Kalpa Publications in Computing

Volume 16, 2023, Pages 6–28

Proceedings of the 6th Workshop on Formal Methods for ML-Enabled Autonomous Systems

Scaling Model Checking for DNN Analysis via State-Space Reduction and Input Segmentation

Mahum Naseer[1], Osman Hasan[2], and Muhammad Shafique[3]

[1] Technische Universität Wien (TU Wien), Austria
[2] National University of Sciences & Technology (NUST), Pakistan
[3] New York University Abu Dhabi (NYUAD), United Arab Emirates

Abstract

Owing to their remarkable learning capabilities and performance in real-world applications, the use of machine learning systems based on Deep Neural Networks (DNNs) has been continuously increasing. However, various case studies and empirical findings in the literature suggest that slight variations to DNN inputs can lead to erroneous and undesirable DNN behavior. This has led to considerable interest in their formal analysis, aiming to provide guarantees regarding a given DNN's behavior. Existing frameworks provide robustness and/or safety guarantees for the trained DNNs, using satisfiability solving and linear programming. We proposed FANNet, the first model checking-based framework for analyzing a broader range of DNN properties. However, the state-space explosion associated with model checking entails a scalability problem, making the FANNet applicable only to small DNNs. This work develops state-space reduction and input segmentation approaches, to improve the scalability and timing efficiency of formal DNN analysis. Compared to the state-of-the-art FANNet, this enables our new model checking-based framework to reduce the verification's timing overhead by a factor of up to 8000, making the framework applicable to DNNs even with approximately 80 times more network parameters. This in turn allows the analysis of DNN safety properties using the new framework, in addition to all the DNN properties already included with FANNet. The framework is shown to be efficiently able to analyze properties of DNNs trained on healthcare datasets as well as the well-acknowledged ACAS Xu networks.

Keywords: Bias, Formal Analysis, Input Node Sensitivity, Noise Tolerance, Robustness, State-Space Reduction

1 Introduction

The continuous improvement of Machine Learning (ML) systems, often wielding Deep Neural Networks (DNNs), has lead to an ever-growing popularity of these systems in real-world applications. These include face identification [48], speech recognition [20], anomaly detection [36], and even safety critical applications like autonomous driving [18] and healthcare [17, 3].

However, as observed in numerous recent works, the DNNs deployed in such systems are rarely *resilient*, i.e., they are extremely susceptible to misclassifying or arriving at an unsafe

output decision in the presence of slight input modifications [40, 32]. Earlier attempts to ensure correct functioning of these DNNs involved empirical approaches, for instance using gradient-based methods [40, 29] to identify the adversarial noise patterns that would lead the DNN to misclassify benign inputs. Although such attempts provide evidence to the lack of resilience of DNNs, they are insufficient to provide any guarantees regarding DNNs' resilience in the case when no adversarial noise is found.

To deal with the aforementioned problem, there has been a great interest towards the rigorous evaluation of DNNs, using formal verification, in recent years [41, 25, 7]. This usually involves checking resilience properties, like robustness and safety, of the DNNs using Satisfiability (SAT) checking or Linear Programming (LP). However, the exploration of formal approaches beyond SAT and LP, to analyze wider variety of DNN's properties, remains largely neglected.

To the best of our knowledge, our prior work on Formal Analysis of Neural Networks (FANNet) [31] was the first attempt to analyze DNN using SAT-based model checking. The framework was applicable for the verification and analysis of multiple DNN properties namely: robustness under constrained noise bounds, noise tolerance, training bias and input node sensitivity. *However, the FANNet framework provided limited scalability for formal analysis owing to the large Kripke structure it generated, even for relatively small DNNs. Hence, the applicability of the framework was limited to small DNNs only.*

This work introduces FANNet+ [1], an optimized model checking-based formal analysis framework for DNNs that overcomes the limitations of FANNet, and provides a significant improvement over FANNet in terms of scalability, timing-efficiency and the scope of DNN properties analyzed by the framework. In particular, **the novel contributions of this work are as follows**:

1. Providing novel state-space reduction techniques to reduce the size of the DNN's Kripke structure (Section 4.1).

2. Providing coarse-grain and input segmentation approaches, to split the input domain into manageable sub-domains, to aid model checking (Section 4.2).

3. Leveraging the framework for the automated collection of a large database of counterexamples, which assist in an improved analysis of the sensitivity of input nodes and the detection of training bias (Section 4.3).

4. Comparing the timing overhead of simulation-based testing, FANNet [31] and the proposed framework. The proposed framework reduces the timing-cost of model checking by a factor of up to 8000 (Section 6.1).

5. Making use of the input sub-domains to verify safety properties of DNNs, in addition to robustness under constrained noise, noise tolerance, training bias and input node sensitivity properties (Section 6.4).

6. Deploying the above techniques to demonstrate the applicability of the new framework on DNN case studies with up to 80 times more parameters than the ones used in FANNet, thereby illustrating better scalability and applicability to more complex networks (Sections 5 and 6).

[1] https://github.com/Mahum123/FANNetPlus

Paper Organization The rest of the paper is organized as follows. Section 2 provides an overview of the formal analysis approaches available in the literature for DNNs. Section 3 defines the basic DNN and model checking concepts and formalism relevant to this paper. Section 4 provides an overview for our proposed framework FANNet+ for the formal DNN analysis. Section 5 highlights the DNNs and datasets used to demonstrate the applicability of our framework for analyzing the various DNN properties. Section 6 presents the results of analysis for the given DNNs, also comparing timing-overhead of testing, FANNet and FANNet+. Finally, Section 7 concludes the paper.

2 Related Work

The earliest attempt [47] to analyze correct DNN behavior involved mapping an DNN to a look-up table. However, the approach lacked the sophistication to allow the analysis of intricate DNN properties. Recent works instead focus mainly on the use SAT solving and LP, which not only allow a better formal representation of DNNs but also the analysis of DNN properties like robustness and safety.

The earlier SAT-based DNN verification attempts [33, 34] focused on the safety properties of single hidden layer DNNs with logistic function as the activation. More popular SAT-based approaches include the ones involving layer-by-layer DNN analysis [22] and leveraging simplex algorithm to provide better splitting heuristics for piecewise linear activations during DNN verification [24, 25]. Some of the recent works [30, 11, 35] instead focus on the verification of Binary Neural Networks (BNNs), which are shown to be more power-efficient for the real-world applications [23] and reduce the complexity of the verification task over conventional DNNs by using binary valued parameters instead of the real numbered values.

Alternatively, the inherent support of LP for the linear constraints are leveraged in the LP-based DNN verification approaches, particularly for those using piecewise linear (for instance, the ReLU) activations [4, 27, 42, 38]. Branch and bound heuristics [43, 42, 8, 7] are also often deployed for an efficient implementation of LP. In addition, some works [12, 15, 27, 6, 41] also propose the use of Big-M technique, where an indicator variable is added to the linear constraints to distinguish the different linear regions of the piecewise linear activation function for DNN verification.

3 Preliminaries

This section describes the formalism of DNN architecture and properties relevant to this paper. The basics of model checking, sufficient to understand this work, are also provided. Interested readers may refer to [2] for more inclusive details on model checking.

3.1 Neural Network Architecture

Essentially, a DNN is an interconnection of nodes arranged in input, output and hidden layers [39]. This work focuses on feed-forward fully-connected DNNs.

Definition 1 (Feed-forward fully-connected neural network)**.** *Given input domain X^0, a feed-forward network $F : X^0 \rightarrow X^L$ maps the input to the output domain X^L such that the nodes in each layer k depends only on the inputs from the preceding layer $k-1$. This results in a loop-free network that can be represented by $x^L = F\ (x^0)\ = f^L\ (\ f^{L-1}\ (...\ f^1\ (x^0)\ ...\ ))$, where f^k encapsulates the linear and non-linear transformations for layer k. The network is*

also fully-connected if each neuron in each network layer is connected to every neuron in the adjoining layer.

Each layer of the network F involves two transformations: a linear and a non-linear transformation. Given N nodes in the layer k of the network, the linear transformation can be given by $y_j^k = b_j^k + \sum_{i=0}^{N} w_{ij}^k x_i^{k-1}$, where w_{ij}^k represents the weight connecting node i from layer $k-1$ to node j in the layer k and b_j^k represents the bias parameter value corresponding node j of the layer k.

The non-linear transformation maps output of the linear transformation via a non-linear activation function. This paper considers Rectified Linear Unit (ReLU) activation function, which is a piecewise linear function mapping negative inputs to zero, while using identity mapping the non-negative inputs, i.e., $x^k = max(0,\ y^k)$. The choice of DNN output is often based on the output with the highest values. As such, maxpool function is used as an activation function for layer L of the network, i.e., $x^L = max(y_1^L,\ y_2^L,\ ...,\ y_C^L)$.

3.2 Neural Network Properties

The following provides formalism to some essential DNN properties to ensure correct DNN behavior under varying input conditions (like the incidence of noise).

Definition 2 (Robustness)**.** *Given a network $F : X \rightarrow Y$, F is said to be robust against the small noise Δx if the application of the noise to an arbitrary input $x \in X$ does not change the output classification of input by the network, i.e., $\forall \eta \leq \Delta x : F(x+\eta) = F(x)$.*

Hence, by definition, a robust DNN does not misclassify inputs in the presence of predetermined noise Δx.

Definition 3 (Noise Tolerance)**.** *Given a network $F : X \rightarrow Y$, F is said to be have a noise tolerance of Δx_{max} if the application of any noise up to Δx_{max} to an arbitrary input $x \in X$ does not change the output classification of input by the network, i.e., $\forall \eta \leq \Delta x_{max} : F(x+\eta) = F(x)$.*

In other words, noise tolerance provides (an estimate of) the upper bound of the noise that the network F can withstand, without showing any discrepancies in its normal behavior, i.e., without compromising the robustness of the network.

Definition 4 (Training/Robustness Bias)**.** *Let $x_1, x_2 \in X$ b arbitrary inputs from the output classes $A, B \subset Y$, respectively. Given a network $F : X \rightarrow Y$, F is said to be biased towards the class A if the addition of noise Δx to x_1 does not cause any misclassification, but the addition of same noise to x_2 causes misclassification (i.e., $\exists \eta \leq \Delta x : (F(x_1+\eta) = A) \wedge (F(x_2+\eta) \neq B)$), for a significantly large number of noise patterns (η).*

Definition 5 (Input Node Sensitivity)**.** *Given a network $F : X \rightarrow Y$, with the input domain comprising of N nodes $X = [X_1, X_2, ..., X_N]$. Let $\Delta x = [\Delta x_1, \Delta x_2, ..., \Delta x_N]$ be the noise applied to an arbitrary input $x \in X$, where the noise applied to each node is within the same bounds. The input node α is said to be insensitive if the application of noise to the node $\forall \eta_\alpha \leq \Delta x_\alpha : x_\alpha + \eta_\alpha$ does not change the output classification of the input x.*

It must be noted that the sensitivity of an input node is analyzed independently of the remaining input nodes. This means, if the application of noise Δx_α to the node x_a does not change output classification of input x, the node is said to be insensitive regardless of what the noise (within the limits Δx) applied to the other input nodes may be.

Definition 6 (Safety). *Given a network $F : X \rightarrow Y$, let $[\underline{X}, \overline{X}] \subseteq X$ be the valid input domain, and $[\underline{Y}, \overline{Y}] \subseteq Y$ be the safe output domain. The network F is said to be safe if all inputs within the valid domain maps to the safe output domain, i.e., $\forall x \in [\underline{X}, \overline{X}]$. $F(x) = y$ s.t. $y \in [\underline{Y}, \overline{Y}]$.*

The safety property can also be subsumed by the concept of reachability, which requires any undesired output to be unreachable by all valid inputs.

3.3 Model Checking

Model checking is an automated formal verification approach, whereby specifications/desired properties are rigorously checked for a formal model/implementation given as a state-transition system, like a Kripke structure, as shown in Fig. 1(a).

Definition 7 (Kripke Structure). *Let AP be the set of all possible atomic propositions for a given system. The Kripke structure M for the system is then a tuple $M = (S, I, \delta, L)$ such that:*

- *S is the set of all the possible states in the formal model M,*
- *$I \subseteq S$ is the set of the possible initial states,*
- *$\delta \subseteq S$ X S is the transition relation between the states, and*
- *$L : S \rightarrow 2^{AP}$ is the labeling function that defines the AP valid for each state in M.*

In case the specification does not hold for the formal model, the model checker generally provides a path to property violation (i.e., a counterexample). The infinite paths, resulting from self-loops or cycles in the model, and may lead to the infamous *state-space explosion* problem [2]. Additionally, the explicit declaration of state variables [21]) may also lead to the generation of a large number states to encapsulate all possible behaviors in the formal model, resulting in a similar problem.

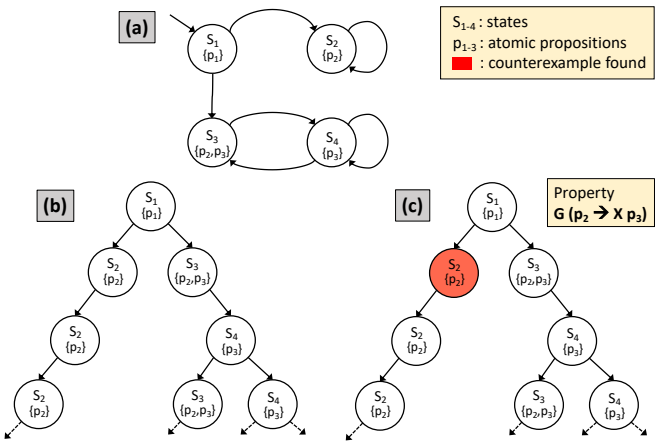

Figure 1: (a) A simple Kripke structure; (b) Unwinding the transition system to generate the computation tree for BMC; (c) searching the computation tree for the temporal property.

Numerous abstraction approaches are available in the model checking literature to reduce the size of the formal model, hence avoiding state-space explosion and scaling the model checking for larger systems. This paper leverages Bounded Model Checking (BMC) [5], Symbolic Model Checking (SMC) [28] and jump transitions [26] to optimize the formal models of DNNs. BMC limits the length of paths considered in the computation tree. SMC declares the state variables symbolically, rather than explicitly. Jump transitions compress the states in the model for which the labeling function L provides the same valid set of AP, as depicted in Fig. 2.

Temporal logic is often the preferred formalism for defining the system properties in model checking. It allows the notion of time to be expressed in the propositions, using temporal operators like: $\mathtt{X}\phi$ that holds true iff ϕ holds true in the *next* state, $\mathtt{G}\phi$ that holds true iff ϕ holds true in *every* state, and $\mathtt{F}\phi$ that holds true iff ϕ holds true *eventually* in the current or a following states.

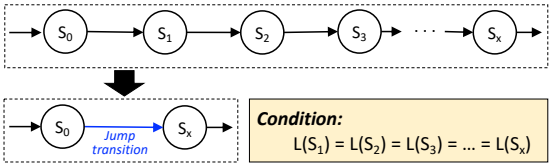

Figure 2: Compressing Kripke structure using jump transition.

Given the formal model M and a property defined in temporal logic, model checking involves unwinding the model (see Fig. 1(b)) into a computation tree and transversing through the tree to search for a violation of the property (as shown in Fig. 1(c)).

4 Proposed Optimizations for Formal DNN Analysis

The earlier model checking-based framework FANNet [31] provided an initial attempt for the analysis of DNN properties leveraging explicit state model checking, due to the efficient performance of explicit state model checking for software verification [16, 13, 9]. Additionally, the independence of our DNN models on any particular hardware technology/components made this choice of the type of model checking even more logical as the first-step in the domain of model checking-based DNN analysis.

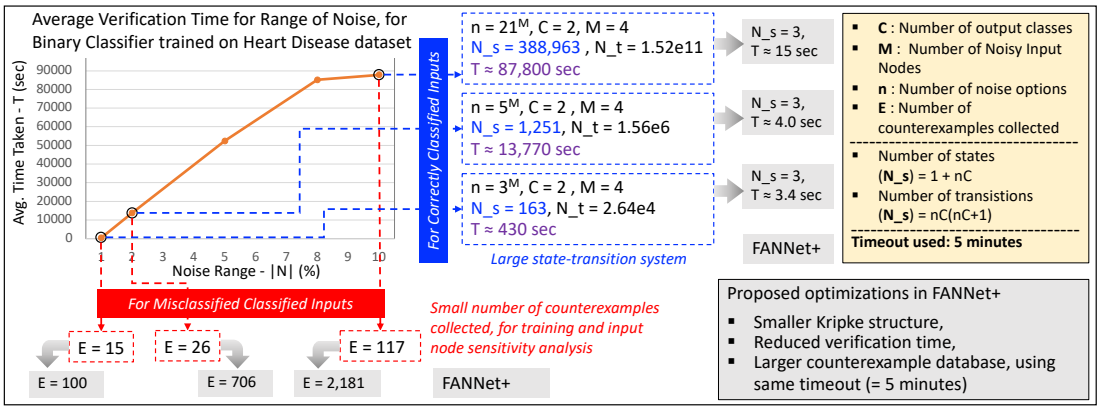

Figure 3: Formal analysis of DNN trained on heart disease dataset, which delineates limited scalability due to: (i) large Kripke structure, (ii) large verification time, and (iii) fewer counterexamples collected within timeout, using FANNet.

Consider a binary classifier trained on heart disease dataset (details in Sections 5 and 6). Not only is the size of formal model generated by FANNet large (also see Appendix H), but as shown in Fig. 3, its average verification time is also large. Moreover, for precise analysis of DNN's training bias and input node sensitivity, a large number of counterexamples is required. This means, the model checking needs to be repeated multiple times, while iteratively updating the specification of this large formal model. Again, as observed in the case study in Fig. 3 that running FANNet for a small time duration like 5 minutes does not provide a large database

of counterexamples for precise training bias and input node sensitivity analysis, hence limiting scalability and timing efficiency.

This section elaborates on our proposed optimizations for reducing the size of the Kripke structure leveraged by our enhanced framework, FANNet+[2]. Moreover, two input splitting approaches are also proposed, which reduce the size of the input domain and hence improve the scalability and timing-efficiency of DNN model. Hence, in addition to the multiple DNN properties analyzed using FANNet, the proposed framework also allows the analysis of DNN safety properties dealing with a large input domain, which was not viable earlier.

4.1 State-Space Reduction

FANNet uses explicit state model checking. Hence, the Kripke structure involves the enumeration of the different noise combinations from the available noise bounds defining the formal model. This leads to the generation of output states with identical AP. In contrast, FANNet+ proposes the use of SMC to reduces such identical state generation. The noise is added to the inputs symbolically, hence reducing the number of states in the model by a factor of approximately n. This can be viewed as the merging of states with identical valid AP in the formal DNN model (also see Appendix H).

Conjecture 1. *Given a model M with $S = [S_a, S_b, S_c]$ and $\delta = [(S_a, S_b), (S_b, S_b), (S_a, S_c), (S_c, S_c)]$ to be the set of all states and transition relations in the model, respectively, the states S_b and S_c can be merged iff $L(S_b) = L(S_c)$ holds.*

This results in a smaller Kripke structure, with a significantly smaller set of paths to transverse in the computation tree. This is further elaborated in Appendix I. This reduces the chances of the infamous state-space explosion associated with model checking. In addition, model checking can be viewed as a formal approach providing binary answers, i.e., either the specification holds (UNSAT) or it is violated (SAT) for the given DNN model. This allows further reduction of the number of states to be reduced by considering the output of the DNN to be either "correctly classified" or "misclassified".

4.2 Input Domain Segmentation

Noise tolerance analysis, as described above, makes use of seed inputs. Hence, model checker only verifies specification for one element of the input domain at a time. However, for DNN properties like safety, the verification often needs to be performed for the entire or subset of the input domain, potentially leading to state-space explosion. This paper proposes two approaches to resolve this problem: coarse-grain verification and random input segmentation.

4.2.1 Coarse-grain Verification

A rather straight-forward approach to verify a formal model with a large input domain is via sampling the input domain into discrete samples with regular intervals, i.e., with a constant step size. Depending on the size of the original DNN, input domain and the available computational resources available to the model checker, the size of the input intervals can be fine-tuned. For DNN specifications, for which the subset of input domain violating the specifications is large, the coarse-grain verification provides an efficient means to reduce the size of Kripke structure, while successfully finding any violations to the DNN specifications. However, for input domains where property violation is a rare occurrence, the approach may overlook the property violations.

[2]Extended version: on arXiv with the same paper title

4.2.2 Random Input Segmentation (RIS)

To address the challenge of dealing with a large input domain, this paper proposes the use of RIS, as shown in Fig. 4. The overall idea here is to divide the input nodes into two mutually exclusive sets: the *variable* and the *fixed* input node sets. The model checking is then carried out using the inputs from the *variable* set represented symbolically while the discrete samples from the *fixed* set are represented as constants in the model. Also see Appendix I for details.

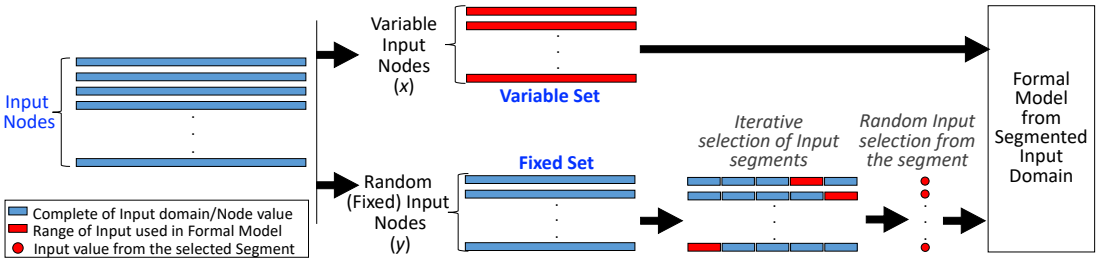

Figure 4: Overview of RIS: the input nodes are split into *variable* and *fixed* sets.

Since the splitting of the input nodes into the two sets and the following model checking are completely independent, this provides an opportunity for high degree of parallelism to the approach by dealing with different combinations of nodes from *fixed* and *variable* sets using a different core. This, in turn, reduces the timing-overhead of the analysis.

It must be noted that RIS is optimal for DNN verification (using FANNet+) iff $I^n > M^n . \frac{I!}{M!M'!}$ hold, where n is the number of noise options, I is the total number of input nodes, M is the number of nodes in the variable set and M' the number of nodes in the fixed set ($I = M + M'$).

4.3 FANNet+: Optimized Framework for Formal DNN Analysis

The proposed optimizations reduce the size of DNN's Kripke structure as well as split the input domain into more manageable sub-domains (as summarized in Fig. 5). Appendix J provides the summary and algorithm for the framework.

Initially, the formal model of the DNN is defined in the appropriate syntax of the model checker, as indicated by the blue box in Fig. 5. This requires the use of trained DNN parameters and architectural details of the network. Input is often normalized prior to being sent to the DNN. Some DNNs may in turn also use inverse-normalization for the DNN output. To validate the functional correctness of the model, the output of the model is checked for the known (testing) inputs.

The robustness verification of the validated formal model is carried out using seed inputs from the testing dataset and noise bounds, as explained in Section 4.1. This is expressed in the yellow box in Fig. 5. The noise bounds are iteratively reduced until the noise tolerance of the given network is obtained or the pre-defined timeout is reached, while collecting misclassifying noise to form the counterexample database. In case the property holds before reaching the timeout, the model checking is immediately terminated.

The counterexample database is then used in an empirical analysis to check the sensitivity of individual input nodes and detect any underlying training bias. The is shown by the green box in Fig. 5. The process is similar to the one used in FANNet. However, the improved timing-efficiency of our current framework allows model checking a large number of times within the

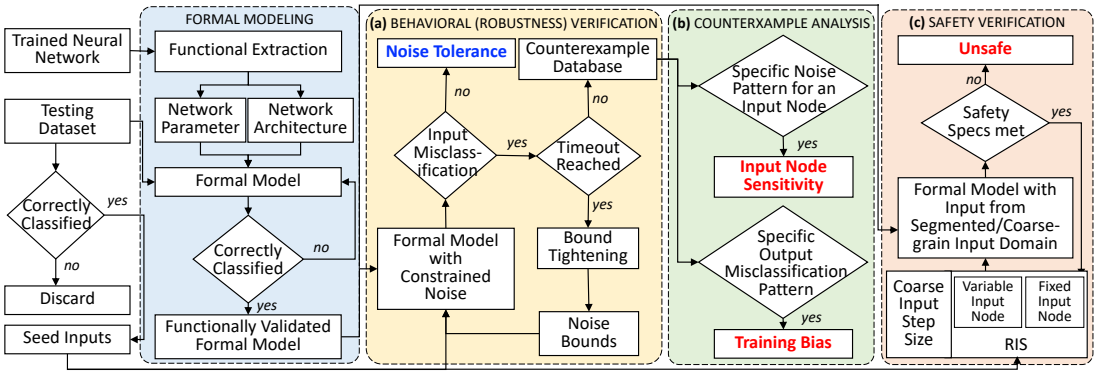

Figure 5: FANNet+ (a) provides robustness verification and noise tolerance, (b) uses the obtained counterexamples to analyze bias and input node sensitivity, and (c) enables safety verification.

pre-defined timeout. This provides a much larger counterexample database than was possible with FANNet, which in turn allows better analysis of the DNN properties in question.

DNN safety properties involve checking the DNN model with a large input domain. The optimizations proposed in Section 4.2.2 allow safety verification of the validated formal model, as shown in the orange box in Fig. 5. Here, either coarse-grain verification or verification with RIS can be opted. As mentioned earlier, the step size (in coarse-grain verification) and the size of input segments (in RIS) is chosen on the basis of the size of the original DNN, input domain and the available computational resources available to the model checker.

5 Experiments

This section describes the datasets corresponding trained DNNs used to demonstrate the application of FANNet+ for the analysis of DNN properties.

Leukemia Type Identification. The leukemia dataset [19] is a collection genetic attributes of Acute Lymphoblast Leukemia (ALL) and Acute Myeloid Leukemia (AML) patients (henceforth referred to as Labels 1 and 0, respectively). Minimum Redundancy and Maximum Relevance (mRMR) feature selection [1] was used to extract the 5 most important genetic attributes representing leukemia for the training. A feed-forward fully-connected DNN with single hidden layer and a total of 141 network parameters was trained for the dataset, with the training and testing accuracies of 100% and 94%, respectively. The following properties are analyzed for this DNN: robustness, noise tolerance, input node sensitivity, and training bias. The results of analysis are presented in Section 6.

Heart Disease Prognosis. The heart disease dataset [14] provides the records of continuous and discrete attributes enabling the prognosis of heart disease in the patients, i.e., patients with and without blood vessels narrowing (henceforth indicated by Label 1 and 0, respectively). A feed-forward fully-connected DNN with 3 hidden layers, and a total of 622 network parameters was trained for this dataset. The training and testing accuracies of the DNNs were 90% and 86%, respectively. To imitate real-world case scenarios, where the noise is more likely to affect

continuous variables as compared to discrete ones, noise was applied to the input nodes with continuous variables. This was in turn used for the analysis of robustness (under constrained noise), noise tolerance, input node sensitivity, and training bias. The results of the analysis are presented in Section 6.

Airborne Collision Avoidance System (ACAS Xu). Airborne Collision Avoidance System (ACAS Xu) comprises of 45 networks that make use of the trajectories of ownship and intruder to ensure safety while maneuvering the ownship. Each DNN is feed-forward and fully-connected, with ReLU activation function, 6 hidden layers and $13,305$ parameters. Let i_1–i_5 be the DNN inputs corresponding distance between ownship and intruder, and their heading angles and speeds, while o_1–o_5 be the ownship's maneuvering decisions namely clear-of-conflict, weak left/right and strong left/right. The DNN's output decision corresponds to the output class with minimal value. We consider 4 well-studied saftey properties of ACAS Xu networks in our analysis (also given in Appendix K). As indicated in Section 4, the analysis of safety properties involves a formal model with large input domains. Hence, coarse-grain verification and RIS are used for the analysis of ACAS Xu networks.

6 Results and Analysis

We use DNNs highlighted in the previous section to perform formal DNN analysis on CentOS-7 systems running on Intel Core i9 − 9900X processors at 3.50GHz. The proposed framework FANNet+ is implemented in Python, C++ and MATLAB, and uses NuXmv model checker [10] back end. The tools Reluplex [24] and Marabou [25] are implemented on virtualbox running Ubuntu 18.04, for comparison. The timeout used for DNNs trained on leukemia and heart disease datasets is 5 minutes, while a timeout of 2 hours is used for ACAS Xu networks. It must be noted that the objective of comparison between FANNet+ and SMT-based tools is to establish the consistency of results, not to compare timing-overhead since model checking is still fairly a new direction for DNN analysis.

6.1 Computational Overhead for Testing and FANNet

Testing is generally considered more user friendly, as compared to model checking. However, the results for model checking are more rigorous, and hence provide more reliable behavioral guarantees than testing. We compared the performance of FANNet with MATLAB-based testing. For testing, we define a matrix for all possible noise combinations, for a predefined noise bounds, before initializing the test. On the other hand, the model checker searches for noise combinations, non-deterministically, at run-time. Both experiments are based on the small DNN trained on the Leukemia dataset, as described in the previous section, since the timing and memory overhead of testing increases rapidly for large noise bounds for larger DNNs. Both experiments run on the same seed inputs.

Considering the time taken until the termination of both experiments, the average timing requirement of the FANNet, although significantly higher than testing's for the given experiment, increases at a slower rate than that for testing. This trend is illustrated in Fig. 6(a). On the other hand, the increase in average memory requirements of FANNet also increases at a significantly slower rate than testing, as shown in Fig. 6(b). The trends for the average time and memory requirements of both experiments indicate that the *strength of model checking is more prominent for larger DNNs.*

6.2 Behavioral (Robustness) Verification and Noise Tolerance Determination

As indicated earlier, the Kripke structure model for DNNs generated by FANNet is quite large, owing to the enumeration of noise applied to seed inputs. In contrast, the formal model generated by the FANNet+ is considerably smaller due to the optimizations for state-space reduction used. Hence, the framework provides same results (i.e., SAT or UNSAT) for both robustness verification and noise tolerance determination. However, the execution times of the frameworks are significantly different.

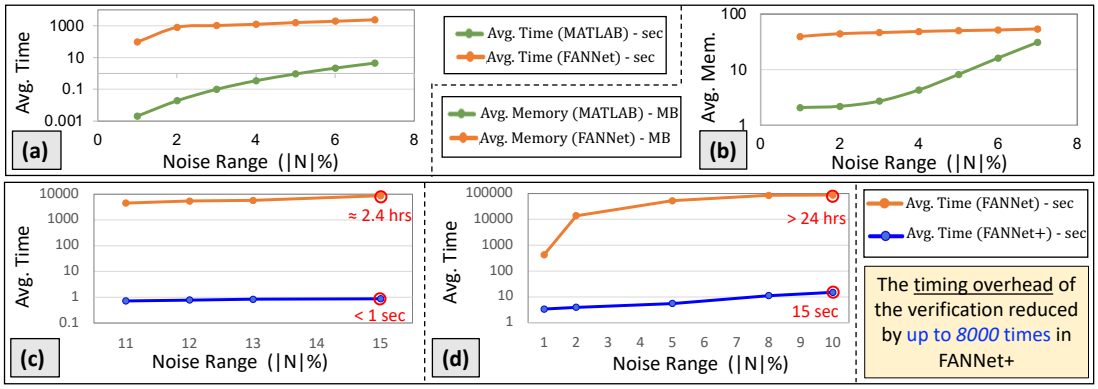

Figure 6: (a) Timing, and (b) memory overhead comparison between exhaustive testing (using MATLAB) and FANNet [31] for DNN trained on Leukemia dataset, b. Comparison of timing overhead using (c) FANNet and (d) FANNet+ for DNN trained on heart disease dataset.

FANNet versus FANNet+ – We run both frameworks for the DNNs trained on the leukemia and the heart disease datasets. Figs. 6 (c) and (d) shows the average execution time for verifying DNNs for both datasets, under identical noise bounds. Both frameworks lead to the same noise tolerance for the DNNs. However, the execution time for property verification is significantly larger for FANNet, as shown in Figs. 6 (c) and (d). For the given DNNs, FANNet+ provides a significant improvement over FANNet in terms of timing-cost, by reducing the timing overheard by a factor of up to 8000 times. This makes FANNet+ suitable for the analysis of relatively larger DNNs.

6.3 Counterexample Analysis for detecting DNN's Training Bias and Input Node Sensitivity

With the reduction in timing overhead, it is possible to run the framework for small timeout, and yet be able to collect a large database of misclassifying noise vectors, i.e., counterexamples. Analyzing the DNN outputs for these counterexamples provide insights regarding training bias and input node sensitivity, as shown in Figs. 7 and 8.

As observed in Fig. 7, for the DNN trained on leukemia dataset, even when the large noise is applied to inputs, ALL inputs are rarely misclassified to AML. From the description of available datasets (discussed in Section 5), it is known that training dataset for leukemia is significantly imbalanced, i.e., approximately 70% inputs belong to ALL. Hence, the obtained results indicate a strong bias in the resulting trained DNN likely due to the imbalance in dataset.

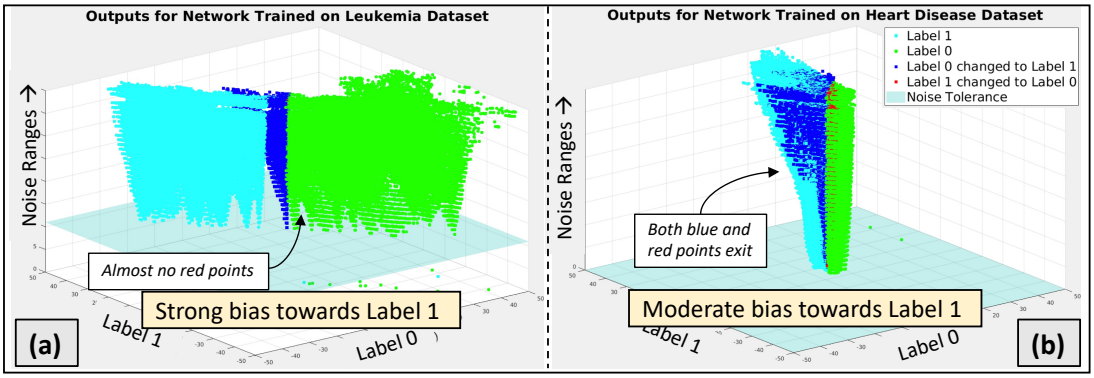

Figure 7: Output classification of DNNs trained on (a) Leukemia dataset and (b) Heart disease dataset, mapped with respect to the input noise, using FANNet+. An unequal number of red and blue points indicates a bias in trained networks.

On the contrary, for the DNN trained on heart disease dataset, outputs from both classes are misclassified even though the misclassifications from Label 0 to Label 1 are more likely. The training dataset for heart disease does not have the same class imbalance as the leukemia dataset. This likely accounts for the relatively moderate bias observed in the DNN.

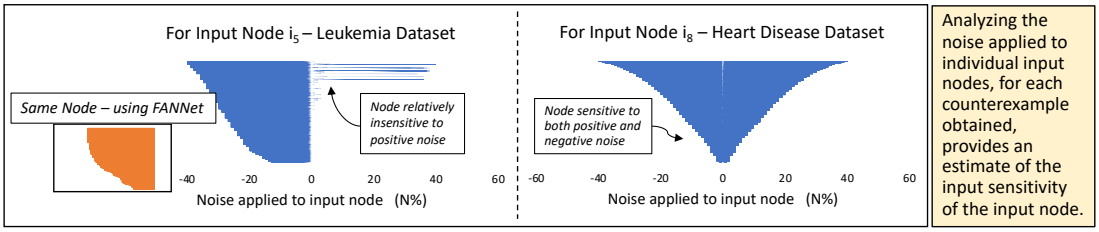

Figure 8: Plots of noise applied to individual input nodes (using FANNet+), that lead to misclassification. Unlike FANNet (in orange), which shows *complete* insensitivity to positive noise, FANNet+ provides more accurate results.

For the misclassifying noise vectors, observing noise applied to the individual input node provides insights to the sensitivity of the input nodes. For instance, input node i_5 from the DNN trained on the leukemia dataset, shown in Fig. 8, was observed to delineate rarely any misclassifications for the positive values of the applied noise. This suggests the node to be insensitive to positive noise, for the trained DNN. However, this was not the case for any of the input nodes for the DNN trained on heart disease dataset. For instance, node i_8 in Fig. 8 gives an example of a typical input node that is not sensitive to any specific input noise.

FANNet versus FANNet+ – We performed the same experiment for DNN trained on Leukemia using FANNet. As discussed earlier, due to the large timing overhead of FANNet, the counterexamples obtained compose a smaller counterexample database. Hence, even though both FANNet and FANNet+ indicate the DNN to be biased towards Label 1 and input node i_5 to be insensitive to noise, the results obtained by FANNet+ are more precise. For instance, given the noise bounds of 40%, Fig. 8 indicate node i_5 to be completely insensitive to

any positive noise. However, FANNet+ predicts the node to be relatively insensitive to positive noise, but may still lead to DNN misclassification with certain noise patterns. Hence, the larger counterexample database with FANNet+ is able to provide more precise results for training bias and input node sensitivity, as compared to FANNet.

6.4 Safety Verification

For DNN properties, like safety, which involve large input domain, coarse-grain verification with input step sizes for ACAS Xu networks as shown in Table 1, is first used. The verification for each property, for each DNN, takes only a few seconds to complete. As mentioned earlier in Section 4, coarse-grain verification is suitable when large segments of input domain violate the DNN property, which does not hold true for ACAS Xu networks.

Node	P1	P2	P3	P4
i_1	$10,000$	$10,000$	10^{-9}	1000
i_2	1	1	10^{-9}	10^{-1}
i_3	1	1	10^{-9}	–
i_4	500	500	10^{-9}	400
i_5	500	500	10^{-9}	400

Table 1: Step sizes used for coarse-grain verification of ACAS Xu safety properties.

Next, RIS is deployed. Based on the chosen *variable* and *fixed* sets, and the input segments of the nodes from *fixed* set, verification of each DNN is split into multiple smaller verification sub-problems. As stated earlier, these verification problems are independent and hence, given sufficient computation resources, they can potentially all be verified in parallel. If any of these sub-problems return a SAT, the property is said to be violated for the DNN. Likewise, if any of the sub-problems times out, the entire property is considered to have timed out, since the model checker is unable to find a result for a sub-section of the input domain. Otherwise, the verification is deemed to have terminated without a solution. It must be highlighted here that the framework does not return UNSAT since the use of *fixed* set introduces a certain degree of incompleteness, with respect to the input domain verified for the property. However, this notion of incompleteness is not the same as the *incompleteness of the formal model* observed in numerous state-of-the-art [37, 46, 38, 45], which may lead to false positive. On the other hand, FANNet+ does not lead to false positives in the results.

The results of the safety verification are compared to those obtained from Reluplex and Marabou. For all problems that provide SAT results with Reluplex and Marabou, FANNet+ is also able to find the property violation unless the verification times out. It is interesting to note that FANNet+ was also able to find a property violation for the network 2_2 with property P4, although both Reluplex and Marabou return UNSAT for the stated property. We confirmed the validity of the obtained counterexample using `Maraboupy`. Detailed results using Reluplex, Marabou and FANNet+ are provided in the Appendix K.

FANNet versus FANNet+ – As opposed to FANNet+, which leverages coarse-grain verification and RIS to split input domain prior to verification, FANNet relies on bounds of entire input domain. Hence, the verification of safety properties of ACAS Xu networks was infeasible with FANNet, even with a timeout of 24 hours.

7 Conclusion

Formally analyzing deep neural networks (DNNs) is an actively sought research domain. Towards this end, a model checking-based framework FANNet was introduced earlier, which could verify robustness of trained DNNs to constrained noise, and also analyze DNNs' noise tolerance,

input node sensitivity and any underlying training bias. This paper proposes a more scalable model checking-based framework FANNet+, which improves over FANNet in multiple directions. Firstly, it employs a state-space reduction strategy to ensure a more manageable size of the resulting Kripke structure. Secondly, it introduced input domain segmentation approaches, including our novel random input segmentation approach, which uses a divide-and-conquer strategy to break the verification problem into smaller sub-problems. Thirdly, apart from the multiple DNN properties mentioned earlier, the new framework FANNet+ extends the scope of DNN analysis by adding the verification of safety properties to the framework. The aforementioned improvements were found to scale the framework for DNN's with up to 80 times more DNN parameters, while reducing the timing overhead by a factor of up to 8000 for the considered examples. The applicability of the framework was shown using multiple DNNs, including the well-known ACAS Xu benchmark.

References

[1] Khan et al., S.: A novel fractional gradient-based learning algorithm for recurrent neural networks. CSSP **37**(2), 593–612 (2018)

[2] Baier, C., Katoen, J.P.: Principles of model checking. MIT press (2008)

[3] Bassi, P.R., Attux, R.: A deep convolutional neural network for covid-19 detection using chest X-rays. Res. on Biomed. Engineering pp. 1–10 (2021)

[4] Bastani, O., Ioannou, Y., Lampropoulos, L., Vytiniotis, D., Nori, A., Criminisi, A.: Measuring neural net robustness with constraints. In: Proc. NeurIPS. pp. 2613–2621 (2016)

[5] Biere, A., Cimatti, A., Clarke, E.M., Strichman, O., Zhu, Y.: Bounded model checking. Handbook of satisfiability **185**(99), 457–481 (2009)

[6] Botoeva, E., Kouvaros, P., Kronqvist, J., Lomuscio, A., Misener, R.: Efficient Verification of ReLU-based Neural Networks via Dependency Analysis. In: Proc. AAAI. pp. 3291–3299 (2020)

[7] Bunel, R., Lu, J., Turkaslan, I., Kohli, P., Torr, P., Mudigonda, P.: Branch and bound for piecewise linear neural network verification. JMLR **21**(2020) (2020)

[8] Bunel, R.R., Turkaslan, I., Torr, P., Kohli, P., Mudigonda, P.K.: A unified view of piecewise linear neural network verification. In: Proc. NeurIPS. pp. 4790–4799 (2018)

[9] Buzhinsky, I., Pakonen, A., Vyatkin, V.: Explicit-state and symbolic model checking of nuclear i&c systems: A comparison. In: Proc. IECON. pp. 5439–5446. IEEE (2017)

[10] Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., Mover, S., Roveri, M., Tonetta, S.: The nuxmv symbolic model checker. In: Proc. CAV. pp. 334–342 (2014)

[11] Cheng, C.H., Nührenberg, G., Huang, C.H., Ruess, H.: Verification of Binarized Neural Networks via Inter-Neuron Factoring. In: Proc. VSTTE. pp. 279–290. Springer (2018)

[12] Cheng, C.H., Nührenberg, G., Ruess, H.: Maximum resilience of artificial neural networks. In: Proc. ATVA. pp. 251–268. Springer (2017)

[13] Cook, B., Kroening, D., Sharygina, N.: Symbolic model checking for asynchronous boolean programs. In: Proc. SPIN. pp. 75–90. Springer (2005)

[14] Dua, D., Graff, C.: UCI machine learning repository (2017), http://archive.ics.uci.edu/ml

[15] Dutta, S., Jha, S., Sankaranarayanan, S., Tiwari, A.: Output range analysis for deep feedforward neural networks. In: Proc. NFM. pp. 121–138. Springer (2018)

[16] Eisner, C., Peled, D.: Comparing symbolic and explicit model checking of a software system. In: Proc. SPIN. pp. 230–239. Springer (2002)

[17] Esteva, A., Robicquet, A., Ramsundar, B., Kuleshov, V., DePristo, M., Chou, K., Cui, C., Corrado, G., Thrun, S., Dean, J.: A guide to deep learning in healthcare. Nat. Medicine **25**(1), 24 (2019)

[18] Fink, M., Liu, Y., Engstle, A., Schneider, S.A.: Deep Learning-Based Multi-scale Multi-object Detection and Classification for Autonomous Driving. In: Fahrerassistenzsysteme, pp. 233–242. Springer (2019)

[19] Golub, T.R., Slonim, D.K., Tamayo, P., Huard, C., Gaasenbeek, M., Mesirov, J.P., Coller, H., Loh, M.L., Downing, J.R., Caligiuri, M.A., et al.: Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. science **286**(5439), 531–537 (1999)

[20] Hinton, G., Deng, L., Yu, D., Dahl, G., Mohamed, A.r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Kingsbury, B., et al.: Deep neural networks for acoustic modeling in speech recognition. Signal Process. magazine **29**(6), 82–97 (2012)

[21] Holzmann, G.J.: Explicit-state model checking. In: Handbook of Model Checking, pp. 153–171. Springer (2018)

[22] Huang, X., Kwiatkowska, M., Wang, S., Wu, M.: Safety verification of deep neural networks. In: Proc. CAV. pp. 3–29. Springer (2017)

[23] Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., Bengio, Y.: Binarized neural networks. In: Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., Garnett, R. (eds.) Proc. NeurIPS. vol. 29, pp. 1–9. Curran Associates, Inc. (2016)

[24] Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient SMT solver for verifying deep neural networks. In: Proc. CAV. pp. 97–117. Springer (2017)

[25] Katz, G., Huang, D.A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljić, A., et al.: The marabou framework for verification and analysis of deep neural networks. In: Proc. CAV. pp. 443–452. Springer (2019)

[26] Kurshan, R., Levin, V., Yenigün, H.: Compressing transitions for model checking. In: Proc. CAV. pp. 569–582. Springer (2002)

[27] Lomuscio, A., Maganti, L.: An approach to reachability analysis for feed-forward ReLU neural networks. arXiv preprint arXiv:1706.07351 pp. 1–10 (2017)

[28] McMillan, K.L.: Symbolic model checking. In: Symbolic Model Checking, pp. 25–60. Springer (1993)

[29] Moosavi-Dezfooli, S.M., Fawzi, A., Fawzi, O., Frossard, P.: Universal adversarial perturbations. In: Proc. CVPR. pp. 1765–1773 (2017)

[30] Narodytska, N., Kasiviswanathan, S.P., Ryzhyk, L., Sagiv, M., Walsh, T.: Verifying Properties of Binarized Deep Neural Networks. In: Proc. AAAI. pp. 6615–6624 (2018)

[31] Naseer, M., Minhas, M.F., Khalid, F., Hanif, M.A., Hasan, O., Shafique, M.: FANNet: Formal Analysis of noise tolerance, training bias and input sensitivity in Neural Networks. In: Proc. DATE. pp. 666–669. IEEE (2020)

[32] Pei, K., Cao, Y., Yang, J., Jana, S.: Deepxplore: Automated whitebox testing of deep learning systems. In: Proc. SOSP. pp. 1–18. ACM (2017)

[33] Pulina, L., Tacchella, A.: Challenging SMT solvers to verify neural networks. AI Communications **25**(2), 117–135 (2012)

[34] Scheibler, K., Winterer, L., Wimmer, R., Becker, B.: Towards Verification of Artificial Neural Networks. In: Proc. MBMV. pp. 30–40 (2015)

[35] Shih, A., Darwiche, A., Choi, A.: Verifying binarized neural networks by local automaton learning. In: Proc. AAAI Spring Symposium on VNN. pp. 1–8 (2019)

[36] Siddiqui, M.A., Stokes, J.W., Seifert, C., Argyle, E., McCann, R., Neil, J., Carroll, J.: Detecting Cyber Attacks Using Anomaly Detection with Explanations and Expert Feedback. In: Proc. ICASSP. pp. 2872–2876. IEEE (2019)

[37] Singh, G., Gehr, T., Mirman, M., Püschel, M., Vechev, M.: Fast and Effective Robustness Certification. In: Proc. NeurIPS. pp. 10802–10813 (2018)

[38] Singh, G., Gehr, T., Püschel, M., Vechev, M.: An Abstract Domain for Certifying Neural Networks. PACMPL **3**(POPL), 41 (2019)

[39] Sze, V., Chen, Y.H., Yang, T.J., Emer, J.S.: Efficient processing of deep neural networks: A tutorial and survey. Proc. of the IEEE **105**(12), 2295–2329 (2017)

[40] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks. preprint arXiv:1312.6199 pp. 1–10 (2013)

[41] Tjeng, V., Xiao, K.Y., Tedrake, R.: Evaluating robustness of neural networks with mixed integer programming. In: Proc. ICLR. pp. 1–21 (2019)

[42] Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Efficient formal safety analysis of neural networks. In: Proc. NeurIPS. pp. 6367–6377 (2018)

[43] Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Formal security analysis of neural networks using symbolic intervals. In: Proc. USENIX Security Symposium. pp. 1599–1614 (2018)

[44] Wu, H., Ozdemir, A., Zeljić, A., Julian, K., Irfan, A., Gopinath, D., Fouladi, S., Katz, G., Pasareanu, C., Barrett, C.: Parallelization techniques for verifying neural networks. In: Proc. FMCAD. pp. 128–137 (2020)

[45] Xiang, W., Tran, H., Yang, X., Johnson, T.T.: Reachable set estimation for neural network control systems: A simulation-guided approach. TNNLS pp. 1–10 (2020)

[46] Xiang, W., Tran, H.D., Johnson, T.T.: Output reachable set estimation and verification for multilayer neural networks. TNNLS **29**(11), 5777–5783 (2018)

[47] Zakrzewski, R.R.: Verification of a trained neural network accuracy. In: Proc. IJCNN. vol. 3, pp. 1657–1662. IEEE (2001)

[48] Zangeneh, E., Rahmati, M., Mohsenzadeh, Y.: Low resolution face recognition using a two-branch deep convolutional neural network architecture. Expert Syst. with Applications **139**, 112854 (2020)

H Kripke Structure generation using FANNet and FANNet+

Fig. 9 presents the Kripke model generation for trained DNNs, using FANNet and FANNet+, respectively. The use of optimizations elaborated in Section 4 allow a significant reduction in the number of states in the model using FANNet+.

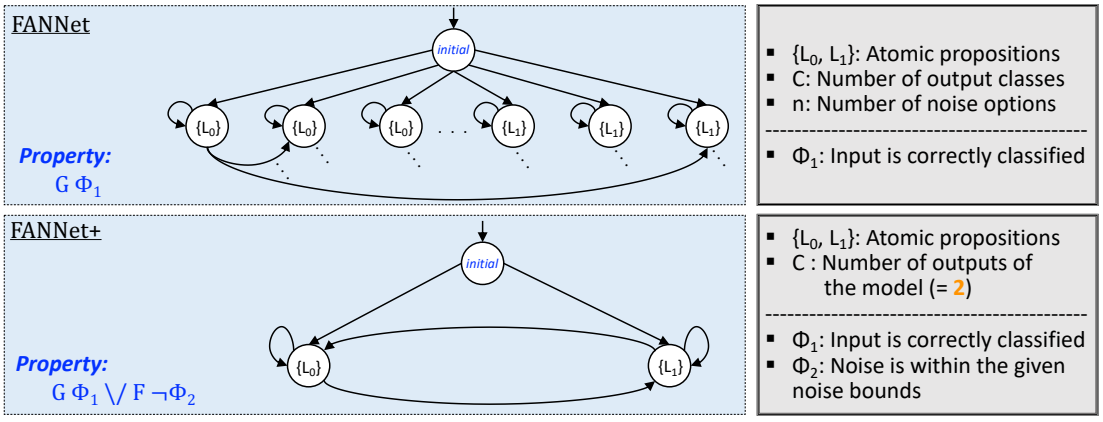

Figure 9: The Kripke structures generated by the model checker for DNN with two (= C) output classes/atomic propositions, using FANNet (top) and FANNet+ (bottom). `G` and `F` are the temporal operators indicating that the property holds *globally* and *eventually*, respectively.

I Random Input Segmentation (RIS): Details and Algorithm

The RIS algorithm, introduced in Section 4 proceeds as follows (also summarized in Algorithm 1):

1. Initially, for each input node, the upper and lower bounds of the (equally-spaced) input segments are calculated, as shown in Lines $3-4$ of Algorithm 1. The number of input segments for each node (i.e., the bins per input node (X)) is pre-defined.

2. The input nodes for the *variable* set are then picked, while the remaining nodes form the *fixed* set. Line 9 of Algorithm 1 illustrates the selection of a single input node i for the *variable* set, while remaining nodes $Btemp$ form the *fixed* set. However, the algorithm can be modified to work for any number of nodes being assigned to either of the sets.

3. Nested loops are used to pick the combination of segments for each input node in *fixed* set. For each combination of segments, a random discrete input value is picked for each input node, as shown in Lines $11-16$ of Algorithm 1.

4. The input domain is then updated with constant random values for input nodes in the *fixed* set, as shown in Lines $17-23$ of Algorithm 1.

5. The model checking is then performed with this updated input domain, shown by Line 25 of Algorithm 1.

6. If a counterexample corresponding to a property violation is obtained (as shown in Line 26 of Algorithm 1), i.e., the property is found to be SAT, model checking can be terminated. However, if no counterexample is found, the algorithm proceeds with the next discrete sample from the combination of segments of input nodes from the *fixed* set.

7. In turn, the process is repeated with a new splitting of input nodes into *variable* and *fixed* sets, as depicted in Lines $7-9$ of Algorithm 1.

As the number of input nodes fed to the DNN increase, the computation requirements (and subsequently the timing overhead) also increase. This "curse of dimensionality" is a known challenge with DNN analysis in the literature [44]. However, the parallelism enabled by the independence of the model checking of each splitting of the input nodes into the two sets alleviates the timing-overhead of the analysis.

Soundness – As shown in Fig. 4 and Lines $17-23$ of the Algorithm 1, the input domain I' contributing to the formal model analyzed by the model checker is the combination of the entire bounds of the input nodes from the *variable* set and random inputs from the selected segments of input nodes from the *fixed* set. Hence, this updated input domain is the proper subset of the original valid input domain I of the DNN, i.e., $I' \subset I$. This preserves soundness of the framework.

Completeness – Since the updated input domain I' is the proper subset of domain I, the model checking using the domain entails incompleteness. This is the direct result of the Lines $14-16$ of Algorithm 1, which bypass the exhaustive coverage of the segments of the input nodes from the *fixed* set. However, it must be noted that such incompleteness is essentially different

Algorithm 1 Random Input Segmentation

Input: Input Bounds (I), Network Parameters (w, b, L, N), Normalization Parameters (μ, ς), Bins per Input Node (X), Specification (Φ)
Output: Counterexample (CE)
Initialize: $CE \leftarrow [\,]$

//Creating bins to split ranges of each input node
for $i = 1$:Size$(I, 2)$ **do** ▷ For each input node
  for $j = 1{:}X(i)$ **do** ▷ For each input segment
    $B[1][j][i] \leftarrow (\frac{I(2,i)-I(1,i)}{X(i)} \times (j-1)) + I(1,i)$
    $B[2][j][i] \leftarrow (\frac{I(2,i)-I(1,i)}{X(i)} \times j) + I(1,i)$
  end for
end for
//Segmentation
for $i = 1$:Size$(I, 2)$ **do**
  $I' \leftarrow I$
  $Btemp \leftarrow B \setminus B\{:,:,i\}$
  $k \leftarrow$Size$(I, 2) - 1$ ▷ Total number of input nodes in *fixed* set
  for $j_1 = 1$:Size$(Btemp\{:,:,1\}, 2)$ **do**
    ...
    for $j_k = 1$:Size$(Btemp\{:,:,k\}, 2)$ **do**
      $temp[1] \leftarrow rand(Btemp[1, j_1, 1], Btemp[2, j_1, 1])$
      ...
      $temp[k] \leftarrow rand(Btemp[1, j_k, k], Btemp[2, j_k, k])$
      for $m = 1 : k$ **do** ▷ For updating I' with $temp$ for input nodes from *fixed* set
        if $i \leq m$ **then**
          $I'[m+1] \leftarrow temp[m]$
        else
          $I'[m] \leftarrow temp[m]$
        end if
      end for
      $temp \leftarrow [\,]$
      $CE \leftarrow$ **FANNet+**$(I', w, b, L, N, \mu, \varsigma, \Phi)$ ▷ Model checking
      if $CE \neq [\,]$ **then return** 1 ▷ Termination of code
      end if
    end for
    ...
  end for
end for

from the incompleteness often observed in DNN analysis literature [37, 46, 38, 45] arising from overapproximation of the input domain and/or activation functions, and hence leading to false positives in the analysis. The analysis based on our algorithm, in contrast, does not lead to any false positives.

J Overall Formalization and Property Checking Procedure used by FANNet+

As indicated in Fig. 5 earlier, FANNet+ makes use of the parameters and architectural details of the trained network to generate the formal DNN model. This formal model is consequently used for checking whether the desired network property holds for the network. The details of the procedure (also summarized in Algorithm 2) are as follows:

1. The input is first normalized, prior to being fed to the formal DNN model (as shown in

Algorithm 2 Network Formalization and Property Checking

Input: Input (I), Network Parameters (w, b, L, N), Normalization Parameters (μ, ς), Specification (Φ)
Output: Counterexample (CE)
Initialize: $CE \leftarrow [\,]$

1: **function** FANNet+$(I, w, b, L, N, \mu, \varsigma, \Phi)$
2: $I_{norm} \leftarrow \frac{I-\mu}{\varsigma}$ ▷ Input normalization
3: $temp\_I \leftarrow I_{norm}$
4: **for** $i = 1 : L$ **do**
5: **for** $j = 1 : N(i)$ **do**
6: $temp\_O[j] \leftarrow \sum(w[j][:][i]) \times temp\_I[j]) + b[j][i])$ ▷ Fully-connected layer
7: **if** $temp\_O[j] < 0$ **then**
8: $temp\_O[j] \leftarrow 0$ ▷ ReLU activation
9: **end if**
10: **end for**
11: $temp\_I \leftarrow temp\_O$ ▷ Input propagation to next DNN layer
12: **end for**
13: $Out_{norm} \leftarrow temp\_O$
14: $Out \leftarrow (Out_{norm} \times \varsigma) + \mu$ ▷ Output inverse-normalization
15: $CE \leftarrow$ `Property`(Φ, I, Out) ▷ Checking DNN specification
16: **return** CE
17: **end function**

Line 2 of Algorithm 2).

2. Input is propagated through layers of the DNN model (as shown in Lines $4-12$ of Algorithm 2).

3. Output may undergo inverse-normalization (as shown in Line 14 of Algorithm 2) prior to decision-making/classification by the DNN.

4. The desired formal property Φ of the network is then checked via SMC as described in detail in Section 4. In case the property does not hold for the network, the evidence of the property violation is returned by the model checker (as shown in Line 15 of Algorithm 2).

K Safety Verification results using Reluplex, Marabou and FANNet+

The safety properties relevant to the ACAS Xu networks studied in this work include:

1. Given a large distance between ownship and intruder, with the intruder travelling at much slower speed than ownship, the clear-of-conflict remains below a certain threshold.

2. Given a large distance between ownship and intruder, with the intruder travelling at much slower speed than ownship, the clear-of-conflict advisory does not have the highest value.

3. Given a constrained distance between ownship and intruder, with the intruder in line of ownship's translation and moving towards it, the DNN's clear-of-conflict advisory is not minimal.

4. Given a constrained distance between ownship and intruder, with the intruder in line of ownship's translation and moving away from it, but with a speed slower than that of the ownship, the DNN's clear-of-conflict advisory is not minimal.

The detailed results obtained after analyzing the safety properties $P1 - P4$ for ACAS Xu networks using Reluplex [24], Marabou [25] and FANNet+ are given in Tables 2 and 3 and Fig. 10. A consistent timeout of two-hours was used for all the experiments. As can be seen from the results in tables, for all experiments, whenever the renowned tools Reluplex and/or Marabou find satisfying solution to the violation of property, FANNet+ is also able to find the counterexample(s) within the pre-defined timeout.

Table 2: Results of verifying properties P1 and P2 for ACAS Xu Networks

Network	**Reluplex**		**Marabou**		**FANNet+**	
	P1	P2	P1	P2	P1	P2
1_1	UNSAT	UNSAT	UNSAT	UNSAT	TO	TO
1_2	UNSAT	SAT	UNSAT	SAT	TO	SAT
1_3	TO	SAT	UNSAT	TO	TO	TO
1_4	UNSAT	SAT	UNSAT	TO	TO	SAT
1_5	UNSAT	SAT	UNSAT	SAT	TO	TO
1_6	UNSAT	SAT	UNSAT	SAT	TO	TO
1_7	UNSAT	SAT	UNSAT	TO	TO	TO
1_8	UNSAT	TO	UNSAT	TO	TO	TO
1_9	UNSAT	UNSAT	UNSAT	TO	TO	TO
2_1	UNSAT	SAT	UNSAT	SAT	TO	SAT
2_2	TO	SAT	UNSAT	SAT	TO	SAT
2_3	TO	SAT	UNSAT	SAT	TO	SAT
2_4	UNSAT	SAT	UNSAT	SAT	TO	SAT
2_5	TO	SAT	TO	SAT	TO	SAT
2_6	TO	SAT	TO	SAT	TO	SAT
2_7	TO	SAT	TO	SAT	TO	SAT
2_8	TO	SAT	TO	SAT	TO	SAT
2_9	TO	SAT	TO	TO	TO	SAT
3_1	UNSAT	SAT	UNSAT	SAT	TO	SAT
3_2	UNSAT	SAT	UNSAT	TO	TO	TO
3_3	UNSAT	TO	UNSAT	TO	TO	TO
3_4	UNSAT	SAT	UNSAT	SAT	TO	SAT
3_5	UNSAT	SAT	UNSAT	SAT	TO	SAT
3_6	TO	SAT	TO	SAT	TO	SAT
3_7	UNSAT	SAT	TO	SAT	TO	SAT
3_8	TO	SAT	TO	SAT	TO	SAT
3_9	TO	SAT	TO	SAT	TO	SAT
4_1	TO	SAT	TO	TO	TO	SAT
4_2	TO	TO	UNSAT	TO	TO	TO
4_3	UNSAT	SAT	UNSAT	SAT	TO	SAT
4_4	UNSAT	SAT	UNSAT	SAT	TO	SAT
4_5	TO	SAT	TO	SAT	TO	SAT
4_6	TO	SAT	UNSAT	SAT	TO	SAT

Network	Reluplex		Marabou		FANNet+	
	P1	P2	P1	P2	P1	P2
4_7	TO	SAT	UNSAT	SAT	TO	SAT
4_8	TO	SAT	TO	SAT	TO	SAT
4_9	TO	SAT	TO	SAT	TO	SAT
5_1	UNSAT	SAT	UNSAT	SAT	TO	SAT
5_2	TO	SAT	UNSAT	SAT	TO	SAT
5_3	UNSAT	SAT	UNSAT	SAT	TO	TO
5_4	UNSAT	SAT	UNSAT	SAT	TO	SAT
5_5	TO	SAT	UNSAT	SAT	TO	SAT
5_6	TO	SAT	TO	SAT	TO	SAT
5_7	TO	SAT	TO	SAT	TO	SAT
5_8	TO	SAT	TO	SAT	TO	SAT
5_9	TO	SAT	TO	SAT	TO	SAT

SAT: satisfiable solution found
UNSAT: property unsatisfiable
TO: timeout (time exceeded 2 hours)
NF: no counterexample found

Table 3: Results of verifying properties P3 and P4 for ACAS Xu Networks

Network	Reluplex		Marabou		FANNet+	
	P3	P4	P3	P4	P3	P4
1_1	UNSAT	UNSAT	UNSAT	UNSAT	TO	NF
1_2	UNSAT	UNSAT	UNSAT	UNSAT	TO	TO
1_3	UNSAT	UNSAT	UNSAT	UNSAT	TO	TO
1_4	UNSAT	UNSAT	UNSAT	UNSAT	TO	NF
1_5	UNSAT	UNSAT	UNSAT	UNSAT	TO	NF
1_6	UNSAT	UNSAT	UNSAT	UNSAT	TO	TO
1_7	SAT	SAT	SAT	SAT	SAT	SAT
1_8	SAT	SAT	SAT	SAT	SAT	SAT
1_9	SAT	SAT	SAT	SAT	SAT	SAT
2_1	UNSAT	UNSAT	UNSAT	UNSAT	TO	TO
2_2	UNSAT	UNSAT	UNSAT	UNSAT	TO	SAT
2_3	UNSAT	UNSAT	UNSAT	UNSAT	TO	NF
2_4	UNSAT	UNSAT	UNSAT	UNSAT	TO	NF
2_5	UNSAT	UNSAT	UNSAT	UNSAT	TO	TO
2_6	UNSAT	UNSAT	UNSAT	UNSAT	TO	TO
2_7	UNSAT	UNSAT	UNSAT	UNSAT	TO	TO
2_8	UNSAT	UNSAT	UNSAT	UNSAT	NF	TO
2_9	UNSAT	UNSAT	UNSAT	UNSAT	TO	NF
3_1	UNSAT	UNSAT	UNSAT	UNSAT	TO	NF
3_2	UNSAT	UNSAT	UNSAT	UNSAT	TO	NF
3_3	UNSAT	UNSAT	UNSAT	UNSAT	TO	NF
3_4	UNSAT	UNSAT	UNSAT	UNSAT	TO	NF
3_5	UNSAT	UNSAT	UNSAT	UNSAT	TO	TO
3_6	UNSAT	UNSAT	UNSAT	UNSAT	TO	NF
3_7	UNSAT	UNSAT	UNSAT	UNSAT	TO	NF

Network	Reluplex		Marabou		FANNet+	
	P3	P4	P3	P4	P3	P4
3_8	UNSAT	UNSAT	UNSAT	UNSAT	TO	NF
3_9	UNSAT	UNSAT	UNSAT	UNSAT	TO	NF
4_1	UNSAT	UNSAT	UNSAT	UNSAT	TO	NF
4_2	UNSAT	UNSAT	UNSAT	UNSAT	TO	NF
4_3	UNSAT	UNSAT	UNSAT	UNSAT	TO	NF
4_4	UNSAT	UNSAT	UNSAT	UNSAT	TO	NF
4_5	UNSAT	UNSAT	UNSAT	UNSAT	TO	NF
4_6	UNSAT	UNSAT	UNSAT	UNSAT	TO	NF
4_7	UNSAT	UNSAT	UNSAT	UNSAT	TO	NF
4_8	UNSAT	UNSAT	UNSAT	UNSAT	TO	TO
4_9	UNSAT	UNSAT	UNSAT	UNSAT	TO	NF
5_1	UNSAT	UNSAT	UNSAT	UNSAT	TO	NF
5_2	UNSAT	UNSAT	UNSAT	UNSAT	NF	NF
5_3	UNSAT	UNSAT	UNSAT	UNSAT	TO	NF
5_4	UNSAT	UNSAT	UNSAT	UNSAT	TO	NF
5_5	UNSAT	UNSAT	UNSAT	UNSAT	TO	NF
5_6	UNSAT	UNSAT	UNSAT	UNSAT	TO	NF
5_7	UNSAT	UNSAT	UNSAT	UNSAT	TO	NF
5_8	UNSAT	UNSAT	UNSAT	UNSAT	TO	NF
5_9	UNSAT	UNSAT	UNSAT	UNSAT	TO	NF

SAT: satisfiable solution found UNSAT: property unsatisfiable
TO: timeout (time exceeded 2 hours) NF: no counterexample found

FANNet+ is able to find violation to property $P4$ for the network 2_2, i.e., the experiment for which both Reluplex and Marabou return UNSAT. The satisfying input leading to violation of the property, as found by FANNet+ is as follows: $[1600, 3, 0, 1100, 720]$. The validity of the counterexample is confirmed using `Maraboupy`.

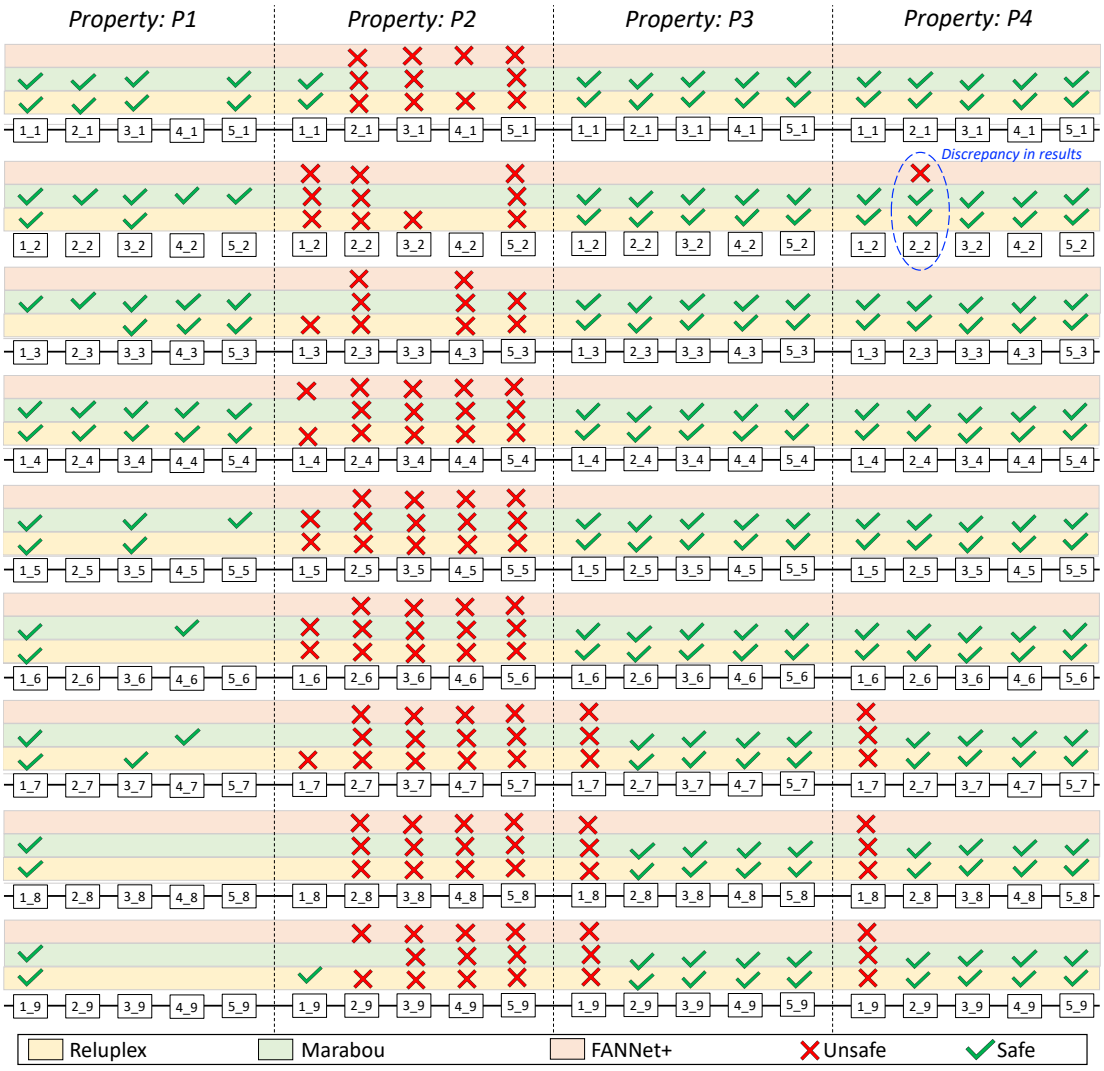

Figure 10: Safety verification using Reluplex, Marabou and FANNet+: in case of discrepancy in result, FANNet+ is found to provide correct result as opposed to the the other frameworks.