Leveraging Over-Parameterization to Improve the Verifiability of Neural Networks

Andrea Gimelli , **Luca Oneto** and **Armando Tacchella**
University of Genoa, Genoa, Italy

Abstract

Over-parameterized neural networks, i.e., models with excess capacity that can fit training data exactly, have demonstrated superior generalization performance compared to classical models with balanced capacity. Nevertheless, their deployment in safety-critical domains remains severely constrained by their susceptibility to, e.g., natural perturbations and adversarial manipulations. Verification techniques can solve such problems, but the computational cost of these methods can be prohibitive when applied to large models. In this work, we demonstrate that over-parameterization can be exploited not merely to enhance generalization, but also to mitigate neuron instability, one of the parameters affecting the efficiency of verification. Our experimental findings suggest that over-parameterization may serve as a crucial mechanism for reconciling the long-standing trade-off between generalization and verifiability of neural networks.

1 Introduction

Neural networks (NNs) have achieved remarkable progress: from solving well-defined challenges such as board games [Silver *et al.*, 2016] and predicting protein structures [Varadi *et al.*, 2024], to enabling broad, open-ended tasks like multimodal conversational agents [OpenAI, 2025]. One of the key factors in this success is over-parameterization, i.e., equipping models with (far) more parameters than training data would dictate [Song *et al.*, 2021; Ding *et al.*, 2022; Belkin *et al.*, 2019; Bunel *et al.*, 2020; Oneto *et al.*, 2023]. In practice, such models often generalize better than traditional approaches constrained to a balanced capacity, even when they can perfectly fit the training set. Despite the outstanding performance of over-parameterized NNs and their promising potential in safety-critical domains [Zhang and Li, 2020; Bacciu *et al.*, 2021; Wicker *et al.*, 2018], their adoption remains hindered by the lack of formal guarantees, which raises significant risks. In fact, several studies have shown that NNs, including over-parameterized ones, are vulnerable to input perturbations, whether arising from accidental noise [Goodfellow *et al.*, 2015] or from carefully designed adversarial attacks [Zhou *et al.*, 2024b; Springer *et al.*, 2021; Su *et al.*, 2019].

Automated verification aims to address these issues by checking whether NNs satisfy desirable properties [Zhang and Li, 2020]. However, research in this area has prompted the inherent difficulty of the task. From a theoretical perspective, verifying properties of NNs composed solely of affine transformations and ReLU neurons is known to be NP-complete [Katz *et al.*, 2017]. From a practical standpoint, this difficulty is evident, e.g., in the outcomes of the yearly competition of verification tools (VNNCOMP) [Brix *et al.*, 2024]. The reasons behind these results are diverse. For instance, Mixed-Integer Linear Programming (MILP)-based and SMT-based verifiers [Ehlers, 2017; Tjeng and Tedrake, 2017; Wang *et al.*, 2018a; Ryou *et al.*, 2019; Singh *et al.*, 2019] are highly sensitive to network size (i.e., number of parameters), since larger models correspond to larger and harder optimization problems. Abstraction-based verifiers [Gehr *et al.*, 2018; Lemesle *et al.*, 2024; Guidotti *et al.*, 2021] mitigate complexity by overapproximating neuron behavior; however, coarse approximations often require refinement. This typically involves some forms of case-splitting which can rapidly escalate the number of subproblems, leading to substantial runtime increase. Similarly, branch-and-bound verifiers [Bunel *et al.*, 2020; Zhou *et al.*, 2025] and constraint-based approaches [Katz *et al.*, 2019] struggle with networks of large size. Given these limitations, recent efforts to accelerate verification focused on reducing NN complexity. Strategies include enforcing sparsity and shrink network size [Xiao *et al.*, 2019], reducing the number of neurons causing case-splitting [Xiao *et al.*, 2019; Li *et al.*, 2023], network pruning [Guidotti *et al.*, 2020], and applying knowledge distillation to produce smaller yet high-performing models [Krizhevsky *et al.*, 2012].

Our work addresses the following research question: *Is it possible to leverage over-parameterization to improve both the accuracy of NNs and their verification time?* In other words, rather than designing small models from the outset, reducing their size a posteriori, or mitigating costs solely by training to avoid expensive branching during verification, we explore how over-parameterization itself can be exploited to simultaneously enhance model accuracy and improve verification rutime. In particular, we propose to modify the learning phase of NNs to simultaneously minimize both some clas-

sical accuracy metric [Bishop and Bishop, 2023] and an auxiliary one that helps to control the number of neurons that turn out to be problematic during verification (e.g., the Rs loss) [Xu *et al.*, 2024; Liu *et al.*, 2025]. By considering a number of different architectures and training configurations, we show that over-parameterization can be exploited as an instrument to achieve high performance in both tasks. Our approach differs significantly from current state-of-the-art methods, where the focus is typically on reducing network size or the number of problematic neurons during or after training, without explicitly investigating the impact of over-parameterization on verification time. Our results demonstrate that, on all the NN architectures and the state-of-the-art tools that we consider, suitable over-parameterization will almost always yield models which are easier to verify and not substantially less accurate than their counterparts with baseline over-parameterization only.

The rest of the paper is organized as follows. In Section 2 we introduce some notations and definitions required to understand our contribution. In Section 3 we present our original methodology to train networks in order to yield improved accuracy and verification runtime. Experimental results involving different NN architectures and verification tools are presented and discussed in Section 4, and contextualized with related work in Section 5. We conclude the paper in Section 6 with some final remarks and directions for future work.

2 Preliminaries

Our notations and basic definitions are taken from [Huang *et al.*, 2020]. An NN is defined as a tuple $N = (\mathbb{S}, \mathbb{T}, \Phi)$, where $\mathbb{S} = \{\mathbb{S}_k \mid k \in \{1, \ldots, K\}\}$ is the set of layers, $\mathbb{T} \subseteq \mathbb{S} \times \mathbb{S}$ denotes the connections between layers, and $\Phi = \{\phi_k \mid k \in \{2, \ldots, K-1\}\}$ is the set of functions associated with each non-input and output layer. In a deep NN, $\mathbb{S}_1$ is the *input* layer, $\mathbb{S}_K$ is the *output* layer, and all other layers are *hidden layers*. Each layer $\mathbb{S}_k$ consists of s_k *neurons*, with the l-th neuron denoted by $n_{k,l}$ and s_k being the *size* of the layer. Each hidden or output node $n_{k,l}$ ($2 \leq k \leq K-1$, $1 \leq l \leq s_k$) has a pre-activation value $u_{k,l}$ and a post-activation value $v_{k,l}$. Input nodes $n_{1,l}$ only have post-activation values $v_{1,l}$, and output nodes $n_{K,l}$ only have pre-activation values $u_{K,l}$. Let u_k and v_k denote the vectors of all pre- and post-activation values in layer k. Each hidden layer is followed by a non-linear activation function to compute $v_{k,l}$ from $u_{k,l}$. In this work, we use the ReLU (Rectified Linear Unit) activation function, defined as $v_{k,l} = \max(0, u_{k,l})$, and we say that the neuron is *active* when $v_{k,l} = u_{k,l}$ and *inactive* otherwise. A neuron is *stable* if it is either always active or always inactive and *unstable* otherwise.

In a fully connected (FC) layer, each neuron is connected to all neurons in the previous layer through a weight matrix W^k of size $\langle s_k, s_{k-1} \rangle$, where $w^k_{l,h}$ represents the weight from neuron h in layer $k-1$ to neuron l in layer k, plus a bias term b^k_{fc}

$$u^k = W^k \cdot v^{k-1} + b^k_{\text{fc}}. \tag{1}$$

In a convolutional layer, connections are local and shared through learnable filters F^k. Each filter is convolved over the input feature map v^{k-1} to produce an output feature map u^k

$$u^k_{i,j,l} = \textstyle\sum_c \sum_m \sum_n F^k_{m,n,c,l} \cdot v^{k-1}_{i+m,j+n,c} + b^k_l, \tag{2}$$

where i, j are spatial indices, l is output channel, c is input channel, m, n are filter indices.

Let θ denote the set of all learnable parameters of a neural network N, i.e., the parameters that are optimized during training. Specifically, for FC and convolutional layers, we have

$$\theta = \{W^k, b^k_{\text{fc}}, F^k, b^k\}, \tag{3}$$

where W^k and b^k_{fc} correspond to the weights and biases of the FC layers, and F^k and b^k correspond to the weights and biases of the convolutional layers. These parameters collectively define all trainable aspects of the network.

For a *classification* problem, the NN assigns a *label* to an input, i.e., the index of the output node with the largest value

$$label = \arg\max_{1 \leq l \leq s_K} u_{K,l}. \tag{4}$$

2.1 Training NNs and the Over-Parameterization Effect

Given a *training set*, i.e., a set of inputs for which the corresponding output labels are known, the process of *training* a NN for a classification task involves finding values for the parameters θ to minimize an objective function that quantifies how well the model performs. Formally, the objective can be expressed as:

$$\mathcal{L}(\theta) = \mathcal{L}_{\text{main}}(\theta) + \textstyle\sum_i \lambda_i \, \mathcal{L}_{\text{aux},i}(\theta), \tag{5}$$

where $\mathcal{L}_{\text{main}}$ represents the *task-specific loss*, while the terms $\mathcal{L}_{\text{aux},i}$ denote *auxiliary objectives*, each scaled by a coefficient λ_i. For classification, the term $\mathcal{L}_{\text{main}}$ is typically the *cross-entropy loss* [Bishop and Bishop, 2023], which acts as a differentiable proxy for the misclassification error over a set of samples, measuring the difference between predicted and true class labels. Auxiliary losses are also commonly defined as differentiable functions, and their aim is to obtain other desirable properties such as adherence to physical principles [Zampini *et al.*, 2025], trustworthiness [Oneto *et al.*, 2024], or improved generalization [Bishop and Bishop, 2023]. Optimization of $\mathcal{L}(\theta)$ usually proceeds iteratively over multiple *epochs*, where one epoch corresponds to a complete pass over the training set. Within each epoch, the dataset is divided into *mini-batches* of manageable size: for each mini-batch, the training algorithm computes predictions, evaluates the loss function, and derives the gradient $\nabla_\theta \mathcal{L}(\theta)$ through a technique known as backpropagation [Bishop and Bishop, 2023]. Modern algorithms, such as *Adam* [Springer *et al.*, 2021], are mostly adaptive gradient-based methods which dynamically adjust learning rates for each parameter.

It is well known [Bishop and Bishop, 2023] that one of the main pitfalls in training a NN is failing to yield a model that generalizes well, for instance by memorizing the training set, but failing to predict correctly samples which are not in the training set - a phenomenon known as *overfitting*. Interestingly, recent studies [Belkin *et al.*, 2019; Oneto *et al.*, 2023; Nakkiran *et al.*, 2021] have shown that *over-parameterization*, i.e., adding many more parameters

than those seemingly required to maintain good generalization capabilities, does not lead to overfitting, but acts as a form of implicit regularization. Contrary to the classical view of statistical learning theory, which posits a trade-off between fitting the training data and achieving good generalization, over-parameterized NNs suggest that the optimization dynamics of gradient-based methods and the structure of high-dimensional parameter spaces enable such NNs to converge toward solutions that generalize surprisingly well despite their apparent ability to memorize the training set.

2.2 Verifying NNs

Given a network N and a property ψ, verification is the process of checking whether N satisfies ψ. Here we focus on local robustness[1] properties: A NN is said to be locally robust around an input x with respect to a perturbation space $\mathcal{P}^\epsilon$ if the network's output does not change within the neighborhood defined by the bound, where ϵ defines the perturbation space. If $x \in \mathbb{R}^n$ we can define the $\mathcal{P}^\epsilon$ as the L_p ball centered in x

$$\mathcal{P}^\epsilon(x) = \{\tilde{x} \in \mathbb{R}^n : \|\tilde{x} - x\|_p \leq \epsilon\}, \tag{6}$$

then a network is locally robust around x if $\forall \tilde{x} \in \mathcal{P}^\epsilon(x), \quad \text{class}(x) = \text{class}(\tilde{x})$. This ensures that, for all inputs within the ϵ-L_p ball centered in x, the network assigns the same *label*. In practice, for verification benchmarks, the L_∞ norm is almost always the metric of choice.

Several approaches have been proposed and implemented in NN verification tools, many of which have been evaluated in VNN-Comp [Brix *et al.*, 2024], an international event where tools are evaluated on a variety of NNs under different perturbation models. The set of constraints defining a property is in the VNN-LIB [Demarchi *et al.*, 2023] language, a standard tailored for NN verification which draws from the SMT-LIB language [Barrett *et al.*, 2010]. We focus on four state of the art tools that participated in recent VNN-Comp events: ABCROWN [Zhang *et al.*, 2018; Xu *et al.*, 2020; Salman *et al.*, 2019; Xu *et al.*, 2021; Wang *et al.*, 2021; Zhang *et al.*, 2022a; Zhang *et al.*, 2022b; Kotha *et al.*, 2023; Zhou *et al.*, 2024a; Zhou *et al.*, 2025], MARABOU [Wu *et al.*, 2024], PYRAT [Lemesle *et al.*, 2024], and PYNEVER [Demarchi *et al.*, 2024b; Demarchi *et al.*, 2024a; Demarchi *et al.*, 2022; Guidotti *et al.*, 2021]. ABCROWN is an efficient verifier based on the linear bound propagation framework, combined with specialized branch-and-bound (B&B) techniques. PYRAT relies on abstract interpretation across multiple abstract domains, e.g., intervals, zonotopes and constrained zonotopes, to over-approximate reachable states and enable analysis of various NN architectures. MARABOU encodes verification tasks as constraint satisfaction problems, which are solved using a combination of simplex-based reasoning and symbolic propagation. Finally, PYNEVER adopts an abstraction-refinement approach based on symbolic bounds propagation and star sets to represent polytopes.

A key distinction in NN verification lies between *incomplete* and *complete* approaches [Huang *et al.*, 2020; Guidotti *et al.*, 2021]. Incomplete verification, e.g., based on abstract intepretation, is efficient but may return inconclusive results, whereas complete verification guarantees a definitive answer, e.g., by refining the abstract analysis. All the tools we consider support complete verification for feedforward NNs with ReLU activation functions. We can observe that the time to perform such verification is largely influenced by the number of unstable neurons [Katz *et al.*, 2017]. Such neurons cannot be treated as linear functions of their inputs and must be handled by more expensive algorithms, e.g., case splitting, which may lead to an exponential growth of verification runtime in the size of the NN.

[1] Verifying local robustness properties may also support checking for more general properties, see [Shriver *et al.*, 2021]

2.3 Training for Verification

Given the impact of unstable neurons on the verification phase, *training for verification* approaches have been proposed to control and reduce their number in the training phase. The main idea is to estimate the lower and upper bounds of neuron pre-activation values, given some interval as input to the NN. The estimate forms the basis of auxiliary objectives that help to maintain the number of unstable neurons small while training. In the literature, we can find two main families of methods based on interval propagation: *Naive Interval Propagation* (NIP) [Xiao *et al.*, 2019] is computationally efficient, but tends to overestimate bounds in NNs with many hidden layers; *Symbolic Interval Propagation* (SIP) [Wang *et al.*, 2018b; Demarchi *et al.*, 2024a], on the other hand, preserves symbolic dependencies to produce tighter bounds at the cost of a higher computational overhead than NIP. Once an estimate of unstable neurons is available, some function of such estimate must be added to the objective function of Eq. (5) as an auxiliary objective. A notable example of such a function is the so-called *Rs loss* [Devlin *et al.*, 2019], a loss function that provides a differentiable measure of neuron stability. *Rs loss* is based on the following definition:

$$F(l_{k,l}, u_{k,l}) = -\tanh(1 + l_{k,l} \cdot u_{k,l}), \tag{7}$$

where $l_{k,l}$ and $u_{k,l}$ are the estimated lower and upper bounds of the pre-activation of neuron l in layer k. Here, $k \in K$ indicates the set of layers intended to be stabilized, and $l \in \{1, \ldots, n_k\}$ corresponds to the indices of the neurons in layer k, with n_k denoting the number of neurons in that layer.

Its output lies in $(-1, 1)$, approaching 1 for very stable neurons and -1 for very unstable ones. When training is performed in b mini-batches with dimension B, the bounds are averaged across the batch to obtain representative measures

$$\bar{l}_{k,l} = \tfrac{1}{B} \textstyle\sum_{b=1}^{B} l_{k,l,b}, \quad \bar{u}_{k,l} = \tfrac{1}{B} \textstyle\sum_{b=1}^{B} u_{k,l,b}. \tag{8}$$

Finally, the overall *Rs loss* is computed by summing contributions from all neurons of the K layers

$$\mathcal{L}_{\text{rs}} = \textstyle\sum_{k \in \mathcal{K}} \sum_{l=1}^{n_k} F(\bar{l}_{k,l}, \bar{u}_{k,l}). \tag{9}$$

The parameter λ controlling the relative weight of $\mathcal{L}_{\text{rs}}$ in Eq. (5) must be carefully tuned to avoid degrading the NN performance [Devlin *et al.*, 2019].

Algorithm 1 Lambda Search

Input: $\mathcal{N}$, dataset, prev_acc, prev_unst, ϵ, prev_lambda, steps_limit
Output : opt_model, metrics, n_unst, next_lambda

```
1:  candidates ← []
2:  min_inc ← MIN_INC
3:  max_inc ← MAX_INC
4:  increment ← (min_inc + max_inc) / 2
5:  opt_model, metrics, next_lambda ← NULL
6:  for step = 1 to steps_limit do
7:    λ ← prev_lambda + increment
8:    candidate, metrics ← TrainModel(N, dataset, λ)
9:    n_unst ← IntervalProp( candidate, ε, dataset.test_set)
10:   if metrics.accuracy > prev_acc then
11:     Append (metrics, n_unst, λ) to candidates
12:     min_inc ← increment
13:     increment ← (min_inc + max_inc) / 2
14:     if n_unst < prev_unst then
15:       next_lambda ← λ
16:       opt_model ← candidate
17:       return opt_model, metrics, n_unst, next_lambda
18:     end if
19:   else
20:     max_inc ← increment
21:     increment ← (min_inc + max_inc) / 2
22:   end if
23: end for
24: if candidates ≠ [] then
25:   opt_model, metrics, n_unst, λ ← SelectMin( candidates, key = n_unst)
26:   next_lambda ← λ
27: end if
28: return opt_model, metrics, n_unst, next_lambda
```

3 Methodology

In the following, we assume that a class of NN architectures is predefined by specifying the number and type of hidden layers - either convolutional or FC - together with input and output layers appropriately sized for the dataset under consideration. In our training protocol, referred to as the *Neuron Stability Protocol* (NSP), we construct an ordered sequence of NN architectures belonging to the aforementioned class, where each architecture in the sequence is characterized by an increasing number of parameters, i.e., neurons in the hidden layers. The total number of parameters in each architecture approximately doubles with respect to the previous one in the sequence. The first architecture is selected so as to be capable of fitting the training set; heuristically, this is achieved by employing a number of parameters comparable to the number of samples in the dataset. The last architecture in the sequence is the largest one that can be trained while maintaining a reasonable use of computational resources[2].

We train each architecture in the sequence using the *Rs loss* as an auxiliary objective. Since the verification process targets local robustness properties, we estimate the pre-activation bounds through an interval propagation technique, considering an ϵ-L_∞ ball centered on each training sample as input to the network. As previously discussed, setting the weight λ of the *Rs loss* term in the objective function of Eq. (5) is crucial. The smallest network in the sequence is trained with $\lambda = 0$, meaning that no neuron stability control is enforced for it. This network serves as a baseline, establishing the minimum acceptable test accuracy and the maximum tolerated number of unstable neurons for the subsequent architectures in the sequence. The goal of the NSP is therefore to identify the largest network and the corresponding nonzero value of λ that together yield effective over-parameterization *while simultaneously* reducing the number of unstable neurons. The underlying intuition is that over-parameterization enables both higher test accuracy and stronger regularization; in other words, larger networks can sustain higher values of λ without a loss in accuracy, thereby exhibiting fewer unstable neurons.

As part of the NSP, we developed an algorithm for the automatic tuning of λ, referred to as the *Lambda Search Algorithm* (LSA), whose pseudo-code is reported in Algorithm 1. Given a NN architecture $\mathcal{N}$, a dataset, an initial minimum acceptable test accuracy prev_acc, and a maximum tolerated number of unstable neurons prev_unst, the algorithm iteratively determines the optimal value of λ starting from a minimum value prev_lambda, within a maximum number of search steps steps_limit, and using propagation intervals of size ϵ. For each architecture in the NSP sequence, the value of prev_lambda corresponds to the λ used for the previous architecture, while prev_unst denotes its associated number of unstable neurons. As customary, the dataset is assumed to be divided into a training set (dataset.train_set) for parameter optimization and a test set (dataset.test_set) for estimating generalization performance. LSA returns the best trained network for the given architecture (optimal_model), along with the corresponding metrics (metrics), the estimated number of unstable neurons (n_unst), and the selected value of λ (next_lambda).

At each iteration of the main loop of the LSA (Lines 5–22), a candidate model is trained using the TrainModel function, which optimizes the following objective function:

$$\mathcal{L}(\theta) = \mathcal{L}_{\text{main}}(\theta) + \lambda\, \mathcal{L}_{\text{rs}}(\theta), \tag{10}$$

where $\mathcal{L}_{\text{rs}}(\theta)$ denotes the *Rs loss*, and $\mathcal{L}_{\text{main}}(\theta)$ is the standard cross-entropy loss. The hyperparameter λ (Line 6) is computed by incrementing prev_lambda by a small adaptive step whose value lies within the constants MIN_INC and MAX_INC, defining the search window for λ. The candidate model is then evaluated via an interval propagation method (Line 8), which estimates the average number of unstable neurons (n_unst) using ϵ-L_∞ balls centered on each test

[2] In principle, there is no upper bound on the degree of *over-parameterization* of a NN. However, in practice, it is unnecessary to further increase the network size once the *over-parameterization effect* becomes evident.

Arch.	H.Dim.	# Par.	Arch.	H.Dim.	# Par.	Arch.	H.Dim.	# Par.
SH-30	30	24k	CX-50	50	40k	CV-5	5	49k / –
SH-100	100	80k	CX-100	100	90k	CV-15	15	147k / 186k
SH-200	200	159k	CX-250	250	262k	CV-25	25	246k / 310k
SH-500	500	398k	CX-500	500	648k	CV-50	50	491k / 620k
SH-1000	1000	795k	CX-1000	1000	1.71M	CV-100	100	981k / 1.24M
SH-2000	2000	1.59M				CV-200	200	1.96M / 2.48M
SH-4000	4000	3.18M				CV-500	500	4.90M / 6.20M
SH-8000	8000	6.36M						
CF-32	32	11.2M						
CF-64	64	11.2M						
CF-256	256	11.2M						
CF-512	512	11.2M						
CF-1024	1024	11.2M						

Table 1: Dimensions of the neural network architectures. The *Arch.* column specifies the architecture class. The *H.Dim.* column reports the hidden dimension h_{dim} for *SH* and *CV* models, while it reports only h_{dim_1} for the *CX* class since $h_{\text{dim}_1} == h_{\text{dim}_2}$. For *CV* architectures, dimensions are reported for two distinct implementations, one using 2×2 convolutional kernels and one using 5×5 convolutional kernels. In the dimension data, *k* stands for "thousands", *M* stands for millions", and all values are rounded to the nearest thousand.

sample. If the candidate model's accuracy improves over the previous iteration (Line 9), the triple containing (i) its performance metrics, (ii) the number of unstable neurons, and (iii) the corresponding λ value is stored, and the increment bounds are updated (Lines 11–12) for larger λ values. If the number of unstable neurons falls below `prev_unst`, the search terminates early, returning the best λ along with the trained model and associated metrics. Otherwise, the search continues with smaller increments of λ (Lines 19–20). After at most `steps_limit` iterations, if one or more candidate models have been collected (Line 23), the model exhibiting the lowest instability is selected as the optimal one by the `SelectMinimum` routine (Line 24). The corresponding λ and test accuracy are then propagated as reference values for the next, larger architecture in the NSP sequence. If a candidate model fails to improve test accuracy compared to previous architectures, it is simply discarded. The NSP introduces additional computational overhead during the training phase. If over-parameterized networks were trained without simultaneously enforcing neuron stability, the time required for interval propagation and for searching an appropriate value of λ could be saved. Nonetheless, this extra cost is justified by the improved neuron stability achieved by the resulting models.

4 Experimental Analysis

To assess whether NSP effectively achieves its intended goal and to collect empirical evidence addressing our main research question, we consider three classes of NN architectures commonly employed across application domains, namely:

- *Shallow (SH)* architectures, comprising a single FC hidden layer of dimension h_{dim};
- *Complex (CX)* architectures, comprising two FC hidden layers with dimensions $h_{\text{dim}1}$ and $h_{\text{dim}2}$;
- *Convolutional (CV)* architectures, consisting of a convolutional layer with either a 5×5 or 2×2 kernel (depending on the dataset), zero padding, stride 1, and 17 filters, followed by an FC hidden layer of dimension h_{dim}. The number and size of the filters were optimized using Optuna [Akiba *et al.*, 2019] to maximize accuracy;
- *Convolutional Fixed (CF)* architectures, which employ a pretrained, over-parameterized ResNet18 backbone followed by a single FC hidden layer of dimension h_{dim}, which is the only component subject to verification.

For each architecture class, a family of networks with progressively increasing numbers of parameters was defined according to NSP. An overview of the corresponding dimensions is reported in Table 1. Each class was trained using both NSP and a standard machine learning approach, referred to as the *baseline training protocol* (BTP). The BTP adopts the same training configuration as NSP, but excludes the *Rs loss* term and the tuning of λ. The training and evaluation procedures are detailed in the following sections.

4.1 Experimental Setup

Three datasets were considered: *MNIST*, a dataset of grayscale handwritten digits, *Fashion-MNIST* (*FMNIST*), a more challenging dataset of grayscale images representing fashion items, and *CIFAR-10*, a dataset of RGB images of 10 different classes[3]. The *SH*, *CX*, and *CV* architectures were trained on *MNIST*. For *FMNIST*, only the *CX* and *CV* architectures were considered, while the *CF* architectures were trained on *CIFAR-10*. Due to the overparameterized design of the networks, all architectures achieved near 100% training accuracy on all datasets.

The *CV* models take as input a single-channel 28×28 image straight from the *MNIST* and *FMNIST* datasets, while the *SH* and *CX* networks receive a corresponding flattened vector of size 784; the *CF* models take as input a 32×32 RGB image. All networks produce an output of 10 neurons each representing one of the ten different classes. Both datasets were divided into $60{,}000$ training samples - `dataset.train_set` in Algorithm (1) - and $10{,}000$ test samples - `dataset.train_set` in Algorithm (1).

The training algorithm[4] was configured with the *Adam* optimizer having a fixed learning rate of 0.001, for $4{,}000$ epochs and a batch size of 128 samples. We also experimented with adaptive learning rate schedulers, such as `StepLR` [Springer *et al.*, 2021], under various configurations. However, maintaining a fixed learning rate ultimately led to more stable convergence and overall better performance. When using NSP, we set ϵ to 0.03 for *MNIST* and *FMNIST* and 0.169 for *CIFAR-10*, consistently with the properties that we verify, and we chose NIP for `IntervalProp` in LSA. In particular, the lower and upper bounds to estimate the stability of hidden layers neurons were computed using our own implementation for *SH*, *CX* and *CF*architectures, and an implementation based on *auto_LiRPA* [Xu *et al.*, 2020] for *CV* networks. We adopted the *NIP* approach over *SIP* to reduce the computational cost of the learning phase and avoid imposing time limits on the overall training procedure.

Verification was carried out considering 100 different local robustness properties for each dataset with the same noise

[3]All data are retrieved from `torchvision.datasets`

[4]Link to repository: https://github.com/NeVerTools/IJCAI-2026

considered during training. For all architectures except *CF*, the noise is applied to the input space, whereas for *CF* architectures it is applied to the feature space.

We checked properties on trained NNs with ABCROWN, PYRAT, MARABOU, and PYNEVER[5]. Whenever possible we considered the same configuration of VNNCOMP 2024 [Brix *et al.*, 2024] ensuring that each solver would performs complete verification. Each solver was allowed a fixed CPU time per instance. For *MNIST*, the limits were 15 seconds for the *SH* and *CX* classes and 60 seconds for the *CV* class. For *FMNIST*, the limits were 180 seconds for the *CX* class and 400 seconds for the *CV* class. For *CIFAR-10*, a time limit of 180 seconds was used for the *CF* architectures. For each neural network architecture, 100 instances were evaluated, and performance was assessed using the median runtime, which is robust to timed-out instances, and the number of cases in which the solver reached the allotted time limit without a conclusive answer. All experiments were conducted on five identical workstations equipped with an Intel Core i7-7400F CPU, 32 GB of RAM, and an NVIDIA RTX 5070 GPU, running `Ubuntu 24.04.2 LTS`. All NNs were implemented and trained using `Python 3.10` and the `PyTorch cu116` library, ensuring consistency across all experimental runs.

4.2 Experimental Results

Our first set of results concerns the learning phase and is reported in Table 2. In the table, for each architecture (column *Architecture*), we show the final accuracy on the training and test sets (column *Acc. (Train/Test)*) and the average estimated number of unstable nodes (column *Unst. Nodes*), both for BTP (first group) and NSP (second group). For NSP, we also report the final value of the parameter λ obtained through LSA. Missing entries in the table correspond to networks for which NSP failed to achieve any accuracy improvement. For both BTP and NSP, the estimate of unstable nodes is computed by NIP at the end of training and is obtained by averaging the number of unstable nodes calculated across all samples in the test set. From Table 2, we observe that the effect of over-parameterization emerges consistently across all architectures and under both training protocols. While the NSP protocol explicitly enforces this trend, under the BTP protocol the increase in accuracy arises naturally as a consequence of the larger network size. However, this trend is not strictly monotonic: for example, in the *CX* architectures trained with BTP, we observe a drop in test accuracy at CX-500, followed by a recovery as the network size further increases. Notably, when using NSP, we successfully reduced the number of unstable neurons compared to BTP across all architectures, except for the smallest ones in each class, where NSP uses $\lambda = 0$. Moreover, with NSP, the number of unstable neurons tends to *decrease* as network size *increases* in the *CX* and *CV* architectures, whereas in the *SH* architectures this pattern is less regular. Finally, the difference in accuracy between networks trained with BTP and NSP remains negligible, reflecting the trade-off introduced by the *Rs loss* term in NSP. As far as training runtime is concerned, the overhead

Architecture	BTP		NSP		
	Acc. (Train/Test)	Unst. Nodes	Acc. (Train/Test)	Unst. Nodes	λ
MNIST					
SH-30	100.0 / 96.42	18.0	100.0 / 96.42	18.0	0.00
SH-100	100.0 / 97.37	30.4	99.9 / 96.32	2.7	0.81
SH-200	100.0 / 97.79	51.1	100.0 / 96.41	2.2	1.63
SH-500	100.0 / 97.95	125.7	99.9 / 96.61	2.0	4.58
SH-1000	100.0 / 97.99	263.3	99.9 / 96.74	2.1	10.19
SH-2000	100.0 / 98.11	566.4	100.0 / 96.99	2.7	15.43
SH-4000	100.0 / 98.19	1226.4	100.0 / 97.59	3.7	21.43
SH-8000	100.0 / 98.24	2662.7	100.0 / 97.77	6.5	27.05
CX-50	99.96 / 97.90	76.1	100.0 / 97.90	76.1	0.00
CX-100	99.94 / 98.02	142.3	100.0 / 97.99	1.9	0.81
CX-250	100.0 / 98.55	310.3	100.0 / 98.08	0.4	3.76
CX-500	100.0 / 98.39	553.6	100.0 / 98.10	0.7	6.71
CX-1000	100.0 / 98.54	969.2	– / –	–	–
CV-5	100.0 / 97.46	1598.2	100.0 / 97.46	1598.2	0.00
CV-15	100.0 / 98.50	2702.3	100.0 / 97.83	84.8	4.45
CV-25	100.0 / 98.34	2255.3	100.0 / 98.03	56.0	8.90
CV-50	100.0 / 98.64	2410.3	100.0 / 98.13	49.2	13.35
CV-100	100.0 / 98.74	3996.9	100.0 / 98.13	44.4	14.54
CV-200	100.0 / 98.78	2982.4	– / –	–	–
CV-500	100.0 / 98.84	2344.2	100.0 / 98.25	46.1	14.65
Fashion-MNIST					
CX-50	99.8 / 86.49	86.2	99.8 / 86.49	86.2	0.00
CX-100	99.94 / 88.05	164.4	98.2 / 87.19	0.6	2.95
CX-250	99.91 / 89.65	338.2	98.9 / 87.52	0.6	7.42
CX-500	99.84 / 89.76	572.5	98.6 / 88.2	1.5	10.37
CX-1000	99.82 / 89.9	1019.7	99.0 / 88.67	13.2	11.59
CV-15	99.91 / 86.68	4824.4	99.91 / 86.68	4824.4	0.00
CV-25	99.99 / 88.37	3792.1	99.86 / 87.39	305.1	4.45
CV-50	99.95 / 88.83	3879.6	99.66 / 87.54	259.5	4.68
CV-100	100.0 / 89.86	3945.5	99.76 / 87.7	157.9	9.14
CV-200	100.0 / 90.28	3598.6	– / –	–	–
CV-500	100.0 / 90.7	3312.8	– / –	–	–
CIFAR-10					
CF-32	100.0 / 93.25	9.6	100.0 / 93.25	9.6	0.00
CF-64	100.0 / 93.36	18.2	100.0 / 93.35	0	0.81
CF-256	100.0 / 93.39	67.5	100.0 / 93.35	0	1.63
CF-512	100.0 / 93.30	119.2	100.0 / 93.39	0	4.58
CF-1024	100.0 / 93.31	171.2	100.0 / 93.42	0	10.19

Table 2: Comparison between BTP and NSP training across architectures. The BTP columns report only accuracy and unstable nodes, since $\lambda = 0$ for all cases. NSP columns include the value of λ.

of NSP over BTP remains confined to a $2\times$ factor for each epoch on *SH*, *CF* and *CV* architectures, whereas it limited to a $3\times$ factor on *CX* architectures. In Table 3, we present the results of running the solvers on the trained NNs. In the table, for each architecture (column *Benchmark*), we report the performance of the solvers (groups of columns labeled by solver name) in terms of median runtime and number of time-outs, expressed as *<time>/<time-outs>*. Results are shown for networks trained under both BTP and NSP (columns *BTP* and *NSP*, respectively, for each solver). The first observation from Table 3 is that, with the exception of the entries underlined, the number of properties causing solver time-outs consistently decreases when using NSP instead of BTP. We interpret this as clear evidence that the most challenging properties for each solver become verifiable within the allotted time limits once the number of unstable neurons is reduced. Importantly, this improvement *does not* come at the expense of accuracy. On the contrary, due to over-parameterization, NSP leads to *increased* accuracy and - somewhat counterintuitively for verification - networks of *larger* size. A second observation concerns the median runtimes, for which the picture is less conclusive. While it is generally true that veri-

[5]The version of the verifiers is the latest one as of Nov. 2025

Benchmark	ABCROWN		PYRAT		MARABOU		PYNEVER	
	BTP	NSP	BTP	NSP	BTP	NSP	BTP	NSP
MNIST								
SH-30	0.20/0	0.02/0	15.00/62	15.00/62	0.33/30	0.33/30	2.28/0	2.28/0
SH-100	0.02/0	0.02/0	2.27/13	2.26/14	0.08/11	0.08/3	2.20/1	1.90/0
SH-200	0.02/0	0.02/0	2.28/9	2.30/16	0.15/7	0.15/11	2.22/3	1.93/0
SH-500	0.03/0	0.03/0	2.38/11	2.37/9	0.40/6	0.39/9	0.28/12	1.92/0
SH-1000	0.03/2	0.03/0	2.55/7	2.54/5	0.86/6	0.84/5	0.29/11	1.93/0
SH-2000	0.03/2	0.04/0	2.91/7	2.88/5	2.08/4	2.03/4	0.33/13	2.04/0
SH-4000	0.05/5	0.05/0	3.67/8	3.58/3	5.55/5	5.55/3	0.42/14	2.20/0
SH-8000	0.08/2	0.08/0	6.04/7	5.64/0	15.00/100	15.00/85	0.46/14	2.58/0
CX-50	0.06/2	0.03/2	15.00/56	15.00/56	0.06/44	0.06/44	12.75/44	12.75/44
CX-100	0.03/9	0.03/0	15.00/63	2.26/11	0.10/45	0.10/6	15.00/77	2.22/0
CX-250	0.03/12	0.03/0	15.00/57	2.32/9	0.29/36	0.28/3	15.00/91	2.24/3
CX-500	0.04/14	0.03/0	15.00/61	2.91/30	0.77/35	0.74/18	15.00/92	2.20/11
CV-5	0.06/24	0.10/24	60.00/77	60.00/77	60.00/52	60.00/52	-/-	-/-
CV-15	0.03/1	0.03/0	3.07/19	2.58/1	15.96/8	27.67/1	-/-	-/-
CV-25	0.03/0	0.03/0	3.08/24	2.62/0	16.16/10	0.66/0	-/-	-/-
CV-50	0.03/0	0.03/0	3.38/22	2.71/1	17.36/7	55.70/10	-/-	-/-
CV-100	0.04/1	0.03/0	60.00/63	2.89/0	19.85/25	1.23/0	-/-	-/-
CV-500	0.09/3	0.09/0	60.00/54	4.57/0	23.61/13	23.07/0	-/-	-/-
Fashion-MNIST								
CX-50	41.616/18	41.616/18	180/100	180/100	180/71	180/71	41.16/18	41.16/18
CX-100	87.18/44	30.32/14	180/100	180/97	180/88	180/72	87.18/44	30.32/14
CX-250	180/73	30.25/23	180/100	180/87	180/99	180/80	190/73	30.25/23
CX-500	180/89	28.89/20	180/100	180/90	180/98	180/74	180/89	28.89/20
CX-1000	180/100	24.26/10	180/100	180/80	180/99	180/73	180/100	24.26/10
CV-15	1.58/14	1.58/14	180/100	180/100	400/100	400/100	-/-	-/-
CV-25	4.52/0	28.27/0	180/100	180/100	400/100	400/99	-/-	-/-
CV-50	326/0	324/0	180/100	180/100	400/100	400/100	-/-	-/-
CV-100	327/0	0.41/1	180/100	180/63	400/100	32.04/46	-/-	-/-
CIFAR-10								
CF-32	0.65/0	0.65/0	180/99	180/100	0.2/47	0.2/47	20.67/0	20.67/0
CF-64	0.63/0	0.56/0	180/99	180/73	0.48/46	0.49/30	30/3	4.79/0
CF-256	0.99/43	0.65/0	180/100	180/65	180/51	180/61	180/63	4.80/0
CF-512	0.43/46	0.58/0	180/100	180/62	180/100	180/59	180/95	4.82/0
CF-1024	0.38/47	0.56/0	180/100	180/62	180/100	180/64	180/98	4.85/0

Table 3: Comparison among solvers for all architectures trained under both BTP and NSP. A value of -/- indicates that the architecture is not supported by the solver.

fying NNs trained with NSP is less computationally expensive than verifying their BTP-trained counterparts, the rate at which runtimes grow within the same class of architectures does not always favor NSP. In other words, although NSP typically reduces overall verification cost, its impact on runtime scalability is not uniformly positive across all architectures. Finally, we note that the trends discussed above appear to be largely independent of the specific internal algorithms used by each solver. Although some solvers perform better *out-of-the-box*, reducing the number of unstable neurons through NSP - particularly for the most challenging properties - tends to yield benefits across nearly all solvers.

4.3 Discussion

Our results suggest that combining structured regularization aimed at reducing the number of unstable neurons with *over-parameterization* can improve both accuracy and verifiability, particularly for properties that are otherwise challenging for verifiers. Furthermore, our experiments provide an interesting perspective on whether NN verifiers are more sensitive to the number of unstable neurons or to network dimensionality. For the architectures we considered, ABCROWN, PYRAT, and PYNEVER exhibit only a modest increase in median runtime as the number of parameters grows, regardless of whether BTP or NSP is used. Conversely, MARABOU shows the greatest sensitivity to dimensionality and the least sensitivity to improvements in neuron stability. Overall, these findings suggest that coupling advanced learning techniques designed with verification in mind with solvers capable of exploiting the resulting network properties could represent a promising pathway toward scalable NN verification.

Despite these encouraging results, several limitations remain. A primary challenge is the computational overhead introduced by interval propagation methods used to estimate pre-activation bounds during training. While these techniques produce tight bounds for shallow networks, their precision and scalability deteriorate as network depth increases. In particular, although SIP yields tighter bounds than NIP, it incurs in significantly higher time and memory costs.

From a machine learning perspective, stacking many FC layers with ReLU activations provides no accuracy benefit, making SIP unnecessary in such settings. In contrast, deep convolutional architectures genuinely benefit from increased depth and are essential for achieving competitive accuracy on datasets such as *CIFAR-10*. However, in the literature regarding "learning-for-verification" approaches typically rely on small networks, achieving limited test accuracy (around 60%) [Xu *et al.*, 2024] and operating far from the over-parameterized regime [Liu *et al.*, 2025]. Architectures required for high accuracy on *CIFAR-10*, remain impractical to train with current SIP-based regularization methods due to their computational cost. For these reasons, we restricted our experiments to relatively small architectures and we verified only the last layer in *CF* architectures.

Additionally, we relied on off-the-shelf solver implementations and did not optimize their internal algorithms to exploit the structural modifications induced by our training procedure. Future work should focus on improving interval analysis methods and adapting solvers to better leverage training-time regularization.

5 Related Works

A broad spectrum of methodologies has been proposed in the literature to enhance the amenability of NNs to formal verification. A significant body of work has focused on improving the efficiency of verification algorithms based on MILP [Ehlers, 2017; Tjeng and Tedrake, 2017; Wang *et al.*, 2018a; Ryou *et al.*, 2019; Singh *et al.*, 2019], which generally benefit from models characterized by a smaller number of variables. This property has been exploited by promoting sparsity within NN architectures [Xiao *et al.*, 2019], thereby reducing the number of variables and accelerating the verification process. Moreover, alternative activation functions have been proposed as substitutes for ReLU, motivated by the observation that its linearization often induces excessive over-approximation. Since linearization constitutes a core strategy in many NN verification frameworks, a piecewise activation function exhibiting tighter linearizability was introduced in [Leofante *et al.*, 2023], effectively mitigating this issue.

A complementary line of research explicitly targets the enforcement of *ReLU stability* [Xiao *et al.*, 2019; Baninajjar *et al.*, 2024; Xu *et al.*, 2024; Li *et al.*, 2023], thereby reducing the number of unstable neurons that exacerbate branching during verification. Such methods either prune marginally unstable neurons with negligible influence on predictive accuracy or incorporate dedicated regularization terms during training to promote neuron stability.

Another prominent research direction addresses the verification of large-scale architectures through model compression techniques, including pruning [Xiao *et al.*, 2019; Guidotti *et al.*, 2020; Min and Motani, 2020] and knowledge distillation [Shriver *et al.*, 2019]. These approaches effectively concentrate the representational capacity of a large network into a smaller, verification-friendly model.

In contrast to these prevailing trends, our work explores an alternative paradigm. Specifically, we demonstrate that, for suitably dimensioned and trained architectures, it is possible to leverage *over-parameterization* [Belkin *et al.*, 2019; Oneto *et al.*, 2023; Nakkiran *et al.*, 2021] to improve both learning performance and verifiability. When combined with a neuron-stability regularizer, over-parameterization enables the training of larger models that remain as verifiable as their smaller counterparts while achieving superior accuracy. This finding suggests that over-parameterization, traditionally regarded as a source of inefficiency, can instead be harnessed to concurrently enhance performance and facilitate verification, provided that appropriate regularization is applied.

6 Conclusions

Our study demonstrates that over-parameterization, traditionally regarded to improve generalization, can also play a decisive role in enhancing the verifiability of NNs. In particular, we introduced the Neuron Stability Protocol and its associated algorithm, which leverage increased network capacity to simultaneously improve accuracy and reduce neuron instability, thereby redefining the training objective to favor more verifiable models. These findings suggest that over-parameterization should not be seen merely as a byproduct of modern deep learning practice or as an obstacle to verification, but rather as a deliberate design principle for achieving a balance between expressivity and formal verifiability. Future work will extend this analysis to more complex architectures and verification scenarios, with the goal of further elucidating the structural relationships among over-parameterization, accuracy, neuron stability, and verifiability in NNs.

Acknowledgements

This work has been partially supported by (i) the project *ELSA – European Lighthouse on Secure and Safe AI*, funded by the European Union's Horizon Europe programme under Grant Agreement No. 101070617; (ii) the project *SERICS* (PE00000014) under the NRRP MUR programme funded by the EU – NGEU; (iii) the project *FAIR* (PE00000013) under the NRRP MUR programme funded by the EU – NGEU; and (iv) the project *FISA-2023-00128*, funded by the MUR programme "Fondo italiano per le scienze applicate".

References

[Akiba *et al.*, 2019] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, and Others. Optuna: A next-generation hyperparameter optimization framework. In *ACM SIGKDD*, 2019.

[Bacciu *et al.*, 2021] Davide Bacciu, Antonio Carta, Daniele Di Sarli, and Others. Towards functional safety compliance of recurrent neural networks. In *International Conference on AI for People: Towards Sustainable AI*, 2021.

[Baninajjar *et al.*, 2024] Anahita Baninajjar, Ahmed Rezine, and Amir Aminifar. Vnn: verification-friendly neural networks with hard robustness guarantees. In *ICML*, 2024.

[Barrett *et al.*, 2010] Clark W. Barrett, Aaron Stump, and C. Tinelli. The smt-lib standard: Version 2.0. In *International workshop on satisfiability modulo theories*, 2010.

[Belkin *et al.*, 2019] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019.

[Bishop and Bishop, 2023] Christopher Bishop and Hugh Bishop. *Deep learning: Foundations and concepts*. Springer Nature, 2023.

[Brix *et al.*, 2024] Christopher Brix, Stanley Bak, Taylor T. Johnson, and Haoze Wu. The fifth international verification of neural networks competition (vnn-comp 2024): Summary and results. *arXiv preprint arXiv:2412.19985*, 2024.

[Bunel *et al.*, 2020] Rudy Bunel, Jingyue Lu, Ilker Turkaslan, and Others. Branch and bound for piecewise linear neural network verification. *Journal of Machine Learning Research*, 21(42):1–39, 2020.

[Demarchi *et al.*, 2022] Stefano Demarchi, Dario Guidotti, Andrea Pitto, and Armando Tacchella. Formal verification of neural networks: A case study about adaptive cruise control. In *ECMS*, 2022.

[Demarchi *et al.*, 2023] Stefano Demarchi, Dario Guidotti, Luca Pulina, and Armando Tacchella. Supporting standardization of neural networks verification with vnnlib and coconet. In *Workshop on Formal Methods for ML-Enabled Autonomous Systems*, 2023.

[Demarchi *et al.*, 2024a] Stefano Demarchi, Andrea Gimelli, and Armando Tacchella. Improving abstract propagation for verification of neural networks. In *ECMS*, 2024.

[Demarchi *et al.*, 2024b] Stefano Demarchi, Dario Guidotti, Luca Pulina, and Armando Tacchella. NeVer2: learning and verification of neural networks. *Soft Computing*, 28(19):11647–11665, 2024.

[Devlin *et al.*, 2019] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2019.

[Ding *et al.*, 2022] Zhiyan Ding, Shi Chen, Qin Li, and Stephen J. Wright. Overparameterization of deep resnet: Zero loss and mean-field analysis. *Journal of Machine Learning Research*, 23(48):1–65, 2022.

[Ehlers, 2017] Rüdiger Ehlers. Formal verification of piecewise linear feed-forward neural networks. In *ATVA*, 2017.

[Gehr *et al.*, 2018] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *IEEE SSP*, 2018.

[Goodfellow *et al.*, 2015] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *ICLR*, 2015.

[Guidotti *et al.*, 2020] Dario Guidotti, Francesco Leofante, Luca Pulina, and Armando Tacchella. Verification of neural networks: Enhancing scalability through pruning. In *ECAI*, 2020.

[Guidotti *et al.*, 2021] Dario Guidotti, Luca Pulina, and Armando Tacchella. pynever: A framework for learning and verification of neural networks. In *ATVA*, 2021.

[Huang *et al.*, 2020] Xiaowei Huang, Daniel Kroening, and Others. A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. *Computer Science Review*, 37:100270, 2020.

[Katz *et al.*, 2017] Guy Katz, Clark W. Barrett, David L. Dill, and Others. Reluplex: An efficient SMT solver for verifying deep neural networks. In *CAV*, 2017.

[Katz *et al.*, 2019] Guy Katz, Derek A. Huang, Duligur Ibeling, and Others. The marabou framework for verification and analysis of deep neural networks. In *CAV*, 2019.

[Kotha *et al.*, 2023] Suhas Kotha, Christopher Brix, Zico Kolter, and Others. Provably bounding neural network preimages. In *NeurIPS*, 2023.

[Krizhevsky *et al.*, 2012] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, 2012.

[Lemesle *et al.*, 2024] Augustin Lemesle, Julien Lehmann, and Tristan Le Gall. Neural network verification with pyrat. *arXiv preprint arXiv:2410.23903*, 2024.

[Leofante *et al.*, 2023] Francesco Leofante, Patrick Henriksen, and Alessio Lomuscio. Verification-friendly networks: the case for parametric relus. In *IJCNN*, 2023.

[Li *et al.*, 2023] Zhangheng Li, Tianlong Chen, Linyi Li, and Others. Can pruning improve certified robustness of neural networks? *arXiv preprint arXiv:2206.07311*, 2023.

[Liu *et al.*, 2025] Zongxin Liu, Zhe Zhao, Fu Song, and Others. Training verification-friendly neural networks via neuron behavior consistency. In *AAAI*, 2025.

[Min and Motani, 2020] John Tan Chong Min and Mehul Motani. Dropnet: Reducing neural network complexity via iterative pruning. In *ICML*, 2020.

[Nakkiran *et al.*, 2021] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, and Others. Deep double descent: where bigger models and more data hurt. *Journal of Statistical Mechanics: Theory and Experiment*, 2021(12):124003–124003, 2021.

[Oneto *et al.*, 2023] Luca Oneto, Sandro Ridella, and Davide Anguita. Do we really need a new theory to understand over-parameterization? *Neurocomputing*, 543:126227, 2023.

[Oneto *et al.*, 2024] Luca Oneto, Sandro Ridella, and Davide Anguita. Towards algorithms and models that we can trust: A theoretical perspective. *Neurocomputing*, 592:127798, 2024.

[OpenAI, 2025] OpenAI. Gpt-5 system card. *https://cdn.openai.com/gpt-5-system-card.pdf*, 2025.

[Ryou *et al.*, 2019] Wonryong Ryou, Jiayu Chen, Mislav Balunovic, and Others. Fast and effective robustness certification. In *NeurIPS*, 2019.

[Salman *et al.*, 2019] Hadi Salman, Greg Yang, Huan Zhang, and Others. A convex relaxation barrier to tight robustness verification of neural networks. In *NeurIPS*, 2019.

[Shriver *et al.*, 2019] David Shriver, Dong Xu, Sebastian G. Elbaum, and Matthew B. Dwyer. Refactoring neural networks for verification. *arXiv preprint arXiv:1908.08026*, 2019.

[Shriver *et al.*, 2021] David Shriver, Sebastian G. Elbaum, and Matthew B. Dwyer. Reducing DNN properties to enable falsification with adversarial attacks. In *ICSE*, 2021.

[Silver *et al.*, 2016] David Silver, Aja Huang, Chris Maddison, and Others. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[Singh *et al.*, 2019] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin T. Vechev. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, 3:1–30, 2019.

[Song *et al.*, 2021] Chaehwan Song, Ali Ramezani-Kebrya, Thomas Pethick, and Others. Subquadratic overparameterization for shallow neural networks. In *NeurIPS*, 2021.

[Springer *et al.*, 2021] Jacob M. Springer, Melanie Mitchell, and Garrett T. Kenyon. Adversarial perturbations are not so weird: Entanglement of robust and non-robust features in neural network classifiers. *arXiv preprint arXiv:2102.05110*, 2021.

[Su *et al.*, 2019] Jiawei Su, Danilo V. Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 23(5):828–841, 2019.

[Tjeng and Tedrake, 2017] Vincent Tjeng and Russ Tedrake. Verifying neural networks with mixed integer programming. In *ICLR Workshop*, 2017.

[Varadi *et al.*, 2024] Mihaly Varadi, Damian Bertoni, Paulyna Magana, and Others. Alphafold protein structure database in 2024: providing structure coverage for over 214 million protein sequences. *Nucleic acids research*, 52(D1):D368–D375, 2024.

[Wang *et al.*, 2018a] Shiqi Wang, Kexin Pei, Justin Whitehouse, and Others. Efficient formal safety analysis of neural networks. In *USENIX Security Symposium*, 2018.

[Wang *et al.*, 2018b] Shiqi Wang, Kexin Pei, Justin Whitehouse, and Others. Formal security analysis of neural networks using symbolic intervals. In *USENIX Security Symposium*, 2018.

[Wang *et al.*, 2021] Shiqi Wang, Huan Zhang, Kaidi Xu, and Others. Beta-CROWN: Efficient bound propagation with per-neuron split constraints for complete and incomplete neural network verification. In *NeurIPS*, 2021.

[Wicker *et al.*, 2018] Matthew Wicker, Xiaowei Huang, and Marta Kwiatkowska. Feature-guided black-box safety testing of deep neural networks. In *TACAS*, 2018.

[Wu *et al.*, 2024] Haoze Wu, Omri Isac, Aleksandar Zeljic, and Others. Marabou 2.0: A versatile formal analyzer of neural networks. In *CAV*, 2024.

[Xiao *et al.*, 2019] Kai Yuanqing Xiao, Vincent Tjeng, Muhammad Shafiullah, and Aleksander Madry. Training for faster adversarial robustness verification via inducing relu stability. In *ICLR*, 2019.

[Xu *et al.*, 2020] Kaidi Xu, Zhouxing Shi, Huan Zhang, and Others. Automatic perturbation analysis for scalable certified robustness and beyond. In *NeurIPS*, 2020.

[Xu *et al.*, 2021] Kaidi Xu, Huan Zhang, Shiqi Wang, and Others. Fast and Complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. In *ICLR*, 2021.

[Xu *et al.*, 2024] Dong Xu, Nusrat Jahan Mozumder, Hai Duong, and Matthew B. Dwyer. Training for verification: Increasing neuron stability to scale DNN verification. In *TACAS*, 2024.

[Zampini *et al.*, 2025] Stefano Zampini, Guido Parodi, Luca Oneto, Andrea Coraddu, and Davide Anguita. A review on full-, zero-, and partial-knowledge based predictive models for industrial applications. *Information Fusion*, page 102996, 2025.

[Zhang and Li, 2020] Jin Zhang and Jingyue Li. Testing and verification of neural-network-based safety-critical control software: A systematic literature review. *Information and Software Technology*, 123:106296, 2020.

[Zhang *et al.*, 2018] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, and Others. Efficient neural network robustness certification with general activation functions. In *NeurIPS*, 2018.

[Zhang *et al.*, 2022a] Huan Zhang, Shiqi Wang, Kaidi Xu, and Others. A branch and bound framework for stronger adversarial attacks of relu networks. In *ICML*, 2022.

[Zhang *et al.*, 2022b] Huan Zhang, Shiqi Wang, Kaidi Xu, and Others. General cutting planes for bound-propagation-based neural network verification. In *NeurIPS*, 2022.

[Zhou *et al.*, 2024a] Duo Zhou, Christopher Brix, Grani A. Hanasusanto, and Huan Zhang. Scalable neural network verification with branch-and-bound inferred cutting planes. In *NeurIPS*, 2024.

[Zhou *et al.*, 2024b] Shuai Zhou, Chi Liu, Dayong Ye, and Others. How deep learning sees the world: A survey on adversarial attacks & defenses. *IEEE Access*, 12:61113–61136, 2024.

[Zhou *et al.*, 2025] Duo Zhou, Christopher Brix, Grani A. Hanasusanto, and Others. Neural network verification with branch-and-bound for general nonlinearities. In *TACAS*, 2025.