An Abstraction-Refinement Approach to Verifying Convolutional Neural Networks

Matan Ostrovsky[1], Clark Barrett[2], and Guy Katz[1](✉)

[1] The Hebrew University of Jerusalem, Jerusalem, Israel
matan.ostrovsky@mail.huji.ac.il, guykatz@cs.huji.ac.il
[2] Stanford University, Stanford, USA
barrett@cs.stanford.edu

Abstract. Convolutional neural networks (CNNs) have achieved immense popularity in areas like computer vision, image processing, speech proccessing, and many others. Unfortunately, despite their excellent performance, they are prone to producing erroneous results — for example, minor perturbations to their inputs can result in severe classification errors. In this paper, we present the Cnn-Abs framework, which implements an abstraction-refinement based scheme for CNN verification. Specifically, Cnn-Abs simplifies the verification problem through the removal of convolutional connections in a way that soundly creates an over-approximation of the original problem; it then iteratively restores these connections if the resulting problem becomes too abstract. Cnn-Abs is designed to use existing verification engines as a backend, and our evaluation demonstrates that it can significantly boost the performance of a state-of-the-art DNN verification engine, reducing runtime by 15.7% on average.

1 Overview

Deep neural networks (*DNN*s) have demonstrated a remarkable ability to solve extremely complex tasks [4,11]. However, they are also notoriously opaque to human engineers, and various errors have been demonstrated in real-world, state-of-the-art DNNs [12]. Such errors are a hindrance to the adoption of DNN-based methods in critical systems and have sparked great interest in DNN verification (e.g., [1,2,5,6,9,10,13], among many others). Unfortunately, the DNN formal verification problem is NP-complete even for simple neural networks and specifications [5], and emperically, it appears to become exponentially harder as the network size increases — making scalability a key challenge for DNN verification tools.

Here, we contribute to the ongoing effort to address this challenge with a new framework called Cnn-Abs, which uses an *abstraction-refinement* based approach for verifying *convolutional neural networks* (*CNNs*). A CNN is a particular type of DNN that uses *convolutions*: constructs that allow for a very

https://doi.org/10.1007/978-3-031-19992-9_25

compact representation of the DNN, and consequently enable engineers to overcome memory-related bottlenecks. CNNs have been shown to perform well in image processing and computer vision tasks [4,11] and are in widespread use. Existing verification tools can verify CNNs, but typically only by reducing them to the general, *fully connected* case, thus failing to leverage the built-in compactness of CNNs. Because the size of the DNN slows down its verification, such transformations are costly. In contrast, our proposed framework aims to utilize the special properties of a CNN in expediting its verification.

At a high level, given a verification query over a CNN, Cnn-Abs first creates an *abstract* network, with significantly fewer neurons, with the property that if the query can be proved for this smaller network, then it also holds for the original network. Notably, the abstract network that we construct is fully connected, and can thus be verified using existing technology. Further, because the verification complexity depends on the number of neurons and edges in the DNN, verifying this smaller network is faster than transforming the CNN into an equivalent, fully connected network and verifying it. Due to the abstraction procedure, verifying the smaller network might produce a spurious counterexample, in which case our framework refines the network and repeats the process.

The overall flow of Cnn-Abs is depicted in Fig. 1. Initially, Cnn-Abs applies bound propagation [10,13] to compute lower and upper bounds for all hidden neurons within the network. Then, it selects a set of neurons and *abstracts* them by removing their incoming edges and treating them as input neurons — which can take on values within the previously-computed range. Any other neurons that become disconnected from the network's outputs as a result are *pruned* entirely; the number of such neurons tends to be high, due to the nature of convolutional layers, where each neuron is only connected to a small number of neurons in following layers. A small illustrative example appears in Fig. 2. For a more thorough and precise description of the technique, as well as a proof of its soundness, see the full version of this paper [8].

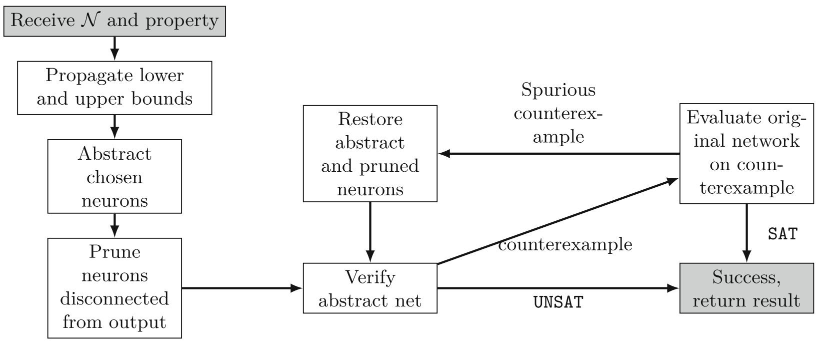
Receive $\mathcal{N}$ and property
Propagate lower and upper bounds
Abstract chosen neurons
Prune neurons disconnected from output
Restore abstract and pruned neurons
Spurious counterexample
Evaluate original network on counterexample
counterexample
SAT
Verify abstract net
UNSAT
Success, return result

Fig. 1. The suggested abstraction-refinement scheme.

Related Work. Abstraction-refinement techniques have been successfully applied in DNN verification [1,3,9], though these attempts were not particularly aimed at CNNs. Specific approaches to CNN verification have also been

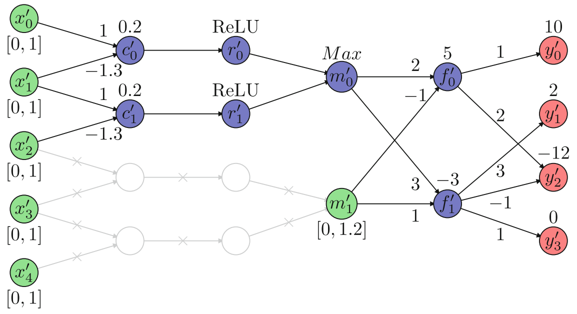

Fig. 2. A toy CNN, abstracted by disconnecting the edges leading to m_1 and pruning the neurons no longer connected to the output neurons (in gray). m'_1 is now treated as an input neuron, bounded by its computed bounds $0 \leq m'_1 \leq 1.2$.

proposed (e.g., [2,14]), but these do not focus on abstraction/refinement. For a more thorough discussion, see [8].

2 Design of Cnn-Abs

We implemented Cnn-Abs as a set of Python modules, available online.[1] Cnn-Abs currently accepts CNNs stored in Tensorflow format as input. The tool's main module, *CnnAbs.py*, implements the abstraction and refinement principles described in Sect. 1 and currently supports five different heuristics for iteratively applying refinement steps when spurious counterexamples are detected (see [8]). Cnn-Abs can be used in verifying arbitrary CNN properties, although it contains a specialized interface for verifying adversarial robustness properties [12], which are the most common kinds of properties in currently available verification benchmarks. The central classes in Cnn-Abs are:

The *CnnAbs* class, which implements Cnn-Abs's main functionality, and manages solving, logging, and heuristic configurations. It includes the following methods: (i) *solveAdversarial(model, abstractionPolicy, sampleIndex, distance)*: solves an adversarial robustness query on *model*, allowing input perturbations in an $\|\|\|_\infty$-ball of radius *distance* around an input sample whose index is *sampleIndex* in the data-set, using *abstractionPolicy* as the abstraction policy; (ii) *solve(model, modelTF, abstractionPolicy, property)*: solves *model*, which encodes both a network and a property, using the abstraction policy *abstractionPolicy*. For technical reasons, this method also receives a property object *property* and a Keras sequential model *modelTF*; and (iii) *propagateBounds(model)*: propagates lower and upper bounds for all neurons in the network and properties encoded in *model*. Cnn-Abs includes a novel technique for bound propagation across Max-Pooling layers — see [8] for details.

Policy Classes: Abstraction policies are implemented as classes inheriting from the *PolicyBase* class. Every child class is required to implement the *rankAbsLayer(model, prop, absLayerPredictions)* function. Its arguments are *model*, a

[1] https://drive.google.com/file/d/1En8f_I8LWFWQ6LFMHF9SSajszfOEKWF4.

property described in *prop*, and the assigned values of the abstracted layer for each point in the test-set. It returns the variable indices of the layer's neurons, sorted by their score: the first element is the least important and will thus be refined last. This modular design allows adding additional heuristics easily.

3 Evaluation

Setup. For our evaluation, we used the Marabou DNN verifier [6] as the backend DNN verifier within Cnn-Abs, and used MILP-based techniques [13] (enhanced to better handle Max-Pooling layers) for neuron bound computation.

We trained three convolutional networks on the MNIST digit recognition data-set [7]. The first network, network A, has two *convolution blocks* (a convolution layer followed by a ReLU layer and a max-pooling layer), another block consisting of a weighted-sum layer and a ReLU layer, and a final weighted-sum layer. When transformed into an equivalent, fully-connected model, it has a total of 2719 neurons and achieves a test-set accuracy of 93.7%. The second network, B, has the same layer sequence as A, but its convolution kernels are larger; consequently, it has 4564 neurons and achieves an accuracy of 96.2%. Network C is similar but has three convolution blocks instead of two; it has 4636 neurons and achieves an accuracy of 86.6%. Additional details appear in Appendix B of [8].

For specifications, we focused on adversarial robustness properties [12], which have become the de-facto standard for DNN verification benchmarks [10,13]. An adversarial robustness query consists of input x^0 and some $\varepsilon > 0$, and its goal is to prove that perturbations to x_0 within a ball of radius ε do not result in a change in the classification. For simplicity, we consider *targeted* adversarial robustness, where the goal is to prove that some perturbation cannot result in the input being classified as some target label l. We select l as the label that received the second-highest score when the DNN is evaluated on x_0.

Experiments. We ran a comprehensive comparison between vanilla Marabou and Cnn-Abs (with Marabou as a backend). All experiments were run with a 1-hour timeout, and individual verification queries on abstract networks were limited to 800 s. Our benchmarks consisted of our three CNNs and robustness properties with varying values of ε, 0.01, 0.02, and 0.03, over 100 input points, resulting in nine combinations and a total of 900 experiments. The results are depicted in Fig. 3. Excluding the $(C, 0.03), (B, 0.03), (A, 0.01)$ queries, in every category the abstraction-enhanced version solved more instances than vanilla and required a shorter total runtime. In the $(A, 0.01)$ category, both

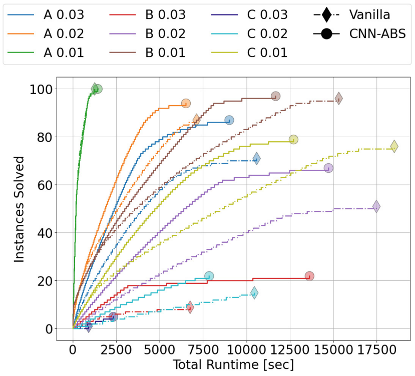

Fig. 3. Performance over different networks and ε values. Each query was ran in vanilla Marabou (dash-dotted line), and with Cnn-Abs (solid line).

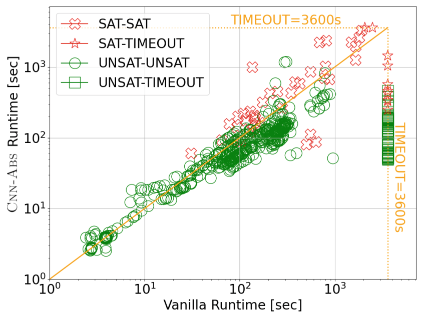

Fig. 4. CNN-ABS's runtime vs. vanilla Marabou's runtime, on a log scale. (Color figure online)

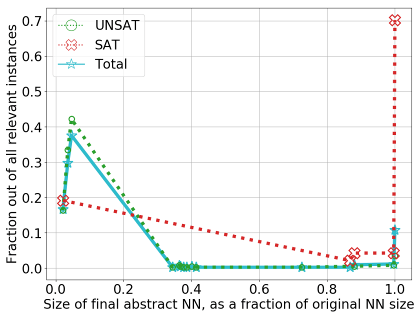

Fig. 5. The size of the abstract network when CNN-ABS terminates, compared to the size of the original network.

frameworks performed similarly; and in $(C, 0.03), (B, 0.03)$, CNN-ABS solved more instances, but at the cost of additional runtime. Aggregating the results over all instances solved by both frameworks, CNN-ABS's average runtime was 84.3% that of vanilla Marabou's runtime, and its median runtime 75.4% that of vanilla Marabou's. Additionally, CNN-ABS solved 1.13 times as many instances as vanilla Marabou. The exact numbers of instances solved, average runtimes, and median runtimes all appear in Appendix C.2 of [8]. This experiment clearly indicates the superior performance of CNN-ABS compared to the vanilla version.

Figure 4 depicts the runtime of CNN-ABS vs. vanilla Marabou for every query solved by at least one of the verifiers. There are 526 `UNSAT` points (green) and 49 `SAT` points (red). The results show that for `SAT` instances, the frameworks achieve similar performance; whereas for `UNSAT` instances, CNN-ABS performs significantly better, solving 61 instances that the vanilla version timed out on. We thus conclude that the CNN-ABS is particularly effective on `UNSAT` instances, presumably because `SAT` instances require multiple refinement steps.

In Fig. 5, we measure the number of refinement steps needed by CNN-ABS before arriving at an answer. Specifically, it depicts the size of the DNN in the final iteration of the abstraction/refinement algorithm, as a fraction of the size of the original DNN. The results differ significantly between `UNSAT` queries, which terminate with small networks and few refinement steps, and `SAT` queries, which often require the network to be refined back to the original DNN. The corollary is that slow, gradual refinement is ineffective; and that CNN-ABS performs better on `UNSAT` queries, as these can often be solved on small, abstract networks.

Conclusion. We presented a novel scheme for CNN verification, which uses abstraction-refinement techniques to effectively reduce network sizes and facilitate verification. Our tool, CNN-ABS, can be used with various existing DNN verifiers as backends. We regard this effort as a step towards more effective verification of real-world CNNs.

Acknowledgements. This work was partially supported by the Semiconductor Research Corporation, the Binational Science Foundation (grant numbers 2017662 and 2020250), the Israel Science Foundation (683/18), and the National Science Foundation (1814369).

References

1. Ashok, P., Hashemi, V., Křetínský, J., Mohr, S.: DeepAbstract: neural network abstraction for accelerating verification. In: Hung, D.V., Sokolsky, O. (eds.) ATVA 2020. LNCS, vol. 12302, pp. 92–107. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-59152-6_5
2. Boopathy, A., Weng, T.W., Chen, P.Y., Liu, S., Daniel, L.: CNN-Cert: an efficient framework for certifying robustness of convolutional neural networks. In: Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI), pp. 3240–3247 (2019)
3. Elboher, Y.Y., Gottschlich, J., Katz, G.: An abstraction-based framework for neural network verification. In: Lahiri, S.K., Wang, C. (eds.) CAV 2020. LNCS, vol. 12224, pp. 43–65. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-53288-8_3
4. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778 (2016)
5. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: an efficient smt solver for verifying deep neural networks. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 97–117. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_5
6. Katz, G., et al.: The marabou framework for verification and analysis of deep neural networks. In: Dillig, I., Tasiran, S. (eds.) CAV 2019. LNCS, vol. 11561, pp. 443–452. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25540-4_26
7. LeCun, Y.: The MNIST database of handwritten digits (1998). http://yann.lecun.com/exdb/mnist/
8. Ostrovsky, M., Barrett, C., Katz, G.: An abstraction-refinement approach to verifying convolutional neural networks (full version). Technical report. arxiv.org/abs/2201.01978
9. Prabhakar, P., Afzal, Z.: Abstraction based output range analysis for neural networks (2020). Technical report. arxiv.org/abs/2007.09527
10. Singh, G., Gehr, T., Puschel, M., Vechev, M.: An abstract domain for certifying neural networks. In: Proceedings of the 46th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL) (2019)
11. Szegedy, C.: Going deeper with convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1–9 (2015)
12. Szegedy, C., et al.: Intriguing properties of neural networks (2013). Technical report. arxiv.org/abs/1312.6199
13. Tjeng, V., Xiao, K., Tedrake, R.: Evaluating robustness of neural networks with mixed integer programming (2017). Technical report. arxiv.org/abs/1711.07356
14. Xu, J., Li, Z., Zhang, M., Du, B.: Conv-Reluplex: a verification framework for convolution neural networks. In: Proceedings of the 33rd International Conference on Software Engineering and Knowledge Engineering (SEKE) (2021)