# CTRAIN - A Training Library for Certifiably Robust Neural Networks

Konstantin Kaulen[1] and Holger H. Hoos[1,2]

[1] Chair for AI Methodology, RWTH Aachen University, Germany
[2] LIACS, Leiden University, The Netherlands
{kaulen,hh}@aim.rwth-aachen.de

**Abstract.** Despite their widespread success, neural networks are susceptible to *adversarial examples*, severely limiting their responsible deployment in safety-critical scenarios. To address this, *neural network verification* techniques have been proposed that rigorously prove the robustness of a given network against specific threats. However, the scalability of these methods remains a major challenge, with networks trained for empirical robustness still proving difficult to verify. Thus, *certified training* has been proposed to produce networks more amenable to formal robustness verification. However, there is currently no comprehensive framework allowing easy access to these training methods. To address this, we introduce `CTRAIN`, a new Python library built upon the `auto_LiRPA` package, which reimplements state-of-the-art certified training methods in a unified, modular and comprehensive manner, while offering user-friendly interfaces, enhancing accessibility for both researchers and practitioners. Additionally, `CTRAIN` integrates `SMAC3` for hyperparameter optimisation and $\alpha\beta$-CROWN for complete verification, empowering users to exploit these systems to achieve state-of-the-art certified robustness. We provide code, documentation, examples and usage instructions at `github.com/ada-research/CTRAIN`.

## 1 Introduction

In recent years, neural networks have shown remarkable performance across various application domains, ranging from computer vision [8] to protein structure prediction [13]. At the same time, it became evident that neural networks are typically not robust, as adversarially crafted, yet imperceptible, changes in the input can lead to to incorrect predictions [35]. This circumstance severely limits the responsible deployment of machine learning models in safety-critical use cases. To mitigate this issue, *neural network verification* techniques have been proposed, which provide provable robustness guarantees using rigorous mathematical frameworks [14, 36]. Generally, these can be divided into two families; cheap incomplete methods attempt to solve the robustness verification problem by bounding the outputs of a network, but may not be able to prove a property due to overly loose bounds. Complete methods will, in principle, always return a result but have to solve an expensive $\mathcal{NP}$-complete problem [32, 19]. Despite

several algorithmic advancements, *e.g.*, the inclusion of sophisticated network over-approximations [5, 29, 34, 38, 43] or search techniques [3], the scalability of complete verification remains a major challenge.

Concurrently, specialised training methods were developed that aim to produce robust neural networks. While state-of-the-art empirical robustness can be achieved using *adversarial training* (see, *e.g.*, [22, 41]), the resulting networks remain hard to verify. Thus, there has been a surge of training methods that yield robust neural networks amenable to formal verification, therefore mitigating the challenge of limited scalability, giving rise to the concept of *certified training* [26, 11]. These methods employ an over-approximation of the worst case adversarial loss using cheap incomplete robustness verification methods as the training objective to be minimised. Several certified loss functions leveraging this concept have been proposed, gradually advancing the state-of-the-art regarding the number of input samples for which the resulting networks are provably robust [6, 11, 24, 28, 33]. However, the community to date lacks a comprehensive library that makes these techniques accessible to potentially inexperienced end users.

Therefore, we propose `CTRAIN`, an extensive Python library for certified training. We provide, for the first time, implementations of all current state-of-the-art methods, based on the popular neural network bounding library `auto-LiRPA` [39], and make these accessible via a Python package. Further, `CTRAIN` provides user-friendly interfaces to certifiably train neural networks based on the PyTorch framework [30]. Therefore, `CTRAIN` easily integrates into existing PyTorch training pipelines, neural network architecture specifications and datasets. Furthermore, we natively support sophisticated hyperparameter optimisation for certified training via the state-of-the-art optimiser `SMAC3` [21]. Last but not least, `CTRAIN` includes several possibilities for robustness evaluation using adversarial attacks, incomplete verification and the state-of-the-art complete neural network verification system $\alpha\beta$-CROWN [38, 40, 44].

## 2 Related Work

In the following, we provide an overview of work related to `CTRAIN`, focusing on tools that provide functionalities for training robust neural networks. To date, several easy-to-use and performant libraries have been proposed that implement adversarial training methods. Among these, the `Adversarial Robustness Toolbox` (ART) [37] and `DeepRobust` [20] constitute the most extensive and popular libraries, having accumulated over five thousand[1] and one thousand[2] stars on GitHub, respectively. These stars allow users to indicate interest in a repository and to bookmark it. Both implement multiple adversarial training methods, *e.g.*, training for robustness on examples created through the *Projected Gradient Descent* (PGD) method [22], a strong iterative adversarial attack. However, these libraries lack proper support for methods that focus on producing easily verifiable networks. Specifically, ART only supports early advancements from the field

[1] `github.com/Trusted-AI/adversarial-robustness-toolbox`
[2] `github.com/DSE-MSU/DeepRobust`

that do not constitute the state of the art anymore [11, 26], while `DeepRobust` implements no certified trainings method at all.

Recently, `CTBench`, a novel and unified library for certified training, has been proposed [23]. `CTBench` implements several state-of-the-art protocols, including SABR [28] and MTL-IBP [6], and the authors reported very strong results using their implementation. Nonetheless, `CTBench` cannot be easily integrated into existing code, since it relies on independent training scripts, and a Python package providing convenient options for running the `CTBench` training code does not exist. In addition, `CTBench` employs the verification system MN-BaB [10] to evaluate the certifiable robustness of trained neural networks, which has been shown to be consistently outperformed by $\alpha\beta$-CROWN [38, 40, 44] in recent studies and competitions [15, 27]. While we acknowledge the importance of `CTBench`, we believe that researchers as well as end users will profit from easy-to-use alternative implementations based on the popular `auto_LiRPA` library. In addition, we believe that the use of $\alpha\beta$-CROWN will lead to more precise assessments of certified training methods.

## 3 Overview of `CTRAIN`

In the following, we describe the key components and features of the `CTRAIN` library, including supported certified training methods, affordances for evaluating the empirical and certified robustness of neural networks, and native support for hyperparameter optimisation of certified training methods.

### 3.1 Certified Training with `CTRAIN`

**Selected Certified Training Methods.** `CTRAIN` implements several state-of-the-art algorithms for certified training. In selecting these, we focused on methods that provide *deterministic* robustness guarantees against all possible perturbations included in the $l_\infty$ norm balls with radius $\epsilon$ around input images. These perturbations constitute the properties typically examined in the neural network verification literature (see, *e.g.*, [2, 15]). Furthermore, we excluded methods that rely on non-standard neural network components not natively supported by the PyTorch library [30].

The best-peforming losses for deterministic certified training are based on Interval Bound Propagation (IBP) [11], the conceptually simplest incomplete verification method. IBP employs interval arithmetic to bound the outputs of a neural network which, in turn, can be used to calculate a sound upper bound of the worst-case loss on adversarial examples. The closely related *CROWN-IBP* [42] relies on the tighter bounding method CROWN [44] in combination with IBP to improve on standard IBP-based certified training. Shi et al. [33] propose further improvements to IBP through an initialisation procedure and loss regularisers that are specifically crafted to stabilise certified training.

Recently, significant advancements have been made by combining PGD-based adversarial training with IBP-based certified training. Those methods rely on

unsound approximations of the worst-case adversarial loss, but yield strongly improved performance. *SABR* [28] uses PGD to identify adversarial examples in the $l_\infty$ norm ball around the training instance, which are in turn used as the centre of a smaller norm ball. This smaller input region is then employed in standard IBP bounding to obtain the overall training loss. *TAPS* [24] combines adversarial and certified training by first propagating an input region through the feature extractor of a network using IBP, and by then adversarially training the classifier using latent adversarial examples that lie in the output region of the feature extractor. *STAPS* [24] works similarly to TAPS, but uses SABR instead of IBP to obtain intermediate bounds. Finally, *MTL-IBP* [6] is a representative member of the family of *expressive losses*, *i.e.*, losses that combine adversarial and certified losses through convex combinations. The MTL-IBP loss consists of the weighted sum of the certified loss obtained using IBP and the PGD-based adversarial loss.

In `CTRAIN`, we have included all previously mentioned certified training methods, *i.e.*, IBP, CROWN-IBP, SABR, TAPS, STAPS and MTL-IBP, ensuring comprehensive coverage of established approaches. This selection provides users with a diverse and relevant set of techniques, since methods combining adversarial and certified losses have shown the strongest results in recent literature and thus constitute the state-of-the-art (see, *e.g.*, [6, 23, 25]). While standard IBP and CROWN-IBP training was surpassed performance-wise, they remain the most computationally efficient and, thus, represent viable alternatives when potent hardware is not available.

**Key Features of `CTRAIN`.** In `CTRAIN` we provide, for the first time, an unified implementation of the state-of-the-art in certified training based on the `auto_LiRPA` [39] library. This package serves as the backbone of the state-of-the-art verification tool $\alpha\beta$-CROWN [40, 38], is actively maintained, implements a variety of incomplete verification techniques and is popular among the neural network verification community, testified by over 250 GitHub stars[3]. Furthermore, it provides extensive support for many popular network architectures, ranging from convolutional networks to transformers.

In `CTRAIN`, we implemented the previously mentioned certified training methods closely following the original literature and codebases, but reimplemented all relevant parts of certified training in a modular and highly configurable fashion. Furthermore, we unified varying implementations of network bounding, loss calculation and adversarial attacks into one comprehensive code base. Therefore, `CTRAIN` enhances comparability between methods by standardising their shared components, such as IBP bounding or regularisation. Additionally, all components of `CTRAIN` are implemented using PyTorch [30]; thus, the package integrates well into common machine learning pipelines and PyTorch components such as optimisers, regularisers, and data augmentations can be seamlessly incorporated.

[3] github.com/Verified-Intelligence/auto_LiRPA

### 3.2 Evaluation

To assess whether the network actually adheres to desired robustness properties, users require easy and extensive possibilities to evaluate neural networks regarding their certified and empirical robustness, which `CTRAIN` provides. Notably, `CTRAIN` can also be used to evaluate models that were trained outside of its training workflow and, thus, also represents a valuable tool for users that do only desire to use the evaluation capabilities of `CTRAIN`.

**Empirical Robustness.** To evaluate the robustness of a given network against adversarial attacks, we implemented the *PGD* attack [22], which to date is the *de-facto* standard method to assess empirical robustness. For example, state-of-the-art verification tools such as $\alpha\beta$-CROWN [38] or MN-BaB [10] use this attack to identify counter-examples.

**Incomplete Verification** To give provable guarantees of the robustness of neural networks, `CTRAIN` implements several incomplete verification methods using `auto_LiRPA`. More specifically, the incomplete bounding methods *IBP* [11], *CROWN-IBP* [42] and *CROWN* [43] are included. These methods differ in the tightness of the network bounds they compute, but also in their computational complexity. *IBP* is the cheapest and loosest incomplete method, *CROWN-IBP* gives tighter bounds at the cost of increased computational costs, and *CROWN* is the tightest and most expensive method. Users can decide whether all inputs that should be investigated are verified using one method, or whether verfication is performed in an *adaptive* fashion. In the latter case, the supported methods are progressively applied to input samples in increasing order of their computational costs. Therefore, easy verification problems are solved with cheap methods, while computationally expensive methods are only applied to problems where their tightness is required to obtain a solution (see, *e.g.*, [6, 23]).

**Complete Verification** Finally, `CTRAIN` also provides an interface to the state-of-the-art complete verification system $\alpha\beta$-CROWN. Complete verification provides the most accurate assessment of the certified robustness of a given network at the cost of significantly increased computational requirements. Especially, networks trained using recent methods based on surrogate losses, require complete verification to obtain precise robustness measurements [6, 28].

To save computational resources, `CTRAIN` first attempts to obtain a solution to the verification query by applying its included incomplete verification techniques and by running the *PGD* adversarial attack, before invoking $\alpha\beta$-CROWN [6, 23].

### 3.3 Hyperparameter Optimisation

All certified training methods are parametrised by a extensive and diverse set of hyperparameters, such as the number of $\epsilon$-annealing epochs or the settings of the PGD attack. Furthermore, the values chosen for those parameters influence the training outcome strongly, ranging from training collapse to state-of-the-art results. Recent works tackle the hyperparameter optimisation problem by employing manual [6] or grid search [23] over an expert-designed configuration

space. In any case, these approaches to hyperparameter tuning for certified training currently require extensive domain knowledge to identify suitable parameter choices. To mitigate this prerequisite and to therefore make hyperparameter tuning more accessible to potentially inexperienced practitioners , `CTRAIN` implements preconfigured hyperparameter optimisation as one of its core components.

We employ the state-of-the-art hyperparameter optimisation system `SMAC3` [21] for the tuning task, since it has demonstrated remarkable performance across various recent benchmarks [9, 31]. For each of the implemented certified training methods, we provide a configuration space, out of which `SMAC3` attempts to find the best-performing configuration. Thus, when using `CTRAIN`, users do not require domain knowledge to achieve state-of-the-art results on novel datasets for which no well-performing configurations are known.

By default, `CTRAIN` aims to optimise the sum of natural, certified and adversarial accuracy, since all of these metrics represent desirable properties of a certifiably trained neural network, *i.e.* strong performance on natural and adversarial inputs and easy verifiability. Nevertheless, the accuracy values that should be included in the optimisation objective can be weighted according to user preferences. To keep the evaluation overhead manageable, these values are by default computed on the first 1000 samples of the validation dataset, using *CROWN*.

The `CTRAIN` hyperparameter optimisation procedure begins by exploring the search space through a random search for the number of iterations determined by the number of hyperparameters in the configuration space. Compared to the default of `SMAC3`, we limit this number, to avoid overspending on random configurations, since training and evaluation is costly. In addition, `CTRAIN` allows users to specify a pre-defined configuration, which is assumed to performing well. This modification to the SMAC3 optimisation procedure facilitates the exploitation of expert user knowledge. Subsequently, `CTRAIN` continues with the optimisation procedure until the budget is exhausted.

## 4 Implementation

In the following, we explain the architectural details and implementation of the `CTRAIN` library, highlighting its modular and well-structured design as well as its easy usability.

First and foremost, we designed `CTRAIN` as a Python library that can easily be installed and set up using package management tools such as `pip`. Furthermore, we made sure that `CTRAIN` seamlessly integrates into common machine learning workflows without the need to run separate scripts or to set up different environments. We implemented `CTRAIN` in Python 3, currently using `torch` in version 2.2.2 and `auto_LiRPA` in version 0.50 as its core libraries. Our implementation can be accessed at `github.com/ada-research/CTRAIN`.

`CTRAIN.model_wrappers.` As a result of our considerations for the design of `CTRAIN`, we provide a package implementing *model wrappers* that can be easily included into existing code. These wrappers encapsulate predefined or pretrained neural networks and expose core functionalities in an accessible manner.

```
from CTRAIN.model_definitions import CNN7_Shi
from CTRAIN.data_loaders import load_cifar10
from CTRAIN.model_wrappers import ShiIBPModelWrapper

train_loader, test_loader = load_cifar10(val_split=False)
in_shape = [3, 32, 32]

model = CNN7_Shi(in_shape=in_shape)
wrapped_model = ShiIBPModelWrapper(model=model,
    input_shape=in_shape, eps=2/255, num_epochs=160)

wrapped_model.train_model(train_loader)
std_acc, cert_acc, adv_acc =
    wrapped_model.evaluate(test_loader)
```

**Code Example 4.1.** `CTRAIN` is easy to use for certifiably training and evaluating neural networks: In twelve lines of code, users can load a dataset, define the standard CNN7 network architecture proposed by Shi et al. [33], certifiably train the network using IBP and evaluate it, using adversarial attacks and incomplete verification.

We show an example of the usage of the model wrappers provided by `CTRAIN` in Code Example 4.1. For each of the supported training methods, there is one separate wrapper. These objects take the neural network, which must inherit from the PyTorch `nn.Module` class, the perturbation magnitude $\epsilon$ that defines the training and verification objectives, and the method-specific training hyperparameters as arguments. Since using a higher $\epsilon$ during training compared to evaluation might be beneficial (see, *e.g.*, [6, 42]), users can define a multiplier to scale the training $\epsilon$. Training is invoked via the `train_model` function, while an evaluation of natural, robust and certified performance can be carried out using the `evaluate` or `evaluate_complete` functions, respectively. The hyperparameter optimisation procedure is implemented in the `hpo` function, for which an optimisation budget should be provided that specifies for how long the optimisation procedure runs. Furthermore, the user may pass a default configuration to be investigated during the optimisation process.

The `model_wrappers` package is easily extensible, since all wrappers inherit from the common base class `CTRAINWrapper`, which implements method-independent functionalities such as evaluation, checkpoint saving and hyperparameter optimisation. In addition, it was of paramount importance for `CTRAIN` to be compatible with common `PyTorch` operations. Therefore, the base class inherits from the `nn.Module` class and, thus, all wrappers can be used in existing training and evaluation workflows.

`CTRAIN.bound.` The `bound` module implements all bounding operations required during training and incomplete verification, based on the `auto_LiRPA` package. More specifically, it implements the sound bounding operations IBP, CROWN-IBP and CROWN as well as the unsound SABR and TAPS bounds.

`CTRAIN.data.` Although `CTRAIN` is fully compatible with standard `PyTorch` data loaders, we provide functions that load the common vision datasets MNIST [18], CIFAR-10 [16] and TinyImageNet [17].

`CTRAIN.eval.` The `eval` package provides all functions required to evaluate standard, robust and certified accuracy of neural networks, including functions to carry out incomplete and complete verification as well as adversarial attacks. Generally, all evaluation methods require the user to pass the network that should be evaluated, the perturbation magnitude $\epsilon$ and a data loader, which holds the inputs for which the robustness should be assessed. Since the evaluation procedure may be costly, especially when employing complete verification, users may also provide a number of samples for which the evaluation is carried out in the given order of the evaluation set. To utilise complete verification, users must pass, in addition to the arguments generally required for evaluation, the allowed maximum running time per verification query and the number of CPU cores $\alpha\beta$-CROWN may utilise. In addition, a dictionary including configuration values for $\alpha\beta$-CROWN that adhere to its documentation may be provided.

`CTRAIN.complete_verification.` Since $\alpha\beta$-CROWN is not intended to be executed directly from external libraries or codebases, `CTRAIN` implements several steps to ensure seamless integration in its `abCROWN` subpackage. First, `CTRAIN` exports the network in `ONNX` format [1] and saves the resulting file to a temporary folder. Then, it formulates the verification property in the standardised `VNN-LIB` format [7] and also saves the resulting file. Thereafter, `CTRAIN` generates a configuration file for $\alpha\beta$-CROWN that specifies the cutoff time and further parameters set by the user as well as the verification property, defined through the previously generated `ONNX` and `VNN-LIB` files. Finally, `CTRAIN` invokes $\alpha\beta$-CROWN by calling the function that serves as the entry point of the verification system, passing along the configuration file.

`CTRAIN.attacks.` `CTRAIN` currently only implements the PGD adversarial attack, which is used in several training losses as well as in the empirical robustness evaluation. We have made the parameters of the attack, *i.e.*, the number of restarts, the number of steps and the step size, configurable. Furthermore, users can define *decay milestones* at which the step size is reduced by a specified factor. When PGD attacks are involved in training, we set the network to *evaluation* mode when carrying out the attacks, while the loss computation based on the obtained adversarial examples is done in *training* mode. Therefore, the statistics of batch normalisation layers [12] are not influenced by the forward and backward passes performed during attacks and employ the mean and variance computed over both unperturbed and perturbed inputs at evaluation time.

`CTRAIN.model_definitions.` While `CTRAIN` is, in principle, compatible with a broad range of neural network definitions, models proposed by Shi et al. [33] emerged as the *de-facto* standard architectures for evaluating certified training methods on (see, *e.g.*, [6, 23, 28]). We provide model definitions of these networks in the `model_definitions` package.

`CTRAIN.train.certified.` This package implements all components of certified training in a functional manner. In the subpackage `losses`, we provide functions

for calculating the various supported certified losses. The `initialisation` and `regularisation` packages provide implementations of the procedures proposed by Shi et al. [33] as well as an implementation of $l_1$ regularisation. Finally, we have implemented each supported certified training method as one separate function that is utilised in the respective model wrappers.

`CTRAIN.util.` Finally, we have implemented utility functions, such as seeding the library or exporting networks to `ONNX`, in the `util` package.

## 5 Conclusions and Future Work

In this work, we presented `CTRAIN`, a new Python library for certified training. `CTRAIN` implements several state-of-the-art certified training protocols and makes them accessible via model wrappers that integrate well into existing machine learning workflows based on PyTorch. Furthermore, `CTRAIN` provides a broad range of evaluation functions that can assess the robustness of a given network using adversarial attacks as well as incomplete and complete verification. Notably, using `CTRAIN`, it becomes possible to invoke the state-of-the-art complete verification system $\alpha\beta$-CROWN using only one function call. Last but not least, `CTRAIN` has native support for sophisticated hyperparameter optimisation using `SMAC3`.

In future work, we aim to maintain and further extend the functionalities of `CTRAIN`. The modular design of the library allows for easy addition of adversarial attack mechanisms, such as *AutoAttack* [4], or of complete verification systems, *e.g.*, *Oval* [5]. Furthermore, we intend to implement further enhancements to certified training, such as ReLU transformer shrinking [28]. In addition, we will perform an extensive empirical evaluation of `CTRAIN`, comparing its performance to reference implementations from the literature and examining potential improvements achieved through the use of `SMAC3` and $\alpha\beta$-CROWN. Finally, we plan to continuously update `CTRAIN` with new certified training and verification methods, maintaining `CTRAIN` as a state-of-the-art resource for certified training and its evaluation, valuable to both end-users and researchers.

**Acknowledgments.** The authors would like to express their sincere gratitude to Hadar Shavit for providing valuable feedback, insightful inspiration, and expert knowledge on hyperparameter optimisation. Holger H. Hoos gratefully acknowledges support through an Alexander-von-Humboldt Professorship in Artifcial Intelligence. Furthermore, the authors thank the reviewers for their valuable comments.

**Disclosure of Interests.** The authors have no competing interests to declare that are relevant to the content of this article.

## References

1. Bai, J., Lu, F., Zhang, K., et al.: ONNX: Open Neural Network Exchange. `https://github.com/onnx/onnx` (2025)

2. Brix, C., Müller, M.N., Bak, S., Johnson, T.T., Liu, C.: First three years of the international verification of neural networks competition (VNN-COMP). International Journal on Software Tools for Technology Transfer **25**(3), 329–339 (2023)
3. Bunel, R., Lu, J., Turkaslan, I., Torr, P.H., Kohli, P., Kumar, M.P.: Branch and Bound for Piecewise Linear Neural Network Verification. Journal of Machine Learning Research **21**(42), 1–39 (2020)
4. Croce, F., Hein, M.: Reliable Evaluation of Adversarial Robustness with an Ensemble of Diverse Parameter-free Attacks. In: Proceedings of the 37th International Conference on Machine Learning, (ICML 2020). vol. 119, pp. 2206–2216 (2020)
5. De Palma, A., Behl, H.S., Bunel, R., Torr, P.H.S., Kumar, M.P.: Scaling the Convex Barrier with Sparse Dual Algorithms. Journal of Machine Learning Research **25**(61), 1–51 (2024)
6. De Palma, A., Bunel, R., Dvijotham, K.D., Kumar, M.P., Stanforth, R., Lomuscio, A.: Expressive Losses for Verified Robustness via Convex Combinations. In: Proceedings of the 12th International Conference on Learning Representations (ICLR 2024). pp. 1–28 (2024)
7. Demarchi, S., Guidotti, D., Pulina, L., Tacchella, A., Narodytska, N., Amir, G., Katz, G., Isac, O.: Supporting Standardization of Neural Networks Verification with VNNLIB and CoCoNet. In: Proceedings of the 6th Workshop on Formal Methods for ML-Enabled Autonomous Systems (FoMLAS 2023). pp. 47–58 (2023)
8. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., Houlsby, N.: An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In: Proceedings of the 9th International Conference on Learning Representations (ICLR 2021). pp. 1–22 (2021)
9. Eggensperger, K., Müller, P., Mallik, N., Feurer, M., Sass, R., Klein, A., Awad, N.H., Lindauer, M., Hutter, F.: HPOBench: A Collection of Reproducible Multi-Fidelity Benchmark Problems for HPO. In: Proceedings of the 35th Conference on Neural Information Processing Systems (NeurIPS 2021) Track on Datasets and Benchmarks. pp. 1–36 (2021)
10. Ferrari, C., Mueller, M.N., Jovanović, N., Vechev, M.: Complete Verification via Multi-Neuron Relaxation Guided Branch-and-Bound. In: Proceedings of the 10th International Conference on Learning Representations (ICLR 2022). pp. 1–15 (2022)
11. Gowal, S., Dvijotham, K.D., Stanforth, R., Bunel, R., Qin, C., Uesato, J., Arandjelovic, R., Mann, T., Kohli, P.: Scalable Verified Training for Provably Robust Image Classification. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 4842–4851 (2019)
12. Ioffe, S., Szegedy, C.: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In: Proceedings of the 32nd International Conference on Machine Learning (ICML 2015). vol. 37, pp. 448–456 (2015)
13. Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., et al.: Highly accurate protein structure prediction with AlphaFold. Nature **596**(7873), 583–589 (2021)
14. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In: Proceedings of the 29th International Conference on Computer Aided Verification (CAV 2017). pp. 97–117 (2017)
15. König, M., Bosman, A.W., Hoos, H.H., van Rijn, J.N.: Critically Assessing the State of the Art in Neural Network Verification. Journal of Machine Learning Research **25**(12), 1–53 (2024)

16. Krizhevsky, A., Hinton, G., et al.: Learning Multiple Layers of Features from Tiny Images (2009)
17. Le, Y., Yang, X.S.: Tiny ImageNet Visual Recognition Challenge (2015)
18. LeCun, Y.: The MNIST Database of Handwritten Digits (1998)
19. Li, L., Xie, T., Li, B.: Sok: Certified robustness for deep neural networks. In: Proceedings of the 44th IEEE Symposium on Security and Privacy (S&P 2023). pp. 1289–1310. IEEE (2023)
20. Li, Y., Jin, W., Xu, H., Tang, J.: DeepRobust: a Platform for Adversarial Attacks and Defenses. In: Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI-21). pp. 16078–16080 (2021)
21. Lindauer, M., Eggensperger, K., Feurer, M., Biedenkapp, A., Deng, D., Benjamins, C., Ruhkopf, T., Sass, R., Hutter, F.: SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization. Journal of Machine Learning Research **23**(54), 1–9 (2022)
22. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards Deep Learning Models Resistant to Adversarial Attacks. In: Proceedings of 6th International Conference on Learning Representations (ICLR 2018). pp. 1–23 (2018)
23. Mao, Y., Balauca, S., Vechev, M.: CTBENCH: A Library and Benchmark for Certified Training. arXiv preprint arXiv:2406.04848 (2024)
24. Mao, Y., Müller, M.N., Fischer, M., Vechev, M.T.: Connecting Certified and Adversarial Training. In: Advances in Neural Information Processing Systems 37 (NeurIPS 2023). pp. 1–19 (2023)
25. Mao, Y., Müller, M.N., Fischer, M., Vechev, M.T.: Understanding Certified Training with Interval Bound Propagation. In: Proceedings of the 12th International Conference on Learning Representations (ICLR 2024). pp. 1–23 (2024)
26. Mirman, M., Gehr, T., Vechev, M.: Differentiable Abstract Interpretation for Provably Robust Neural Networks. In: Proceedings of the 35th International Conference on Machine Learning (ICML 2018). pp. 3578–3586. PMLR (2018)
27. Müller, M.N., Brix, C., Bak, S., Liu, C., Johnson, T.T.: The Third International Verification of Neural Networks Competition (VNN-COMP 2022): Summary and Results. arXiv preprint arXiv:2212.10376 (2022)
28. Müller, M.N., Eckert, F., Fischer, M., Vechev, M.T.: Certified Training: Small Boxes are All You Need. In: Proceedings of the 11th International Conference on Learning Representations (ICLR 2023). pp. 1–21 (2023)
29. Müller, M.N., Makarchuk, G., Singh, G., Püschel, M., Vechev, M.: PRIMA: General and Precise Neural Network Certification via Scalable Convex Hull Approximations. In: Proceedings of the 6th ACM on Programming Languages (POPL). pp. 1–33 (2022)
30. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: Advances in Neural Information Processing Systems 33 (NeurIPS 2019). pp. 1–12 (2019)
31. Pfisterer, F., Schneider, L., Moosbauer, J., Binder, M., Bischl, B.: YAHPO Gym - An Efficient Multi-Objective Multi-Fidelity Benchmark for Hyperparameter Optimization. In: Proceedings of the First International Conference on Automated Machine Learning (AutoML-Conf 2022). vol. 188, pp. 3/1–39. PMLR (2022)
32. Sälzer, M., Lange, M.: Reachability is NP-complete even for the simplest neural networks. In: Proceedings of the 15th International Conference on Reachability Problems (RP 2021). pp. 149–164. Springer (2021)

33. Shi, Z., Wang, Y., Zhang, H., Yi, J., Hsieh, C.: Fast Certified Robust Training with Short Warmup. In: Advances in Neural Information Processing Systems 34 (NeurIPS 2021). pp. 18335–18349 (2021)
34. Singh, G., Gehr, T., Püschel, M., Vechev, M.: An Abstract Domain for Certifying Neural Networks. In: Proceedings of the 3rd ACM on Programming Languages (POPL 2019). pp. 1–30 (2019)
35. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks. In: Proceedings of the 2nd International Conference on Learning Representations (ICLR 2014). pp. 1–10 (2014)
36. Tjeng, V., Xiao, K.Y., Tedrake, R.: Evaluating Robustness of Neural Networks with Mixed Integer Programming. In: Proceedings of the 7th International Conference on Learning Representations (ICLR 2019). pp. 1–21 (2019)
37. Trusted-AI: Adversarial Robustness Toolbox. `https://github.com/Trusted-AI/adversarial-robustness-toolbox` (2025)
38. Wang, S., Zhang, H., Xu, K., Lin, X., Jana, S., Hsieh, C.J., Kolter, J.Z.: 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). pp. 29909–29921 (2021)
39. Xu, K., Shi, Z., Zhang, H., Wang, Y., Chang, K., Huang, M., Kailkhura, B., Lin, X., Hsieh, C.: Automatic Perturbation Analysis for Scalable Certified Robustness and Beyond. In: Advances in Neural Information Processing Systems 33 (NeurIPS 2020). pp. 1–13 (2020)
40. Xu, K., Zhang, H., Wang, S., Wang, Y., Jana, S., Lin, X., Hsieh, C.J.: 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). pp. 1–15 (2021)
41. Zhang, H., Yu, Y., Jiao, J., Xing, E., Ghaoui, L.E., Jordan, M.: Theoretically Principled Trade-off between Robustness and Accuracy. In: Proceedings of the 36th International Conference on Machine Learning (ICML 2019). vol. 97, pp. 7472–7482. PMLR (2019)
42. Zhang, H., Chen, H., Xiao, C., Gowal, S., Stanforth, R., Li, B., Boning, D., Hsieh, C.J.: Towards Stable and Efficient Training of Verifiably Robust Neural Networks. In: Proceedings of the 8th International Conference on Learning Representations (ICLR 2020). pp. 1–15 (2019)
43. Zhang, H., Wang, S., Xu, K., Li, L., Li, B., Jana, S., Hsieh, C.J., Kolter, J.Z.: General Cutting Planes for Bound-Propagation-Based Neural Network Verification. Advances in Neural Information Processing Systems 35 (NeurIPS 2022) pp. 1656–1670 (2022)
44. Zhang, H., Weng, T.W., Chen, P.Y., Hsieh, C.J., Daniel, L.: Efficient Neural Network Robustness Certification with General Activation Functions. In: Advances in Neural Information Processing Systems 31 (NeurIPS 2018). pp. 4944—-4953 (2018)