Verifying Structural Robustness of Deep Neural Network

ANONYMOUS AUTHOR(S)

Neural network verification has emerged as a useful technique for improving the reliability of deep learning systems. Current verification approaches primarily focus on local robustness, where perturbations are applied independently to each input element. Despite its common use, local robustness does not capture perturbations that exhibit coordinated relationships between input elements. Such perturbations arise from systematic transformations or filtering operations that preserve structural characteristics of the data. These perturbations, which we call "structural robustness", represent a significant gap in existing verification capabilities.

This work focuses on structural robustness verification by formalizing two important classes of structured perturbations: linear position-invariant and linear position-varying. Those perturbations allow input elements to be modified in coordinated ways while preserving essential data structure. The main challenge is that structural perturbations cannot be directly expressed using standard interval-based specification formats that existing verification tools typically support.

To address this limitation, we introduce VeriS, a technique that reformulates structural robustness into standard local robustness problems by creating specialized subnetworks that encode perturbation behavior and integrate them with the original network architecture. VeriS enables verification across continuous spaces defined by structural robustness specifications while maintaining compatibility with existing verification tools. VeriS also introduces optimizations that significantly enhance verification performance such as converting complex operations into standard representations.

We implement and evaluate VeriS on benchmarks involving neural networks across three domains: image classification, audio processing, and healthcare applications. Our evaluation, which encompasses 5508 verification problems, demonstrates that VeriS successfully verifies 78% of structural robustness specifications when integrated with state-of-the-art verification tools. These results show that VeriS enables the verification of complex structural perturbations that were previously beyond the reach of existing neural network verification.

Additional Key Words and Phrases: neural network verification, structural robustness, local robustness

ACM Reference Format:
Anonymous Author(s). 2025. Verifying Structural Robustness of Deep Neural Network. 1, 1 (September 2025), 21 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Deep Neural Networks (DNNs) are increasingly being employed as components of mission-critical systems across a range of application domains, including autonomous driving, medicine, and infrastructure monitoring. As with traditional software, testing DNNs using rigorous coverage criteria [10, 14, 24, 30, 33, 47, 56, 68] is necessary but not sufficient for critical deployments. To provide further assurance, researchers have developed a wide range of techniques for *verifying* that DNNs satisfy required properties. In recent years, many dozens of DNN verifiers have been reported in the literature and a yearly competition has documented advances in the capabilities of state-of-the-art DNN verifiers [2, 5, 6].

Among desired properties to evade adversarial attacks, robustness [3, 8, 39, 48] is a fundamental property for DNNs that ensures consistent behavior when inputs undergo perturbations. *Local*

ACM XXXX-XXXX/2025/9-ART
https://doi.org/10.1145/nnnnnnn.nnnnnnn

robustness (LR) is a common formulation that requires small, arbitrary changes to individual input elements, e.g., small noises added to each pixel of an image, do not affect the DNN predictions [25, 29]. To check that a neural network satisfies LR, existing analyses [1, 8, 18, 21, 28, 29, 34, 38, 39, 57, 59] define and encode LR specification using interval constraints that bound the perturbation added to each input, e.g., $\hat{x} = x + \delta$ where δ is a small noise bounded by an interval constraint. LR is well-studied and can be solved efficiently by modern DNN verifiers [2, 6, 15, 16, 64, 65], and LR benchmarks, e.g., image classification, made up a large portion of DNN verification evaluations [2, 5, 6].

Despite its simplicity and well-supported by existing work, LR does not capture real-world perturbations that create interdependencies between input elements and require more complex modifications than just adding noise to individual dimensions. For example, in image processing, common transformations include spatial translation and scaling [49], which systematically shift or resize pixel values rather than adding independent noise to each pixel. Scaling transformations in images uniformly adjust pixel coordinates, creating a zoomed view while preserving the semantic information, which should not affect a DNN's classification. Likewise, in audio processing, time-warping [11, 66] modifies playback speed without altering the essential acoustic content, representing natural variations that are expected to maintain the DNN's prediction.

In this work, we study robustness properties with two such types of perturbations that are common in real-world applications: *linear position-invariant* (LPI), which applies the same transformation across all input positions, e.g., filtering in audio processing and convolution in image processing [49], and *linear position-varying* (LPV), which applies different transformations at different input positions, e.g., time-warping in audio processing and spatial transformations in image processing [11, 66]. We call them *structural robustness* (SR) properties, since they capture how input data can change structurally while preserving input information. We introduce and formally define SR properties, which generalize LR properties, by using a more expressive form that allows a wide range of perturbations. Specifically, instead of adding small noise δ to each input element independently as in LR, we define SR properties with a noise transformation matrix P that captures the desired perturbation operations such as translation, scaling, time-warping, and filtering. This formulation allows analysis of new applications that require SR properties, e.g., classification with spatial transformations and temporal distortions.

Next, because SR properties are more complex (e.g., the transformation matrix involves non-linear operations) and not supported natively by DNN verification techniques, we introduce a reformulation approach, VeriS, that reduces SR verification problems into specialized LR ones. This process involves representing the structural perturbation as a new subnetwork that is prepended to the original network, whereas structural perturbation strength is encoded as interval constraints. The resulting network can then be verified using existing LR verification techniques.

Finally, for many SR transformations, the combined networks are larger and contain non-linear activation functions that existing verifiers do not handle well. Thus, VeriS introduces optimizations to improve the efficiency of verifying the new networks. For example, we show how to encode new activation functions, e.g., absolute operations that are not commonly used in DNNs, using ReLU operations that are well-supported by existing verifiers. VeriS also includes optimizations that reduce the size of the new network, e.g., by merging layers and removing redundant operations.

We implement VeriS and evaluate it on a set of 5508 SR problems including neural networks from three different domains (image classification, voice classification, and health monitoring). Our results show that through the reformulation and optimizations from VeriS enable existing DNN verifiers to solve 4289/5508 (78%) SR problems within a 60s timeout, while without VeriS none of the SR problems can be directly solved by existing verifiers.

The key contributions of the paper lie in:

- We introduce and formalize LPI and LPV SR properties, which are more expressive than LR and allow a wide range of perturbations. This enables new applications such as signal and audio processing and richer types of image classification that require SR properties.
- We show how to encode SR properties as LR properties, enabling existing verifiers to verify the considered SR properties. The main idea is to create a subnetwork that encodes the structural perturbation and integrates it into the original network.
- We introduce optimizations including layer merging and activation function encoding that allows existing verifiers to solve these new LR problems efficiently.
- We implement VERIS and evaluate it on 5508 SR problems from three different domains (audio, health, and image classification). We show that existing verifiers, with the help of VERIS, can effectively solve SR problems that were previously not considered and solved.

2 From Local to Structural Robustness

DNN Verification. Given a DNN N and a property (or specification) ϕ, the *DNN verification problem* asks if ϕ is a valid property of N. Typically, ϕ is a formula of the form $\phi_{in} \Rightarrow \phi_{out}$, where ϕ_{in} is a property over the inputs of N and ϕ_{out} is a property over the outputs of N.

Modern DNN techniques [1, 15, 16, 19, 59, 62, 64, 65] treat this verification problem as a satisfiability problem by encoding the DNN N and the property ϕ as a logical formula:

$$N \wedge \phi_{in} \wedge \neg\phi_{out} \tag{1}$$

If Eq. 1 is unsatisfiable (UNSAT), the considered property holds. Otherwise, it is satisfiable (SAT) and a counterexample exists that disproves the property.

2.1 Local Robustness (LR)

Existing DNN analyses mainly focus on LR properties[1], defined as follows:

DEFINITION 1 (LOCAL ROBUSTNESS). *Given a neural network $N : \mathbb{R}^d \longrightarrow \mathbb{R}^c$ and an input $x \in \mathbb{R}^d$, it is locally ϵ-robust at x with respect to norm $\|\cdot\|_p$ if:*

$$\forall \hat{x}, \quad \|x - \hat{x}\|_p \leq \epsilon \implies N(x) = N(\hat{x}).$$

where d is the input dimension, c is the number of outputs, and $\|x - \hat{x}\|_p \leq \epsilon$ indicates that the difference between the two points is within a certain (small) threshold ϵ.

This formulation treats each input dimension independently, allowing arbitrary element-wise modifications as long as the overall perturbation magnitude remains bounded. Typically, the perturbation space is defined by directly varying the input signal x within an ℓ_∞-ball:

$$\hat{x} = x + \delta \quad \text{where} \quad \|\delta\|_\infty \leq \epsilon$$

Note that DNN verification mainly focuses on LR properties of ℓ_∞ norm [2, 6, 15, 16, 25, 29, 64, 65]. More specifically, these work represent LR specifications using *interval* constraints, in which each input dimension of DNN is bounded by a lower bound and upper bound. This approach is simple and equivalent to ℓ_∞ norm ($p = \infty$) in Def. 1.

[1]The literature also mentions about *DNNs global robustness*, which requires that network maintain a separation of width at least ϵ (in input space) between any pair of regions that are assigned different prediction labels [31]. In other words, LR specifies space around a specific point of interest, while global robustness specifies space for every input. However, global robustness is often considered impractical as it is computationally intractable for continuous input spaces

Example. Assume we want to verify the DNN in Fig. 2a is locally ϵ-robust at $x = [1.0, 2.0, 3.0, 4.0]$ with respect to ℓ_∞ norm with $\epsilon = 0.5$. Then the interval constraints are:

$$\forall \hat{x}_1 \in [0.5, 1.5], \hat{x}_2 \in [1.5, 2.5], \hat{x}_3 \in [2.5, 3.5], \hat{x}_4 \in [3.5, 4.5] \implies N(x) = N(\hat{x})$$

where $\hat{x} = [\hat{x}_1, \hat{x}_2, \hat{x}_3, \hat{x}_4]$ is the perturbed input. This property ensures each input dimension is changed by at most 0.5 from its original value (in other words, perturbed inputs are within a ℓ_∞-ball of radius 0.5 around the center x) will produce the same output as the original input x.

2.2 Structural Robustness (SR)

LR properties do not capture perturbations that involve more complex interactions among input elements. We thus introduce structural robustness (SR) properties that generalizes LR and considers perturbations that affect the input in a structured manner.

In our observation from the literature [36, 40, 49], SR properties can be broadly categorized into two classes: linear position-invariant (LPI) and linear position-varying (LPV) perturbations. LPI specifications model systematic effects that apply uniformly across the entire input sequence, such as global filtering operations or environmental factors that consistently affect all data elements. Examples include lowpass and highpass filtering for audio signal processing [36] and blurring and sharpening for image processing [49]. LPV specifications capture localized structural distortions that vary depending on the position within the input sequence, such as timing variations or position-dependent transformations that affect different parts of the data differently. Examples include time-warping for time series data such as electrocardiogram (ECG) and audio [41, 50].

We formally define these two classes as follows:

Definition 2 (Linear Position-Invariant Perturbation). *Given an input $x \in \mathbb{R}^d$, a linear position-invariant (LPI) perturbation, characterized by a matrix $P \in \mathbb{R}^k$ where $k \leq d$, produces a perturbed input $\hat{x} \in \mathbb{R}^d$ through a convolution operation ($*$):*

$$\hat{x} = P * x$$

Example. Assume we want to perturb $x = [1.0, 2.0, 3.0, 4.0]$ following LPI perturbation using Echo kernel $P = [1.0, 0.0, 0.5]$. The perturbed input $\hat{x}$ can be computed as convolution of P and x:

$$\hat{x} = \underbrace{[1.0, 0.0, 0.5]}_{P} * \underbrace{[1.0, 2.0, 3.0, 4.0]}_{x} = [1.0, 2.5, 4.0, 3.0]$$

The input x is padded with 0.0 to the left and right to make output length of $\hat{x}$ is the same as x.

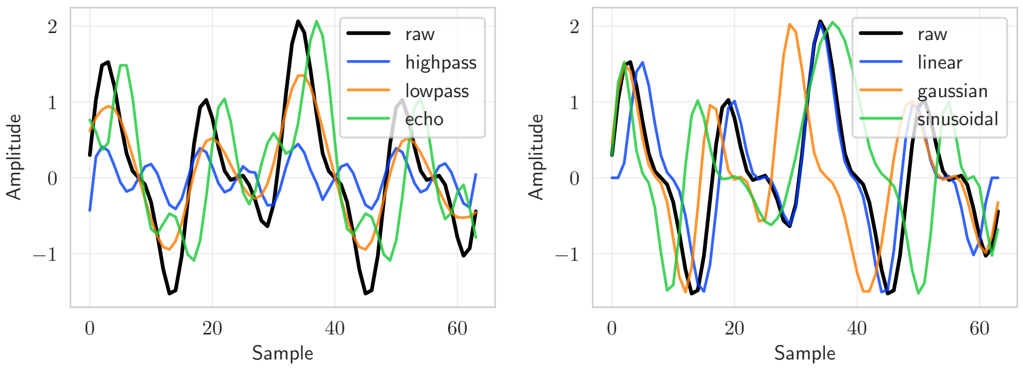
Definition 3 (Linear Position-Varying Perturbation). *Given an input $x \in \mathbb{R}^d$, a linear position-varying (LPV) perturbation, characterized by a matrix $P \in \mathbb{R}^{d \times d}$ where each row of P adds up to 1, produces a perturbed input $\hat{x} \in \mathbb{R}^d$ through a linear operation:*

$$\hat{x} = Px^T$$

Example. Assume we want to perturb $x = [1.0, 2.0, 3.0, 4.0]$ following LPV perturbation using Sinusoidal offset $c = [0.0, 2.0, 0.0, -2.0]$. The perturbed input $\hat{x}$ can be computed as:

$$\hat{x} = \underbrace{\begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \end{bmatrix}}_{P} \underbrace{\begin{bmatrix} 1.0 \\ 2.0 \\ 3.0 \\ 4.0 \end{bmatrix}}_{x^T} = [1.0, 4.0, 3.0, 2.0]^T$$

where the matrix P is constructed from the offset c as shown in Eq. 6.



(a) Position-invariant perturbation. (b) Position-varying perturbation.

Fig. 1. Differences between LPI and LPV perturbations using different perturbation types.

Fig. 1, which are the same types of filters as shown in the examples above, illustrates the differences between LPI and LPV used in signal processing. Fig. 1a shows lowpass and highpass perturbations that can be modeled using LPI perturbations, while Fig. 1b shows the LPV perturbation with different types representing Gaussian and sinusoidal methods. The key distinction is that LPI perturbations alter data characteristics (e.g., amplitude, shape, etc.) while LPV perturbations capture dynamic structural distortions and preserve local data characteristics. Due to uniform effects, LPI is often used in signal processing applications (because filters affect signals consistently) and image processing applications (because transformations affect images uniformly), while LPV is often used in time series analysis (because patterns vary locally).

Assumptions. For LPI, we assume P is scaling proportionally to z and for LPV, we assume c is changing proportionally to z. In other words, the perturbation space represented by VeriS formulation might not contain some patterns that values of P (or c) varies arbitrarily. One solution is to use more than one variable $[z_1, z_2, \ldots, z_n]$ to control the values of P separately. However, this approach does not guarantee some constraints among values of P (e.g., summing to 1) and might return some spurious solutions as encountered in [37].

2.3 Challenges and Approach Overview

SR can capture more complex and realistic perturbations than LR, but verifying SR properties presents two fundamental challenges that prevent the direct application of existing DNN verifiers. We address them through a two-step process: (i) reformulating SR properties into LR ones and (ii) optimizing the unique structure of the reformulated problems to make them more amenable to existing verifiers.

2.3.1 Challenge 1: Reformulation. LR properties can be formulated using interval constraints to represent the bounded ranges for additive noise δ and therefore are supported by existing DNN verifiers. In contrast, SR creates interdependencies between input elements (e.g., summation of each row of P to 1) and cannot be represented by intervals, and therefore are not supported by any existing verification tools.

To address this challenge, VeriS reformulates SR specifications in two steps (i) creating a perturbation network and (ii) integrating it into the original network. From the given SR specification, VeriS creates a new perturbation function P_z that represents a series of linear and non-linear functions. The operations in P_z also vary with respect to a single variable $z \in [0, 1]$ that controls how the perturbation is applied.

Next, VeriS represents P_z as a perturbation network and prepend it into the original network to create a new network $N \circ P_z$ to be verified. Fig. 2 shows an example of a DNN with 4 inputs and its modified version with a prepended perturbation network P_z and on input z that controls how the perturbation is applied.

Thus, VeriS transformed the original SR problem (N, P_z) into an LR one $(N \circ P_z, \phi)$, where the property ϕ is defined as:

$$\phi \equiv \forall z \in [0, 1] \implies (N \circ P_z)(z) = N(x) \tag{2}$$

Thus, ϕ is an LR and asks whether the network $N \circ P_z$ produces the same classification result as the original network N on input x for all possible perturbations done to x as controlled by z.

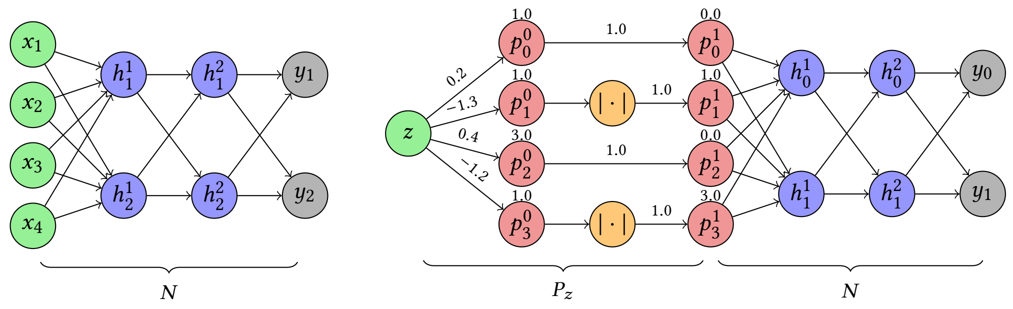
2.3.2 Challenge 2: Optimizations. While the newly formed LR problem can now be run by existing verifiers, it is quite unique and complex (e.g., with non-linear operations in the perturbation network P_z). Existing DNN verifiers was never designed for this kind of LR problem and in fact was not able to solve any benchmark problem of this form (e.g., in §5.4).

To address this limitation, we develop two new optimization techniques that enhance the verification process. First, VeriS combines multiplication and addition into a single FC layer to simplify the perturbation subnetwork P_z, making it easier to reason about. Second, VeriS transforms absolute operations (e.g., $|x|$, which is not a standard activation function and thus verifiers, or more specifically, abstraction domains used by verifiers, are not optimized to handle and become imprecise over large network) into an equivalent standard ReLU activation that existing verifiers are more comfortable with. These optimizations make the perturbation subnetwork P_z more compatible with existing DNN verifiers (e.g., all verifiers are optimized to support FC layer and ReLU natively).

2.4 Illustration Example

Consider an example where we have a simple network N with 4 inputs and 2 outputs as shown in Fig. 2a. We want to verify its SR on the input $x = [1.0, 2.0, 3.0, 4.0]$ when it is perturbed. For illustration, we use a *time-warping* perturbation, a transformation often used in time series based applications like signal processing and audio [36, 66] that shifts each input by an offset and compresses or stretches the input sequence. Assume the perturbation is characterized by an offset array $c = [0.2, -1.3, 0.4, -1.2]$. For example, after the perturbation we might have $x = [1.2, 0.7, 3.4, 2.8]$ if we apply the full offset. Of course, this is only one possible perturbed input, and we want to verify N is robust for all possible perturbed inputs that can be generated by this time-warping perturbation.

Reformulation. VeriS converts the given SR problem into an LR one as follows. First, it represents the perturbation as a small subnetwork P_z, which can also be interpreted as a function, that takes a single input $z \in [0, 1]$ to control how the perturbation is applied. For example, if $z = 0$, then no perturbation, and if $z = 1$, then the full offset c is applied. To construct P_z, VeriS uses a generic interpolation function $\psi(k) = \max\{0, 1 - |k|\}$ [40, 49] to compute the weighted sum ($\psi(k)$ ensures that the weights are between 0 and 1) and shifted the inputs using operations including additions and multiplications (e.g., $1.0 + 0.2z, 3.0 + 0.4z$), and absolute functions (e.g., $|1 - 1.3z|$). For this



x_1 x_2 x_3 x_4 h_1^1 h_2^1 h_1^2 h_2^2 y_1 y_2
N
z 0.2 −1.3 0.4 −1.2
p_0^0 p_1^0 p_2^0 p_3^0 1.0 1.0 3.0 1.0
$|\cdot|$ $|\cdot|$
p_0^1 p_1^1 p_2^1 p_3^1 0.0 1.0 0.0 3.0
h_0^1 h_1^1 h_0^2 h_1^2 y_0 y_1
P_z N

(a) Original network (b) Perturbed network

Fig. 2. A new network $N \circ P_z$ with $x = [1.0, 2.0, 3.0, 4.0]$ and $c = [0.2, -1.3, 0.4, -1.2]$.

example, the fully constructed P_z is

$$P_z = \underbrace{\begin{bmatrix} 1.0 - 0.2z & 0.2z & 0.0 & 0.0 \\ 1.0 - |1.0 - 1.3z| & |1.0 - 1.3z| & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 - 0.4z & 0.4z \\ 0.0 & 0.0 & 1.0 - |1.0 - 1.2z| & |1.0 - 1.2z| \end{bmatrix}}_{P} \underbrace{\begin{bmatrix} 1.0 \\ 2.0 \\ 3.0 \\ 4.0 \end{bmatrix}}_{x^T} = \begin{bmatrix} 1.0 + 0.2z \\ 1.0 + |1.0 - 1.3z| \\ 3.0 + 0.4z \\ 3.0 + |1.0 - 1.2z| \end{bmatrix}$$

VeriS represents P_z as a subnetwork parameterized by a single input z and operations including multiplications and additions represented as weights and biases, and absolute functions represented as activation functions, as shown in Fig. 2b. The new problem is now LR as shown Eq. 2 which checks if $N \circ P_z$ produces the same output as the original network N on input x for all perturbations parameterized by z.

Optimizations. While existing DNN verifiers can now run the new LR problem, they could not solve it because of the complexity of the subnetwork P_z, e.g., absolutes are non-standard nonlinear activation functions that are difficult to analyze. To address this challenge, VeriS applies two optimizations to simplify P_z architecture. VeriS first combines the multiplication and addition into a single fully-connected (FC) layer (instead of multiple layers in the original P_z subnetwork). For example, VeriS simplifies the following equation into a single FC layer:

$$\begin{bmatrix} 1.0 + 0.2z \\ 1.0 - 1.3z \\ 3.0 + 0.4z \\ 1.0 - 1.2z \end{bmatrix} = z \underbrace{\begin{bmatrix} 0.2 & -1.3 & 0.4 & -1.2 \end{bmatrix}^T}_{W} + \underbrace{\begin{bmatrix} 1.0 \\ 1.0 \\ 3.0 \\ 1.0 \end{bmatrix}}_{b} = Linear_{W,b}(z)$$

Next, VeriS converts the absolute function (e.g., $|1 - 1.3z|$) into a series of standard ReLU activation functions, which are natively supported by existing verifiers.

$$|1 - 1.3z| = ReLU(1 - 1.3z) + ReLU(-(1 - 1.3z)) \quad (3)$$

After these optimizations, we have a simpler perturbation subnetwork P_z that consists of only FC layers and ReLU activations, which are well-supported by existing verifiers. For the running example, which originally was not solvable by existing verifiers, is now easily solved them (proven valid by both the $\alpha\beta$-Crown [59, 64, 65] and by NeuralSat [15, 16] verifiers).

We describe the general VeriS algorithmic approach in the next section §3 and evaluate it in §5.

Alg. 1: VeriS Verification Framework

```
input   : DNN $N$, input $x$, structural robustness $SR$ ($P$ for LPI or $c$ for LPV), verifier $V$
output  : Verification result: sat, unsat, or timeout
   // Step 1: Construct perturbation subnetwork $P_z$
1  if $SR \equiv LPI$ then // LPI perturbation subnetwork construction (§3.1)
2      $W \leftarrow P * x - x$ ;                          // Construct weight matrix Eq. 5
3      $b \leftarrow x$ ;                                  // Construct bias vector Eq. 5
4      $P_z \leftarrow \text{Linear}_{W,b}$ ;              // Perturbation subnetwork construction
5  else if $SR \equiv LPV$ then // LPV perturbation subnetwork construction (§3.2)
6      $\psi(x) \leftarrow ReLU(1 - ReLU(x) - ReLU(-x))$ ;   // Convert operator optimization §3.3
7      $P_z \leftarrow \text{Linear}_{x,0} \circ \psi \circ \text{Sub}_j \circ \text{Linear}_{c,i}$ ;   // Perturbation subnetwork construction
   // Step 2: Formulate verification problem
8  $M \leftarrow N \circ P_z$ ;                            // Construct perturbed network
9  $\phi \leftarrow \forall(z) \in [0,1] : M(z) = N(x)$ ;   // Formulate verification property
   // Step 3: Verify the verification property $\phi$
10 return $V(\phi)$ ;                                      // Invoke oracle verifier $V$
```

3 The VeriS Approach

Alg. 1 presents the high-level workflow of VeriS's verification approach, which transforms SR verification problems into standard LR ones that existing tools can handle directly. The algorithm takes as inputs the target DNN N, an input x, a SR specification (either LPI characterized by P or LPV characterized by c), and an oracle verifier V. The algorithm returns possible verification outcomes: (1) **solved** (either *sat* meaning violation found or *unsat* meaning property verified), or (2) **unsolved** (unknown as runtime limit exceeded or an error occurred such as out-of-memory or implementation issue).

First, VeriS constructs a perturbation subnetwork P_z that encodes the SR specification (line 1-line 7). For LPI specifications, the VeriS computes weight W and bias b from the perturbation parameters and given input, then creates a single FC layer $\text{Linear}_{W,b}$ that represents the desired perturbation (line 4) as explained in §3.1. For LPV specifications, the VeriS constructs a multi-layer subnetwork (e.g., FC and ReLU layers, see §3.2) that models the perturbation (line 7). This step also employs several optimization techniques to further improve the verification performance (see §3.3).

Next, VeriS transforms the SR problem into an LR one by combining the original network N with the perturbation subnetwork P_z to create a perturbed network $M \equiv N \circ P_z$ (line 8), with one single input z that controls the perturbation. The problem ϕ now is an LR and specifies that for all $z \in [0,1]$, the M's output must match the N's output on the original input x (line 9).

VeriS invokes the oracle verifier V to solve the formulated problem ϕ (line 10). Note that VeriS applies optimizations to simplify the networks being checked and make them compatible with existing verification tools. Moreover, VeriS uses LR representation to represent SR specification, reduces the number of input dimensions to 1, thus, making problems more manageable.

3.1 LPI Formulation

To model LPI perturbations, VeriS uses Def. 2 employs standard convolution operations to naturally represent uniform transformations across input. This formulation ensures the same transformation applies consistently across all positions.

To verify SR problem of different variants of noise matrix P for LPI specification, VeriS converts the convolution transformation to a perturbation subnetwork P_z taking as input a single variable

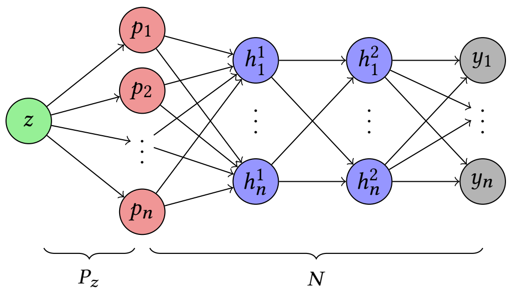


Fig. 3. LPI perturbation subnetwork construction.

$z \in [0, 1]$. The P_z is constructed to satisfy two essential properties: when $z = 0$, the transformation reduces to the identity operation (no perturbation), and when $z = 1$, it applies the target perturbation. Formally, P_z for LPI specification is formulated as:

$$P_z = z \cdot P * x + (1 - z) \cdot x = z \cdot (P * x - x) + x \tag{4}$$

This linear interpolation between the original input x and the fully perturbed input $P * x$ creates smooth and valid variation across the perturbation space.

To integrate the P_z into the original DNN being checked, VeriS leverages the fact that $P * x$ produces a fixed outcome given the input x and the perturbation matrix P being checked. The perturbation subnetwork P_z can then be transformed to a standard FC layer $\text{Linear}_{W,b}$ with weight matrix W and bias b as follows:

$$P_z = zW^T + b = \text{Linear}_{W,b}(z) \tag{5}$$

where $W = P * x - x$ and $b = x$.

Fig. 3 illustrates the construction of the P_z. This formulation uses a FC layer, which are universally supported by all DNN verification tools as the most fundamental layer in DNNs. While the underlying operations (multiplication and addition) appear simple, expressing them as a standard linear layer ensures broad compatibility across verification frameworks, as not all tools support standalone arithmetic operations.

3.2 LPV Formulation

Similar to LPI perturbations, VeriS uses Def. 3 to systematically model LPV perturbations, which employs interpolation-based transformations that redistribute input values across neighboring positions. VeriS can essentially model various types of LPV perturbations by constructing the noise matrix P using different interpolation ψ, where each specific perturbation type requires its own mathematical characterization. As a concrete example, this work demonstrates the approach using time-warping [11, 66], which has been widely used in practice [27, 36, 41, 50, 66, 67]. The time-warping formulation constructs the noise matrix P using two parameters: offset matrix c and interpolation function ψ. The offset matrix c determines the offset weights of input elements, and ψ determines the weight distribution between adjacent positions while ensuring weights remain between 0 and 1.

VeriS uses a generic interpolation $\psi(k) = \max\{0, 1 - |k|\}$ widely used in literature [40, 49] and varies the offset c by scaling it by z, denoted as $c_z = z \cdot c$. Formally, the noise matrix P for LPV specification is defined element-wise as:

$$P[i, j] = \psi(i + z \cdot c[i] - j) \tag{6}$$

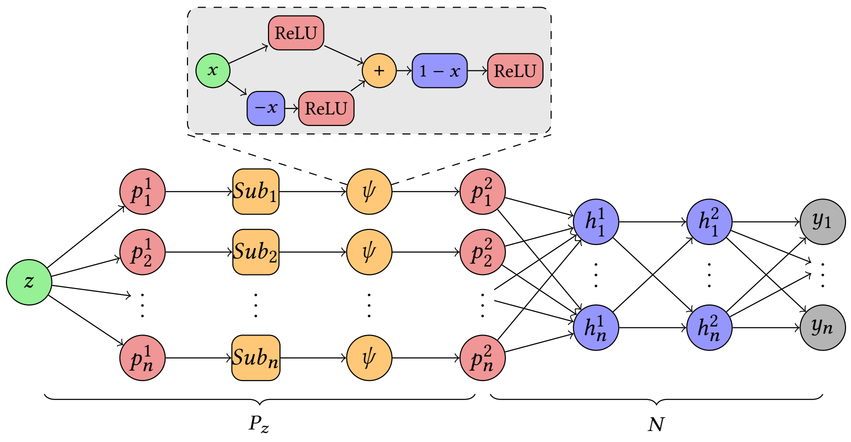


Fig. 4. LPV perturbation subnetwork construction.

Intuitively, $P[i,j]$ captures the weights of the neighboring positions j to the perturbed position i. Moreover, with this construction, P becomes the identity matrix when $z=0$, and target perturbation matrix when $z=1$, thus ensuring the desired properties of the perturbation subnetwork P_z.

For a given input x and offset c, the perturbation subnetwork P_z for LPV specification is constructed through a sequence of operations as:

$$P_z = Px^T = \text{Mul}_x \circ \psi \circ \text{Sub}_j \circ \text{Add}_i \circ \text{Mul}_c(z) \tag{7}$$

This operation sequence implements exactly Eq. 6 through a series of computational steps. Starting with parameter z, the operations compute $(i + z \cdot c[i] - j)$ for all pairs of indices i and j. The Mul_c and Add_i operations together compute $(i + z \cdot c[i])$ for each position i. Then Sub_j subtracts each j to produce the full matrix of differences. Finally, ψ converts these differences into interpolation weights, and Mul_x applies them to input x. Fig. 4 illustrates in detail the construction of the perturbation subnetwork P_z.

Compare to LPI perturbations in Eq. 5, the LPV perturbation subnetwork in Eq. 9 is more complex and involves more non-linear functions (e.g., absolute operation from ψ) that are generally more challenging for verification tools.

3.3 Optimization

VeriS also introduces several optimization techniques to further improve the verification performance. The construction of the perturbation subnetwork P_z involves several non-linear functions, such as absolute function ($|\cdot|$), which is generally not well-supported or not well-optimized by existing verification tools (e.g., $\alpha\beta$-Crown [64] and NeuralSat [15] fail miserably). To cope with this issue, VeriS introduces two optimization techniques to further improve the verification performance. We give a detailed description of the optimization techniques below.

3.3.1 *Merging Linear Operations.*

In Eq. 7 the perturbation subnetwork P_z is built through a sequence of operations as:

$$P_z = Px^T = \overbrace{\text{Mul}_x}^{\text{Linear}_{x,0}} \circ \underbrace{\psi \circ \text{Sub}_j \circ \overbrace{\text{Add}_i \circ \text{Mul}_c}^{\text{Linear}_{c,i}}(z)}_{P} \tag{8}$$

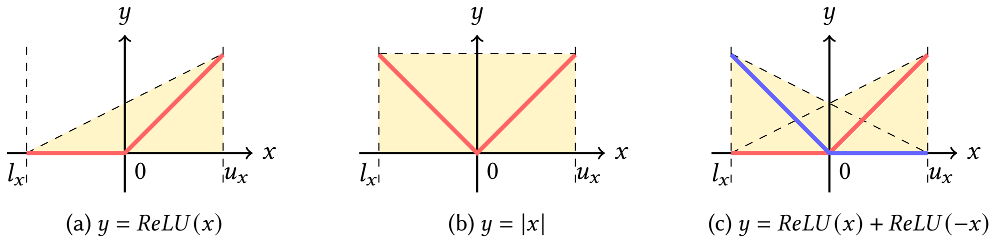


Fig. 5. Abstractions of ReLU and absolute function over $x \in [l_x, u_x]$.

The combination of scaling Mul_c and addition Add_i operations can be implemented as a standard FC layer $\text{Linear}_{c,i}$ (e.g., weight c and bias i), while the Mul_x can be converted to another FC layer $\text{Linear}_{x,0}$ (e.g., weight x and zero bias). Therefore, the perturbation subnetwork P_z is converted to:

$$P_z = \text{Linear}_{x,0} \circ \psi \circ \text{Sub}_j \circ \text{Linear}_{c,i}(z) \tag{9}$$

These simplifications make the perturbation subnetwork more compatible with existing DNN verifiers as FC is the most fundamental layer in DNNs reasoning. This also leverages the fact that existing DNN verifiers analyze the network layer by layer, so Linear counts as one layer while combining Mul and Add counts as 2, thus reducing the workload for verifiers.

3.3.2 Transforming Non-Linear Operations. The perturbation subnetwork P_z in Eq. 9 involves a non-linear interpolation function ψ, containing the absolute ($|\cdot|$). The absolute function is not a standard activation function and thus verifiers, or more specifically, abstraction domains used by verifiers, are not optimized to handle and become imprecise over large networks.

Fig. 5 illustrates the polytope abstractions for different cases. Fig. 5a shows one typical method to abstract ReLU [63] and Fig. 5b shows the abstraction for absolute function over $x \in [l_x, u_x]$, demonstrating the imprecision of the abstraction for absolute function. Fig. 5c shows the abstraction for combining $ReLU(x)$ and $ReLU(-x)$, which is equivalent to absolute function. Though it requires two separate abstractions for ReLU, it is more accurate than the abstraction for absolute function. Moreover, the abstraction of absolute function occurs early in the perturbed network (within the perturbation subnetwork P_z), the imprecision accumulates through out the entire network, making verifiers unable to solve the problem. More importantly, ReLU is common and well-optimized by verifiers, therefore, it scales much better than absolute function.

To cope with this issue, VeriS transforms the absolute operator into a standard ReLU activation function in a semantic-preserving manner. In particular, the absolute can be transformed into $|x| = ReLU(x) + ReLU(-x)$, and the interpolation function ψ in Eq. 9 can be transformed to:

$$\psi(x) = \max\{0, 1 - |x|\} = ReLU\ (1 - ReLU(x) - ReLU(-x)) \tag{10}$$

This transformation ensures that the absolute operation is preserved while being making existing DNN verifiers more comfortable with.

4 Experimental Design

We evaluate VeriS using the following research questions:
RQ1 (§5.1): How does VeriS perform on LPI and LPV perturbations?
RQ2 (§5.2): How does VeriS show robustness and vulnerability patterns?
RQ3 (§5.3): How compatible is VeriS with existing verification tools?
RQ4 (§5.4): How does optimization impact VeriS's performance?

Tab. 1. Benchmark instances.

Task	Network Type	Model	SR	Params	Neurons	Problems
KWS	FC + CNN + Pooling + ReLU	M3	LPI	39K	41K	540
			LPV	47K	48M	540
		M5	LPI	52K	41K	540
			LPV	60K	48M	540
ECG	FC + CNN + Pooling + ReLU	M3	LPI	36K	27K	432
			LPV	42K	22M	432
		M5	LPI	49K	28K	432
			LPV	54K	22M	432
Image	FC + CNN + ReLU	Oval21	LPI	112K	3K	540
	FC + ResNet + CNN + ReLU	Sri_Resnet_A	LPI	360K	11K	540
	FC + ResNet + CNN + ReLU	Cifar100	LPI	2.5M	55K	540
Total		**7**				**5508**

RQ5 (§5.5): How do structural perturbations compare to over-approximation approaches?

4.1 Verification Benchmarks

We use three domains to answer the RQs: (i) Keyword Spotting (KWS) for voice command recognition [audio], (ii) ECG classification for cardiac rhythm monitoring [health], and (iii) image and object recognition [image].

Network Datasets. Tab. 1 shows our networks, which comprise both domain-specific trained models and standard benchmark networks used in the literature and competitions [5] For KWS and ECG, we train M3 and M5 networks [12] which are deep CNNs for raw waveforms prediction task. We vary the number of channels for convolution layers in these networks to 32 and 64. We train KWS networks using the Google Speech Commands dataset [60], focusing on short utterances (approximately 1 second) of common voice commands recorded under diverse acoustic conditions. For ECG networks, we use the CardiacArrhythmia dataset [26], which provides cardiac rhythm data across four distinct arrhythmia classes. For image classification, we use pre-trained networks from recent VNN-COMPs [2, 5, 6]. These networks include Oval21, Sri_Resnet_A, and Cifar100 architectures, providing diverse baselines for evaluating structural robustness verification on computer vision tasks.

SR Specifications. Our benchmarks include both LPI and LPV perturbations across various perturbation levels and transformation configurations. For LPI specifications, we construct verification instances by defining specific kernels, e.g., Echo, Low-pass, and High-pass filters for audio and health domains, and Motion Blur kernels for image data. The perturbation space spans multiple kernel sizes and $z \in \{[0.0, 0.1], [0.0, 0.5], [0.0, 1.0]\}$ to capture diverse modification patterns.

LPV perturbations employ varying position matrices c defined by Linear, Sinusoidal, and Gaussian coefficient patterns with three lower intensity levels $z \in \{[0.0, 0.1], [0.0, 0.2], [0.0, 0.3]\}$. Due to the increased computational complexity inherent in LPV specifications, we employ these moderate perturbation intensities to ensure reasonable verification times while maintaining sufficient robustness assessment coverage.

Tab. 1 presents the benchmark statistics across all evaluation domains. The resulting benchmark contains 5508 problems that span diverse networks of sizes from 3K to 48M neurons. Note that we also list the number of neurons in addition to network parameters since the complexity of verification often depends on the number of neurons.

4.2 Verifiers and Experimental Setup

DNN Verifiers. We experiment VeriS using $\alpha\beta$-Crown [64, 65] and NeuralSat [15, 16], the two top performers in the recent VNN-COMP competitions. $\alpha\beta$-Crown has been consistently the winner in VNN-COMPs while NeuralSat is a new comer that ranked 2nd back-to-back in VNN-COMPs'24 [5] and '25 [55].

State-of-the-art DNN verifiers typically employ Branch-and-Bound (BaB) algorithm [7], in which "branch" refers to either neuron splitting or input splitting strategies to determine unsatisfiability or counterexamples. The former splits the hidden neuron boundaries during verification and performs abstraction to estimate bounds. The latter is often invoked on networks with low input dimensions, splitting the input space (instead of neurons) into smaller subspaces. For $\alpha\beta$-Crown, we use two different variants: neuron splitting $\alpha\beta$-Crown (N) and input splitting $\alpha\beta$-Crown (I). We use the default setting of NeuralSat as it automatically determines and switches between neuron and input splitting based on the input problem.

Experimental Envionment. Our experiments were run on a Linux machine with an Intel(R) Xeon(R) 8-core 2.20GHz CPU, 32GB RAM, and an NVIDIA L4 GPU with 24 GB VRAM.

We borrowed the timeout setting from recent VNN-COMPs [5, 6] which allows up to 6 hours per benchmark. For example, for the KWS M3 benchmark, the timeout can be up to $6 \times 3600/1080 = 20$ seconds per instance. To compensate for differences in platforms (CPU and GPU) used for evaluation, we settled down the timeout for each problem instance to 30 seconds for LPI instances and 60 seconds for LPV instances due to the increased complexity.

5 Results and Analysis

5.1 RQ1: VeriS performances on LPI and LPV perturbations

LPI Specifications. Tab. 2 presents the LPI verification performance of VeriS (used with the NeuralSat tool) when applied to KWS/ECG tasks with three different filters (Lowpass, Echo, and Highpass) and Image task under motion blur perturbations across three blur angles (0, 45, and 90 degrees). Overall, VeriS was able to solve 3342/3564 problems (94%).

Among the three tasks, ECG has a higher number of timeout instances (128 instances) compared to KWS (29 instances) and Image (65 instances). This difference can be attributed to the distinct characteristics of each data type. ECG signals are relatively unstructured, and filtering operations significantly alter the signal characteristics, creating diverse perturbation spaces that are challenging to verify. In contrast, KWS and image data have more structured representations that are less affected by filtering operations. Images maintain visual coherence after filtering, and audio signals remain interpretable for keyword recognition even when perturbed. Finally, and unsurprisingly, when the range of perturbation intensity z increases, the search space increases (e.g., $z \in [0.0, 0.1]$ vs. $[0.0, 1.0]$), and the number of solved instances decreases.

LPV Specifications. Tab. 3 shows that LPV problems are more challenging, with a total of 947/1944 (49%) solved instances in total. This performance degradation is expected since the networks of LPV perturbations with prepended subnetworks have many more neurons (e.g., 48M) compared to LPI ones (e.g., 41K). Still, despite the additional complexity for representing LPV characteristics, Tab. 2. Results on LPI (solved/unsolved)

Task	z	Lowpass	Echo	Highpass
ECG	[0.0, 0.1]	86/10	86/10	81/15
	[0.0, 0.5]	78/18	79/17	75/21
	[0.0, 1.0]	86/10	75/21	90/6
KWS	[0.0, 0.1]	120/0	120/0	120/0
	[0.0, 0.5]	120/0	113/7	117/3
	[0.0, 1.0]	116/4	108/12	117/3
Total		606/42	581/67	600/48
Task	**z**	**Blur 0**	**Blur 45**	**Blur 90**
Image	[0.0, 0.1]	180/0	180/0	180/0
	[0.0, 0.5]	177/3	163/17	179/1
	[0.0, 1.0]	162/18	172/8	162/18
Total		519/21	515/25	521/19

Tab. 3. Results on LPV (solved/unsolved)

Task	z	Linear	Sinusoidal	Gaussian
ECG	[0.0, 0.1]	85/11	73/23	82/14
	[0.0, 0.2]	79/17	59/37	65/31
	[0.0, 0.3]	66/30	23/73	35/61
KWS	[0.0, 0.1]	112/8	80/40	89/31
	[0.0, 0.2]	41/79	16/104	23/97
	[0.0, 0.3]	18/102	0/120	1/119
Total		401/247	251/397	295/353

Tab. 4. Results on LPI (unsat/sat/timeout)

Task	$z \in [0.0, 0.1]$	$z \in [0.0, 0.5]$	$z \in [0.0, 1.0]$
KWS	360/0/0	349/1/10	296/45/19
Image	523/17/0	374/145/21	202/294/44
ECG	248/5/35	153/79/56	77/174/37

Tab. 5. Results on LPV (unsat/sat/timeout)

Task	$z \in [0.0, 0.1]$	$z \in [0.0, 0.2]$	$z \in [0.0, 0.3]$
KWS	281/0/79	80/0/280	19/0/341
ECG	240/0/48	203/0/85	124/0/164

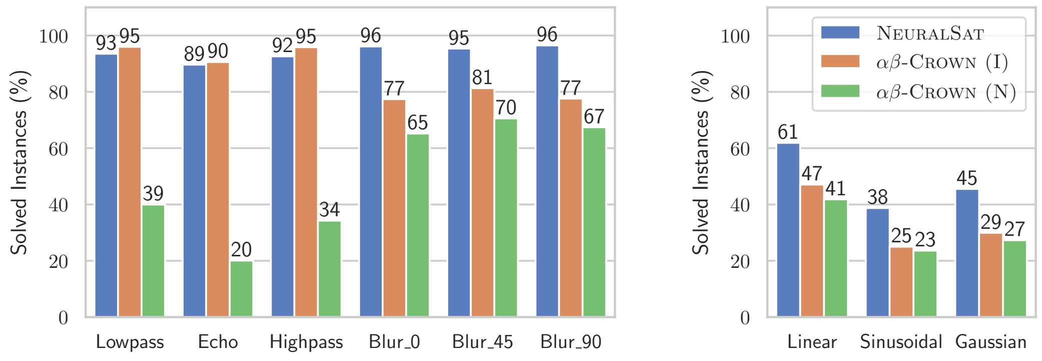
VeriS was able to solve 49% of LPV problems, which is significant given the novelty and difficulty of LPV verification that was not possible before.

A closer look reveals that VeriS performs on Linear problems better than Sinusoidal and Gaussian ones across all tasks and all perturbation configurations. This is due to Linear slightly perturbs the input compared to Sinusoidal and Gaussian (see Fig. 1b). More specifically, Linear marginally changes the input and creates a smaller perturbation space, in which the verification problems are easier to solve. On the other hand, Sinusoidal and Gaussian drastically alter the input, resulting in a larger perturbation space and thus their problems become harder to verify.

5.2 RQ2: Attacks and Patterns

The main goal of robustness verification is to show whether a network is robust or vulnerable to (adversarial) attacks. The results in §5.1 give the overall performance of VeriS and here we look closer into the results to determine vulnerable patterns and robustness of the networks. Recall that DNN verification tools return either `unsat` (the property is verified), `sat` (a counterexample is found, i.e., an adversarial example), or `timeout` (the tool is unable to solve the problem).

Tab. 4 presents the aggregated performance of LPI perturbations, revealing distinct vulnerability patterns among the three tasks. As the perturbation intensity z increases (i.e., more aggressive perturbations), all tasks exhibit the expected trend where problems become easier to attack and



(a) Time-invariant perturbations. (b) Time-varying perturbations.

Fig. 6. VeriS performances using different underlying verification tools.

harder to verify. KWS shows remarkable robustness, with no attack at $z \in [0.0, 0.1]$ and $z \in [0.0, 0.5]$ and only 45 attacks at maximum intensity. This resilience can be attributed to the structured nature of speech signals, where filtering operations preserve the essential acoustic features necessary for keyword recognition. In contrast, ECG demonstrates high vulnerability, with the number of attacks increasing from 5 to 174 instances as z grows.

Tab. 5 shows the verification results on LPV perturbations, revealing a different pattern compared to LPI results. Notably, no successful attacks were found across any task or perturbation intensity level, indicating that LPV perturbations used preserve the structural integrity of the input signals. However, as perturbation intensity increases, verification becomes increasingly challenging, with the number of verified instances decreasing and timeout instances growing substantially. KWS demonstrates particularly challenging verification characteristics, with verified instances dropping substantially from 281 to 19 as perturbation intensity increases. This difficulty stems from the longer input sequences in KWS tasks (4000) compared to ECG tasks (2714), which result in larger networks (e.g., 48M vs 22M neurons) when combined with LPV subnetworks.

5.3 RQ3: Compatibility with Existing Verification Tools

One of the contributions of VeriS lies in enabling existing verification tools to handle SR problems that were previously impossible to express or solve. Fig. 6 demonstrates this compatibility across different verifier configurations, though with varying degrees of success depending on perturbation complexity and verifier configurations.

For simpler perturbations like LPI, the transformation proves highly effective, with solved percentages reaching 95% for Highpass and Lowpass, and 96% for Motion Blur 0 and 90. However, more complex perturbations present significant challenges: while Linear LPV perturbations achieve moderate verification rates (up to 61%), Gaussian and Sinusoidal patterns exhibit lower rates due to their intrinsic computational complexity. Note that LPV problems are harder than LPI ones as LPV networks are a lot larger as in Tab. 1. This performance variation reflects the inherent difficulty of the underlying mathematical transformations rather than limitations in VeriS compatibility. The key achievement is that existing verifiers can now solve these structured robustness problems, whereas before VeriS such verification was impossible.

VeriS with backbone $\alpha\beta$-Crown worked well with input splitting (I) configuration, while neuron splitting (N) struggles to solve many problems. This performance pattern aligns with VeriS's

Tab. 6. VeriS performances on LPV perturbations (solved/unsolved)

Variant	**z** $\in [0.0, 0.1]$	**z** $\in [0.0, 0.2]$	**z** $\in [0.0, 0.3]$
Unoptimized	0/648	0/648	0/648
Optimized	521/127	283/365	143/505

formulation design, which reduces the effective input dimension to a single dimension z, making input splitting strategies particularly effective for space exploration. Conversely, NeuralSat, which automatically selects input or neuron splitting depending on the input problem, allowed VeriS to solve many problems and maintain high performance across perturbation types and domains.

5.4 RQ4: Effectiveness of VeriS Optimizations

We compare the performance of VeriS when it is unoptimized (i.e., the original formulation of the perturbation subnetwork P_z as shown in Eq. 7) and optimized (e.g., compressing layers and converting to ReLU as shown in Eq. 7). Note that we only show for LPV problems because the perturbation subnetwork P_z of LPI transformations already has just one linear layer (e.g., no activation function).

Tab. 6 shows that optimization is critically necessary. All unoptimized problems fail to solve within the time limit, with all 648 instances per perturbation level resulting in timeouts. In contrast, the optimized formulation successfully solves up to 80% (521 instances) at $z = 0.1$, 44% (283 instances) at $z = 0.2$, and 22% (143 instances) at $z = 0.3$. Performance degrades when perturbation strength increases because it creates a larger input space to explore, thus, problems are more challenging to solve. More specifically, larger z causes more imprecise abstraction, given that LPV problems inherently has many neurons to abstract (e.g., 48M), making the problems unsolvable. Additionally, as shown in Fig. 5, the abstraction of the absolute is less precise compared to the one using ReLUs, and the imprecision propagates through the network resulting in being unsolvable. This substantial improvement highlights how the ReLU conversion optimization transforms computationally intractable verification problems into solvable ones for existing verifiers.

5.5 RQ5: Comparison to Overapproximation Approaches

The abstraction-based approach in [37, 43] uses an over-approximation for verifying a subset of LPI properties for image classifiers. They work by computing the worst-case of the SR perturbation (overapproximated bounds of perturbed inputs) under some assumptions, e.g., assuming convolutional perturbation with the kernel values are from $[0, 1]$ and summing to 1 [37], or pixel-level perturbations under some spatial smoothness constraints [43]. In addition, the considered robustness is strictly less expressive than our LPI specification because it does not consider the constraints among kernel elements and interactions between kernels and input, which are crucial for structural perturbations.

To compare this approach with VeriS, we extend it to handle arbitrary kernels to capture VeriS's specifications with kernel bounds (K_{min}, K_{max}), where $K_{min} < 0 < K_{max}$. It computes upper (ub) and lower (lb) bounds for outputs by analyzing input neighborhoods of the kernel size, then computes bounds as:

$$\begin{aligned} ub &= \max\{neighbor, 0\} \times K_{max} + \min\{neighbor, 0\} \times K_{min} \\ lb &= \min\{neighbor, 0\} \times K_{max} + \max\{neighbor, 0\} \times K_{min} \end{aligned} \tag{11}$$

Tab. 7. Performances of Overapproximation and VeriS approaches on SR perturbations (unsat/sat/timeout)

Method	Lowpass	Echo	Highpass	Blur 0	Blur 45	Blur 90
Overapproximation	0/630/18	0/648/0	0/648/0	0/540/0	0/540/0	0/540/0
VeriS	485/121/42	485/96/67	513/87/48	386/133/21	330/185/25	383/138/19

where $\max\{neighbor, 0\}$ and $\min\{neighbor, 0\}$ are the positive and negative parts of the neighborhood, respectively. Intuitively, these equations perform a standard interval propagation for the output by considering the worst-case of the perturbation. The verification problem of over-approximation approach [37, 43] is then formulated as LR specification as:

$$\forall \hat{x} \in [lb, ub] \implies N(\hat{x}) = N(x)$$

The results in Tab. 7 using LPI benchmarks, which are the primary focus of the abstraction-based approach, show that the specifications generated by the over-approximation approach are all violations, e.g., counterexamples are found for all problems. It is due to either large intervals of inputs created by the over-approximation or the high-dimensional input space (e.g., the same as the original input size). Even for the smallest perturbation strength of 0.1, none of these properties could be verified for any networks considered in our evaluation. Note that those counterexamples are considered as *spurious* counterexamples since they do not comply with SR constraints. In contrast, VeriS was able to verify many properties across perturbation strengths and types. More importantly, when VeriS found counterexamples, they are all valid counterexamples that satisfy the SR constraints.

6 Threats to Validity

Regarding threats to internal validity, we built VeriS on top of established verification tools ($\alpha\beta$-Crown and NeuralSat) rather than implementing verification algorithms from scratch, thereby leveraging extensively tested codebases. We validated our algorithm through unit testing, including verification that identity transformations are produced when $z = 0$ and that maximum perturbations are produced when $z = 1$ for both LPI and LPV specifications.

Regarding threats to the generalizability of our results, our evaluation focuses primarily on audio, health and image domains. This domain selection was motivated by the natural applicability of SR, but it may limit the application of our work to other domains where different types of SR are relevant. Furthermore, our LPV evaluation was restricted to time-warping perturbations. Other LPV perturbations such as complex spatial transformations may exhibit different verification behaviors.

Regarding threats to the validity of our metrics and experimental design, we used standard verification metrics (number of solved instances, timeout, etc.) that are well-established in the DNN verification literature [2, 5, 6, 55], ensuring comparability with prior work. However, these metrics may not fully capture the practicality of SR verification compared to LR approaches. Our comparison in §5.5 relies on constructing interval bounds that may not represent the tightest possible approximation, potentially affecting the fairness of the comparison.

7 Related Work

LPI and LPV are common in many tasks and applications. LPV perturbations have been applied in machine learning for sequence alignment [11] and pattern recognition [41, 50], such as managing temporal variations in computer vision [66], audio processing for signal analysis [36], enhancing activity recognition through data augmentation [54], and improving accuracy in time series classification by applying temporal modifications [27]. LPI perturbations have been extensively investigated in computer vision, by assessing DNN models against uniform corruptions [22, 32, 51] and developing consistent training algorithms [61]. In audio processing, uniform acoustic characteristics have been utilized for speaker verification [13] and device-consistent classification [23]. Despite being widely used in practice, the robustness of DNNs against LPI and LPV perturbations has not been formally defined or verified, which is the focus of this work.

The work in [37, 43], as mentioned in §5.5, considered a limited subset of our defined LPI properties, e.g., restricting convolution kernels to values in $[0, 1]$ that sum to 1 [37], or pixel-level spatial smoothness constraints [43]. Those approaches compute worst-case bounds given the perturbation boundaries and formulate the problem as a standard LR verification task. While enabling existing verification techniques, the resulting overapproximated spaces makes the work ineffective in practice and unable to solve many problems (as illustrated in §5.5). Additionally, they do not consider LPV properties, which represent an important class of perturbations and is much more challenging to verify as shown in §5.1. VeriS addresses both complete LPI and LPV properties through an approach that incorporates perturbation subnetworks directly into the network architecture.

DNN verification work has primarily focused on LR specifications [5, 15, 19, 29, 65], in which specifications are created by adding small perturbations to each input independently. However, no prior work has focused specifically on verifying SR such as LPI and LPV specifications. To the best of our knowledge, VeriS is the first framework to define and verify SR properties for DNNs.

Constraint-based solvers, like Planet [17] and Marabou [62], which encode the DNN verification problem as an SMT formula, are potentially capable of encoding complex constraints in SR properties, but they do not scale sufficiently to handle realistic DNNs [2, 6]. In contrast, abstraction-based DNN verifiers overapproximate nonlinear computations (e.g., ReLU) of the network using abstract domains, such as interval [58], zonotope [44], polytope [45, 63], starset/imagestar [52], to scale verification. Such techniques and tools include Mn-Bab [19], ReluVal [58], Neurify [57], Nnv [53], Nnenum [1], $\alpha\beta$-Crown [59, 64, 65], etc. This work leverages two state-of-the-art abstraction-based DNN verifiers, $\alpha\beta$-Crown and NeuralSat, to solve SR problems efficiently.

8 Conclusion and Future Work

This work introduced SR properties that extend DNN verification beyond the limitations of traditional LR formulations. By defining LPI and LPV perturbation classes, we captured the structured transformations that occur in many domains but cannot be expressed through interval constraints. The key insight of our approach lies in transforming complex SR verification problems into LR ones, allowing existing verification tools to be solve problems they could not previously handle.

VeriS enables the verification of structural robustness of DNNs against a wide range of perturbation types. VeriS provides a tractable, compatible with state-of-the-art DNN verifiers, and optimized representation of the structured perturbations. It allows for an efficient verification of DNNs for multi-domain tasks under diverse perturbations, with 94% and 49% of verified properties for LPI and LPV, respectively.

Several promising directions emerge from this work. The perturbation subnetwork encoding approach can be extended to capture additional classes of structured transformations beyond convolution-based and time-warping perturbations, including elastic deformations [9], perspective transformations [35], and domain-specific perturbations in robotics and autonomous systems such as those in [20, 42]. Furthermore, the general principle of encoding complex verification properties as neural network components suggests broader applications beyond robustness analysis, potentially enabling verification of other structured properties such as fairness and domain adaptation [4, 46].

9 Data Availability

VeriS is available at: https://anonymous.4open.science/r/VeriS/

References

[1] Stanley Bak. 2021. nnenum: Verification of ReLU Neural Networks with Optimized Abstraction Refinement. In *NASA Formal Methods Symposium*. Springer, 19–36. doi:10.1007/978-3-030-76384-8_2

[2] Stanley Bak, Changliu Liu, and Taylor Johnson. 2021. The Second International verification of Neural Networks Competition (VNN-COMP 2021): Summary and Results. *arXiv preprint arXiv:2109.00498* (2021). doi:10.48550/arXiv.2109.00498

[3] Osbert Bastani, Yani Ioannou, Leonidas Lampropoulos, Dimitrios Vytiniotis, Aditya Nori, and Antonio Criminisi. 2016. Measuring neural net robustness with constraints. *Advances in neural information processing systems* 29 (2016).

[4] Sumon Biswas and Hridesh Rajan. 2023. Fairify: Fairness verification of neural networks. In *2023 ieee/acm 45th international conference on software engineering (icse)*. IEEE, 1546–1558.

[5] Christopher Brix, Stanley Bak, Taylor T Johnson, and Haoze Wu. 2024. The Fifth International Verification of Neural Networks Competition (VNN-COMP 2024): Summary and Results. *arXiv preprint arXiv:2412.19985* (2024).

[6] Christopher Brix, Mark Niklas Müller, Stanley Bak, Taylor T Johnson, and Changliu Liu. 2023. First three years of the international verification of neural networks competition (VNN-COMP). *International Journal on Software Tools for Technology Transfer* (2023), 1–11. doi:10.48550/arXiv.2301.05815

[7] Rudy Bunel, Jingyue Lu, Ilker Turkaslan, Philip HS Torr, Pushmeet Kohli, and M Pawan Kumar. 2020. Branch and bound for piecewise linear neural network verification. *Journal of Machine Learning Research* 21, 42 (2020), 1–39.

[8] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*. Ieee, 39–57.

[9] Isaac Chao, Ulrich Pinkall, Patrick Sanan, and Peter Schröder. 2010. A simple geometric model for elastic deformations. *ACM transactions on graphics (TOG)* 29, 4 (2010), 1–6.

[10] Jinyin Chen, Chengyu Jia, Yunjie Yan, Jie Ge, Haibin Zheng, and Yao Cheng. 2024. A miss is as good as a mile: Metamorphic testing for deep learning operators. *Proceedings of the ACM on Software Engineering* 1, FSE (2024), 2005–2027.

[11] Marco Cuturi and Mathieu Blondel. 2017. Soft-dtw: a differentiable loss function for time-series. In *International conference on machine learning*. PMLR, 894–903.

[12] Wei Dai, Chia Dai, Shuhui Qu, Juncheng Li, and Samarjit Das. 2017. Very deep convolutional neural networks for raw waveforms. In *2017 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 421–425.

[13] Brecht Desplanques, Jenthe Thienpondt, and Kris Demuynck. 2020. Ecapa-tdnn: Emphasized channel attention, propagation and aggregation in tdnn based speaker verification. *arXiv preprint arXiv:2005.07143* (2020).

[14] Swaroopa Dola, Matthew B Dwyer, and Mary Lou Soffa. 2023. Input Distribution Coverage: Measuring Feature Interaction Adequacy in Neural Network Testing. *ACM Transactions on Software Engineering and Methodology* 32, 3 (2023), 1–48. https://dl.acm.org/doi/10.1145/3576040

[15] Hai Duong, ThanhVu Nguyen, and Matthew B Dwyer. 2025. NeuralSAT: A High-Performance Verification Tool for Deep Neural Networks. In *International Conference on Computer Aided Verification*. to appear.

[16] Hai Duong, Dong Xu, Thanhvu Nguyen, and Matthew B. Dwyer. 2024. Harnessing Neuron Stability to Improve DNN Verification. *Proc. ACM Softw. Eng.* 1, FSE, Article 39 (2024), 23 pages. doi:10.1145/3643765

[17] Ruediger Ehlers. 2017. Formal verification of piece-wise linear feed-forward neural networks. In *International Symposium on Automated Technology for Verification and Analysis*. Springer, 269–286. doi:10.1007/978-3-319-68167-2_19

[18] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. 2019. Adversarial attacks on deep neural networks for time series classification. In *2019 International joint conference on neural networks (IJCNN)*. IEEE, 1–8.

[19] Claudio Ferrari, Mark Niklas Mueller, Nikola Jovanović, and Martin Vechev. 2022. Complete Verification via Multi-Neuron Relaxation Guided Branch-and-Bound. In *International Conference on Learning Representations*. doi:10.48550/arXiv.2205.00263

[20] Rod Frehlich. 2001. Errors for space-based Doppler lidar wind measurements: Definition, performance, and verification. *Journal of Atmospheric and Oceanic Technology* 18, 11 (2001), 1749–1772.

[21] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

[22] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. 2022. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 16000–16009.

[23] Hu Hu, Chao-Han Huck Yang, Xianjun Xia, Xue Bai, Xin Tang, Yajian Wang, Shutong Niu, Li Chai, Juanjuan Li, Hongning Zhu, et al. 2021. A two-stage approach to device-robust acoustic scene classification. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 845–849.

[24] Qiang Hu, Yuejun Guo, Xiaofei Xie, Maxime Cordy, Lei Ma, Mike Papadakis, and Yves Le Traon. 2024. Test optimization in dnn testing: a survey. *ACM Transactions on Software Engineering and Methodology* 33, 4 (2024), 1–42.

[25] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. 2017. Safety verification of deep neural networks. In *International conference on computer aided verification*. Springer, 3–29. doi:10.1007/978-3-319-63387-9_1

[26] The PhysioNet/Computing in Cardiology Challenge. 2017. Cardiac Arrhythmia Dataset. https://physionet.org/content/challenge-2017/1.0.0/

[27] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. 2019. Deep learning for time series classification: a review. *Data mining and knowledge discovery* 33, 4 (2019), 917–963.

[28] Fazle Karim, Somshubra Majumdar, and Houshang Darabi. 2020. Adversarial attacks on time series. *IEEE transactions on pattern analysis and machine intelligence* 43, 10 (2020), 3309–3320.

[29] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. 2017. Towards proving the adversarial robustness of deep neural networks. *Proc. 1st Workshop on Formal Verification of Autonomous Vehicles (FVAV), pp. 19-26* (2017).

[30] Jinhan Kim, Robert Feldt, and Shin Yoo. 2019. Guiding deep learning system testing using surprise adequacy. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 1039–1049. https://dl.acm.org/doi/10.1109/ICSE.2019.00108

[31] Klas Leino, Zifan Wang, and Matt Fredrikson. 2021. Globally-robust neural networks. In *International Conference on Machine Learning*. PMLR, 6212–6222.

[32] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. 2022. A convnet for the 2020s. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 11976–11986.

[33] Yu-Seung Ma, Shin Yoo, and Taeho Kim. 2021. Selecting test inputs for DNNs using differential testing with subspecialized model instances. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1467–1470.

[34] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2017. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083* (2017).

[35] Jack Mezirow. 1978. Perspective transformation. *Adult education* 28, 2 (1978), 100–110.

[36] Meinard Müller. 2015. *Fundamentals of music processing: Audio, analysis, algorithms, applications*. Vol. 5. Springer.

[37] Mallek Mziou-Sallami and Faouzi Adjed. 2022. Towards a Certification of Deep Image Classifiers against Convolutional Attacks.. In *ICAART (2)*. 419–428.

[38] Paarth Neekhara, Shehzeen Hussain, Prakhar Pandey, Shlomo Dubnov, Julian McAuley, and Farinaz Koushanfar. 2019. Universal adversarial perturbations for speech recognition systems. *arXiv preprint arXiv:1905.03828* (2019).

[39] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. 2016. The limitations of deep learning in adversarial settings. In *2016 IEEE European symposium on security and privacy (EuroS&P)*. IEEE, 372–387.

[40] Matt Pharr, Wenzel Jakob, and Greg Humphreys. 2023. *Physically based rendering: From theory to implementation*. MIT Press.

[41] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh. 2012. Searching and mining trillions of time series subsequences under dynamic time warping. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. 262–270.

[42] Vijay Rengarajan, Yogesh Balaji, and AN Rajagopalan. 2017. Unrolling the shutter: Cnn to correct motion distortions. In *Proceedings of the IEEE Conference on computer Vision and Pattern Recognition*. 2291–2299.

[43] Anian Ruoss, Maximilian Baader, Mislav Balunović, and Martin Vechev. 2021. Efficient certification of spatial robustness. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 35. 2504–2513.

[44] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. 2018. Fast and effective robustness certification. *Advances in neural information processing systems* 31 (2018). https://dl.acm.org/doi/10.5555/3327546.3327739

[45] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. 2019. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages* 3, POPL (2019), 1–30. doi:10.1145/3290354

[46] Bing Sun, Jun Sun, Ting Dai, and Lijun Zhang. 2021. Probabilistic verification of neural networks against group fairness. In *International Symposium on Formal Methods*. Springer, 83–102.

[47] Youcheng Sun, Xiaowei Huang, Daniel Kroening, James Sharp, Matthew Hill, and Rob Ashmore. 2019. Structural Test Coverage Criteria for Deep Neural Networks. *ACM Transactions on Embedded Computing Systems (TECS)* 18, 5s (2019), 1–23. doi:10.1145/3358233

[48] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199* (2013).

[49] Richard Szeliski. 2022. *Computer vision: algorithms and applications.* Springer Nature.

[50] Romain Tavenard, Johann Faouzi, Gilles Vandewiele, Felix Divo, Guillaume Androz, Chester Holtz, Marie Payne, Roman Yurchak, Marc Rußwurm, Kushal Kolar, et al. 2020. Tslearn, a machine learning toolkit for time series data. *Journal of machine learning research* 21, 118 (2020), 1–6.

[51] Hugo Touvron, Matthieu Cord, and Hervé Jégou. 2022. Deit iii: Revenge of the vit. In *European conference on computer vision.* Springer, 516–533.

[52] Hoang-Dung Tran, Neelanjana Pal, Diego Manzanas Lopez, Patrick Musau, Xiaodong Yang, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, and Taylor T Johnson. 2021. Verification of piecewise deep neural networks: a star set approach with zonotope pre-filter. *Formal Aspects of Computing* 33 (2021), 519–545.

[53] Hoang-Dung Tran, Neelanjana Pal, Patrick Musau, Diego Manzanas Lopez, Nathaniel Hamilton, Xiaodong Yang, Stanley Bak, and Taylor T Johnson. 2021. Robustness Verification of Semantic Segmentation Neural Networks Using Relaxed Reachability. In *International Conference on Computer Aided Verification.* Springer, 263–286. doi:10.1007/978-3-030-81685-8_12

[54] Terry T Um, Franz MJ Pfister, Daniel Pichler, Satoshi Endo, Muriel Lang, Sandra Hirche, Urban Fietzek, and Dana Kulić. 2017. Data augmentation of wearable sensor data for parkinson's disease monitoring using convolutional neural networks. In *Proceedings of the 19th ACM international conference on multimodal interaction.* 216–220.

[55] VNN-COMP 2025. 2025. VNN-COMP 2025 Slides. https://docs.google.com/presentation/d/1ep-hGGotgWQF6SA0JIpQ6nFqs2lXoyuLMM-bORzNvrQ/edit?usp=sharing

[56] Longtian Wang, Xiaofei Xie, Xiaoning Du, Meng Tian, Qing Guo, Zheng Yang, and Chao Shen. 2023. DistXplore: Distribution-guided testing for evaluating and enhancing deep learning systems. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering.* 68–80.

[57] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. 2018. Efficient formal safety analysis of neural networks. *Advances in Neural Information Processing Systems* 31 (2018). https://dl.acm.org/doi/10.5555/3327345.3327533

[58] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. 2018. Formal security analysis of neural networks using symbolic intervals. In *27th USENIX Security Symposium (USENIX Security 18).* 1599–1614. https://dl.acm.org/doi/10.5555/3277203.3277323

[59] Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. 2021. Beta-CROWN: Efficient Bound Propagation with Per-neuron Split Constraints for Complete and Incomplete Neural Network Robustness Verification. *Advances in Neural Information Processing Systems* 34 (2021), 29909–29921. doi:10.48550/arXiv.2103.06624

[60] P. Warden. 2018. Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. *ArXiv e-prints* (April 2018). arXiv:1804.03209 [cs.CL] https://arxiv.org/abs/1804.03209

[61] Ross Wightman, Hugo Touvron, and Hervé Jégou. 2021. Resnet strikes back: An improved training procedure in timm. *arXiv preprint arXiv:2110.00476* (2021).

[62] Haoze Wu, Omri Isac, Aleksandar Zeljić, Teruhiro Tagomori, Matthew Daggitt, Wen Kokke, Idan Refaeli, Guy Amir, Kyle Julian, Shahaf Bassan, et al. 2024. Marabou 2.0: a versatile formal analyzer of neural networks. In *International Conference on Computer Aided Verification.* Springer, 249–264.

[63] Kaidi Xu, Zhouxing Shi, Huan Zhang, Yihan Wang, Kai-Wei Chang, Minlie Huang, Bhavya Kailkhura, Xue Lin, and Cho-Jui Hsieh. 2020. Automatic perturbation analysis for scalable certified robustness and beyond. *Advances in Neural Information Processing Systems* 33 (2020), 1129–1141. https://dl.acm.org/doi/10.5555/3495724.3495820

[64] Huan Zhang, Shiqi Wang, Kaidi Xu, Linyi Li, Bo Li, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. 2022. General cutting planes for bound-propagation-based neural network verification. *Proceedings of the 36th International Conference on Neural Information Processing Systems* (2022). https://dl.acm.org/doi/10.5555/3600270.3600391

[65] Duo Zhou, Christopher Brix, Grani A Hanasusanto, and Huan Zhang. 2024. Scalable Neural Network Verification with Branch-and-bound Inferred Cutting Planes. *arXiv preprint arXiv:2501.00200* (2024).

[66] Feng Zhou and Fernando De la Torre. 2015. Generalized canonical time warping. *IEEE transactions on pattern analysis and machine intelligence* 38, 2 (2015), 279–294.

[67] Feng Zhou and Fernando Torre. 2009. Canonical time warping for alignment of human behavior. *Advances in neural information processing systems* 22 (2009).

[68] Tahereh Zohdinasab, Vincenzo Riccio, Alessio Gambi, and Paolo Tonella. 2021. DeepHyperion: exploring the feature space of deep learning-based systems through illumination search. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis.* 79–90. doi:10.1145/3460319.3464811