Critically assessing the state of the art in neural network verification

König, H.M.T.; Bosman, A.W.; Hoos, H.H.; Rijn, J.N. van

Journal of Machine Learning Research 25 (2024) 1-53 Submitted 1/23; Published 1/24

Critically Assessing the State of the Art in Neural Network Verification

Matthias König[1] H.M.T.KONIG@LIACS.LEIDENUNIV.NL
Annelot W. Bosman[1] A.W.BOSMAN@LIACS.LEIDENUNIV.NL
Holger H. Hoos[1,2,3] HH@CS.RWTH-AACHEN.DE
Jan N. van Rijn[1] J.N.VAN.RIJN@LIACS.LEIDENUNIV.NL

[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*

Editor: Scott Niekum

Abstract

Recent research has proposed various methods to formally verify neural networks against minimal input perturbations; this verification task is also known 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, the problem instances encountered when performing local robustness verification differ based on the network to be verified, the property to be verified and the specific network input. This raises the question of which verification algorithm is most suitable for solving specific types of instances of the local robustness verification problem. To answer this question, we performed a systematic performance analysis of several CPU- and GPU-based local robustness verification systems on a newly and carefully assembled set of 79 neural networks, of which we verified a broad range of robustness properties, while taking a practitioner's point of view – a perspective that complements the insights from initiatives such as the VNN competition, where the participating tools are carefully adapted to the given benchmarks by their developers. Notably, we show that no single best algorithm dominates 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. We quantify this complementarity using various performance measures, such as the Shapley value. 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., 2019). Concurrently, it is now well known that neural networks are vulnerable

to *adversarial attacks* (Szegedy et al., 2014), where a given input is manipulated, often in subtle ways, such that it is misclassified by the network.

In the case of image recognition tasks, the perturbation required to trigger a misclassification, 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 (Bastani et al., 2016; Botoeva et al., 2020; Bunel et al., 2018; Dvijotham et al., 2018; Ehlers, 2017; Gehr et al., 2018; Henriksen and Lomuscio, 2020; Katz et al., 2017; Scheibler et al., 2015; Tjeng et al., 2019; Wang et al., 2018b; Xiang et al., 2018). 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 pre-defined norms, such as the l_∞-norm (Goodfellow et al., 2015; Papernot et al., 2016).

Neural network verification with respect to local robustness 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, and 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 non-trivial for researchers or practitioners to assess and decide which method is most suitable for verifying a given neural network (Casadio et al., 2022). 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, as we will further outline in Section 2.2.

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 (Müller et al., 2023). While the results from these competitions have provided valuable insights into the general progress in neural network verification, several questions remain unexplored. Most importantly, the ranking of algorithms based on their aggregated performance scores makes it difficult to assess in detail the strengths or weaknesses of verifiers on different instances. Indeed, looking at the competition results, one easily gets 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. (2008) or Kadioglu et al. (2011) 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 (Shriver et al., 2021); we, therefore, believe that our findings are broadly applicable. Moreover, we seek to go beyond existing benchmarking

approaches and shed light on previously unanswered questions regarding the state of the art in local robustness verification from a practitioner's point of view – a perspective that complements the insights from the VNN competition, where the participating tools are carefully adapted to the given benchmarks by their developers. Our contributions are as follows:

- We analyse the current state of practice in benchmarking verification algorithms;
- we perform a systematic benchmarking study of several, carefully chosen GPU- and CPU-based verification methods based on a *newly assembled* and diverse set of networks, including 38 CIFAR and 41 MNIST networks with different activation functions, representing a much larger number of networks than typically considered, each verified against several robustness properties, for which we expended a total of approximately 1 GPU and 16 CPU years in running time;
- we present a categorisation of verification benchmarks based on verifier compatibilities with different layer types and operations;
- 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, a finding that we also show to hold for the results of the 2022 VNN Competition – *e.g.*, while the verifiers nnenum and PeregriNN achieved competitive performance in the *FC* category of the competition, the former solved many instances unsolved by the latter and vice versa;
- lastly, we provide a public repository containing all experimental data, along with the newly assembled network collection.[1]

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.*, a pixel in a given input image, is predefined and indicated by the perturbation radius ϵ. It should be noted that recent work has proposed to move beyond pixel-based perturbations, which account for realistic sensor errors, to *semantic* perturbations, *i.e.*, linear transformations representing changes in contrast, luminosity, scaling, rotation, and other factors (Mohapatra et al., 2020; Henriksen et al., 2021). This specific type of verification, however, falls outside the scope of our study.

Formal verification algorithms can be either *complete* or *incomplete* (Li et al., 2020). An algorithm that is incomplete does not guarantee to report a solution for every given instance; however, incomplete verification algorithms are typically *sound*, which means

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

they will report that a property holds only if the property actually holds. On the other hand, an algorithm that is sound and complete, when given sufficient resources to be run to completion, will correctly state that a property holds *whenever* it holds, and, in particular, will determine accurately when the property does not hold. 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. Furthermore, we focus on the verification of real-valued networks, which are typically considered in the verification literature, although there exist methods for the verification of other network types; see, *e.g.*, the work of Narodytska et al. (2018) or Jia and Rinard (2020) on binarised networks.

2.1 Algorithmic Approaches

Early work on complete verification of neural networks utilised *satisfiability modulo theories (SMT)* solvers (Katz et al., 2017, 2019; Pulina and Tacchella, 2011b,a, 2012), which determine whether a set of logic constraints is satisfiable (Moura and Bjørner, 2009). The resulting verification problems are NP-complete and challenging to solve in practice. Some SMT-based verification algorithms, such as those proposed by Katz et al. (2017, 2019), employ the well-known simplex algorithm (Dantzig, 2002) for assigning values to the SMT variables.

Alternatively, it is possible to formulate the verification task as a constraint optimisation problem using *mixed integer programming (MIP)* (Botoeva et al., 2020; Dutta et al., 2018; Lomuscio and Maganti, 2017; Tjeng et al., 2019). MIP solvers essentially optimise an objective function subject to a set of constraints. Generally, optimisation problems are well studied, and by approaching verification tasks from that angle, techniques and insights from well-developed areas of computer science and operations research can be leveraged. MIP-based verification algorithms assign variables to each node in the network and, more specifically, encode non-linearities by means of binary variables indicating whether a node is in an active or inactive state. Approaches differ in the way perturbations are encoded into the program as well as in the specification of their objective function. 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 (Land and Doig, 2010) for solving the verification problem, regardless of whether it is modelled as MIP or SMT (Bunel et al., 2020, 2018; De Palma et al., 2021; Ehlers, 2017; Wang et al., 2018b). Neural network verification algorithms based on branch-and-bound consist of two main steps: (*i*) branching and (*ii*) bounding. Branching involves splitting the domain of one or more variables (based on the nodes in the network) into sub-problems of the original problem. These (relaxed) sub-problems are then solved by cheap but incomplete verification algorithms, which determine a lower bound to the verification problem, while upper bounds are found via falsification algorithms (Dong et al., 2018). By repeating these steps, the bounds, *i.e.*, the upper and lower limits on the possible value of a solution to the verification problem, are tightened in each iteration. There exist many different branching schemes and bounding algorithms, which vary in tightness of the lower bounds and general performance.

To formulate the constraints used in the above-mentioned methods, the non-linear activation functions of a neural network are usually relaxed. This is mostly done by approximating the original non-linear activation function by at least two linear functions, forming upper and lower bounds (Ehlers, 2017; Singh et al., 2019b; Weng et al., 2018; Wong and Kolter, 2018; Zhang et al., 2018). Employing the linear bounds as relaxation to the activation function increases the feasible region of each variable in the model, and as the nodes in each layer are dependent on the previous layer, the bounds on each consecutive layer become looser. The approximation, thus, provides a trade-off, as loose and fast bounds lead to large feasible regions while obtaining tight bounds tends to be computationally expensive. The way in which non-linearities are approximated presents a key distinguishing factor amongst complete verification algorithms.

Alternatively, it has been proposed to use *symbolic interval propagation* to compute bounds on the output range of the network for a given input and use those as additional constraints (Botoeva et al., 2020; Henriksen and Lomuscio, 2020; Wang et al., 2018a, 2021). The output range of the output layer is obtained by propagating the bounds through the network, which renders it unnecessary to encode the entire network and use computationally expensive solvers. Symbolic interval propagation lends itself as an incomplete verification method, but it can also complement complete methods, improving their efficiency by reducing the size of the feasible region of the problem, compared to the looser approximation described above.

Other bound approximation methods include *polyhedra*, *zonotope* and *star-set abstraction*. Polyhedra abstraction produces one lower bound for the approximation based on the trained network, instead of two bounds as used in symbolic interval propagation, where the latter results in tighter bounds (Singh et al., 2019b; Zhang et al., 2018). Zonotope abstraction is similar to polyhedra abstraction and is able to model dependencies between the zonotope representation of different network layers (Gehr et al., 2018; Singh et al., 2018). In contrast to polyhedron transformations, the zonotope transformations scale polynomially, and optimisation is efficient. Star sets are a generalisation of the zonotope abstraction, as they are not restricted to being symmetric (Bak et al., 2020). They are similar to zonotopes; however, optimisation is less efficient, as it requires solving a linear program.

2.2 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 *always* outperforms all others. 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 (Müller et al., 2023). The VNN competition was held in the years 2020, 2021 and 2022, with different protocols (*e.g.*, for running experiments, scoring, etc.), benchmarks and participants. Here, we focus on the most recent, 2022 edition. Within VNN 2022, a total of 12 benchmarks were considered, of which 6 represented test cases for local robustness verification of image classification networks. Notice that one of these benchmarks considers bias field perturbations, which are reduced to a standard l_∞-norm specification. Benchmarks were proposed by the participants themselves and included a total of 13 CIFAR, 2 MNIST and 2 (Tiny)ImageNet 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, skip connections). Networks were trained on the CIFAR-10, CIFAR-100, MNIST, TinyImageNet and ImageNet 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.

First and foremost, 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 (Müller et al., 2023) 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.

Furthermore, in our study, we conducted both a joint and separate analysis of CPU- and GPU-based methods. This choice was motivated by the inherent challenges that arise when attempting to compare these two types of algorithms. Indeed, the competition results suggest that GPU-based methods are more efficient than CPU-based algorithms (Müller et al., 2023); however, GPU resources are typically more expensive to run. Additionally, while CPU-based methods can run a single verification query on each CPU core, allowing for multiple instances to be solved in parallel on the same machine, GPU-based methods utilise the full GPU when solving a single verification query. In fact, running multiple queries in parallel, each utilising a single CPU core, might be a more efficient approach than running each query sequentially, while utilising all cores. Thus, overall, it remains challenging to set up a comparison between CPU- and GPU-based verification algorithms in an unbiased manner, which is why we present both a direct comparison and a separate analysis.

Finally, the competition approaches the state of the art from the perspective of a tool developer, where the developer is given access to the benchmarks beforehand and can adapt their implementations as well as hyperparameter settings accordingly. On the other hand, in this study, we assess the state of the art from the perspective of a practitioner, who typically

uses a verification tool out of the box, is bounded by the limitations of the implementations, and might also not be able to tune the hyperparameters of these tools. We believe that both these perspectives on the state of the art are valid and give complementary insights.

2.3 Algorithm Portfolios

In cases where the performance of an algorithm varies greatly from one instance to another and where the performance of several different algorithms complements each other across an instance distribution, one can make use of *algorithm portfolios* (Gomes and Selman, 2001; Huberman et al., 1997). Such portfolios combine multiple algorithms in such a way that a much broader range of performance characteristics can be exploited, compared to running a single algorithm on its own.

Algorithm portfolios can employ all algorithms in a parallel fashion or, alternatively, provide the basis for per-instance algorithm selection mechanisms. The latter are based on instance-specific features, which are used to train a statistical model subsequently used for selecting the algorithm to be run on a given problem instance, *e.g.*, based on performance predictions for each individual algorithm in the portfolio (Xu et al., 2011). In the former case, all algorithms are run in parallel on a given problem instance, and the portfolio terminates once one algorithm has returned a solution. This implicitly ensures that we always benefit from the best-performing algorithm in the portfolio; however, it comes at the cost of increased use of parallel resources when compared to per-instance selection from a portfolio of algorithms. Thus, when evaluating a parallel algorithm portfolio, it is important to ensure that all algorithms together do not exceed the global computing budget used by a single baseline algorithm.

While parallel portfolios have already been shown to increase the efficiency of MIP-based verification methods (König et al., 2022), they have so far not been exploited more broadly in the context of neural network verification.

3. Verification Algorithms under Assessment

We consider 8 complete neural network verification algorithms in this work; 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 and 2022 VNN competitions or (*ii*) supported by the recently published DNNV framework (Shriver et al., 2021). Table 1 presents an overview of all methods we reviewed and their eligibility for inclusion based on the criteria specified above. Notice that some verification methods, such as Neurify (Wang et al., 2018a) or BaDNB (De Palma et al., 2021), did not participate in the latest edition of the VNN competition. On the other hand, it can be assumed that these methods also contribute to the state of the art in neural network verification. For example, BaDNB, which is part of the OVAL framework, reached third place in the 2021 edition of the competition (Bak et al., 2021) but did not compete in 2022. Altogether, we consider our set of algorithms to be representative of recent and important 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. In other words, one aspect of our study is to capture the situation someone using existing tools "out of the box" might face. We note that the

Table 1: Overview of reviewed verification methods and their eligibility for inclusion in our assessment based on their (*i*) completeness **and** (*ii*) presence in the top five ranking of the 2021 or 2022 VNN Competition **or** (*iii*) support through DNNV. Check marks indicate that a verifier satisfies the criterion, while cross marks indicate that it does not. If a verifier satisfies the inclusion criteria but is superseded by another, more recent method, the former is not included.

Verifier	Complete?	In VNN Comp?	In DNNV?	GPU/GPU?	Reference
BaB	✓	✗	✓	CPU	Bunel et al. (2018)
BaDNB	✓	✓	✗	GPU	De Palma et al. (2021)
β–CROWN	✓	✓	✗	GPU	Wang et al. (2021)
ERAN[1]	✓	✓	✓	GPU	Singh et al. (2019a)
Marabou	✓	✓	✓	CPU	Katz et al. (2019)
MIPVerify[2]	✓	✗	✓	CPU	Tjeng et al. (2019)
MN-BaB	✓	✓	✗	GPU	Ferrari et al. (2022)
Neurify	✓	✗	✓	CPU	Wang et al. (2018a)
nnenum	✓	✓	✓	CPU	Bak et al. (2020)
Planet[3]	✓	✗	✓	CPU	Ehlers (2017)
Reluplex[4]	✓	✗	✓	CPU	Katz et al. (2017)
VeriNet	✓	✗	✓	CPU	Henriksen and Lomuscio (2020)

[1]Superseded by MN-BaB.
[2]Local robustness verification not supported via DNNV.
[3]Superseded by BaB.
[4]Superseded by Marabou.

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., 2022).

3.1 CPU-Based Methods

The CPU-based verification algorithms we considered are the following.

BaB. The algorithm proposed by Bunel et al. (2018) restates the verification problem as a global optimisation problem, which is then solved using branch-and-bound search. It further incorporates algorithmic improvements 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. This method supports ReLU-based networks; for the remainder of this article, we refer to it as BaBSB.

Marabou. The Marabou framework (Katz et al., 2019) 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 activation functions as well as MaxPooling operations.

Neurify. The verification algorithm proposed by Wang et al. (2018a) 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 activation functions.

nnenum. The verifier proposed by Bak et al. (2020) 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 activation functions.

VeriNet. The verifier developed by Henriksen and Lomuscio (2020) 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 activation functions.

3.2 GPU-Based Methods

Next, we present the GPU-based verification algorithms we considered.

BaDNB. The BaDNB verifier introduced by De Palma et al. (2021) builds on earlier versions of the BaB framework; however, it uses a novel dual formulation of the MIP, which it solves via branch-and-bound. The novel formulation allows for extensive parallelisation on GPUs. Furthermore, it employs a bounding heuristic which significantly reduces the number of branches necessary for solving the verification problem. BaDNB is limited to ReLU-based networks and MaxPooling operations.

Beta-CROWN. β-CROWN (Wang et al., 2021) is a bound propagation method combined with neuron-split constraints, which divides the original problem into sub-problems based on the activation function's range. β-CROWN leverages neuron-split constraints, while, in general, other bound propagation methods are not able to handle this type of constraint. Using the framework presented by Bunel et al. (2018), the verifier is complete and can be efficiently parallelised using GPUs. β-CROWN can handle ReLU, Sigmoid and Tanh activations as well as MaxPooling layers.

MN-BaB. The MN-BaB verifier (Ferrari et al., 2022) builds on the multi-neuron constraints underlying the ERAN toolkit (Müller et al., 2021; Singh et al., 2019a,b; Singh and Gehr, 2019; Singh et al., 2018) as well as GPU-enabled linear bound propagation in a branch-and-bound framework. MN-BaB uses different verification modes, including input-domain splitting with bound propagation and full MIP encodings for complete verification. It is capable of handling various activation functions and layer operations such as ReLU, Sigmoid, Tanh, and MaxPooling.

4. 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 that, 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

Table 2: 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. We considered any instance that was found to be *sat* or *unsat* as solved . The number of *sat* and *unsat* instances, respectively, can be found in brackets. The column "Verifiers employed" lists (1) BaBSB, (2) Marabou, (3) Neurify, (4) nnenum, (5) VeriNet, (6) BaDNB, (7) β-CROWN or (8) MN-BaB as the matching suitable algorithm(s) to the respective category.

CPU methods											
Category	MNIST					CIFAR					Verifiers employed
	Total	Solvable				Total	Solvable				
		Any	(*sat*/*unsat*)	All	(*sat*/*unsat*)		Any	(*sat*/*unsat*)	All	(*sat*/*unsat*)	
ReLU	2 500	1 913	(169/1 744)	42	(38/4)	2 500	972	(946/26)	0	(0/0)	(1),(2),(3),(4),(5)
ReLU + MaxPool	400	5	(0/5)	0	(0/0)	100	0	(0/0)	0	(0/0)	(2)
Tanh	600	556	(29/527)	0	(0/0)	600	0	(0/0)	0	(0/0)	(5)
Sigmoid	600	581	(37/544)	0	(0/0)	600	0	(0/0)	0	(0/0)	(2),(5)
GPU methods											
ReLU	2 500	2 308	(128/2 180)	948	(53/895)	2 500	2 364	(2 262/102)	1 048	(1 048/0)	(6),(7),(8)
ReLU + MaxPool	400	128	(40/88)	84	(25/59)	100	64	(64/0)	0	(0/0)	(6),(7),(8)
Tanh	600	319	(28/291)	0	(0/0)	600	497	(494/3)	0	(0/0)	(7),(8)
Sigmoid	600	307	(35/272)	304	(0/0)	600	547	(481/66)	0	(0/0)	(7),(8)

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.

4.1 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 (Hoos and Stützle, 2004; Bartz-Beielstein et al., 2020), 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 instance set should cover 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 our benchmark 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 instance set to be diverse. Therefore, we paid special attention to ensure that the networks we considered 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) and layer operations (*e.g.*, pooling or convolutional layers) they employ. Notice that some of the networks we considered were also used in the 2022 VNN Competition. A full overview of the networks used in our study and their respective sources is provided in Appendix A.

Of 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 the perturbation radius ϵ set at $\{0.004, 0.005, 0.008, 0.01, 0.012, 0.02, 0.025,$

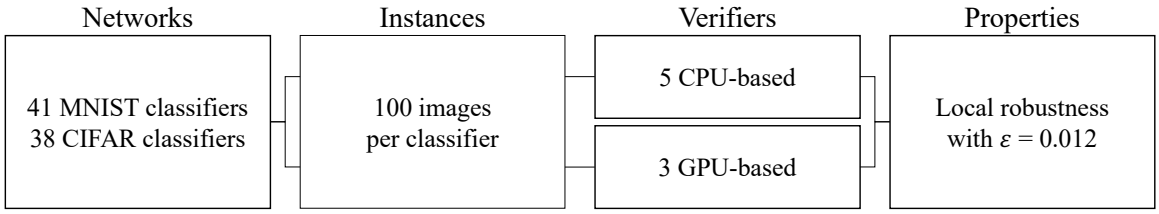


Figure 1: Schematic overview of the experimental setup.

0.03, 0.04}. To avoid over-aggregation, we firstly focused our analysis on a single value of ϵ, where $\epsilon = 0.012$, which represents a radius larger than $1/255$, the smallest ϵ-ball distance used in existing literature (Li et al., 2020), and centred around commonly chosen values for ϵ (Wu et al., 2022; Botoeva et al., 2020; Wang et al., 2018b).

Lastly, we split our benchmark set into different categories based on verifier compatibilities. This means a verifier is only applied to categories it can process. The categories as well as the instance set size for each category are shown in Table 2.

4.2 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 *relative Shapley value* (Fréchette et al., 2016) of each verifier to the parallel portfolio containing all (applicable) verifiers. The first two of these reflect stand-alone performance, while the last two capture performance complementarity between verifiers and their contribution to the overall state of the art. Although these metrics present aggregated measures, they reflect algorithm performance on an instance level and in relation to other methods included in our 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.

The marginal contribution is computed as follows. Define V as a set of verifiers and let $s(V)$ be the total score of set V. Here, the total score $s(V)$ consists of the number of instances verified by at least one verifier in set V within a given cutoff time. 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. Formally, to determine the marginal contribution of any of the verifiers v to portfolio V, one needs to know the value of $s(V)$ and $s(V \setminus \{v\})$, where $V \setminus \{v\}$ is the portfolio minus verifier v. Thus, the marginal contribution of verifier v is expressed as

$$MC_v(V) = s(V) - s(V \setminus \{v\}) \tag{1}$$

Following this terminology, we can define the number of solved instances by verifier v as a set consisting only of verifier v, $Solved_v = s(v) - s(\emptyset)$, where $s(\emptyset) = 0$. In other words, the number of solved instances employs a set of size one whereas the marginal contribution employs a set of all verifiers under consideration. The *relative* marginal contribution

represents the marginal contribution of a given verifier as a fraction 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 complements 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 and in which it is the only one added to the portfolio, whereas the marginal contribution metric assumes a joining order in which it comes last. However, using fixed orders, as 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 algorithm with highly correlated performance. In such a case, both algorithms would be assigned very low marginal contribution, even though one of them should be included in a potential portfolio. Moreover, the fixed joining order leads to the marginal contribution metric being very sensitive to the composition of the portfolio in question; *i.e.*, this metric might change drastically if only a subset of methods would be included in a given portfolio.

This is captured by the Shapley value: Consider a set of verifiers V of size n (*i.e.*, $|V| = n$) and Π^V as the set of all permutations of V. Notice that each permutation π in Π^V is of size n, which results in set Π^V being of size $n!$. Now define V_v^π as the set of verifiers where all verifiers joining after v – *i.e.*, appearing after v in permutation π – are discarded from π. The Shapley value of verifier v, ϕ_v, is then calculated as follows:

$$\phi_v(V) = \frac{1}{n!} \cdot \sum_{\pi \in \Pi^V} \left(s(V_v^\pi) - s(V_v^\pi \setminus \{v\}) \right) \tag{2}$$

The *relative* Shapley value of a verifier v is obtained by dividing ϕ_v by the sum over the (absolute) Shapley values for all verifiers under consideration; it intuitively represents the fraction of the jointly achieved Shapley values over all verifiers that is attributed to verifier v.

4.3 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. Each verification method was limited to using a single CPU core per run. Each query (*i.e.*, attempt to solve a verification problem instance) 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 robustness property into a unified format, which can then be solved by a given method (Shriver et al., 2021). 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 the results in a standardised manner, where the output is either *sat* if the property was falsified or *unsat* if the property was proven to hold. In cases where a violation is found, DNNV also returns a counter-example to the property and validates

it by performing inference with the network. We note that for the VeriNet toolkit, its implementation in DNNV lags behind the standalone implementation of the verifier. While we acknowledge that this could affect observed performance, we still chose to run each CPU method through the DNNV interface to benefit from the broader benchmark support provided by DNNV.

For GPU-accelerated methods, we used machines equipped with NVIDIA GeForce GTX 1080 Ti GPUs with 11 GB video memory. We provided the same time budget but did not impose any memory constraints. The GPU-based methods we considered are not supported by DNNV. Hence, we used the standalone implementations of these algorithms through the β-CROWN[2], OVAL$-$BaB[3], and MN-BaB[4] framework, respectively. These methods also return a counter-example to the property in cases where a violation is found.

5. Results and Discussion

In the following, we provide an in-depth discussion of the results obtained from our experiments. We distinguish between CPU-based algorithms and algorithms that also utilise GPU resources. Table 2 shows the categories we devised based on layer types present in the network, along with the resulting instance set sizes as well as information on which verifier has been employed for each category. Moreover, we investigate whether there exists a single algorithm that performs best on all instances within a given category. If we find this to not be the case, we analyse to what extent the algorithms we considered complement each other in performance, *i.e.*, show strong performance on different problem instances.

5.1 CPU-Based Methods

Table 3 contains the results from our experiments using CPU-based verification algorithms. It reports the number of problem instances solved by each verifier per network category (see Table 2 for the total number of problem instances per category), the relative marginal contribution, the relative Shapley value and the average running time computed over the subset of *solvable* instances, *i.e.*, instances that could be solved by at least one of the considered methods. The relative marginal contribution and the relative Shapley value are calculated based on the number of solved problem instances. We provide absolute values for both the marginal contribution and Shapley value in Appendix B. Notice that instances that were not solved within the time limit were attributed the maximum running time, *i.e.*, 3 600 seconds.

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 relative Shapley value of 0.32. 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 other method.

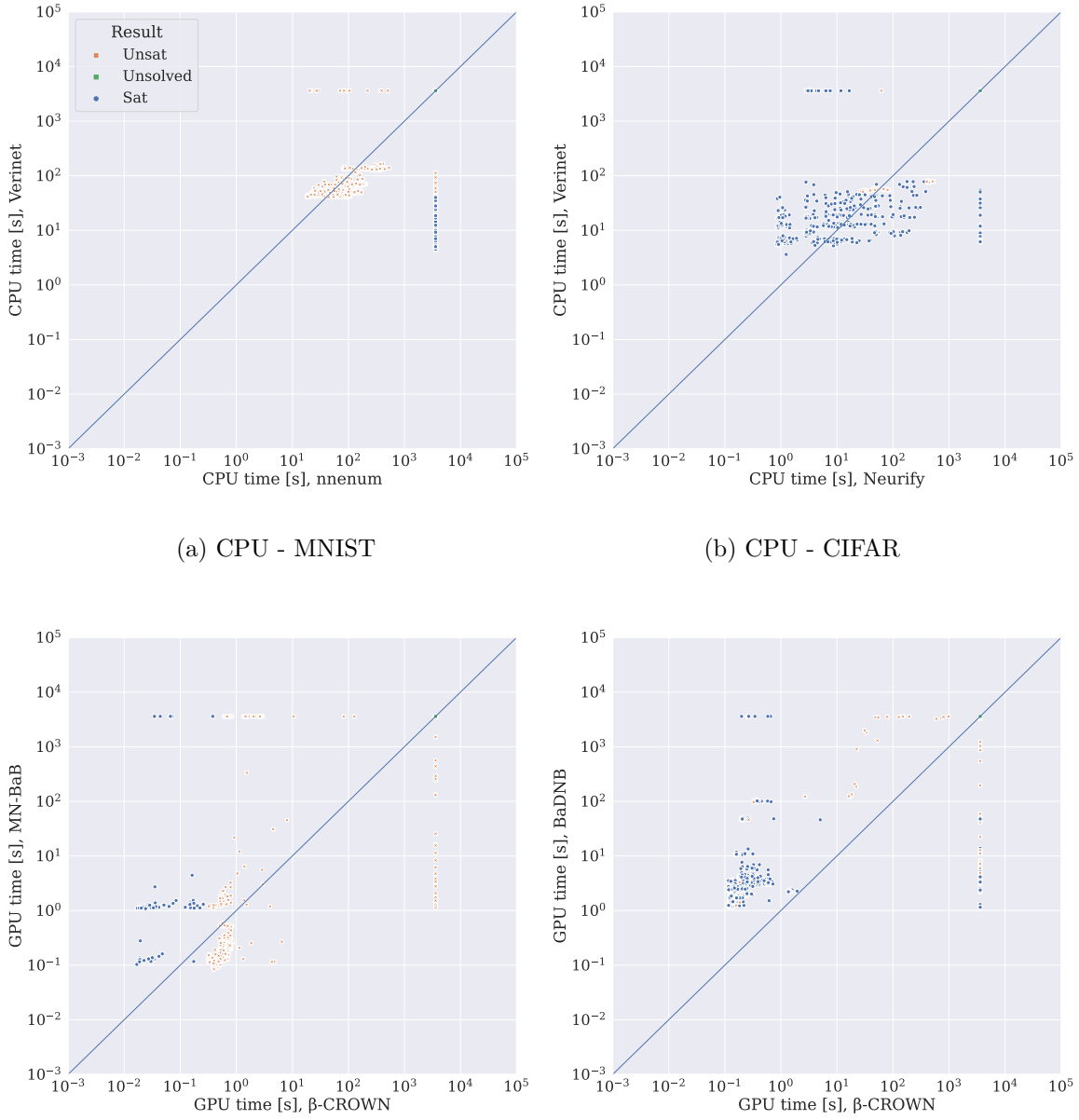
2. Commit 7a46097192207dfbb2fa7135857d6bc4ae7d6cd5
3. Commit 9e1606044759da5693f226ce489e9d4dded21bd6
4. Commit 2aa12b145bb61342f4c464b64be3467b3a275e46

Table 3: Performance comparison of CPU-based verification algorithms in terms of the number of solved instances, relative marginal contribution (RMC), relative Shapley value (ϕ) and CPU running time averaged per problem instance, computed for each category for $\epsilon = 0.012$.

ReLU								
Verifier	MNIST				CIFAR			
	Solved	RMC	ϕ	Avg. Time [CPU s]	Solved	RMC	ϕ	Avg. Time [CPU s]
BaBSB	358	0.22	0.06	3 241	307	0.00	0.09	2 924
Marabou	1 001	0.19	0.16	1 801	400	0.00	0.12	2 153
Neurify	871	0.25	0.14	1 964	915	0.75	0.42	235
nnenum	1 754	0.17	0.31	389	76	0.05	0.03	3 337
VeriNet	1 799	0.16	0.32	263	841	0.20	0.34	500
ReLU+MaxPool								
Verifier	MNIST				CIFAR			
	Solved	RMC	ϕ	Avg. Time	Solved	RMC	ϕ	Avg. Time
Marabou	5	1.00	1.00	57	0	0.00	0.00	3 600
Tanh								
Verifier	MNIST				CIFAR			
	Solved	RMC	ϕ	Avg. Time	Solved	RMC	ϕ	Avg. Time
VeriNet	556	1.00	1.00	55	0	0.00	0.00	3 600
Sigmoid								
Verifier	MNIST				CIFAR			
	Solved	RMC	ϕ	Avg. Time	Solved	RMC	ϕ	Avg. Time
Marabou	0	0.00	0.00	3 600	0	0.00	0.00	3 600
VeriNet	581	1.00	1.00	55	0	0.00	0.00	3 600

On ReLU-based CIFAR networks, it should first be noted that there is no verification problem instance that can be solved by *all* verifiers, highlighting the structural differences between instances and the sensitivity of the verification approaches to those differences. 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.

Figure 2a and 2b show an instance-level comparison of the two best-performing algorithms (in terms of relative Shapley value) in the ReLU category for each dataset. In Figure 2a, we see that on MNIST networks, both VeriNet and nnenum solved instances that the other one, in turn, could not solve within the given time budget. Concretely, when considering a parallel portfolio containing both algorithms (see Section 2.3), the number of solved instances slightly



(a) CPU - MNIST

(b) CPU - CIFAR

(c) GPU - MNIST

(d) GPU - CIFAR

Figure 2: Performance comparison of the two top-performing verification methods (in terms of relative Shapley value) in the ReLU category for CPU-based methods on (a) MNIST and (b) CIFAR networks as well GPU-based methods on (c) MNIST and (d) CIFAR networks. Each data point represents an instance, and its position on a given axis represents the performance in terms of running time of the respective solver. The diagonal line represents the point on which both verifiers perform equally well. The verifier represented on the x-axis performs better on instances above the diagonal line, and the verifier represented on the y-axis performs better on instances below the diagonal line. Instances that were not solved within the time limit are displayed with the maximum running time (*i.e.*, 3 600 seconds).

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, adding up to the same combined maximum running time as running a single verifier with 3 600 CPU seconds). We note that leveraging parallel portfolios has already been shown to significantly improve the performance of MIP-based verification methods (König et al., 2022).

On CIFAR instances, we found Neurify and VeriNet to also have distinct strengths over each other. This is shown in Figure 2b, 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 much lower relative Shapley values and relative marginal contribution scores, indicating that they would not substantially strengthen the performance of a portfolio already containing Neurify and VeriNet.

Figure 3a 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.

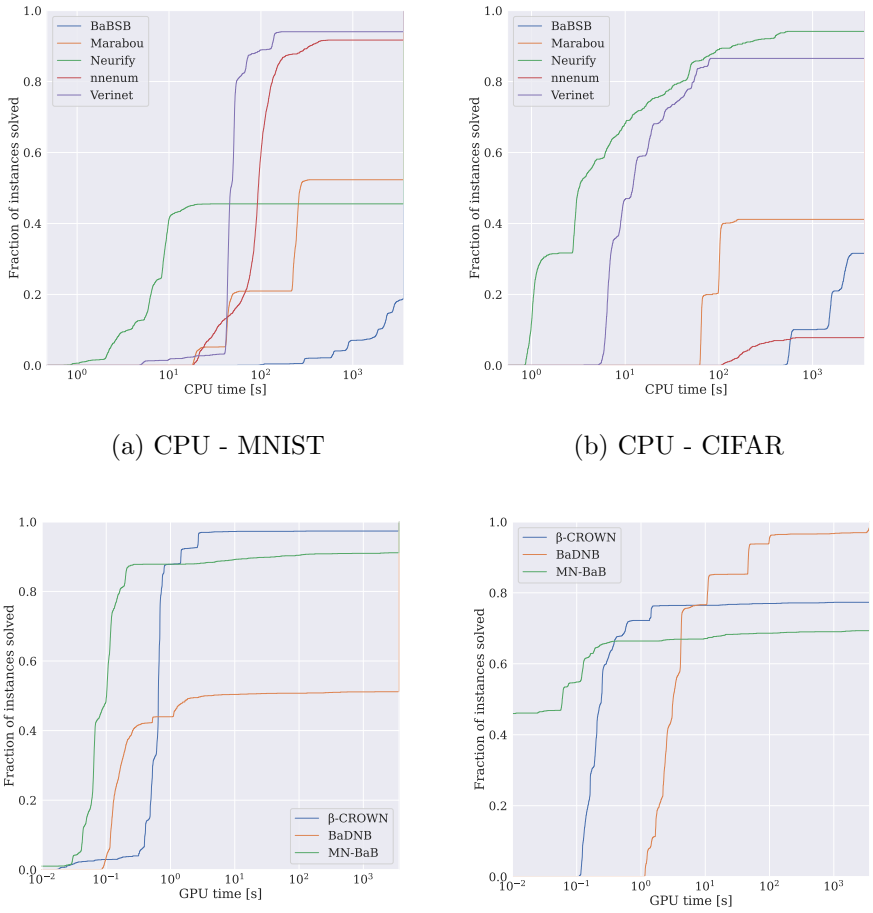
Figure 3b shows a similar plot for the CIFAR problem instances. Here, Neurify solved the largest fraction in less time than other methods. This suggests that Neurify is a very competitive verifier when applicable to the specific network or input format.

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 activation functions can, in principle, be verified by VeriNet but the algorithm did nonetheless 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 time and memory budget.

5.2 GPU-Based Methods

Table 4 summarises the results from our experiments using GPU-based verification algorithms. On ReLU-based MNIST networks, β-CROWN outperformed other methods in terms of both the number of solved problem instances as well as the average running time. At the same time, the relative Shapley values of β-CROWN and MN-BaB indicate that these methods complement each other with respect to their performance on this instance set.

On ReLU-based CIFAR networks, Table 4 shows that BaDNB outperformed both MN-BaB and β-CROWN, with the former solving 2 332 and the latter solving 1 639 and 1 828 out of 2 500 verification problem instances, respectively. Furthermore, both BaDNB and β-CROWN achieve large relative Shapley values, suggesting their complementarity in an algorithm portfolio.



(a) CPU - MNIST

(b) CPU - CIFAR

(c) GPU - MNIST

(d) GPU - CIFAR

Figure 3: Cumulative distribution of the fraction of instances solved by the considered verification algorithms in the ReLU category as a function of CPU running time. The plots at the top are for CPU-based algorithms, whereas those at the bottom are for GPU-based algorithms, on MNIST and CIFAR.

Figure 2c and 2d show the instance-level comparison of the two best-performing algorithms (in terms of relative Shapley value) in the ReLU category for each dataset. Looking at Figure 2c, one can see that there is a fairly large number of MNIST instances unsolved by β-CROWN but solved by MN-BaB as well as the other way around.

On the other hand, BaDNB and β-CROWN seem to have distinctive strengths over each other on CIFAR instances, as can be seen in Figure 2d: The data points indicating performance on each verification instance are spread out widely around the line of equal performance, showing that there are many instances that one method can solve faster than the other and vice versa.

Table 4: Performance comparison of GPU-based verification algorithms in terms of the number of solved instances, relative marginal contribution (RMC), relative Shapley value (ϕ) and average GPU running time, computed for each category for $\epsilon = 0.012$.

ReLU								
Verifier	MNIST				CIFAR			
	Solved	RMC	ϕ	Avg. Time [GPU s]	Solved	RMC	ϕ	Avg. Time [GPU s]
BaDNB	1 188	0.31	0.19	1 760	2 332	0.90	0.45	116
β-CROWN	2 247	0.00	0.42	96	1 828	0.03	0.29	814
MN-BaB	2 103	0.69	0.39	325	1 639	0.07	0.26	1 110
ReLU+MaxPool								
Verifier	MNIST				CIFAR			
	Solved	RMC	ϕ	Avg. Time	Solved	RMC	ϕ	Avg. Time
BaDNB	85	0.00	0.22	1 399	0	0.00	0.00	3 600
β-CROWN	128	1.00	0.44	0.4	0	0.00	0.00	3 600
MN-BaB	115	0.00	0.34	366	64	1.00	1.00	0.008
Tanh								
Verifier	MNIST				CIFAR			
	Solved	RMC	ϕ	Avg. Time	Solved	RMC	ϕ	Avg. Time
β-CROWN	319	1.00	1.00	1.16	497	1.00	1.00	0.70
MN-BaB	0	0.00	0.00	3 600	0	0.00	0.00	3 600
Sigmoid								
Verifier	MNIST				CIFAR			
	Solved	RMC	ϕ	Avg. Time	Solved	RMC	ϕ	Avg. Time
β-CROWN	306	0.66	0.50	13	538	0.95	0.68	60
MN-BaB	305	0.33	0.50	24	338	0.05	0.32	1 376

Concurrently, MN-BaB solves a large fraction of CIFAR instances in less time than other methods, although BaDNB solves more instances overall, which is also reflected in Figure 3d. On MNIST instances, MN-BaB solves more instances in less time than β-CROWN, although β-CROWN solves more instances overall; see also Figure 3c.

On MNIST networks containing ReLU activation functions and MaxPooling operations, we again found relatively large Shapley values for both β-CROWN and MN-BaB, as presented in Table 4, indicating their potential complementarity in an algorithm portfolio. However, the relative marginal contribution values indicate that there are no instances unsolved by β-CROWN that could be solved by other methods. CIFAR instances in this category could only be verified by MN-BaB, due to verifier incompatibilities with the respective network structures unrelated to the MaxPooling operations.

Table 4 further shows results for the Tanh category. We found that instances in this category could effectively only be handled the β-CROWN verifier. Concretely, MN-BaB

returned an error for the instances in this category; see also Appendix C for additional details.

Lastly, networks containing Sigmoid activation functions can be handled by both BaDNB and β-CROWN and achieve perfectly similar relative Shapley values on the MNIST instances in this category, indicating their complementarity in an algorithm portfolio. However, as seen in Table 4, this does not hold for CIFAR instances, where β-CROWN seems to dominate in performance.

5.3 Error Analysis

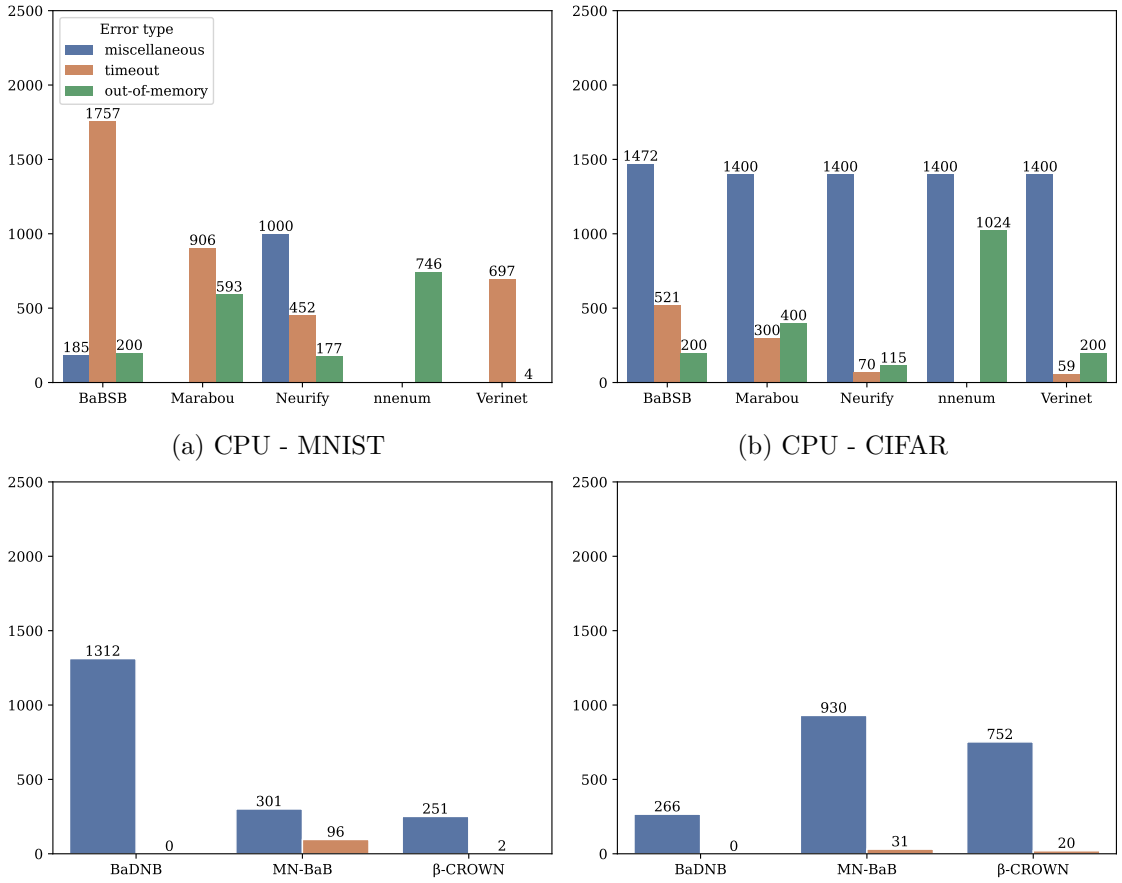
Although the verification methods should, in principle, be able to solve the instances in the category they are applied to, we found many instances left unsolved, not only due to time or memory constraints but also due to other, unexpected issues. Hence, to understand better why certain instances could not be solved by a given verifier, we categorised and counted the errors returned by each verification system. For this analysis, we focused on instances in the ReLU category; results for the remaining categories are presented in Appendix C. The number of instances solved by each method can be found in Table 3 for CPU- and Table 4 for GPU-based algorithms. The total number of instances in the ReLU category is 2 500 for MNIST and CIFAR, respectively. We distinguish between timeouts, out-of-memory and *miscellaneous* errors, where the latter includes verifier-specific errors of which most are undefined and not trivial to resolve, especially without in-depth knowledge of the verifier at hand.

Figure 4a and 4b show the errors returned by CPU-based methods for MNIST and CIFAR instances, respectively. On MNIST, most verifiers failed to solve a given instance due to timeouts, except for nnenum, which mostly ran into memory issues, and Neurify, which requires images used as network inputs to have 2 or 4 dimensions, as mentioned in Section 5.1. Notice that when supplied with a larger memory budget, nnenum could not solve substantially more instances, but produced a comparably large number of timeouts instead; more details can be found in Appendix D.1.

Interestingly, we made different observations with regard to the CIFAR instances. Here, each method mostly returned errors related to the network structure (or undefined errors). Besides this, nnenum again failed to verify a sizable fraction of instances due to memory limitations. Overall, we found CIFAR networks to be much less supported by the CPU-based methods we considered (as implemented in the DNNV framework) than MNIST networks, arguably due to the increased complexity of the former. We note that some of these errors could potentially be circumvented by resorting to the standalone implementations of the respective verifiers. However, overall, DNNV provides the broadest support for different network structures and operations (Shriver et al., 2021).

On the other hand, GPU-based verifiers show greater support for the considered networks than CPU-based methods. As seen in Figure 4c, only BaDNB failed to solve a relatively large number of MNIST instances due to unsupported network structures or other, unspecified technical reasons.

In contrast, BaDNB could solve almost all CIFAR instances, as shown in Figure 4d. However, both MN-BaB and β-CROWN returned several errors of which most are undefined.



(a) CPU - MNIST

(b) CPU - CIFAR

(c) GPU - MNIST

(d) GPU - CIFAR

Figure 4: Frequency of error types returned by the considered verification algorithms on instances in the ReLU category.

Overall, our results suggest that many verification toolkits only support a limited set of networks. This occurs despite the fact that these networks are provided in onnx format, which should, in principle, be supported by each method considered in this study. Similar findings have been reported in the literature (see, *e.g.*, Meng et al., 2022).

5.4 Analysis on Broader Set of Perturbation Radii

So far, we have considered a single value of ϵ, but it stands to reason that changing the perturbation radius may affect algorithm behaviour. Therefore, we conducted further analysis on a broader set of perturbation radii, *i.e.*, with ϵ set to values of 0.004, 0.005, 0.008, 0.01, 0.012, 0.02, 0.025, 0.03 and 0.04.

Table 5 shows the results for the CPU-based algorithms on this extended set of problem instances. Overall, we found VeriNet remains the best-performing CPU-based verifier (in terms of solved instances and relative Shapley value) on ReLU-based MNIST networks.

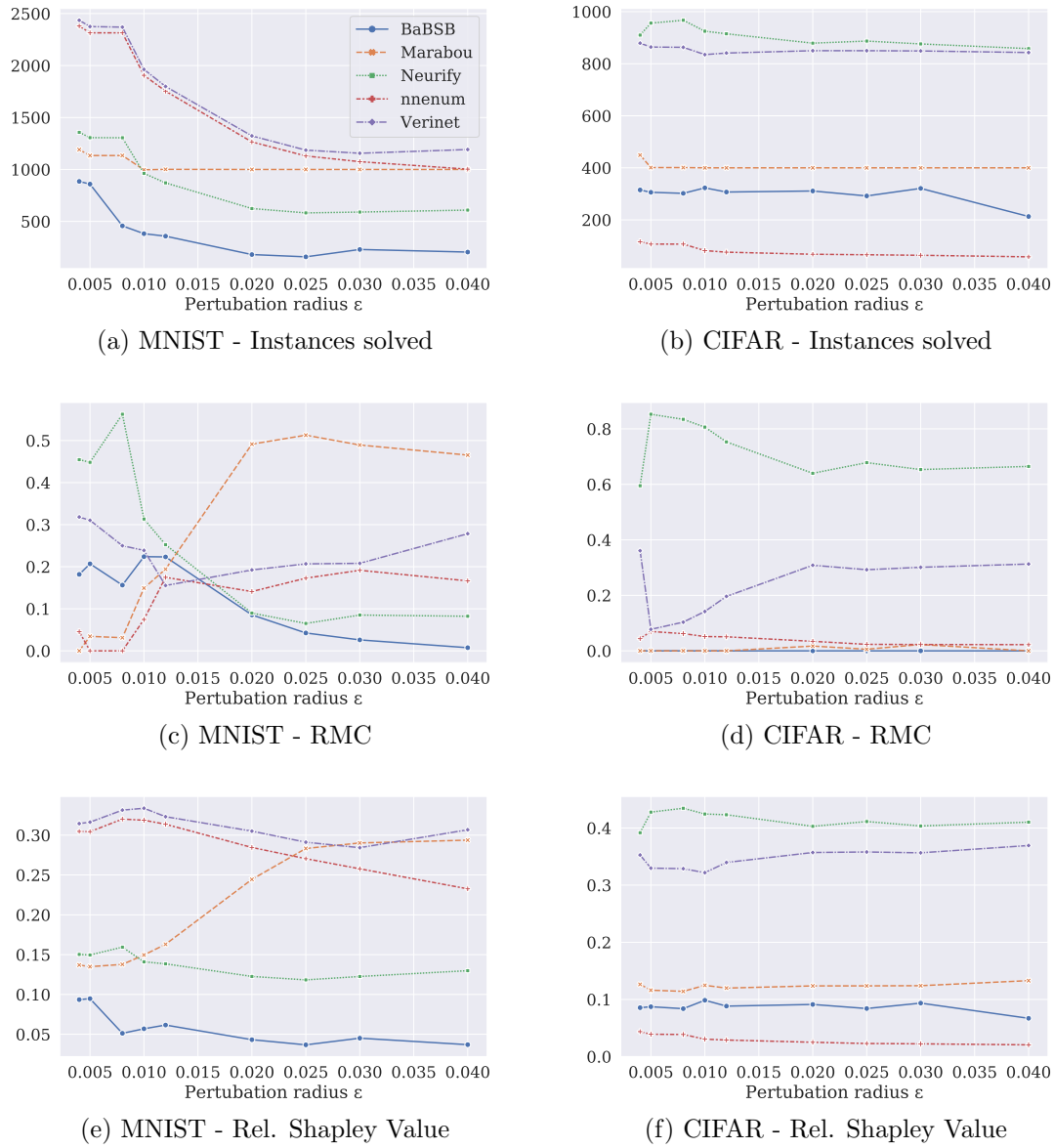
Table 5: Performance comparison of CPU-based verification algorithms in terms of the number of solved instances, relative marginal contribution (*RMC*), relative Shapley value (ϕ) and average CPU running time, computed for each category and $\epsilon \in \{0.004, 0.005, 0.008, 0.01, 0.012, 0.02, 0.025, 0.03, 0.04\}$.

ReLU								
Verifier	MNIST				CIFAR			
	Solved	*RMC*	ϕ	Avg. Time [CPU s]	Solved	*RMC*	ϕ	Avg. Time [CPU s]
BaBSB	3 716	0.06	0.06	3 223	2 690	0.00	0.09	2 964
Marabou	9 457	0.44	0.19	1 721	3 651	0.01	0.12	2 145
Neurify	8 206	0.12	0.14	1 899	8 173	0.71	0.41	289
nnenum	15 144	0.16	0.30	543	744	0.04	0.03	3 315
VeriNet	15 800	0.23	0.32	367	7 674	0.24	0.35	486
ReLU+MaxPool								
Verifier	MNIST				CIFAR			
	Solved	*RMC*	ϕ	Avg. Time	Solved	*RMC*	ϕ	Avg. Time
Marabou	316	1.00	1.00	50	0	0.00	0.00	3 600
Tanh								
Verifier	MNIST				CIFAR			
	Solved	*RMC*	ϕ	Avg. Time	Solved	*RMC*	ϕ	Avg. Time
VeriNet	4 307	1.00	1.00	59	0	0.00	0.00	3 600
Sigmoid								
Verifier	MNIST				CIFAR			
	Solved	*RMC*	ϕ	Avg. Time	Solved	*RMC*	ϕ	Avg. Time
Marabou	0	0.00	0.00	3 600	0	0.00	0.00	3 600
VeriNet	4 728	1.00	1.00	59	0	0.00	0.00	3 600

With regard to ReLU-based CIFAR networks, Table 5 shows that, overall, Neurify remained the best-performing CPU-based method.

However, we observed substantial differences between small and large values of ϵ in the relative marginal contribution for each algorithm. More precisely, we analysed the relative marginal contribution of each verification algorithm for every given value of ϵ and show this in Figure 5c. Interestingly, one can see how the relative marginal contribution of Marabou steeply increases for increasingly larger epsilons, while that of other methods declines. Similarly, the solved instances and relative Shapley value achieved by each method changes as the perturbation radius varies; this is visualised in Figure 5a and 5e. In terms of both metrics, Marabou is strongly outperformed by most of the other algorithms for small values of ϵ but ends up achieving competitive or even better performance when ϵ is large.

An analogous investigation for CIFAR is shown in Figure 5d. In contrast to MNIST, one can see that the relative marginal contribution of each method is relatively weakly affected by the perturbation radius and, except for some divergence around $\epsilon = 0.005$, remains at a



(a) MNIST - Instances solved

(b) CIFAR - Instances solved

(c) MNIST - RMC

(d) CIFAR - RMC

(e) MNIST - Rel. Shapley Value

(f) CIFAR - Rel. Shapley Value

Figure 5: Performance of CPU-based verifiers for different values of ϵ in the ReLU category.

stable level. This holds for both solved instances and relative Shapley value as shown in Figure 5b and 5f.

We performed a similar analysis for GPU-based methods and present results, aggregated over all values of ϵ, in Table 6. Among these algorithms, β-CROWN performed best on MNIST networks in the ReLU category, while BaDNB performed best on CIFAR networks in the same category. However, we found that the relative marginal contribution of each algorithm for every considered value of ϵ differs substantially between small and large values of ϵ on MNIST instances, as shown in Figure 6c. For example, when $\epsilon = 0.02$, BaDNB and

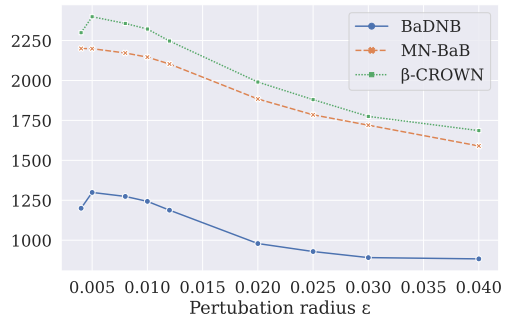
Table 6: Performance comparison of GPU-based verification algorithms in terms of the number of solved instances, relative marginal contribution (*RMC*), relative Shapley value (ϕ) and average GPU running time, computed for each category and aggregated $\epsilon \in \{0.004, 0.005, 0.008, 0.01, 0.012, 0.02, 0.025, 0.03, 0.04\}$.

ReLU								
Verifier	MNIST				CIFAR			
	Solved	*RMC*	ϕ	Avg. Time [GPU s]	Solved	*RMC*	ϕ	Avg. Time [GPU s]
BaDNB	9 886	0.71	0.19	1 864	21 438	0.90	0.45	100
β-CROWN	18 955	0.02	0.42	148	17 014	0.03	0.30	783
MN-BaB	17 799	0.27	0.39	363	14 675	0.07	0.25	1 174
ReLU+MaxPool								
Verifier	MNIST				CIFAR			
	Solved	*RMC*	ϕ	Avg. Time	Solved	*RMC*	ϕ	Avg. Time
BaDNB	720	0.03	0.22	1 493	0	0.00	0.00	3 600
β-CROWN	1 127	0.96	0.46	19	0	0.00	0.00	3 600
MN-BaB	966	0.01	0.32	366	576	1.00	1.00	0.008
Tanh								
Verifier	MNIST				CIFAR			
	Solved	*RMC*	ϕ	Avg. Time	Solved	*RMC*	ϕ	Avg. Time
β-CROWN	2 576	1.00	1.00	1.16	4 535	1.00	1.00	0.75
MN-BaB	0	0.00	0.00	3 600	0	0.00	0.00	3 600
Sigmoid								
Verifier	MNIST				CIFAR			
	Solved	*RMC*	ϕ	Avg. Time	Solved	*RMC*	ϕ	Avg. Time
β-CROWN	2 617	0.66	0.50	23	4 961	0.97	0.69	46
MN-BaB	2 601	0.33	0.50	44	3 042	0.03	0.31	1 420

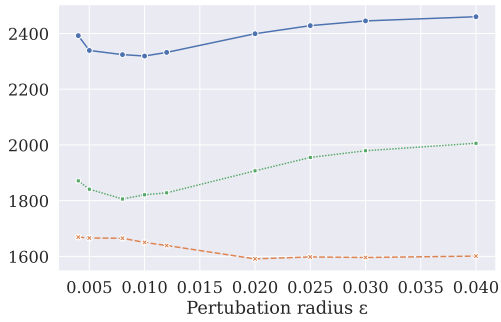
MN-BaB both achieve relative marginal contribution scores close to 0.5 but then strongly converge as ϵ becomes larger. Notably, these changes are not reflected in the relative Shapley values achieved by each method, where β-CROWN and MN-BaB both reach values close to 0.40 for every value of ϵ; see Figure 6e for more details.

On CIFAR instances, Figure 6d indicates that the relative marginal contribution scores are only marginally affected by the chosen perturbation radius. More precisely, BaDNB achieves the largest relative marginal contribution for every value of ϵ, while the relative marginal contributions of β-CROWN and MN-BaB only change slightly as the perturbation radius increases. At the same time, the observed Shapley values are mostly stable with regard to the perturbation radius, as shown in Figure 6f.

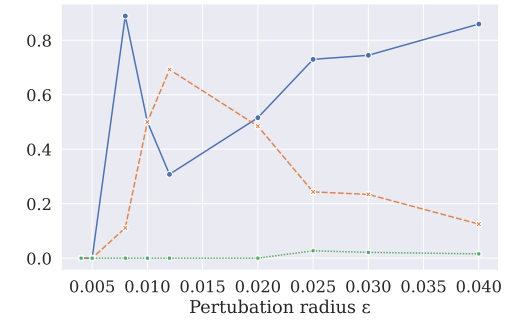
Lastly, we again compared the performance of the two best-performing CPU as well as GPU methods on an instance-level for all MNIST and CIFAR networks, respectively, from the ReLU category and show the results in Figure 7. In each case, we found that one method



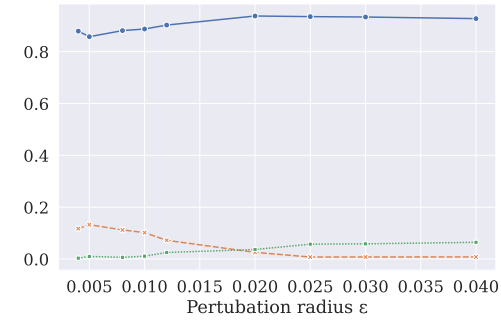
(a) MNIST - Instances solved



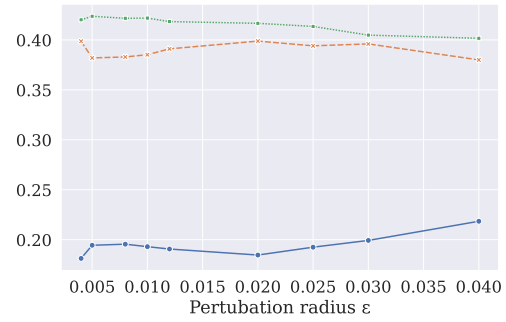
(b) CIFAR - Instances solved



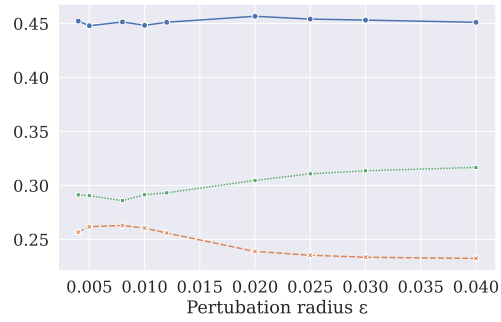
(c) MNIST - RMC



(d) CIFAR - RMC



(e) MNIST - Rel. Shapley value

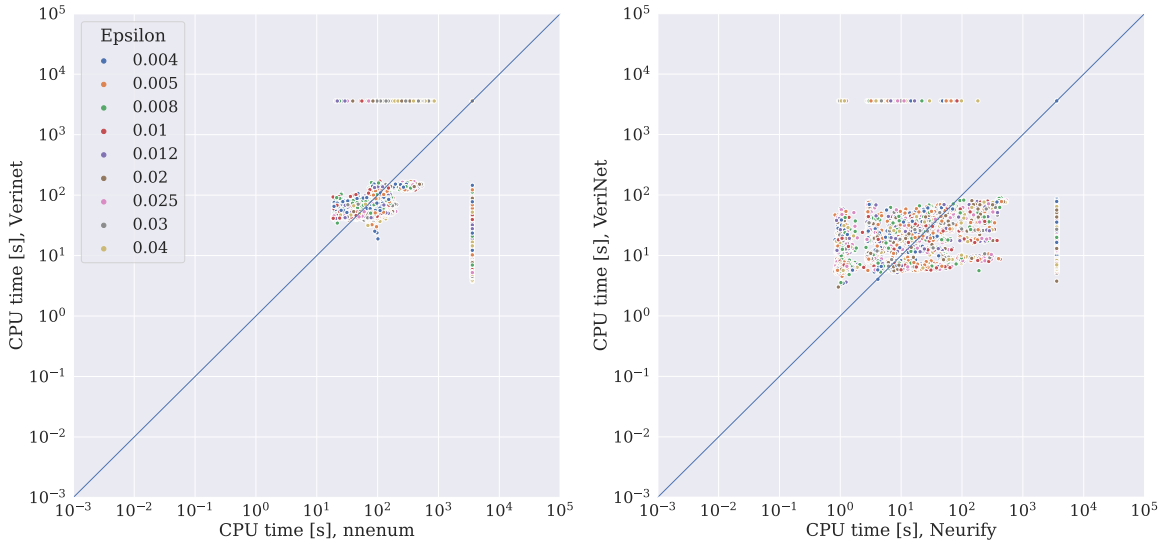


(f) CIFAR - Rel. Shapley value

Figure 6: Performance of GPU-based verifiers for different values of ϵ in the ReLU category.

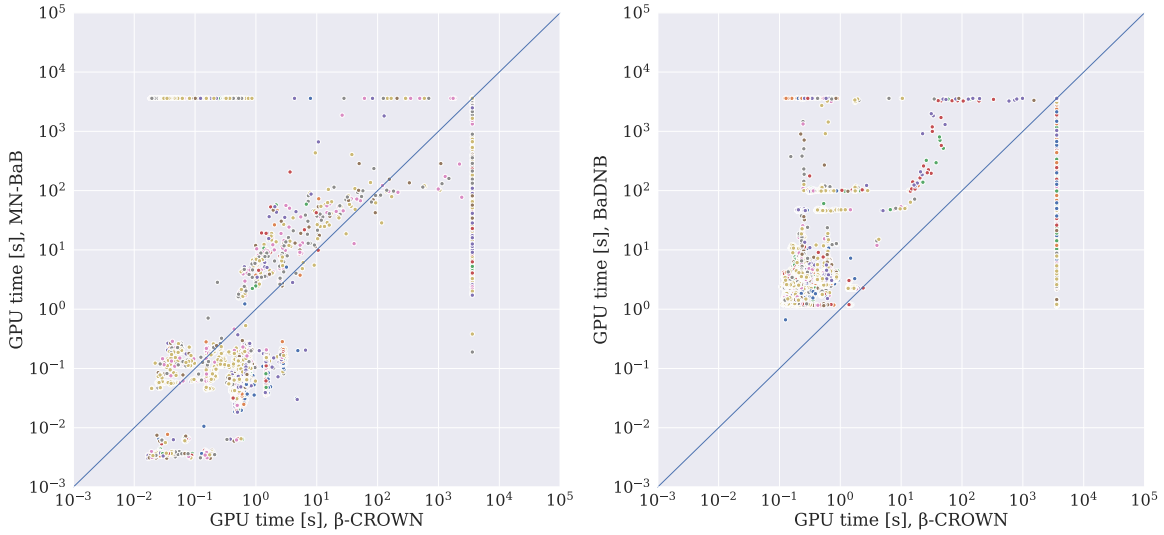
could solve some instances that were unsolved by the other, irrespective of the perturbation radius. Notice that our findings hold even for a much larger value of ϵ. Specifically, we ran the two best-performing CPU-based algorithms, nnenum and VeriNet, on the MNIST instances for $\epsilon = 0.2$ and present the results in Appendix D.2.

Overall, this clearly demonstrates that our observation of performance complementarity between verification algorithms holds for a broad range of perturbation radii.



(a) CPU - MNIST

(b) CPU - CIFAR



(c) GPU - MNIST

(d) GPU - CIFAR

Figure 7: Performance comparison of the two top-performing verification methods (in terms of relative Shapley value) in the ReLU category for CPU-based methods on (a) MNIST and (b) CIFAR networks as well GPU-based methods on (c) MNIST and (d) CIFAR networks, using multiple values of the perturbation radius ϵ.

5.5 Joint Analysis of CPU- and GPU-Based Methods

As previously explained, directly comparing CPU- and GPU-based algorithms is a challenging endeavour, due to the different parallelisation schemes as well as the costs associated with running these algorithms. Here, we seek to capture both of these aspects by conducting a cost-calibrated analysis. More concretely, we compared these methods whilst factoring in the price of operating them on a prominent cloud computing platform. To this end, we

investigated the price difference between Amazon EC2 CPU instances comparable to the resources allocated in this study.[5] Notice that this hardware is not the exact hardware used in our experiments but is being used here as a substitute for calculating the cost of running similar hardware. Based on this cost difference, we reduced the time budget for GPU-based methods by a factor of 46.9, thereby ensuring that these methods cannot exceed the cost budget given to the CPU-based algorithms. While we carefully calibrated this factor based on existing prices, it must be noted that this analysis is based on many assumptions, and therefore, the comparison between CPU and GPU-based solvers serves only illustrative purposes.

Results from this analysis can be found in Table 7 for $\epsilon = 0.012$ and Table 8 for the full range of values of ϵ we considered. First and foremost, it can be seen that despite the higher costs associated with GPU resources, GPU-based verification tools (in particular β-CROWN, MN-BaB) are in many scenarios the most cost-efficient verifiers. However, the results also show that there exist scenarios in which CPU-based methods complement GPU-based methods in their performance. More concretely, Table 8 shows that the CPU-based verifier Marabou achieved the largest relative marginal contribution among all methods on MNIST networks from the ReLU category, indicating that it could solve a sizeable number of instances, which none of the other CPU- or GPU-based methods were able to solve within the same budget. In addition, the CPU-based verifier VeriNet achieved competitive marginal contribution and Shapley values. Furthermore, in the Tanh category, VeriNet was able to solve a large fraction of instances for which β-CROWN failed to return a solution; this observation holds when analysing both a single value of ϵ as well as the whole set of considered perturbation radii.

5.6 Analysis of *unsat* instances

To gain further insights, we performed an analysis of *unsat* (*i.e.*, robust) instances; see Table 2 for the number of *unsat* instances that were found in each network category. More concretely, we considered only *unsat* instances as solved, since several verification methods considered in this study use counter-example generation mostly as an early stopping opportunity. Thus, *unsat* instances pose an interesting subset of the benchmark, as it measures the ability of a method to determine robustness in cases where no such counter-example exist. Furthermore, commonly used robustness metrics, such as adversarial accuracy, are computed by means of the fraction of *unsat* instances in a given instance set. Therefore, verification methods that can efficiently solve those instances enable a more accurate calculation of these metrics.

Table E.1 shows result from this analysis for $\epsilon = 0.012$ while Table E.2 shows results aggregated over the full range of ϵ values we considered. First of all, we found that the total number of solved instances decreases when only *unsat* instances are considered. This is particularly noticeable for CIFAR, where the majority of instances are non-robust or, in other words, *sat*. Furthermore, we observed only minor changes in the relative performance and complementarity of the given verifiers on MNIST instances across all categories. Specifically, we found that for the broader set of ϵ values, the *RMC* and Shapley value of Marabou

5. We selected the t2.medium and the g4dn.8xlarge instances, which cost \$0.0464 and \$2.176 per hour, respectively, see `https://aws.amazon.com/ec2/pricing/on-demand/`. Notice that there also exists the even cheaper t2.small instance with only a single CPU core; however, we did not select this machine as it has only 2 GB RAM.

improve substantially, while those for VeriNet strongly deteriorate. This indicates that on *unsat* instances, Marabou can solve a large fraction of instances unsolved by other methods, while VeriNet mainly contributes when *sat* instances are also considered. For CIFAR, we also noticed that the relative performance of the given verifiers changed. Specifically, MN-BaB, which previously achieved competitive relative performance does not seem to complement other methods on *unsat* instances; instead, most instances are solved by BaDNB and β-CROWN, which also show strong complementarity in the ReLU category.

5.7 Analysis of the 2022 VNN Competition Results

To see if and to what extent our observations hold for a larger set of verifiers as well as different benchmarks, we analysed the results of the 2022 edition of the VNN competition. We refer to the accompanying report (Müller et al., 2023) for more information about the participating tools, benchmarks and further technical details. Again, we present a joint as well as a separate analysis of CPU- and GPU-based verification algorithms. We excluded CGDTest from the set of methods considered in our analysis, as it represents the only incomplete verification approach participating in the competition, while our work focuses on complete verification. In addition, CGDTest produced a substantial number of incorrect results in the competition, casting doubts on the soundness of the method.

Table F.1 in the appendix shows the results from the VNN competition for CPU-based verification algorithms. It reports the number of problem instances solved by each verifier per network category, marginal contribution as well as Shapley values, both in absolute and relative terms. Most notably, we observe strong complementarity between the verifiers considered in two of the three benchmark categories. Concretely, in the CNN + ResNet category, Marabou and VeraPak achieved relative Shapley values of 0.44 and 0.24, respectively. Indeed, as depicted in Figure F.1c, there are several instances solved by one of the verifiers but unsolved by the other.

In the FC category, Marabou, nnenum and PerigiNN achieved a similar relative Shapley value of 0.24, again highlighting the complementarity between these algorithms. Given the similar relative Shapley values, we resort to the relative marginal contribution to determine the two best-performing methods in this context; *i.e.*, among these three methods, nnenum and PerigiNN achieved the largest relative marginal contributions and are, thus, considered the two best-performing methods. Again, we compare their performance on an instance level, as shown in Figure F.1e. As can be observed, instances spread out widely around the equal performance line of the plot, with many instances solved by nnenum but unsolved by PerigiNN, and vice versa.

In the Complex category, nnenum and PeregriNN achieved Shapley values of 0.46 and 0.50, respectively. However, Figure F.1a reveals that nnenum dominates in performance over PeregriNN on most instances. We note that the Shapley value represents the average contribution made by a given verifier over all possible sets of algorithms in a portfolio. Hence, it indicates that nnenum could solve many instances unsolved by other methods from the full set of algorithms under consideration; however, nnenum does not complement PeregriNN in terms of solved instances.

Next, we discuss the results from the 2022 VNN Competition for GPU-based verification algorithms; these are presented in Table F.2 in the appendix. Surprisingly, for GPU-

Table 7: Performance comparison of CPU- and GPU-based verification algorithms in terms of the number of solved instances, relative marginal contribution (RMC), relative Shapley value (ϕ), computed for each category and $\epsilon = 0.012$.

ReLU						
Verifier	MNIST			CIFAR		
	Solved	RMC	ϕ	Solved	RMC	ϕ
BaDNB	1 171	0.12	0.25	2 217	0.46	0.43
BaBSB	358	0.00	0.00	307	0.00	0.00
β-CROWN	2 245	0.00	0.23	1 819	0.27	0.34
Marabou	1 001	0.06	0.03	400	0.00	0.00
MN-BaB	2 083	0.71	0.38	1 622	0.28	0.19
Neurify	871	0.06	0.03	915	0.00	0.03
nnenum	1 754	0.00	0.03	76	0.00	0.00
VeriNet	1 799	0.06	0.03	841	0.00	0.01
ReLU+MaxPool						
Verifier	MNIST			CIFAR		
	Solved	RMC	ϕ	Solved	RMC	ϕ
BaDNB	69	0.00	0.05	0	0.00	0.00
β-CROWN	128	1.00	0.67	0	0.00	0.00
Marabou	5	0.00	0.00	0	0.00	0.00
MN-BaB	115	0.00	0.27	64	1.00	1.00
Tanh						
Verifier	MNIST			CIFAR		
	Solved	RMC	ϕ	Solved	RMC	ϕ
β-CROWN	319	0.09	0.19	198	1.00	1.00
MN-BaB	0	0.00	0.00	0	0.00	0.00
VeriNet	556	0.91	0.81	0	0.00	0.00
Sigmoid						
Verifier	MNIST			CIFAR		
	Solved	RMC	ϕ	Solved	RMC	ϕ
β-CROWN	306	0.00	0.03	538	0.96	0.82
Marabou	0	0.00	0.00	0	0.00	0.00
MN-BaB	305	0.00	0.03	338	0.04	0.18
VeriNet	581	1.00	0.93	0	0.00	0.00

based methods, our findings from analysing the competition results differ from those made in our previous assessment, as they do not reveal strong complementarity between the algorithms. Specifically, β-CROWN dominates in performance on every instance in each

Table 8: Performance comparison of CPU- and GPU-based verification algorithms in terms of the number of solved instances, relative marginal contribution (RMC), relative Shapley value (ϕ), computed for each category and aggregated $\epsilon \in \{0.004, 0.005, 0.008, 0.01, 0.012, 0.02, 0.025, 0.03, 0.04\}$.

ReLU						
Verifier	MNIST			CIFAR		
	Solved	RMC	ϕ	Solved	RMC	ϕ
BaDNB	9 455	0.10	0.20	20 408	0.48	0.43
BaBSB	3 716	0.00	0.00	2 690	0.00	0.00
β-CROWN	18 907	0.03	0.18	16 997	0.31	0.36
Marabou	9 457	0.44	0.20	3 651	0.00	0.00
MN-BaB	17 601	0.14	0.24	14 581	0.20	0.16
Neurify	8 206	0.04	0.02	8 173	0.00	0.03
nnenum	15 144	0.00	0.03	744	0.00	0.00
VeriNet	15 800	0.24	0.12	7 674	0.00	0.01
ReLU+MaxPool						
Verifier	MNIST			CIFAR		
	Solved	RMC	ϕ	Solved	RMC	ϕ
BaDNB	580	0.00	0.00	0	0.00	0.00
β-CROWN	1 127	0.99	0.74	0	0.00	0.00
Marabou	316	0.00	0.00	0	0.00	0.00
MN-BaB	966	0.00	0.22	576	1.00	1.00
Tanh						
Verifier	MNIST			CIFAR		
	Solved	RMC	ϕ	Solved	RMC	ϕ
β-CROWN	2 576	0.17	0.24	4 535	1.00	1.00
MN-BaB	0	0.00	0.00	0	0.00	0.00
VeriNet	4 307	0.83	0.76	0	0.00	0.00
Sigmoid						
Verifier	MNIST			CIFAR		
	Solved	RMC	ϕ	Solved	RMC	ϕ
β-CROWN	2 617	0.00	0.05	4 961	0.97	0.83
Marabou	0	0.00	0.00	0	0.00	0.00
MN-BaB	2 601	0.00	0.05	3 042	0.03	0.17
VeriNet	4 728	1.00	0.90	0	0.00	0.00

category, although relative Shapley values indicate complementary (for similar reasons as those outlined above).

This reflected in Figure F.1b, Figure F.1d and Figure F.1f. Concretely, these plots show the performance on an instance level for the two top-performing methods in each category (in terms of relative Shapley values). In the Complex and FC category, these are β-CROWN and MN-BaB, while in the CNN + ResNet category, these are β-CROWN and VeriNet. The latter category represents the only category in which a small degree of complementarity can be observed, as both verifiers solved some instances unsolved by the other. However, the fraction solved by VeriNet remains comparably small.

Finally, Table F.3 presents the joint analysis of CPU- and GPU-based methods based on the competition results. Notice that we did not perform a cost calibration in this case, as verifiers were employed on hardware with about equal costs. Most interestingly, we observed performance complementary between these methods in the CNN+ResNet category. More specifically, the CPU-based Marabou solver could solve several instances unsolved by GPU-based β-CROWN verifier, although the latter solved the most instances overall, as reflected in the relative Shapley values (0.53 *vs* 0.32). Again, this shows that there exist scenarios in which CPU-based methods complement GPU-based methods in their performance.

Overall, we find that the biggest difference between the results of the VNN competition and the results obtained in this study is the degree of complementarity between the GPU-based verification algorithms, as reflected by the marginal contribution and Shapley values. While the results from the VNN competition suggest that there is a single best GPU-based verifier that broadly dominates all other methods, the results presented in our study reveal a more nuanced story. This difference can most likely be attributed to the size and the diversity of the proposed benchmark: while the 2022 VNN Competition considered 17 neural networks as test cases for local robustness verification, our benchmark consists of 79 networks. At the same time, the competition provides valuable insights into how the considered verifiers perform when carefully adapted to a specific benchmark. Moreover, while both analyses have clear contributions, our results highlight the importance of introducing a larger and more diverse benchmark set.

6. Conclusions and Future Work

In this work, we assessed the performance of a collection of well-known, complete local robustness verification algorithms, *i.e.*, algorithms used to verify the robustness of an image classification network against small input perturbations. We found that all of these methods support ReLU-based networks, while other network types are strongly under-supported. While this has been suspected in the community, it has, to our knowledge, not yet been subject to formal study. Generally, we observed that all considered verification algorithms show severe limitations with regard to the network structures they can process – in many cases due to unsupported layer operations and in others due to undefined errors.

Furthermore, and more importantly, we presented evidence for strong performance complementarity: even within the same benchmark category (as defined based on verifier compatibility), any two verification systems outperform each other on distinct subsets of instances. As we have demonstrated, this complementarity can be exploited by combining individual verifiers into parallel portfolios. At the same time, automated portfolio construc-

tion comes with its own challenges (see, *e.g.*, König et al., 2022), leaving room for further research into the development and evaluation of appropriate frameworks.

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 addition, even for the same dataset, we found that the performance of a given verifier can change drastically depending on the perturbation radius; *i.e.*, an algorithm that performs well for a small value of ϵ might degrade in performance as the value of ϵ increases.

In future work, it would be interesting to analyse in more detail how the relative performance of verifiers depends on the given perturbation radius and on other performance-relevant characteristics of the given networks and image classification tasks. We suspect this to be an interesting yet challenging research direction, as it requires a novel definition of features specific to neural network verification problems. To the best of our knowledge, no research on the development of such meta-features has been conducted yet. Due to the specifics of both the verification problem instances as well as the verification algorithms that should be systematically explored, we consider this a non-trivial but important challenge to be solved in future work. Finally, we are interested in expanding our analysis to other datasets and machine learning tasks beyond supervised image classification.

Acknowledgments

This research was partially supported by TAILOR, a project funded by EU Horizon 2020 research and innovation program under GA No. 952215. The authors would like to thank Bram Renting, Corné Spek and Hadar Shavit for their support with installing and running the verification engines on our compute cluster.

References

Stanley Bak, Hoang-Dung Tran, Kerianne Hobbs, and Taylor T. Johnson. Improved Geometric Path Enumeration for Verifying ReLU Neural Networks. In *Proceedings of the 32nd International Conference on Computer Aided Verification (CAV 2020)*, pages 66–96, 2020.

Stanley Bak, Changliu Liu, and Taylor Johnson. The Second International Verification of Neural Networks Competition (VNN-COMP 2021): Summary and Results. *arXiv preprint arXiv:2109.00498*, 2021.

Thomas Bartz-Beielstein, Carola Doerr, Daan van den Berg, Jakob Bossek, Sowmya Chandrasekaran, Tome Eftimov, Andreas Fischbach, Pascal Kerschke, William La Cava, Manuel Lopez-Ibanez, et al. Benchmarking in Optimization: Best Practice and Open Issues. *arXiv preprint arXiv:2007.03488*, 2020.

Osbert Bastani, Yani Ioannou, Leonidas Lampropoulos, Dimitrios Vytiniotis, Aditya Nori, and Antonio Criminisi. Measuring Neural Net Robustness with Constraints. In *Advances in Neural Information Processing Systems 29 (NeurIPS 2016)*, pages 2613–2621, 2016.

Elena Botoeva, Panagiotis Kouvaros, Jan Kronqvist, Alessio Lomuscio, and Ruth Misener. Efficient Verification of ReLU-based Neural Networks via Dependency Analysis. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI-20)*, pages 3291–3299, 2020.

Rudy Bunel, Ilker Turkaslan, Philip Torr, Pushmeet Kohli, and Pawan K Mudigonda. A Unified View of Piecewise Linear Neural Network Verification. In *Advances in Neural Information Processing Systems 31 (NeurIPS 2018)*, pages 1–10, 2018.

Rudy Bunel, Jingyue Lu, Ilker Turkaslan, Philip H. S. Torr, Pushmeet Kohli, and M. Pawan Kumar. Branch and Bound for Piecewise Linear Neural Network Verification. *Journal of Machine Learning Research*, 21:42:1–42:39, 2020.

Marco Casadio, Ekaterina Komendantskaya, Matthew L. Daggitt, Wen Kokke, Guy Katz, Guy Amir, and Idan 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)*, pages 219–231, 2022.

George B. Dantzig. Linear programming. *Operations Research*, 50(1):42–47, 2002.

Alessandro De Palma, Rudy Bunel, Alban Desmaison, Krishnamurthy Dvijotham, Pushmeet Kohli, Philip H. S. Torr, and M. Pawan Kumar. Improved Branch and Bound for Neural Network Verification via Lagrangian Decomposition. *arXiv preprint arXiv:2104.06718*, 2021.

Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, and Jianguo Li. Boosting Adversarial Attacks With Momentum. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2018)*, pages 9185–9193, 2018.

Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. Output Range Analysis for Deep Neural Networks. In *Proceedings of the Tenth NASA Formal Methods Symposium (NFM 2018)*, pages 121–138, 2018.

Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy A Mann, and Pushmeet Kohli. A Dual Approach to Scalable Verification of Deep Networks. In *Proceedings of the 38th Conference on Uncertainty in Artificial Intelligence (UAI 2018)*, pages 550–559, 2018.

Ruediger 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)*, pages 269–286, 2017.

Claudio Ferrari, Mark Niklas Mueller, Nikola Jovanović, and Martin Vechev. Complete Verification via Multi-Neuron Relaxation Guided Branch-and-Bound. In *Proceedings of the 10th International Conference on Learning Representations (ICLR 2022)*, pages 1–15, 2022.

Alexandre Fréchette, Lars Kotthoff, Tomasz Michalak, Talal Rahwan, Holger Hoos, and Kevin Leyton-Brown. Using the shapley value to analyze algorithm portfolios. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI-16)*, pages 3397–3403, 2016.

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 *Proceedings of the 39th IEEE Symposium on Security and Privacy (IEEE S&P 2018)*, pages 3–18, 2018.

Carla P Gomes and Bart Selman. Algorithm portfolios. *Artificial Intelligence*, 126(1–2): 43–62, 2001.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and Harnessing Adversarial Examples. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015)*, pages 1–11, 2015.

Patrick Henriksen and Alessio Lomuscio. Efficient Neural Network Verification via Adaptive Refinement and Adversarial Search. In *Proceedings of the 24th European Conference on Artificial Intelligence (ECAI 2020)*, pages 2513–2520, 2020.

Patrick Henriksen, Kerstin Hammernik, Daniel Rueckert, and Alessio Lomuscio. Bias Field Robustness Verification of Large Neural Image Classifiers. In *Proceedings of the 32nd British Machine Vision Conference 2021 (BMVC 2021)*, pages 202–2016, 2021.

Holger H. Hoos and Thomas Stützle. *Stochastic Local Search: Foundations & Applications*. Elsevier / Morgan Kaufmann, 2004. ISBN 1-55860-872-9.

Bernardo A. Huberman, Rajan M. Lukose, and Tad Hogg. An economics approach to hard computational problems. *Science*, 275(5296):51–54, 1997.

Kai Jia and Martin C. Rinard. Efficient exact verification of binarized neural networks. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*, pages 1–14, 2020.

Kyle D Julian, Mykel J Kochenderfer, and Michael P Owen. Deep neural network compression for aircraft collision avoidance systems. *Journal of Guidance, Control, and Dynamics*, 42 (3):598–608, 2019.

Serdar Kadioglu, Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann. Algorithm Selection and Scheduling. In *Proceedings of the Seventeenth International Conference on Principles and Practice of Constraint Programming (CP2011)*, pages 454–469, 2011.

Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel 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)*, pages 97–117, 2017.

Guy Katz, Derek A. Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, David L. Dill, Mykel J. Kochenderfer, and Clark 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)*, pages 443–452, 2019.

Matthias König, Holger H Hoos, and Jan N van Rijn. Speeding up neural network robustness verification via algorithm configuration and an optimised mixed integer linear programming solver portfolio. *Machine Learning*, 111(12):4565–4584, 2022.

Ailsa H. Land and Alison G. Doig. An Automatic Method for Solving Discrete Programming Problems. In *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*, pages 105–132. Springer, 2010.

Linyi Li, Xiangyu Qi, Tao Xie, and Bo Li. Sok: Certified robustness for deep neural networks. *arXiv preprint arXiv:2009.04131*, 2020.

Alessio Lomuscio and Lalit Maganti. An approach to reachability analysis for feed-forward ReLU neural networks. *arXiv preprint arXiv:1706.07351*, 2017.

Mark Huasong Meng, Guangdong Bai, Sin Gee Teo, Zhe Hou, Yan Xiao, Yun Lin, and Jin Song Dong. Adversarial robustness of deep neural networks: A survey from a formal verification perspective. *IEEE Transactions on Dependable and Secure Computing*, 2022.

Jeet Mohapatra, Tsui-Wei Weng, Pin-Yu Chen, Sijia Liu, and Luca Daniel. Towards Verifying Robustness of Neural Networks Against A Family of Semantic Perturbations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2020)*, pages 241–249, 2020.

Leonardo de Moura and Nikolaj Bjørner. Satisfiability Modulo theories: An Appetizer. In *Proceedings of the Brazilian Symposium on Formal Methods (SBMF 2018)*, pages 23–36, 2009.

Christoph Müller, François Serre, Gagandeep Singh, Markus Püschel, and Martin Vechev. Scaling Polyhedral Neural Network Verification on gpus. In *Proceedings of Machine Learning and Systems 3 (MLSys 2021)*, pages 1–14, 2021.

Mark Niklas Müller, Christopher Brix, Stanley Bak, Changliu Liu, and Taylor T. Johnson. The Third International Verification of Neural Networks Competition (VNN-COMP 2022): Summary and Results. *arXiv preprint arXiv:2212.10376*, 2023.

Nina Narodytska, Shiva Prasad Kasiviswanathan, Leonid Ryzhyk, Mooly Sagiv, and Toby Walsh. Verifying properties of binarized deep neural networks. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI-18)*, pages 6615–6624, 2018.

Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram 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)*, pages 582–597, 2016.

Luca Pulina and A. Tacchella. NeVer: A Tool for Artificial Neural Networks Verification. *Annals of Mathematics and Artificial Intelligence*, pages 403–425, 2011a.

Luca Pulina and A. Tacchella. Checking Safety of Neural Networks with SMT Solvers: A Comparative Evaluation. In *AI*IA*, pages 127–138, 2011b.

Luca Pulina and Armando Tacchella. Challenging SMT Solvers to Verify Neural Networks. *AI Communications*, pages 117–135, 2012.

Karsten Scheibler, Leonore Winterer, Ralf Wimmer, and Bernd 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)*, pages 30–40, 2015.

David Shriver, Sebastian Elbaum, and Matthew B. Dwyer. DNNV: A Framework for Deep Neural Network Verification. In *Proceedings of the 33rd International Conference on Computer Aided Verification (CAV 2021)*, pages 137–150, 2021.

Gagandeep Singh and Timon Gehr. Boosting Robustness Certification of Neural networks. In *Proceedings of the 7th International Conference on Learning Representations (ICLR 2019)*, pages 1–12, 2019.

Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. Fast and Effective Robustness Certification. In *Advances in Neural Information Processing Systems 31 (NeurIPS 2018)*, pages 1–12, 2018.

Gagandeep Singh, Rupanshu Ganvir, Markus Püschel, and Martin Vechev. Beyond the Single Neuron Convex Barrier for Neural Network Certification. In *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*, pages 1–12, 2019a.

Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An Abstract Domain for Certifying Neural Networks. In *Proceedings of the 46th ACM SIGPLAN Symposium on Principles of Programming Languages (ACMPOPL 2019)*, pages 1–30, 2019b.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR 2014)*, pages 1–10, 2014.

Vincent Tjeng, Kai Xiao, and Russ Tedrake. Evaluating Robustness of Neural Networks with Mixed Integer Programming. In *Proceedings of the 7th International Conference on Learning Representations (ICLR 2019)*, pages 1–21, 2019.

Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Formal Security Analysis of Neural Networks using Symbolic Intervals. In *Proceedings of the 27th USENIX Security Symposium (USENIX Security 18)*, pages 1599–1614, 2018a.

Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Efficient Formal Safety Analysis of Neural Networks. In *Advances in Neural Information Processing Systems 31 (NeurIPS 2018)*, pages 6369–6379, 2018b.

Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and Zico Kolter. Beta-CROWN: Efficient Bound Propagation with Per-neuron Split Constraints for Neural Network Robustness Verification. In *Advances in Neural Information Processing Systems 34 (NeurIPS 2021)*, pages 29909–29921, 2021.

Lily Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Luca Daniel, Duane Boning, and Inderjit Dhillon. Towards Fast Computation of Certified Robustness for ReLU Networks. In *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)*, pages 5276–5285, 2018.

Eric Wong and Zico Kolter. Provable Defenses against Adversarial Examples via the Convex Outer Adversarial Polytope. In *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)*, pages 5286–5295, 2018.

Haoze Wu, Aleksandar Zeljic, Guy Katz, and Clark W. Barrett. Efficient neural network analysis with sum-of-infeasibilities. In Dana Fisman and Grigore Rosu, editors, *Proceedings of the 28th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2022)*, volume 13243, pages 143–163, 2022.

Weiming Xiang, Hoang-Dung Tran, and Taylor T Johnson. Output Reachable Set Estimation and Verification for Multilayer Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 29(11):5777–5783, 2018.

Lin Xu, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Satzilla: Portfolio-based algorithm selection for sat. *Journal of Artificial Intelligence Research*, 32:565–606, 2008.

Lin Xu, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Hydra-MIP: Automated Algorithm Configuration and Selection for Mixed Integer Programming. In *RCRA Workshop on Experimental evaluation of Algorithms for Solving Problems with Combinatorial Explosion*, pages 16–30, 2011.

Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient Neural Network Robustness Certification with General Activation Functions. *Advances in Neural Information Processing Systems 31 (NeurIPS 2018)*, 31:4944–4953, 2018.

Appendix A. Overview of considered neural networks

Table A.1: Considered neural networks trained on the MNIST dataset, along with their training method, employed activation function and source repository.

Network	Training	Activation	Source
cnn_max_mninst2[1]	Standard	ReLU	Marabou
cnn_max_mninst3[1]	Standard	ReLU	Marabou
convBig$_A$	DiffAI	ReLU	ERAN
convMed$_A$	PGD, $\epsilon = 0.1$	ReLU	ERAN
convMed$_B$	PGD, $\epsilon = 0.1$	Sigmoid	ERAN
convMed$_C$	PGD, $\epsilon = 0.1$	Tanh	ERAN
convMed$_D$	PGD, $\epsilon = 0.3$	ReLU	ERAN
convMed$_E$	PGD, $\epsilon = 0.3$	Sigmoid	ERAN
convMed$_F$	PGD, $\epsilon = 0.3$	Tanh	ERAN
convMed$_G$	Standard	ReLU	ERAN
convMed$_H$	Standard	Sigmoid	ERAN
convMed$_I$	Standard	Tanh	ERAN
convnet[1]	Standard	ReLU	ERAN
convSmall$_A$	DiffAI	ReLU	ERAN
convSmall$_B$	PGD	ReLU	ERAN
convSmall$_C$	Standard	ReLU	ERAN
convSuper	DiffAI	ReLU	ERAN
ffnn_6×500$_A$	PGD, $\epsilon = 0.1$	ReLU	ERAN
ffnn_6×500$_B$	PGD, $\epsilon = 0.1$	Sigmoid	ERAN
ffnn_6×500$_C$	PGD, $\epsilon = 0.1$	Tanh	ERAN
ffnn_6×500$_D$	PGD, $\epsilon = 0.3$	ReLU	ERAN
ffnn_6×500$_E$	PGD, $\epsilon = 0.3$	Sigmoid	ERAN
ffnn_6×500$_F$	PGD, $\epsilon = 0.3$	Tanh	ERAN
ffnn_6×500$_G$	Standard	ReLU	ERAN
ffnn_6×500$_H$	Standard	Sigmoid	ERAN
ffnn_6×500$_I$	Standard	Tanh	ERAN
mnist-net	Standard	ReLU	Venus
mnist-net_256×2	Standard	ReLU	VNN-COMP
mnist-net_256×4	Standard	ReLU	VNN-COMP
mnist-net_256×6	Standard	ReLU	VNN-COMP
mnist_3×100	Standard	ReLU	ERAN
mnist_3×50	Standard	ReLU	ERAN
mnist_4×1024	Standard	ReLU	ERAN
mnist_5×100	Standard	ReLU	ERAN
mnist_6×100	Standard	ReLU	ERAN
mnist_6×200	Standard	ReLU	ERAN
mnist_9×100	Standard	ReLU	ERAN
mnist_9×200	Standard	ReLU	ERAN
mnist_conv[1]	Standard	ReLU	ERAN
mnist_nn	Standard	ReLU	VeriNet
rsl18a-linf01	SDP	ReLU	MIPVerify

[1]Employs MaxPooling layers

Table A.2: Considered neural networks trained on the CIFAR-10 dataset, along with their training method, employed activation function and source repository.

Network	Training	Activation	Source
cifar_base_kw	Wong and Kolter (2018), $\epsilon = 1/255$	ReLU	OVAL
cifar_deep_kw	Wong and Kolter (2018), $\epsilon = 1/255$	ReLU	OVAL
cifar_wide_kw	Wong and Kolter (2018), $\epsilon = 1/255$	ReLU	OVAL
cifar_base_kw_simp	Wong and Kolter (2018), $\epsilon = 1/255$	ReLU	Marabou
cifar_deep_kw_simp	Wong and Kolter (2018), $\epsilon = 1/255$	ReLU	Marabou
cifar_wide_kw_simp	Wong and Kolter (2018), $\epsilon = 1/255$	ReLU	Marabou
cifar-net	Standard	ReLU	Venus
cifar_conv[1]	Standard	ReLU	ERAN
cifar_4×100	Standard	ReLU	ERAN
cifar_6×100	Standard	ReLU	ERAN
cifar_7×1024	Standard	ReLU	ERAN
cifar_9×200	Standard	ReLU	ERAN
cifar_4×100	Standard	ReLU	ERAN
cifar10_2_255	COLT, $\epsilon = 2/255$	ReLU	VNN-COMP
cifar10_8_255	COLT, $\epsilon = 8/255$	ReLU	VNN-COMP
cifar10_2_255_simplified	COLT, $\epsilon = 2/255$	ReLU	VNN-COMP
cifar10_8_255_simplified	COLT, $\epsilon = 8/255$	ReLU	VNN-COMP
convBig$_B$	PGD, $\epsilon = 2/255$	ReLU	ERAN
convMed$_J$	PGD, $\epsilon = 2/255$	ReLU	ERAN
convMed$_K$	PGD, $\epsilon = 2/255$	Sigmoid	ERAN
convMed$_L$	PGD, $\epsilon = 2/255$	Tanh	ERAN
convMed$_M$	PGD, $\epsilon = 8/255$	ReLU	ERAN
convMed$_N$	PGD, $\epsilon = 8/255$	Sigmoid	ERAN
convMed$_O$	PGD, $\epsilon = 8/255$	Tanh	ERAN
convMed$_P$	Standard	ReLU	ERAN
convMed$_Q$	Standard	Sigmoid	ERAN
convMed$_R$	Standard	Tanh	ERAN
convSmall$_E$	DiffAI	ReLU	ERAN
convSmall$_F$	Standard	ReLU	ERAN
ffnn_6×500$_J$	PGD, $\epsilon = 2/255$	ReLU	ERAN
ffnn_6×500$_K$	PGD, $\epsilon = 2/255$	Sigmoid	ERAN
ffnn_6×500$_L$	PGD, $\epsilon = 2/255$	Tanh	ERAN
ffnn_6×500$_M$	PGD, $\epsilon = 8/255$	ReLU	ERAN
ffnn_6×500$_N$	PGD, $\epsilon = 8/255$	Sigmoid	ERAN
ffnn_6×500$_O$	PGD, $\epsilon = 8/255$	Tanh	ERAN
ffnn_6×500$_P$	Standard	ReLU	ERAN
ffnn_6×500$_Q$	Standard	Sigmoid	ERAN
ffnn_6×500$_R$	Standard	Tanh	ERAN

[1]Employs MaxPooling layers

Appendix B. Absolute marginal contributions and Shapley values

Table B.1: Performance comparison of CPU-based verification algorithms in terms of the number of solved instances, absolute marginal contribution (MC), absolute Shapley value (ϕ_{abs}) and CPU running time averaged per problem instance, computed for each category with ϵ set at 0.012.

ReLU								
Verifier	MNIST				CIFAR			
	Solved	MC	ϕ_{abs}	Avg. Time [CPU s]	Solved	MC	ϕ_{abs}	Avg. Time [CPU s]
BaBSB	358	23	118	3 241	307	0	86	2 924
Marabou	1 001	20	312	1 801	400	0	117	2 153
Neurify	871	26	265	1 964	915	119	411	235
nnenum	1 754	18	600	389	76	8	28	3 337
VeriNet	1 799	16	618	263	841	31	330	500
ReLU+MaxPool								
Verifier	MNIST				CIFAR			
	Solved	MC	ϕ_{abs}	Avg. Time	Solved	MC	ϕ_{abs}	Avg. Time
Marabou	5	5	5	57	0	0	0	3 600
Tanh								
Verifier	MNIST				CIFAR			
	Solved	MC	ϕ_{abs}	Avg. Time	Solved	MC	ϕ_{abs}	Avg. Time
VeriNet	556	556	556	55	0	0	0	3 600
Sigmoid								
Verifier	MNIST				CIFAR			
	Solved	MC	ϕ_{abs}	Avg. Time	Solved	MC	ϕ_{abs}	Avg. Time
Marabou	0	0	0	3 600	0	0	0	3 600
VeriNet	581	581	581	55	0	0	0	3 600

Table B.2: Performance comparison of GPU-based verification algorithms in terms of the number of solved instances, absolute marginal contribution (MC), absolute Shapley value (ϕ_{abs}) and average GPU running time, computed for each category with ϵ set at 0.012.

ReLU								
Verifier	MNIST				CIFAR			
	Solved	MC	ϕ_{abs}	Avg. Time [GPU s]	Solved	MC	ϕ_{abs}	Avg. Time [GPU s]
BaDNB	1 188	8	440	1 760	2 332	250	1 066	116
β-CROWN	2 247	0	966	96	1 828	7	693	818
MN-BaB	2 103	18	903	325	1 639	20	604	1 110
ReLU+MaxPool								
Verifier	MNIST				CIFAR			
	Solved	MC	ϕ_{abs}	Avg. Time	Solved	MC	ϕ_{abs}	Avg. Time
BaDNB	85	0	29	1 399	0	0	0	3 600
β-CROWN	128	12	56	0.4	0	0	0	3 600
MN-BaB	115	0	44	366	64	64	64	0.008
Tanh								
Verifier	MNIST				CIFAR			
	Solved	MC	ϕ_{abs}	Avg. Time	Solved	MC	ϕ_{abs}	Avg. Time
β-CROWN	319	319	319	1.16	497	496	497	0.70
MN-BaB	0	0	0	3 600	0	0	0	3 600
Sigmoid								
Verifier	MNIST				CIFAR			
	Solved	MC	ϕ_{abs}	Avg. Time	Solved	MC	ϕ_{abs}	Avg. Time
β-CROWN	306	2	154	13	538	209	374	60
MN-BaB	305	1	153	24	338	9	174	1 376

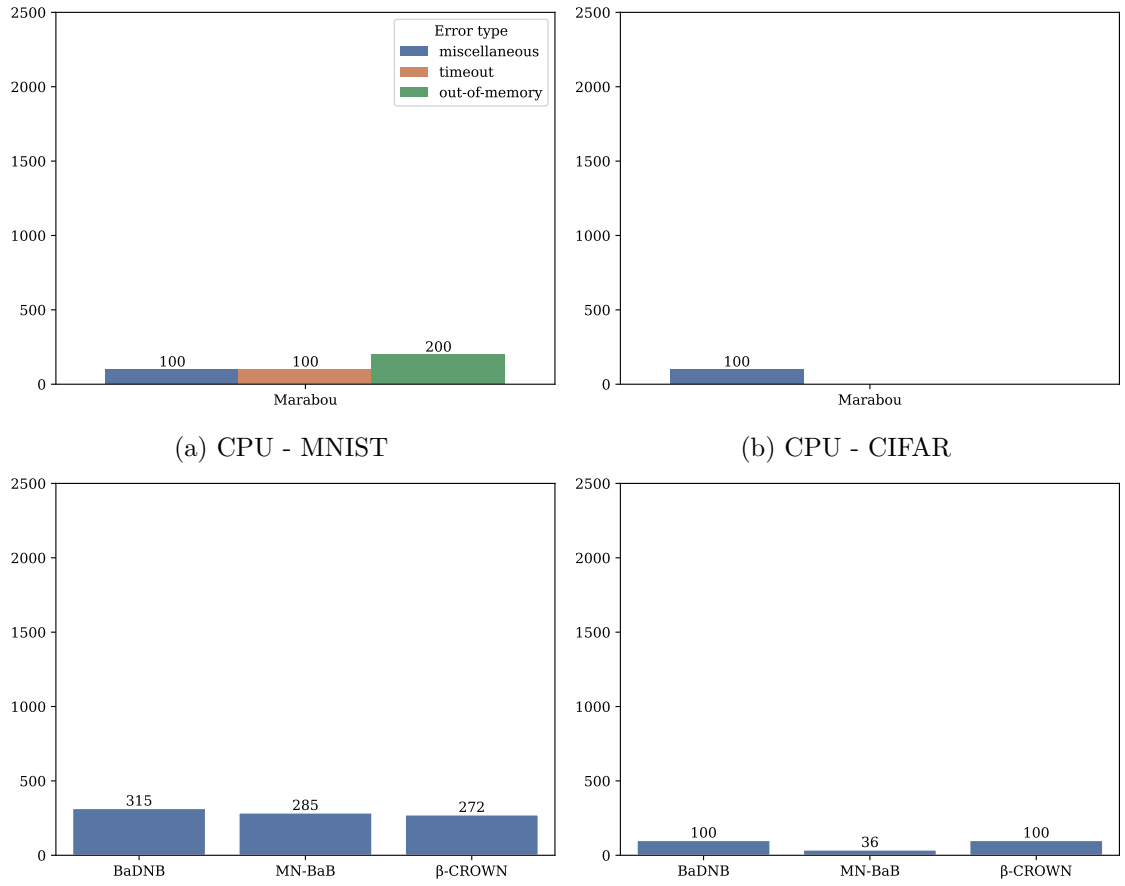
Table B.3: Performance comparison of CPU-based verification algorithms in terms of the number of solved instances, absolute marginal contribution (MC), absolute Shapley value (ϕ_{abs}) and average CPU running time, computed for each category with aggregated $\epsilon \in \{0.004, 0.005, 0.008, 0.01, 0.012, 0.02, 0.025, 0.03, 0.04\}$.

ReLU								
Verifier	MNIST				CIFAR			
	Solved	MC	ϕ_{abs}	Avg. Time [CPU s]	Solved	MC	ϕ_{abs}	Avg. Time [CPU s]
BaBSB	3 716	103	1 062	3 223	2 690	0	759	2 964
Marabou	9 457	784	3 309	1 721	3 651	8	1 078	2 145
Neurify	8 206	212	2 418	1 899	8 173	1 059	3 662	289
nnenum	15 144	288	5 093	543	744	61	268	3 315
VeriNet	15 800	411	5 442	367	7 674	365	3 061	486
ReLU+MaxPool								
Verifier	MNIST				CIFAR			
	Solved	MC	ϕ_{abs}	Avg. Time	Solved	MC	ϕ_{abs}	Avg. Time
Marabou	316	316	316	50	0	0	0	3 600
Tanh								
Verifier	MNIST				CIFAR			
	Solved	MC	ϕ_{abs}	Avg. Time	Solved	MC	ϕ_{abs}	Avg. Time
VeriNet	4 307	4 307	4 307	59	0	0	0	3 600
Sigmoid								
Verifier	MNIST				CIFAR			
	Solved	MC	ϕ_{abs}	Avg. Time	Solved	MC	ϕ_{abs}	Avg. Time
Marabou	0	0	0	3 600	0	0	0	3 600
VeriNet	4 728	4 728	4 728	59	0	0	0	3 600

Table B.4: Performance comparison of GPU-based verification algorithms in terms of the number of solved instances, absolute marginal contribution (MC), absolute Shapley value (ϕ_{abs}) and average GPU running time, computed for each category with $\epsilon \in \{0.004, 0.005, 0.008, 0.01, 0.012, 0.02, 0.025, 0.03, 0.04\}$.

ReLU								
Verifier	MNIST				CIFAR			
	Solved	MC	ϕ_{abs}	Avg. Time [GPU s]	Solved	MC	ϕ_{abs}	Avg. Time [GPU s]
BaDNB	9 886	287	3 832	1 864	21 438	2 251	9 823	100
β-CROWN	18 955	6	8 226	148	17 014	72	6 521	784
MN-BaB	17 799	110	7 700	363	14 675	170	5 401	1 174
ReLU+MaxPool								
Verifier	MNIST				CIFAR			
	Solved	MC	ϕ_{abs}	Avg. Time	Solved	MC	ϕ_{abs}	Avg. Time
BaDNB	720	5	244	1 493	0	0	0	3 600
β-CROWN	1 127	160	525	19	0	0	0	3 600
MN-BaB	966	1	365	531	576	576	576	0.009
Tanh								
Verifier	MNIST				CIFAR			
	Solved	MC	ϕ_{abs}	Avg. Time	Solved	MC	ϕ_{abs}	Avg. Time
β-CROWN	2 576	2 576	2 576	1.16	4 535	4 535	4 535	0.75
MN-BaB	0	0	0	3 600	0	0	0	3 600
Sigmoid								
Verifier	MNIST				CIFAR			
	Solved	MC	ϕ_{abs}	Avg. Time	Solved	MC	ϕ_{abs}	Avg. Time
β-CROWN	2 617	32	1 325	23	4 961	1 983	3 472	46
MN-BaB	2 601	16	1 309	44	3 042	64	1 553	1 421

Appendix C. Error analysis for ReLU+MaxPool, Tanh and Sigmoid categories



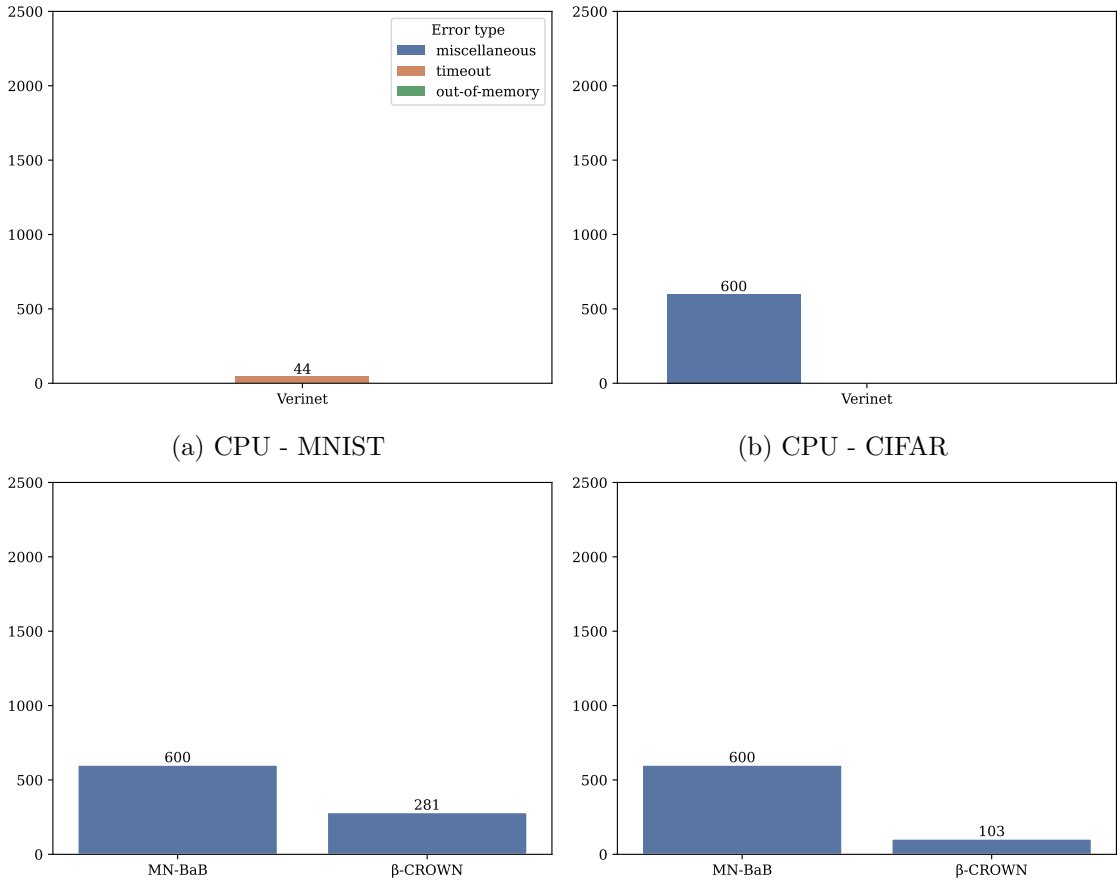
(a) CPU - MNIST

(b) CPU - CIFAR

(c) GPU - MNIST

(d) GPU - CIFAR

Figure C.1: Frequency of error types returned by the considered verification algorithms on instances in the ReLU+MaxPool category. The total number of instances in this category is 400 for MNIST and 100 for CIFAR.



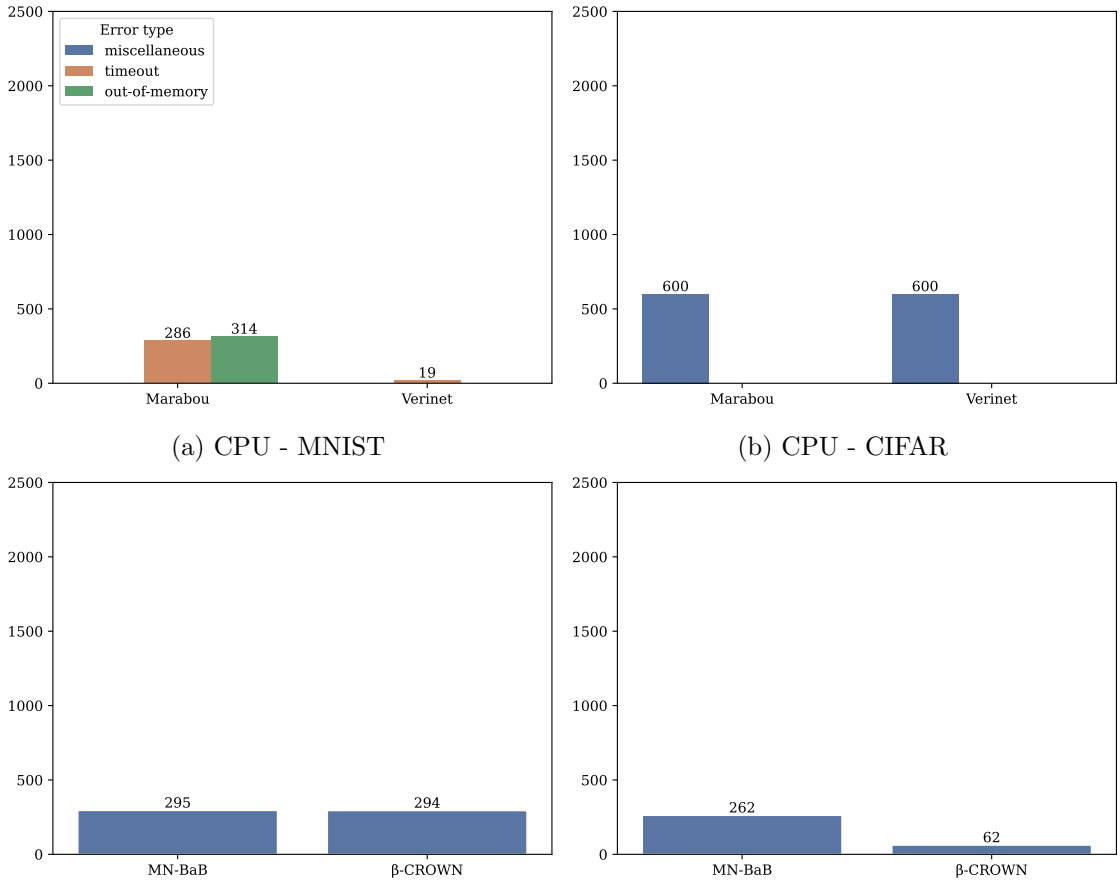
(a) CPU - MNIST

(b) CPU - CIFAR

(c) GPU - MNIST

(d) GPU - CIFAR

Figure C.2: Frequency of error types returned by the considered verification algorithms on instances in the Tanh category. The total number of instances in this category is 600 for MNIST and 600 for CIFAR.



(a) CPU - MNIST

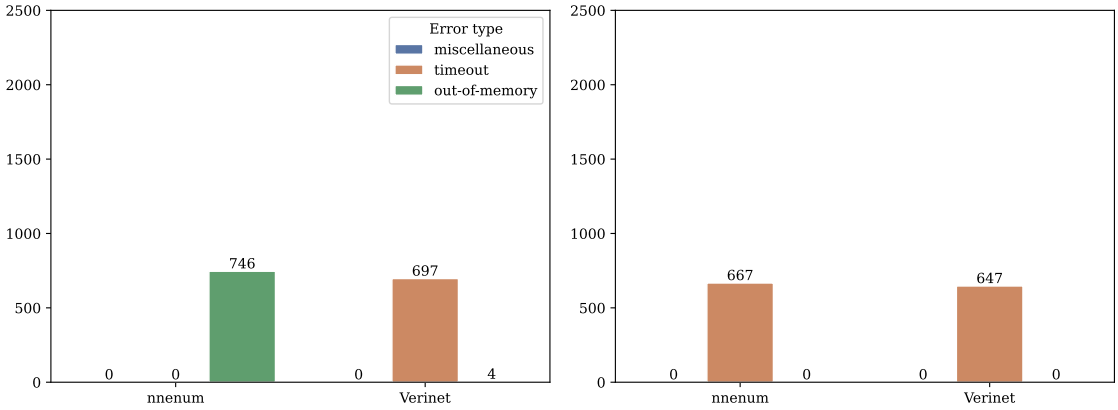
(b) CPU - CIFAR

(c) GPU - MNIST

(d) GPU - CIFAR

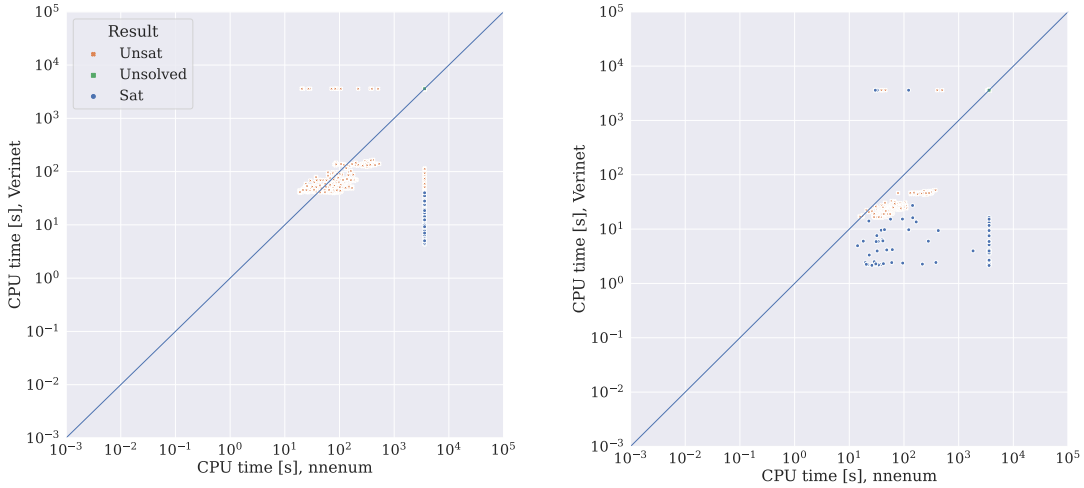
Figure C.3: Frequency of error types returned by the considered verification algorithms on instances in the Sigmoid category. The total number of instances in this category is 600 for MNIST and 600 for CIFAR.

Appendix D. Analysis of CPU-based methods with larger memory budget and perturbation radius



(a) Memory limit = 3GB

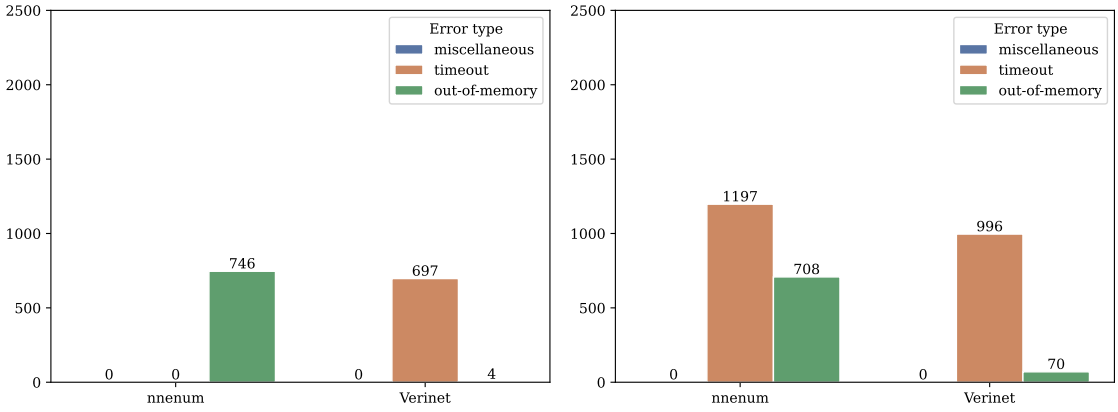
(b) Memory limit = 30GB



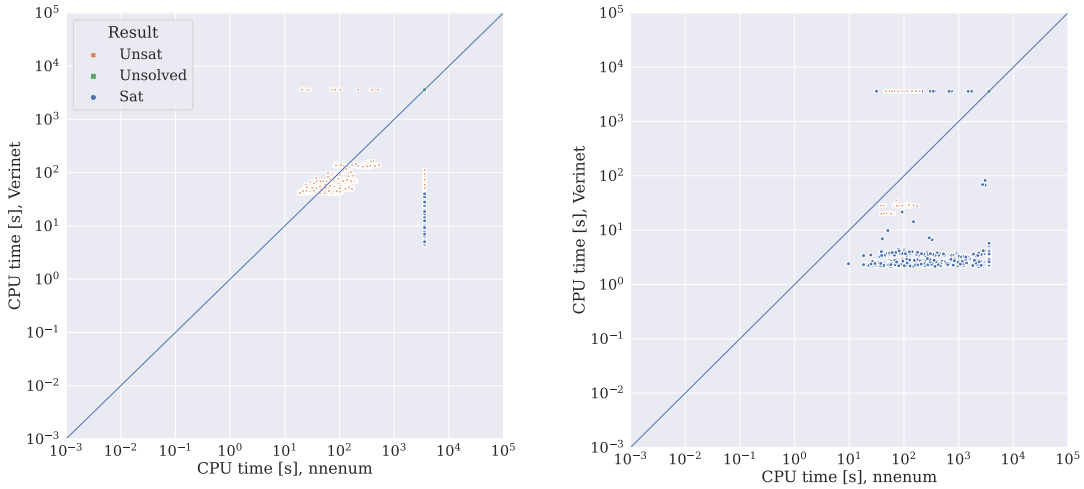
(c) Memory limit = 3GB

(d) Memory limit = 30GB

Figure D.1: **Top row**: Frequency of error types returned by the two top-performing verification methods (in terms of the number of solved instances) on instances in the ReLU category, with a memory limit of (a) 3GB or (b) 30GB. **Bottom row**: Performance comparison of the two top-performing verification methods (in terms of relative Shapley value) in the ReLU category for CPU-based methods, with a memory limit of (c) 3GB or (d) 30GB.



(a) $\epsilon = 0.012$ (b) $\epsilon = 0.2$



(c) $\epsilon = 0.012$ (d) $\epsilon = 0.2$

Figure D.2: **Top row**: Frequency of error types returned by the two top-performing verification methods (in terms of the number of solved instances) on instances in the ReLU category, when (a) $\epsilon = 0.012$ or (b) $\epsilon = 0.02$. **Bottom row**: Performance comparison of the two top-performing verification methods (in terms of relative Shapley value) in the ReLU category for CPU-based methods, when (c) $\epsilon = 0.012$ or (d) $\epsilon = 0.02$.

Appendix E. Analysis of *unsat* Instances

Table E.1: Performance comparison of CPU- and GPU-based verification algorithms in terms of the number of *unsat* instances, relative marginal contribution (RMC), relative Shapley value (ϕ), computed for each category and $\epsilon = 0.012$.

ReLU						
Verifier	MNIST			CIFAR		
	Solved	RMC	ϕ	Solved	RMC	ϕ
BaDNB	1 072	0.13	0.25	86	0.65	0.63
BaBSB	161	0.00	0.00	0	0.00	0.00
β-CROWN	2 143	0.00	0.23	61	0.35	0.36
Marabou	995	0.07	0.03	6	0.00	0.00
MN-BaB	2 025	0.80	0.40	16	0.00	0.00
Neurify	748	0.00	0.00	20	0.00	0.00
nnenum	1 686	0.00	0.03	26	0.00	0.00
VeriNet	1 675	0.00	0.03	20	0.00	0.00
ReLU+MaxPool						
Verifier	MNIST			CIFAR		
	Solved	RMC	ϕ	Solved	RMC	ϕ
BaDNB	59	0.00	0.10	0	0.00	0.00
β-CROWN	88	1.00	0.52	0	0.00	0.00
Marabou	5	0.00	0.00	0	0.00	0.00
MN-BaB	86	0.00	0.38	0	0.00	0.00
Tanh						
Verifier	MNIST			CIFAR		
	Solved	RMC	ϕ	Solved	RMC	ϕ
β-CROWN	291	0.09	0.18	3	1.00	1.00
MN-BaB	0	0.00	0.00	0	0.00	0.00
VeriNet	527	0.91	0.82	0	0.00	0.00
Sigmoid						
Verifier	MNIST			CIFAR		
	Solved	RMC	ϕ	Solved	RMC	ϕ
β-CROWN	272	0.00	0.03	66	1.00	0.00
Marabou	0	0.00	0.00	0	0.00	0.00
MN-BaB	272	0.00	0.03	0	0.00	0.00
VeriNet	544	1.00	0.94	0	0.00	0.00

Table E.2: Performance comparison of CPU- and GPU-based verification algorithms in terms of the number of *unsat* instances, relative marginal contribution (*RMC*), relative Shapley value (ϕ), computed for each category and aggregated $\epsilon \in \{0.004, 0.005, 0.008, 0.01, 0.012, 0.02, 0.025, 0.03, 0.04\}$.

ReLU						
Verifier	MNIST			CIFAR		
	Solved	*RMC*	ϕ	Solved	*RMC*	ϕ
BaDNB	8 059	0.15	0.21	1 069	0.63	0.59
BaBSB	2 303	0.00	0.00	0	0.00	0.00
β-CROWN	17 433	0.04	0.19	866	0.36	0.39
Marabou	9 290	0.61	0.25	60	0.00	0.00
MN-BaB	16 588	0.20	0.28	144	0.00	0.00
Neurify	6 992	0.00	0.00	168	0.00	0.00
nnenum	14 601	0.00	0.03	223	0.00	0.01
VeriNet	14 317	0.00	0.02	177	0.00	0.00
ReLU+MaxPool						
Verifier	MNIST			CIFAR		
	Solved	*RMC*	ϕ	Solved	*RMC*	ϕ
BaDNB	418	0.00	0.08	0	0.00	0.00
β-CROWN	573	1.00	0.50	0	0.00	0.00
Marabou	274	0.00	0.02	0	0.00	0.00
MN-BaB	566	0.00	0.40	0	0.00	0.00
Tanh						
Verifier	MNIST			CIFAR		
	Solved	*RMC*	ϕ	Solved	*RMC*	ϕ
β-CROWN	2 248	0.15	0.22	151	1.00	1.00
MN-BaB	0	0.00	0.00	0	0.00	0.00
VeriNet	3 993	0.85	0.78	0	0.00	0.00
Sigmoid						
Verifier	MNIST			CIFAR		
	Solved	*RMC*	ϕ	Solved	*RMC*	ϕ
β-CROWN	2 290	0.00	0.04	575	1.00	1.00
Marabou	0	0.00	0.00	0	0.00	0.00
MN-BaB	2 302	0.00	0.04	0	0.00	0.00
VeriNet	4 448	1.00	0.92	0	0.00	0.00

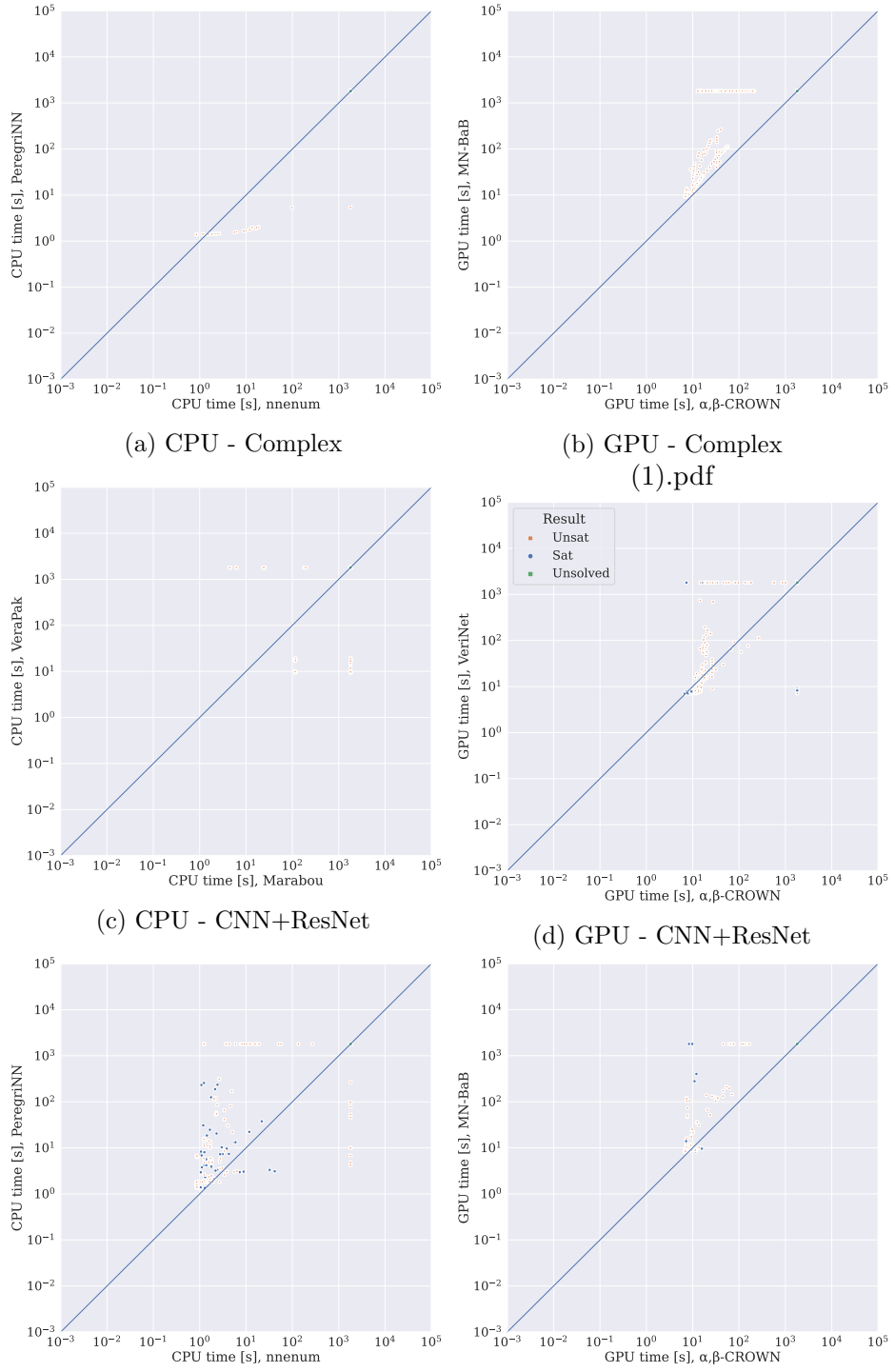
Appendix F. Analysis of the 2022 VNN Competition results

Table F.1: Performance comparison of CPU-based verification algorithms in terms of the number of solved instances, absolute and relative marginal contribution (*MC*, *RMC*), absolute and relative Shapley value (ϕ_{abs}, ϕ) as well as average running time, computed for each category from the 2022 VNN Competition.

Complex Verifier						
	Solved	*MC*	*RMC*	ϕ_{abs}	ϕ	Avg. Time
AveriNN	0	0	0.00	0	0.00	192
Debona	2	0	0.00	1	0.04	192
FastBATLLNN	0	0	0.00	0	0.00	192
Marabou	0	0	0.00	0	0.00	192
nnenum	23	0	0.00	11	0.46	190
PeregriNN	24	1	1.00	12	0.50	189
VeraPak	0	0	0.00	0	0.00	192
CNN + ResNet Verifier						
	Solved	*MC*	*RMC*	ϕ_{abs}	ϕ	Avg. Time
AveriNN	0	0	0.00	0	0.00	357
Debona	0	0	0.00	0	0.00	357
FastBATLLNN	0	0	0.00	0	0.00	357
Marabou	122	91	0.61	106	0.44	264
nnenum	81	17	0.11	48	0.20	273
PeregriNN	57	0	0.00	28	0.12	325
VeraPak	72	42	0.28	57	0.24	254
FC Verifier						
	Solved	*MC*	*RMC*	ϕ_{abs}	ϕ	Avg. Time
AveriNN	100	0	0.00	20	0.05	166
Debona	339	3	0.30	82	0.19	91
FastBATLLNN	32	1	0.10	10	0.0	0.5
Marabou	404	0	0.00	102	0.24	53
nnenum	411	1	0.10	105	0.24	37
PeregriNN	397	2	0.20	102	0.24	48
VeraPak	50	3	0.30	13	0.03	66

Table F.2: Performance comparison of GPU-based verification algorithms in terms of the number of solved instances, absolute and relative marginal contribution (*MC*, *RMC*), absolute and relative Shapley value (ϕ_{abs}, ϕ) as well as average running time, computed for each category from the 2022 VNN Competition.

Complex Verifier						
	Solved	*MC*	*RMC*	ϕ_{abs}	ϕ	Avg. Time
β-CROWN	191	66	1.00	0	0.62	72
MN-BaB	125	0	0.00	0	0.28	164
VeriNet	60	0	0.00	0	0.10	187
CNN + ResNet Verifier						
	Solved	*MC*	*RMC*	ϕ_{abs}	ϕ	Avg. Time
β-CROWN	312	28	1.00	0	0.42	107
MN-BaB	254	0	0.00	0	0.28	179
VeriNet	259	0	0.00	0	0.30	171
FC Verifier						
	Solved	*MC*	*RMC*	ϕ_{abs}	ϕ	Avg. Time
β-CROWN	448	11	1.00	0	0.35	15
MN-BaB	433	0	0.00	0	0.33	30
VeriNet	435	0	0.00	0	0.32	21



(a) CPU - Complex

(b) GPU - Complex

(1).pdf

(c) CPU - CNN+ResNet

(d) GPU - CNN+ResNet

(e) CPU - FC

(f) GPU - FC

Figure F.1: Performance comparison of the two top-performing verification methods (in terms of relative Shapley value) in each category from the 2022 VNN Competition. For better interpretability, instances that were not solved within their respective time limit are displayed with the maximum running time attributed to any instance in the benchmark set (*i.e.*, 1 800 seconds).

Table F.3: Performance comparison of GPU- and CPU-based verification algorithms in terms of the number of solved instances, absolute and relative marginal contribution (*MC*, *RMC*) as well as absolute and relative Shapley value (ϕ_{abs}, ϕ), computed for each category from the 2022 VNN Competition.

Complex Verifier	Solved	*MC*	*RMC*	ϕ_{abs}	ϕ
AveriNN	0	0	0.00	0	0.00
β-CROWN	191	66	0.99	20	0.91
Debona	2	0	0.00	0	0.04
FastBATLLNN	0	0	0.00	0	0.00
Marabou	0	0	0.00	0	0.00
MN-BaB	125	0	0.00	2	0.09
nnenum	23	0	0.00	0	0.46
PeregriNN	24	1	0.01	0	0.50
VeraPak	0	0	0.00	0	0.00
VeriNet	60	0	0.00	0	0.10
CNN + ResNet Verifier	Solved	*MC*	*RMC*	ϕ_{abs}	ϕ
AveriNN	0	0	0.00	0	0.00
β-CROWN	312	15	0.28	6	0.32
Debona	0	0	0.00	0	0.00
FastBATLLNN	0	0	0.00	0	0.00
Marabou	122	36	0.68	10	0.53
MN-BaB	254	0	0.00	1	0.05
nnenum	81	0	0.00	0	0.00
PeregriNN	57	0	0.00	0	0.00
VeraPak	72	2	0.04	1	0.05
VeriNet	259	0	0.00	1	0.05
FC Verifier	Solved	*MC*	*RMC*	ϕ_{abs}	ϕ
AveriNN	100	0	0.00	0	0.00
β-CROWN	448	9	1.00	3	1.00
Debona	339	0	0.00	0	0.00
FastBATLLNN	32	0	0.00	0	0.00
Marabou	404	0	0.00	0	0.00
MN-BaB	433	0	0.00	0	0.00
nnenum	411	0	0.00	0	0.00
PeregriNN	397	0	0.00	0	0.00
VeraPak	50	0	0.00	0	0.00
VeriNet	435	0	0.00	0	0.00