.36 seconds for $\epsilon = 0.05$ and $\epsilon = 0.10$ respectively. +TE indicates the use of Template Expansion.

Box	shapes matched [%]			verification time [s]		
k	$m=1$	$m=3$	$m=25$	$m=1$	$m=3$	$m=25$
$\epsilon = 0.05$						
3	07.5	14.4	28.4	282.60 ± 2.18	275.32 ± 1.34	259.87 ± 0.68
4	19.5	38.0	57.6	279.87 ± 0.07	270.08 ± 0.81	258.06 ± 1.23
3+4	20.7	39.3	59.2	274.86 ± 0.31	259.19 ± 0.91	243.11 ± 0.81
$\epsilon = 0.10$						
3	05.1	10.2	19.4	286.67 ± 1.17	280.45 ± 1.03	272.01 ± 0.68
4	15.0	29.6	45.8	283.06 ± 1.49	275.39 ± 1.04	268.05 ± 1.04
3+4	15.8	30.7	47.4	281.14 ± 1.12	272.17 ± 1.18	257.92 ± 0.97

Box+TE	shapes matched [%]			verification time [s]		
k	$m=1$	$m=3$	$m=25$	$m=1$	$m=3$	$m=25$
$\epsilon = 0.05$						
3	7.8	15.0	30.5	284.27 ± 0.63	274.26 ± 1.78	257.77 ± 1.21
4	20.1	38.9	59.0	280.44 ± 1.66	268.97 ± 1.13	258.02 ± 1.38
3+4	21.3	36.0	60.6	275.31 ± 2.62	260.95 ± 0.70	241.36 ± 1.34
$\epsilon = 0.10$						
3	5.4	10.6	21.2	285.76 ± 0.71	278.94 ± 1.12	266.85 ± 2.07
4	15.6	30.5	47.7	285.15 ± 0.47	274.82 ± 1.41	266.62 ± 0.45
3+4	16.3	31.5	49.2	280.49 ± 0.70	269.90 ± 0.34	257.82 ± 3.01

Star	shapes matched [%]			verification time [s]		
k	$m=1$	$m=3$	$m=25$	$m=1$	$m=3$	$m=25$
$\epsilon = 0.05$						
3	10.8	25.2	40.4	281.33 ± 1.44	269.24 ± 1.30	254.41 ± 0.87
4	25.8	40.3	62.7	282.51 ± 2.10	281.68 ± 0.60	271.63 ± 0.89
3+4	27.9	45.1	64.3	277.89 ± 1.38	266.55 ± 0.93	246.39 ± 0.31
$\epsilon = 0.10$						
3	7.9	18.2	29.0	284.31 ± 2.19	277.96 ± 0.60	267.72 ± 0.85
4	20.2	31.9	51.0	285.68 ± 2.10	286.16 ± 0.79	278.29 ± 1.30
3+4	21.7	35.8	52.9	283.43 ± 0.91	278.00 ± 0.60	262.96 ± 1.04

Star+TE	shapes matched [%]			verification time [s]		
k	$m=1$	$m=3$	$m=25$	$m=1$	$m=3$	$m=25$
$\epsilon = 0.05$						
3	11.1	25.8	41.8	282.99 ± 0.45	271.34 ± 1.89	254.62 ± 1.05
4	25.9	40.6	63.6	282.98 ± 1.46	281.51 ± 0.23	270.44 ± 0.30
3+4	28.3	45.6	65.2	278.00 ± 0.36	269.08 ± 0.80	247.43 ± 0.09
$\epsilon = 0.10$						
3	8.1	18.7	30.3	285.82 ± 1.91	280.71 ± 2.99	267.55 ± 0.77
4	20.3	32.3	52.8	285.49 ± 1.04	286.34 ± 0.79	276.14 ± 0.30
3+4	22.0	36.2	54.5	284.39 ± 2.29	278.90 ± 0.34	262.60 ± 1.71

Technical Details We use a feed forward neural network with five linear layers of size 100 and ReLU activations, trained with DiffAI [34]. The network has an accuracy of 0.94 and a certified accuracy of 0.93 and 0.92 for $\epsilon = 0.1$ and $\epsilon = 0.2$ respectively.

For a CLUSTER_PROOFS we set an initial cluster size depending on the number of verifiable images per label, in order that the clusters contain on average 50 images. For the verification of a cluster's union, we allow up to $n_{\text{hs}} = 30$ half-space constraints and set κ of 0.05. Taking a low value leads to a larger truncation, but reduces the number of half-space constraints, which speeds up the template generation as well as the containment check at inference. We take the same values also for verifying unions after merging two clusters. For expanding the templates, we use up to 10 iterations, in which we widen by 5% in each dimension and then allow up to 10 hyperplanes to verify the expanded template. To avoid truncating previously verified volume, we increase κ linearly by 0.02 for each expansion step, starting with an initial κ of 0.4