Critically Assessing the State of the Art in CPU-based Local Robustness Verification

Matthias König[1,*], Annelot W. Bosman[1], Holger H. Hoos[1,2,3] and Jan N. van Rijn[1]

[1]Leiden Institute of Advanced Computer Science, Leiden University, The Netherlands
[2]Chair for AI Methodology, RWTH Aachen University, Germany
[3]University of British Columbia, Canada

Abstract

Recent research has proposed various methods to formally verify neural networks against minimal input perturbations. This type of verification is referred to as local robustness verification. The research area of local robustness verification is highly diverse, as verifiers rely on a multitude of techniques, including mixed integer programming and satisfiability modulo theories. At the same time, problem instances differ based on the network to be verified, the verification property and the specific network input. This gives rise to the question of which verification algorithm is most suitable to solve a given verification problem. To answer this question, we perform a systematic performance analysis of several CPU-based local robustness verification systems on a newly and carefully assembled set of 79 neural networks, each verified against 100 robustness properties. Notably, we show that there is no single best algorithm that dominates in performance across all verification problem instances. Instead, our results reveal complementarities in verifier performance and illustrate the potential of leveraging algorithm portfolios for more efficient local robustness verification. Furthermore, we confirm the notion that most algorithms only support ReLU-based networks, while other activation functions remain under-supported.

Keywords

Benchmark Analysis, Neural Network Verification, Adversarial Robustness, Shapley Value, Algorithm Portfolios

1. Introduction

In recent years, deep learning methods based on neural networks have been increasingly applied within various safety-critical domains and use contexts, ranging from manoeuvre advisory systems in unmanned aircraft to face recognition systems in mobile phones (see, *e.g.*, Julian et al. [1]). Furthermore, it is now well known that neural networks are vulnerable to *adversarial examples* [2], where a given input is manipulated in such a way that it is misclassified by the network. In the case of image recognition tasks, the required perturbation, whether it is adversarially crafted or arises accidentally, can be so small that it remains virtually undetectable to the human eye. Against this background, much work has focused on developing methods to provide formal guarantees regarding the behaviour of a given neural network [3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]. For instance, a network employed in autonomous driving for detecting traffic signs should always produce accurate predictions, even when the input is slightly perturbed; failing to do so could have fatal consequences. This specific type of assessment refers to *local robustness verification*, a broadly studied verification task, in which a network is systematically tested against various input perturbations under predefined norms, such as the l_∞ norm [15, 16].

Neural network verification is a highly diverse research area, and existing methods rely on a broad range of techniques. At the same time, neural networks differ in terms of their architecture, such as the number of hidden layers and nodes, the type of non-linearities, *e.g.*, ReLU, Sigmoid or Tanh, or the type of operations they employ, *e.g.*, pooling or convolutional layers. This diversity, both in terms of verification approaches and neural network design, makes it challenging for researchers or practitioners to assess and decide which method is most suitable for verifying a given neural network [17]. This challenge is amplified by the fact that the neural network verification community does not (yet) use commonly agreed evaluation protocols, which makes it difficult to draw clear conclusions from the literature regarding the capabilities and performance of existing verifiers. More precisely, existing studies use different benchmarks and, so far, have not provided an in-depth performance comparison of a broad range of verification algorithms.

Recently, a competition series has been initiated, in which several verifiers were applied to different benchmarks (*i.e.*, networks, properties and datasets) and compared in terms of various performance measures, including the number of verified instances as well as running time [18]. While the results from these competitions have provided highly valuable insights into the general

SafeAI 2023: Workshop on Artificial Intelligence Safety, Feb 13-14, 2023 Washington, D.C., US @AAAI-23
*Corresponding author.
h.m.t.konig@liacs.leidenuniv.nl (M. König);
a.w.bosman@liacs.leidenuniv.nl (A. W. Bosman);
hh@aim.rwth-aachen.de (H. H. Hoos);
j.n.van.rijn@liacs.leidenuniv.nl (J. N. van Rijn)

progress in neural network verification, several questions are left unanswered. For instance, it remains unclear why a verification system participated in certain competition categories but not in others, although this information would help in understanding the technical capabilities and limitations of a given verifier. Furthermore, aggregation of the results makes it difficult to assess in detail the strengths or weaknesses of verifiers on different instances. Instead, looking at aggregated results, one can easily get the impression that a single approach dominates 'across the board' – an assumption that is known to be inaccurate for other problems involving formal verification tasks; see, *e.g.*, Xu et al. [19] or Kadioglu et al. [20] for SAT.

In this work, we focus exclusively on local robustness verification in image classification against perturbations under the l_∞ norm. This scenario represents a widely studied verification task, with a large number of networks being publicly available and many verifiers providing off-the-shelf support. Notice that most verification tasks can be translated into local robustness verification queries [21]; we, therefore, believe that our findings are broadly applicable. Moreover, we seek to overcome the limitations of existing benchmarking approaches and shed light on previously unanswered questions with regard to the state of the art in local robustness verification.

Our contributions are as follows:

(i) We analyse the current state of practice in benchmarking verification algorithms;

(ii) we perform a systematic benchmarking study of several, carefully chosen verification methods based on a *newly assembled*, large and diverse set of networks, including 38 CIFAR and 41 MNIST networks with different activation functions, each verified against 100 robustness properties, for which we consumed a total of 1.69 CPU years in running time;

(iii) we present a categorisation of verification benchmarks based on verifier compatibilities with different layer types and operations;

(iv) we quantify verifier performance in terms of the number of solved instances, running time, as well as marginal contribution and Shapley value, showing that top-performing verification algorithms strongly complement rather than consistently dominate each other in terms of performance – *e.g.*, while the verifiers Neurify and Verinet achieved competitive performance on the CIFAR networks we considered, the former solved many instances unsolved by the latter and vice versa;

(v) lastly, we provide a public repository containing all experimental data, along with the newly assembled network collection.[1]

[1] https://github.com/ADA-research/nn-verification-assessment

2. Background

Neural network verification methods seek to formally assess whether a trained neural network adheres to some predefined input-output property. In this study, we focus on local robustness properties. Given a trained neural network and a set of images as inputs, a robustness verification algorithm aims to verify whether or not there exist slight perturbations to an image that lead the network to predict an incorrect label. The maximum perturbation of each variable in the input, *i.e.*, pixel in a given image, is predefined.

Formal verification algorithms can be either *complete* or *incomplete* [22]. An algorithm that is incomplete does not guarantee to report a solution for every given example; however, incomplete verification algorithms are typically *sound*, which means they will report that a property holds only if the property actually holds. On the other hand, an algorithm that is sound and complete will correctly state that a property holds *whenever* it holds when given sufficient resources to be run to completion. In this work, we focus on complete algorithms, as those arguably represent the most ambitious form of neural network verification, making them preferable over incomplete methods, especially in safety-critical applications.

Early work on complete verification of neural networks utilised *satisfiability modulo theories* (SMT) solvers [10, 23, 24, 25, 26], which determine whether a set of logic constraints is satisfiable [27]. The resulting verification problems are NP-complete and challenging to solve in practice. Alternatively, it is possible to formulate the verification task as a constraint optimisation problem using *mixed integer programming* (MIP) [4, 28, 29, 12]. MIP solvers essentially optimise an objective function subject to a set of constraints. MIP problems, similar to SMT problems, can be challenging to solve and tend to be computationally expensive (in terms of CPU time and memory).

To overcome the computational complexity of SMT and MIP, it has been proposed to use the well-known *branch-and-bound* algorithm [30] for solving the verification problem, regardless of whether it is modelled as MIP or SMT [31, 5, 32, 7, 13].

Besides these three popular approaches, recent work has studied *symbolic interval propagation* [4, 9, 33, 34] and *polyhedra* [35, 36], *zonotope* [8, 37] and *star-set* abstraction [38].

3. Common Practices in Benchmarking Neural Network Verifiers

Considering the diversity in neural network verification problems, it is quite natural to assume that a single best

algorithm does not exist, *i.e.*, a method that outperforms all others under *all* circumstances. It is still hard to identify to what extent a method contributes to the state of the art, mainly because verification methods are typically evaluated *(i)* on a small number of benchmarks, which have often been created for the sole purpose of evaluating the method at hand, and *(ii)* against baseline methods for which it is often unclear how they were chosen, leading to several methods claiming state-of-the-art performance without having been directly compared. We note that in the context of local robustness verification, a benchmark most often represents a neural network classifier trained on the MNIST or CIFAR-10 dataset, respectively.

As previously mentioned, a competition series has been established, with the goal of providing an objective and fair comparison of the state-of-the-art methods in neural network verification, in terms of scalability and speed [18]. The VNN Competition was held in the years 2020, 2021 and 2022, yet with different protocols (*e.g.*, for running experiments, scoring, etc.), benchmarks and participants. In this discussion, we focus on the 2021 edition, as the report from 2022 has not yet been published at the time of this writing.

Within VNN 2021, a total of 9 benchmarks were considered, of which 7 represented test cases for local robustness verification of image classification networks. Benchmarks were proposed by the participants themselves and included a total of 11 CIFAR and 7 MNIST networks, which differed in terms of architecture components, such as non-linearities (*e.g.*, ReLU, Tanh, Sigmoid) and layer operations (*e.g.*, convolutional or pooling layers). Networks were trained on the CIFAR-10 and MNIST datasets, respectively. Moreover, each benchmark was composed of random image subsets, excluding images that were misclassified by the given network, along with varying perturbation radii.

This competition overcame several of the previously reported limitations with regard to the evaluation of network verifiers. Most notably, it covered a relatively large and diverse set of neural networks. Moreover, thanks to the active participation from the community, 12 verification algorithms were included in the competition. At the same time, we see room for further research into the performance of neural network verifiers.

Firstly, it is not clearly stated in the VNN competition report [18] why verifiers participated in certain competition categories, but not in others. While we assume this to be due to compatibility reasons, we could not find any formal explanation in the report, although this information could provide insight into relevant problem classes and suitable algorithms for solving these.

Secondly, GPU-accelerated tools were directly compared to those that rely solely on CPU resources, which we argue does not give due credit to the structural differences between these classes of algorithms. GPU-accelerated approaches employ advanced parallelisation schemes, giving rise to substantially higher computational demands than those required by methods running on CPUs, especially when restricted to a single core. For example, an Amazon EC2 GPU-based instance with 32 CPU cores costs around 1.4 times more than an equivalent CPU instance.[2]

Lastly, the competition seeks to determine the current state of the art; however, the competition ranking and scores do not sufficiently quantify the extent to which an algorithm actually contributes to the state of the art. In other words, it is in the nature of competitions to determine a winner, at least implicitly suggesting that a single approach generally outperforms all competitors. However, some verification algorithms might have limited but distinct areas of strength, which cannot be identified through aggregated performance measures, such as the total number of verified instances. Although the competition report [18] shows that individual verifier performance differs among benchmarks, it remains unclear whether all algorithms solve the same set of instances in the given benchmark, or if they complement each other. Similarly, it does not reveal whether or not methods are correlated in their performance.

4. Verification Algorithms under Assessment

We consider 5 complete, CPU-based neural network verification algorithms; each of these was chosen because it fulfilled one of the following conditions: it was (*i*) ranked among the top five verification methods according to the 2021 VNN competition or (*ii*) supported by the recently published DNNV framework [21]. Altogether, we consider our set of algorithms to be representative of the developments in the area of complete neural network and, more specifically, local robustness verification.

All methods were employed with their default hyperparameter settings, as they would likely be used by practitioners. We note that the performance of a verifier might improve if its hyperparameters were optimised specifically for the given benchmark; however, most verifiers have dozens of hyperparameters (or employ combinatorial solvers that come with their own, extensive set of hyperparameters), which makes this a non-trivial task, requiring additional expertise and resources (see, *e.g.*, König et al. [39]). The verification algorithms we considered are the following:

BaB. The algorithm proposed by Bunel et al. [5] restates the verification problem as a global optimisation problem, which is then solved using branch-and-bound search. It further incorporates algorithmic improvements

[2] See https://aws.amazon.com/ec2/pricing/on-demand/

with regard to branching and bounding procedures such as *smart branching*; *i.e.*, before splitting, it computes fast bounds on each of the possible subdomains and chooses the one with the tightest bounds. We refer to this method as BaBSB for the remainder of this paper. BaBSB supports ReLU-based networks.

Marabou. The Marabou framework [23] employs SMT solving techniques, specifically the lazy search technique for handling non-linear constraints. Furthermore, Marabou employs deduction techniques to obtain information on the activation functions that can be used to simplify them. The core of the SMT solver is simplex-based, which means that the variable assignments are made using the simplex algorithm. Marabou supports ReLU and Sigmoid activations as well as MaxPooling operations.

Neurify. The verification algorithm proposed by Wang et al. [33] relies on symbolic interval propagation to create over-approximations, followed by a refinement strategy based on symbolic gradient information. The constraint refinement aims to tighten the bounds of the approximation of activation functions. Neurify can process networks containing ReLU activations.

nnenum. The verifier proposed by Bak et al. [38] utilises star sets to represent the values each layer of a neural network can attain. By propagating these through the network, it checks whether one or more of the star sets results in an adversarial example. This verifier can handle networks with ReLU activations.

Verinet. The verifier developed by Henriksen and Lomuscio [9] combines symbolic intervals with gradient-based adversarial local search for finding counter-examples. The authors further propose a splitting heuristic for interval propagation based on the influence of a given node on the bounds of the network output. Verinet supports networks containing ReLU, Sigmoid and Tanh activations.

5. Setup for Empirical Evaluation

In this work, we seek to provide a clearer picture of the state of the art in neural network verification. More specifically, we argue that the state of the art is not just defined by a single verification algorithm, as there might be verifiers which on their own perform poorly but still make meaningful contributions by excelling on limited instance subsets that are challenging for other verification methods.

In the following, we will present an overview of how we set up our benchmark study, *i.e.*, how we selected problem instances and verification algorithms. Furthermore, we will provide details on the software we used and the execution environment in which our experiments were carried out.

5.1. Problem Instances

For our assessment, we compiled a high-quality set of problem instances for local robustness verification. Following best practices in other research areas, such as optimisation [40, 41], the benchmark should be *representative* and *diverse*, where the former refers to how well the difficulty of the benchmark is aligned with that of real-world instances from the same problem class, and the latter means that the problem set should generally contain problems with a wide range of difficulties.

Overall, our benchmark is comprised of 79 image classification networks, of which 38 are trained on the CIFAR-10 dataset and 41 are trained on the MNIST dataset. To ensure the representativeness of the problem set, all networks were sampled from the neural network verification literature, *i.e.*, networks used in existing work on local robustness verification and provided in public repositories; in other words, the characteristics of the networks in our benchmark are assumed to match those of networks generally used for evaluating verification algorithms.

We further want our problem set to be diverse. Therefore, we ensured that the considered networks differ in size, *i.e.*, the number of hidden layers and nodes, as well as the type of non-linearities (*e.g.*, ReLU or Tanh) or layer operations (*e.g.*, pooling layers) they employ.

For each network, we verified 100 local robustness properties; more precisely, we sampled 100 images from the dataset on which the network has been trained and verified for local robustness with perturbation radius ϵ set at 0.012. This perturbation radius was chosen to be a median of values we found in literature: the smallest value we found was 1/255 in the work by Li et al. [22], whereas Botoeva et al. [4] or Wang et al. [13] utilised larger values, such as 0.05.

Lastly, we split our benchmark set into different categories based on verifier compatibilities. This means a verifier is only employed to categories it is able to process. The categories as well as the instance set size for each category are shown in Table 1.

5.2. Evaluation Metrics

In order to assess the performance of the various methods, we compute four performance metrics: the average running time, the number of solved instances, the *relative marginal contribution* and the *Shapley value* [42] of each verifier. Although these metrics present aggregated measures, they reflect algorithm performance on an instance level and in relation to other methods included in the comparison; a more detailed explanation will be provided in the following paragraphs. Notice that we do not penalise timeouts when computing average running time; *i.e.*, the maximum running time equals the given time limit.

Table 1
Instance set size for each benchmark category. Solvable instances are those solved by at least one (*i.e.*, any) or all of the considered verifiers. The column "Verifiers employed" lists the matching suitable verification algorithm(s) to the respective category.

Category	MNIST			CIFAR			Verifiers employed
	Total	Solvable		Total	Solvable		
		Any	All		Any	All	
ReLU	2 500	1 913	42	2 500	972	0	BaBSB, Marabou, Neurify, nnenum, Verinet
ReLU + MaxPool	400	5	5	100	0	0	Marabou
Tanh	600	556	556	600	0	0	Verinet
Sigmoid	600	581	0	600	0	0	Marabou, Verinet

The marginal contribution is computed as follows. Define c as a set of given verifiers and let $v(c)$ be the total score of set c. Here, the total score $v(c)$ consists of the number of instances verified by at least one verifier in set c. We compute the marginal contribution per algorithm to determine how much the total performance of all algorithms (in terms of solved instances) decreases when the given algorithm is removed from the set of all algorithms if they were employed in a parallel algorithm portfolio. Such portfolios are sets of algorithms that are run in parallel on each given problem instance and complement each other in terms of performance across an instance set [43]. Formally, to determine the marginal contribution of any of the verifiers i to portfolio c, one needs to know the value of $v(c)$ and $v(c \setminus \{i\})$, where $c \setminus \{i\}$ is the portfolio minus verifier i. Thus, the marginal contribution of verifier i is expressed as

$$MC_i = v(c) - v(c \setminus \{i\}) \tag{1}$$

Following this terminology, we can define the number of solved instances by verifier i as a set consisting only of verifier i, $Solved_i = v(i) - v(\emptyset)$. In other words, the number of solved instances employs a set of size one and the marginal contribution employs a set of all verifiers under consideration. The *relative* marginal contribution then represents the marginal contribution of a given verifier as a proportion of the sum of every method's absolute marginal contribution.

Lastly, the Shapley value is the average marginal contribution of a verifier over all possible joining orders, where joining order refers to the order in which the verifiers are added to a parallel portfolio. This value is complementing the previous two metrics, as it does not assume a particular order in which algorithms are added to the portfolio. To be precise, the number of solved instances simply represents a joining order in which the considered algorithm comes first, whereas the marginal contribution metric assumes a joining order in which it comes last. However, using fixed orders, as it is the case for the marginal contribution, might not reveal possible interactions between the given method and other algorithms, *e.g.*, it might understate the importance of a single algorithm given the presence of another, highly correlated algorithm. In such a case, both algorithms would not be assigned any marginal contribution, even though one of them should be included in a potential portfolio. This is captured by the Shapley value: define n as the number of verifiers under consideration and C as the set of all combinations of all subsets of verifiers under consideration including the empty set, where set C will be of size $\sum_{k=0}^{n} \frac{n!}{k! \cdot (n-k)!}$. Finally, define C_i as all sets in which verifier i is present. The Shapley value of verifier i is then calculated as

$$\phi_i = \frac{1}{n!} \cdot \sum_{j \in C_i} (v(j) - v(j \setminus \{i\})) \tag{2}$$

5.3. Execution Environment and Software Used

Our experiments were carried out on a cluster of machines equipped with Intel Xeon E5-2683 CPUs with 32 cores, 40 MB cache size and 94 GB RAM, running CentOS Linux 7. For each verification method, we limited the number of available CPU cores to a single core. Each query was given a time budget of 3 600 seconds and a memory budget of 3 GB. Generally, we executed the verification algorithms through the DNNV interface, version 0.4.8. DNNV is a framework that transforms a network and property specification into a unified format, which can then be solved by a given method [21]. More specifically, DNNV takes as input a network in the ONNX format, along with a property specification, and then translates the network and property to the input format required by the verifier. After running the verifier on the transformed problem, it returns either sat if the property was falsified or $unsat$ if the property was proven to hold.

Table 2
Performance comparison of local robustness verification algorithms in terms of the number of solved instances, relative marginal contribution (*RMC*), Shapley value (ϕ) and average CPU running time, computed for each category with ϵ set at 0.012.

ReLU Verifier	MNIST				CIFAR			
	Solved	*RMC*	ϕ	Avg. Time [CPU s]	Solved	*RMC*	ϕ	Avg. Time [CPU s]
BaBSB	358	0.22	118	3 241	307	0.00	86	2 924
Marabou	1 001	0.19	312	1 801	400	0.00	117	2 153
Neurify	871	0.25	265	1 964	915	0.75	411	235
nnenum	1 754	0.17	600	389	76	0.05	28	3 337
Verinet	1 799	0.16	618	263	841	0.20	330	500
ReLU+Maxpool Verifier	MNIST				CIFAR			
	Solved	*RMC*	ϕ	Avg. Time	Solved	*RMC*	ϕ	Avg. Time
Marabou	5	1.00	5	57	0	0.00	0	3 600
Tanh Verifier	MNIST				CIFAR			
	Solved	*RMC*	ϕ	Avg. Time	Solved	*RMC*	ϕ	Avg. Time
Verinet	556	1.00	556	55	0	0.00	0	3 600
Sigmoid Verifier	MNIST				CIFAR			
	Solved	*RMC*	ϕ	Avg. Time	Solved	*RMC*	ϕ	Avg. Time
Marabou	0	0.00	0	3 600	0	0.00	0	3 600
Verinet	581	1.00	581	55	0	0.00	0	3 600

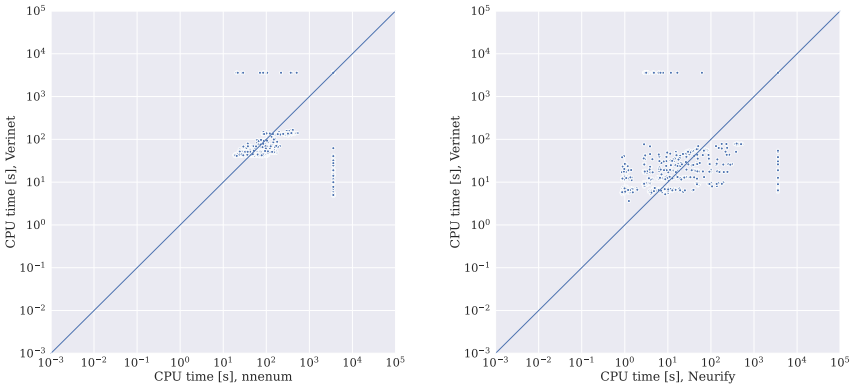
6. Results and Discussion

In the following, we provide an in-depth discussion of the results obtained from our experiments. Table 1 shows the categories we devised, along with the resulting instance set sizes as well as information on which verifier has been employed for each category.

Table 2 reports the number of problem instances solved by each verifier per network category, the relative marginal contribution, the Shapley value and the average running time. On ReLU-based MNIST networks, we found Verinet to be the best-performing verifier, solving 1 799 out of 2 500 instances, while achieving a Shapley value of 618. However, taking relative marginal contribution into account, we found that Neurify achieved the highest relative marginal contribution of 0.25 (compared to 0.16 for Verinet), indicating that it could verify a sizable fraction of instances on which other methods failed to return a solution. Moreover, the relative marginal contribution scores show that each method could solve a sizeable fraction of instances unsolved by any of the other methods.

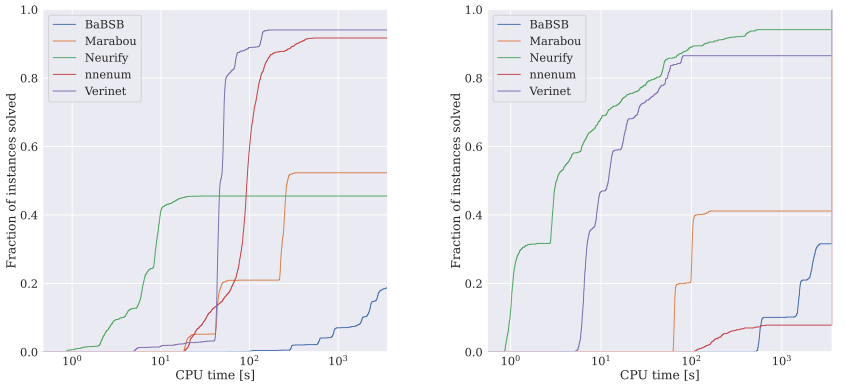
On ReLU-based CIFAR networks, it should first be noted that no verification problem instance can be solved by *all* verifiers, highlighting the structural differences between instances and the sensitivity of the verification approaches to those instances. That said, Neurify slightly outperformed Verinet in terms of the number of solved instances (915 *vs* 841 out of 2 500). Furthermore, Neurify achieved a much larger relative marginal contribution than Verinet (0.75 *vs* 0.20), which means that the former could solve a relatively large number of instances which could not be solved by the other methods. Generally, relative marginal contribution scores are much less evenly distributed among verifiers when compared to the MNIST dataset.

Figures 1a and 1b show an instance-level comparison of the two best-performing algorithms (in terms of Shapley value) in the ReLU category for each dataset. In Figure 1a, we see that on MNIST networks, both Verinet and nnenum solved instances that the other one could not solve within the given time budget. Concretely, when considering a parallel portfolio containing both algorithms, the number of solved instances slightly increases to 1 817 out of 2 500 (*vs* 1 799 solved by Verinet and 1 754 solved by nnenum alone), while supplied with similar CPU resources (*i.e.*, 1 800 CPU seconds per verifier, which represents half of the total given time budget of 3 600



(a) MNIST (b) CIFAR

Figure 1: Performance comparison of the two top-performing verification methods (in terms of Shapley value) in the ReLU category on (a) MNIST and (b) CIFAR networks. The plots show that each verifier outperforms the other on some instances, while none of the methods is dominating in performance across the entire instance set. This illustrates the complementary strengths of the verification algorithms. The diagonal line indicates equal performance of the two methods.



(a) MNIST (b) CIFAR

Figure 2: Fraction of instances solved by the considered verification algorithms in the ReLU category as a function of CPU running time for (a) MNIST and (b) CIFAR networks. On (a), we find that Verinet solves most instances in the least time, while on (b) Neurify does.

CPU seconds per verification query as used in our experiments). Notice that leveraging parallel portfolios has already been shown to significantly improve the performance of MIP-based verification methods [39].

On CIFAR instances, we found Neurify and Verinet to also have distinctive strengths over each other. This is shown in Figure 1b, where both algorithms could solve a substantial amount of instances that the other could not return a solution for. Thus, when combined in a parallel portfolio, 963 instances can be solved (*vs* 915 solved by Neurify and 841 solved by Verinet alone, out of 2 500 instances), while using the same amount of CPU resources, *i.e.*, 1 800 CPU seconds per verifier. These findings further emphasise the complementarity between the verification algorithms considered in our study. All remaining verifiers achieved substantially lower Shapley values and relative marginal contribution scores, indicating that they would not complement Neurify and Verinet well in a portfolio.

Figure 2a shows the cumulative distribution function of running times over the MNIST problem instances. As seen in the figure, Verinet tends to solve these problem instances fastest; however, Neurify tended to show even better performances on those instances it was able to solve. We note that most of the instances unsolved by Neurify represent networks that were trained on images with 3 dimensions, whereas Neurify requires images used as network inputs to have 2 or 4 dimensions.

Figure 2b shows a similar plot for the CIFAR problem instances. Here, Neurify solved the largest fraction in less time than other methods. This suggests that Neurify is a very competitive verifier when applicable to the specific network or input format.

For each of the remaining categories, we found that there is only one verifier that could effectively handle the respective problem instances. Specifically, instances from the ReLU+MaxPooling category can be processed by Marabou, although, only a modest number of MNIST instances could be solved in this way. Networks containing Tanh activations can, in principle, be verified by Verinet, but the algorithm did also not solve any CIFAR instances. Lastly, Sigmoid-based networks can be handled by both Verinet and Marabou, however, only the former could solve MNIST instances within the given compute budget.

7. Conclusions and Future Work

In this work, we assessed the performance of several local robustness verification algorithms, *i.e.*, algorithms used to verify the robustness of an image classification network against small input perturbations. To conclude, we found that all considered methods support ReLU-based networks, while other network types are strongly undersupported. While this has been suspected in the community, it has, to our knowledge, not yet been subject to formal study. Furthermore, we presented evidence for strong performance complementarity: even within the same benchmark category, two verification systems outperform each other on distinct sets of instances. As we have demonstrated, this complementarity can be exploited by combining individual verifiers into parallel portfolios. Lastly, we showed that, in general, the performance of verifiers strongly differs between image datasets, with some methods achieving the best performance on MNIST (in terms of the number of solved instances and average running time) while falling behind on CIFAR and vice versa. In future work, it would be interesting to include a broader set of perturbation radii and analyse in more detail how the relative performance of verifiers depends on the given radius. Furthermore, we are interested in expanding our analysis to GPU-based verification algorithms.

Acknowledgements

This research was partially supported by TAILOR, a project funded by EU Horizon 2020 research and innovation program under GA No. 952215.

References

[1] K. D. Julian, M. J. Kochenderfer, M. P. Owen, Deep neural network compression for aircraft collision avoidance systems, Journal of Guidance, Control, and Dynamics 42 (2019) 598–608.

[2] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, R. Fergus, Intriguing properties of neural networks, arXiv preprint arXiv:1312.6199 (2014).

[3] O. Bastani, Y. Ioannou, L. Lampropoulos, D. Vytiniotis, A. Nori, A. Criminisi, Measuring Neural Net Robustness with Constraints, in: Advances in Neural Information Processing Systems (NeurIPS 2016), 2016, pp. 2613–2621.

[4] E. Botoeva, P. Kouvaros, J. Kronqvist, A. Lomuscio, R. Misener, Efficient Verification of ReLU-based Neural Networks via Dependency Analysis, in: Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI-20), 2020, pp. 3291–3299.

[5] R. R. Bunel, I. Turkaslan, P. Torr, P. Kohli, P. K. Mudigonda, A Unified View of Piecewise Linear Neural Network Verification, in: Advances in Neural Information Processing Systems (NeurIPS 2018), 2018, pp. 1–10.

[6] K. Dvijotham, R. Stanforth, S. Gowal, T. A. Mann, P. Kohli, A Dual Approach to Scalable Verification of Deep Networks, in: Proceedings of the 38th Conference on Uncertainty in Artificial Intelligence (UAI 2018), 2018, pp. 550–559.

[7] R. Ehlers, Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks, in: Proceedings of the 15th International Symposium on Automated Technology for Verification and Analysis (ATVA 2017), 2017, pp. 269–286.

[8] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, M. Vechev, AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation, in: Proceedings of the 39th IEEE Symposium on Security and Privacy (IEEE S&P 2018), 2018, pp. 3–18.

[9] P. Henriksen, A. Lomuscio, Efficient Neural Network Verification via Adaptive Refinement and Adversarial Search, in: Proceedings of the 24th European Conference on Artificial Intelligence (ECAI 2020), 2020, pp. 2513–2520.

[10] G. Katz, C. Barrett, D. L. Dill, K. Julian, M. J. Kochenderfer, Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks, in: Proceedings of the 29th International Conference on Computer Aided Verification (CAV 2017), 2017, pp. 97–117.

[11] K. Scheibler, L. Winterer, R. Wimmer, B. Becker, Towards Verification of Artificial Neural Networks, in: Proceedings of the 18th Workshop on Methoden und Beschreibungssprachen zur Modellierung

und Verifikation von Schaltungen und Systemen (MBMV 2015), 2015, pp. 30–40.

[12] V. Tjeng, K. Xiao, R. Tedrake, Evaluating Robustness of Neural Networks with Mixed Integer Programming, in: Proceedings of the 7th International Conference on Learning Representations (ICLR 2019), 2019, pp. 1–21.

[13] S. Wang, K. Pei, J. Whitehouse, J. Yang, S. Jana, Efficient Formal Safety Analysis of Neural Networks, in: Advances in Neural Information Processing Systems (NeurIPS 2018), 2018, pp. 6369–6379.

[14] W. Xiang, H.-D. Tran, T. T. Johnson, Output Reachable Set Estimation and Verification for Multilayer Neural Networks, IEEE Transactions on Neural Networks and Learning Systems 29 (2018) 5777–5783.

[15] I. J. Goodfellow, J. Shlens, C. Szegedy, Explaining and Harnessing Adversarial Examples, arXiv preprint arXiv:1412.6572 (2014).

[16] N. Papernot, P. McDaniel, X. Wu, S. Jha, A. Swami, Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks, in: Proceedings of the 37th IEEE Symposium on Security and Privacy (IEEE S&P 2016), 2016, pp. 582–597.

[17] M. Casadio, E. Komendantskaya, M. L. Daggitt, W. Kokke, G. Katz, G. Amir, I. Refaeli, Neural Network Robustness as a Verification Property: A Principled Case Study, in: Proceedings of the 34rd International Conference on Computer Aided Verification (CAV 2022), 2022, pp. 219–231.

[18] S. Bak, C. Liu, T. Johnson, The Second International Verification of Neural Networks Competition (VNN-COMP 2021): Summary and Results, arXiv preprint arXiv:2109.00498 (2021).

[19] L. Xu, F. Hutter, H. H. Hoos, K. Leyton-Brown, Satzilla: Portfolio-based algorithm selection for sat, Journal of Artificial Intelligence Research 32 (2008) 565–606.

[20] S. Kadioglu, Y. Malitsky, A. Sabharwal, H. Samulowitz, M. Sellmann, Algorithm Selection and Scheduling, in: Proceedings of the Seventeenth International Conference on Principles and Practice of Constraint Programming (CP2011), 2011, pp. 454–469.

[21] D. Shriver, S. Elbaum, M. B. Dwyer, DNNV: A Framework for Deep Neural Network Verification, in: Proceedings of the 33rd International Conference on Computer Aided Verification (CAV 2021), 2021, pp. 137–150.

[22] L. Li, X. Qi, T. Xie, B. Li, Sok: Certified robustness for deep neural networks, arXiv preprint arXiv:2009.04131 (2020).

[23] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljić, D. L. Dill, M. J. Kochenderfer, C. Barrett, the Marabou Framework for Verification and Analysis of Deep Neural Networks, in: Proceedings of the 31st International Conference on Computer Aided Verification (CAV 2019), 2019, pp. 443–452.

[24] L. Pulina, A. Tacchella, Checking Safety of Neural Networks with SMT Solvers: A Comparative Evaluation, in: AI*IA, 2011, pp. 127–138.

[25] L. Pulina, A. Tacchella, NeVer: A Tool for Artificial Neural Networks Verification, Annals of Mathematics and Artificial Intelligence (2011) 403–425.

[26] L. Pulina, A. Tacchella, Challenging SMT Solvers to Verify Neural Networks, AI Communications (2012) 117–135.

[27] L. d. Moura, N. Bjørner, Satisfiability Modulo theories: An Appetizer, in: Proceedings of the Brazilian Symposium on Formal Methods (SBMF 2009), 2009, pp. 23–36.

[28] S. Dutta, S. Jha, S. Sankaranarayanan, A. Tiwari, Output Range Analysis for Deep Neural Networks, in: Proceedings of the Tenth NASA Formal Methods Symposium (NFM 2018), 2018, pp. 121–138.

[29] A. Lomuscio, L. Maganti, An approach to reachability analysis for feed-forward ReLU neural networks, arXiv preprint arXiv:1706.07351 (2017).

[30] A. H. Land, A. G. Doig, An Automatic Method of Solving Discrete Programming Problems, Econometrica (1960) 497–520.

[31] R. Bunel, I. Turkaslan, P. H. S. Torr, M. P. Kumar, J. Lu, P. Kohli, Branch and Bound for Piecewise Linear Neural Network Verification, Journal of Machine Learning Research (2020) 1574–1612.

[32] A. De Palma, R. Bunel, A. Desmaison, K. Dvijotham, P. Kohli, P. H. S. Torr, M. P. Kumar, Improved Branch and Bound for Neural Network Verification via Lagrangian Decomposition, arXiv preprint arXiv:2104.06718 (2021).

[33] S. Wang, K. Pei, J. Whitehouse, J. Yang, S. Jana, Formal Security Analysis of Neural Networks using Symbolic Intervals, in: Proceedings of the 27th USENIX Security Symposium (USENIX Security 18), 2018, pp. 1599–1614.

[34] S. Wang, H. Zhang, K. Xu, X. Lin, S. Jana, C.-J. Hsieh, Z. Kolter, Beta-CROWN: Efficient Bound Propagation with Per-neuron Split Constraints for Neural Network Robustness Verification, in: Advances in Neural Information Processing Systems (NeurIPS 2021), 2021, pp. 29909–29921.

[35] G. Singh, T. Gehr, M. Püschel, M. Vechev, An Abstract Domain for Certifying Neural Networks, in: Proceedings of the 46th ACM SIGPLAN Symposium on Principles of Programming Languages (ACM-POPL 2019), 2019, pp. 1–30.

[36] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, L. Daniel, Efficient Neural Network Robustness Certification with General Activation Functions, Advances in Neural Information Processing Sys-

tems (NeurIPS 2018) 31 (2018) 4944–4953.
[37] G. Singh, T. Gehr, M. Mirman, M. Püschel, M. Vechev, Fast and Effective Robustness Certification, in: Advances in Neural Information Processing Systems (NeurIPS 2018), 2018, pp. 1–12.
[38] S. Bak, H.-D. Tran, K. Hobbs, T. T. Johnson, Improved Geometric Path Enumeration for Verifying ReLU Neural Networks, in: Proceedings of the 32nd International Conference on Computer Aided Verification (CAV 2020), 2020, pp. 66–96.
[39] M. König, H. H. Hoos, J. N. v. Rijn, Speeding up neural network robustness verification via algorithm configuration and an optimised mixed integer linear programming solver portfolio, Machine Learning 111 (2022) 4565–4584.
[40] H. H. Hoos, T. Stützle, Stochastic Local Search: Foundations & Applications, Elsevier / Morgan Kaufmann, 2004.
[41] T. Bartz-Beielstein, C. Doerr, D. v. d. Berg, J. Bossek, S. Chandrasekaran, T. Eftimov, A. Fischbach, P. Kerschke, W. La Cava, M. Lopez-Ibanez, et al., Benchmarking in Optimization: Best Practice and Open Issues, arXiv preprint arXiv:2007.03488 (2020).
[42] A. Fréchette, L. Kotthoff, T. Michalak, T. Rahwan, H. Hoos, K. Leyton-Brown, Using the shapley value to analyze algorithm portfolios, in: Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI-16), 2016, pp. 3397–3403.
[43] C. P. Gomes, B. Selman, Algorithm portfolios, Artificial Intelligence 126 (2001) 43–62.