Formal Analysis of Neural Network-Based Systems in the Aircraft Domain

Panagiotis Kouvaros[1], Trent Kyono[2], Francesco Leofante[1(✉)], Alessio Lomuscio[1], Dragos Margineantu[2], Denis Osipychev[2], and Yang Zheng[1]

[1] Imperial College London, London, UK
f.leofante@imperial.ac.uk
[2] Boeing Research and Technology, Seattle, USA

Abstract. Neural networks are being increasingly used for efficient decision making in the aircraft domain. Given the safety-critical nature of the applications involved, stringent safety requirements must be met by these networks. In this work we present a formal study of two neural network-based systems developed by Boeing. The Venus verifier is used to analyse the conditions under which these systems can operate safely, or generate counterexamples that show when safety cannot be guaranteed. Our results confirm the applicability of formal verification to the settings considered.

Keywords: Trustworthy AI · Formal verification · Neural networks

1 Introduction

Neural Networks (NN) have achieved impressive breakthroughs across several domains of science – see [20] for a comprehensive catalogue of success stories. Although being versatile and efficient, neural networks are known be susceptible to adversarial perturbations [10,27], i.e., imperceptible modifications to their inputs can lead to unexpected, and often inexplicable, consequences on the outputs produced by the network. This lack of reliability and transparency has hindered a wider adoption of neural networks in safety and security-critical settings.

Verification of neural networks (VNN) offers a promising solution to alleviate this problem. Since its inception, the field of VNN has experienced an impressive growth, with several methods and tools being developed for different classes of neural architectures and specifications [29]. In this paper, we consider two neural network-based industrial systems developed by Boeing for the DARPA Assured Autonomy programme and present results pertaining to their formal verification.

Work partly supported by the DARPA Assured Autonomy programme (FA8750-18-C-0095). The views, opinions and/or findings expressed are those of the author and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government. Approved for Public Release, Distribution Unlimited.

M. Huisman et al. (Eds.): FM 2021, LNCS 13047, pp. 730–740, 2021.
https://doi.org/10.1007/978-3-030-90870-6_41

Specifically, the VENUS verification toolkit [6] is used to analyse *local robustness* properties [16,25] of *(i.)* an object detection system trained for open category detection and *(ii.)* a neural controller trained to assist landing in non-towered airports. VENUS is used to identify the conditions under which these systems satisfy local robustness or generate counterexamples that comprehensively show the circumstances under which safety cannot be guaranteed.

Related Work. In recent years, there has been a large amount of work on reasoning on safety of neural networks. Formal verification for neural networks concerns checking whether a network satisfies an input/output specification defining the set of possible inputs and the set of admissible outputs, respectively.

Efficient methods exist to check this property for different classes of neural architectures, e.g., [2,4–6,12,17,24,26,28]. While an exhaustive review of existing approaches is outside the scope of this paper, we remark that only some methods are *complete*, i.e., if a counterexample exists, these approaches are guaranteed to find it. In contrast, *incomplete* approaches are normally based on various forms of convex relaxations of the network, and they can be conservative and thus fail to find some counterexamples. Despite considerable achievements in the field, the verification problem remains challenging for industry-scale networks. Indeed, the problem is known to be NP-complete [16], thereby motivating a very active area of research to improve the scalability of neural network verification.

An also recent body of work concerns the verification of closed-loop and cyber-physical systems equipped with learned controllers – see, e.g., [1,14,15,30]. Verification at this level typically entails checking richer specifications pertaining to the systems' temporal evolutions. Since verification over specifications expressed in standard temporal logics, such as LTL and CTL [13], is often undecidable [1], verification procedures in this context are typically restricted to checking *bounded* specifications which express properties concerning only a bounded number of execution steps.

2 Network Verification with the Venus Toolkit

VENUS is a state-of-the-art sound and complete verification toolkit for ReLU-based feed-forward neural networks. In this section, we give a formal description of the verification problem tackled by VENUS and an outline of the procedure in VENUS to solve it. We begin with the definition of feed-forward neural networks.

Feed-Forward ReLU Networks. A *feed-forward neural network* (FFNN) is a vector-valued function $\mathbf{f}\colon \mathbb{R}^m \to \mathbb{R}^n$ that composes a sequence of $k > 1$ *layers*, where each layer $\mathbf{f}^{(i)}$ is the composition of an input transformation and a non-linear *activation* function. In *linear* layers, $\mathbf{f}^{(i)}(\mathbf{x}^{(i-1)}) \triangleq \mathsf{act}^{(i)}(\mathbf{W}^{(i)}\mathbf{x}^{(i-1)} + \mathbf{b}^{(i)})$, where $\mathbf{x}^{(i-1)}$, $i > 0$, is the input to the i-th layer, $\mathbf{x}^{(0)}$ is the input to the network, $\mathsf{act}^{(i)}$ is the activation function of the i-th layer, and $z^{(i)} = \mathbf{W}^{(i)}\mathbf{x}^{(i-1)} + \mathbf{b}^{(i)}$ is the vector of *pre-activations* of the layer which is the affine transformation of the previous layer's output for a weight matrix $\mathbf{W}^{(i)}$ and a bias vector $\mathbf{b}^{(i)}$. In *convolutional* layers, $\mathbf{f}^{(i)}$ computes the convolution between

$\mathbf{x}^{(i-1)}$ a learned kernel. Venus focuses on the Rectified Linear Unit (ReLU) activation function, $\mathsf{ReLU}(\mathbf{z}^{(i)}) \triangleq \max(0, \mathbf{z}^{(i)})$, $i > 0$, where the maximum function is applied element-wise on $\mathbf{z}^{(i)}$.

Verification Problem. Given a feed-forward neural network, Venus answers positively or negatively as to whether the output of the network for every input within a linearly definable set[1] of inputs is contained within a linearly definable set of outputs. Formally, we have:

Definition 1 (Verification problem). *Given a FFNN $f\colon \mathbb{R}^m \to \mathbb{R}^n$, a linearly definable set of inputs $\mathcal{X} \subset \mathbb{R}^m$ and a linearly definable set of outputs $\mathcal{Y} \subset \mathbb{R}^n$, the verification problem concerns determining whether $\forall \mathbf{x} \in \mathcal{X}\colon \mathbf{f}(\mathbf{x}) \in \mathcal{Y}$.*

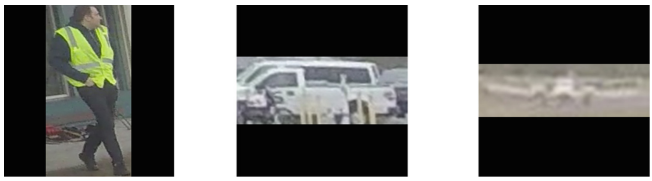
Among the many instantiations of the above verification problem, the *local adversarial robustness problem* is perhaps the most studied [16,25]. Answering the local adversarial robustness problem requires establishing whether all images within a norm-ball of a given image are classified equivalently by f, and can be instantiated by setting $\mathcal{X} = \{\mathbf{x}' \mid \|\mathbf{x} - \mathbf{x}'\|_p \leq \epsilon\}$ and $\mathcal{Y} = \{\mathbf{y} \mid \forall i \neq c\colon y_i < y_c\}$ for a given image $\mathbf{x}$ with class label c, perturbation radius $\epsilon \geq 0$ and norm $\|\cdot\|_p$.

Verification with Venus. Venus implements a verification method whereby the verification problem is translated into a Mixed Integer Linear Program (MILP). An MILP is an optimisation problem whereby an objective function is sought to be maximised subject to a set of linear constraints over real-valued and integer variables. The feasibility status of the MILP program associated with a verification problem, i.e., whether there is an assignment to the variables of the MILP that satisfies all constraints, has a strict correspondence to the satisfaction of the verification problem: the verification problem is satisfied if and only if its MILP program has no feasible solution. Venus makes use of the big-M method [3,23] to translate the verification problem into MILP and relies on Gurobi [11] as solving back-end. In addition, Venus implements a number of methods that aim at reducing the search space of feasible solutions that needs to be considered by Gurobi, including dependency analysis and input domain splitting [6].

3 Neural Network-Based Systems for the Aircraft Domain

In this section, we present our main results pertaining to the verification of two neural network-based industrial systems developed by Boeing. First we will focus on verifying the robustness of a high-dimensional image classifier for open object detection. Then, we will consider the verification of a landing assistant for non-towered airports.

[1] A linearly definable set is a set that can be expressed as a finite set of linear constraints over real-valued and integer variables.



(a) Person (b) Vehicle (c) Airplane

Fig. 1. Sample images in the data set for open-category detection.

3.1 Object Detection with Open Categories

Problem Description. In real-world applications, a neural network classifier is likely to encounter novel obstacles (i.e., open categories), such as novel types of ground vehicles and static objects. It is desirable that the classifier is able to detect these novel objects without previous training on the objects thereof while producing only a small number of false alarms, i.e., an alarm on the detection of a novel object even though the object is not novel. The classifier should also have a certain robustness against adversarial perturbations on its input. The following study is concerned with providing risk-bounded assurance against open category instances for an image classification system developed by Boeing.

Data Set and Neural Network Classifier. For this problem, Boeing has created a labelled data set that includes contiguous image sequences as aircraft approach novel obstacles or conditions. In particular, the data set contains three classes of objects: *ground vehicle*, *person*, and *airplane*. Figure 1 shows three sample images in each category. Each image has three channels, with 255×255 pixels in each channel. The image input is thus expressed as a three-dimensional matrix with a high dimension $255 \times 255 \times 3$. Boeing has also trained a large convolutional neural network (CNN) with roughly 10^5 neurons using two categories (vehicle and person) as the training set. The other category (i.e., airplane) is used as open objects for the neural network to test its performance of detecting novel objects.

The architecture of this large CNN model has two convolutional layers of 16 and 32 filters respectively (size 5×5 and 3×3, respectively), followed by one global average pooling layer, two fully connected layers with 128 and 2 neurons respectively, and one final softmax layer. The softmax layer normalizes the output into the interval $[0, 1]$ such that its outputs add up to one. The activation function in each layer is the standard ReLU operator. Therefore, the CNN model maps an image input $x \in \mathbb{R}^{255\times255\times3}$ to an output $y \in \mathbb{R}^2$ and classifies x using the following logic: if $y_1 \geq t_1$, it outputs the label *"vehicle"* with confidence y_1; if $y_2 \geq t_2$, it outputs the label *"person"* with confidence y_2; otherwise, it reports the input image as a novel object. The thresholds t_1 and t_2 were carefully chosen to balance the false alarm rate of (20%) and the overall prediction error rate (30%).

Table 1. Numerical results via VENUS on randomly selected 20 images

Radius ϵ	Robust		Non-robust		Time-out (7200 s)
	Number	Avg time (s)	Number	Avg time (s)	
1.0×10^{-4}	20	2500	0	—	0
1.0×10^{-3}	6	2720	2	4850	12
1.0×10^{-2}	0	—	10	4200	10

Robustness Verification Setup. We aim to verify the local robustness of the above CNN against adversarial perturbations on the image input. Formally, given a correctly classified image x_0 (person, vehicle, or open object) and a perturbation radius $\epsilon > 0$, we would like to certify

$$\text{for all } x \text{ s.t. } \|x - x_0\|_\infty \leq \epsilon \text{ we have } f(x) = f(x_0), \tag{1}$$

or to identify a counterexample $\hat{x}$ to falsify the condition above. In (1), $f(\cdot)$ denotes the CNN model. If (1) is true, then the neural network is certified to be locally robust for the image input x_0 against any adversarial perturbation up to ϵ. Otherwise, a counterexample can be found which can be used to augment the training set to improve the robustness of the model. Note that (1) is a particular instantiation of the verification problem in Sect. 2.

Extension of VENUS *and Numerical Results:* The images in this case study have a very high dimension, i.e., $x \in \mathbb{R}^{255\times255\times3}$, similar to the dimension of those in Imagenet [9]. We note that most existing complete robustness verifiers only test their verification performance on standard benchmarks, such as the MNIST dataset [19] or the CIFAR10 dataset [18], where the images have a much lower dimension ($28 \times 28 \times 1$ and $32 \times 32 \times 3$, respectively). In addition, the CNN model is large (approximately 10^5 neurons) as compared to the standard models considered in neural network verification. As a consequence, VENUS could not efficiently generate the MILP encoding of the robustness verification problem (1), either because of memory issues or because of exceeding the time consumption limit (see below). To overcome this, we have improved VENUS's implementation by exploiting the special structure of CNNs, i.e., that the value of a neuron depends only on a small subset of the neurons in the previous layers, to obtain a much more scalable MILP encoding method.

For the numerical experiments, we used a machine with an Intel Core i9 9900X 3.5 GHz 10-core CPU, 128 GB RAM running Fedora 30 with Linux kernel 5.3. We randomly selected 20 images from the data set that are correctly classified by the CNN model. We then used the extended version of VENUS to solve the robustness verification problem (1) with a timeout limit of 7200 s. To test the robustness of the CNN model, we varied the perturbation radius from 1.0×10^{-4} to 1.0×10^{-2}. Table 1 lists the results. We can see that the CNN model is robust with respect to the small perturbation (1.0×10^{-4}); in this case VENUS could return a positive certificate within 1 h. Concerning the bigger

perturbation radii, the CNN model becomes less robust; in this case VENUS could identify concrete counterexamples for 10 images out of 20 within 7200 s. In particular, VENUS identified a counterexample for the "vehicle" image shown in Fig. 1(b) for $\epsilon = 10^{-2}$. This counterexample fooled the CNN to classify it an open object. Finally, we note that since VENUS is a complete verifier the robustness of the CNN can by certified w.r.t the rest of the images by increasing the timeout.

3.2 Collision Avoidance for Landing at Non-towered Airports

Problem Description. Automated collision avoidance has become an essential component of every autonomous system. Whilst the safety of commercial aviation relies on centralized guidance from Air Traffic Control (ATC), private aviation must rely on onboard systems, especially when operating at non-towered airports. To prevent collisions, the regulator requires all vehicles to maintain a minimum safe horizontal separation which is advised via ATC commands, a set of rules defined in the pilot's handbook and the airport-specific traffic pattern.

This case study poses the collision avoidance problem as a data-driven detection problem. The goal is to detect the collisions when approaching landing by analysing the dynamic state of the airplane (Agent) and another vehicle (Intruder). The detection is done via an ML-based inference model trained on the previous (simulated) experience. A *positive* detection triggers the rejection of the landing and makes the Agent go for another landing approach. This rejection guarantees safe separation in the vertical space.

Simulation Environment. Boeing developed a lightweight Python simulation integrated with the OpenAI GYM framework [7] to facilitate the study of the problem described above. The surrogate environment simulates a 2D collision avoidance problem that mimics the real task (on the ground collision avoidance for the aircraft landing). The environment simulates the movements of multiple Intruder vehicles on a $10 \times 10\,\mathrm{km}^2$. The Agent has to either continue the automated landing or reject it to provide minimal horizontal separation. The environment incorporates a probabilistic behaviour model for the Intruder and the use of scripted configuration files: the former allows to randomize the trajectory of the Intruder vehicle and simulate a violation of the traffic rules; the latter enables quick reconfiguring of simulated scenarios.

The surrogate simulation imitates the dynamics of both the Agent and the Intruder vehicles selected for the experiment. All ground vehicles are represented by a simplistic dynamic model (mass-less Dubin's car). Each vehicle uses a base class that takes control inputs in form of steering and acceleration commands in return for its updated position and speed. For the Agent vehicle, we simulate the dynamics of a single-engine turboprop Cessna 208 Grand Caravan. The aircraft control is different from 2D car-like dynamics and does not have conventional brakes, acceleration, and steering. The effectiveness of the controls depends on the aircraft's altitude, airspeed, etc. A PID-based waypoint controller provides low-level control for the Agent and Intruders. The controller defines trajectories

Table 2. Robustness results for safe configurations and collision configurations.

	Safe configurations				Collision configurations			
ϵ	Robust	Avg time (s)	Not robust	Avg time(s)	Robust	Avg time (s)	Not robust	Avg time(s)
0.05	50	0.09	0	–	48	0.11	2	0.33
0.1	50	0.09	0	–	48	0.10	2	0.38
0.2	50	0.08	0	–	48	0.10	2	0.39
0.5	50	0.08	0	–	48	0.11	2	0.39
1	50	0.08	0	–	47	0.10	3	0.33
1.5	50	0.07	0	–	45	0.11	5	0.37

as lists of waypoints to be followed and adds a controlled level of noise to the controller state to make trajectories more natural.

Neural Network Classifier. The rejection action is commanded by a pre-trained neural network classification model that predicts the probability of collision between the Agent and Intruder vehicles projected on a 2D surface. The network is fully-connected and is trained using data gathered from the simulator. Each data sample represents a state vector that captures the position, heading and speed of the Agent and the Intruder vehicles. Samples are labelled either as 0 or 1, depending on whether the simulation run from which they were gathered resulted in a collision or not. The neural network has 8 input neurons, 2 hidden layers with 128 nodes each and ReLU activations, and 1 output neuron that uses a Sigmoid activation to produce a continuous collision probability within $[0, 1]$.

To evaluate the performance of the system, Boeing developed a run-time rejection system that utilizes the neural network to predict the probability of collision and compares this with a predefined threshold λ to command the rejection. The runtime system evaluates the chance of collision with each Intruder vehicle on the ground individually, and triggers a rejection if the inference model predicts a probability greater than λ for any Intruder.

Robustness Analysis. We conducted a formal analysis of the neural classifier to assess its local robustness against adversarial perturbations. Formally, given an input x_0 and a perturbations radius $\epsilon > 0$, we check if for all x such that $\|x - x_0\|_\infty \leq \epsilon$ it holds that $f(x) \diamond \lambda$[2], where $f(\cdot)$ denotes the network trained by Boeing and $\diamond \in \{\geq, \leq\}$ depending on whether the initial input x_0 belonged to a collision trajectory or not. Intuitively, the inference model should always compute prediction probabilities that are $\geq \lambda$ (resp. $\leq \lambda$) for inputs x in the vicinity of an x_0 for which a collision happened (resp. did not happen).

We collected 50 samples from the simulator for which the neural network correctly predicted collision and 50 for which collisions did not occur. Table 2 reports our results for a rejection threshold of $\lambda = 0.75$. The inference model appears to be robust for perturbations up to $\varepsilon = 1.5$ for inputs that belong to a

[2] Note that Venus does not support the Sigmoid function used in the output layer; we therefore use its inverse to compute the preimage of λ and compare this value with the pre-activation value of the output node.

safe trajectory, i.e., the model consistently predicted low collision probabilities for the perturbations considered. However decisions appear to be less robust in the other case; we hypothesise this is because of the fact that increasing ϵ may effectively create input configurations for which rejection commands should not be triggered. In all cases, Venus was able to solve the verification problem within less than a second.

Analysing Sensitivity to Single Inputs. Our previous analysis considered adversarial perturbations that are allowed to modify all input components at the same time. This corresponds to a scenario where all sensors of the Agent vehicle are subject to failure. However, more realistic failures may involve only a subset of the sensors; moreover, different sensor failures may have different consequences on the final prediction of the inference model.

To investigate this, we carry out a formal analysis where the inference model is verified against a variation of the local robustness property previously used. Namely, we apply adversarial ϵ-bounded perturbations to a single input component at the time and verify robustness in the resulting scenario. Similarly to the previous experiments, we sampled 50 instances for which collisions were flagged in the simulator and tested increasing perturbation radii. Table 3 reports sample results obtained for $\epsilon = 1$. Our analysis helped us identify that predictions seem to be comparably less robust to perturbations applied to the input component related to the speed of the Intruder vehicle. This can be explained by the fact that changes in speed may create scenarios where the Intruder proceeds slowly and thus collisions can be avoided. Again, Venus was able to solve all verification queries within less than a second.

Table 3. Sensitivity analysis results for perturbations applied to single input components and perturbation radius $\epsilon = 1$.

Component	Not robust	Avg time (s)
x_E	1	0.25
y_E	1	0.26
θ_E	1	0.26
s_E	1	0.25
x_I	1	0.24
y_I	1	0.24
θ_I	1	0.26
s_I	3	0.29

4 Conclusions and Outlook

We considered two neural network-based industrial systems developed by Boeing and showed how formal verification can be used to provide assurance guarantees for said systems. The Venus verification toolkit was successfully employed to formally analyse the behaviour of these models, providing proofs of safety or counterexamples to show when safety could not be guaranteed. Despite promising results, several challenges remain for the wider application of formal methods to neural network-based industrial systems.

Scalability is admittedly the biggest challenge in this arena, as industrial applications often require checking neural models that contain hundreds of thousands parameters, if not millions. While recent efforts in VNN have contributed

to impressive scalability improvements – especially as far as incomplete verification methods are concerned [21,22] – more work still needs to be done to address industrial-scale problems. This need becomes even more evident when verification is performed at system level, i.e., when the neural network-based system is considered in its entirety and closed-loop behaviours are analysed.

Another important direction where the expertise of the formal methods community may be determinant is that of formalising *richer specifications* beyond local adversarial robustness, especially in the case of perception systems such as the one studied in Sect. 3. Recent developments in VNN have addressed new specifications, such as robustness against semantic perturbations that can alter, e.g., saturation or contrast in an image [17], and geometric perturbations that apply transformations such as rotations to input images [5].

We conclude by highlighting that safe deployment of neural network-based systems will also require *runtime assurance methods* that are able to detect anomalous behaviour during execution. Different proposals have been made within the formal verification community, see, e.g., [8]; however several interesting open questions remain in this domain.

References

1. Akintunde, M., Botoeva, E., Kouvaros, P., Lomuscio, A.: Formal verification of neural agents in non-deterministic environments. In: Proceedings of the 19th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS20), pp. 25–33. IFAAMAS (2020)
2. Akintunde, M., Kevorchian, A., Lomuscio, A., Pirovano, E.: Verification of RNN-based neural agent-environment systems. In: Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI19), pp. 6006–6013. AAAI Press (2019)
3. Anderson, R., Huchette, J., Ma, W., Tjandraatmadja, C., Vielma, J.: Strong mixed-integer programming formulations for trained neural networks. Mathe. Programm., 1–37 (2020)
4. Bak, S., Tran, H.-D., Hobbs, K., Johnson, T.T.: Improved geometric path enumeration for verifying ReLU neural networks. In: Lahiri, S.K., Wang, C. (eds.) CAV 2020. LNCS, vol. 12224, pp. 66–96. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-53288-8_4
5. Balunovic, M., Baader, M., Singh, G., Gehr, T., Vechev, M.: Certifying geometric robustness of neural networks. In: NeurIPS19, pp. 15287–15297 (2019)
6. Botoeva, E., Kouvaros, P., Kronqvist, J., Lomuscio, A., Misener, R.: Efficient verification of neural networks via dependency analysis. In: Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI20), pp. 3291–3299. AAAI Press (2020)
7. Brockman, G., et al.: OpenAI Gym. arXiv preprint **1606**, 01540 (2016)
8. Cheng, C., Nührenberg, G., Yasuoka, H.: Runtime monitoring neuron activation patterns. In: Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE19), pp. 300–303. IEEE (2019)
9. Deng, J., Dong, W., Socher, R., Li, L., Li, K., Fei-Fei, L.: ImageNet: a large-scale hierarchical image database. In: Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR09), pp. 248–255. IEEE (2009)

10. Goodfellow, I., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572 (2014)
11. Gu, Z., Rothberg, E., Bixby, R.: Gurobi optimizer reference manual (2020). http://www.gurobi.com
12. Henriksen, P., Lomuscio, A.: Efficient neural network verification via adaptive refinement and adversarial search. In: Proceedings of the 24th European Conference on Artificial Intelligence (ECAI20), pp. 2513–2520. IOS Press (2020)
13. Huth, M.A., Ryan, M.: Logic in Computer Science: Modelling and Reasoning about Systems. Cambridge University Press (2000)
14. Ivanov, R., Carpenter, T., Weimer, J., Alur, R., Pappas, G., Lee, I.: Verifying the safety of autonomous systems with neural network controllers. ACM Trans. Embed. Comput. Syst. **20**(1), 7:1–7:26 (2021)
15. Johnson, T., et al.: ARCH-COMP20 category report: artificial intelligence and neural network control systems (AINNCS) for continuous and hybrid systems plants. In: Proceedings of the 7th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH20), pp. 107–139. EasyChair (2020)
16. 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
17. Kouvaros, P., Lomuscio, A.: Formal verification of CNN-based perception systems. arXiv preprint arXiv:1811.11373 (2018)
18. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images (2009)
19. LeCun, Y.: The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist/ (1998)
20. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (2015)
21. Li, L., Qi, X., Xie, T., Li, B.: SoK: certified robustness for deep neural networks. arXiv preprint arXiv:2009.04131 (2020)
22. Liu, C., Arnon, T., Lazarus, C., Barrett, C., Kochenderfer, M.: Algorithms for verifying deep neural networks. arXiv preprint arXiv:1903.06758 (2019)
23. Lomuscio, A., Maganti, L.: An approach to reachability analysis for feed-forward ReLU neural networks. arXiv preprint arXiv:1706.07351 (2017)
24. Narodytska, N., Kasiviswanathan, S., Ryzhyk, L., Sagiv, M., Walsh, T.: Verifying properties of binarized deep neural networks. arXiv preprint arXiv:1709.06662 (2017)
25. Pulina, L., Tacchella, A.: An abstraction-refinement approach to verification of artificial neural networks. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 243–257. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14295-6_24
26. Singh, G., Gehr, T., Mirman, M., Püschel, M., Vechev, M.: Fast and effective robustness certification. In: Advances in Neural Information Processing Systems (NeurIPS18), pp. 10802–10813 (2018)
27. Szegedy, C., et al.: Intriguing properties of neural networks. In: Proceedings of the 2nd International Conference on Learning Representations (ICLR14) (2014)
28. Tran, H.-D., Bak, S., Xiang, W., Johnson, T.T.: Verification of deep convolutional neural networks using ImageStars. In: Lahiri, S.K., Wang, C. (eds.) CAV 2020. LNCS, vol. 12224, pp. 18–42. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-53288-8_2

29. VNN-COMP: Verification of Neural Networks Competition (VNN-COMP20) (2020). https://sites.google.com/view/vnn20/vnncomp. Accessed 23 Mar 2021
30. Xiang, W.H., Rosenfeld, J., Johnson, T.: Reachable set estimation and safety verification for piecewise linear systems with neural network controllers. In: 2018 Annual American Control Conference (ACC), pp. 1574–1579. AACC (2018)