Dynamic Algorithm Termination for Branch-and-Bound-based Neural Network Verification

Konstantin Kaulen[1], **Matthias König**[2], **Holger H. Hoos**[1,2]
[1] Chair for AI Methodology, RWTH Aachen University, Germany
[2] LIACS, Leiden University, The Netherlands
kaulen@aim.rwth-aachen.de, h.m.t.konig@liacs.leidenuniv.nl, hh@aim.rwth-aachen.de

Abstract

With the rising use of neural networks across various application domains, it becomes increasingly important to ensure that they do not exhibit dangerous or undesired behaviour. In light of this, several neural network robustness verification algorithms have been developed, among which methods based on Branch and Bound (BaB) constitute the current state of the art. However, these algorithms still require immense computational resources. In this work, we seek to reduce this cost by leveraging running time prediction techniques, thereby allowing for more efficient resource allocation and use.

Towards this end, we present a novel method that dynamically predicts whether a verification instance can be solved in the remaining time budget available to the verification algorithm. We introduce features describing BaB-based verification instances and use these to construct running time, and more specifically, timeout prediction models. We leverage these models to terminate runs on instances early in the verification process that would otherwise result in a timeout. Overall, using our method, we were able to reduce the total running time by 64% on average compared to the standard verification procedure, while certifying a comparable number of instances.

Code — github.com/ADA-research/BaB_DynTerm
Extended version — ada.liacs.nl/papers/KauEtAl25.pdf

Introduction

In recent years, there has been a surge in the application of deep neural networks across various safety-critical domains and usage scenarios, ranging from facial recognition systems embedded in mobile phones to unmanned aircraft manoeuvre advisory systems (see, *e.g.*, Julian, Kochenderfer, and Owen 2019). Concurrently, it became evident that neural networks sometimes show unintended and potentially dangerous behaviour; *e.g.*, it has been shown that neural networks are vulnerable to *adversarial attacks*, where slight input modifications can lead to seriously misleading outputs (Szegedy et al. 2014). Generally, this lack of robustness of neural networks necessitates the development of methods to formally analyse their behaviour, using *formal verification* techniques. These techniques assess whether a given network adheres to specific input-output properties; otherwise, they produce a counter-example that violates the property. In the context of a classifier, a prominently studied property asserts that inputs within the l_∞ norm ball of a specified input x with radius ϵ should be classified into the same class as x (Goodfellow, Shlens, and Szegedy 2015).

Despite substantial advances in recent years, neural network verification remains computationally challenging, with even basic properties known to be NP-complete (Katz et al. 2017). State-of-the-art verification algorithms rely on sophisticated solvers using the branch and bound method (Bunel et al. 2020), requiring significant computational resources and time, even for simple networks (see, *e.g.*, König et al. 2024). Moreover, in order to reason about the robustness of a network over the distribution of possible inputs (*e.g.*, using its *verified accuracy*), a sizeable number of verification queries have to be carried for a given network; this circumstance further contributes to the substantial computational burden.

We propose a novel approach enabling the efficient allocation of compute resources during the verification procedure. Specifically, we introduce a method to classify verification instances as solvable or unsolvable within a predefined time budget based on cheaply computable features. Furthermore, we operationalise these predictions to terminate the verification procedure early for instances where a solution can not be obtained within the given time budget. Thereby, we avoid spending compute resources on attempting to verify instances that ultimately do not inform us about the robustness of the network. We evaluated our approach on a broad set of state-of-the-art verification algorithms and benchmarks, including benchmarks from recent editions of the International Verification of Neural Networks (VNN) competition (Brix et al. 2023; Müller et al. 2022a; Bak, Liu, and Johnson 2021), and show that we can reliably terminate verification runs for instances that are unsolvable within a given cutoff time without solving considerably fewer instances overall. In summary, our contributions are as follows:

- We present, for the first time, features of branch-and-bound-based neural network verification instances that enable predictions about their solvability within a given time budget;

- we introduce a novel method based on those features that reliably identifies instances that cannot be solved within a given time budget;
- we evaluate our method on a broad set of benchmarks and across multiple verification tools;
- we show how this approach can be leveraged to terminate unsolvable instances early in the verification process, leading to savings of 64% in terms of running time on average with a comparable number of solved instances relative to the current state-of-the-art approach.

Background

Neural network verification refers to the task of proving that a given input-output property *holds* or is *violated* for a given neural network. The most commonly studied properties in the literature are local l_∞ robustness properties in the context of image classification tasks (see, *e.g.*, König et al. 2024; Bak, Liu, and Johnson 2021).

Verifying even simple properties is known to be an NP-complete task (Katz et al. 2017). Therefore, the problem is often solved by optimising lower and upper bounds for each neuron in the classification layer (Wang et al. 2021; De Palma et al. 2021c). The extremes of these bounds are referred to as the *global* bounds and are used to decide a given instance of the robustness verification problem. Verification algorithms are either *complete* or *incomplete*. While incomplete methods are not guaranteed to find a solution to a given input query, they are typically *sound*. Additionally, complete methods are not only sound but always return a definitive result given sufficient computational resources. In the following, we will provide further details of the algorithmic approaches to neural network robustness verification, both incomplete and complete, as those form the basis for our choice of features that aim to describe the hardness of verification instances.

Incomplete Neural Network Verification

The most efficient and scalable incomplete verification approaches approximate the concrete output bounds of a given network by applying convex relaxations to its non-linear activation functions (Zhang et al. 2018; Wong and Kolter 2018). Therefore, in the case of ReLU activation functions, each ReLU neuron whose inputs span over both the positive and negative domain must be relaxed. These neurons are referred to as being *unstable*.

To find a linear bounding function of a network with respect to its input, symbolic bounding equations of the last layer can be back-propagated, recursively replacing each input variable with a bounded formulation depending on the previous layer, until the input is reached (see, *e.g.*, Xu et al. 2021; Zhang et al. 2018). These techniques are referred to as *bound propagation* methods.

Finally, adversarial attacks can also be considered an incomplete verification approach, as the existence of an adversarial example proves the violation of the given property (Goodfellow, Shlens, and Szegedy 2015).

Complete Verification via Branch and Bound

In recent years, it has been proposed to use the well-known *branch and bound* (BaB) method, which employs the previously introduced incomplete bounding techniques in a complete verification framework (De Palma et al. 2021c; Wang et al. 2021; Henriksen and Lomuscio 2020; Bunel et al. 2020). The BaB algorithm starts by applying an incomplete verification technique to derive an initial lower and upper bound. If these do not suffice to solve the instance, the original problem is split into equivalent sub-problems. For ReLU networks, this means choosing an unstable neuron and splitting it into its two linear sub-parts (Bunel et al. 2020). Alternatively, splits can also be performed on the input domain of the ReLU activation (Wang et al. 2018). The resulting branches are then easier to bound by the incomplete methods, as fewer non-linearities need to be approximated. If a counter-example is found in any of the branches, or if the global lower bound is positive for both branches, the verification problem is solved. Otherwise, the algorithm is applied recursively to both sub-problems. Eventually, when all unstable ReLU neurons have been split, the verification task boils down to a linear programming problem that can be solved exactly; hence, the algorithm is complete for ReLU networks. The branch and bound framework leaves open many design choices, *e.g.,* the bounding technique and the branching heuristic that selects the next neuron to split. In the following, we present the branch and bound variants relevant to our work.

Branch-and-Bound-based Algorithms

Currently, BaB-based algorithms represent the state of the art in neural network verification; see, *e.g.,* Müller et al. (2022a) and König et al. (2024) for extensive performance evaluations of a broad range of verifiers. Specifically, these BaB-based algorithms are $\alpha\beta$-CROWN (Zhang et al. 2018; Xu et al. 2021; Wang et al. 2021), Oval (De Palma et al. 2021c,b,a), VeriNet (Henriksen and Lomuscio 2020) and MN-BaB (Ferrari et al. 2022). The repository of MN-BaB is not maintained and in an experimental state that includes only a few usable configurations tailored to specific benchmarks; therefore, we did not further consider MN-BaB.

$\alpha\beta$-CROWN is mainly built upon the incomplete bound propagation method CROWN (Zhang et al. 2018). Its extensions, α- and β-CROWN, enable tighter bounds by optimising the slope of lower bounds and by incorporating split constraints into the bounding process, respectively (Wang et al. 2021; Xu et al. 2021). Furthermore, it is possible to use the α-CROWN intermediate layer bounds in combination with the *Gurobi* solver to solve the MIP formulation of the verification problem (Tjeng, Xiao, and Tedrake 2019).

Oval works in a similar way as $\alpha\beta$-CROWN; specifically, it implements CROWN and its improvements, α- and β-CROWN. In addition, Oval includes solvers that operate on the dual of a tighter ReLU relaxation (Anderson et al. 2020), which may produce tighter bounds than $\alpha\beta$-CROWN (De Palma et al. 2021a,b).

VeriNet employs an error-based bound propagation method that only computes the network's lower bound, while representing upper bounds through a concrete error value per relaxation (Henriksen and Lomuscio 2020). The resulting symbolic bounds are used as inputs to the *XPress* LP Solver in combination with the input constraints to check whether the robustness property holds. However, if a satisfying variable assignment is returned, a gradient-based local search around this assignment is initiated to distinguish spurious from valid counter-examples. If this search remains unsuccessful, the current problem is split, and resulting split constraints are added to the LP sub-problem formulations.

Oval and $\alpha\beta$-CROWN rely on GPUs that execute the BaB algorithm in batches, while VeriNet distributes its workload among CPU workers; these can use GPUs to accelerate tensor operations. Lastly, all verifiers we considered attempt to generate adversarial examples to quickly calculate an upper bound of the given verification problem.

Running Time Prediction

In the context of other NP-complete problems, such as SAT, MIP or TSP, it has been shown that running time can vary drastically depending on the instance and algorithm used (see, *e.g.*, Hutter et al. 2014). Recently, this phenomenon has also been observed for neural network verification (König et al. 2024). While overall, this behaviour is not well understood, it is possible to predict running times of previously unseen SAT, MIP and TSP problem instances reasonably accurately, based on cheaply computable instance features by fitting a statistical model on the running time of a given algorithm (Hutter et al. 2014). Running time prediction has various applications that seek to minimise costs and improve the efficiency of machine learning systems; those range from algorithm selection to scheduling (Kerschke et al. 2019). Furthermore, running time prediction provides insights into the relationship between instance characteristics and algorithm running time and, thereby, informs us about instance complexity. To the best of our knowledge, the present study is the first to employ running time prediction techniques in the context of neural network verification.

Method

As previously explained, solving the neural network verification task is computationally challenging, especially when attempting to verify several instances in order to reason about the robustness of a network over the distribution of possible inputs. In addition, it is not known what makes certain instances harder to solve than others. Therefore, it cannot be decided *a priori* whether an instance could be solved successfully within a given time budget, potentially leading to ineffective resource allocation; *i.e.*, allocating compute time to unsolvable instances. In light of this, we leverage running time prediction techniques to classify instances as solvable or unsolvable within a given time budget. This allows us to greatly accelerate the verification procedure, by ensuring that resources are allocated towards solvable instances.

Problem Formulation

As we are interested in the task of terminating instances that cannot be solved within a given time budget, we consider a binary classification problem. Our goal is to reduce the computational burden demanded by the verification procedure, ensuring the most effective use of resources by not spending the full budget on unsolvable instances. In addition, we need to avoid classifying solvable instances as unsolvable; otherwise, the number of certified instances would be reduced, which might lead to inaccurate conclusions about the robustness of a given neural network.

Dynamic Algorithm Termination for BaB-based Neural Network Verification

To predict the running time of BaB-based neural network verification algorithms, we utilise cheaply computable features that, in part, relate directly to the internal operations of the given verifier.

We distinguish between *static* and *dynamic* features, where the former are computed only once and do not change during the solving process. Examples of static features include the lower bound obtained by an incomplete verification method at the beginning of the verification process. Conversely, dynamic features aim to capture the dynamically changing state of the verification algorithm at any point in time; examples include the current number of nodes in the BaB tree and the current global bounds. Generally, static features reflect the inherent complexity of the verification instance, while dynamic features capture the progress made thus far in solving the query. A detailed discussion of the features we have developed is provided later in this section.

To best leverage the evolving nature of our dynamic features, we propose a novel method that dynamically terminates verification queries once a classification model determines that the given instance will not be solved in the remaining time budget. We give a schematic overview of our method in Algorithm 1. The procedure is parameterised by the frequency t_{freq} at which the current progress of the verification process is assessed, and by the maximum allotted running time per instance, t_{cutoff}. We refer to the points in time at which the verification query is examined as a *checkpoint*. For each checkpoint, we train a classifier $\mathcal{C}_t$ with t denoting the time of the checkpoint. $\mathcal{C}_t$ is trained on the feature values of the verification instances in the training set at time t along with their corresponding label indicating whether the instances were solved within t_{cutoff} seconds. Moreover, we also trained the classifier on the verification instances from our training set that were successfully solved before the current checkpoint with their feature values when the verification process was completed; thereby the classifier can learn which feature values define a completed instance.

In addition, our proposed method is configurable via a confidence parameter θ, which defines the threshold that the prediction value for the positive class must exceed such that an instance is labelled accordingly. The incorporation of this parameter ensures that a user can choose whether the algorithm should stop potentially unsolvable instances as soon as possible ($\theta = 0.5$) or whether, in case of doubt, more information should be collected. The verification algorithm is then terminated only in case of a highly confident classifier prediction ($\theta = 0.99$). We note that θ can also be understood as a tuning parameter between *exploitation* and *explo-*

Algorithm 1: Dynamic termination for BaB-based neural network verification

Input: Verification instance (x_0, ϵ); maximum per-instance running time t_{cutoff}; dynamic termination frequency t_{freq}; set of classifiers $\mathcal{C} = \{\mathcal{C}_{t_{\text{freq}}}, \mathcal{C}_{2t_{\text{freq}}}, ..., \mathcal{C}_{t_{\text{cutoff}}}\}$; confidence parameter θ; verification algorithm VERIFY((x, ϵ), t_{freq}) that pauses after t_{freq} seconds to return the features and result of the instance.

Output: result or *unknown*

```
solved, features ← VERIFY($(x_0, \epsilon)$, $t_{\text{freq}}$)
$t_{\text{elapsed}} \leftarrow t_{\text{freq}}$
while $\neg$ solved $\wedge$ $t_{\text{elapsed}} < t_{\text{cutoff}}$ do
  if $\mathcal{C}_{t_{\text{elapsed}}}$(features) $> \theta$ then
    return unknown
  else
    solved, features ← VERIFY($(x_0, \epsilon)$, $t_{\text{freq}}$)
    $t_{\text{elapsed}} \leftarrow t_{\text{elapsed}} + t_{\text{freq}}$
  end if
end while
return solved
```

ration. Therefore, θ should be chosen according to the user's needs, prioritising either a substantial reduction of the computational burden or a higher number of certified instances.

In summary, our method operates as follows. Given t_{freq}, t_{cutoff}, θ, a verification algorithm, a neural network and a training set of verification instances, we initially collect feature values for each training instance at every checkpoint t by executing the verification algorithm on each query for t_{cutoff} seconds. In addition, we record whether the instance was solved or not. Subsequently, we train a classification model C_t for every checkpoint t on the collected data. During classification, given a verification query, we start by executing the verification algorithm for t_{freq} seconds to collect an initial set of features for the given instance. Thereafter, we employ the classifier for the first checkpoint to predict whether the instance will be solved in the remaining time budget. If the confidence of this prediction exceeds θ, we terminate the verification run for the given instance and record its result as *unknown*; otherwise, we continue the verification process for t_{freq} seconds and update the dynamic instance features accordingly. Next, we query the classification model for the following checkpoint and decide whether to terminate the run. We repeat this process until the verification algorithm solves the instance under consideration or reaches the given cutoff time.

Static Instance Features

To perform running time prediction, we need to define instance features that allow us to make performance predictions for a given algorithm. We begin by introducing our proposed static features.

Prediction margin (Δ). This feature is defined as the difference between the two highest class scores. *i.e.*, given a neural network f with input $x_0 \in \mathcal{X}$ and corresponding correct label $y_0 \in \mathcal{Y}$, we have $\Delta := f_{y_0}(x_0) - \max_{y \in \mathcal{Y} \setminus y_0} f_y(x_0)$, where f_y refers to the output for class y. The prediction margin can be seen as a proxy for the closeness of the input image to the decision boundary. This feature has recently been used in the context of adversarially robust model selection (König, Hoos, and van Rijn 2024).

Initial Incomplete Bound. Each verifier we consider first attempts to solve the verification instance using an incomplete method. We utilised the resulting global upper and lower bounds as features.

Improved Incomplete Bound. If the initial problem bounds do not suffice to solve the problem, Oval and $\alpha\beta$-CROWN follow up with a tighter bounding method to further optimise last layer bounds. The initial and improved bounds give an estimate of how much improvement on the lower bound is realisable through (incomplete) bound optimisation methods. Furthermore, these bounds are the starting point for BaB and, thus, indicate the improvements required during BaB for solving the problem.

Initial Percentage of Safe Constraints. While VeriNet does not employ bound optimisation, the first call to the LP solver with the initial SIP bounds can already determine that some (or all) linear equations are unsatisfiable; these output constraints do then not have to be examined further during BaB. Thus, the percentage of initial safe constraints also provides an indication of the additional computation VeriNet will require subsequently.

Adversarial Attack Margin. Each of the considered verifiers initially carries out an adversarial attack that seeks to minimise the margin between the correct and incorrect classes. If the attack remains unsuccessful, its output can still be utilised to estimate the upper bound of the verification problem. Therefore, we included the adversarial attack margin, *i.e.*, the difference between the two highest scoring classes on the adversarial candidate, as an estimation of the upper bound of the given verification instance.

Number of Unstable Neurons. Lastly, we also included the absolute number of unstable neurons in our feature set. This number does not only indicate how many non-linearities have to be approximated but also bounds the maximum depth of the BaB tree.

Dynamic Instance Features

The *dynamic* features of BaB-based verification instances are subject to change during the BaB process, as they capture the progress made while solving the given problem instance.

Branch Characteristics. We included the number of *visited* branches that are already bounded as well as the total number of branches, also including those that have been created through branch splits but still need to be bounded. We further included the fraction of verified branches; these correspond to the leaves of the BaB tree and do not need to be split further. Once this number reaches a value of 1, the verification system has proven that the property holds.

Current Global Bounds. Furthermore, we included the current global bounds of the BaB tree. When the MIP formulation of the problem was solved by $\alpha\beta$-CROWN, we also recorded the resulting global bounds. This constitutes another way of capturing the progress of the given query, as once any global bound changes its sign, the verification process has been completed.

		$\alpha\beta$-CROWN		VeriNet		Oval	
Benchmark	# Inst.	# Solved	Time [GPU h]	# Solved	Time [GPU h]	# Solved	Time [GPU h]
5 100	960	868	31.44	580	66.65	430	90.77
8 100	947	767	41.32	501	76.64	387	94.69
Conv Big	929	918	1.50	868	11.29	842	15.34
Conv Small	980	979	1.26	931	11.96	958	6.06
ResNet 2B	703	619	15.16	576	22.72	-	-
Marabou	500	193	51.73	176	54.28	187	53.32
Oval21	500	210	50.47	158	58.45	201	52.50
ViT	500	251	41.86	-	-	-	
SRI ResNet A	500	198	51.74	133	62.01	-	-
CIFAR-100	500	361	24.73	279	40.25	-	-
Tiny ImageNet	500	421	14.63	356	29.47	-	-

Table 1: Overview of benchmarks used in our evaluation, along with the number of certified instances and used running time for each included verification tool. The number of instances refers to the correctly classified instances from the first 1000 test set images for the first 5 benchmarks and otherwise to the number of instances generated following the VNN Competition (Brix et al. 2023; Müller et al. 2022a; Bak, Liu, and Johnson 2021) instance selection procedure. All experiments ran with a per-instance timeout of 600 seconds in wall-clock time and GPU acceleration.

Depth of the BaB Tree. One important characteristic of the BaB tree that indicates instance complexity is its current depth, as it indicates how many neuron splits are present in the leaf nodes.

Number of GPU Batches. For Oval and $\alpha\beta$-CROWN, which perform the BaB algorithm in batches on a GPU, we included the number of batches that have been already computed. This feature enables a running time predictor to relate the BaB features to the internal operations of the verifiers.

Batch Computation Time. In addition, we computed the time used for the computation of the last completed batch; this number indicates the computational hardness of the problem instance at hand, also in relation to the execution environment used for running the verifier. If feature collection occurs while a batch is still being processed, we additionally considered the computation time already spent on that batch.

Classification Model

For each checkpoint, we trained a random forest classifier using the *scikit-learn* (Pedregosa et al. 2011) implementation with 200 decision trees and otherwise default hyperparameter settings. It has been shown in the past that random forests perform very well in the context of running time prediction tasks (Hutter et al. 2014). We also experimented with automatic hyperparameter configuration using *auto-sklearn* (Feurer et al. 2015), but did not observe substantial improvements. Before training and classification, all features were standardised, *i.e.*, we removed the mean of each feature and scaled it to have unit variance, using the mean and standard deviation of each feature over the training set.

Experiments

We evaluated our approach on several benchmarks, which we will introduce in the following, along with details on the performance data collection and feature computation process. We provide the source code used to conduct all experiments publicly on GitHub.

Each benchmark was run on a compute cluster node equipped with two Intel Xeon Platinum 8480+ processors with 56 cores and a cache size of 105MB, 2TB of RAM and four NVIDIA H100 GPUs with 80GB of video memory, running Rocky Linux 9.4. Each run utilised 28 CPU cores, one GPU and 448GB of RAM.

Benchmarks

We considered a wide and diverse set of benchmarks taken from the ERAN repository (Müller et al. 2022b; Singh et al. 2019a) and the VNN Competition (Brix et al. 2023; Müller et al. 2022a; Bak, Liu, and Johnson 2021), which have been commonly used by the neural network verification community (see, *e.g.*, König et al. 2024; Singh et al. 2019b).

For our evaluation, we included two usage scenarios. First, we considered an approach aligned with an end-user's needs in assessing the robustness of a neural network. Here, we verified the correctly classified images from the first 1000 test set instances. We also included a competition scenario, where we generated problem instances according to the VNN Competition instance generation protocols.

In the first scenario, we included two convolutional (*Conv Big* and *Conv Small*) and two fully connected networks (*5 100* and *8 100*) trained on the MNIST dataset that were taken from the ERAN repository. For the CIFAR-10 dataset, we considered a small ResNet proposed by Wang et al. (2021) (*ResNet 2B*). We verified the first 1000 test images against l_∞ perturbations with ϵ-values chosen in line with those used in previous studies (König et al. 2024; Wang et al. 2021; Singh et al. 2019b).

For the second scenario, we included benchmarks directly taken from different editions of the VNN competition (Brix et al. 2023; Müller et al. 2022a; Bak, Liu, and

	$\alpha\beta$-CROWN			VeriNet			Oval		
Benchmark	Acc.	TPR	FPR	Acc.	TPR	FPR	Acc.	TPR	FPR
5 100	0.99	0.95	0.00	0.89	0.87	0.04	0.97	0.96	0.00
8 100	0.99	0.99	0.00	0.92	0.91	0.02	0.99	0.99	0.07
Conv Big	0.47	0.43	0.00	0.88	0.74	0.00	0.78	0.75	0.05
Conv Small	0.82	1.00	0.20	0.81	0.39	0.00	0.79	0.09	0.00
ResNet 2B	0.98	0.98	0.00	0.77	0.71	0.00	-	-	-
Marabou	0.99	0.99	0.10	0.93	0.95	0.53	0.96	0.96	0.13
Oval21	0.97	0.98	0.05	0.89	0.88	0.07	0.96	0.95	0.03
ViT	0.99	0.98	0.00	-	-	-	-	-	-
SRI ResNet A	0.99	1.00	0.02	0.91	0.90	0.00	-	-	-
CIFAR-100	0.99	0.99	0.02	0.86	0.78	0.00	-	-	-
Tiny ImageNet	0.99	0.99	0.00	0.90	0.69	0.01	-	-	-

Table 2: Results for timeout prediction with continuous feature collection in terms of accuracy, true positive and false positive rate as averages over five folds. We display results for $\theta = 0.99$, *i.e.*, the confidence threshold that must be reached before an instance is terminated.

Johnson 2021). We employed the instance generation scripts provided in the competitions to generate 500 instances per benchmark that follow specific selection criteria such as correct classification or robustness against adversarial attacks. Concretely, we included the *Marabou*, *Oval21*, *SRI ResNet* and *ViT* benchmarks that consist of networks trained on the CIFAR-10 dataset. If the benchmarks included multiple networks or ϵ value specifications, we chose the configurations that yielded the most timeouts in the VNN competition, *i.e.*, the presumably most challenging problem instances.

Lastly, we included two benchmarks from the VNN Competition that consider the more complex CIFAR-100 and Tiny ImageNet datasets. For both datasets, we chose the medium-size models for our evaluation. With this collection of networks and benchmarks, we ensured to include instances that have been studied extensively in the literature and that are challenging to solve by state-of-the-art verification tools. We give an additional overview of the selected networks along with their source, training method and specific verification property in the supplementary material.

Evaluation Setup

First, we collected all performance data and feature values by running the verification tools and saving the result of the verification query, the consumed running time and the values of the considered instance features during the verification procedure. In Table 1, we report the number of solved instances and the running time for each verification tool and benchmark. Missing values indicate that the benchmarks could not be used with the respective verification tools due to unsupported network architectures.

We then evaluated our method by simulating it on the collected data. Generally, we conducted our method independently for each benchmark and verification system, following a 5-fold cross validation protocol. To ensure that our training and testing sets were representative, we included in each fold the same proportion of verification instances solved before the first checkpoint, after the first checkpoint and unsolved instances; however, we only report metrics on instances that ran beyond the first checkpoint, as otherwise, we would predict timeouts after the instance has already been solved.

We evaluated the performance of our method in terms of *accuracy*, *true positive rate* (TPR) and *false positive rate* (FPR). The TPR reflects the fraction of correctly classified timeouts out of all unsolved instances while the FPR indicates the fraction of solved instances wrongly classified as timeouts out of all solved instances. On the convolutional networks for $\alpha\beta$-CROWN, some folds did not include true negatives or true positives. If these folds were used as the holdout set, we excluded them when computing the average TPR and FPR. In addition, we also compared our method to the standard verification procedure in terms of the overall number of solved instances (including those completed before the first checkpoint) and the required running time.

We conducted additional experiments, in which we examined and validated the relevance of our proposed instance features through a Shapley value analysis and a feature ablation study. Moreover, we conducted experiments for different choices of θ and assessed whether our features could also be used to predict running times in a regression scenario. We present the results of these experiments in the supplementary material.

Hyperparameter Configuration. To run the verification algorithms, we used the configurations provided by the respective authors for their entries in the VNN Competitions. We chose a maximum running time of 600 seconds in wall-clock time per instance ($t_{\text{cutoff}} = 600s$) and predicted whether the instance will be solved within the remaining time budget every 10 seconds ($t_{\text{freq}} = 10s$). Lastly, we set the decision threshold θ to 0.99 to ensure that our method solves as many instances as possible.

Results and Discussion

In the following, we present results from our experimental evaluation of our dynamic algorithm termination method for the various verification algorithms we considered, and we

Benchmark	$\alpha\beta$-CROWN				VeriNet				Oval			
	Running Time [GPU hours]		# Solved		Running Time [GPU hours]		# Solved		Running Time [GPU hours]		# Solved	
5 100	21.97	(70%)	868	(±0)	18.81	(28%)	576	(-4)	7.88	(9%)	430	(±0)
8 100	17.86	(43%)	766	(-1)	16.75	(22%)	500	(-1)	3.57	(4%)	386	(-1)
Conv Big	1.01	(68%)	918	(±0)	5.48	(49%)	868	(±0)	6.36	(41%)	841	(-1)
Conv Small	1.00	(80%)	969	(-10)	11.30	(94%)	931	(±0)	6.08	(100%)	958	(±0)
ResNet 2B	4.30	(28%)	619	(±0)	10.45	(46%)	576	(±0)	-		-	
Marabou	2.47	(5%)	192	(-1)	6.36	(12%)	168	(-8)	4.97	(9%)	185	(-2)
Oval21	7.57	(15%)	207	(-3)	15.04	(26%)	155	(-3)	10.02	(19%)	199	(-2)
ViT	1.94	(5%)	251	(±0)	-		-		-		-	
SRI ResNet A	3.86	(7%)	197	(-1)	8.71	(14%)	133	(±0)	-		-	
CIFAR-100	4.91	(20%)	360	(-1)	19.52	(49%)	279	(±0)	-		-	
Tiny ImageNet	4.54	(31%)	421	(±0)	19.4	(66%)	354	(-2)	-		-	

Table 3: Results for dynamic termination of verification queries with $\theta = 0.99$. We display the running time and the number of solved instances accumulated over five folds. In parentheses, we provide the fraction of running time used and the difference in the number of solved instances compared to the standard verification procedure.

show how our approach can be leveraged to allocate available resources more efficiently by terminating instances that will result in timeouts earlier in the verification process.

Classification Metrics. We report the classification metrics of our proposed method in Table 2 as averages over all five folds. We obtained very high TPR scores while maintaining a FPR close to 0 for most verifiers and benchmarks. Concretely, on average, our classifier correctly identified 84% of timeouts while incorrectly classifying 5% of solvable instances. Noticeably, across all verifiers, there were several benchmarks with TPRs above 90% and FPRs of almost zero. Lower TPR scores on the *Conv Big* and *Conv Small* benchmarks were due to the relatively small number of timeouts occurring in these benchmarks, leading to a lack of training examples for this class. Similarly, we observed higher FPR scores for the *Marabou* benchmark. This is likely due to the small number of queries solved after the first checkpoint, again resulting in less diverse training data.

Dynamic Algorithm Termination. We display the results of our method in terms of total cumulative running time and number of solved instances aggregated over all folds for each benchmark and verifier in Table 3.

Most importantly, we obtained substantial speed-ups, while only a small amount of solvable instances was terminated prematurely. On average, our approach solved comparably many instances in 36% of the original running time. Notably, the largest acceleration occurred on the Marabou benchmark, where up to 95% of the standard running time could be saved. However, we also observed moderate penalties in terms of the absolute difference of solved instances for the *Marabou* benchmark on VeriNet and the *Conv Small* benchmark on $\alpha\beta$-CROWN, due to the reasons stated earlier. Moreover, on several benchmarks, all solvable instances were certified using substantially reduced running time; *e.g.*, the *5 100* benchmark for Oval and $\alpha\beta$-CROWN or the *ResNet A* benchmark for VeriNet.

Overall, we found that our approach substantially accelerates neural network verification across several verification algorithms and a broad range of benchmarks.

Conclusions and Future Work

In this study, we have, for the first time, shown that the computational resources demanded by neural network robustness verification can be greatly reduced by identifying and terminating runs on verification instances that will not be solved within their remaining time budget. Concretely, we showed that our method accelerates the verification procedure by 64% on average compared to the current state-of-the-art approach across a diverse set of benchmarks from the verification literature, while certifying a comparable number of instances. To predict whether an instance will be solved, we leveraged running time prediction techniques that employ novel static and dynamic features capturing both characteristics of the verification instance as well as features related to the internal operations of the given verifier.

The success of the proposed method was enabled by several design decisions. First, we leveraged the evolving nature of our dynamic features by regularly predicting timeouts throughout the verification procedure. Moreover, we included a confidence parameter θ that controls the threshold the prediction value of the timeout class must exceed before an instance is terminated. Using this parameter, a user can adjust the method to either prioritise savings in compute resources or a higher number of solved instances. We show that for a high value of θ our method substantially accelerates the verification procedure while solving comparably many instances as the standard verification.

In future work, we seek to extend our approach to further BaB-based verification approaches (*e.g.*, MN-BaB). Furthermore, we plan to investigate if our proposed features could be applied in other contexts, such as algorithm selection or satisfiability prediction. Lastly, we are interested in further studying the running time prediction capabilities of our features, possibly enabling empirical scaling models of BaB-based verification.

Acknowledgments

This research was partially supported by TAILOR, a project funded by EU Horizon 2020 research and innovation program under GA No. 952215. HH gratefully acknowledges support through an Alexander-von-Humboldt Professorship in Artificial Intelligence. Simulations were performed with computing resources granted by RWTH Aachen University under project *thes1552*. The authors would like to thank Anja Jankovic, Annelot Bosman, Hadar Shavit and Jan van Rijn for insightful discussions and helpful feedback. Furthermore, the authors thank the reviewers for their valuable comments.

References

Anderson, R.; Huchette, J.; Ma, W.; Tjandraatmadja, C.; and Vielma, J. P. 2020. Strong mixed-integer programming formulations for trained neural networks. *Mathematical Programming*, 183(1-2): 3–39.

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

Brix, C.; Bak, S.; Liu, C.; and Johnson, T. T. 2023. The Fourth International Verification of Neural Networks Competition (VNN-COMP 2023): Summary and Results. *arXiv preprint arXiv:2312.16760.*

Bunel, R.; Lu, J.; Turkaslan, I.; Torr, P. H.; Kohli, P.; and Kumar, M. P. 2020. Branch and Bound for Piecewise Linear Neural Network Verification. *Journal of Machine Learning Research*, 21(42): 1–39.

De Palma, A.; Behl, H. S.; Bunel, R.; Torr, P.; and Kumar, M. P. 2021a. Scaling the Convex Barrier with Active Sets. In *Proceedings of the 9th International Conference on Learning Representations (ICLR 2021)*, 1–27.

De Palma, A.; Behl, H. S.; Bunel, R.; Torr, P. H.; and Kumar, M. P. 2021b. Scaling the convex barrier with sparse dual algorithms. *arXiv preprint arXiv:2101.05844.*

De Palma, A.; Bunel, R.; Desmaison, A.; Dvijotham, K.; Kohli, P.; Torr, P. H.; and Kumar, M. P. 2021c. Improved Branch and Bound for Neural Network Verification via Lagrangian Decomposition. *arXiv preprint arXiv:2104.06718.*

Ferrari, C.; Mueller, M. N.; Jovanović, N.; and Vechev, M. 2022. Complete Verification via Multi-Neuron Relaxation Guided Branch-and-Bound. In *Proceedings of the 10th International Conference on Learning Representations (ICLR 2022)*, 1–15.

Feurer, M.; Klein, A.; Eggensperger, K.; Springenberg, J.; Blum, M.; and Hutter, F. 2015. Efficient and Robust Automated Machine Learning. In *Advances in Neural Information Processing Systems 28 (NeurIPS 2015)*, 2962–2970.

Goodfellow, I. J.; Shlens, J.; and Szegedy, C. 2015. Explaining and Harnessing Adversarial Examples. In *3rd International Conference on Learning Representations, (ICLR 2015)*, 1–11.

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

Hutter, F.; Xu, L.; Hoos, H. H.; and Leyton-Brown, K. 2014. Algorithm Runtime Prediction: Methods & Evaluation. *Artificial Intelligence*, 206: 79–111.

Julian, K. D.; Kochenderfer, M. J.; and Owen, M. P. 2019. Deep Neural Network Compression for Aircraft Collision Avoidance Systems. *Journal of Guidance, Control, and Dynamics*, 42(3): 598–608.

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

Kerschke, P.; Hoos, H. H.; Neumann, F.; and Trautmann, H. 2019. Automated Algorithm Selection: Survey and Perspectives. *Evolutionary Computation*, 27(1): 3–45.

König, M.; Bosman, A. W.; Hoos, H. H.; and van Rijn, J. N. 2024. Critically Assessing the State of the Art in Neural Network Verification. *Journal of Machine Learning Research*, 25(12): 1–53.

König, M.; Hoos, H. H.; and van Rijn, J. N. 2024. Accelerating Adversarially Robust Model Selection for Deep Neural Networks via Racing. In *Proceedings of the 38th AAAI Conference on Artificial Intelligence (AAAI-24)*, 1–9.

Müller, M. N.; Brix, C.; Bak, S.; Liu, C.; and Johnson, T. T. 2022a. The Third International Verification of Neural Networks Competition (VNN-COMP 2022): Summary and Results. *arXiv preprint arXiv:2212.10376.*

Müller, M. N.; Makarchuk, G.; Singh, G.; Püschel, M.; and Vechev, M. 2022b. PRIMA: General and Precise Neural Network Certification via Scalable Convex Hull Approximations. In *Proceedings of the 6th ACM on Programming Languages (POPL)*, 1–33.

Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; and Duchesnay, E. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12: 2825–2830.

Singh, G.; Ganvir, R.; Püschel, M.; and Vechev, M. 2019a. Beyond the Single Neuron Convex Barrier for Neural Network Certification. In *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*, 15098—-15109.

Singh, G.; Gehr, T.; Püschel, M.; and Vechev, M. 2019b. An Abstract Domain for Certifying Neural Networks. In *Proceedings of the 3rd ACM on Programming Languages (POPL 2019)*, 1–30.

Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; and Fergus, R. 2014. Intriguing properties of neural networks. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR 2014)*, 1–10.

Tjeng, V.; Xiao, K. Y.; and Tedrake, R. 2019. Evaluating Robustness of Neural Networks with Mixed Integer Programming. In *Proceedings of the 7th International Conference on Learning Representations (ICLR 2019)*, 1–21.

Wang, S.; Pei, K.; Whitehouse, J.; Yang, J.; and Jana, S. 2018. Formal Security Analysis of Neural Networks using Symbolic Intervals. In *Proceedings of the 27th USENIX Security Symposium (USENIX Security 18)*, 1599–1614.

Wang, S.; Zhang, H.; Xu, K.; Lin, X.; Jana, S.; Hsieh, C.-J.; and Kolter, J. Z. 2021. 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)*, 29909–29921.

Wong, E.; and Kolter, Z. 2018. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International conference on machine learning*, 5286–5295. PMLR.

Xu, K.; Zhang, H.; Wang, S.; Wang, Y.; Jana, S.; Lin, X.; and Hsieh, C.-J. 2021. Fast and Complete: Enabling Complete Neural Network Verification with Rapid and Massively Parallel Incomplete Verifiers. In *Proceedings of the 9th International Conference on Learning Representations (ICLR 2021)*, 1–15.

Zhang, H.; Weng, T.-W.; Chen, P.-Y.; Hsieh, C.-J.; and Daniel, L. 2018. Efficient Neural Network Robustness Certification with General Activation Functions. In *Advances in Neural Information Processing Systems 31 (NeurIPS 2018)*, 4944—-4953.