Uncovering the Limits of Proof Sharing for Neural Networks

Kanak Das[1⋆], Shubham Ugare[2], Bor-Yuh Evan Chang[3], Sasa Misailovic[2], Gagandeep Singh[2], and Manu Sridharan[1]

[1] University of California, Riverside
kdas006@ucr.edu, manu@cs.ucr.edu
[2] University of Illinois Urbana-Champaign
sugare2@illinois.edu, misailo@illinois.edu, ggnds@illinois.edu
[3] University of Colorado Boulder & Amazon[⋆⋆]
evan.chang@colorado.edu

Abstract. Robustness verification of neural networks is increasingly important, due to their use in many critical domains. In certain scenarios, proof sharing has been shown to accelerate incomplete verification techniques by reusing intermediate-layer abstract states, or *templates*, across queries. However, questions remain as to the robustness of template-based acceleration across varying network architectures, properties, datasets, and training methods. In this work, we perform a systematic study of the effectiveness of template-based acceleration and its limits. Our study shows that template subsumption rates can vary widely across scenarios. We present a novel metric of *jointly stable neurons* to explain this variation, showing that in some cases template-based techniques are very unlikely to provide any speedup. Then, we present FastCert, a novel technique for *automatically* distributing templates across neural network layers to increase performance impact, eschewing templates entirely if they are unlikely to produce a speedup. Across a large set of covering-design based L_0-verification tasks, FastCert achieved an average speedup of 1.13× over an extant template-based reuse technique.

⋆ Corresponding author.
⋆⋆ Bor-Yuh Evan Chang holds concurrent appointments at the University of Colorado Boulder and as an Amazon Scholar. This paper describes work performed at the University of Colorado Boulder and is not associated with Amazon.

1 Introduction

Neural networks have emerged as the dominant paradigm for image classification, powering applications from autonomous driving to medical diagnosis. Despite their success, it is established that carefully crafted perturbations known as adversarial examples can cause networks to misclassify inputs, while remaining imperceptible to human observers [5, 12, 35].

To mitigate these risks, there has been significant interest in creating certifiably robust neural networks, resistant to adversarial threats [18]. These methods expose networks to adversarial perturbations during training, often leveraging formal verification techniques to guide and refine the optimization process [20, 41]. Then, formal verifiers [17, 42] are used post-training to guarantee the network's robustness to specified perturbations, such as sound but incomplete verifiers based on abstract interpretation [30, 31, 41].

Recent work has shown that intermediate abstractions computed by such incomplete verifiers can be generalized into templates [10, 37], enabling proofs to be *reused* and accelerating subsequent verification queries. Typically, a small set of templates are derived by verifying local L_∞ properties [12, 18, 35], each permitting perturbations within a related region of the input space. These methods have been shown to be effective on related properties of the same input over the same or quantized networks, such as shifted patches or random t-pixel perturbations over an image. But, the effectiveness of template-based acceleration has not been studied carefully in the context of other properties such as L_0 verification, a realistic threat model [15] where there have been significant recent advances in verification practicality [26, 28]. Further, no work has studied the limits of template-based acceleration across a variety of network architectures, properties, data sets, and training methods.

Our first key contribution is a systematic framework characterizing when template reuse is a principled accelerator for robustness verification and when the verifier should instead run without it. For a template-based verifier to achieve speedups, a high percentage of verification queries must be subsumed by a small set of templates, but at the same time, each template must be precise enough to enable query verification. We devise a novel metric of *jointly stable neurons* across a set of abstract states to explain the potential for template reuse given this tradeoff. Through a limit study, we show that if the percentage of jointly stable neurons is low, extant template-based proof sharing techniques are unlikely to provide any verification speedup. Our measurements across several networks and training methods show that the potential for template-based acceleration varies widely across scenarios.

Given the observed variance in template effectiveness, there exists a need to automatically determine how best to employ templates for a given verification task. Our second key contribution is a novel technique FastCert to *automatically* distribute templates across layers of a neural network to maximize performance impact, or eschew templates if they are unlikely to produce a speedup. FastCert performs profiling and template generation before verification, predicting template subsumption rates by sampling a very small set of queries. We

implement FastCert and show that for a state-of-the-art L_0 verifier, it nearly always provides a greater speedup than previous proof-sharing techniques in cases where templates can be effective, while significantly mitigating the slowdown when templates are ineffective.

This paper makes the following contributions:

- We perform a limit study of the effectiveness of template-based proof reuse, characterize its potential success using a novel metric of jointly stable neurons and showing the success rate varies widely.
- We give a method to automatically determine whether, where, and how to apply template reuse for a given network and set of verification tasks.
- We implement our approach in a tool FastCert and demonstrate its practicality for accelerating covering-design-based L_0 verification, with an average speedup of 1.13× over the state-of-the-art approach for applying templates (a 7 hour reduction in wall-clock time.).

2 Background

Neural Networks A neural network N is a function $N : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}}$, commonly built from a composition of individual layers $N_L \circ N_{L-1} \circ \cdots \circ N_1$. We consider fully-connected feed-forward networks, where each layer $N_i(\mathbf{x}) = \text{ReLU}(\mathbf{A}\mathbf{x} + \mathbf{b})$. The Rectified Linear Unit (ReLU) activation applies $\max(\cdot, 0)$ element-wise. For a classification task with c classes, the network outputs $d_{\text{out}} := c$ scores, assigning the class with the highest score as its prediction. For $k < L$, $N_{1:k}$ denotes the application of the first k layers and $N_{k+1:L}$ denote the final $L - k$ layers.

Local Robustness Verification Given a set of inputs and a postcondition ψ, neural network verification aims to prove that ψ holds on the network output for all given inputs [1]. *Local robustness verification* proves that ψ holds for all network outputs corresponding to an input region $\mathcal{I}(\mathbf{x}_0)$ formed around some input $\mathbf{x}_0$. Formally, local robustness verification proves that $\forall \mathbf{z} \in \mathcal{I}(\mathbf{x}_0), N(\mathbf{z}) \models \psi$. We write $\mathcal{I}(\mathbf{x}_0) \models \psi$ if the property holds.

Threat Models and Specifications The L_0 threat model captures *sparse* perturbations: given an input $\mathbf{x}_0$, the admissible region $\mathcal{I}_{L_0}(\mathbf{x}_0, k)$ is $\{\mathbf{z} \mid \|\mathbf{x}_0 - \mathbf{z}\|_0 \leq k\}$, allowing an adversary to arbitrarily modify up to k input features[8, 21, 34]. Unlike norm-bounded dense noise (e.g., small L_∞ deviations) that yields a single convex region, the L_0 model induces a combinatorial family of verification subproblems for one image, corresponding to the many choices of which features are altered. In the patch threat model [6, 9], an adversary may arbitrarily perturb the pixels inside a single contiguous $w \times h$ region, leaving all other pixels unchanged; the input region $\mathcal{I}_{\text{patch}}(\mathbf{x}_0, w, h)$ allows arbitrary changes to any $w \times h$ patch within the image, yielding multiple verification subproblems across patch locations. In geometric perturbations, the adversary applies a transform T_δ from a prescribed family (e.g., bounded rotation/contrast/brightness), with

parameters $\delta \in \Delta$; the input region $\mathcal{I}_{\text{geo}}(\mathbf{x}_0, \Delta) = \{T_\delta(\mathbf{x}_0) \mid \delta \in \Delta\}$ is typically non-convex due to resampling/interpolation and is commonly verified by splitting Δ into finitely many r cases [2].

Robustness Under Adversarial Threats To defend against the adversarial threat, specialized types of neural network training is employed. It pursues a dual objective: minimize classification loss on data while maximizing robustness so predictions remain stable within the threat model. PGD [18] training does this empirically by finding worst-case perturbations per example and updating the model to classify them; IBP [20] training does it with sound guarantees by propagating bounds to certify that all inputs in a region keep the label and optimizing the certified margin. Recent methods such as SABR [22] compute bounds on small, carefully selected subregions of the adversarial space to reduce approximation error and improve both standard and certified accuracy, while TAPS [19] combines IBP and PGD to optimize more precise but unsound worst-case loss approximations. CURE [13] targets multi-norm robustness by regularizing and shaping networks so bounds remain tight across multiple norms. For localized threats, certified patch defenses [6] extend guarantees from norm-bounded noise to contiguous regions by reasoning over all patch placements.

Verification via Abstract Interpretation Scalable neural network verifiers typically rely on *abstract interpretation* [7, 30–32] to soundly overapproximate the set of reachable outputs. The verifier first encodes the input region with a shape in some *abstract domain* (e.g., boxes, zonotopes, or polyhedra) that encloses $\mathcal{I}(\mathbf{x}_0)$. This shape is propagated layer by layer via *abstract transformers* that overapproximate each layer's concrete operations. After k layers, the resulting *abstract state* S_k overapproximates the concrete states reachable at that layer. At the output layer, S_L bounds all possible network outputs; if it also satisfies the postcondition ψ, the property is certified to hold. This approach is sound but incomplete: the overapproximation may be too coarse to prove a valid ψ.

3 Proof Templates

In this section, we describe template-based proof sharing for incomplete verification and present a simple performance model for when reuse is beneficial.

3.1 Preliminaries

Neural network verification via abstract interpretation has been optimized using *proof templates* [10, 37]. A template at an intermediate layer k is an abstract state T_k that has been certified to satisfy the postcondition ψ, i.e., $N_{k+1:L}(T_k) \models \psi$. To use a template for a new verification query, the verifier propagates the query to layer k, obtaining abstract state Q_k. If Q_k is fully contained within the template, termed *subsumption*, and denoted $Q_k \sqsubseteq T_k$, then $N_{k+1:L}(T_k) \models \psi$ implies $N_{k+1:L}(Q_k) \models \psi$, verifying the query without further propagation.

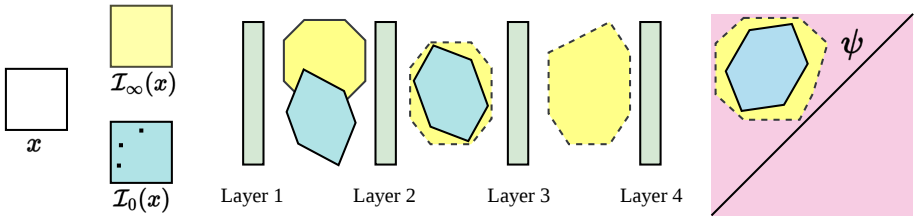


Fig. 1: Template reuse in a 4-layer network. From an input image x, we first verify an initial query under $\mathcal{I}_\infty(x)$, then build a layer-k=2 template by iteratively relaxing the layer-2 abstract state while it remains verifiable (represented by the yellow region with dotted boundary). For a later query under $\mathcal{I}_0(x)$, the propagated layer-2 query state is subsumed by the template, so the property is proved immediately and propagation through layers 3 and 4 is skipped, saving tail cost.

In practice, the box domain is employed for representing templates [10, 37]: at layer k, a box is a vector of intervals, $S_k = \langle [l_j, u_j] \rangle_{j=1}^{n_k}$, with each interval bounding the output of neuron j. Subsumption checking in the box domain is very efficient, making it desirable for templates.

3.2 Template Construction and Use

In prior work [10, 37], a verified query is leveraged to construct a proof template by generalizing its intermediate-layer abstract state. In fig. 1, the abstract interpreter first verifies an L_∞ query $\mathcal{I}_\infty(x)$—it computes an abstract state after each layer over-approximating the result of passing the input region through the previous layers. The query is considered verified if the final abstract state, computed after layer 4, satisfies the post-condition ψ. For fig. 1, as shown by the decision boundary after layer 4, the query is successfully verified.

To construct a template, first, a set of layers is chosen at which to use templates. At a selected layer k, the previously-computed abstract state is iteratively relaxed while preserving verifiability through the suffix $N_{k+1:L}$, yielding a more general state that can subsume subsequent queries whose layer-k states are similar. Existing approaches commonly instantiate this procedure from L_∞ specifications [10, 37]: they first use binary search to find the largest verifiable L_∞-ball around the input, then relax the resulting layer-k abstract state to maximize generality subject to successful propagation and discharge on $N_{k+1:L}$. The resulting relaxed state is stored as a template at layer k, which can be reused for potential subsumption of future queries. Moreover, prior work shows that replacing a single L_∞-ball over the input space with multiple masked L_∞-balls over subregions yields a more diverse template set and increases subsumption rates [10, 37].

Using templates requires checking if the abstract shape computed for a subsequent query is subsumed by a template (see section 3.1). For fig. 1, a template

is placed after layer 2 (the yellow region with dotted boundary), and a subsequent query aims to verify an L_0 query $\mathcal{I}_0(x)$. Since the abstract state for this query is contained within the layer 2 template, the property is verified, without further propagation of the abstract state through the remaining layers.

3.3 A Performance Model for Template-Based Verification

The potential performance improvement provided by proof sharing with templates is impacted by various factors, and depending on the template subsumption rate, templates may even lead to a slowdown (which we observed in practice). We will refer back to the model presented here when discussing the potential for template-based speedups for different networks (section 4) and presenting our technique for automatically optimizing template usage (section 5).

Consider a network N with L layers and a fixed input x inducing r verification specifications. Fix a reuse layer k and a template set $\mathcal{T}$ of size m. Let t_{gen} denote the one-time cost to construct $\mathcal{T}$; $t_{\text{pref}}(k)$ the per-specification cost to propagate through $N_{1:k}$; $t_{\text{match}}(m)$ the per-specification cost to test subsumption $Q_k \sqsubseteq T$ for all $T \in \mathcal{T}$; and $t_{\text{tail}}(k)$ the per-specification cost to complete verification over $N_{k+1:L}$ when no match occurs. Let $\rho_k \in [0,1]$ be the fraction of the r specifications whose layer-k abstract state is subsumed by at least one template in $\mathcal{T}$. The expected end-to-end verification time with template reuse is then,

$$t_{\text{gen}} + r\Big(t_{\text{pref}}(k) + t_{\text{match}}(m) + (1-\rho_k)\, t_{\text{tail}}(k)\Big) \tag{1}$$

Under the assumption that the baseline verification exhibits approximately constant per-layer computational cost ν, it follows that $t_{\text{pref}}(k) = k\nu$, $t_{\text{tail}}(k) = (L-k)\nu$, $t_{\text{gen}} = \lambda m L \nu$ for template creation time parameter $\lambda > 0$ and $t_{\text{match}}(m) = \eta m$ for template matching time parameter $\eta > 0$. Speedup S, relative to baseline verification time $rL\nu$ is therefore,

$$S \;=\; \Big(1 + \lambda\tfrac{m}{r} + \tfrac{\eta m}{L\nu} - \rho_k \tfrac{L-k}{L}\Big)^{-1} \tag{2}$$

Hence $S > 1$ is precisely when the *saved tail cost* $\rho_k(L-k)/L$ exceeds the *amortized overheads* of template generation and lookup, $\lambda m/r + \eta m/(L\nu)$. Operationally, this favors (i) *earlier viable layers* where $(L-k)/L$ is large, provided the queries yields a sufficiently high ρ_k; (ii) *a small number of templates* to keep lookup and generation costs small; and (iii) *large number* of specifications, which amortize t_{gen}. Using templates at very deep layers (small residual) or oversized template sets (large overhead) eliminate gains, in which case the template based approach might not be beneficial.

With this speedup model, prior work [10, 37] has shown that template reuse can yield significant performance gains for patch verification, fixed-size L_0 verification, and geometric perturbations. In such scenarios, verification instances count r is typically a few hundred and template set length m is up to 4. However, prior work did not perform a rigorous study of the impacts of neural network type, dataset, training method, and layer choice on the effectiveness of template-based proof sharing.

4 The Limits of Template-Based Proof Sharing

Here, we first present a model for understanding the limits of existing template-based proof sharing techniques to speed neural network verification, based on the concept of *neuron stability*. Then, we present a limit study of template-based proof reuse, showing its applicability varies widely depending on specific settings.

4.1 Efficient Proof Sharing and Neuron Stability

For efficient proof sharing via templates, section 3.3 shows that we need (1) a high template match rate, (2) low cost to compute templates, and (3) low cost to determine template subsumption for an abstract shape. Further, there is a baseline correctness condition that (4) the abstract interpreter can verify the property for the template. Here, we explore how neuron stability interacts with these conditions, specifically how many unstable neurons make it very unlikely all four conditions can be simultaneously satisfied.

Neural network ReLU activation layers (see section 2) act element-wise, suppressing negative pre-activations and preserving positive ones. ReLU therefore partitions neurons into active (positive output) and inactive (zero output) sets, creating an activation pattern. In abstract interpretation, when propagating an abstract state through the network, ReLU layers are the primary source of imprecision. For the box domain, given an abstract state for a neuron j represented by the interval $[l_j, u_j]$, the abstract interpreter must determine the effect of applying ReLU on the interval. This requires a case analysis based on the containment of 0 in $[l_j, u_j]$, as 0 is the ReLU decision boundary. Hence, the precision of the abstract interpretation hinges on whether the abstract state of the pre-activation value of a neuron is sufficient to determine the outcome of its ReLU activation.

A neuron is considered *stable* [4, 25] if its pre-activation interval, $[l_j, u_j]$, does *not* cross the ReLU's decision boundary of 0. This occurs under two conditions:

- The interval is entirely non-negative ($l_j \geq 0$), meaning the neuron is *always active* and the ReLU function consistently acts as the identity.
- The interval is entirely non-positive ($u_j \leq 0$), meaning the neuron is *always inactive* and the ReLU behaves as the constant zero function.

In both cases, the activation behavior is fixed across all concrete behaviors represented by the abstract state, hence the term *stability*.

A neuron is *unstable* if its interval $[l_j, u_j]$ has a negative lower bound and a positive upper bound ($l_j < 0 < u_j$). For such neurons, the abstract state is too coarse to resolve the sign of the pre-activation value. Consequently, a sound abstract transformer for the ReLU function must conservatively account for both outcomes (an active or inactive neuron), typically leading to precision loss. For an abstract state S_k, we denote the sets of indices for stable and unstable neurons as $\text{Stable}(S_k)$ and $\text{Unstable}(S_k)$, respectively.

Induced Partial Orders The subsumption relation $\sqsubseteq$ on abstract states (section 3.1) has direct implications for neuron stability. If a state Q_k is subsumed by a template T_k, any unstable neuron in Q_k must also be unstable in T_k. This is because the query's interval $[l_j^Q, u_j^Q]$ containing zero implies the necessarily wider template interval $[l_j^T, u_j^T]$ also contains zero. This yields the following inclusion relation unstable neuron sets:

$$Q_k \sqsubseteq T_k \quad \Longrightarrow \quad \text{Unstable}(Q_k) \subseteq \text{Unstable}(T_k)$$

Hence, a template must accommodate all the instabilities present in any query it subsumes. Similarly, the sets of stable neurons are related by the inclusion $\text{Stable}(T_k) \subseteq \text{Stable}(Q_k)$.

Template Generality vs. Stability Variance The effectiveness of proof sharing hinges on a trade-off between template generality and the stability of individual queries. For a small set of templates to be effective, each template must be general enough to subsume a large number of queries. But, for a single template T_k capable of subsuming a set of queries $\{Q_k^i\}_{i=1}^m$, $\text{Unstable}(T_k)$ must include the *union* of the unstable neurons of the queries:

$$\bigcup_i \text{Unstable}(Q_k^i) \subseteq \text{Unstable}(T_k) \tag{3}$$

Hence, a very general template must account for all the instabilities present in the queries it subsumes. This instability set can grow large if the queries exhibit *diverse* instability patterns, with differing unstable neurons across the abstract states for individual queries.[4] Such instability variance poses a fundamental challenge to creating a compact and effective set of templates: a template that abstracts over a large number of unstable neurons is likely too imprecise to enable verification of the property of interest. (Every unstable ReLU necessitates an over-approximation, introducing precision loss that compounds layer-by-layer, often rendering the final bounds too loose to be conclusive.) And, since efficient template computation and matching is required for speedups, this issue cannot be sidestepped by larger template sets or more complex templates.

4.2 Neuron Instability in Practice

Here, we describe a study that shows the limits of proof sharing via templates in practice and the relationship of neuron stability patterns to proof sharing potential.

Methodology We used the MNIST and CIFAR-10 datasets in our study. We considered six neural networks for each dataset, varying in training methods. Alongside standard training, we tested the training methods PGD [18], SABR [22],

[4] In principle, a neuron that is stable in each individual query can still be forced unstable in a template if it is stably active in some queries and stably inactive in others. We observed no such cases in our experiments.

TAPS [19], CURE [13], and CertifiedPatchDefense (CPD) [6]. All the networks used in the limit study consist of 7 fully connected layers, with 200 neurons per layer.

We considered three perturbation types to generate verification queries from a common input image that yield large number of queries and are amenable to be analyzed via abstract interpretation based verifiers: (1) random L_0 perturbations of 1 to 30 pixels generating 1000 queries, (2) localized 2×2 patch perturbations generating 729 queries for MNIST images and 1024 queries for CIFAR-10 images, and (3) composite geometric transformations used in prior work on MNIST [2, 10] including $\pm 2^\circ$ rotation, $\pm 10\%$ contrast and $\pm 1\%$ brightness changes split into r perturbations. We present L_0 perturbations as a representative case; results for patch and geometric perturbations, which exhibit similar trends, are deferred to Appendix 1.

For each generated query, we propagate the input region with DeepZ and record the abstract state Q_k^i at each ReLU layer k with n_k neurons. For a query set $\{Q_k^i\}_{i=1}^m$, *Jointly Stable Neurons* at percentile $p \in [0, 1]$ is the fraction of neurons stable in at least a p-fraction of the queries:

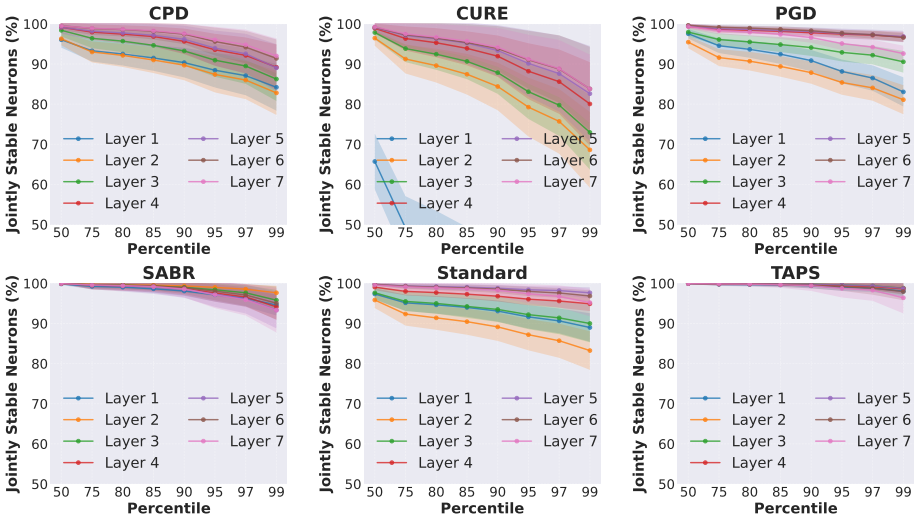
$$\text{JointlyStableNeurons}_p(\{Q_k^i\}_{i=1}^m) = \frac{\left|\left\{j : \frac{|\{i : j \notin \text{Unstable}(Q_k^i)\}|}{m} \geq p\right\}\right|}{n_k} \quad (4)$$

At $p = 1$, this is the fraction of neurons outside the union of unstable neurons across all queries at that layer, i.e., $1 - |\bigcup_i \text{Unstable}(Q_k^i)|/n_k$. By eq. (3), a smaller value means any template subsuming all queries must contain many unstable neurons. A steep drop as p increases indicates that instability is dispersed across queries, so templates covering an increasing fraction of queries must include more unstable neurons, limiting their effectiveness for proof sharing.

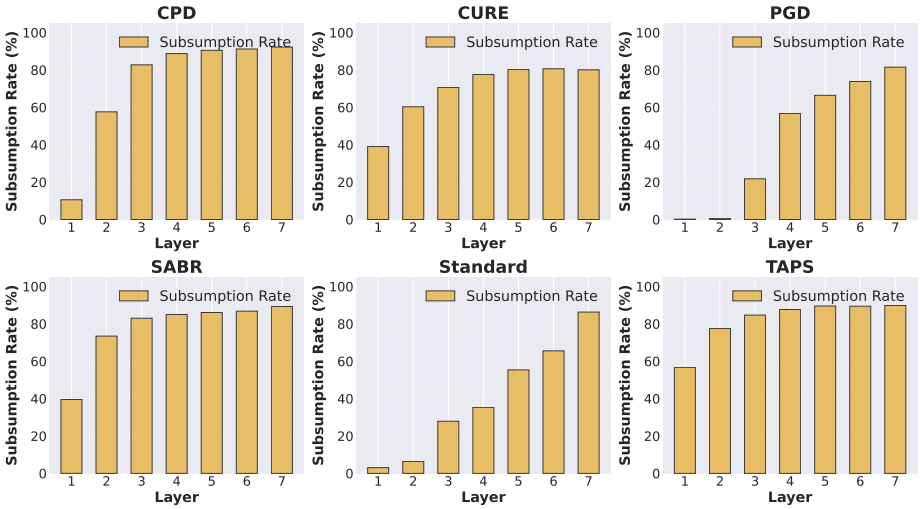
In figs. 2 and 3, we plot *Jointly Stable Neurons* and template subsumption for all the ReLU layers of networks trained on MNIST and CIFAR-10, respectively, using 100 input images per network; shaded bands indicate variability across images. The x-axis gives the percentile threshold p expressed as a percentage, and the y-axis gives $\text{JointlyStableNeurons}_p(\{Q_k^i\}_{i=1}^m)$ as a percentage of neurons at that layer. We use one template generated under L_∞ perturbation for subsumption checking; following the template generation technique described in section 3.2.

Subsumption Rates First, focusing on the template subsumption rates in figs. 2 and 3, we see a significant variance across datasets, layers, and training methods. Consistent with prior observations [37], later layers consistently have higher subsumption rates, as by that point the networks tend to distill the semantic features of the input, enabling similar queries to yield similar abstract states.

Training method has a substantial impact on potential subsumption rates. Networks trained without specific certifiable guarantees, the Standard and PGD trained models, exhibit generally lower subsumption rates: for MNIST this is most pronounced in the earlier layers, while for CIFAR-10 it persists across



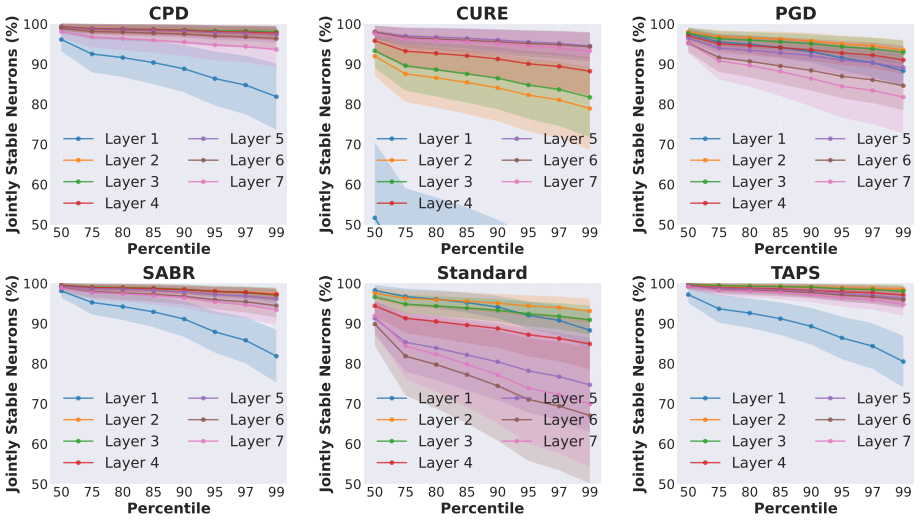
(a) Jointly stable neurons



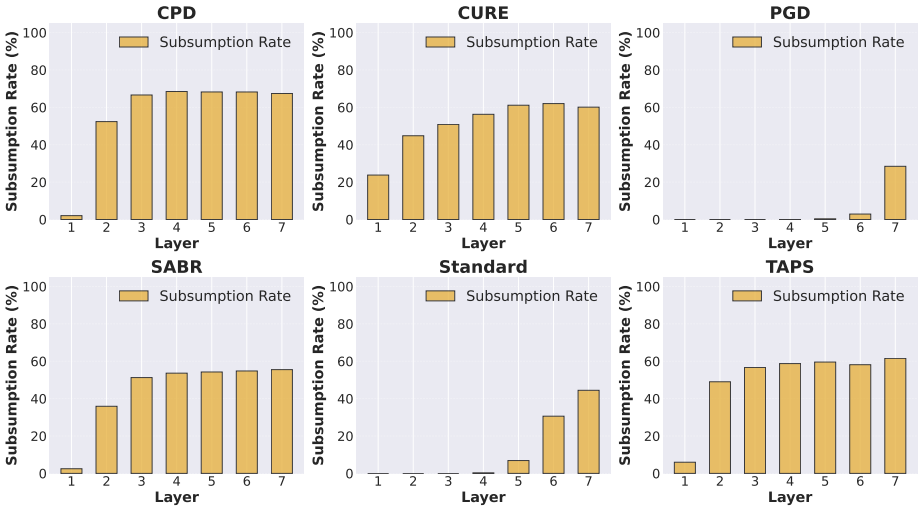
(b) Template subsumption rates

Fig. 2: Jointly stable neurons and template subsumption rates under random-L_0 perturbations in MNIST networks.

all layers. In contrast, networks trained with certified objectives (CURE, CPD, SABR, TAPS) are designed to satisfy worst-case guarantees against perturbations, and thus tend to emphasize features that remain informative despite variations in the perturbations. We see these behaviors reflected in generally higher subsumption rates for these training methods across both datasets.



(a) Jointly stable neurons



(b) Template subsumption rates

Fig. 3: Jointly stable neurons and template subsumption rates under random-L_0 perturbations in CIFAR-10 networks.

However, even for certified-trained networks, the subsumption rates vary significantly across layers, and across datasets. E.g., the subsumption rate at layer 2 on MNIST for TAPS roughly 80% whereas for CPD its less than 60%. This unpredictability led us to develop the profiling technique for automating template configuration described in section 5.

Joint Stability Figures 2 and 3 show that in nearly all cases, greater joint stability across abstract states correlates with higher template subsumption, as expected. For the majority of the cases, we observe that jointly stable neurons drop quickly as p increases at the first layer for all networks. Hence, if a single template were to subsume a majority of these abstract states, it must have a high number of unstable neurons, and hence would be unlikely to enable verification of the desired property. The very low subsumption rates at the first layer confirm this predicted ineffectiveness. Conversely, at later layers, the drop in jointly stable neurons is significantly less, indicating that the template will not have to accommodate high instability. And consequently, we observe higher subsumption at these layers.

We do observe some interesting exceptions to these trends. On CIFAR-10 Standard and PGD in fig. 3, earlier layers exhibit high joint stability across percentile of queries, but subsumption remains low. These cases indicate that while joint stability captures a key prerequisite for reuse, it does not guarantee that the resulting per-neuron bounds align tightly enough across queries to be subsumed by a few templates. Even with similar joint stability, early-layer abstract states may be insufficiently distilled (or too high-variance in interval geometry). Dataset complexity likely plays a role as well. As MNIST is far less feature-rich than CIFAR-10, homogenic semantic structure may emerge earlier in the network, allowing early-layer templates to be useful. For example, consider the TAPS and SABR-trained MNIST networks in fig. 2; unlike their CIFAR-10 counterparts, they exhibit some subsumption even in the first layer, and joint stability is maintained.

CURE on MNIST in fig. 2 illustrates a nuanced case. Although joint stability drops at stricter percentile thresholds, the drop is not severe at the coverage levels relevant to the observed subsumption rates. In particular, subsumption rates remain around $60-80\%$ starting from layer 2, and the joint stability curves indicate that a large fraction of neurons remain jointly stable for a comparable fraction of queries. Thus, the low stability at the highest percentiles does not rule out reuse: it says that a single template is unlikely to cover nearly all queries without absorbing additional instability. Templates can still be effective when targeting a substantial, but not exhaustive, subset of the queries.

Overall, the results indicate that joint stability effectively explains subsumption rates for template-based proof sharing.[5] The data show that the effectiveness of such proof sharing is highly configuration-dependent; a fixed policy for where and how to use templates will not generalize across these settings. In the next section, we describe a new adaptive strategy to automatically adapt to these varying conditions.

[5] Similar trends hold for patch and geometric perturbations; results are deferred to Appendix 1.

5 Automatic Configuration of Templates

The practical efficacy of template-based proof sharing is critically dependent on a combination of factors: the training methodology, the specific property under verification, the cardinality of the template set used, and the choice of layer for reuse. An uninformed use of templates could actually degrade overall verification performance, a phenomenon we have observed in practice.

Prior work [10, 37] largely sidestepped this parameterization challenge, relying on manual, empirical measurements to select a fixed set of L_∞-masked templates, and up to two promising layers for template reuse. While this methodology has demonstrated viability in constrained settings, manual tuning will not scale to a wide variety of real-world scenarios; the experiments of section 4.2 alone (including the results in Appendix 1) consider hundreds of combinations of training method, layer choice, and perturbation type.

Further, there have been significant advances in L_0 verification via techniques like Calzone [26] and CoverD [28], making L_0 verification increasingly practical but posing new challenges for proof sharing. These methods have expanded verification settings from fixed-size t-pixel perturbations to arbitrary k-pixel perturbations where $k > t$. Calzone [26] exploits the observation that robustness for any $k > t$ perturbed pixels implies robustness for every t subset. It predicts a large k via dynamic programming and sampling, then uses covering designs to generate k-sized queries, submitted to a verifier, refining to smaller sets as needed. CoverD [28] advances this covering-based approach by proposing covering verification designs, an algorithm that selects between candidate coverings without constructing them but rather predicting their block-size distributions from closed-form mean/variance, then constructing the chosen covering on-the-fly. The volume and structural complexity of queries generated by such methods can vary dramatically. Prior work [37] demonstrates template reusability for random 3-pixel perturbations, but covering-design-based approaches involve verifying perturbations that may alter 100 or more pixels, making them a challenging testbed for template reuse.

To improve the applicability of template reuse, we introduce a lightweight, *a priori* profiling technique. This technique provides a principled method to (i) identify and prune verification scenarios where using templates is futile, and (ii) automatically select a set of advantageous layers and curate a compact set of templates to maximize anticipated performance gain.

5.1 Layer Selection and Template Set Refinement

In practice, not all layers are equally suitable for template reuse. Early layers may exhibit low subsumption rates (see section 4.2), while very late layers offer limited potential savings due to the small residual depth. And, the number of templates must be balanced against lookup costs, as larger template sets increase both generation and matching overheads. Identification of a set of layers and template sets that can exhibit the highest potential for subsumption is governed by the performance model from section 3.3. We leverage this model in algorithm 1

Algorithm 1 FastCert Layer and Templates Selection

Input: Network N, image x, candidate layers $\mathcal{L}_{\text{cand}}$, sampling size P, candidate template counts $\mathcal{M}$.
Output: Set of pairs $(k, \mathcal{T}_k)$ of selected layers and their template sets
viable $\leftarrow$ []
for each $k \in \mathcal{L}_{\text{cand}}$ **do**
    $(m^\star, \rho_k^\star, \mathcal{T}_k^\star) \leftarrow$ None
    for each $m \in \mathcal{M}$ **do**
        // Build m templates at layer k from image x
        $\mathcal{T}_k^{(m)} \leftarrow \emptyset$
        Generate m number of L_∞ *masks*
        for $i = 1$ **to** m **do**
            Compute template T_i at layer k using $mask_i$; add T_i to $\mathcal{T}_k^{(m)}$
        end for
        // Estimate subsumption on the set of P L_0 queries on image x
        $c\_\text{hit} \leftarrow 0$
        for $j = 1$ **to** P **do**
            Sample a L_0 query on x; compute Q_j at layer k
            for $i = 1$ **to** m **do**
                if $Q_j \sqsubseteq T_i$ **then**
                    $c\_\text{hit} \leftarrow c\_\text{hit} + 1$ **break**
                end if
            end for
        end for
        $\rho_k(m) \leftarrow c\_\text{hit}/P$
        // Cost-model viability test
        if $\rho_k(m) \geq \rho_k^{\min}(k, m, r)$ **then**
            // Keep the best viable (m, ρ) for this layer determined by subsumption
            $(m^\star, \rho_k^\star, \mathcal{T}_k^\star) \leftarrow (m, \rho_k(m), \mathcal{T}_k^{(m)})$
        end if
    end for
    if $(m^\star, \rho_k^\star, \mathcal{T}_k^\star) \neq$ None **then**
        Append $(k, \rho_k^\star, m^\star, \mathcal{T}_k^\star)$ to viable
    end if
end for
return $\{(k, \mathcal{T}_k^\star) : (k, \cdot, \cdot, \mathcal{T}_k^\star, \cdot) \in \text{viable}\}$

to guide the selection of layers and templates for FastCert. The algorithm takes as input a network, an input image, a set of candidate reuse layers $\mathcal{L}_{\text{cand}}$, sampling size P, and a set of candidate template counts $\mathcal{M}$. It evaluates each layer-template configuration (k, m) to assess its viability for template reuse, and outputs the set of viable (k, m) configurations.

At each layer, to achieve a speedup $S > 1$, the subsumption fraction ρ_k must exceed a minimum threshold $\rho_k^{\min}$ that depends on the amortized overheads and

the residual network depth, $L - k$. This threshold is derived from eq. (2):

$$\rho_k^{\min} \;=\; \left(\lambda\,\tfrac{m}{r} + \tfrac{\eta\,m}{L\,\nu}\right)\frac{L}{L-k} \qquad (k < L).$$

Layers with subsumption fractions below a minimum threshold $\rho_k^{\min}$ will not yield sufficient savings to justify the overhead of matching. In algorithm 1, FastCert evaluates each candidate layer $k \in \mathcal{L}_{\text{cand}}$ and each candidate template count $m \in \mathcal{M}$, using an estimation of the subsumption fraction $\rho_k(m)$ by sampling a set of diverse L_0 queries and computing the fraction subsumed by a set of m templates. A layer-template configuration (k, m) is viable if it meets the threshold $\rho_k(m) \geq \rho_k^{\min}(k, m, r)$.

$$\mathcal{L}_{\text{viable}} = \{k \in \mathcal{L}_{\text{cand}} : \rho_k \geq \rho_k^{\min}\},$$

If no layer-template configuration meets the viability threshold, the algorithm returns an empty set and verification proceeds without templates. This design ensures that template-based verification is applied only when empirical evidence suggests a net performance gain, gracefully degrading to standard verification when template reuse is unprofitable.

6 Methodology

In this section, we describe the methodology used to experimentally validate the effectiveness of FastCert.

6.1 Configuration

We instantiate FastCert with verification queries generated by Calzone [26] and CoverD [28], two state-of-the-art L_0 robustness verification frameworks. Both frameworks generate verification queries using covering designs to efficiently explore the space of possible pixel perturbations. While the original Calzone and CoverD frameworks fall back to complete solvers when search tree over pixel sets is exhausted and reaches leaf nodes, we exclusively use the incomplete verifier for all queries. This isolates the performance impact of template reuse as an improvement on the core abstract interpretation backend.

To create a comprehensive yet tractable evaluation, we generate a total of 750,000 verification queries per network. This total was derived from 10 images per network. We instantiate the workload for t-pixel perturbation tasks with $t = 3$ with Calzone and CoverD. They employ covering-design to generate query sets whose perturbed pixel counts, $k > t$, can vary significantly, producing a heterogeneous set of verification problems. For each image, we cap the workload at 75,000 queries. To ensure consistent query set across different methods compared, we cache the sampling phase of Calzone/CoverD and reuse across all evaluations.

To estimate the subsumption rate ρ_k at each candidate layer, we perform a brief profiling run on a small sample of $P = 200$ queries per image. Across networks and layers, subsumption estimates vary substantially for small samples (up to ∼150 queries), but stabilize around ∼200 queries. Importantly, the threshold-crossing decision is not sensitive to sampling variability beyond this point (e.g., fig. 4). We give more data justifying our choice of P in Appendix 2.

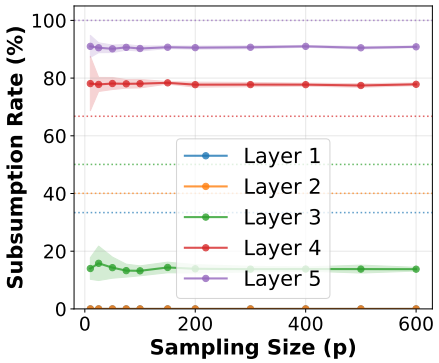


Fig. 4: Sampling size vs. subsumption rate estimates for MNIST PGD with 4 templates.

To balance template diversity against lookup costs, we consider a set of candidate template set sizes of, $\mathcal{M} = \{1, 2, 4\}$. $\mathcal{L}_{\text{cand}}$ includes the layers where the minimum subsumption fraction $\rho_k^{\min}$ is less than or equal to 1 for at least one $m \in \mathcal{M}$, ensuring that only layers with potential for speedup are considered.

We evaluate FastCert on fully-connected and convolutional networks trained on MNIST and CIFAR-10 (summarized in table 1). The suite includes standard, robust, and certified-trained models of varied sizes used in prior work [10, 26, 28, 37]. As we observed that SABR and TAPS trained networks exhibited substantially lower verifiability under Calzone/CoverD-generated L_0 perturbation queries with our incomplete verifier domains, we exclude them from the final evaluation set.

We base our evaluation using DeepZ [10, 30] and Box [20] abstract domains, and compare FastCert against two baseline techniques for both the domains. The first one is a standard verification approach that does not attempt template-based proof sharing. The other one uses a static, hand-picked reuse strategy described in prior work. It always enables template reuse with one template ($m = 1$) at fixed, pre-selected layers (layers 2 and 3 [10, 37]). This allows us to measure the benefit of FastCert's layer and template selection strategy over the state-of-the-art policy.

Table 1: Networks used in the FastCert evaluation.

Dataset	Model	Architecture	#Neurons
MNIST, CIFAR-10	CPDMed	7×200 linear layers	1400
	CPDBig	9×500 linear layers	4500
	CURE	9×500 linear layers	4500
	PGD	6×500 linear layers	3000
	Standard	6×500 linear layers	3000
	CPDConv	4 Conv layers, 4 linear layers	8448

6.2 Implementation and Environment

FastCert is implemented in Python and extends the DeepZ [10, 30] verifier. We integrated the query generation algorithms from Calzone [26] and CoverD [28] to create our verification tasks. All experiments were conducted on a Google Cloud Platform instance equipped with a 12-core Intel Cascade Lake CPU, 85GB of RAM, and an NVIDIA A100 GPU with 40GB of memory.

7 Experimental Evaluation

We conducted an empirical evaluation to validate the efficacy of FastCert. The evaluation showed that the naive reuse of existing policy for template-based proof sharing can be detrimental to performance, and that our automated, profile-guided approach can both mitigate this pitfall and maximize gains. Our investigation was structured around the following research questions:

- RQ1: What is the impact of the previous fixed-layer template reuse strategy on end-to-end verification time for L_0 verification queries?
- RQ2: In cases where the previous strategy reduces verification performance, can FastCert's ability to detect when template reuse is futile mitigate the performance reduction? What is the overhead of the profiling process in this scenario?
- RQ3: In cases where the extant reuse strategy improves verification performance, what further performance gains are obtained from FastCert's automated selection of layers and template count?
- RQ4: Does FastCert generalize across the abstract domains used by incomplete verifiers, in settings where template reuse is applicable?

We answer RQ1 through RQ4 respectively in sections 7.1 to 7.4. Finally, we discuss limitations in section 7.5.

7.1 Performance of Extant Template Reuse Policy

Extant reuse yields substantial speedups on networks where subsumption is moderately high, but consistently slows verification when subsumption is very low. From tables 2 and 3 and fig. 5, we observe two distinct behaviors, *When reuse is viable,* extant reuse achieves speedup ranging $1.01\times - 1.91\times$ on Calzone, and $1.02\times - 1.88\times$ in CoverD. *When reuse is futile,* extant reuse slows down verification by $1\% - 14\%$ on Calzone and $7\% - 14\%$ on CoverD. This performance degradation arises because the overhead of template matching outweighs the negligible savings from subsumption. The performance degradation in these cases is predicted by the template reuse cost model in section 3.3, and by the measures of joint stability from eq. (4) presented in section 4, particularly in figs. 2 and 3.

Table 2: Calzone results (750K queries, 10 images, t=3) with DeepZ and Box domains. V=Verified(%), T=Baseline (No Template) Time in seconds, Sub=Subsumption rate (%), Spd=Speedup, E=Extant Template Reuse Strategy, F=FastCert.

(a) MNIST

Net	DeepZ						Box					
	V	T	Sub		Spd		V	T	Sub		Spd	
	(%)	(s)	E	F	E	F	(%)	(s)	E	F	E	F
CPDMed	92.4	5293	57.8	**85.2**	1.28	**1.41**	94.6	4310	28.7	**92.0**	1.02	**1.40**
CPDBig	88.4	6309	87.0	**98.6**	1.91	**2.10**	92.8	4966	62.9	**98.7**	1.39	**2.00**
CPDConv	84.5	4781	68.4	**72.0**	**1.08**	1.06	84.5	8752	52.6	**68.4**	0.99	**1.00**
CURE	44.5	4704	43.7	**85.3**	1.12	**1.24**	31.5	2890	33.0	**82.9**	1.03	**1.10**
PGD	92.6	5166	0.5	**88.7**	0.91	**1.17**	86.7	4247	0.0	**40.5**	0.87	**1.03**
Std	96.5	4925	56.0	**90.2**	1.21	**1.34**	89.7	4265	**0.0**	**0.0**	0.87	**0.98**

(b) CIFAR-10

Net	DeepZ						Box					
	V	T	Sub		Spd		V	T	Sub		Spd	
	(%)	(s)	E	F	E	F	(%)	(s)	E	F	E	F
CPDMed	97.6	5997	43.1	**60.9**	1.20	**1.25**	98.9	4935	33.6	**55.1**	1.13	**1.34**
CPDBig	95.2	7406	46.3	**68.4**	1.31	**1.45**	97.7	6032	62.8	**79.4**	1.50	**1.80**
CPDConv	85.3	3451	58.5	**84.5**	1.01	**1.08**	96.8	6721	67.7	**88.6**	1.01	**1.03**
CURE	94.8	7161	3.5	**31.5**	0.94	**1.02**	82.7	6031	0.5	**9.0**	0.90	**1.01**
PGD	58.1	5227	0.0	**10.2**	0.92	**1.05**	14.4	4298	**0.0**	**0.0**	0.86	**0.99**
Std	54.3	5071	**0.0**	**0.0**	0.90	**0.99**	13.0	4328	**0.0**	**0.0**	0.88	**0.99**

7.2 Efficacy of Automated Futility Detection

In scenarios where template reuse is detrimental, FastCert's automated futility detection effectively avoids slowdowns. Automated futility detection in FastCert can identify these scenarios and disable reuse, preventing slowdowns. Our profiling step figures out quickly that no layers are viable for reuse, returning empty layer-template configuration set (algorithm 1). This avoids the added overhead of template matching, resulting in verification times comparable to the baseline without reuse, as seen in tables 2 and 3. The remaining slowdown stems from template creation and matching against 200 L_0 queries per image during profiling, but this cost is negligible compared to the 75,000 verification queries per image. FastCert only incurs an average slowdown of 1.6% (ranging over $1\% - 4\%$), a significant improvement over the 10.1% (ranging over $1\% - 14\%$) average slowdown observed with extant template reuse. This is demonstrated across both Calzone and CoverD in fig. 5, where FastCert consistently outperforms extant reuse in futile scenarios.

Table 3: CoverD results (750K queries, 10 images, t=3) with DeepZ and Box domains. V=Verified(%), T=Baseline (No Template) Time in seconds, Sub=Subsumption rate (%), Spd=Speedup, E=Extant Template Reuse Strategy, F=FastCert.

(a) MNIST

Net	DeepZ						Box					
	V (%)	T (s)	Sub E	Sub F	Spd E	Spd F	V (%)	T (s)	Sub E	Sub F	Spd E	Spd F
CPDMed	89.9	4768	59.7	**85.8**	1.24	**1.35**	91.7	3850	32.2	**91.2**	1.07	**1.45**
CPDBig	87.5	5842	89.2	**98.7**	1.88	**2.04**	90.8	4916	60.1	**98.8**	1.33	**2.00**
CPDConv	84.9	4530	69.3	**72.3**	**1.02**	**1.02**	84.3	8602	53.7	**70.1**	1.04	**1.05**
CURE	43.3	4321	53.4	**89.8**	1.12	**1.24**	30.3	2925	41.8	**78.8**	1.08	**1.14**
PGD	89.4	4896	0.4	**86.5**	0.89	**1.09**	79.5	4323	0.0	**48.4**	0.86	**1.02**
Std	96.9	4716	62.8	**93.5**	1.25	**1.34**	89.4	4429	**0.0**	**0.0**	0.88	**0.96**

(b) CIFAR-10

Net	DeepZ						Box					
	V (%)	T (s)	Sub E	Sub F	Spd E	Spd F	V (%)	T (s)	Sub E	Sub F	Spd E	Spd F
CPDMed	96.5	6047	47.3	**68.5**	1.21	**1.29**	98.4	4858	37.6	**58.1**	1.12	**1.36**
CPDBig	95.4	7495	54.3	**75.4**	1.37	**1.53**	97.2	6314	63.4	**81.1**	1.51	**1.86**
CPDConv	84.4	3376	60.8	**84.1**	1.02	**1.04**	94.6	5508	75.0	**91.7**	1.00	**1.01**
CURE	95.9	7174	6.4	**40.4**	0.93	**1.05**	83.9	6203	0.1	**8.3**	0.91	**1.02**
PGD	58.1	5444	**0.0**	**0.0**	0.92	**0.99**	16.2	4645	**0.0**	**0.0**	0.88	**1.00**
Std	51.1	5362	**0.0**	**0.0**	0.91	**1.00**	13.1	4614	**0.0**	**0.0**	0.87	**0.99**

7.3 Advantage in Favorable Scenarios

In configurations where template reuse is effective, FastCert's automated selection of reuse layers and template counts improves over extant reuse, ultimately achieving an average of 1.13× (ranging over $0.98 \times -1.5\times$) performance gain overall.

Overall, FastCert achieves speedups of up to 2.10× on MNIST networks (e.g., CPDBig) and up to 1.86× on CIFAR-10 networks (e.g., CPDBig) compared to no template reuse (tables 2 and 3 and fig. 5). These peak improvements arise because FastCert (i) identifies layers with higher template subsumption and (ii) chooses template counts that increase hit rate while controlling template-matching overhead. For example, for a representative MNIST input on CPDMed for Calzone, FastCert applies two templates each at layers 2 and 3, and one template each at layers 4 and 5. In contrast, extant reuse relies on a static, manually chosen configuration; FastCert adapts reuse layers and template counts to the subsumption behavior of each network and dataset.

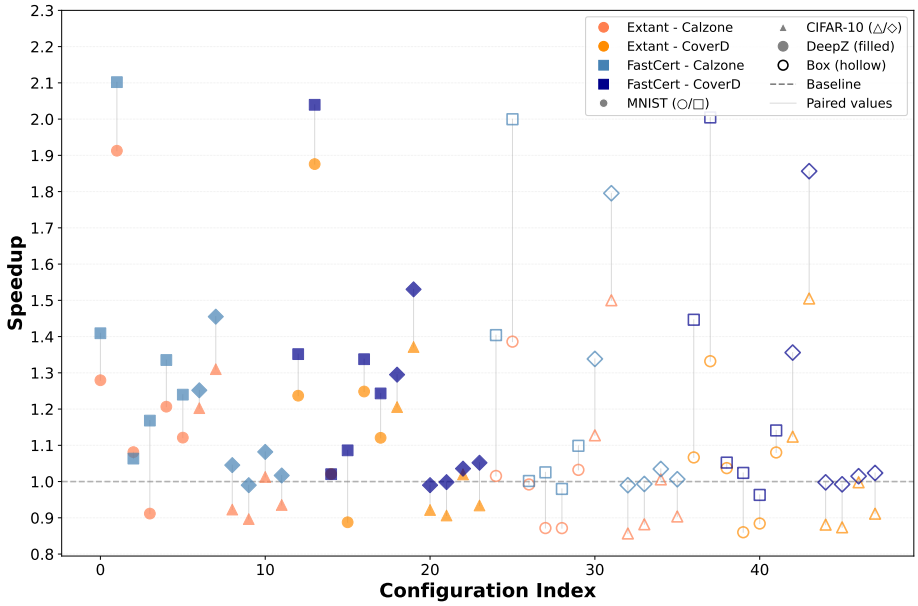


Fig. 5: Combined speedup results across all 48 configurations: FastCert outperforms extant reuse in 46, ties in one, and is slightly slower in one; the overall improvement is statistically significant according to a paired two-sided Wilcoxon signed-rank test ($W = 4$, $p = 3.1 \times 10^{-9}$).

In CPDConv the gains are minor since reuse occurs only at late linear layers followed by convolutional layers, which constrains subsumption potential and leaves little room for improvement [37].

7.4 Generality Across Abstract Domains

We evaluate FastCert with both DeepZ and Box, two abstract domains that are compatible with template reuse as identified in prior work [10]. We also experimented with another widely used abstract domain, Polyhedra (with DeepPoly [31]), but found them impractically slow at our evaluation scale: for example, on CPDMed (MNIST) under CoVerD with 750k queries and no template reuse, DeepPoly is 8.08× slower than DeepZ and 9.54× slower than Box. Accordingly, we focus our evaluation on DeepZ and Box.

From fig. 5, we observe that FastCert achieves almost similar speedup trends in both domains, with slightly better performance in Box. Specifically, FastCert achieves $0.98\times - 1.29\times$ speedup over extant reuse in DeepZ with an average of 1.10×, and $1.01\times - 1.5\times$ speedup over extant reuse in Box domain with an average of 1.17× across the tools and datasets. This shows that FastCert generalizes well across both the abstract domains.

7.5 Limitations

We study template reuse for robustness verification in the incomplete-verifier setting, and instantiate FastCert with the incomplete verification backends of Calzone and CoVerD. While these tools may ultimately invoke a complete procedure when the abstract domain cannot discharge a query at the leaf levels, FastCert operates only on the incomplete backend and is agnostic to any subsequent complete verification step. Our work is limited to verification queries generated from a single input image (local robustness). Extending template reuse across multiple inputs; for example, to support global robustness properties or batched/multi-input settings, is another promising direction. The performance model used in this work employs a constant per layer cost to derive viability threshold. Verification cost can depend on the layer type, size and on the abstract domain; although our end-to-end measurements capture this variation, extending FastCert with a layer-aware cost model may further improve its configuration decisions.

8 Related Work

DNN Verification. Existing DNN verifiers can be broadly classified as complete or incomplete [11, 24, 26, 28–31, 43, 45, 47]. Incomplete methods trade some precision for scalability, enabling the verification of larger and more complex networks that complete methods struggle to handle. Most of the prior works on DNN verification is focused on L_∞ verification [11, 30, 31]. Unlike L_∞ robustness, L_0 robustness induces a discrete, combinatorial, non-convex perturbation space, making scalable certification substantially more challenging. Calzone [26] and CoverD [28] address this challenge with covering designs, reducing t-pixel robustness to large collections of variable-size k-pixel queries with $k > t$; in practice, this can produce tens to hundreds of thousands of related queries per image. More recent work [27] advances this line by characterizing the convex hull of an L_0 ball as the intersection of its bounding box and a scaled L_1-like polytope. Prior template-reuse work demonstrates benefits on much smaller fixed-size L_0 workloads, but none study whether template reuse remains effective in covering-design setup.

Incremental Verification. Incremental verification has improved the scalability of traditional program verification to an industrial scale [14, 16, 23, 33]. Incremental program analysis tasks reuse partial results [44], constraints [39], and precision information [3] from previous runs for faster analysis of individual commits. Program changes typically affect a small, localized part of the code, whereas DNN updates modify weights across many layers without altering control flow. This key difference makes incremental DNN verification a distinct challenge in comparison to program analysis.

Similarly, many recent works [10, 36–38, 46] have successfully used incremental verification for improving the efficiency of DNN verification. These works include incremental complete verification [36, 46], incremental probabilistic verification [38] and incremental incomplete verification [10, 37]. Our work extends

this line by systematically analyzing when and how verification effort can be reused effectively, providing both theoretical and automated support for template reuse.

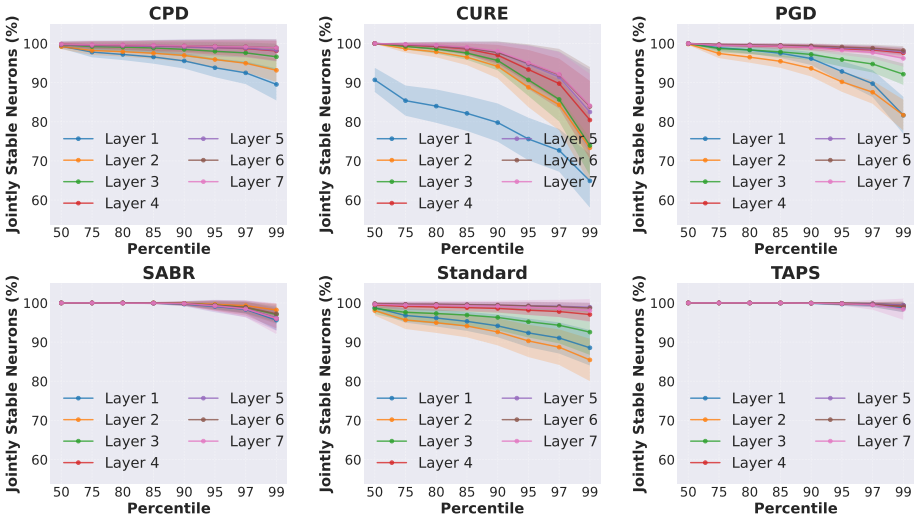
Wei et al. [40] considers incremental incomplete verification of relatively small DNNs with last-layer perturbation. Fischer et al. [10] introduced proof transfer to reuse verification effort across different input specifications of a single network, achieving up to 2.9x reduction in verification cost but with limited applicability to convolutional layers. Complementarily, Ugare et al. [37] extended this idea to transfer proofs across approximate networks by introducing a template transformation technique that enables sound and efficient verification for networks with convolutional architectures. Our work builds on these ideas by conducting a limit study of template effectiveness, identifying conditions where template reuse is counterproductive, and proposing an automated system that decides when, where, and how to apply template reuse efficiently.

9 Conclusions

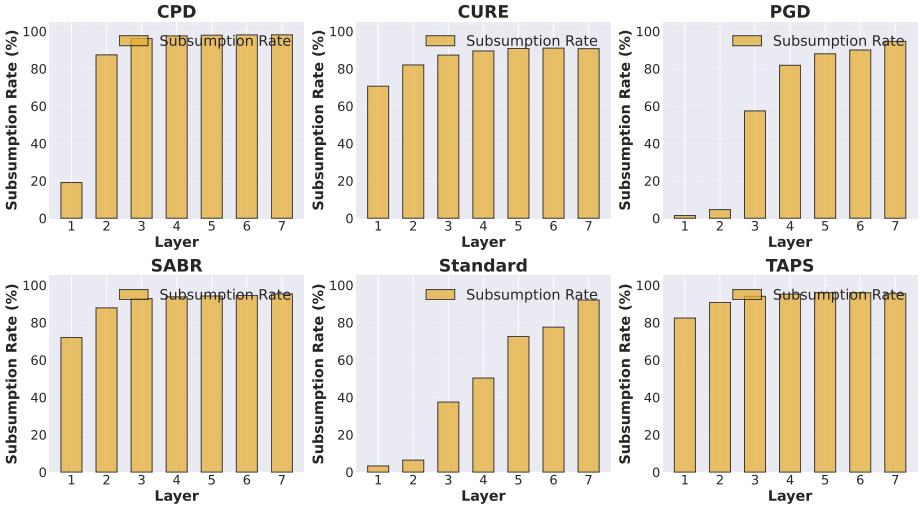
We have presented a study of the limits of template-based proof sharing for neural network verification. We showed that certain combinations of networks, training methods, and datasets are fundamentally unsuited to template-based speedups, due to excessive variance in jointly stable neurons. Then, we gave a new technique for automatically determining how best to apply templates to a verification task, including automatic detection of when templates should not be used. We implemented our technique in a tool FastCert, and an experimental evaluation showed that it improves over a state-of-the-art template technique, providing greater speedup in cases where templates work well and significantly reducing slowdown in cases where they are unsuitable.

Appendix 1 Joint Stability for Other Perturbations

Section 4.2 presented trends in joint stability and subsumption rates for L_0 perturbations. We observed similar joint stability and template subsumption trends across patch and geometric perturbations as well. In figs. 6 and 7, we plot the jointly stable neurons and template subsumption rates across different 2×2 patch perturbation scenarios in MNIST and CIFAR-10 networks. And in figs. 8 to 11, we plot the jointly stable neurons and template subsumption rates across geometric perturbations for MNIST networks. Here, $\pm 2^\circ$ rotation, $\pm 10\%$ contrast and $\pm 1\%$ brightness changes are split into $r = 4$, 6, 8, and 10 perturbations, respectively resulting in 64, 216, 512 and 1000 queries per image [2, 10]. The common trends of earlier layers having lower joint stability and subsumption rates, and later layers having higher joint stability and subsumption rates hold in these perturbation scenarios as well. The interesting cases observed in section 4.2; e.g., CIFAR-10 Standard and PGD networks, with earlier layers having higher joint stability but still very low subsumption rates in fig. 3 for L_0 perturbations are also observed in fig. 7 for patch perturbations. Similarly,



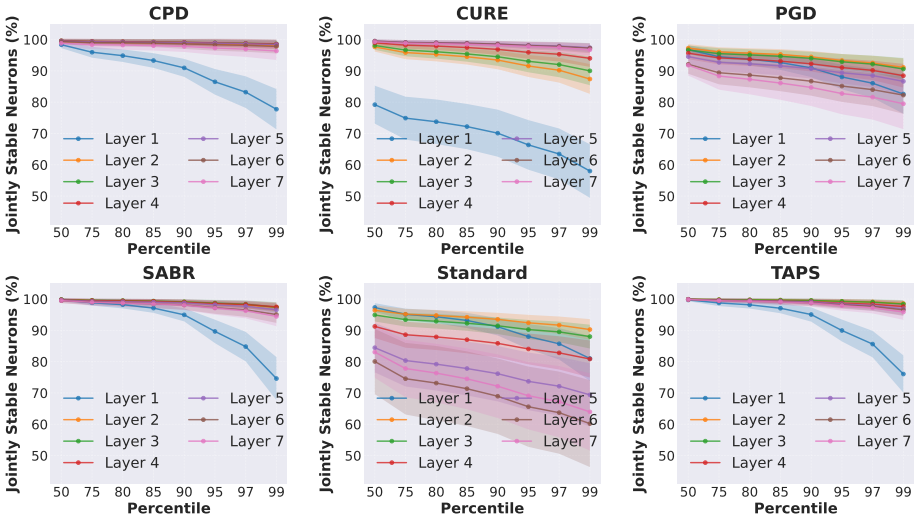
(a) Jointly stable neurons



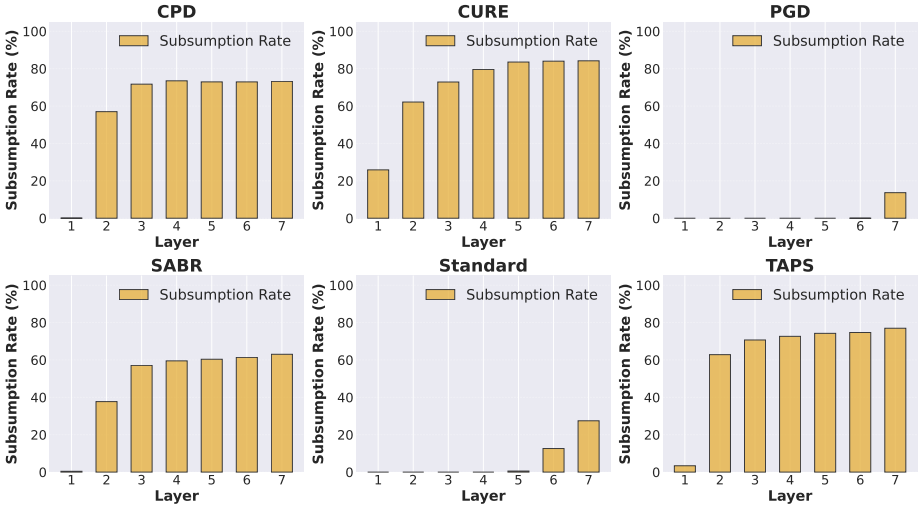
(b) Template subsumption rates

Fig. 6: Jointly stable neurons and template subsumption rates under 2×2 patch perturbations in MNIST networks.

trends reported for SABR, TAPS, and CURE MNIST networks in fig. 2 remain identical in the patch and geometric perturbation scenarios under figs. 6 and 8 to 11.



(a) Jointly stable neurons

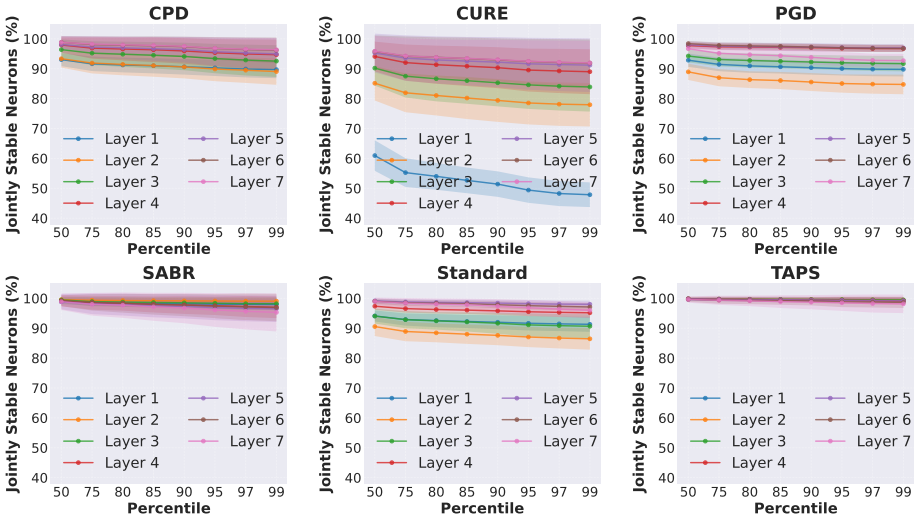


(b) Template subsumption rates

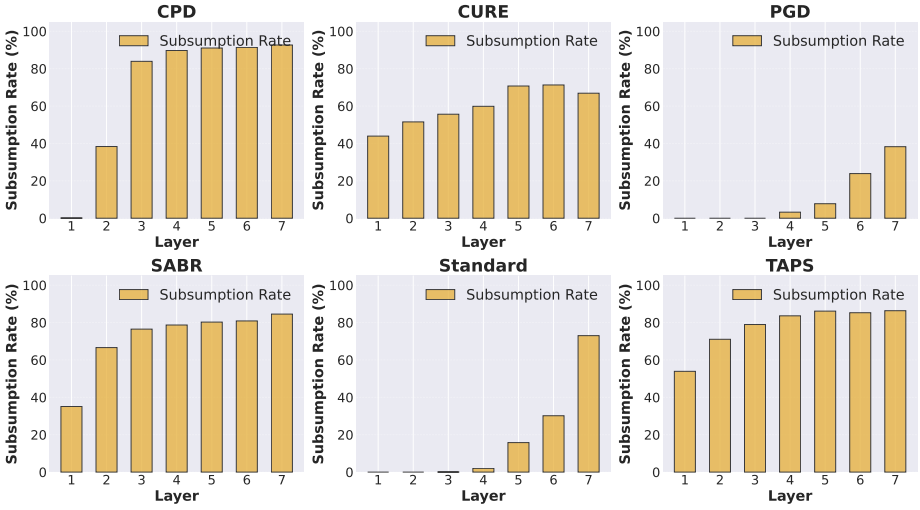
Fig. 7: Jointly stable neurons and template subsumption rates under 2×2 patch perturbations in CIFAR-10 networks.

Appendix 2 Sampling Sizes for Subsumption Estimation

Figures 12 to 14 fully detail our evaluation of how many sampled L_0 queries are needed to reliably estimate layer-wise subsumption for the algorithm of section 5. For each network, we select 10 input images (MNIST/CIFAR-10) and sample



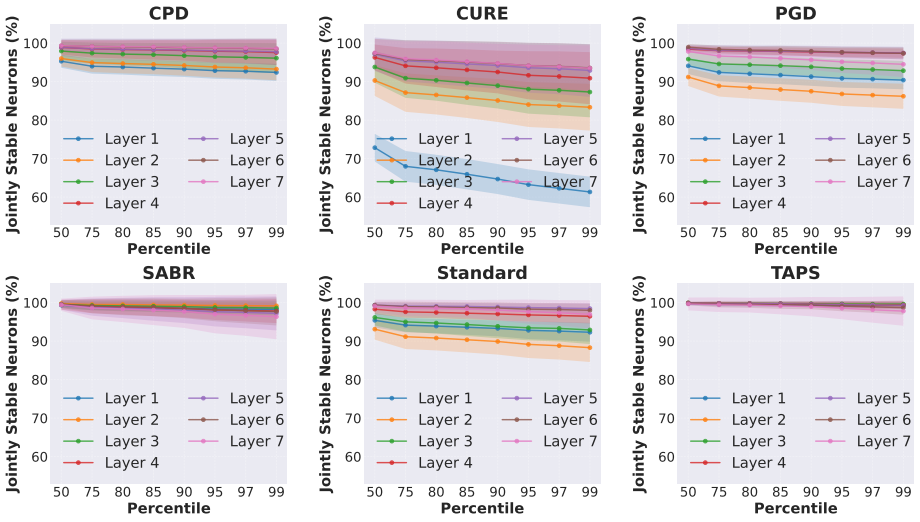
(a) Jointly stable neurons



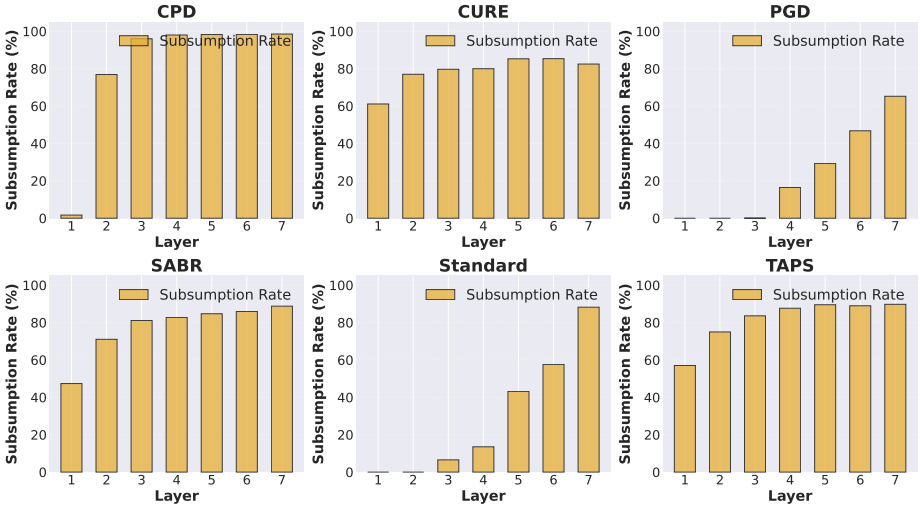
(b) Template subsumption rates

Fig. 8: Jointly stable neurons and template subsumption rates under geometric perturbations (4 splits) in MNIST networks.

verification queries by perturbing 5 to 20 randomly chosen pixels. For each layer, we plot the estimated subsumption rate against a fixed set of templates as a function of the number of sampled queries (10-600). For each configuration, we repeat this process 10 times and plot the mean (as solid lines) and standard deviation (as shaded regions). We also plot the minimum subsumption rate re-



(a) Jointly stable neurons

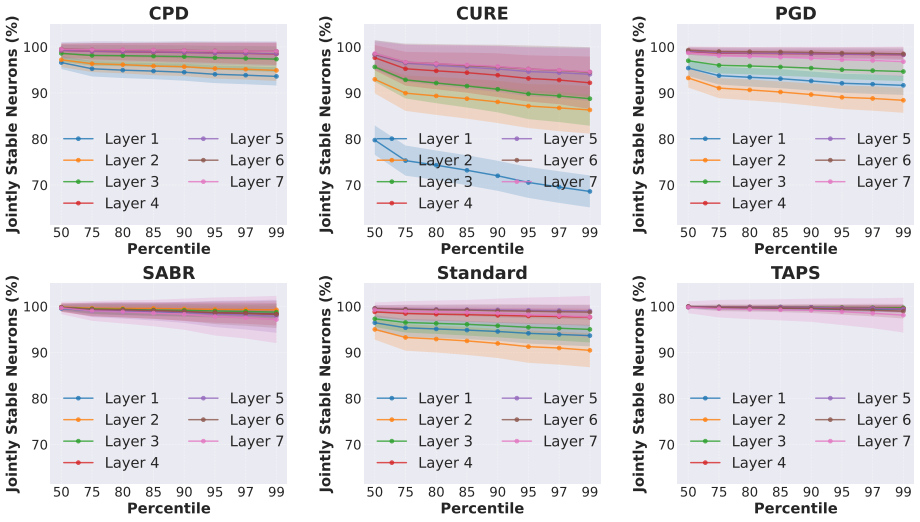


(b) Template subsumption rates

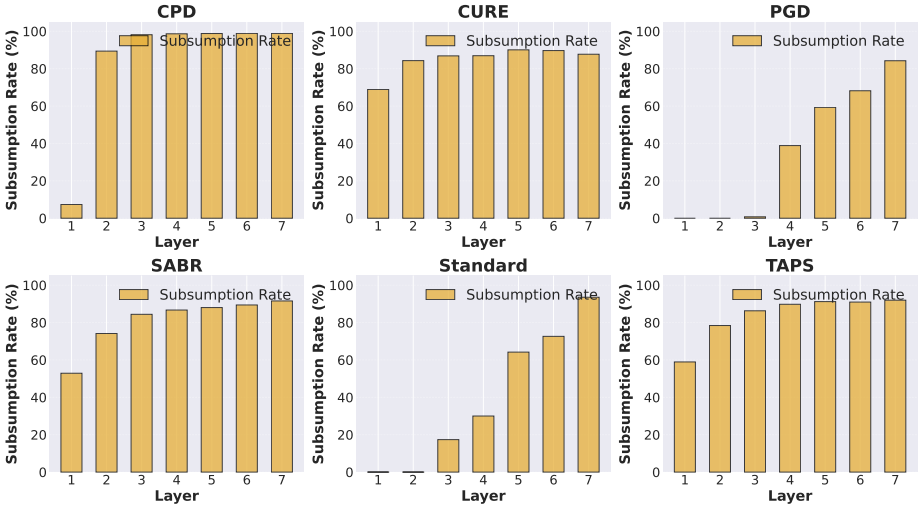
Fig. 9: Jointly stable neurons and template subsumption rates under geometric perturbations (6 splits) in MNIST networks.

quired to yield speedup (dotted), computed by FastCert's performance model (section 3.3). The plots are generated against different template counts of 1, 2, and 4.

Across networks and layers, subsumption estimates vary substantially for small samples (up to ∼150 queries), but stabilize around ∼200 queries. Impor-



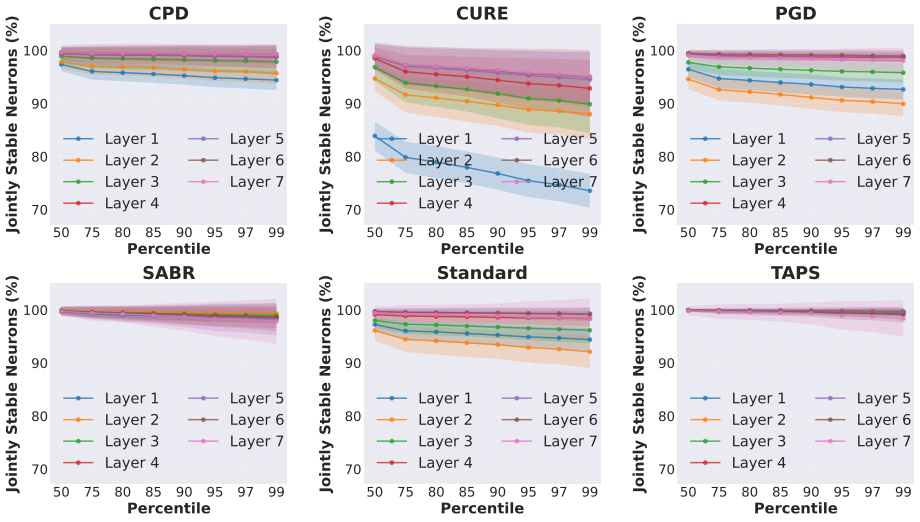
(a) Jointly stable neurons



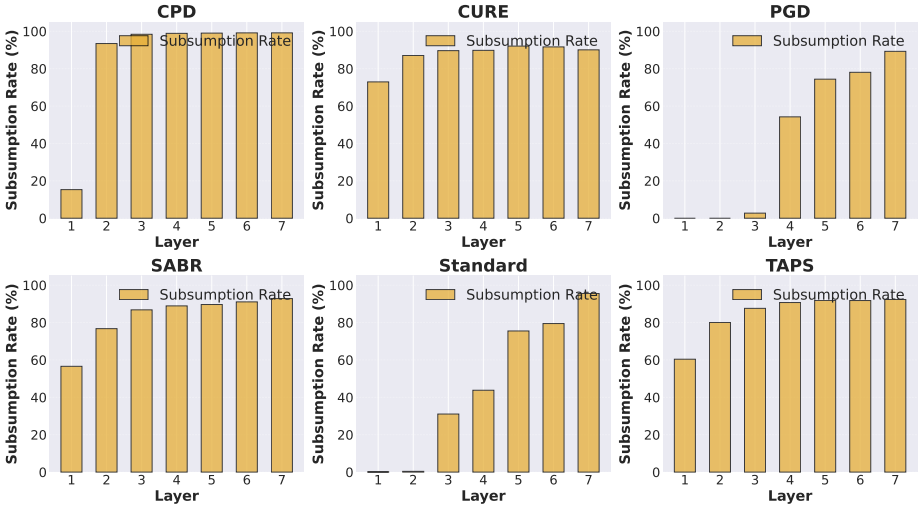
(b) Template subsumption rates

Fig. 10: Jointly stable neurons and template subsumption rates under geometric perturbations (8 splits) in MNIST networks.

tantly, the threshold-crossing decision is not sensitive to sampling variability beyond this point (e.g., fig. 4). While in rare cases, such as MNIST CPDMed with 2 templates fig. 13a, at layer 1, estimated subsumption falls very close to the threshold, even with larger samples of 600 queries, the variability remains



(a) Jointly stable neurons

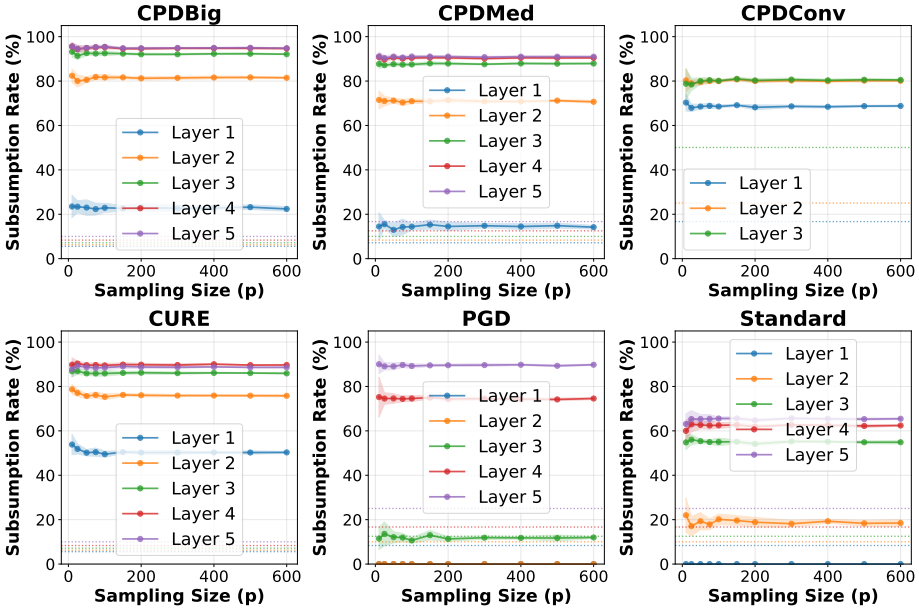


(b) Template subsumption rates

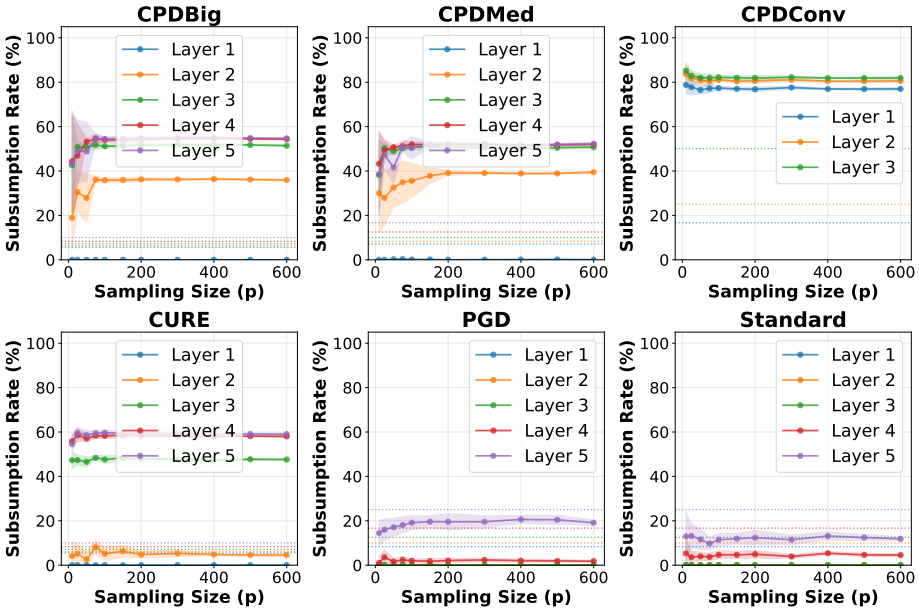
Fig. 11: Jointly stable neurons and template subsumption rates under geometric perturbations (10 splits) in MNIST networks.

similar. In almost all other cases, the gap between the estimated subsumption rate and the threshold to select or skip a layer for template reuse remains clear.

Thus we conclude that our choice of $P = 200$ queries per image for subsumption estimation strikes a good balance between profiling overhead and reliable layer selection.

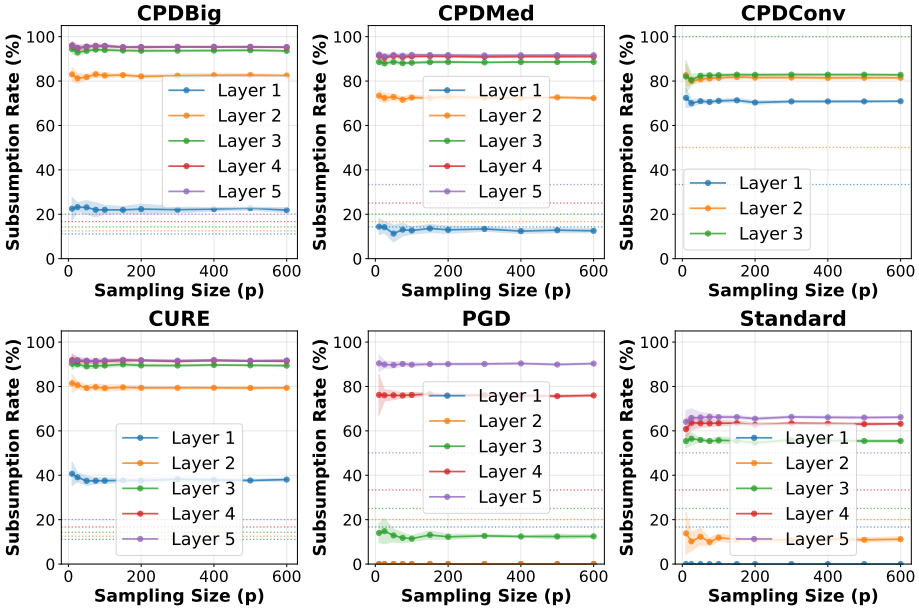


(a) MNIST

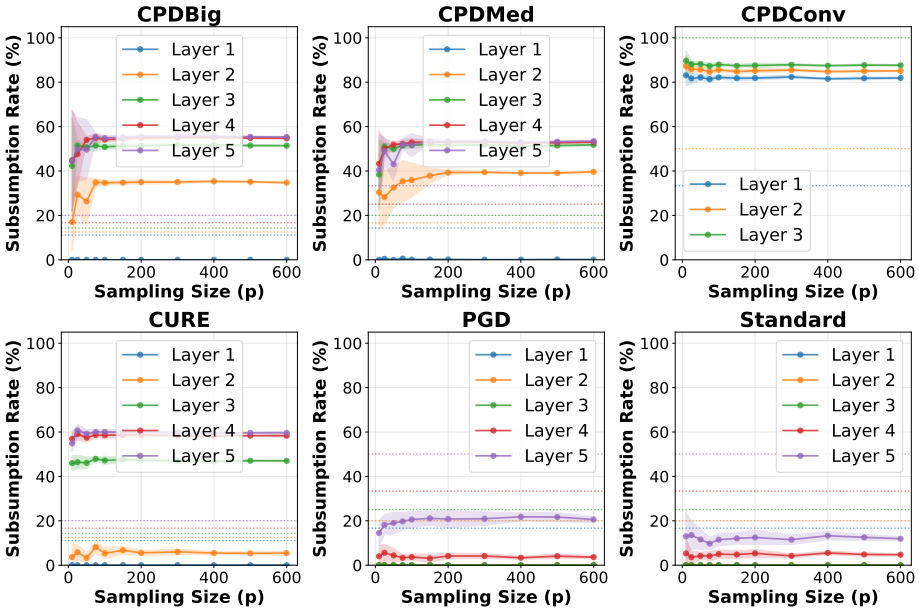


(b) CIFAR-10

Fig. 12: Comparing predicted minimum subsumption thresholds with observed subsumption rates across different sampling sizes with 1 template.

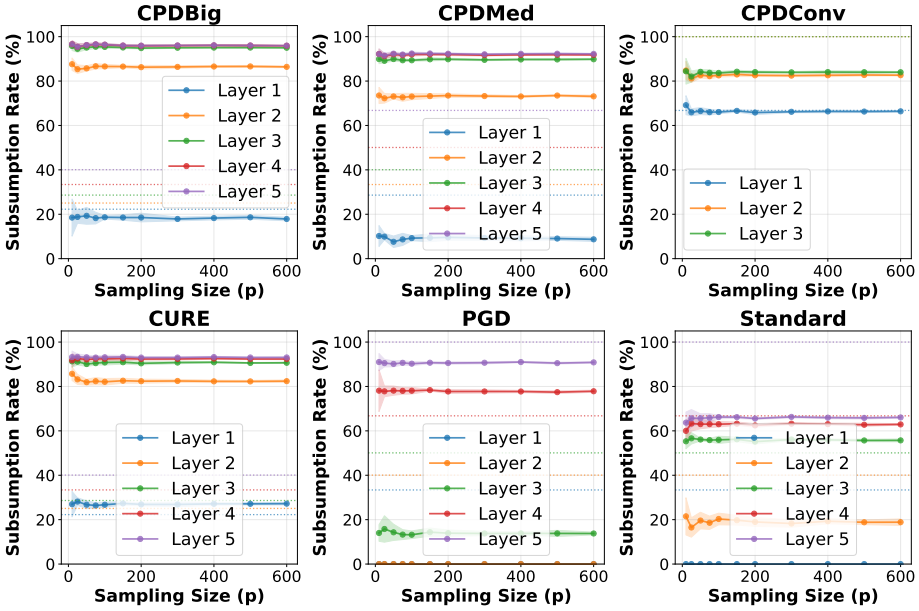


(a) MNIST

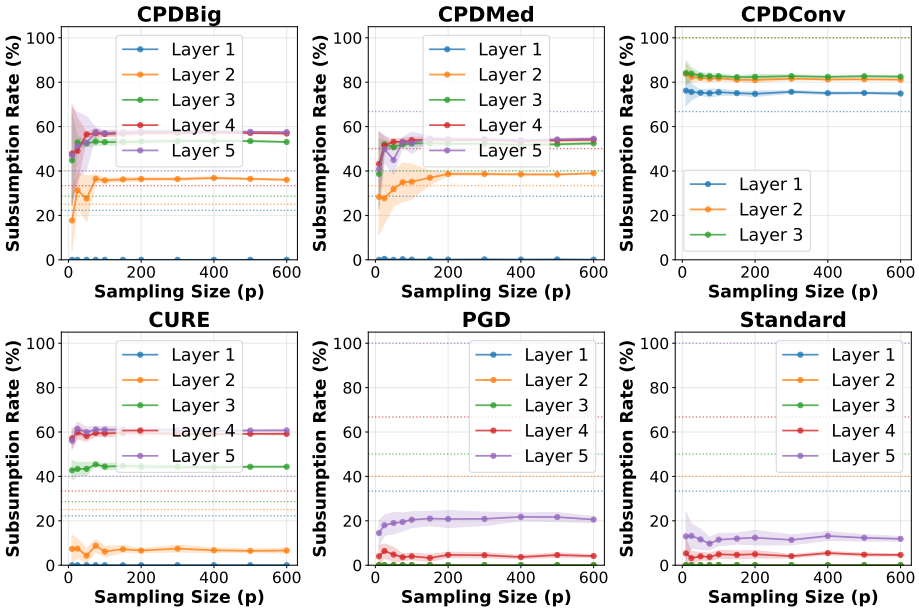


(b) CIFAR-10

Fig. 13: Comparing predicted minimum subsumption thresholds with observed subsumption rates across different sampling sizes with 2 templates.



(a) MNIST



(b) CIFAR-10

Fig. 14: Comparing predicted minimum subsumption thresholds with observed subsumption rates across different sampling sizes with 4 templates.

Artifact Availability. The artifact accompanying this paper, including the FastCert source code, networks, and experimental scripts, is available on Zenodo (DOI: 10.5281/zenodo.21314767).

Acknowledgments. This research was supported in part by the National Science Foundation under grants CCF-2238079, CCF-2313028, CCF-2223825, CCF-2223826, a gift from Oracle Labs, and a Google Research Award. The views expressed herein are those of the authors and do not necessarily reflect those of our funders.

Bibliography

[1] Albarghouthi, A.: Introduction to neural network verification (2021), https://arxiv.org/abs/2109.10317

[2] Balunovic, M., Baader, M., Singh, G., Gehr, T., Vechev, M.: Certifying geometric robustness of neural networks. Advances in Neural Information Processing Systems **32** (2019)

[3] Beyer, D., Löwe, S., Novikov, E., Stahlbauer, A., Wendler, P.: Precision reuse for efficient regression verification. In: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering. p. 389–399. ESEC/FSE 2013 (2013). https://doi.org/10.1145/2491411.2491429

[4] Botoeva, E., Kouvaros, P., Kronqvist, J., Lomuscio, A., Misener, R.: Efficient verification of relu-based neural networks via dependency analysis. In: Proc. of Advances in Artificial Intelligence (AAAI) (2020)

[5] Carlini, N., Wagner, D.A.: Towards evaluating the robustness of neural networks. In: Symposium on Security and Privacy (S&P) (2017)

[6] Chiang, P., Ni, R., Abdelkader, A., Zhu, C., Studer, C., Goldstein, T.: Certified defenses for adversarial patches. In: Proc. of International Conf. on Learning Representations (ICLR) (2020)

[7] Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proc. of Principles of Programming Languages (POPL) (1977)

[8] Croce, F., Andriushchenko, M., Singh, N.D., Flammarion, N., Hein, M.: Sparse-RS: A versatile framework for query-efficient sparse black-box adversarial attacks. In: Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI (2022)

[9] Eykholt, K., Evtimov, I., Fernandes, E., Li, B., Rahmati, A., Xiao, C., Prakash, A., Kohno, T., Song, D.: Robust physical-world attacks on deep learning visual classification. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1625–1634 (2018)

[10] Fischer, M., Sprecher, C., Dimitrov, D.I., Singh, G., Vechev, M.: Shared certificates for neural network verification. In: Computer Aided Verification: 34th International Conference, CAV 2022. p. 127–148. Springer-Verlag (2022). https://doi.org/10.1007/978-3-031-13185-1_7

[11] Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M.: Ai2: Safety and robustness certification of neural networks with abstract interpretation. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 3–18. IEEE (2018)
[12] Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. In: 3rd International Conference on Learning Representations, ICLR (2015)
[13] Jiang, E., Cheung, D.S., Singh, G.: Towards generalized certified robustness with multi-norm training. Trans. Mach. Learn. Res. **2026** (2026), https://openreview.net/forum?id=U5U7pazr6X
[14] Johnson, K., Calinescu, R., Kikuchi, S.: An incremental verification framework for component-based software systems. In: Proceedings of the ACM SIGSOFT Symposium on Component-Based Software Engineering. p. 33–42. CBSE '13 (2013). https://doi.org/10.1145/2465449.2465456
[15] Kotyan, S., Vargas, D.V.: Adversarial robustness assessment: Why in evaluation both l_0 and l_∞ attacks are necessary. PloS one **17**(4), e0265723 (2022)
[16] Lakhnech, Y., Bensalem, S., Berezin, S., Owre, S.: Incremental verification by abstraction. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 98–112. Springer (2001)
[17] Li, L., Xie, T., Li, B.: SoK: Certified robustness for deep neural networks. In: 44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, 22-26 May 2023. IEEE (2023)
[18] Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards deep learning models resistant to adversarial attacks. In: Proc. International Conference on Learning Representations (ICLR) (2018)
[19] Mao, Y., Müller, M., Fischer, M., Vechev, M.: Connecting certified and adversarial training. Advances in Neural Information Processing Systems **36**, 73422–73440 (2023)
[20] Mirman, M., Gehr, T., Vechev, M.: Differentiable abstract interpretation for provably robust neural networks. In: Proceedings of the 35th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 80, pp. 3578–3586. PMLR (10–15 Jul 2018), https://proceedings.mlr.press/v80/mirman18b.html
[21] Modas, A., Moosavi-Dezfooli, S.M., Frossard, P.: Sparsefool: a few pixels make a big difference. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 9087–9096 (2019)
[22] Müller, M.N., Eckert, F., Fischer, M., Vechev, M.: Certified training: Small boxes are all you need. In: International Conference on Learning Representations (2023), https://openreview.net/forum?id=7oFuxtJtUMH
[23] O'Hearn, P.W.: Continuous reasoning: Scaling the impact of formal methods. In: Dawar, A., Grädel, E. (eds.) Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018 (2018). https://doi.org/10.1145/3209108.3209109, https://doi.org/10.1145/3209108.3209109
[24] Salman, H., Yang, G., Zhang, H., Hsieh, C.J., Zhang, P.: A convex relaxation barrier to tight robustness verification of neural networks. Advances in Neural Information Processing Systems **32** (2019)

[25] Serra, T., Yu, X., Kumar, A., Ramalingam, S.: Scaling up exact neural network compression by relu stability. Advances in neural information processing systems **34**, 27081–27093 (2021)
[26] Shapira, Y., Avneri, E., Drachsler-Cohen, D.: Deep learning robustness verification for few-pixel attacks. Proc. ACM Program. Lang. **7**(OOPSLA1) (2023)
[27] Shapira, Y., Drachsler-Cohen, D.: Tight robustness certification through the convex hull of ℓ_0 attacks. Proceedings of the AAAI Conference on Artificial Intelligence **40**(44), 37913–37922 (2026). https://doi.org/10.1609/aaai.v40i44.41128
[28] Shapira, Y., Wiesel, N., Shabelman, S., Drachsler-Cohen, D.: Boosting few-pixel robustness verification via covering verification designs. In: International Conference on Computer Aided Verification. pp. 377–400. Springer (2024)
[29] Singh, G., Ganvir, R., Püschel, M., Vechev, M.: Beyond the single neuron convex barrier for neural network certification. Advances in Neural Information Processing Systems **32** (2019)
[30] Singh, G., Gehr, T., Mirman, M., Püschel, M., Vechev, M.: Fast and effective robustness certification. Advances in Neural Information Processing Systems **31**, 10802–10813 (2018)
[31] Singh, G., Gehr, T., Püschel, M., Vechev, M.: An abstract domain for certifying neural networks. Proceedings of the ACM on Programming Languages **3**(POPL), 1–30 (2019)
[32] Singh, G., Laurel, J., Misailovic, S., Banerjee, D., Singh, A., Xu, C., Ugare, S., Zhang, H.: Safety and trust in artificial intelligence with abstract interpretation. Found. Trends Program. Lang. **8**(3–4), 250–408 (Jun 2025). https://doi.org/10.1561/2500000062, https://doi.org/10.1561/2500000062
[33] Stein, B., Chang, B.E., Sridharan, M.: Demanded abstract interpretation. In: Freund, S.N., Yahav, E. (eds.) PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 2021. pp. 282–295. ACM (2021). https://doi.org/10.1145/3453483.3454044, https://doi.org/10.1145/3453483.3454044
[34] Su, J., Vargas, D.V., Sakurai, K.: One pixel attack for fooling deep neural networks. IEEE Trans. Evol. Comput. **23**(5) (2019)
[35] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I.J., Fergus, R.: Intriguing properties of neural networks. In: 2nd International Conference on Learning Representations, ICLR (2014)
[36] Ugare, S., Banerjee, D., Misailovic, S., Singh, G.: Incremental verification of neural networks. Proc. ACM Program. Lang. **7**(PLDI) (Jun 2023). https://doi.org/10.1145/3591299, https://doi.org/10.1145/3591299
[37] Ugare, S., Singh, G., Misailovic, S.: Proof transfer for fast certification of multiple approximate neural networks. Proc. ACM Program. Lang. **6**(OOPSLA1) (Apr 2022). https://doi.org/10.1145/3527319, https://doi.org/10.1145/3527319

[38] Ugare, S., Suresh, T., Banerjee, D., Singh, G., Misailovic, S.: Incremental randomized smoothing certification. In: The Twelfth International Conference on Learning Representations (2024), https://openreview.net/forum?id=SdeAPV1irk

[39] Visser, W., Geldenhuys, J., Dwyer, M.B.: Green: Reducing, reusing and recycling constraints in program analysis. In: Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering. FSE '12 (2012). https://doi.org/10.1145/2393596.2393665

[40] Wei, T., Liu, C.: Online verification of deep neural networks under domain or weight shift. CoRR **abs/2106.12732** (2021)

[41] Wong, E., Kolter, Z.: Provable defenses against adversarial examples via the convex outer adversarial polytope. In: International conference on machine learning. pp. 5286–5295. PMLR (2018)

[42] Wu, H., Isac, O., Zeljić, A., Tagomori, T., Daggitt, M., Kokke, W., Refaeli, I., Amir, G., Julian, K., Bassan, S., et al.: Marabou 2.0: a versatile formal analyzer of neural networks. In: International Conference on Computer Aided Verification. pp. 249–264. Springer (2024)

[43] Xu, K., Shi, Z., Zhang, H., Wang, Y., Chang, K.W., Huang, M., Kailkhura, B., Lin, X., Hsieh, C.J.: Automatic perturbation analysis for scalable certified robustness and beyond. Advances in Neural Information Processing Systems **33**, 1129–1141 (2020)

[44] Yang, G., Dwyer, M.B., Rothermel, G.: Regression model checking. In: 2009 IEEE International Conference on Software Maintenance. pp. 115–124 (2009). https://doi.org/10.1109/ICSM.2009.5306334

[45] Yang, R., Laurel, J., Misailovic, S., Singh, G.: Provable defense against geometric transformations. In: The Eleventh International Conference on Learning Representations (2023), https://openreview.net/forum?id=ThXqBsRI-cY

[46] Zhang, G., Zhang, Z., Bandara, H.D., Chen, S., Zhao, J., Sui, Y.: Efficient incremental verification of neural networks guided by counterexample potentiality. Proc. ACM Program. Lang. **9**(OOPSLA1) (Apr 2025). https://doi.org/10.1145/3720417

[47] Zhang, H., Weng, T.W., Chen, P.Y., Hsieh, C.J., Daniel, L.: Efficient neural network robustness certification with general activation functions. In: Advances in neural information processing systems (2018)