Universiteit
Leiden
The Netherlands

Automated machine learning for neural network verification

König, H.M.T.

Citation
König, H. M. T. (2025, October 9). *Automated machine learning for neural network verification*. Retrieved from https://hdl.handle.net/1887/4266921

Version:	Publisher's Version
License:	Licence agreement concerning inclusion of doctoral thesis in the Institutional Repository of the University of Leiden
Downloaded from:	https://hdl.handle.net/1887/4266921

Note: To cite this publication please use the final published version (if applicable).

Chapter 3

Critically Assessing the State of the Art in Neural Network Verification

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 [15]. 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 3.1.

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 [87]. 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.*, [117] or [52] for SAT.

In this chapter, 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 [95]; 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 in this chapter are as follows and, altogether, seek to answer RQ1 of this thesis:

- 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.

3.1 Common Practices in Benchmarking Neural Network Verifiers

As explained in Chapter 2, formal verification algorithms can be either *complete* or *incomplete* [71]. 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 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 study, 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 [88] or [49] on binarised networks.

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 [87]. The VNN competition was held every year since 2020, with different protocols (*e.g.*, for running experiments, scoring, etc.), benchmarks and participants. Here, we focus on the 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 [87] 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 [87]; 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

Table 3.1: 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	[12]
BaDNB	✓	✓	✗	GPU	[22]
$\alpha\beta$-CROWN	✓	✓	✗	GPU	[111]
ERAN[1]	✓	✓	✓	GPU	[96]
Marabou	✓	✓	✓	CPU	[56]
MIPVerify[2]	✓	✗	✓	CPU	[104]
MN-BaB	✓	✓	✗	GPU	[29]
Neurify	✓	✗	✓	CPU	[110]
nnenum	✓	✓	✓	CPU	[4]
Planet[3]	✓	✗	✓	CPU	[27]
Reluplex[4]	✓	✗	✓	CPU	[55]
VeriNet	✓	✗	✓	CPU	[40]

[1]Superseded by MN-BaB.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[3]Superseded by BaB.
[4]Superseded by Marabou.

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.

3.2 Verification Algorithms under Assessment

We consider eight complete neural network verification algorithms in this study; 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 [95]. Table 3.1 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 [110] or BaDNB [22], did not participate in the 2022 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 [3] 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 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.

3.2.1 CPU-Based Methods

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

BaB. The algorithm proposed by Bunel et al. [12] 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 chapter, we refer to it as BaBSB.

Marabou. The Marabou framework [56] 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 func-

tions 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. [110] 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. [4] 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 & Lomuscio [40] 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.

3.2.2 GPU-Based Methods

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

BaDNB. The BaDNB verifier introduced by DePalma et al. [22] 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. $\alpha\beta$-CROWN [111] 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. $\alpha\beta$-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. [12], the verifier is complete and can be efficiently parallelised using GPUs. $\alpha\beta$-CROWN can handle ReLU, Sigmoid and Tanh activations as well as MaxPooling layers.

MN-BaB. The MN-BaB verifier [29] builds on the multi-neuron constraints un-

Table 3.2: 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) $\alpha\beta$-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)

derlying the ERAN toolkit [85, 96, 99, 97, 98] 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.

3.3 Setup for Empirical Evaluation

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

3.3.1 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 [41, 5], 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

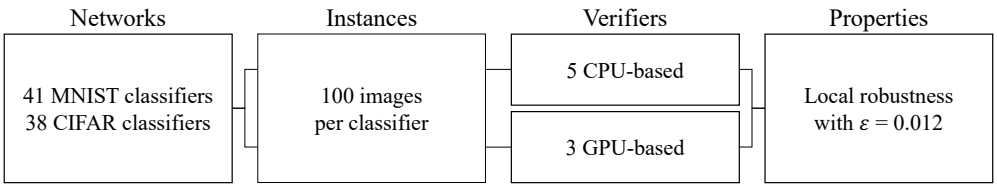


Figure 3.1: Schematic overview of the setup of experiments.

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 Table 3.3 and Table 3.3.

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 $\{0.004, 0.005, 0.008, 0.01, 0.012, 0.02, 0.025, 0.03, 0.04\}$. 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 [71], and centred around commonly chosen values for ϵ [114, 8, 109].

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 3.2. Notice that, in general, the ground truth for any given problem instance is not known a priori. At the same, even state-of-the-art verifiers are known to sometimes produce different results for the same instance [10]. As some of the considered verifiers do not return counterexamples by default, we treated these instance as *unsat*.

3.3. Setup for Empirical Evaluation

Table 3.3: 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 3.4: 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	[113], $\epsilon = 1/255$	ReLU	OVAL
cifar_deep_kw	[113], $\epsilon = 1/255$	ReLU	OVAL
cifar_wide_kw	[113], $\epsilon = 1/255$	ReLU	OVAL
cifar_base_kw_simp	[113], $\epsilon = 1/255$	ReLU	Marabou
cifar_deep_kw_simp	[113], $\epsilon = 1/255$	ReLU	Marabou
cifar_wide_kw_simp	[113], $\epsilon = 1/255$	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

3.3.2 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* [33] 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

$$MC_v(V) = s(V) - s(V \setminus \{v\}) \tag{3.1}$$

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:

$$\phi_v(V) = \frac{1}{n!} \cdot \sum_{\pi \in \Pi^V} \left(s(V_v^\pi) - s(V_v^\pi \setminus \{v\}) \right) \tag{3.2}$$

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.

3.3.3 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 [95]. 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 $\alpha\beta$-CROWN[1], OVAL$-$BaB[2], and MN-BaB[3] framework, respectively. These methods also return a counter-example to the property in cases where a violation is found.

3.4 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 3.2 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.

3.4.1 CPU-Based Methods

Table 3.5 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 3.2 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

[1]Commit 7a46097192207dfbb2fa7135857d6bc4ae7d6cd5
[2]Commit 9e1606044759da5693f226ce489e9d4dded21bd6
[3]Commit 2aa12b145bb61342f4c464b64be3467b3a275e46

Table 3.5: Performance comparison of CPU-based verification algorithms in terms of the number of solved instances, relative marginal contribution (*RMC*), relative Shapley value (ϕ) and 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

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 Table 3.6, 3.8, 3.10 and 3.12. 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

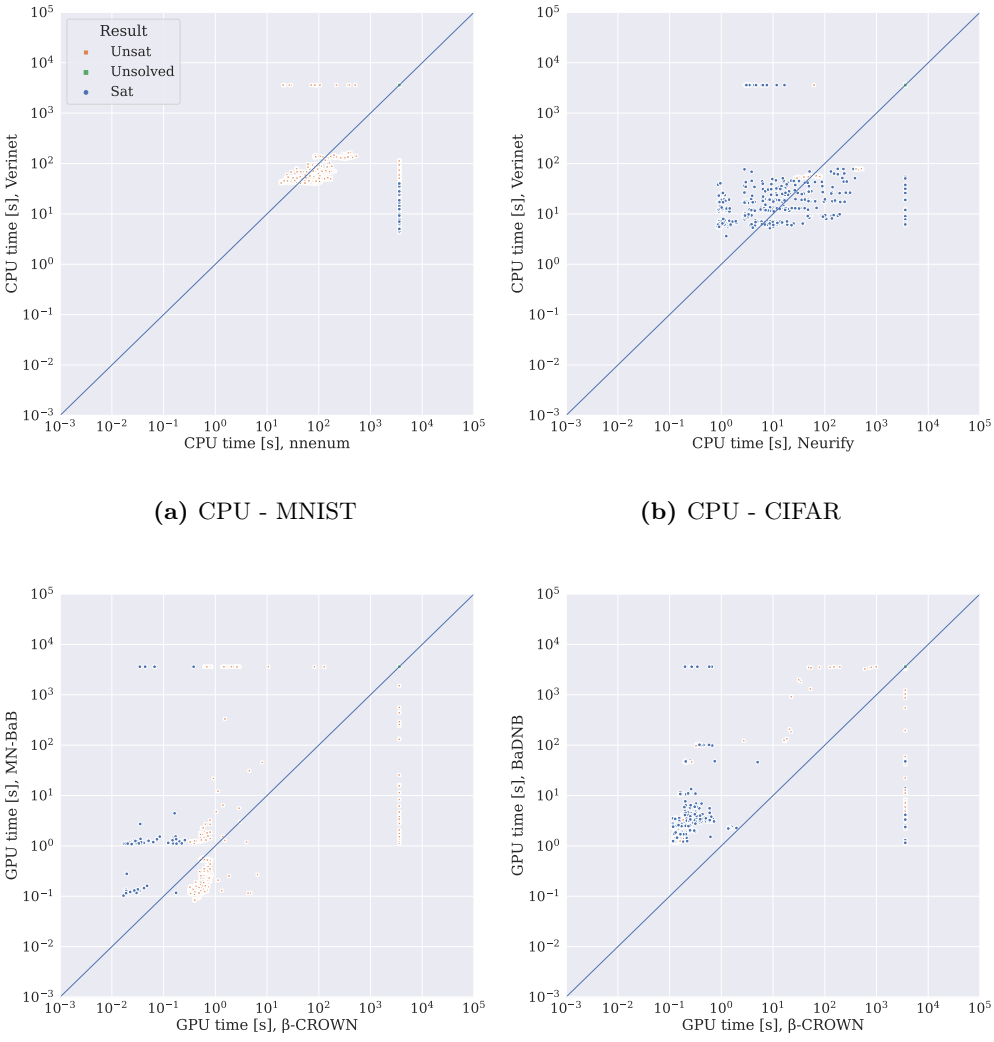
Table 3.6: Performance comparison of CPU-based verification algorithms in terms of the number of solved instances, absolute marginal contribution (MC), absolute Shapley value (ϕ_{abs}) and CPU running time averaged per problem instance, computed for each category with ϵ set at 0.012.

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

any other method.

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 3.2a and 3.2b show an instance-level comparison of the two best-performing algorithms (in terms of relative Shapley value) in the ReLU category for each dataset.



(a) CPU - MNIST

(b) CPU - CIFAR

(c) GPU - MNIST

(d) GPU - CIFAR

Figure 3.2: 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).

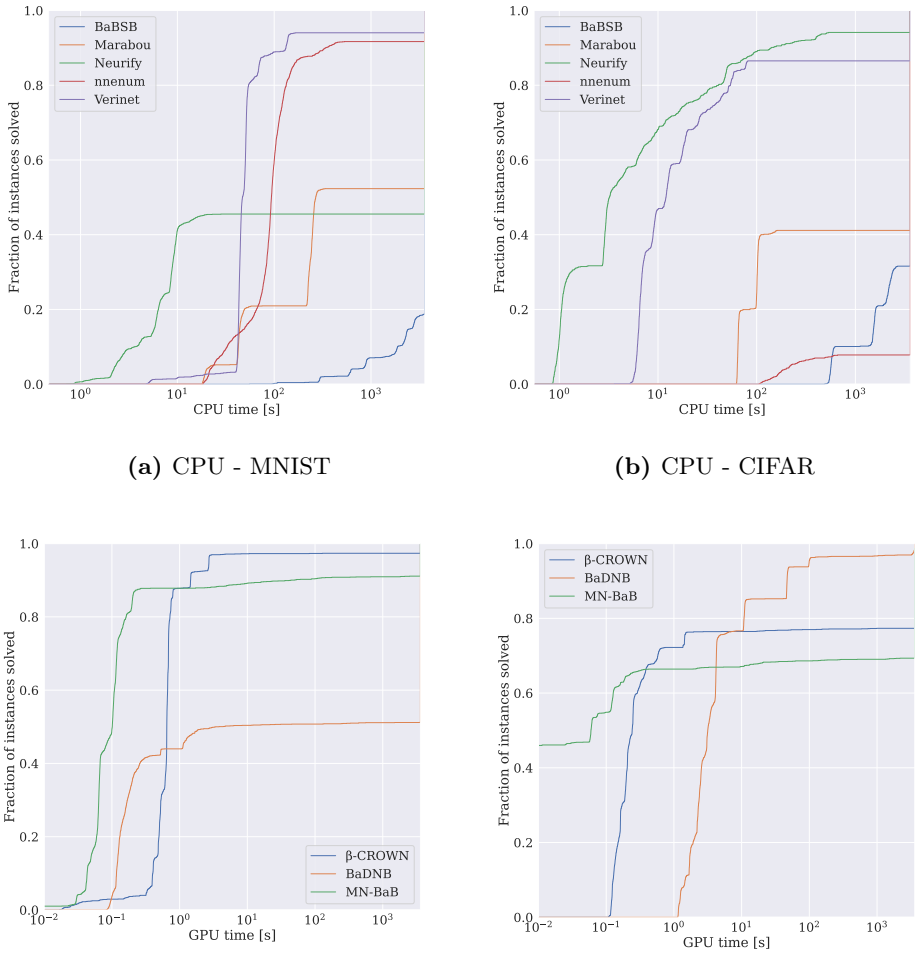
In Figure 3.2a, we see that on MNIST networks, both VeriNet and nnenum solved instances that the other one, in turn, could not solve within the given time budget. Concretely, when considering a parallel portfolio containing both algorithms (see Section 2.4), the number of solved instances slightly increases to 1 817 out of 2 500 (*vs* 1 799 solved by VeriNet and 1 754 solved by nnenum alone), while supplied with similar CPU resources (*i.e.*, 1 800 CPU seconds per verifier, adding up to the same combined maximum running time as running a single verifier with 3 600 CPU seconds).

On CIFAR instances, we found Neurify and VeriNet to also have distinct strengths over each other. This is shown in Figure 3.2b, 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 3.3a 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 3.3b 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.



(a) CPU - MNIST

(b) CPU - CIFAR

(c) GPU - MNIST

(d) GPU - CIFAR

Figure 3.3: 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.

3.4.2 GPU-Based Methods

Table 3.7 summarises the results from our experiments using GPU-based verification algorithms. On ReLU-based MNIST networks, $\alpha\beta$-CROWN outperformed other methods in terms of both the number of solved problem instances as well as the average

Table 3.7: 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

running time. At the same time, the relative Shapley values of $\alpha\beta$-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 3.7 shows that BaDNB outperformed both MN-BaB and $\alpha\beta$-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 $\alpha\beta$-CROWN achieve large relative Shapley values, suggesting their complementarity in an algorithm portfolio.

Figure 3.2c and 3.2d 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 3.2c, one can see that there is a fairly large number of MNIST instances unsolved by $\alpha\beta$-CROWN but solved by MN-BaB as well as the other way

Table 3.8: 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.

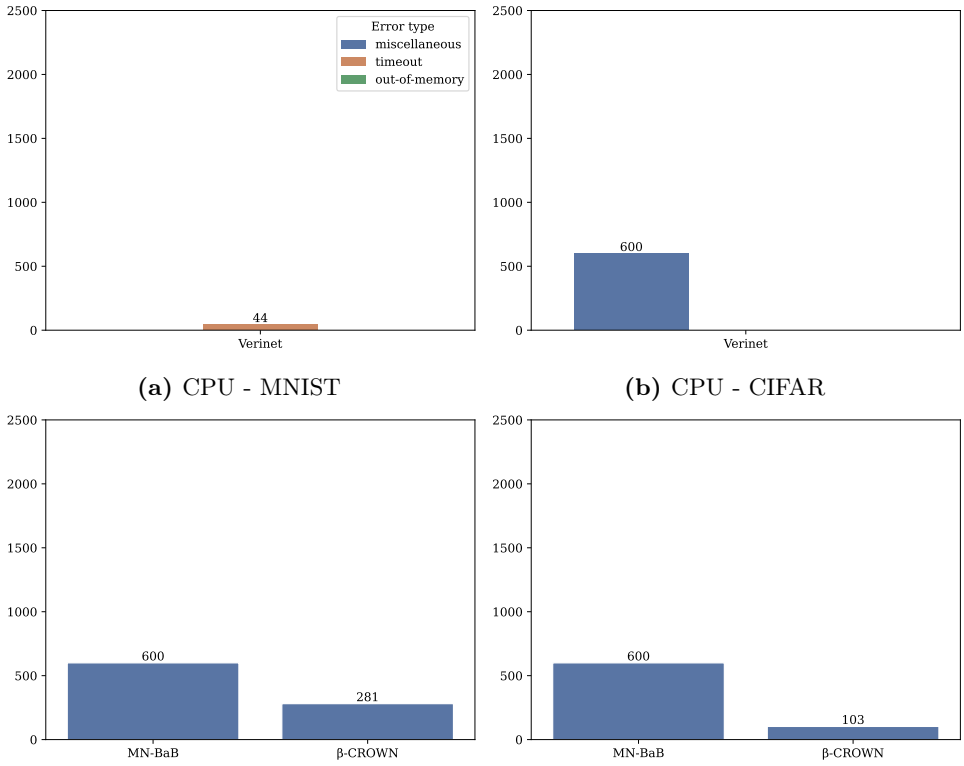
ReLU								
Verifier	MNIST				CIFAR			
	Solved	MC	ϕ_{abs}	Avg. Time [GPU s]	Solved	MC	ϕ_{abs}	Avg. Time [GPU s]
BaDNB	1 188	8	440	1 760	2 332	250	1 066	116
β-CROWN	2 247	0	966	96	1 828	7	693	818
MN-BaB	2 103	18	903	325	1 639	20	604	1 110
ReLU+MaxPool								
Verifier	MNIST				CIFAR			
	Solved	MC	ϕ_{abs}	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

around.

On the other hand, BaDNB and $\alpha\beta$-CROWN seem to have distinctive strengths over each other on CIFAR instances, as can be seen in Figure 3.2d: 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.

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 3.3d. On MNIST instances, MN-BaB solves more instances in less time than $\alpha\beta$-CROWN, although $\alpha\beta$-CROWN solves more instances overall; see also Figure 3.3c.

On MNIST networks containing ReLU activation functions and MaxPooling operations, we again found relatively large Shapley values for both $\alpha\beta$-CROWN and



(a) CPU - MNIST **(b)** CPU - CIFAR

(c) GPU - MNIST **(d)** GPU - CIFAR

Figure 3.4: 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.

MN-BaB, as presented in Table 3.7, indicating their potential complementarity in an algorithm portfolio. However, the relative marginal contribution values indicate that there are no instances unsolved by $\alpha\beta$-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 3.7 further shows results for the Tanh category. We found that instances in this category could effectively only be handled the $\alpha\beta$-CROWN verifier. Concretely, MN-BaB returned an error for the instances in this category; see also Figure 3.4 for additional details.

Lastly, networks containing Sigmoid activation functions can be handled by both

BaDNB and $\alpha\beta$-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 3.7, this does not hold for CIFAR instances, where $\alpha\beta$-CROWN seems to dominate in performance.

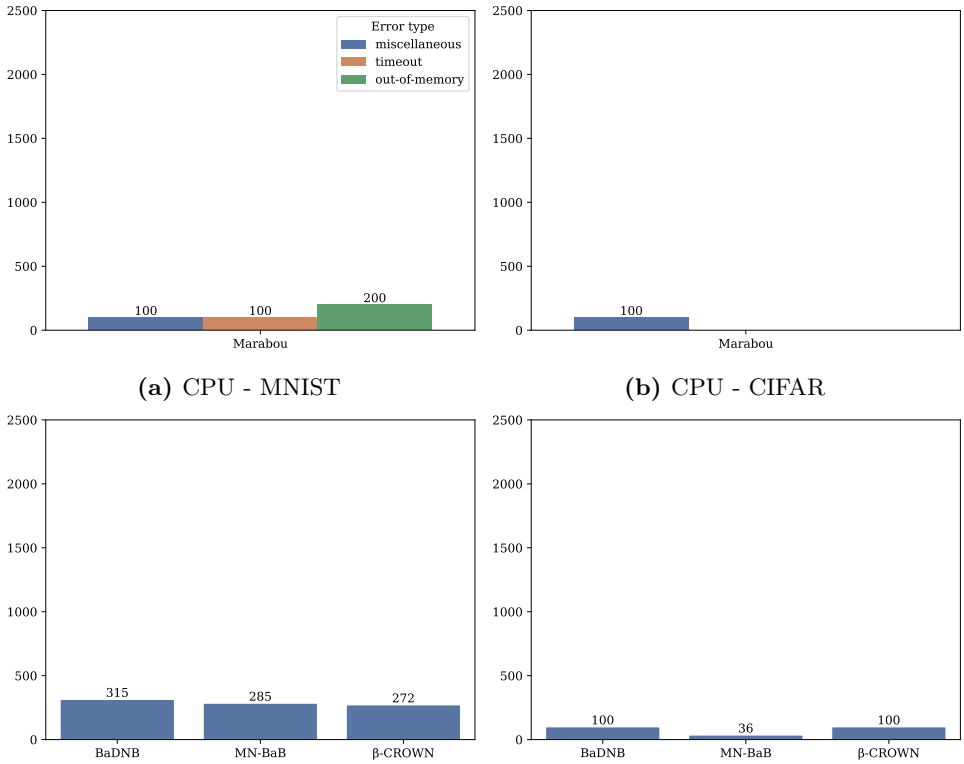
3.4.3 Error Analysis

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 Figure 3.5 and 3.6.

The number of instances solved by each method can be found in Table 3.5 for CPU- and Table 3.7 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 3.7a and 3.7b 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 Figure 3.8.

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 [95].



(a) CPU - MNIST **(b)** CPU - CIFAR

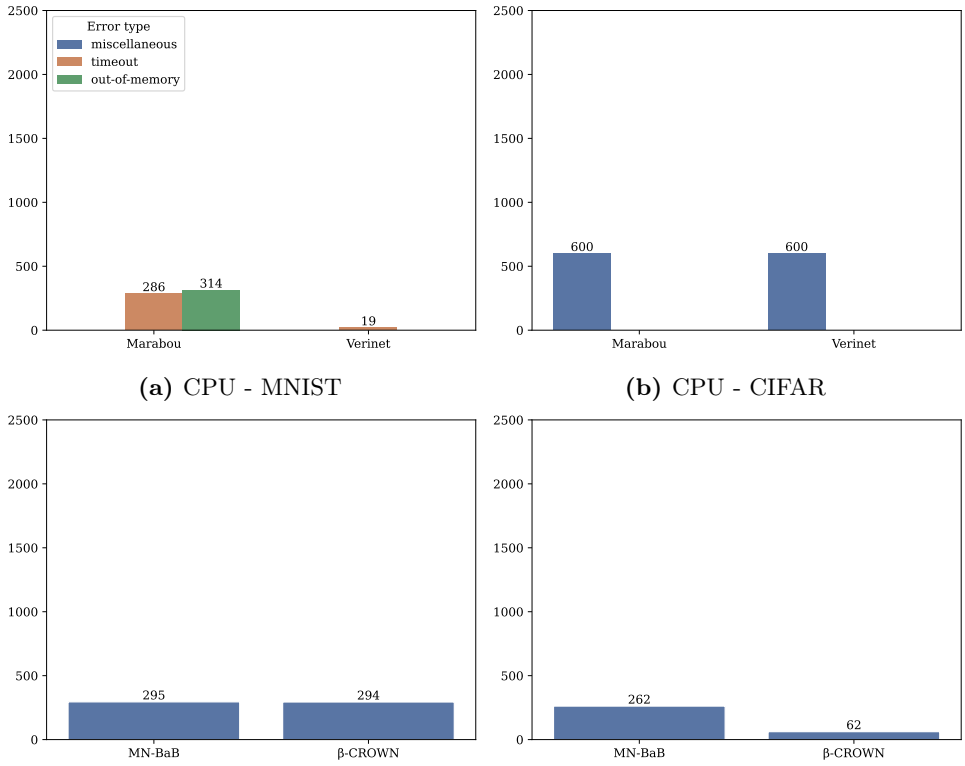
(c) GPU - MNIST **(d)** GPU - CIFAR

Figure 3.5: 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.

On the other hand, GPU-based verifiers show greater support for the considered networks than CPU-based methods. As seen in Figure 3.7c, 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 3.7d. However, both MN-BaB and $\alpha\beta$-CROWN returned several errors of which most are undefined.

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.*, [80]).



(a) CPU - MNIST **(b)** CPU - CIFAR

(c) GPU - MNIST **(d)** GPU - CIFAR

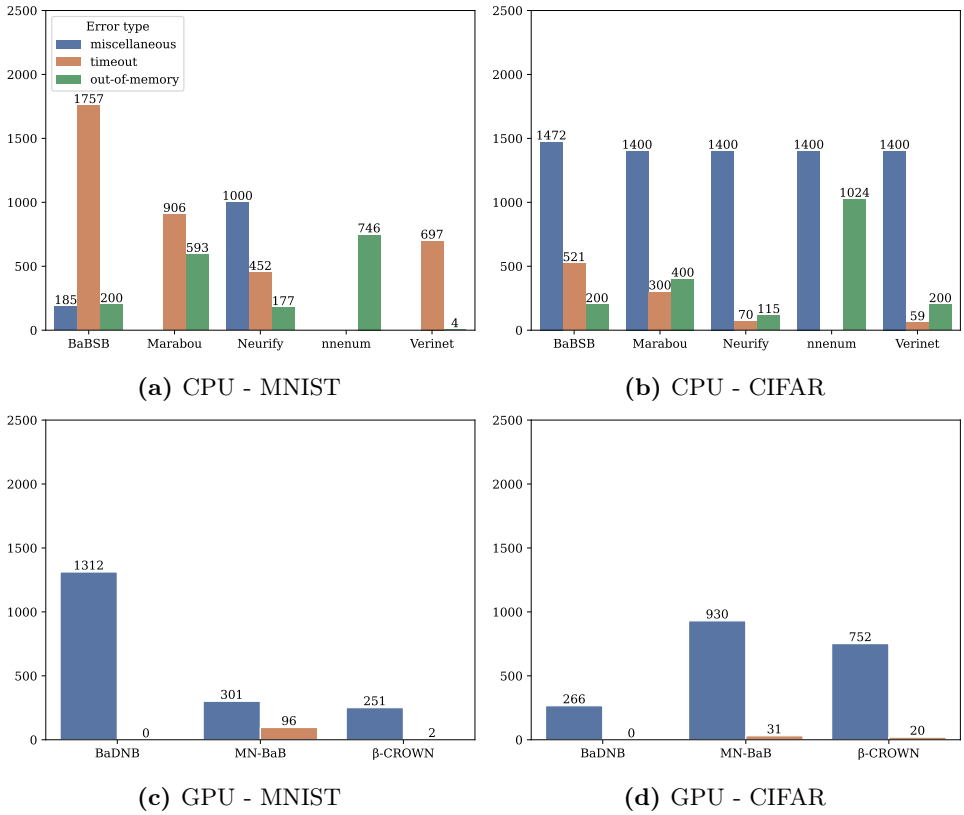
Figure 3.6: Frequency of error types returned by the considered verification algorithms on instances in the Sigmoid category. The total number of instances in this category is 600 for MNIST and 600 for CIFAR.

3.4.4 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 3.9 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. With regard to ReLU-based CIFAR networks, Table 3.9 shows that, overall, Neurify remained the best-performing CPU-based method.

However, we observed substantial differences between small and large values of ϵ in



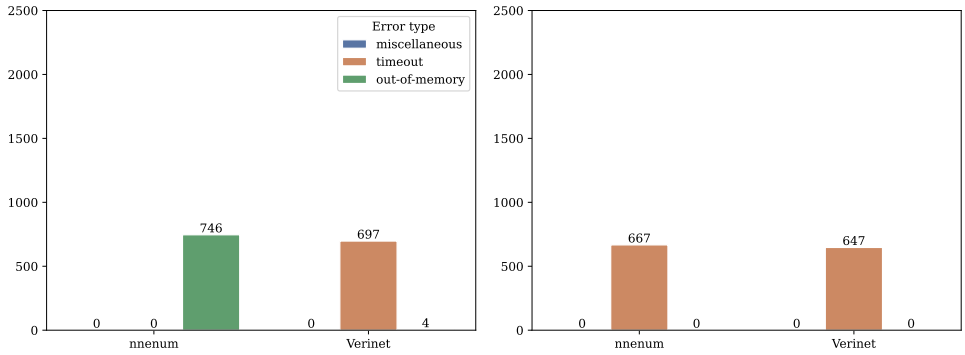
(a) CPU - MNIST **(b)** CPU - CIFAR

(c) GPU - MNIST **(d)** GPU - CIFAR

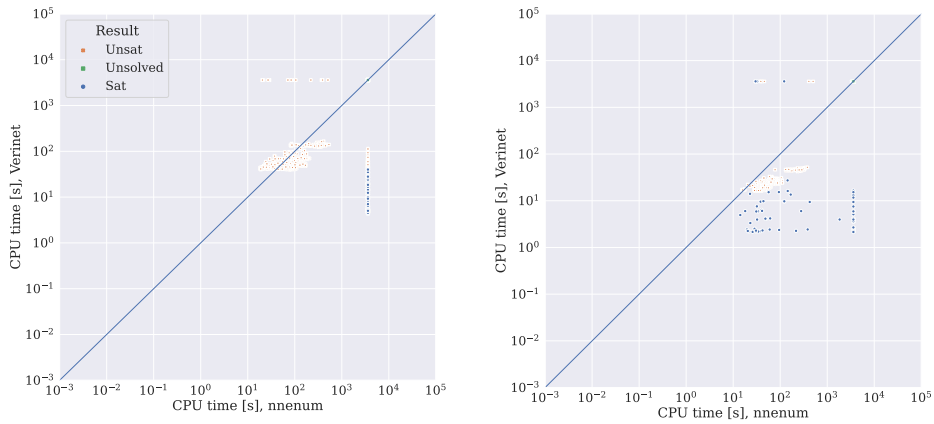
Figure 3.7: Frequency of error types returned by the considered verification algorithms on instances in the ReLU category.

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 3.9c. 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 3.9a and 3.9e. 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 3.9d. 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



(a) Memory limit = 3GB **(b)** Memory limit = 30GB



(c) Memory limit = 3GB **(d)** Memory limit = 30GB

Figure 3.8: 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.

$\epsilon = 0.005$, remains at a stable level. This holds for both solved instances and relative Shapley value as shown in Figure 3.9b and 3.9f.

We performed a similar analysis for GPU-based methods and present results, aggregated over all values of ϵ, in Table 3.11. Among these algorithms, $\alpha\beta$-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

Table 3.9: 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

between small and large values of ϵ on MNIST instances, as shown in Figure 3.10c. For example, when $\epsilon = 0.02$, BaDNB and 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 $\alpha\beta$-CROWN and MN-BaB both reach values close to 0.40 for every value of ϵ; see Figure 3.10e for more details.

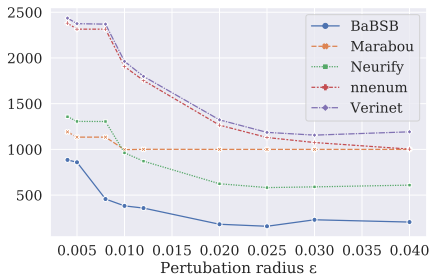
On CIFAR instances, Figure 3.10d indicates that the relative marginal contribution scores are only marginally affected by the chosen perturbation radius. More precisely, BaDNB achieves the largest relative marginal contribution for every value of ϵ, while the relative marginal contributions of $\alpha\beta$-CROWN 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 3.10f.

Table 3.10: 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 $\epsilon \in \{0.004, 0.005, 0.008, 0.01, 0.012, 0.02, 0.025, 0.03, 0.04\}$.

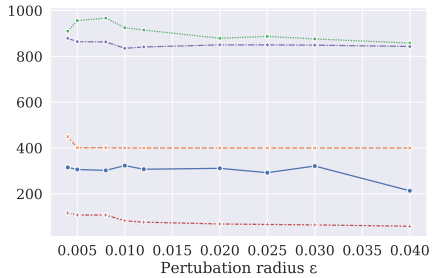
ReLU								
Verifier	MNIST				CIFAR			
	Solved	MC	ϕ_{abs}	Avg. Time [CPU s]	Solved	MC	ϕ_{abs}	Avg. Time [CPU s]
BaBSB	3 716	103	1 062	3 223	2 690	0	759	2 964
Marabou	9 457	784	3 309	1 721	3 651	8	1 078	2 145
Neurify	8 206	212	2 418	1 899	8 173	1 059	3 662	289
nnenum	15 144	288	5 093	543	744	61	268	3 315
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

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 3.11. In each case, we found that one method 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 Figure 3.12.

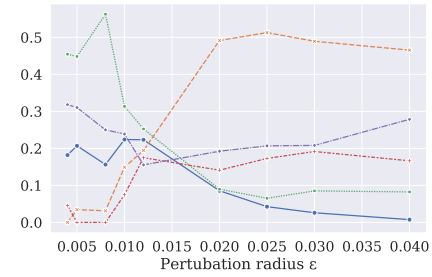
Overall, this clearly demonstrates that our observation of performance complementarity between verification algorithms holds for a broad range of perturbation radii.



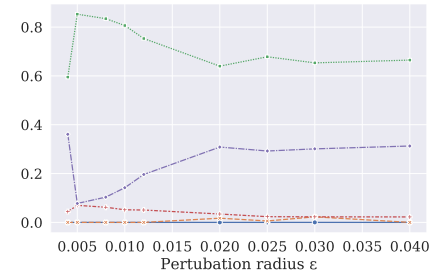
(a) MNIST - Instances solved



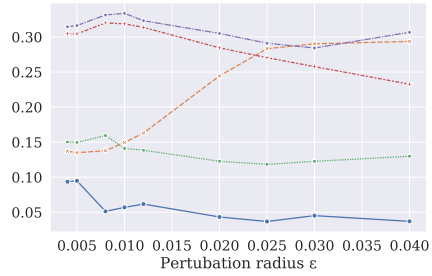
(b) CIFAR - Instances solved



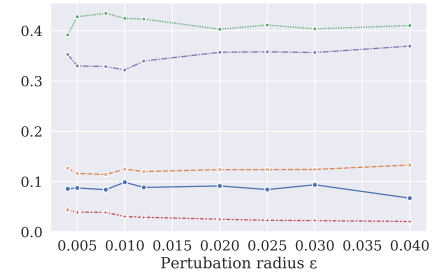
(c) MNIST - RMC



(d) CIFAR - RMC



(e) MNIST - Rel. Shapley Value



(f) CIFAR - Rel. Shapley Value

Figure 3.9: Performance of CPU-based verifiers for different values of ϵ in the ReLU category.

3.4.5 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

Table 3.11: 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

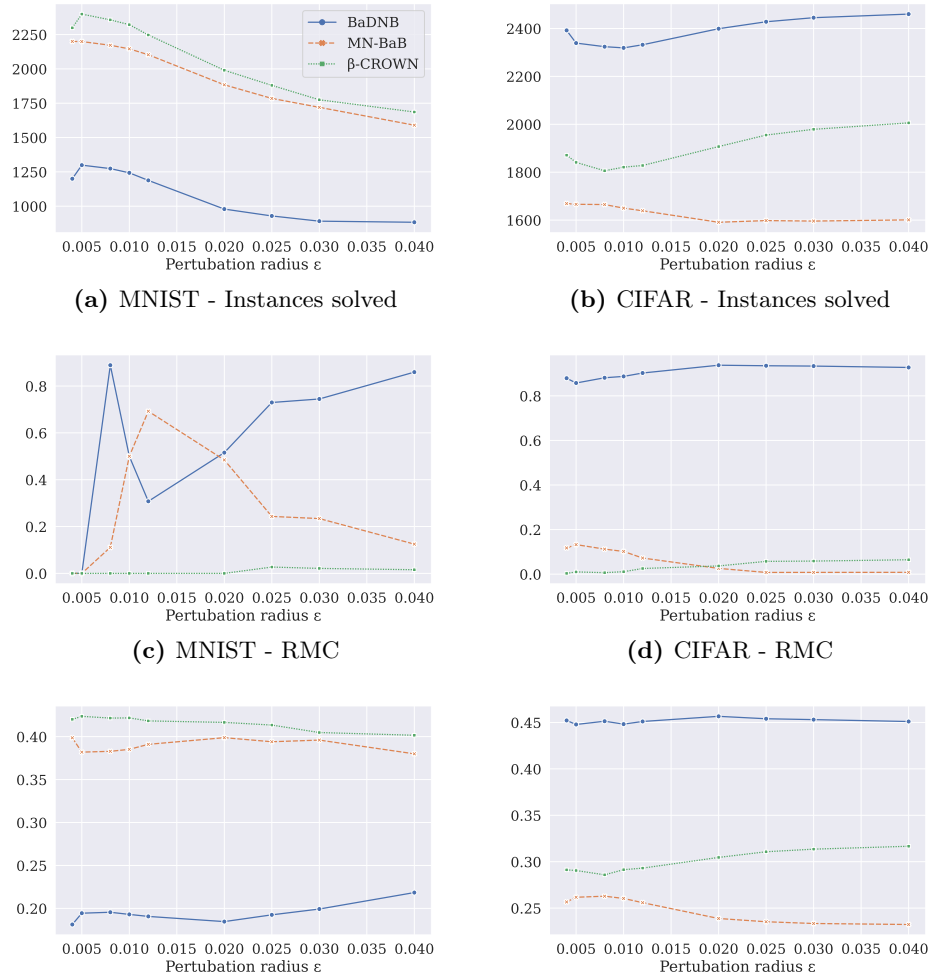
computing platform. To this end, we investigated the price difference between Amazon EC2 CPU instances comparable to the resources allocated in this study.[4] Notice that this hardware is not the exact hardware used in our experiments but is being used here as a substitute for calculating the cost of 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.

[4]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.

Table 3.12: 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

Results from this analysis can be found in Table 3.13 for $\epsilon = 0.012$ and Table 3.14 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 3.14 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



(a) MNIST - Instances solved

(b) CIFAR - Instances solved

(c) MNIST - RMC

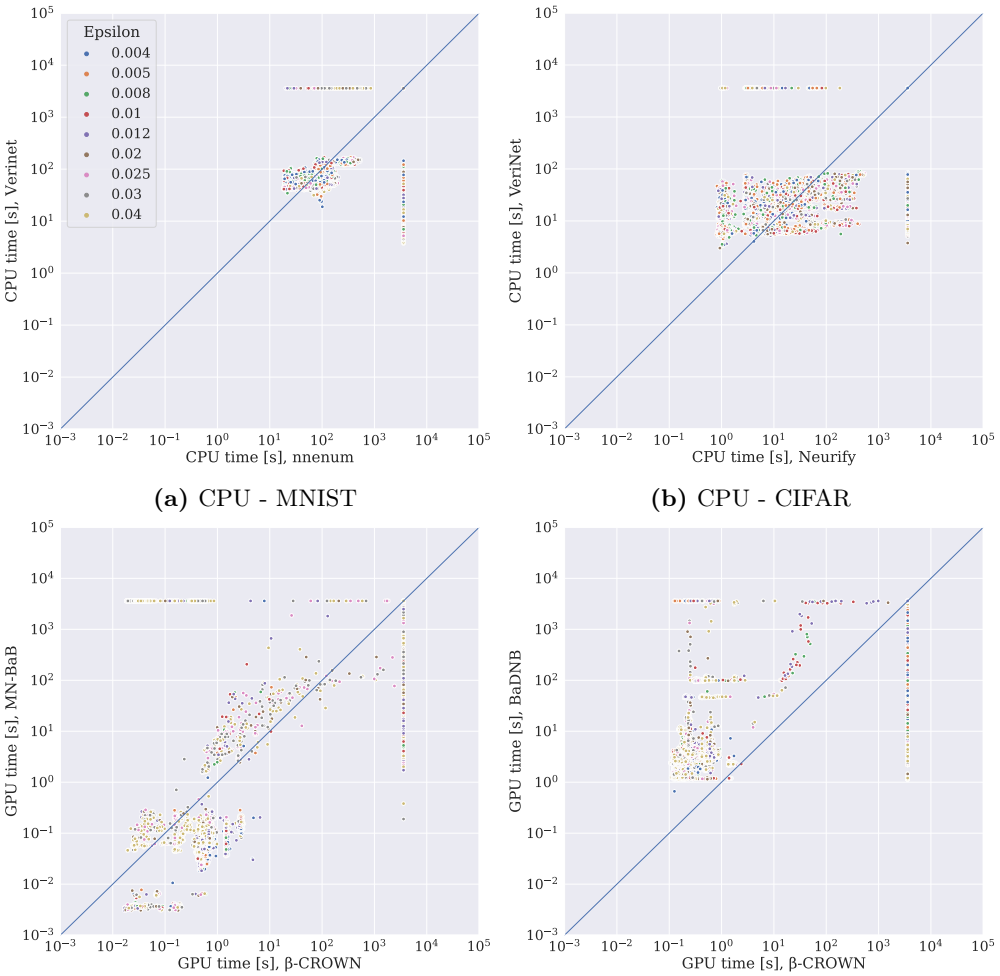
(d) CIFAR - RMC

(e) MNIST - Rel. Shapley value

(f) CIFAR - Rel. Shapley value

Figure 3.10: Performance of GPU-based verifiers for different values of ϵ in the ReLU category.

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.



(a) CPU - MNIST

(b) CPU - CIFAR

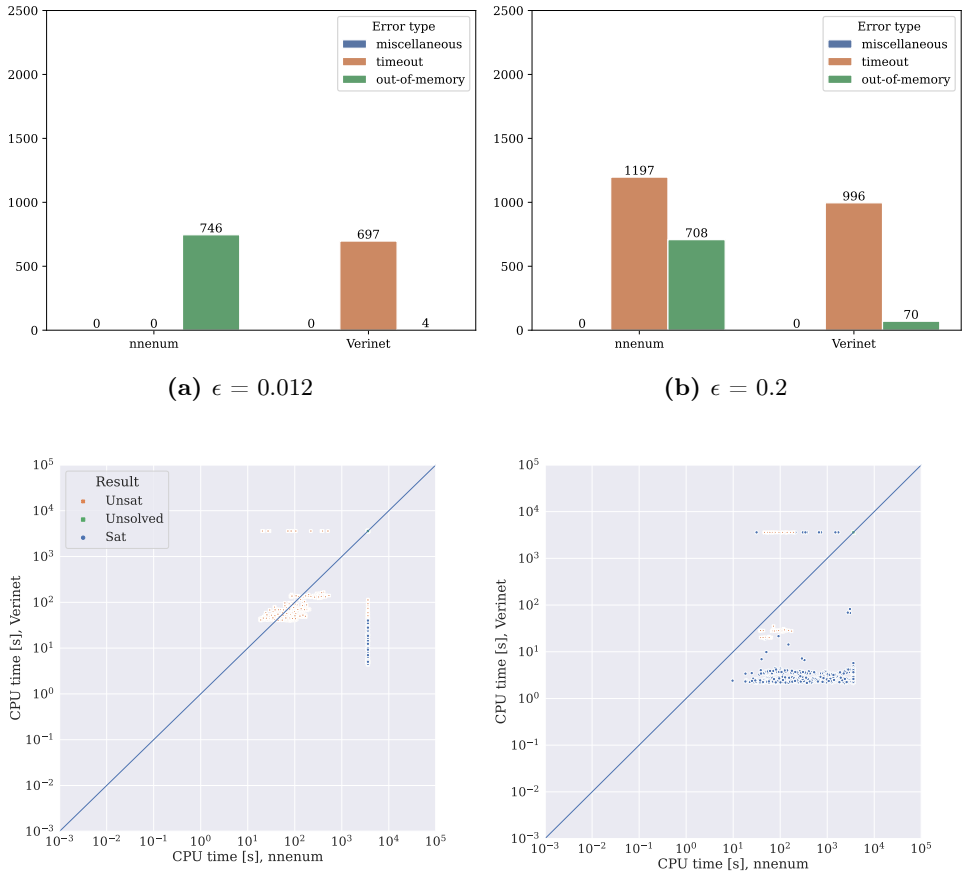
(c) GPU - MNIST

(d) GPU - CIFAR

Figure 3.11: 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.

3.4.6 Analysis of *unsat* Instances

To gain further insights, we performed an analysis of *unsat* (*i.e.*, robust) instances; see Table 3.2 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



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

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

Figure 3.12: 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) $\epsilon = 0.02$.

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 3.15 shows result from this analysis for $\epsilon = 0.012$ while Table 3.16 shows

Table 3.13: 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

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

Table 3.14: 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

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

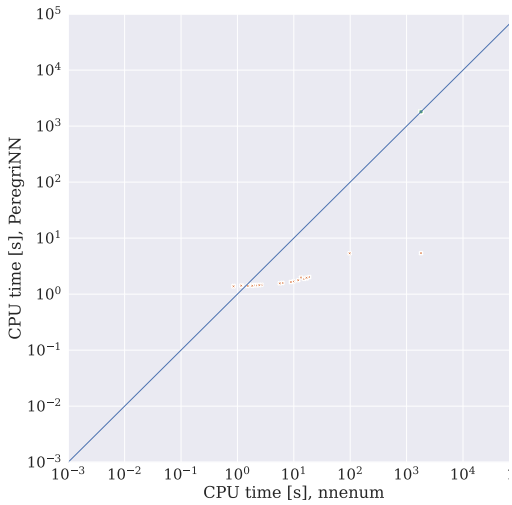
instances across all categories. Specifically, we found that for the broader set of ϵ values, the *RMC* and Shapley value of Marabou improve substantially, while those for VeriNet strongly deteriorate. This indicates that on *unsat* instances, Marabou can solve a large fraction of instances unsolved 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 $\alpha\beta$-CROWN, which also show strong complementarity in the ReLU category.

3.4.7 Analysis of the 2022 VNN Competition Results

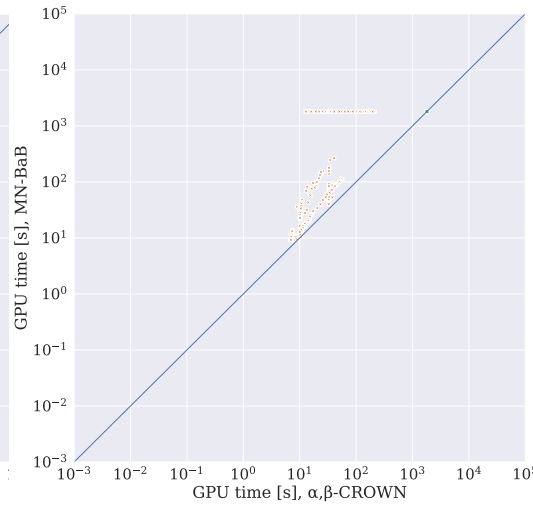
To see if and to what extent our observations hold for a larger set of verifiers as well as different benchmarks, we analysed the results of the 2022 edition of the VNN competition. We refer to the accompanying report [87] for more information about the participating tools, benchmarks and further technical 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 3.17 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 3.13c, there are several instances solved by one of the verifiers but unsolved by the other.

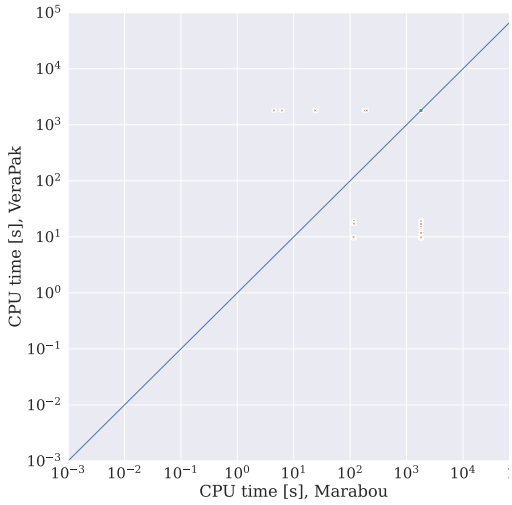
In the FC category, Marabou, nnenum and PerigiNN achieved a similar relative Shapley value of 0.24, again highlighting the complementarity between these algorithms. Given the similar relative Shapley values, we resort to the relative marginal contribution to determine the two best-performing methods 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



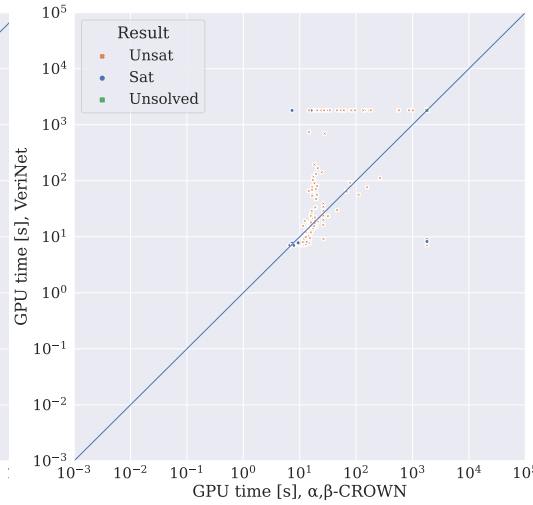
(a) CPU - Complex



(b) GPU - Complex



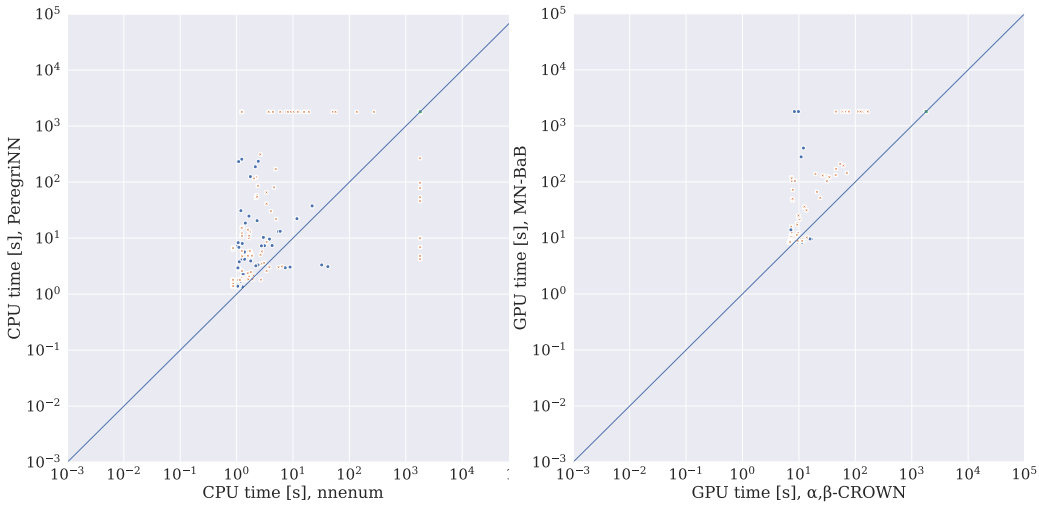
(c) CPU - CNN+ResNet



(d) GPU - CNN+ResNet

Figure 3.13: Performance comparison of the two top-performing verification methods (in terms of relative Shapley value) in each category from the 2022 VNN Competition. 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). (Part 1 of 2)

performance on an instance level, as shown in Figure 3.14a. As can be observed, instances spread out widely around the equal performance line of the plot, with many



(a) CPU - FC **(b)** GPU - FC

Figure 3.14: Performance comparison of the two top-performing verification methods (in terms of relative Shapley value) in each category from the 2022 VNN Competition. 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). (Part 2 of 2)

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 3.13a 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 3.18. Surprisingly, for GPU-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 category, although relative Shapley values indicate complementary (for similar reasons as those outlined above).

This reflected in Figure 3.13b, Figure 3.13d and Figure 3.14b. 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 3.19 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.

3.5 Conclusions and Future Work

In this chapter, we sought to answer the question of what constitutes the state of the art in neural network verification and, thus, address RQ1 of this thesis. To this end, 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. Thereby, the state of the art in neural network verification cannot be described by a single algorithm but rather several algorithms that contribute to varying degrees with their own strengths. As we have demonstrated, this complementarity can be exploited by combining individual verifiers into parallel portfolios. At the same time, automated portfolio construction comes with its own challenges, 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 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 problem. 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. This line of research would also enable empirical performance modelling. An empirical performance model is a model that predicts the performance, *e.g.*, the running time, of algorithms on previously unseen input, including previously unseen problem instances. Finally, it would be interesting to expand this analysis to other datasets and machine learning tasks beyond supervised image classification.

Table 3.15: 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 3.16: 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

Table 3.17: 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 3.18: 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

Table 3.19: 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