Learning DNN Abstractions using Gradient Descent

Anonymous Author(s)

ABSTRACT

Deep Neural Networks (DNNs) are being trained and trusted for performing fairly complex tasks, even in business- and safety-critical applications. This necessitates that they be formally analyzed before deployment. Scalability of such analyses is a major bottleneck in their widespread use. There has been a lot of work on abstraction, and counterexample-guided abstraction refinement (CEGAR) of DNNs to address the scalability issue. However, these abstraction-refinement techniques explore only a subset of possible abstractions, and may miss an *optimal* abstraction. In particular, the refinement updates the abstract DNN based only on local information derived from the spurious counterexample in each iteration. The lack of a global view may result in a series of bad refinement choices, limiting the search to a region of sub-optimal abstractions. We propose a novel technique that parameterizes the construction of the abstract network in terms of continuous real-valued parameters. This allows us to use gradient descent to search through the space of possible abstractions, and ensures that the search never gets restricted to sub-optimal abstractions. Moreover, our parameterization can express more general abstractions than the existing techniques, enabling us to discover better abstractions than previously possible.

KEYWORDS

Deep Neural Networks, Abstraction, Formal Verification

1 INTRODUCTION

Advancements in the training of Deep Neural Networks (DNNs) have provided a scalable and practical solution to several previously intractable problems such as image and voice recognition, natural language processing, etc. This has encouraged their use in a variety of applications, even business- and safety-critical ones like medical diagnosis and prediction [14, 20] and autonomous vehicles [4]. However, DNNs are known to be vulnerable to adversarial [6, 7, 17, 19, 22, 32] and backdoor attacks [9], and therefore may not be *trustworthy* for such an application unless formally proven otherwise.

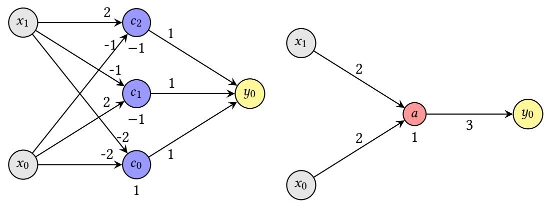
Several techniques have been proposed to determine the reliability of DNNs, including formal verification [18, 31, 33, 37] and explanation [3, 21]. Typically, the safety check is reduced to a *sat* query: given a property P on the output of a DNN $\mathcal{N}$, $\mathcal{N} \vdash P$ checks if there exists an input x for which the output violates P, i.e., if $\neg P(y) \wedge (y = \mathcal{N}(x))$ is satisfiable. Such a query can be handled by a variety of solvers, including complete solvers based on integrating LP solvers with *ReLU* phase fixing [18], DPLL(T) [15], and branch and bound [5, 26, 29, 33–36], as well as incomplete bound propagation based techniques [31, 37]. However, this problem is NP-hard [18], and the size of the DNNs continues to be a limiting factor in the scalability of these techniques.

ASE NIER'24, October 2024, Sacramento, California, United States of America
2024. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

A common approach in formal methods to address the scalability issue is *abstraction*, and naturally they have been explored in the context of safety checks for DNNs [12, 16, 38]. Given a DNN $\mathcal{N}$ (also referred to as the *original* or the *concrete* DNN), several abstraction techniques aim to produces a smaller, *abstract*, DNN $\mathcal{N}'$, possibly by *merging* groups of neurons in $\mathcal{N}$ into a single neuron in $\mathcal{N}'$. This sort of a reduction is useful for verification if it preserves soundness, i.e. if $\mathcal{N}' \vdash P$ holds, then $\mathcal{N} \vdash P$ holds as well. The underlying motivation being that an existing solver would find it easier to query the abstract DNN, irrespective of the technique used within this back-end solver. This is made practicable through a counterexample guided abstraction-refinement (CEGAR) [11] loop: if at any point the $\mathcal{N}'$ is not strong enough to prove the property, i.e. there is a counterexample to $\mathcal{N}' \vdash P$ which is not a counterexample to $\mathcal{N} \vdash P$, this *spurious* counterexample is used to drive heuristics that select certain merges to undo. In this way, $\mathcal{N}'$ is progressively refined until $\mathcal{N}' \vdash P$ holds or an actual counterexample is found.

Existing CEGAR approaches [12, 16, 38] use heuristics to choose which merge to undo in each iterations, and to maintain a small computational overhead of iterating through the abstractions, these heuristics only look at a *local* view of the behavior of the DNN. Such heuristics are susceptible to making bad choices, and even a single bad choice may restrict future iterations to explore only sub-optimal $\mathcal{N}'$. Besides, there may exist more efficient $\mathcal{N}'$ that are not reachable via the merge and un-merge operations utilized in existing CEGAR approaches [12, 16, 38]. The following example illustrates these points.

1.1 Illustrative Example



(a) Concrete Network $\mathcal{N}$ **(b) Fully Merged Network**

Figure 1

Consider the network in Fig. 1a, and the property P_1 that would state that $\boldsymbol{x} \in [0, 1] \Rightarrow \mathcal{N}(\boldsymbol{x}) \leq 4$, where $\boldsymbol{x} = [x_0\ x_1]$. For proving upper bound properties of this kind, it is sufficient to find $\mathcal{N}'$ so that **C**: $\forall \boldsymbol{x} \in [0, 1], \mathcal{N}'(\boldsymbol{x}) \geq \mathcal{N}(\boldsymbol{x})$, since proving an upper bound on the output of such an $\mathcal{N}'$ immediately gives an upper bound on $\mathcal{N}$. In the rest of the paper, we assume that the property is of this form, and focus on finding $\mathcal{N}'$ so that **C** holds. Note that it is shown in [16] that such an abstraction technique can be generalized to arbitrary networks and properties.

Notice that in this network the nodes c_0, c_1, c_2 are all such that, if one increases the value produced at these nodes, the value of the output of the network increases. Therefore, following [16], one can

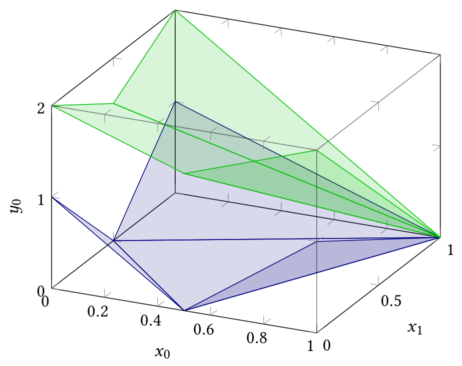


Figure 2: Plot of Output vs Input of $\mathcal{N}$ and $\mathcal{N}'$

soundly replace these nodes by another node that produces a larger output value using a *merge* operation that takes the maximum of incoming weights and sum of outgoing weights, as in Fig. 1b

Existing CEGAR approaches [12, 16, 38] start with this *fully merged* network, attempting to prove the property. However, notice that in our case, for the input $x_0 = 0, x_1 = 1$, the original network produces output 1, while the fully merged network produces 6. Thus, we have a spurious counterexample, and must refine the network by undoing some of the merges. Note that there are many possible choices one can make for un-merging nodes - one may un-merge c_0, c_1 or c_2. While existing CEGAR approaches [12, 16, 38] typically use the spurious counterexample to drive heuristics to chose which nodes to un-merge, it is a-priori hard to predict which choice will lead to an abstract network strong enough to prove the property. In this case, if we un-merge c_0, the resulting network yields a maximum upper bound of 8 on the output y_0, and therefore is not strong enough to prove the property. With this choice, subsequent refinement steps will end up un-merging all the merges, returning to the original network from Fig. 1a. However, if we un-merge c_2, the resulting network has output upper bounded by 4, enabling us to prove the property P_1. In general, the number of possible abstract networks grows exponentially after each refinement, while the number of these networks that are sufficiently strong to prove the property is limited.

Moreover, there may be several abstract networks that are not reachable via the merge and un-merge operations used by existing CEGAR approaches [12, 16, 38]. Consider the same network as in Fig. 1a, but this time with the property P_2 asking that the upper bound of the output be 2. In Fig. 2 the blue surface plots the output of the original network in Fig. 1a against the two inputs x_0 and x_1. Above it, the green surface is upper bounded by 2, and is piece-wise linear with three linear pieces and two linear boundaries separating these two pieces. Since it is possible to emulate such a surface with a *ReLU* DNN with 2 neurons, the existence of this surface suggests that we may be able to find an abstract network with 2 neurons that is strong enough to prove the required property. Motivated by this, we attempt to arrive at an abstraction that emulates this green surface. Indeed, Fig. 3 is a 2-neuron DNN that produces the green surface shown in Fig. 2. Starting from the concrete network in Fig. 1a, we can arrive at the abstract network in Fig. 3 by performing the following steps:

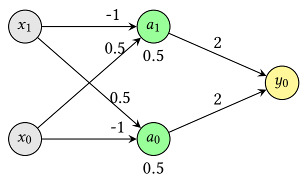
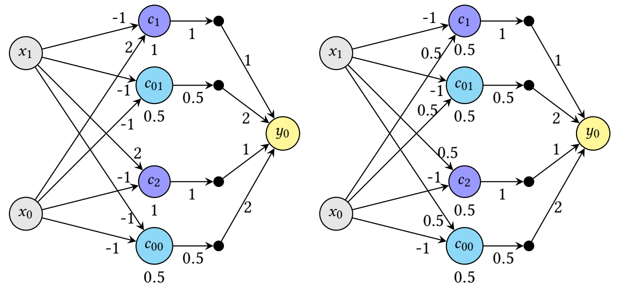


Figure 3: Optimal Abstract Network



(a) Equivalent Network (b) After Weight Replacement

Figure 4

We firstly create two copies two copies of c_0, c_{00} and c_{01}, multiplying the output of each by 0.5. Note that this does not change the behavior of the network, since the combined contribution of these copies are exactly the same as c_0. Then, we scale down the incoming weights and biases of c_{00} and c_{01} by 2 and scale the outgoing weights by 2. Again, we can do this without changing the behavior of the network since $\forall x, 2 \cdot ReLU(x) = ReLU(2x)$. The resulting network after these steps is shown in Fig. 4a.

Now, we notice that for both the nodes c_{00} and c_2, replacing the incoming weights and biases with $(-1, 0.5)$ and 0.5 would only lead to an increase in the output value of c_{00} and c_2 when $x_0, x_1 \in [0, 1]$, and thus only lead to an increase in the output value of the network y_0. Similarly, replacing the incoming weights and bias of c_{01} and c_1 by $(-1, 0.5)$ and 0.5 only leads to an increase in the output of the network. So, we can soundly replace the incoming weights and bias with these values to get Fig. 4b. Then, since c_{00}, c_2 and c_{01}, c_1 are pairs of neurons that calculate the exact same function, we can merge them by taking the sums of the outgoing edges to get the network in Fig. 3.

Note that in this case, no sequence of over-approximating merging and un-merging of neurons similar to those used in existing CEGAR approaches [12, 16, 38] will ever lead to the abstract network we have obtained. Thus, we have indeed found an abstraction that is not reachable using existing techniques.

We made several implicit choices during this construction, for example, the values we multiply the outputs of c_{00} and c_{01} with, the scaling factor, etc. Thus, the process we used to produce the abstract network may be thought of as being parameterized by these values. Based on this, we propose a novel approach where:

- We introduce a continuous and differentiable almost everywhere *parameterized abstract network* $\mathcal{N'}_k(\boldsymbol{x}; \boldsymbol{\theta})$, where the parameters $\boldsymbol{\theta}$ encodes the choices made and values selected.

This parameterization can encode a wider range of possible abstractions than those reachable via simple merge and un-merge operations.

- Then, we reduce the search for an optimal $\mathcal{N}'$ to a search for optimal $\boldsymbol{\theta}$ via *gradient descent.* This has two main benefits:
  - At each step of this gradient descent process, the combined behavior of every layer of the network is taken into account, which allows for a more global view of the abstraction search problem, increasing the chances of finding an optimal $\mathcal{N}'$.
  - This allows us to utilize the extensive engineering effort that has gone into efficient implementations of gradient descent frameworks like pytorch, and leverage specialized hardware like GPUs to accelerate the process.

We formalize the construction of this $\mathcal{N}'_k(\boldsymbol{x};\boldsymbol{\theta})$ in Sect. 2 and demonstrate via a preliminary set of experiments (Sect. 3) that out approach indeed may find abstractions not reachable via existing methods.

2 OUR APPROACH

Following the example in Sect. 1.1, we propose an approach based on constructing a parameterized abstract network $\mathcal{N}'_k(\boldsymbol{x};\boldsymbol{\theta})$ where the values of $\boldsymbol{\theta}$ encode the various choices that can be made during the construction. Intuitively, varying the value of $\boldsymbol{\theta}$ continuously will create a continuous change in the behavior of $\mathcal{N}'_k(\boldsymbol{x};\boldsymbol{\theta})$ as a function. This is not unlike a DNN, where changing the weights and biases continuously changes the behavior of the DNN. Therefore, we can *train* the $\mathcal{N}'_k(\boldsymbol{x};\boldsymbol{\theta})$ it using gradient descent similar to a manner in which a DNN is trained.

2.1 Construction of $\mathcal{N}'_k(\boldsymbol{x};\boldsymbol{\theta})$

As a pre-processing step we perform the *inc-dec* splitting technique described in [16], transforming the given network into an equivalent network so that the behavior of the output is monotonic with respect to every neuron in the hidden (non-input) layers. In the subsequent discussion, we assume that this step has already been performed, and $\mathcal{N}$ is the resulting network. We borrow terminology from [16] to talk about *inc* and *dec* nodes in $\mathcal{N}$.

We start with a budget k for the number of abstract nodes in each layer in $\mathcal{N}'_k(\boldsymbol{x};\boldsymbol{\theta})$. Then, the construction of $\mathcal{N}'_k(\boldsymbol{x};\boldsymbol{\theta})$ may be described in three stages:

2.1.1 Copying and Multiplying.

In general, we do not know which abstract nodes in $\mathcal{N}'$ should take the responsibility of abstracting which nodes in $\mathcal{N}$. Indeed, as in the example in Sect. 1.1, multiple abstract nodes may need to share the responsibility of abstracting a single concrete node, and vice versa. To allow for all these possibilities, we create one copy c_{ij} of each concrete node c_i for each abstract node a_j, representing a version of c_i that will get abstracted by a_j. We multiply the output of c_{ij} by learn-able parameters $\lambda_{ij} \in [0,1]$ representing the responsibility assumed by a_j in abstracting c_i. By maintaining that $\sum_j \lambda_{ij} = 1$ we ensure that the resulting network after this step is identical in behavior to $\mathcal{N}$. Thus, at this stage, we have not introduced any over-approximation.

Notice that in the case when all λ_{ij} are either 0 or 1, the concrete nodes get separated into disjoint groups, and each abstract nodes simply merges all nodes within the corresponding group. This is exactly what is done in existing CEGAR approaches [12, 16, 38], where these groups are referred to as *merge groups*. So our parameterization is able to express all abstractions possible via existing techniques, and potentially more.

2.1.2 α Multiplication.

Often, the weights of two concrete nodes which contribute similar values to later layers may have vastly different scales. This may create an issue for abstraction, because although they should be abstracted together into one abstract node, any sound abstraction would have to compensate for the large difference in the weights, leading to over-approximation the abstraction. To tackle this potential issue, we allow the input and output weights of each copy of each concrete node to be multiplied and divided by the same learn-able parameter, $\alpha > 0$. This allows for re-balancing of the scales of the weights of the concrete nodes. Since $ReLU(\alpha x) = \alpha \cdot ReLU(x)$ for $\alpha > 0$, at this stage the network is still identical in behavior to $\mathcal{N}$. Thus, we still have not introduced any over-approximation.

2.1.3 Sound Replacement by Abstract Nodes.

Finally, we replace the c_{ij} with weights $\boldsymbol{w_i}$ by abstract nodes a_{ij} with weights $\boldsymbol{w'_j}$ to get the abstract network. Each of these $\boldsymbol{w'_j}$ are learn-able parameters whose values are discovered via gradient descent. For such a replacement to be sound, it is sufficient for the following condition to hold at each *inc* node: $\mathbf{S} : \forall \boldsymbol{x} \in I, ReLU(\boldsymbol{x}.\boldsymbol{w'_j}) \geq ReLU(\boldsymbol{x}.\boldsymbol{w_i})$ and vice-versa for *dec* nodes. Here, I represents an interval covering all reachable values of $\boldsymbol{x}$, and may be obtained via interval propagation. This condition S builds on the sufficient condition derived in [16] by adding the additional constraint that the output value at each node only needs to increase for reachable input values, as opposed to all possible input values. This allows us to take into consideration several possible sound choices of $\boldsymbol{w'_j}$ that could not be considered by existing CEGAR approaches [12, 16, 38], and thus find several $\mathcal{N}'$ that may not be found by existing approaches.

However, S is hard to enforce within a gradient descent framework, since gradient descent does not allow for effective constrained optimization. Therefore, we relax the condition to: $\mathbf{R} : \forall \boldsymbol{x} \in I, ReLU(\boldsymbol{x}.\boldsymbol{w'_j}) + \delta_{ij} \geq ReLU(\boldsymbol{x}.\boldsymbol{w_i})$, where δ_{ij} represents a *residual value* that can be calculated via maximizing $ReLU(\boldsymbol{x}.\boldsymbol{w_i}) - ReLU(\boldsymbol{x}.\boldsymbol{w'_j})$ within I. Since this function is piece-wise linear with 4 pieces, a case analysis gives us an efficient closed-form solution to the maximization problem. Then exploiting the monotonicity properties of *inc-dec* split networks [16], we can propagate the contribution of these δ_{ij} to the output layer of the network in a manner similar to interval propagation, and add the resulting values to the output. This gives us a $\mathcal{N}'_k(\boldsymbol{x};\boldsymbol{\theta})$ that is a sound abstraction, while at the same time being able to potentially represent a much wider set of abstract networks.

Note that in the running example (Sect. 1.1, we chose $\boldsymbol{w'_j}$ in a way that S holds, therefore the residuals in that case were 0. While in practice it is unlikely that we would find $\boldsymbol{w'_j}$ so that the residuals are 0, even with non-zero residuals, one can still find a $\mathcal{N}'_k(\boldsymbol{x};\boldsymbol{\theta})$ that produces the green plot in Fig. 2.

2.2 Learning Abstractions via Gradient Descent

The $\boldsymbol{\theta}$ in $\mathcal{N'}_k(\boldsymbol{x};\boldsymbol{\theta})$ will consist of all the learn-able parameters described in the previous section, that is, λ_{ij}, α_{ij}, and $\boldsymbol{w}'_j$. Then, one can run gradient descent, optimizing all of these parameters together to find a good $\mathcal{N'}$. Note that the choice of various parameter values at each layer can affect the output of the network in complex and non-linear ways due to the effect of subsequent layers. It would be very hard to design CEGAR heuristics that takes this global non-linearity into account. On the other hand, gradient descent is typically able to handle such non-linearities. In fact, training *ReLU* DNNs itself is a problem where such non-linear dependencies between changes across layer is seen regularly. Therefore, we expect gradient descent to perform well on this search for $\boldsymbol{\theta}$.

We wish to find an $\mathcal{N'}$ that is strong enough to prove the required property, and to do so the loss function we run gradient descent on should capture how close $\mathcal{N'}_k(\boldsymbol{x};\boldsymbol{\theta})$ is to being strong enough to prove the property. We construct such a loss in the following section

2.2.1 Counterexample Loss. Say the property is of the form $\boldsymbol{x} \in I_i \Rightarrow \mathcal{N}(\boldsymbol{x}) \leq u_o$, where I_i are interval bounds for the input, and u_o is an upper bound for the output. Note that, following a construction detailed in [16], any general property may be reduced to this form. Then, we perform a uniform sampling of I_i, and, for each sample, calculate the amount by which the output condition is violated, that is, $max(\mathcal{N'}_k(\boldsymbol{x};\boldsymbol{\theta}) - u_o, 0)$. The average of this value over all the samples gives the loss.

This loss estimates the expected extent to which a random input would violate the property. Intuitively, if there are a large number of spurious counterexamples, or if there are very severe spurious counterexamples that are very far from satisfying the property, this loss will produce a high value. So running gradient descent on this loss will drive $\mathcal{N'}_k(\boldsymbol{x};\boldsymbol{\theta})$ towards becoming strong enough to prove the property.

Once training is complete, we extract a $\mathcal{N'}$ from the $\mathcal{N'}_k(\boldsymbol{x};\boldsymbol{\theta})$ produced by simply taking a network with nodes c_j and incoming weights w_j, and scale the output weights by $\sum_i \lambda_{ij}$. Note that this network behaves exactly equivalently to the $\mathcal{N'}_k(\boldsymbol{x};\boldsymbol{\theta})$, and so is unlikely to have a spurious counterexample. This $\mathcal{N'}$ only has k nodes in each layer, and thus the effort needed to check $\mathcal{N'} \vdash P$ will be less than $\mathcal{N} \vdash P$.

3 EXPERIMENTAL EVALUATION

Set-up. To demonstrate the practical feasibility of our method, we have implemented a basic prototype in Python. We used PyTorch as the gradient descent framework. Using this prototype implementation, preliminary experiments were run on a system with a 11th Gen Intel(R) Core(TM) i5-1145G7 @ 2.60GHz CPU and 16 GB of RAM. For our preliminary experimental evaluation, we did not use a system with a GPU.

Benchmarks. We chose a small (3x50) MNIST [13] classifier and three ϵ-robustness properties from the ERAN [2, 23, 24, 27, 28, 30, 31] benchmarks. Then, we set a budget of 30 abstract neurons per layer (representing a 40% reduction in size) and ran our gradient descent framework for 2000 steps to obtain abstract networks. As a baseline to compare against, we also ran the existing CEGAR based method from [16], stopping the CEGAR loop once the budget of 30 abstract neurons per layer is hit.

Table 1: Comparison of Counterexample Loss

Property	Existing Cex Loss	Our Cex Loss
Instance 1, $\epsilon = 0.008$	514.5	174.7
Instance 0, $\epsilon = 0.004$	217.8	110.9
Instance 0, $\epsilon = 0.031$	238.0	201.9

Results. In Table 1, we compare the counterexample loss (from Sect. 2.2.1) achieved by our networks and those produced using CEGAR. We find that our approach is consistently able to achieve a lower loss than CEGAR, indicating that we are able to find abstractions that existing techniques are not able to find.

4 RELATED WORK

Our results ally closely with work on syntactic structural abstraction based techniques [12, 16, 25, 38], which rely on merging neurons of a given network to obtain smaller abstract networks. However, unlike our approach, none of these techniques incorporate global behavioral information of the DNN to do the merges leading to potentially sub-optimal abstractions.

Conversely, heuristics based neural network compression techniques [10] and semantic abstraction techniques [1, 8] do account for the global semantic behavior of the network to achieve significant size reduction. Even so, these techniques fail either completely, as in the case of compression or partially, as in the case of semantic abstraction, to retain formal guarantees on the resultant network. In particular, the semantic abstractions in [1] lead to abstract networks using which only certain kinds of interval propagation based analyses can be lifted back to the concrete network. Similarly, [8] which uses linear combinations of neurons for network compression provides guarantees only on a finite data-set. In contrast, our approach provides clear soundness guarantees irrespective of any data-set (which may be finite or unbounded), property or proof technique used on $\mathcal{N'}$, similar to [16].

5 CONCLUSION AND FUTURE WORK

The paper puts forth the idea of parameterizing the space of DNN abstractions. Besides allowing us to encode the space of possible abstractions more generally, it also enables searching over these abstractions to find an optimal one using gradient descent. The proposed formalization and our preliminary evaluation demonstrate that this is a practicable approach. Going ahead, we plan to develop our prototype implementation into a robust tool, and validate our idea with thorough experimental evaluation. It will also be interesting to see how this idea can be extended to property-preserving compression of DNNs.

REFERENCES

[1] Ashok, P., Hashemi, V., Kretínský, J., and Mohr, S. Deepabstract: Neural network abstraction for accelerating verification. In *Automated Technology for Verification and Analysis - 18th International Symposium, ATVA 2020, Hanoi, Vietnam, October 19-23, 2020, Proceedings* (2020), D. V. Hung and O. Sokolsky, Eds., vol. 12302 of *Lecture Notes in Computer Science*, Springer, pp. 92–107.

[2] Balunovic, M., Baader, M., Singh, G., Gehr, T., and Vechev, M. T. Certifying geometric robustness of neural networks. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems*

2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada (2019), H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, Eds., pp. 15287–15297.

[3] Bassan, S., and Katz, G. Towards formal XAI: formally approximate minimal explanations of neural networks. In *Tools and Algorithms for the Construction and Analysis of Systems - 29th International Conference, TACAS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Paris, France, April 22-27, 2023, Proceedings, Part I* (2023), S. Sankaranarayanan and N. Sharygina, Eds., vol. 13993 of *Lecture Notes in Computer Science*, Springer, pp. 187–207.

[4] Bojarski, M., Testa, D. D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J., and Zieba, K. End to end learning for self-driving cars. *CoRR abs/1604.07316* (2016).

[5] Bunel, R., Lu, J., Turkaslan, I., Torr, P. H. S., Kohli, P., and Kumar, M. P. Branch and bound for piecewise linear neural network verification. *J. Mach. Learn. Res. 21* (2020), 42:1–42:39.

[6] Carlini, N., Katz, G., Barrett, C., and Dill, D. L. Provably minimally-distorted adversarial examples, 2018.

[7] Carlini, N., and Wagner, D. A. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017* (2017), IEEE Computer Society, pp. 39–57.

[8] Chau, C., Kretínský, J., and Mohr, S. Syntactic vs semantic linear abstraction and refinement of neural networks. In *Automated Technology for Verification and Analysis - 21st International Symposium, ATVA 2023, Singapore, October 24-27, 2023, Proceedings, Part I* (2023), É. André and J. Sun, Eds., vol. 14215 of *Lecture Notes in Computer Science*, Springer, pp. 401–421.

[9] Chen, X., Liu, C., Li, B., Lu, K., and Song, D. Targeted backdoor attacks on deep learning systems using data poisoning. *CoRR abs/1712.05526* (2017).

[10] Cheng, Y., Wang, D., Zhou, P., and Zhang, T. A survey of model compression and acceleration for deep neural networks. *CoRR abs/1710.09282* (2017).

[11] Clarke, E. M., Grumberg, O., Jha, S., Lu, Y., and Veith, H. Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM 50*, 5 (2003), 752–794.

[12] Cohen, E., Elboher, Y. Y., Barrett, C. W., and Katz, G. Tighter abstract queries in neural network verification. In *LPAR 2023: Proceedings of 24th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Manizales, Colombia, 4-9th June 2023* (2023), R. Piskac and A. Voronkov, Eds., vol. 94 of *EPiC Series in Computing*, EasyChair, pp. 124–143.

[13] Deng, L. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine 29*, 6 (2012), 141–142.

[14] Djavanshir, G. R., Chen, X., and Yang, W. A review of artificial intelligence's neural networks (deep learning) applications in medical diagnosis and prediction. *IT Professional 23*, 3 (2021), 58–62.

[15] Duong, H., Nguyen, T., and Dwyer, M. A dpll(t) framework for verifying deep neural networks, 2024.

[16] Elboher, Y. Y., Gottschlich, J., and Katz, G. An abstraction-based framework for neural network verification. In *Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part I* (2020), S. K. Lahiri and C. Wang, Eds., vol. 12224 of *Lecture Notes in Computer Science*, Springer, pp. 43–65.

[17] Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (2015), Y. Bengio and Y. LeCun, Eds.

[18] Katz, G., Barrett, C. W., Dill, D. L., Julian, K., and Kochenderfer, M. J. Reluplex: An efficient SMT solver for verifying deep neural networks. In *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I* (2017), R. Majumdar and V. Kuncak, Eds., vol. 10426 of *Lecture Notes in Computer Science*, Springer, pp. 97–117.

[19] Kurakin, A., Goodfellow, I. J., and Bengio, S. Adversarial examples in the physical world. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings* (2017), OpenReview.net.

[20] Mangal, A., Kalia, S., Rajgopal, H., Rangarajan, K., Namboodiri, V. P., Banerjee, S., and Arora, C. Covidaid: COVID-19 detection using chest x-ray. *CoRR abs/2004.09803* (2020).

[21] Marques-Silva, J., and Ignatiev, A. Delivering trustworthy AI through formal XAI. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022* (2022), AAAI Press, pp. 12342–12350.

[22] Moosavi-Dezfooli, S., Fawzi, A., and Frossard, P. Deepfool: A simple and accurate method to fool deep neural networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016* (2016), IEEE Computer Society, pp. 2574–2582.

[23] Müller, C., Serre, F., Singh, G., Püschel, M., and Vechev, M. T. Scaling polyhedral neural network verification on gpus. In *Proceedings of Machine Learning and Systems 2021, MLSys 2021, virtual, April 5-9, 2021* (2021), A. Smola, A. Dimakis, and I. Stoica, Eds., mlsys.org.

[24] Müller, M. N., Makarchuk, G., Singh, G., Püschel, M., and Vechev, M. T. PRIMA: general and precise neural network certification via scalable convex hull approximations. *Proc. ACM Program. Lang. 6*, POPL (2022), 1–33.

[25] Ostrovsky, M., Barrett, C. W., and Katz, G. An abstraction-refinement approach to verifying convolutional neural networks. In *Automated Technology for Verification and Analysis - 20th International Symposium, ATVA 2022, Virtual Event, October 25-28, 2022, Proceedings* (2022), A. Bouajjani, L. Holík, and Z. Wu, Eds., vol. 13505 of *Lecture Notes in Computer Science*, Springer, pp. 391–396.

[26] Palma, A. D., Bunel, R., Desmaison, A., Dvijotham, K., Kohli, P., Torr, P. H. S., and Kumar, M. P. Improved branch and bound for neural network verification via lagrangian decomposition. *CoRR abs/2104.06718* (2021).

[27] Ruoss, A., Baader, M., Balunovic, M., and Vechev, M. T. Efficient certification of spatial robustness. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021* (2021), AAAI Press, pp. 2504–2513.

[28] Salman, H., Yang, G., Zhang, H., Hsieh, C.-J., and Zhang, P. A convex relaxation barrier to tight robustness verification of neural networks. *Advances in Neural Information Processing Systems 32* (2019), 9835–9846.

[29] Shi, Z., Jin, Q., Kolter, J. Z., Jana, S., Hsieh, C.-J., and Zhang, H. Formal verification for neural networks with general nonlinearities via branch-and-bound. *2nd Workshop on Formal Verification of Machine Learning (WFVML 2023)* (2023).

[30] Singh, G., Gehr, T., Mirman, M., Püschel, M., and Vechev, M. T. Fast and effective robustness certification. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada* (2018), S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., pp. 10825–10836.

[31] Singh, G., Gehr, T., Püschel, M., and Vechev, M. T. An abstract domain for certifying neural networks. *Proc. ACM Program. Lang. 3*, POPL (2019), 41:1–41:30.

[32] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I. J., and Fergus, R. Intriguing properties of neural networks. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings* (2014), Y. Bengio and Y. LeCun, Eds.

[33] Wang, S., Zhang, H., Xu, K., Lin, X., Jana, S., Hsieh, C., and 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: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual* (2021), M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, and J. W. Vaughan, Eds., pp. 29909–29921.

[34] Xu, K., Zhang, H., Wang, S., Wang, Y., Jana, S., Lin, X., and Hsieh, C. Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. *CoRR abs/2011.13824* (2020).

[35] Zhang, H., Wang, S., Xu, K., Li, L., Li, B., Jana, S., Hsieh, C.-J., and Kolter, J. Z. General cutting planes for bound-propagation-based neural network verification. *Advances in Neural Information Processing Systems* (2022).

[36] Zhang, H., Wang, S., Xu, K., Wang, Y., Jana, S., Hsieh, C.-J., and Kolter, Z. A branch and bound framework for stronger adversarial attacks of ReLU networks. In *Proceedings of the 39th International Conference on Machine Learning* (2022), vol. 162, pp. 26591–26604.

[37] Zhang, H., Weng, T., Chen, P., Hsieh, C., and Daniel, L. Efficient neural network robustness certification with general activation functions. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada* (2018), S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., pp. 4944–4953.

[38] Zhao, Z., Zhang, Y., Chen, G., Song, F., Chen, T., and Liu, J. CLEVEREST: accelerating cegar-based neural network verification via adversarial attacks. In *Static Analysis - 29th International Symposium, SAS 2022, Auckland, New Zealand, December 5-7, 2022, Proceedings* (2022), G. Singh and C. Urban, Eds., vol. 13790 of *Lecture Notes in Computer Science*, Springer, pp. 449–473.