TU WIEN Informatics

Establishing Formal Behavioral Guarantees for Trained Neural Networks

Towards Reliable Machine Learning Systems

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktorin der Technischen Wissenschaften

by

Projektass.in Mahum Naseer, MSc
Registration Number 11838956

to the Faculty of Informatics

at the TU Wien

Advisor: Prof. Dr. Muhammad Shafique, Ph.D.
Second advisor: Prof. Dr. Osman Hasan, Ph.D.

The dissertation has been reviewed by:

Sachin Sapatnekar

Hiren Patel

Vienna, May 15, 2024

Mahum Naseer

Erklärung zur Verfassung der Arbeit

Projektass.in Mahum Naseer, MSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 15. Mai 2024

Mahum Naseer

Acknowledgements

This work was partially supported by Doctoral College Resilient Embedded Systems which is run jointly by TU Wien's Faculty of Informatics and FH-Technikum Wien, and partially by Moore4Medical project funded by the ECSEL Joint Undertaking under grant number H2020-ECSEL-2019-IA-876190.

Abstract

Machine Learning (ML)-based systems, particularly those deploying deep neural networks (DNNs), are widely adopted into real-world applications due to their ability to be trained without being explicitly programmed and high output accuracy. However, despite their high classification accuracy and optimal decision-making in testing scenarios, they are often found to be vulnerable under unseen (but realistic) inputs. This points to the lack of generalization of these data-driven models under unseen input scenarios, hence highlighting the need for behavioral guarantees to ensure their reliable classification and decision-making in the real world.

Research efforts constantly provide empirical evidence for the lack of reliable DNN behavior (under seed inputs) for various ML applications. Orthogonally, formal efforts attempt to provide concise formal guarantees for behavioral properties/specifications like robustness and safety to hold for the DNN models. However, due to the scalability challenges associated with formal methods, not only are these efforts often restricted to providing qualitative (binary) guarantees but they also focus only on limited DNN behaviors and verification techniques.

To address the aforementioned limitations, this research provides model checking and scalable sampling-based formal frameworks for DNN analysis, focusing on a diverse range of DNN behavioral specifications. These include DNN noise tolerance, input node sensitivity (to noise), node robustness bias, robustness under constrained noise, robustness bias against tail classes and safety under bounded inputs. Realistic noise modeling is proposed for practical DNN analysis, while restraining from the use of unrealistic assumptions during analysis. These lead to formal guarantees that may potentially be leveraged to identify reliable ML systems.

The research additionally leverages our DNN analysis to improve training for robust DNNs. The resulting frameworks designed and developed during the research are all accompanied by case studies based on DNNs trained on real-world datasets, hence supporting the efficacy of the research efforts and provide behavioral guarantees essential to ensure more reliable ML systems in practice.

Contents

Thesis Publications

[1] **Mahum Naseer**, Osman Hasan, and Muhammad Shafique. QuanDA: GPU Accelerated Quantitative Deep Neural Network Analysis. In *Transactions on Design Automation of Electronic Systems*, volume 28, pages 1–21. ACM, 2023.

[2] **Mahum Naseer**, Osman Hasan, and Muhammad Shafique. Scaling Model Checking for Neural Network Analysis via State-Space Reduction and Input Segmentation. In *Formal Methods for ML-Enabled Autonomous Systems (FoMLAS)*, volume 16, pages 6–28. Kalpa Publications in Computing, 2023.

[3] **Mahum Naseer** and Muhammad Shafique. Poster: Link between Bias, Node Sensitivity and Long-Tail Distribution in trained DNNs. In *Conference on Software Testing, Verification and Validation (ICST)*, pages 474–477. IEEE, 2023.

[4] **Mahum Naseer**, Bharath Srinivas Prabakaran, Osman Hasan, and Muhammad Shafique. UnbiasedNets: a Dataset Diversification Framework for Robustness Bias Alleviation in Neural Networks. *Machine Learning*, pages 1–28, 2023.

[5] Iram Tariq Bhatti, **Mahum Naseer**, Muhammad Shafique, and Osman Hasan. A Formal Approach to Identifying the Impact of Noise on Neural Networks. *Communications of the ACM*, 65(11):70–73, 2022. *[Iram Tariq Bhatti and Mahum Naseer contributed equally to the work]*.

[6] Muhammad Shafique, **Mahum Naseer**, Theocharis Theocharides, Christos Kyrkou, Onur Mutlu, Lois Orosa, and Jungwook Choi. Robust Machine Learning Systems: Challenges, current Trends, Perspectives, and the Road ahead. *IEEE Design & Test*, 37(2):30–57, 2020. *[Mahum Naseer contributed to the Formal Verification and Empirical Analysis aspects of the work.]*.

[7] **Mahum Naseer**, Mishal Fatima Minhas, Faiq Khalid, Muhammad Abdullah Hanif, Osman Hasan, and Muhammad Shafique. FANNet: Formal Analysis of Noise Tolerance, Training Bias and Input Sensitivity in Neural Networks. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 666–669. IEEE, 2020.

CHAPTER 1

Introduction

Machine Learning (ML) systems, often based on Deep Neural Networks (DNNs), are widely used in the present-day embedded devices. The ability of DNNs to perform decision-making tasks without being explicitly programmed, makes them an appealing choice in real-world applications like spam filtering [ZSAT19, SSS$^+$19], speech recognition [HDY$^+$12], face recognition [HYY$^+$15, ZRM20], and smart-grids [GSK$^+$11]. Additionally, the high accuracy of these systems and their relearning capabilities make them an appropriate choice even for safety-critical applications like autonomous driving [BDTD$^+$16, FLES19] and healthcare [ERR$^+$19, BA21].

1.1 Research Gaps and Problem Statement

Despite the successful outlook of DNNs in numerous applications, there remain gaps in the ML research domain regarding the reliability of such networks, in turn challenging the reliability of DNN systems. This section elaborates on two of these research gaps, i.e., the vulnerabilities of the trained DNNs and the limitations of the DNN design cycle[1] challenging the generalization capabilities of the resulting networks, ultimately reducing the reliability of the systems deploying the networks in the real world. They are, in turn, used to establish the relevance of the research problem dealt with in this thesis.

1.1.1 DNNs are Vulnerable

DNNs comprise the data-driven part of the ML systems. This means, the networks learn to make decisions based on the data samples available to DNN models. As a result, the accuracy of the trained DNN is dependent on the data available to the DNN for learning.

[1]Since this thesis deals with trained DNN models, we limit our preliminary discussion of DNN training to supervised learning. Interested readers are encouraged to read [Bro19] for a summary of more learning algorithms.

Yet, many tasks making use of DNNs involve the recognition/identification of the rare events [Mea01, RTVG18, Wea14], for which only insufficient data is available for the DNNs to learn. Such data is said to exhibit *long-tail distribution*, and provides the first source of vulnerability to the DNNs.

As shown in Fig. 1.1(left), the number of samples in the *head* (i.e., frequently occurring) and *tail* (rarely occurring) classes vary significantly in the datasets with the long-tail distribution. As a result, the DNNs trained on such datasets are often able to identify the head classes significantly better than the tail classes [Lea18, Fea22].

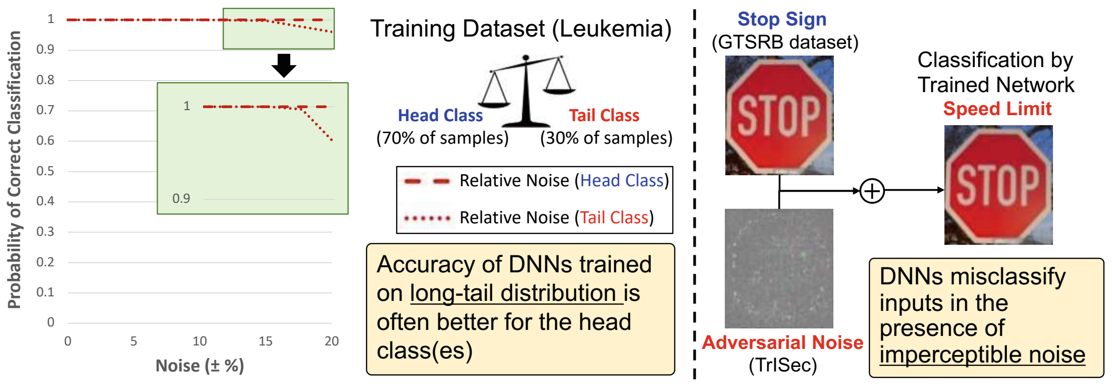

Probability of Correct Classification
Noise (± %)
Training Dataset (Leukemia)
Head Class
(70% of samples)
Tail Class
(30% of samples)
Relative Noise (Head Class)
Relative Noise (Tail Class)
Accuracy of DNNs trained on long-tail distribution is often better for the head class(es)
Stop Sign
(GTSRB dataset)
Adversarial Noise
(TrISec)
Classification by Trained Network
Speed Limit
DNNs misclassify inputs in the presence of imperceptible noise

Figure 1.1: Low classification accuracy of tail class for DNN trained on dataset with long-tail distribution (left), and misclassification of seed input by trained DNN in the presence of small, imperceptible noise (right).

Orthogonally, DNNs are also observed to provide incorrect outputs when the network input is subjected to small, even imperceptible, noise (as depicted in Fig. 1.1 (right)). Such noise, often referred to as *adversarial noise*, has been observed to trigger network misclassification in various DNNs, including those dealing with visual [SZS+13] and audio [GP18] inputs. Noise forms a ubiquitous part of the real-world systems [KGB17a], and appears even in the digital domain (for instance, as quantization noise). This makes dealing with the DNNs' vulnerability to noise an important research challenge that needs to be dealt with for reliable ML systems.

1.1.2 DNN Design Cycle

A generic design cycle of a DNN (based on supervised learning) involves the use of *training*, *validation* and *testing* datasets, as shown in Fig. 1.2. The training algorithm makes use of the training dataset to optimize network parameters. The validation dataset is employed for hyperparameter tuning and to ensure optimal training of the network. The trained DNN is finally checked against the testing dataset to ensure high accuracy of the DNN on unseen inputs.

However, the entire DNN design cycle makes use of a finite subset of the actual input domain for training, validation and testing. Hence, there remains a possibility that the available datasets do not (adequately) represent parts of the input domain, ultimately leading to a lack of generalization of the trained DNN.

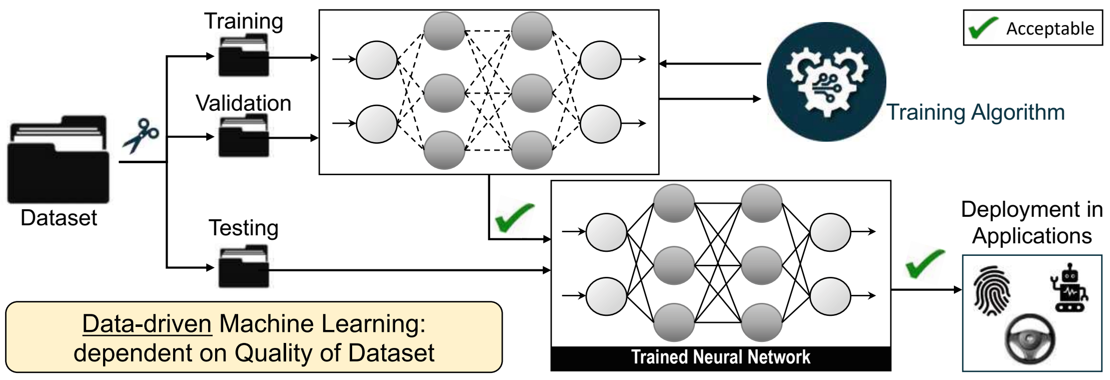

Training
Validation
Testing
Dataset
Acceptable
Training Algorithm
Deployment in Applications
Trained Neural Network
Data-driven Machine Learning: dependent on Quality of Dataset

Figure 1.2: Generic DNN design cycle: the generalization capability of the trained network is dependent on the quality of the available dataset.

Problem Statement: Owing to the data-dependent nature of the training and testing of DNNs, the DNNs' vulnerability to noise and their varying accuracy for different output classes are consequential. However, with the increasing deployment of these DNNs in real-world ML systems, *concrete guarantees, under real-world setting, are essential for the trained DNNs prior to their deployment.*

1.2 Summary of the State of the Art and the Open Challenges

Numerous research efforts are targeted to provide behavioral guarantees for the trained DNNs. These can be broadly categorized into: *empirical* efforts and *formal* efforts. The *empirical* efforts often make use of optimization with varying objectives to find behavioral discrepancies of the DNNs [SZS+13, PCYJ17, KHR+19]. Although, in practice, these efforts have been successful in identifying erroneous DNN behavior, these efforts are insufficient to provide any guarantees in case no evidence of erroneous DNN behavior is found.

In contrast, the *formal* efforts are capable of providing the behavioral guarantees essential for trained DNNs prior to being deployed. However, given the non-linearity and complexity of DNN models, and the curse-of-dimensionality of the DNN inputs [WOZ+20], these efforts are instead challenged by their limited scalability. For better scalability, the completeness of the efforts is sometimes traded off to allow behavioral verification of larger DNNs [SGPV19, TPL+21, MMS+22]. Yet, most of these formal efforts provide only *qualitative* results, i.e., whether the DNNs obey certain behavioral specifications or not. Since the DNNs rarely obey the desired behavior, the emerging efforts instead provide behavioral guarantees for trained DNNs instead focus on *quantitative* results, i.e., the precise degree of the DNNs to follow the desired specifications, hence pointing ultimately the level of reliability of the system deploying them.

Among the behavioral properties of trained DNNs commonly focused on in these works are robustness under constrained noise and safety under bounded input. However, this leaves numerous open problems and challenges in this research domain:

1. Limited exploration of the *formalization* techniques for formal DNN analysis.
2. Dependence on norm-based *noise modeling* for analysis of DNNs.
3. Focus on a narrow range of behavioral *properties/specifications* for trained DNNs.
4. Lack of precise *confidence* intervals of the results during incomplete analysis.
5. High *timing* overhead of the DNN analysis.

1.3 Scientific Goals of the Research

The aforementioned research challenges give rise to numerous scientific goals that are addressed in this thesis. The following provides a detailed account of these scientific goals of this research:

1. **Formalizing DNNs and their properties:**
   The state of the art in formal DNN analysis mostly relies on Satisfiability (SAT) solving and Linear Programming (LP) for providing (qualitative) behavioral guarantees for trained networks. The quantitative analysis on the other hand makes use of assumptions while formalizing DNNs. Being hinged on propositional logic, pure SAT-based verification approaches have a limited scope, i.e., they are only applicable for the verification of Binary Neural Networks (BNNs) [NKR$^+$18, SDC19a]. Satisfiability Modulo Theories (SMT) and LP-based approaches, on the other hand, make use of mathematical theories like the theory of linear arithmetic to extend the verification objective to a general class of neural networks, with real-numbered network parameters and piecewise linear activation functions [KHI$^+$19, BLT$^+$20]. The assumptions (for instance, on the output probability distribution of the network nodes) used in the quantitative analysis are often unrealistic [CFGP20, PCFG20]. In general, there are only limited works [SDC19b, ZZC$^+$21] exploring alternative approaches for the formalization techniques for DNN models and their properties.

   Goal: *While state-of-the-art DNN analyses have seen significant improvement over the years, they are generally restricted to only a few classical analysis approaches like SAT, SMT and LP. The objective of this research is to explore alternate formal verification approaches, for instance, model checking, to broaden the scope* *of the existing DNN analyses and provide alternative, and potentially useful, means for formal behavioral guarantees for the trained DNNs.*

2. **DNN analysis under practical settings:**
   Noise is often an unavoidable component of the real-world environment. Hence, for analysis in practical settings, such noise must be accounted for. The current DNN analysis efforts focus on L^p-norm based noise impacting *normalized* DNN inputs [DJST18, BKK+20]. However, in practice, noise affects the raw/*unnormalized* inputs, which are often not focused on in the existing efforts.

   Goal: *While current DNN analyses often consider environmental factors like noise, they are not considered in the most practical setting. The objective of this research is to study the vulnerabilities of trained DNNs in a more realistic setting, via leveraging the relative noise model.*

3. **Identifying DNN vulnerabilities and expanding the bounds of analysis to cater to a diverse range of DNN vulnerabilities:**
   As indicated earlier, the current efforts to provide formal behavioral guarantees for trained DNNs are often restricted to analyzing only the robustness under constrained noise and safety under bounded inputs [HKWW17, KHI+19]. However, this does not encapsulate all possible vulnerabilities of the trained DNNs, and hence requires enlarging the focus of DNN analysis.

   Goal: *While robustness and safety are essential behavioral properties for trained DNNs, these are not the only properties to ascertain reliable networks. The objective of this research is hence to target a diverse range of DNN properties, extending beyond only robustness and safety, that could improve the reliability of DNNs deployed in various embedded applications.*

4. **Providing the confidence intervals for the subsequent analysis results in case completeness is traded-off in favor of scalability:**
   The trade-off between scalability and completeness of analysis is an ongoing challenge. However, in cases where the completeness of analysis is compromised in favor of better scalability, such analysis is scarcely accompanied by the confidence intervals for the incomplete results [CFGP20, YLL+21b], and may often lead to false positives [GMDC+18, SGPV19].

   Goal: *While incomplete formal analyses of trained DNNs come with their own challenges, this research aims to leverage incompleteness while minimizing its limitations. This is achieved in two ways: by restricting incompleteness only to the input domain (and not formal DNN modeling via overapproximation and abstraction) to avoid false positives, and by providing confidence intervals with the analysis results whenever exact results are not computable.*

5. **Optimizing DNN analysis for better scalability and reduced timing overhead:**
Due to the complexity and non-linearity of trained DNNs, and the curse of DNN input dimensionality, scaling DNN analysis and minimizing timing overhead (in terms of analysis runtime) is a big challenge. Numerous strategies are explored in the literature to overcome these challenges, including input segmentation [BTT+18, GKPB18], overapproximation [GMDC+18, WPW+18a] and parallel computations [WOZ+20]. Nevertheless, finding better solutions to improve timing efficiency of DNN analysis is an ongoing scientific goal.

Goal: *While trading off completeness for scalability already improves the timing efficiency of the DNN analysis, this research also explores model reduction and input segmentation techniques to further reduce the timing of the analysis. Additionally, we leverage GPU implementation to enable efficient parallelization, further improving our analysis.*

1.4 Thesis Contributions

With the increasing reliance of ML systems on DNNs, providing behavioral guarantees for such trained DNNs, prior to their deployment in real-world applications, is an important research problem. However, the problem is far from trivial. As discussed in the previous sections, numerous open challenges and scientific goals still remain to be addressed. To address them, this thesis contributes both to the design, development and improvement of DNN analysis frameworks, ultimately ensuring reliable ML systems deploying such DNNs. Additionally, once the vulnerabilities in the DNNs have been identified, we also explore prospective directions to overcome the vulnerabilities, hence robustifying the networks.

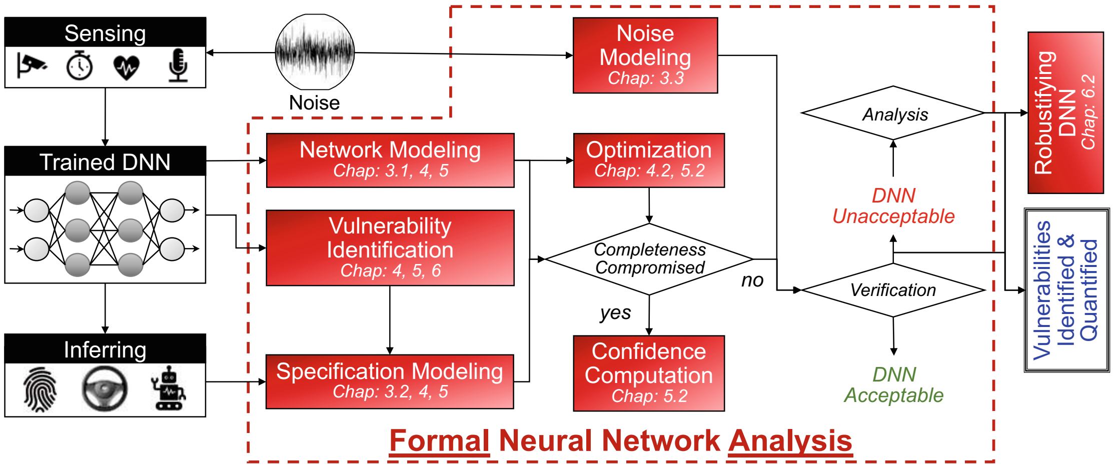


Figure 1.3: Thesis overview and how it fits with the standard DNN design cycle.

The novel contributions of this research are summarized in Fig. 1.3 and elaborated as follows:

1. **Alternate formalization approaches:** (*Chapters: 3.1,4,5*)
   This work provides trained network formalization for model checking and sampling-based DNN analysis. These form the basis for our qualitative and quantitative analysis frameworks.

2. **Formalizing DNN specifications:** (*Chapters:3.2,4,5*)
   A diverse range of DNN behavioral specifications are studied extensively in this thesis, qualitatively and quantitatively. These include network robustness under (practical) constrained noise, noise tolerance, robustness bias to specific output classes, input node sensitivity, and output safety under bounded inputs.

3. **Realistic noise modeling:** (*Chapter: 3.3*)
   Relative noise modeling is introduced in this research, which perturbs the raw, unnormalized inputs proportional to the magnitude of the input. This encapsulates a more realistic notion of noise under practical settings.

4. **Identifying unexplored DNN vulnerabilities:** (*Chapters: 4,5,6*)
   Existing literature indicates persistent efforts toward analyzing the lack of robustness of trained DNNs under noise or questionable safety guarantees. This work additionally identifies output- and node-specific vulnerabilities of DNNs namely, robustness bias and node robustness bias.

5. **Unified (qualitative) framework for analyzing co-existing vulnerabilities:** (*Chapter: 4*)
   This work proposes *FANNet+*, a unified model checking-based framework, which is able to analyze multiple co-existing vulnerabilities. These range from overall network-specific vulnerabilities to individual output- and input node-specific vulnerabilities.

6. **Leverage GPU for scalability and reduced analysis time (of the quantitative analysis):** (*Chapter: 5*)
   The computations involved in determining the output of the nodes of a single DNN layer are generally independent of one another. This leaves potential for massive parallelization during DNN analysis, which is leveraged by our framework *QuanDA* via GPU-based acceleration.

7. **Providing confidence intervals for incomplete analysis:** (*Chapter: 5.2*)
   Compromising completeness is often necessary to improve scalability. However, the incompleteness in DNN analysis is rarely accompanied by the confidence of the obtained results in existing DNN analysis literature. On the contrary, our framework *QuanDA* provides precise confidence intervals for the deviation of the exact results from the estimated ones.

8. **Potential direction for minimizing DNN vulnerability**: (*Chapter: 6.2*) DNN analysis identifies the vulnerabilities in the trained networks. To make the most of the obtained results, the next step is to use the knowledge of DNN vulnerabilities to overcome them. Towards this end, this thesis also explores prospective direction to solve the robustness bias vulnerability of DNNs. This is accomplished by developing the data-centric framework *UnbiasedNets* to minimize the bias at its core, i.e., from within the training dataset itself.

1.5 Thesis Outline

The rest of this thesis is organized as follows:

Chapter 2 provides the details of the existing works for DNN analysis. It encompasses the efforts both in empirical and formal DNN analysis domains. It further discusses the most prominent analysis methodologies used and DNN vulnerabilities highlighted in existing works. Particular attention is given to the formal DNN analysis, dealing separately with the qualitative and quantitative efforts at length.

Chapter 3 presents the formalism and notation used throughout the rest of the thesis. In particular, the chapter provides the functional (i.e., implementation) and behavioral (i.e., specification) formalism of the trained DNNs. Additionally, noise formalism, including the relative noise modeling, is elaborated.

Chapter 4 explains the step-by-step construction of our model checking-based qualitative DNN analysis framework, *FANNet+*. This includes DNN Kripke structure generation, state-space reduction, and input segmentation. The applicability of the framework is shown using three case studies for networks trained on real-world datasets, for analyzing multiple DNN vulnerabilities.

Chapter 5 deals with quantitative DNN analysis, discussing both the probabilistic and statistical approaches. The probabilistic DNN analysis extends on *FANNet+*, while the statistical analysis is implemented using the novel GPU-accelerated sampling-based framework, *QuanDA*. The applicability of the frameworks is again shown using case studies of DNNs trained on real-world datasets.

Chapter 6 discusses the future potential of the DNN analysis research. It includes a detailed investigation of the node and output-specific network behavior. A data-centric framework, *UnbiasedNets*, is also proposed as a potential means to minimize the robustness bias in trained DNNs. These are accompanied by case studies and a comparative analysis with analogous frameworks.

Chapter 7 concludes the thesis, recapitulating the main findings and results from this research, and presenting a future outlook for the research domain.

CHAPTER 2

Background and Related Work

In general, DNNs comprise of groups of nodes arranged in layers. The networks receive (normalized) inputs via the *input layer* and outputs the result via the *output layer*. Placed between them are the *hidden layer(s)*. This thesis focuses only on the feed-forward DNNs, i.e., the DNNs with unidirectional input propagation. An example of feed-forward DNNs is represented in Fig. 2.1(left). Nodes from each layer are connected to those of the subsequent layer via a sequential arrangement of linear and non-linear transformations. The linear transformations include affine and convolution operations. The popular non-linear transformations include ReLU and maxpool (see Fig. 2.1(right)). The details for network architecture and formalism will be discussed later in Chap. 3.1. Once trained, this seemingly straightforward (but not trivial) interconnection of nodes is capable of performing classification and decision-making with high accuracy [ZSAT19, ZRM20, ERR$^+$19].

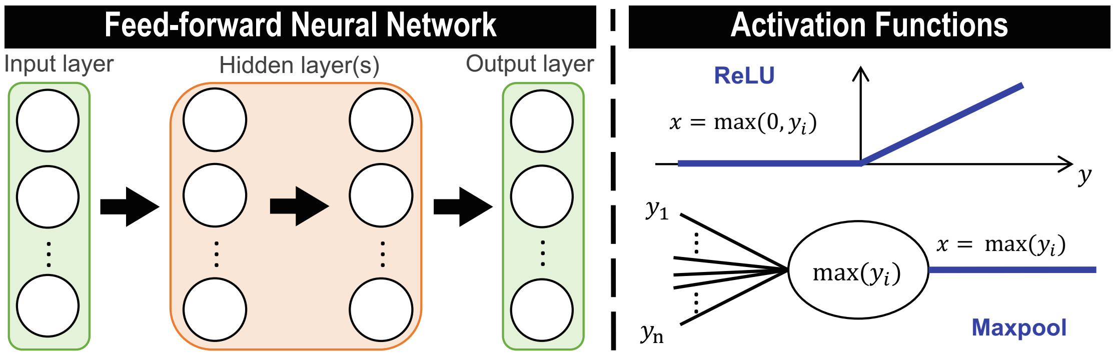


Figure 2.1: General DNN architecture and popular activation functions.

However, at least a decade of research indicates the vulnerability of these networks under different input scenarios. For instance, the addition of small, imperceptible noise to inputs has been observed to stimulate incorrect output for DNNs in various applications, including image and speech recognition [SZS+13, GP18]. The DNNs are additionally found to be vulnerable to various kinds of biases, including data bias, representation bias, and robustness bias [HDB20, MMS+21, NDS+21], whereby the trained DNNs have a varying degree of classification accuracy of different output classes, either in presence or absence of noise.

Other known DNN vulnerabilities include data poisoning (for instance, via backdoor [GLDGG19]) and bit flips causing DNN parameters to change (for instance, under cosmic radiation [StT+20]). Naturally, there remains the possibility of new vulnerability discoveries for DNNs. This chapter provides an in-depth overview of the various DNN vulnerabilities (including those indicated above) and the analysis approaches used to identify them. These are categorized under the empirical (elaborated in Sec. 2.1) and formal approaches (elaborated in Sec. 2.2). Sec. 2.3 summarises the related work discussed in the chapter.

2.1 Empirical Exploration of Network Vulnerabilities

Among the popular DNN vulnerabilities studied empirically is the DNN misclassification under noise. The small, often imperceptible, noise leading to such misclassifications, when added to network input, is referred to as the adversarial noise. The analysis approaches used for the identification of such noise generally deal with the identification of the closest decision boundary, direction of the loss function, and the most important input features [GSS15, KGB17b, CW17, WX18, B18, KHR+19, KAH+19]. Optimization algorithms are then employed, with optimization objectives ranging from the imperceptibility of noise to the targeted or untargeted misclassifications.

Another class of DNN vulnerability studied empirically is the network bias. This includes the *data bias*, i.e., the bias of the trained DNN emerging due to non-optimal training procedures, for instance, using small datasets. This results trained network with bad generalization capabilities [LLV18, LV19]. The *representation bias*, on the other hand, results due to an unequal representation of all output classes in the training dataset. This is a fairly likely phenomenon for training datasets dealing with rare events/ having long-tail distribution [RTVG18, Mea01, Wea14]. Hence, the performance metrics like class-wise accuracy, class-wise variance, and maximum class discrepancy of the trained networks vary significantly for the individual output classes [TKJ+21].

The deployment of DNNs in edge-devices and applications has encouraged energy-efficient DNN architectures. This motivated the (empirical) analysis of network node significance to prune away the nodes of the DNN that have the least impact on the accuracy of the trained network [ZMC94, HKHF00, CRF+20]. Node sensitivity, particularly that of the input nodes, is also studied to ensure the privacy of sensitive input attributes [ZL19, MFVRT20]. These analyses either take the forward-pass approach, whereby the impact of noise to input nodes is studied at the output nodes [HKHF00], or the backward-approach where partial gradients are leveraged to study insensitivity of individual network nodes [ZMC94].

2.2 Formal Guarantees to Network Resilience

Due to the high dependence of DNNs on training datasets and their vulnerability to input uncertainties, the empirical approaches discussed above have been fairly successful in providing evidence of undesirable DNN behavior. Nevertheless, in theory, such approaches are incapable of providing conclusive guarantees in cases when no undesirable DNN behavior is identified. This is because the undesirable behavior may compose a corner-case, which may be neglected by the empirical analysis. This may loosely be referred to as the *incompleteness* of the analysis.

Formally, a proof system is said to be *complete* if there exists a formal proof for every true property expressed in the proof system [HR04]. The concept of completeness is often studied along with its ties to the formal methods (or formal verification), which aim to provide concrete system guarantees for the entire input domain and/or of all possible system configurations. To obtain similar guarantees for trained DNNs, numerous efforts have studied the formal verification DNNs, to provide concrete behavioral guarantees. However, due to the non-convexity and non-linearity of DNNs, complete formal verification is often infeasible – the theory of transcendental functions sometimes used as the activation functions in DNNs is known to be undecidable [Ric69], and even the verification of DNNs using the piecewise linear activations is an NP-complete problem [KBD$^+$17]. The verification of quantized neural networks, i.e., the networks with fixed (and often small) bit-width parameter sizes, is also shown to be a PSPACE-hard problem [HLŽ21].

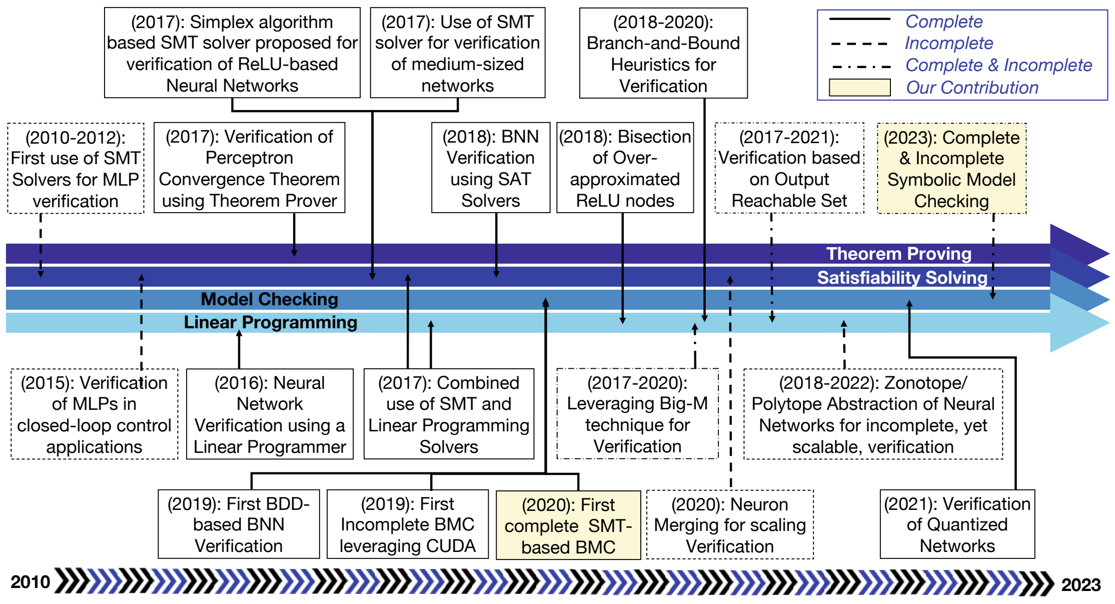
Nevertheless, constant efforts are made to provide both qualitative and quantitative formal behavioral guarantees for trained DNNs, as elaborated in the rest of this section.

2.2.1 Qualitative Efforts

The qualitative DNN verification efforts rely on classical formal method techniques to provide qualitative results for whether (or not) the trained network delineates desirable output behavior. Hence, the verification provides binary results, i.e., either the desirable behavior holds for the DNN or not. Towards this, numerous works have proposed qualitative efforts, as summarized in Fig. 2.2, over the past years. The major portion of these efforts leverages satisfiability solving and linear programming, while theorem proving and model checking have been sparingly studied for DNN verification as well. The following discusses the qualitative efforts in each of these four formal method domains

©2020 IEEE. Parts reproduced, with permission, from M. Shafique, M. Naseer, T. Theocharides, C. Kyrkou, O. Mutl, L. Orosa, J. Choi: Robust machine learning systems: Challenges, current trends, perspectives, and the road ahead, *IEEE Design & Test (D&T 2020)*, pp. 666-669. [DOI:`doi.org/0.1109/MDAT.2020.2971217`].

(provided in reverse alphabetical order). It must also be noted that the qualitative DNN verification is an active research domain, with continual progress and annual tool competitions since 2020 [BMB+23].



(2017): Simplex algorithm based SMT solver proposed for verification of ReLU-based Neural Networks
(2017): Use of SMT solver for verification of medium-sized networks
(2018-2020): Branch-and-Bound Heuristics for Verification
Complete
Incomplete
Complete & Incomplete
Our Contribution
(2010-2012): First use of SMT Solvers for MLP verification
(2017): Verification of Perceptron Convergence Theorem using Theorem Prover
(2018): BNN Verification using SAT Solvers
(2018): Bisection of Over-approximated ReLU nodes
(2017-2021): Verification based on Output Reachable Set
(2023): Complete & Incomplete Symbolic Model Checking
Theorem Proving
Satisfiability Solving
Model Checking
Linear Programming
(2015): Verification of MLPs in closed-loop control applications
(2016): Neural Network Verification using a Linear Programmer
(2017): Combined use of SMT and Linear Programming Solvers
(2017-2020): Leveraging Big-M technique for Verification
(2018-2022): Zonotope/ Polytope Abstraction of Neural Networks for incomplete, yet scalable, verification
(2019): First BDD-based BNN Verification
(2019): First Incomplete BMC leveraging CUDA
(2020): First complete SMT-based BMC
(2020): Neuron Merging for scaling Verification
(2021): Verification of Quantized Networks
2010
2023

Figure 2.2: Research Progress in Qualitative DNN Analysis to date.

Theorem Proving

Theorem proving is a type of formal verification in which the system and its properties are defined mathematically, and the properties are verified for the system by rules of natural deduction [CW96]. Generally, for propositional logic and simple circuits, for instance, the half-adder shown in Fig. 2.3(left), the state-of-the-art theorem provers are able to verify the system without human intervention, i.e., these systems can be verified by automatic theorem provers. However, for complex systems, like DNNs (as shown in Fig. 2.3(right)), human guidance is essential, and hence verification of such systems is done via interactive theorem proving. This speaks for the complexity of theorem proving-based DNN verification.

For verification, the system is represented as a logical model governed by mathematical principles. The property is similarly expressed as a formal proof goal. The objective is to use axioms and rules derived from these axioms to check if the properties, i.e., system specifications, hold for the system model, i.e., the implementation.

As expected from a human-guided verification approach, interactive theorem proving is difficult to execute for two reasons. First, it requires an in-depth knowledge of the underlying system for realistic system modeling. Second, it demands the human verifier to have an expert understanding of 1) why a certain property holds for the system, 2)

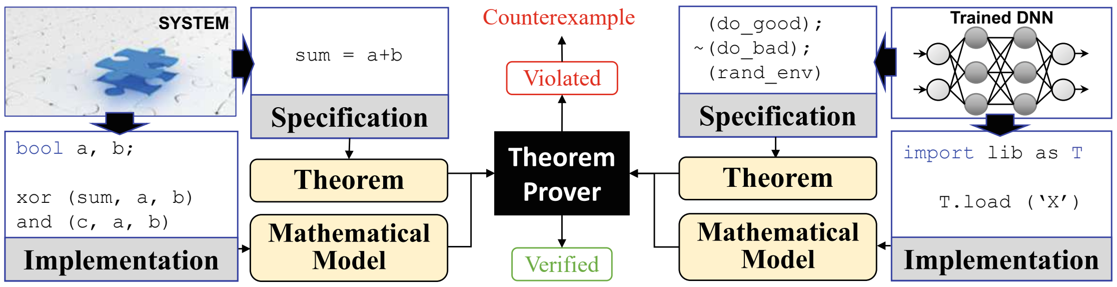


Figure 2.3: Basic theorem proving (left) and for DNNs (right).

what are the required assumptions, and 3) how to prove the property on the basis of sound mathematical principles. [MGS17] verifies the perceptron convergence theorem, but the work focuses on a very small subset of DNNs called binary classifiers, and hence may not be easy to adopt for large state-of-the-art DNNs. For more practical theorem proving-based DNN verification approaches, the basic need is to understand how DNNs work, why they make certain decisions, and what are the mathematical reasons behind their behavior. The perceptron convergence theorem [Ros57, Ros61] was proposed almost six decades before it was formally verified by [MGS17].

More recently, an early attempt to verify small network have been proposed [BS23]. Nevertheless, understanding and developing the theory behind DNN operations seems to be a logical step before theorem proving can be successfully employed for practical DNN verification.

Satisfiability Solving

SAT solving is the branch of formal verification where the system model and the property to be verified for the system are expressed in propositional logic, and written into Conjunctive Normal Form (CNF) [BCC+09]. The formula is then checked by an automatic SAT solver, as shown in Fig. 2.4. Having a *satisfiable (SAT)* output implies that a satisfying solution to the negation of the property, i.e., a counterexample indicating that the property does not hold for the system model, has been found. An *unsatisfiable (UNSAT)* output implies the absence of any counterexample, and hence indicates that the stated property holds for the system model. SMT is a variant of SAT that works similar to SAT solving but allows the use of theories beyond propositional logic, for instance, finite linear arithmetic.

Since SAT solving allows the use of only propositional variables (i.e., atoms), it is often the verification approach of choice for BNN [NKR+18, CNHR18]. Due to the rising popularity and practical efficiency [HCS+16] of BNN in ML systems, these efforts are of relevance to the research community. The network is first formalized into integer/mixed-integer linear program, which is then transformed to CNF via sequential encoding [NKR+18].

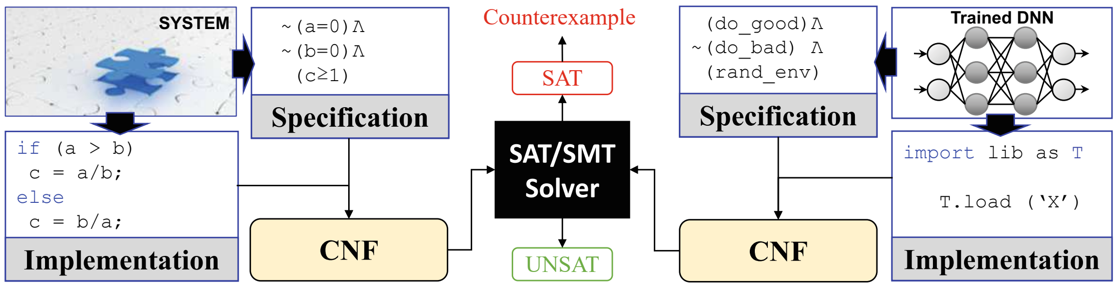


Figure 2.4: Basic SAT solving (left) and for DNNs (right).

An implementation-specification miter[1] is then used for the verification of small and medium-sized BNNs [CNHR18].

SMT solvers, on the other hand, have been the preferred choice for verifying DNNs with real and/or integer network parameters. The early attempts using SMT solvers involved the discretization of sigmoid activation functions [PT10, PT12, SWWB15]. This was inspired by Counterexample Guided Abstraction Refinement (CEGAR) [CGJ$^+$03]), which refines the verification results by iteratively improving the network model using counterexamples provided by the verification engine. HySAT, MathSAT, and Yices [FH07, ABC$^+$02] were used on the backend for the verification of networks with ~ 10 network nodes in these works.

The next wave of SMT-based efforts targeted DNNs using piecewise linear activation functions (like Rectified Linear Unit (ReLU)) [KBD$^+$17, HKWW17, KHI$^+$19]. Additional concepts, like simplex algorithm, were also leveraged in the works. Ultimately, the efforts were able to verify DNNs with up to $20,000$ nodes. To address the scalability challenge, approaches like K-factoring [CNHR18], input domain segmentation [GKPB18] and node merging [EGK20, AHKM20] have been proposed. Abstraction refinement to prune away network nodes and verification of over-approximated networks [OBK22] have also been provided as scalable verification alternatives - however, these make the verification incomplete, possibly leading to false positive verification results.

Nevertheless, SAT-based verification still suffers from the scalability problem. Existing works allow verification of small (less than 10 neurons) to medium (up to $20,000$ neurons) sized networks. The application of scalability solutions, like K-factoring, may also be computationally costly. Another challenge is the design of more efficient SAT/SMT solvers dedicated for DNN verification.

[1]A miter [Bra93] is a circuit configuration where the outputs of two circuits are XORed. Hence, the miter generates logic-1 only if the two circuits generate different outputs. In SAT solving, the outputs XORed lead from the implementation and specification(s) of the system/network under verification.

Model Checking

Model checking (as shown in Fig. 2.5) is an automated formal verification approach, whereby specifications are rigorously checked for a formal model/implementation. The model is given as a state-transition system, for instance, a Kripke structure, while the specifications are provided in the temporal logic [BCC+09]. Similar to the SAT-solving procedure stated earlier, the verification tool returns *UNSAT* (in case the specification holds) or *SAT* (property violation).

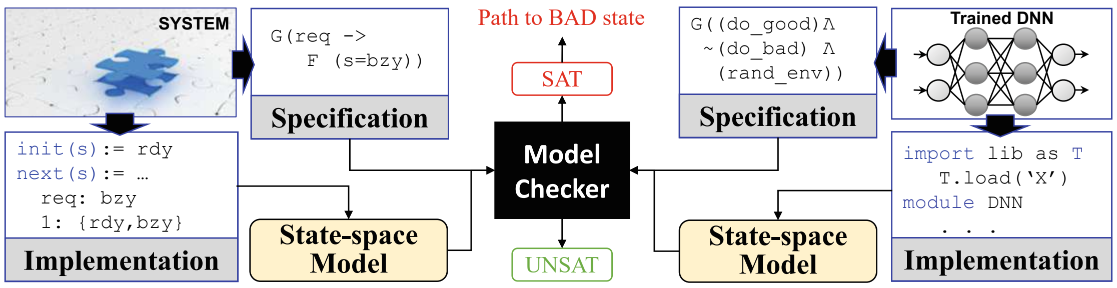


Figure 2.5: Basic model checking (left) and for DNNs (right).

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. Among the popular ones include: partial order reduction [BK08], counterexample guided abstraction refinement (CEGAR) [CGJ+03], and bisimulation [Mil82].

While an established branch of formal methods, MC has only recently been explored for DNN verification. These include leveraging tseitin encoding for the formalization of BNNs into Ordered Binary Decision Diagram (OBDD) [SDC19b], and the use of bounded model checking for the verification of networks with sigmoid activation. The formalization of the sigmoid activation uses lookup table-based function discretization, hence making the formal model incomplete (i.e., probable false positives in the results) [SBG+19]. Further verification of DNNs with quantized network parameters and look-up table-based discretization for non-linear activation functions, using SMT-based model checking, have also been explored [SMS+21]. Yet, the scalability challenge still persists for the existing works.

Linear Programming

With the success of DNNs with piecewise linear activations, particularly ReLU activation [NH10], in real-world DNNs, LP has become a natural choice for numerous DNN verification efforts. LP works by defining the system as a set of linear constraints, and the property to be verified as an objective function, as shown in Fig. 2.6. The objective function can be either a minimization or a maximization function. The search of the minima/maxima is automatic, and involves the use of linear programmers [Gur, cpl].

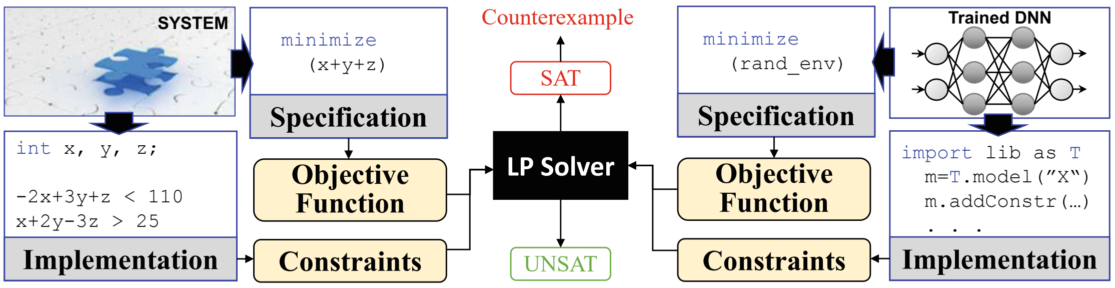


Figure 2.6: Basic linear programming (left) and for DNNs (right).

The earlier LP verification efforts were targeted at checking the robustness of the network against adversarial attacks. The objective was to determine the smallest noise (or noise margin) that satisfies linear constraints of the network but causes misclassification at network output [BIL+16, TXT19]. Orthogonally, the iterative application of LP and SMT for bound tightening and verification, respectively [Ehl17] was also experimented.

The use of Big-M encoding [Gro02] is proposed in several recent works [LM17, DJST18, FJ18, ALMP18, BKK+20] for DNN verification. Here, a binary indicator variable Y is added to the linear constraints to indicate the linear region of the activation function to which the constraint belongs, while M provides a valid output upper bound that is greater than the maximum output value of every ReLU node in the network.

Another concept common to the ReLU-based DNN verification is the branch and bound (BaB) heuristics. The BaB heuristics aim to determine the optimal location to segment input domain and/or activation functions to ensure that the DNN can in turn be represented as a set of linear constraints. A naive BaB implementation has an exponential worst-case complexity [TDG09], and hence would not scale well for DNNs with large number of neurons. Several works [BTT+18, BLT+20, TPL+21] have explored different strategies for BaB in DNNs, while attempting to reduce the time cost of the input segmentation and analysis. However, there is no optimal BaB heuristic for all DNN [BTT+18, BLT+20]; depending on the input domain/range and the DNN model/architecture, a different BaB heuristic may provide better linear constraints for the analysis. For non-ReLU activation functions, Taylor expansion can alternatively be deployed to transform the activation into a piecewise function [LYC+19]. The verification of DNN with this new piecewise activation again follows the same BaB principle. Ideally, for optimal speed of verification, the number of linear segments must be kept minimum [RRCST19].

A common problem with the aforementioned LP-based approaches is their focus on identifying linear regions to formulate linear constraints: this is a time-consuming process. Moreover, the segmentation of ReLU to obtain two linear regions increases the number of linear constraints in the DNN, hence augmenting the scalability problem associated with DNN verification. To overcome these problems, *convex relaxation* of the ReLU activation is used to over-approximate the piecewise linear function into a single linear function, bounding box, zonotope or polytope [GMDC$^+$18, SGM$^+$18, XTJ18, XTJ17, SGPV19, TLM$^+$19, TPM$^+$21, RRCST19, YLL$^+$21a, MMS$^+$22]. Although the transformation is sound [GMDC$^+$18, SGM$^+$18, SGPV19], the over-approximation of ReLU sacrifices the completeness of verification results.

To reduce the effect of incompleteness (i.e., the false positives in results), techniques like symbolic interval analysis [WPW$^+$18b, WPW$^+$18a, LLY$^+$19] and backward substitution [SGPV19, PWW20] are used for tightening output bounds, and hence reducing the over-approximation error. These can be combined with the additional slack variables in the objective function to prioritize the nodes that need to be dealt with earlier [KFS21], ideally reducing the timing overhead of the verification. The merging of network nodes to convert DNN to an interval DNN has also been found to potentially scale the verification [PRA19].

Output reachable set estimation via bound propagation for safety specifications is another popular DNN verification approach using LP [TPL$^+$21, TYML$^+$20]. Depending on whether the propagation is exact [TPM$^+$21, TLM$^+$19] or over-estimated [XTJ17, XTJ18], the verification can be deemed complete or incomplete. More recently, semi-definite programming has also been proposed for DNNs (with feedback loop) [FMP20], which leverages quadratic constraints instead.

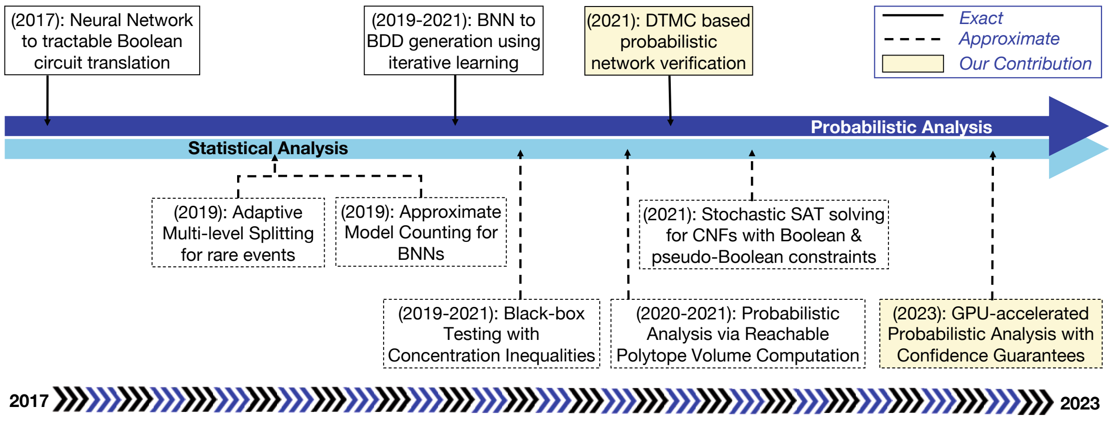
2.2.2 Quantitative Efforts

As indicated earlier, the qualitative efforts described in the previous subsection provide a binary result. As observed in numerous works [SZS$^+$13, PCYJ17, Fea22], trained DNNs are indeed vulnerable to input variations. The likely result of the qualitative efforts is then to simply state the obvious, i.e., the desired DNN specification does not hold for the trained DNN model. This calls for more precise verification results, which may enable deducing the degree to which the DNN does/does not delineate the desired behavior. This is a fairly recent research dilemma for the DNN verification community, with only limited efforts (as summarized in Fig. 2.7) to date.

Probabilistic Methods

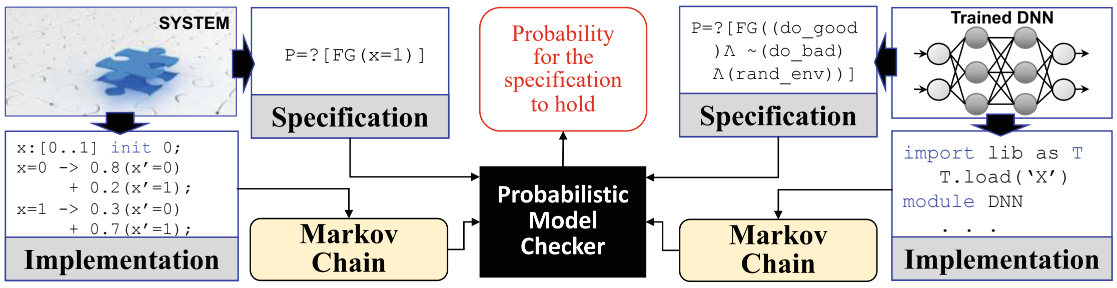
The major branch of quantitative verification efforts is the probabilistic verification [KNP18]. In general, the idea here is to explore the entire valid input domain to calculate the precise probability of specification(s) holding for the given implementation (formal model). The implementation is usually presented as Markov chain, while the specifications are stated in the temporal logic (as shown in Fig. 2.8)[2]. Unlike qualitative efforts, which generally terminate as soon as the first counterexample to specification violation is found, the search of the entire input domain makes probabilistic verification computationally harder.

[2]Note that, since the formal model in probabilistic verification is a Markov chain, and the specifications checked for the model operate temporally, it is common to see the use of terms *probabilistic verification* and *probabilistic model checking* interchangeably.



(2017): Neural Network to tractable Boolean circuit translation
(2019-2021): BNN to BDD generation using iterative learning
(2021): DTMC based probabilistic network verification
Exact
Approximate
Our Contribution
Probabilistic Analysis
Statistical Analysis
(2019): Adaptive Multi-level Splitting for rare events
(2019): Approximate Model Counting for BNNs
(2021): Stochastic SAT solving for CNFs with Boolean & pseudo-Boolean constraints
(2019-2021): Black-box Testing with Concentration Inequalities
(2020-2021): Probabilistic Analysis via Reachable Polytope Volume Computation
(2023): GPU-accelerated Probabilistic Analysis with Confidence Guarantees
2017
2023

Figure 2.7: Research Progress in Quantitative DNN Analysis to date.



SYSTEM
P=?[FG(x=1)]
Specification
Probability for the specification to hold
P=?[FG((do_good)∧ ~(do_bad) ∧(rand_env))]
Specification
Trained DNN
x:[0..1] init 0;
x=0 -> 0.8(x'=0) + 0.2(x'=1);
x=1 -> 0.3(x'=0) + 0.7(x'=1);
Implementation
Markov Chain
Probabilistic Model Checker
Markov Chain
import lib as T
T.load('X')
module DNN
. . .
Implementation

Figure 2.8: Basic probabilistic model checking (left) and for DNNs (right).

Owing to the complexity of this quantitative verification branch, it has only been scarcely explored for DNN verification. The prominent efforts include the transformation of small networks to boolean circuits, which can, in turn, be transformed to OBDD and eventually Sentential Decision Diagram (SDD) for the probabilistic verification [CSSD17]. Alternatively, the transformation of networks (particularly BNNs) have also been used for Binary Decision Diagram (BDD) generation. This is followed by an iterative learning algorithm [SDC19a] or sequential analysis [ZZC^{+}21]. Model counting, i.e., counting the SAT and UNSAT instances of the formal model for given specifications, is then used for exact probability computing for the desired DNN properties, like safety. Added to the existing challenge of scalability of the aforementioned DNN transformation is then the #P complete computational complexity of the model counting.

Statistical Methods

Similar to sacrificing completeness for scalability in qualitative efforts, statistical verification achieves the same for quantitative verification. The formal model is again composed of Markov chain and the specifications formalized in temporal logic (see Fig. 2.9)[3]. However, the probability computed is no longer exact, but only an estimate of the exact result [LLT+19].

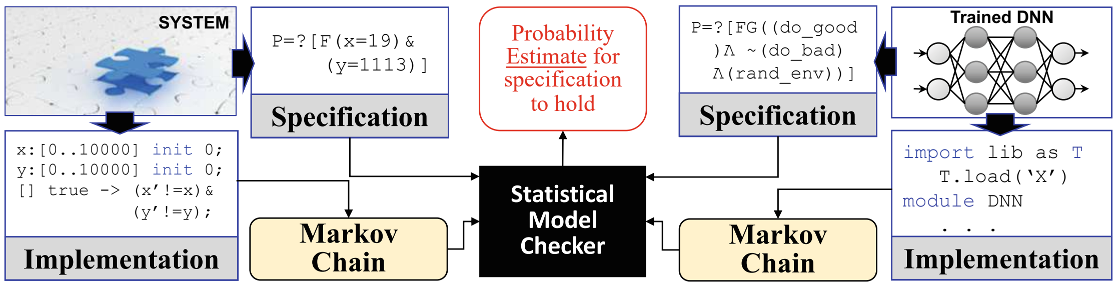


Figure 2.9: Basic statistical model checking (left) and for DNNs (right).

Compared to probabilistic verification, statistical verification has been able to achieve better scalability in terms of DNN verification. Notable efforts include the use of importance sampling and its variants to discover rare events, with known or assumed underlying probability distribution. For instance, AMLS [WRTK19] has been successfully implemented for DNN analysis, while using a combination of importance sampling with constraint solving (for fast convergence) has also been found to be useful [LFZ21]. However, it is important to note that importance sampling, in essence, is not a verification approach to estimate probability, but merely an analysis strategy for checking the occurrence of rare events. The use of concentration inequalities has also been leveraged in multiple works [BZSL19, FMP19, BCMS21], which provide probability estimates for specification to hold. Hypothesis testing has also been adapted to provide statistical results for the DNN specifications to hold [FMP19, BCMS21]. The trained DNNs are treated here as black-boxes.

Similar to the case in probabilistic verification, model counting has also been explored in terms of statistical verification. Here, the approximate counting is used to provide PAC-style guarantees for BNNs [NSM+19, BSS+19]. Another statistical variant of the classical formal methods is the stochastic SAT, which has been used for estimating (fairness) probabilities within error bounds [GBM21]. However, this approach is applicable only to boolean and linear classifiers with pseudo-boolean constraints, which can be presented in CNF.

Among the non-conventional statistical DNN verification includes the computation of reachable polytope volume, to in turn compute reachability probability estimates [CFGP20, PCFG20, YLL+21b]. The works rely on the (unrealistic) assumption that all node distributions are *uniform*.

[3]Similar to the probabilistic case, *statistical verification* and *statistical model checking* are often used interchangeably as well.

2.3 Summary of Related Work

Overall, the efforts to obtain behavioral guarantees for trained DNNs can be broadly partitioned into empirical analysis and formal verification efforts. The empirical efforts aim to identify the undesirable DNN behavior (possibly for a specific output class), in the presence or absence of input perturbations. They are also employed to reduce DNN complexity (for instance via pruning) while having minimal impact on DNN behavior. The formal verification, on the other hand, takes a more holistic approach for ensuring behavioral guarantee of the trained DNN as their prime focus. Among them, the qualitative approaches use classical techniques like satisfiability solving and linear programming for behavioral verification of DNN models. To provide more precise behavioral guarantees, quantitative efforts compute (exact or approximate) probability for the DNNs to delineate the desired behavior.

The empirical analysis provides limited completeness of results. On the other hand, formal verification suffers from limited scalability. The following chapters focus mostly on the formal verification aspects of providing behavioral guarantees for trained DNNs, starting with the appropriate formalism for DNN implementation and specifications (in Chap. 3).

CHAPTER 3

Formalism

Numerous aspects pertaining to DNNs do not have well-defined definitions, allowing subjective interpretations of the notions. This chapter aims to eliminate such discrepancies in these notions by providing formal definitions of the concepts relevant to this research. Table 3.1 lists the notations, and their corresponding relations, used throughout this thesis to denote various aspects of DNNs and their properties.

	Notation		Notation
Trained DNN	$\mathcal{N}$	Output Domain	$\mathcal{Y}$
Number of DNN Layers	L	Output Label	$L(\mathcal{X})$
Input Domain	$\mathcal{X}$	Number of Output Classes	$\mathcal{C}$
Input Dataset	$D(\mathcal{X})$	Noise Bounds	Δx
Number of Input Nodes	n	Maximum Noise Bounds	Δx_{max}
Applied Noise	η		

where:

$\mathcal{X} \in \mathbb{R}^n$	$\mathcal{Y} \in \mathbb{R}^{\mathcal{C}}$
$D(\mathcal{X}) \subset \mathcal{X}$	$\forall x \in \mathcal{X} : \mathcal{N}(x) \in \mathcal{Y}$
$\mathcal{N} : \mathcal{X} \rightarrow \mathcal{Y}$	$L(\mathcal{X}) = one\_hot\_encode(\mathcal{N}(\mathcal{X}))$
$\mathcal{C} = \lvert L(\mathcal{X}) \rvert$	$\eta \leq \Delta x \leq \Delta x_{max}$

Table 3.1: Notations and relations used for DNN formalism throughout the thesis.

©2023 Springer. Parts reproduced, with permission, from M. Naseer, B. S. Prabakaran, O. Hasan, M. Shafique: UnbiasedNets: a dataset diversification framework for robustness bias alleviation in neural networks, *Machine Learning (Mach Learn 2023)*, pp. 1-28. [DOI:`doi.org/10.1007/s10994-023-06314-z`].

©2023 IEEE. Parts reproduced, with permission, from M. Naseer, M. Shafique: Poster: Link between Bias, Node Sensitivity and Long-Tail Distribution in trained DNNs, *Conference on Software Testing, Verification and Validation (ICST)*, Dublin, Ireland, 16-20 April, 2023, pp. 474-477. [DOI:`doi.org/10.1109/ICST57152.2023.00054`].

Prior to defining concepts related to trained DNNs, it is worthwhile to understand the notion of balanced dataset. Such datasets, in particular balanced training dataset, defines a dataset where each output class in the dataset is equally-representative. This ensures that the DNN, once trained on this dataset, delineates minimal bias. More formally, such datasets can be defined as follows.

Definition 1 (Balanced Dataset)**.** *Given a dataset $D(\mathcal{X})$ with $\mathcal{C}$ output classes, the dataset is said to be balanced/the output classes are equally represented iff density ρ of inputs from each class in the input hyperspace is (approximately) equal, i.e., $\rho(D(\mathcal{X}_1)) \approx \rho(D(\mathcal{X}_2)) \approx ... \approx \rho(D(\mathcal{X}_\mathcal{C}))$. Note that density ρ of input here refers to the average number of input samples contained within the unit hypervolume of the valid input domain $\mathcal{X}$ for an output class, and $\mathcal{X}$ belongs to the Euclidean space $\mathbb{R}^n$.*

Rest of this chapter provides the necessary formalism required for developing DNN formal model and their corresponding specifications. In particular, Sec. 3.1 provides the *functional* formalism corresponds to the implementation of the trained DNNs (i.e., the formal model). Sec. 3.2 provides the *behavioral* formalism, i.e., the formalization of DNN behavioral properties. Note that, the behavioral properties of DNNs point to the desired DNN specifications - hence, the terms specifications and properties can be used interchangeably. To study the behavior of DNNs in practical setting, it is often desirable to consider the impact of environmental noise on the network inputs. The formalism of such noise is provided in Sec. 3.3. Sec. 3.4 finally summarizes the important aspects of the chapter.

3.1 Functional Formalism of Neural Networks

This research focuses on the feed-forward networks, i.e., the networks with no loops connecting nodes of a layer to those of preceding layer(s). More formally, such networks can be defined as follows.

Definition 2 (Feed-forward neural network)**.** *Given input domain $\mathcal{X}$, a feed-forward network $\mathcal{N} : \mathcal{X} \to \mathcal{Y}$ maps the input to the output domain $\mathcal{Y}$ 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 $y = \mathcal{N}(x) \; = \mathcal{N}^L \; (\; \mathcal{N}^{L-1} \; (... \; \mathcal{N}^1 \; (x) \; ... \;))$, where $\mathcal{N}^k$ encapsulates the linear and non–linear transformations for layer k, $x \in \mathcal{X}$ is an input from the input domain, and $y \in \mathcal{Y}$ is an output class from the output domain.*

The network is also fully-connected if each node in each network layer is connected to every node in the adjoining layer. As briefly pointed out in the previous chapter, the nodes are in each layer are connected to those of an adjoining layer via a sequence of linear and non-linear transformation. For fully-connected networks, this linear transform is an affine relation presented mathematically as:

$$y_j^k = b^k + \sum_{i=0}^{N} w_{ij}^k x_i^{k-1} \tag{3.1}$$

Here, w_{ij}^k represents the weight connecting node i from layer $k-1$ to node j in the layer k, while b^k represents the vector of bias parameter values associated with layer k. x_i^{k-1} is the input originating from node i of the layer $k-1$, while the result of the affine transform for node j in layer k is saved in y_j^k.

The transformed result y_j^k then goes through the non-linear activation. The result of this next transformation is eventually the input for the subsequent network layer. Among them, ReLU is a popular choice, which works analogous to half-wave rectifier in an electrical circuit, clipping negative inputs to zero while passing through the positive inputs unaltered.

$$x_i^k = max_{i \in \mathbb{Z}^+ : i \leq \mathrm{n}}(0,\ y_i^k) \tag{3.2}$$

Another choice for activation function is the maxpool, which passes the maximum among input node values to the output.

$$x^L = max_{i \in \mathbb{Z}^+ : i \leq \mathrm{n}}(y_i^L) \tag{3.3}$$

3.2 Behavioral Formalism of Neural Networks

In general, it is desired for a trained DNN to compute correct results for all inputs, under all possible input conditions. However, for *verification* or *analysis*[1] of the DNN, it is helpful to be more precise regarding the desired DNN behavior. The following provides the formalism of the DNN specifications considered in this thesis, to provide the behavioral guarantees regarding the trained networks.

Definition 3 (Robustness)**.** *Given a trained network $\mathcal{N} : \mathcal{X} \to \mathcal{Y}$, $\mathcal{N}$ is said to be robust against the noise Δx if the application of an arbitrary noise $\eta \leq \Delta x$ to the input $x \in \mathcal{X}$ does not change network's classification of x, i.e., $\forall \eta \leq \Delta x : \mathcal{N}(x + \eta) = \mathcal{N}(x)$.*

It must be noted that x corresponds to inputs that the network $\mathcal{N}$ does not originally misclassify, i.e., $\mathcal{N}(x)$ corresponds to the true output class for input x.

Definition 4 (Noise Tolerance)**.** *Given a trained network $\mathcal{N} : \mathcal{X} \to \mathcal{Y}$, noise tolerance is defined as the maximum noise Δx_{max}, which can be applied to a correctly classified input $x \in \mathcal{X}$ such that $\mathcal{N}$ does not misclassify the input. Hence, for any arbitrary noise $\eta \leq \Delta x_{max}$, the application of noise to an input $x \in X$ does not change the network's classification of x, i.e., $\forall \eta \leq \Delta x_{max} : \mathcal{N}(x + \eta) = \mathcal{N}(x)$.*

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

[1](Formal) Verification is the means to mathematically check whether the specifications hold for the system's formal model. (Empirical) Analysis, on the other hand, is part of the testing methodology where the results from the system are used to deduce whether the desired specifications hold for the system or not.

Definition 5 (Robustness Bias)**.** *Given a dataset $D(\mathcal{X}) \subset \mathcal{X}$ with $\mathcal{C}$ output classes, and $\mathcal{X}_1, \mathcal{X}_2, ..., \mathcal{X}_\mathcal{C}$ as the of input hyperspace representing each output class. $D(\mathcal{X})$ is said to exhibit robustness bias iff the hyperspaces $\mathcal{X}_1, \mathcal{X}_2, ..., \mathcal{X}_\mathcal{C}$ are not equidistant from the decision boundary. Note that, the decision boundary is the hyperplane separating the hyperspace belonging to the individual output classes.*

However, as long as the hyperspaces are equidistant from the decision boundary, the dataset is said to be free from a robustness bias.

Definition 6 (Biased Network)**.** *Given a trained network $\mathcal{N} : \mathcal{X} \rightarrow \mathcal{Y}$, $\mathcal{N}$ is said to be biased if the application of an arbitrary noise $\eta \leq \Delta x$ to any (correctly classified) input from class $i \in \mathcal{C}$ does not change network's output classification, $\forall \eta \leq \Delta x, x \in \mathcal{X}_i : \mathcal{N}(x + \eta) = \mathcal{N}(x)$. However, application of the same noise to any input from another class $j \in \mathcal{C}$ makes the network misclassify the originally correctly classified input from the class $\forall \eta \leq \Delta x, x' \in \mathcal{X}_j : \mathcal{N}(x' + \eta) \neq \mathcal{N}(x')$.*

It must be noted that even though unbiasedness (i.e., the property of a trained NN to be unbiased) and classification accuracy may intuitively seem similar, they are not identical. Obtaining an accurate DNN involves identifying the decision boundary that *separates* the output classes in the dataset. In contrast, obtaining an unbiased NN involves identifying a decision boundary that is *equidistant* from all the hyperspaces encapsulating the different output classes. The resulting unbiased network, in turn, may or may not have the highest classification accuracy. However, all the output classes will likely be equally robust to noise in an unbiased network.

Definition 7 (Node Sensitivity)**.** *Given a network $\mathcal{N} : \mathcal{X} \rightarrow L(\mathcal{X})$, where each input $x \in \mathcal{X}$ comprises of* n *input nodes, node sensitivity determines the robustness of individual input nodes under the incidence of the node noise $\eta \leq \Delta x$, i.e., node $i \in$* n *is insensitive to noise iff $\mathcal{N}(x) = \mathcal{N}(x \setminus x^i, x^i + \eta)$.*

In principle, an input node may be sensitive or insensitive to a specific kind of noise, for instance to the positive noise or the noise bounded by specific constraints.

Definition 8 (Node Robustness Bias)**.** *Given a network $\mathcal{N} : \mathcal{X} \rightarrow L(\mathcal{X})$, where $L(\mathcal{X})$ comprises of $\mathcal{C}$ output classes (i.e., $L(\mathcal{X}) = \{1, ..., \mathcal{C}\}$) and each input comprises of n input nodes, node robustness bias defines the robustness of individual input nodes for each output class. This means, node robustness bias holds for the input node $x_k^i \in x_k$ iff the probability of correct classification for input x_k belonging to all output classes $k \in L(\mathcal{X})$, when noise $\eta \leq \Delta x$ is incident to node x_k^i, is equal, i.e., $\forall k \in L(\mathcal{X}), \forall i \in n : \mathbb{P}[\mathcal{N}(x_k) = \mathcal{N}(x_k \setminus x_k^i, x_k^i + \eta)] = const.$*

The intuition behind the analysis of node (robustness) bias is to ensure that each input node has a consistent sensitivity for inputs belonging to all output classes.

3.3 Formalizing Noise

To study the behavior of trained DNNs in realistic settings, it is desirable to consider the impact of environmental uncertainties like noise on the network input. This is in line with the DNNs in real-world applications since the DNNs deployed in practical applications often receive noisy inputs via data acquisition and pre-processing. The final (noisy) input is then normalized to limit the bounds of inputs to remain within $[0, 1]$ range. Eventually, the input is fed to the trained DNN.

3.3.1 L^p Norm based Noise Models

This is the popular noise modeling for DNN analysis and verification. However, the noise is added to the normalized inputs, while implicitly assuming the unnormalized input to be clean from noise. The noise to be added to the normalized inputs can essentially be modeled in multiple ways. But the general L^p norm-based noise can be obtained via the following relation [Car00]:

$$||\Delta x||_p = \sqrt[p]{\sum_i |\Delta x_i|^p} = \sqrt[p]{\sum_i |x'_i - x_i|^p} \tag{3.4}$$

L^1 Norm

Also known as the *Manhattan distance*, this noise model computes noise as the sum of absolute distances between nodes of true and perturbed inputs. Naturally, this only involves linear operations and is hence simple to implement.

L^2 Norm

Also known as the *Euclidean distance*, this noise model is computationally harder compared to the previous noise model. Owing to the squaring operations involved, it magnifies the distance between true and perturbed inputs. Hence, compared to L^1 norm, L^2 norm provides a less robust measure of distance between the inputs. This means that even a small magnitude of distance is magnified in L^2 norm due to the squaring operations.

L^∞ Norm

This is the simplest and most popular noise model used in literature. It gives the maximum difference between true and perturbed inputs:

$$||\Delta x||_\infty = max_i(\Delta x_i) = max_i(|x'_i - x_i|)$$

©2022 ACM. Parts reprinted, with permission, from I. T. Bhatti, M. Naseer, M. Shafique, O. Hasan: A formal approach to identifying the impact of noise on neural networks, *Communications of the ACM (CACM 2022)*, pp. 70-73. [DOI:`doi.org/10.1145/3550492`].

As shown in Fig. 3.1, L^∞ norm encapsulates all other L^p norms. This means that the noise explored under L^p norm, for $p < \infty$, is also explored for L^∞ of the same magnitude.

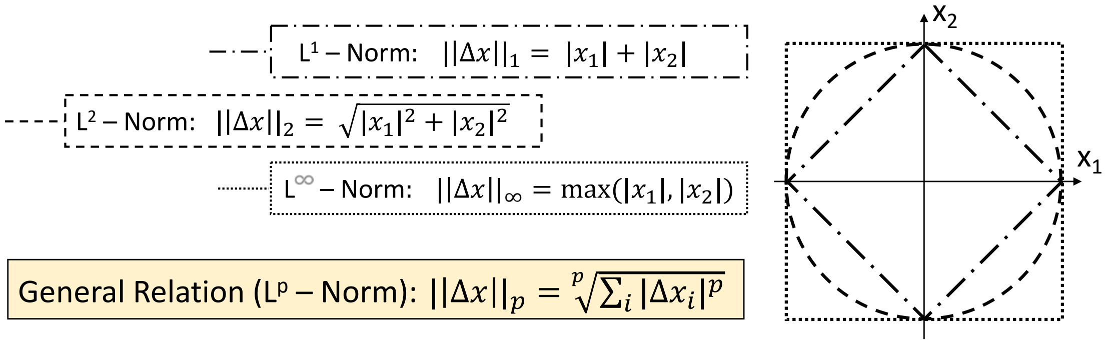

L¹ – Norm: $||\Delta x||_1 = |x_1| + |x_2|$
L² – Norm: $||\Delta x||_2 = \sqrt{|x_1|^2 + |x_2|^2}$
L^∞ – Norm: $||\Delta x||_\infty = \max(|x_1|, |x_2|)$
General Relation (Lᵖ – Norm): $||\Delta x||_p = \sqrt[p]{\sum_i |\Delta x_i|^p}$
x_2
x_1

Figure 3.1: Visualization of L^p norm-based noise for two-dimensional input domain.

3.3.2 Relative Noise Models

As stated earlier, the L^p norm-based noise is applied to the normalized inputs. This is often unrealistic since noise is more likely to impact raw/unnormalized noise. Additionally, for applications where input nodes receive data from domains with varying input ranges, reverse engineering the impact of the same $||\Delta x||_p$ is likely to differ for the different input nodes. Hence we instead propose the relative noise model, whereby noise is added proportionally to the magnitude of input, to the unnormalized inputs. This provides a more realistic interpretation of noise for the verification.

$$\Delta x_i = 0.01 \times \epsilon \times x_i \tag{3.5}$$

The factor 0.01 ensure the magnitude of noise is significantly smaller than the input x_i, to reflect practical world (where magnitude of noise is in fact much smaller than the magnitude of input). Hence, the noise Δx_i applied to node x_i is ϵ% of the input.

3.4 Summary of Formalism

This chapter provided the formal definitions for DNNs considered in this works, the specifications studied to ensure desirable DNN behavior, and the noise models used to verify DNNs in practical settings. The next chapters will provide details of the qualitative and quantitative formal analysis frameworks developed in this work, based on the formalism provided in this chapter.

CHAPTER 4

Qualitative Neural Network Analysis

As indicated in the previous chapters, earlier attempts to ensure correct functioning of DNNs included empirical approaches, for instance using gradient-based methods [SZS+13, MDFFF17] to identify the adversarial noise patterns that would lead the DNN to misclassify benign inputs. Although such attempts provide evidence to the lack of robustness of DNNs, they are insufficient to provide any guarantees regarding DNNs' robustness in the case when no adversarial noise is found. To deal with the aforementioned problem, in recent years, there has been a great interest towards the rigorous evaluation of DNNs using formal verification [TXT19, KHI+19, BLT+20]. This usually involves checking resilience properties, like robustness and safety, of the DNNs using SAT checking or LP. However, the exploration of formal approaches beyond SAT and LP, to analyze wider variety of DNNs' properties, remains largely neglected.

To deal with the limitations of existing literature to explore formal tools for DNN verification and handle diverse range of DNN properties, this chapter provides the step-by-step procedure for qualitative DNN analysis. This is summarized in Fig. 4.1, while the novel contributions of the work are listed as follows:

1. Explore the limits of model checking for DNN Analysis.
2. Use realistic and practical noise mode noise model, i.e., the relative noise model (as elaborated in Chap. 3.3.2).
3. Focus on diverse range of properties namely: robustness, noise tolerance, robustness bias, input nodes sensitivity and safety.
4. Combine the pros of classical formal verification and empirical analysis to develop *formal analysis* frameworks, FANNet and FANNet+.

5. Present initial framework FANNet, to achieve aforementioned targets.

6. Propose novel input segmentation methodology, i.e., *random input segmentation.*

7. Develop FANNet+ to improve scalability of model checking-based analysis.

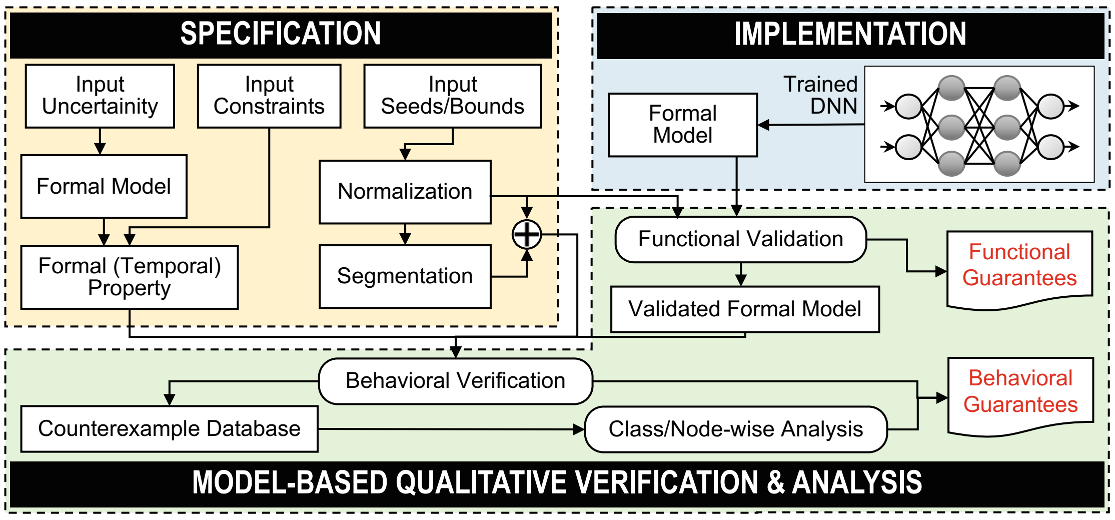


Figure 4.1: The overall qualitative analysis methodology presented in this chapter.

Sec. 4.1 provides our initial efforts toward developing a model checking-based framework, FANNet, for the DNN formal verification and empirical analysis (henceforth referred to simple as *formal analysis*). Sec. 4.2 later provide methodologies for scaling the framework, hence proposing the extended framework FANNet+. The works are also accompanied by supporting case studies, results and discussions. Sec. 4.4 summarizes our formal qualitative DNN analysis.

4.1 Initial Efforts

As elaborated in previous chapters, verification for ensuring *network robustness* has been an active domain of research for DNNs. These works focus on either expressing the network and its robustness property in CNF, and verifying it using a SAT solver, and/or transforming the problem into a set of linear constraints and objective function, and verifying the network as an optimization problem using a LP solver, under a specified

©2020 IEEE. Parts reproduced, with permission, from M. Naseer, M. F. Minhas, F. Khalid, M. A. Hanif, O. Hasan, M. Shafique: FANNet: formal analysis of noise tolerance, training bias and input sensitivity in neural networks, *Design, Automation & Test in Europe Conference & Exhibition (DATE 2020)*, Grenoble, France, 09-13 March, 2020, pp. 666-669. [DOI:`doi.org/10.23919/DATE48585.2020.9116247`].

$L^p - norm$ space around the seed inputs. Unlike the current literature, we analyze DNN properties beyond network robustness, using model checking.

To conduct the aforementioned studies, we propose a formal analysis methodology, FANNet (summarized in Fig. 4.2), to address the challenges associated with DNN analysis (elaborated in Chap. 1.2). The novel contributions of our initial qualitative analysis include:

1. Formal modeling and analysis methodology to analyze the trained DNN as a state-space model, using model checking.
2. Providing realistic estimates for the network tolerance in the presence of adversarial noise.
3. Studying input node sensitivity in the presence of noise.
4. Analyzing the effects of robustness bias on network accuracy.
5. Providing a comparative analysis of the computational resources, in terms of timing and memory, required by testing and our model checking based methodology for the above DNN analysis.
6. Performing a case study on Leukemia Detection to demonstrate the practical significance of the above analysis.

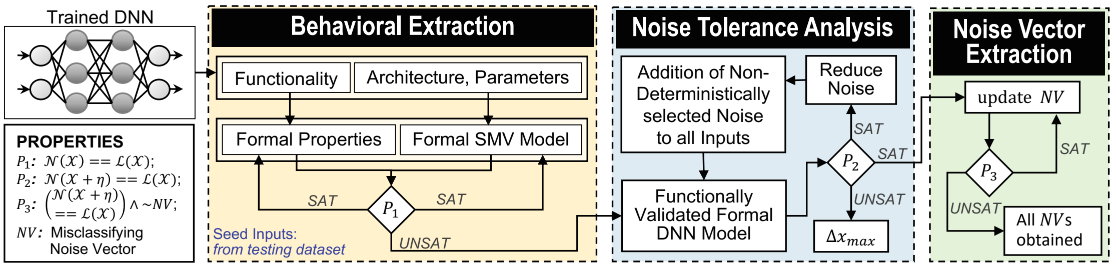

Trained DNN
PROPERTIES
P_1: $\mathcal{N}(\mathcal{X}) == \mathcal{L}(\mathcal{X})$;
P_2: $\mathcal{N}(\mathcal{X}+\eta) == \mathcal{L}(\mathcal{X})$;
P_3: $\begin{pmatrix}\mathcal{N}(\mathcal{X}+\eta)\\ == \mathcal{L}(\mathcal{X})\end{pmatrix} \wedge \sim NV$;
NV: Misclassifying Noise Vector
Behavioral Extraction
Functionality
Architecture, Parameters
Formal Properties
Formal SMV Model
SAT
P_1
SAT
UNSAT
Seed Inputs: from testing dataset
Noise Tolerance Analysis
Addition of Non-Deterministically selected Noise to all Inputs
Reduce Noise
SAT
P_2
SAT
UNSAT
Functionally Validated Formal DNN Model
Δx_{max}
Noise Vector Extraction
update NV
SAT
P_3
UNSAT
All NVs obtained

Figure 4.2: FANNet: the first complete SMT-based bounded model checking framework for formal DNN analysis.

4.1.1 FANNet Methodology

FANNet, the initial qualitative DNN analysis framework, uses model checking to obtain the noise tolerance of the trained DNNs. The counterexamples obtained from model checking are subsequently leveraged to estimate sensitivity of the individual input nodes as well as detect any robustness bias in the trained DNN. The overall framework operates as follows.

Generating Kripke Structure of the Neural Networks

Consider the network shown in Fig. 4.3(top-left). A binary noise[1], i.e., the noise with magnitudes either 0 or 1, is incident to the input nodes of the network. Recall (from Chap. 2.2) that the model checking involves rigorous analysis of the state-transition system. So in case of explicit state model checking, there are four possible ways to apply noise to the input nodes. Hence, the resulting Kripke structure contains four states leading from the initial state, as shown in Fig. 4.3(right).

Additionally, the input can lead to either of the possible outputs, which provide the Atomic Propositions (APs) for the formal model. Hence, a transition from initial state, corresponding to any of the possible noise options, can lead to any possible outputs. This leads to the resulting formal model (i.e., the Kripke structure) to have $1 + |\eta|C$ states, where $|\eta|$ is the number of noise options and C is the number of output classes in the network. Again, a rigorous analysis using model checking entails the possibility of network changing its state. Hence, the number of transitions making up the model adds to $|\eta|C(|\eta|C + 1)$. It must be noted that the hidden nodes are not presented in the formal model. This can be attributed to the use of jump transitions, which allows the compression of states having same set of AP.

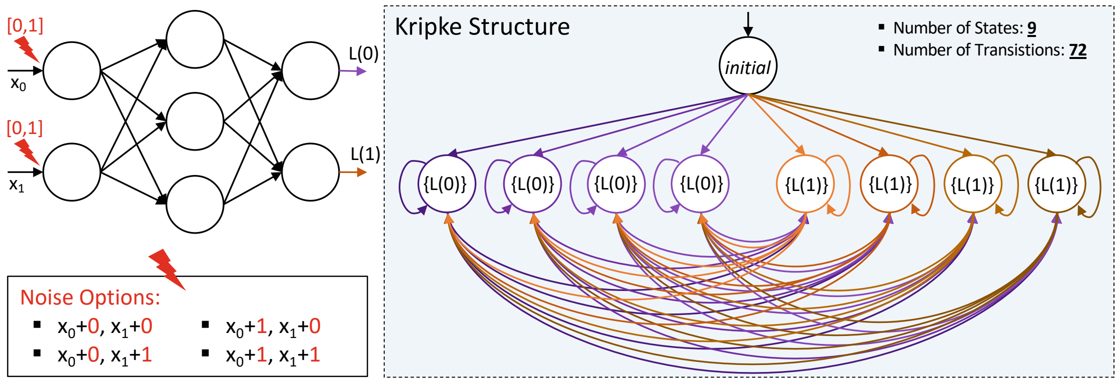


Figure 4.3: The formal model (Kripke structure) for a small fully-connected network (shown on top left), as generated by FANNet. A binary noise is incident to the network's inputs.

Analyzing the Formal Network Model

The DNN testing dataset (with seed input and their true labels) is available at the input to the model checker. The weights and activations of the trained DNN, which determine the network's architecture and functionality, are first translated into the *SMV model*[2] and the temporal properties. Prior to the analysis with input noise, the correctness of

[1]Note that, FANNet uses relative noise in our actual experimental analysis. The use of binary noise in the given proof-of-concept in Fig. 4.3 is solely to ease understandability of the formal model generation.

[2]Modeling standard used by nuXmv model checker to generate Kripke structures

Algorithm 1 Noise Tolerance Analysis using Model Checking

Input: Input Matrix (X), Weight Matrices (w_1, w_2), Mean (μ) and Standard Deviation (ς) of Training Dataset, True output labels ($L(x)$), Noise Range (Δx)
Output: No Counterexamples flag (NC), Adversarial Noise Matrix (e)
Initialize: $Noise = [-\Delta x : +\Delta x]$, $e = \{\}$

```
for $i = 1$:Size($X$) do                                   ▷ Step 7)
    $NC = \bot$; $count = 0$
    while $!NC \mid count \leqslant Size(Noise)$ do
        $NV = \text{rand}(Noise)$                            ▷ Step 2)
        $X_i = X(i) + \frac{X(i) \times NV}{100}$            ▷ Step 3)
        $X_n = \text{normalize}(X_i, \mu, \varsigma)$        ▷ Step 4)
        $OH = \text{Layer1}(x_n, w_1)$                       ▷ Step 5)
        $OC_n = \text{Layer2}(OH, w_1)$
        if $! \ (\mathtt{G} \ \mathtt{X} \ (OC_n = L(x(i)) \mid e))$ then
            $NC = \top$                                      ▷ Step 6)
        else
            $e = NV \mid e$
        end if
        $count = count + 1$
    end while
end for
```

the DNN model without noise is ensured by comparing the model's calculated outputs against, as shown by P_1 in Fig. 4.2.

The formal analysis for noise tolerance, as explained in Algorithm 1, then proceeds as follows:

1. The noise range for the analysis is initialized as a set of integer values. The values are initially chosen large, but the objective is to eventually reduce it to obtain the minimum adversarial noise and ultimately estimate DNN's tolerance.

2. From this specified range, a *unique* noise vector (NV) is non-deterministically selected in each iteration, to be added to the input.

3. The noise is then added to the input as a percentage of the input value, i.e., the relative noise. As indicated earlier, this is a more practical noise implementation approach for most DNN-based systems.

4. As a standard procedure in DNN, the input is normalized prior to sending it to the network layers, using mean μ and standard deviation ς from training dataset:

$$X_n = \frac{X_i - \mu}{\varsigma} \tag{4.1}$$

5. The input then transverses through the network layers.

6. The computed network output OC_n for is then compared to original output label $L(x)$ the input while recording the adversarial noise obtained as counterexamples e. If no counterexample is found, the flag (NC) becomes `true`. This signifies that the input is tolerant to the noise. This is represented by *P2* in Fig. 4.2.

7. The process is repeated for all available inputs in the dataset, until NC becomes `true` for a given noise range.

If $OC_n \neq L(x)$ and the adversarial noise NV is not already contained in e, then the NV obtained from the generated counterexample is added to e, as shown by *P3* in Fig. 4.2 and Line 11 of Algorithm 1. To accelerate the procedure, the adversarial noise from one input is *tested* with the remaining inputs in the dataset. Hence, the NV generated from one input is simultaneously added to e for all the forthcoming inputs. This ensures that the model checker obtains only the *new* adversarial noise patterns in each iteration.

4.1.2 Case Study on Leukemia Diagnosis

Experimental Setup

We considered a shallow fully-connected neural network architecture. It is composed of an input layer, one hidden layer and an output layer, with 6, 20 and 2 nodes respectively. The activations used in the network are ReLU and maxpool. The network is trained using MATLAB with a learning rate of 0.5 for the 40 initial epochs, and a learning rate of 0.2 for the remaining 40 epochs. The training accuracy of the network is 100% and the testing accuracy is 94.12%. It is used to diagnose the type of leukemia using the standard leukemia database [GST$^+$99], consisting of 38 training samples and 34 testing samples containing genetic attributes for Acute Lymphoblast Leukemia (ALL) and Acute Myeloid Leukemia (AML). In total, each data sample has 7129 of these genetic attributes. From these, the top five most significant genes are picked as network inputs using the Minimum Redundancy and Maximum Relevance (mRMR) feature selection method [Kea18].

Varying noise ranges were input to the network, as explained in the previous subsection. The main goals of the analysis are: (a) to determine noise tolerance for the given network by gradually reducing the applied noise until no noise pattern that causes the true label $L(x)$ to change can be found, and (b) to study network properties like robustness bias and input node sensitivity on the basis of the obtained counterexamples. It must also be highlighted that the objective of this work is not to exhaustively search for counterexamples, but rather to explore network properties on the basis of obtained counterexamples.

Results and Analysis

We initiate the experiment using *nuXmv* with a large input noise, and gradually reduce the noise until *nuXmv* can verify the absence of any counterexamples for the given noise. The observations and analysis from our experiments (Fig. 4.4), are as follows:

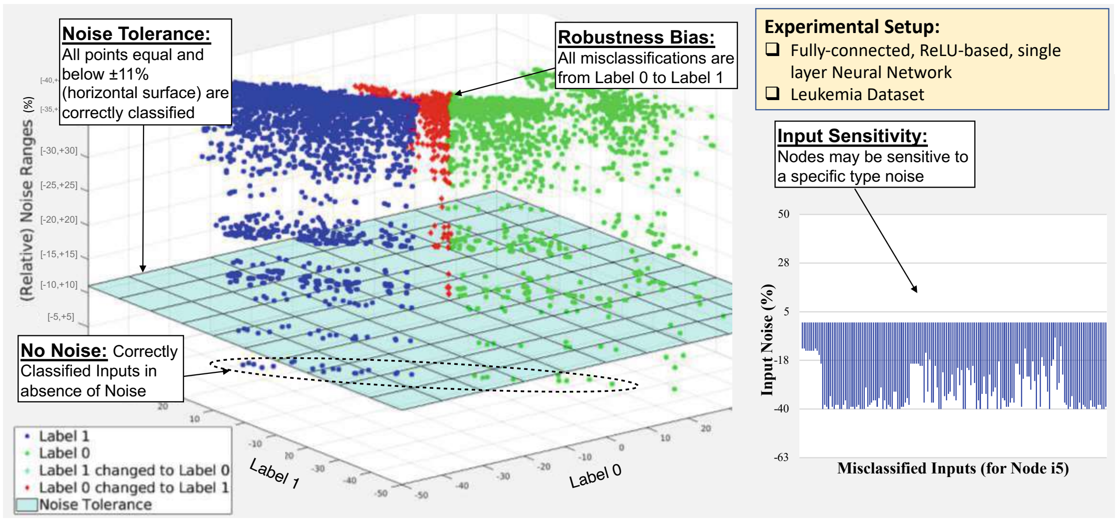


Figure 4.4: Estimated noise tolerance, impact of noise on inputs closer to classification boundary, effects of robustness bias on network accuracy, and sensitivity of input node.

For all the correctly classified inputs in the testing dataset, addition of a noise ±11% or less does not trigger misclassification for the given neural network. Hence, assuming input noise to be the integer percentage values, the given network has an estimated noise tolerance of ±11%. A few inputs among the dataset (i.e., inputs closer to the classification boundary) were observed to be highly susceptible to input noise. On the other hand, for other inputs, noise even as large as 50% of the input did not trigger misclassification at the output. This knowledge can be used to estimate the network's classification boundary in the input hyperspace.

Inputs with $L(x) = \textit{Label } 0$ were observed as more likely to be misclassified than the inputs with $L(x) = \textit{Label } 1$. On a closer inspection of the training dataset, it is observed that approximately 70% of the data samples belong to the *Label* 1 output class i.e., the training is biased towards the *Label* 1. This is corroborated by our formal analysis, where the misclassification of inputs with *Label* 0 are more probable than the misclassification of inputs with belonging to *Label* 1. This points to a robustness bias in the trained network.

No counterexamples were obtained with positive noise at input node i_5. Moreover, the counterexamples suggest more noise patterns with positive noise at input node i_2 than the other way around. The knowledge of the input node sensitivity, in some applications, could be exploited in the design of variable-precision data acquisition methodologies, where the resource-greedy measurements could be reserved for obtaining the sensitive inputs.

4.1.3 Comparative Analysis of the Computational Overhead for Testing and Model Checking

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 behavioural guarantees than testing. We compared the performance of *nuXmv*-based model checking with *MATLAB* (R2019a) based validation testing. For testing, we define a matrix for all possible noise combinations, for a predefined noise range, 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 same network (trained on leukemia dataset) and both run on the same input samples.

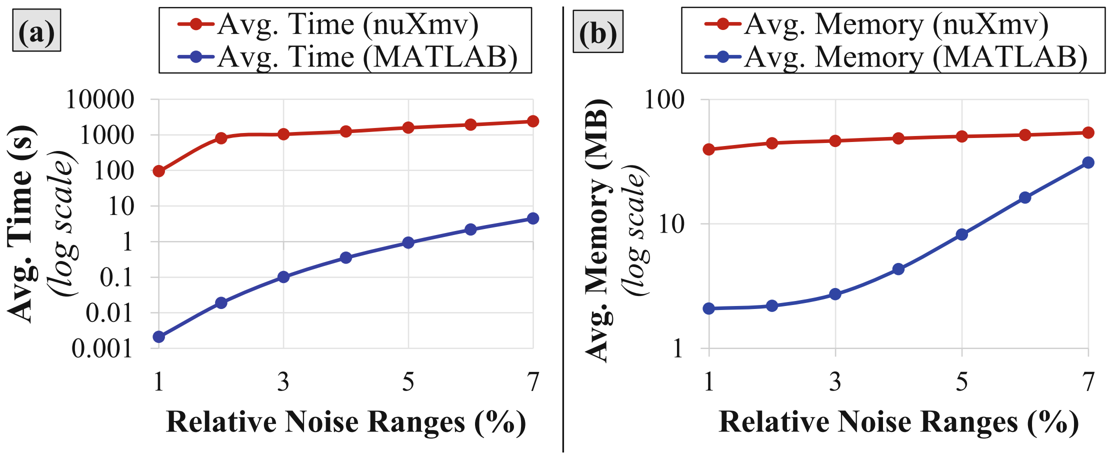


Figure 4.5: A comparison between the (a) Timing and (b) Memory overhead, of MATLAB and nuXmv experiments.

Considering the time taken until the termination of both experiments, the average timing requirement of the nuXmv, although significantly higher than MATLAB for our experiment, increases at a slower rate than that for MATLAB. This trend is illustrated in Fig. 4.5(a). On the other hand, the increase in average memory requirements of *nuXmv* increases at a significantly slower rate than MATLAB, as shown in Fig. 4.5(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 networks. As the size of the network grows, the state-space for model checking grows, resulting in the increased latency and high utilized memory resources. On the other hand, the computational requirements for (exhaustive) testing also increase with the increase in input noise range, often at a higher rate than that for model checking.

4.2 Scaling Model Checking

FANNet was an initial model checking-based framework, relying on explicit state model checking for the analysis of DNN properties. Given the efficient performance of explicit state model checking for software verification problems in literature [EP02, CKS05, BPV17], and the independence of our DNN models on any particular hardware technology/components, the choice of the model checking approach used in FANNet was a logical first-step in the domain of model checking-based DNN analysis. However, as evident from the Kripke structure in Fig. 4.3, even a binary classifier will generate a formal model with $1 + 2|\eta|$ states, where $|\eta|$ depends on the size and precision of the noise bounds used.

This section elaborates on our proposed optimizations for reducing the size of the Kripke structure leveraged by our enhanced framework, FANNet+. 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.

4.2.1 State-Space Reduction

As observed in the Kripke structure in Fig. 4.3, the state-space for the DNN model contains multiple states leading to the same output class. This is because of the enumeration of the different noise combinations from the available noise bounds defining the formal model, leading to identical output states.

In contrast, FANNet+ proposes the use of Symbolic Model Checking (SMC) to reduce the identical output classes. The noise is added to the inputs symbolically, hence reducing the number of states in the model by a factor of approximately n. In terms of Kripke structure, the use of SMC to present noise symbolically is equivalent to merging of the states with the identical valid AP in the DNN model. In other words, if the transition relations $\delta(S_a, S_b)$ and $\delta(S_a, S_c)$ hold, and the labelling function defines identical AP for the states S_b and S_c, i.e., $L(S_b) = L(S_c)$, then the states S_b and S_c can be merged.

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.*

To understand the above state-space reduction, consider again the network given in Fig. 4.3. As shown in Fig. 4.6, the noise is added to the inputs symbolically, i.e., $(x_0 + \Delta x)$

©2023 IEEE. Parts reproduced, with permission, from M. Naseer, O. Hasan, M. Shafique: Scaling Model Checking for Neural Network Analysis via State-Space Reduction and Input Segmentation, *Formal Methods for ML-Enabled Autonomous Systems (FoMLAS 2023)*, Paris, France, 17-18 July, 2023, pp. xxx-xxx. [DOI:doi.org/xxx].

and $(x_1 + \Delta x)$. Hence, all the states with the same AP (shown in Fig. 4.6) can be merged. The resulting model is considerably smaller, containing only $1 + C$ states and $C(C+1)$ transitions, where C is the number of output classes.

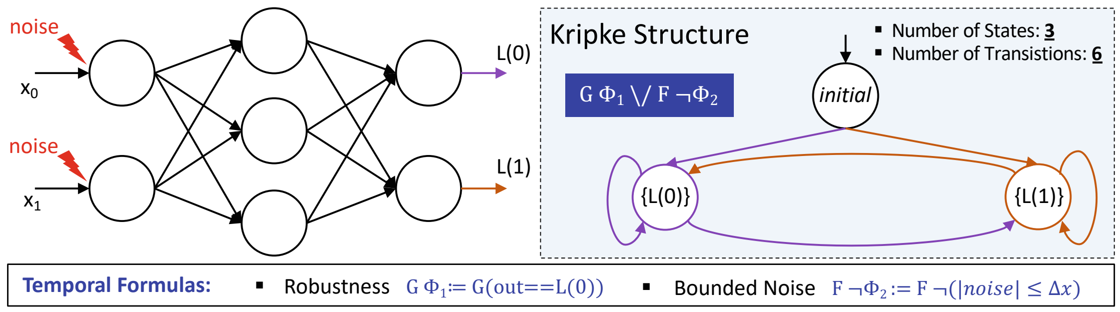
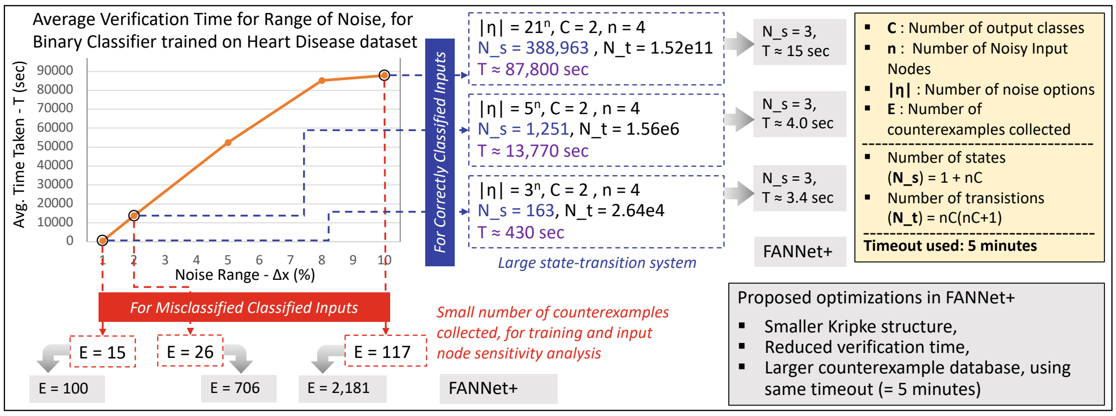


Figure 4.6: The formal model for a small fully-connected network (shown on left), as generated by FANNet+, with noise added symbolically to the inputs.

It must be noted that by using SMC to reduce the state-space of the model, the noise bounds are instead added to the existing formal specifications of the DNN model. This is evident in the example given in Fig. 4.6, where the bounds to the noise are incorporated using $\mathtt{F}\neg\Phi_2$. This is added to the existing property $\mathtt{G}\Phi_1$. Together, the temporal property states that either the output is *globally* correct or the noise *eventually* exceeds the noise bounds. Naturally, this incurs additional complexity for checking the specification (i.e., the stated robustness property), but for a considerably less number of states in the model. Overall, this has a positive impact on both the scalability and the timing efficiency of the approach.

Again from the Kripke structure in Fig. 4.3, it can be observed that the number of states with distinct valid AP depends on the number of output classes of the DNN. However, model checking provides binary answers while checking the specifications, i.e., either the specification holds (UNSAT) or it is violated (SAT). This allows the number of states to be reduced even further by considering the output of the DNN to be either "correctly classified" or "misclassified" instead being one of the C-output classes of the network. The same reduction is also applicable while checking other DNN properties, like safety.

Consider a small binary classifier (network details not essential to the comparison). The classifier is analyzed using FANNet and FANNet+ (i.e., after leveraging the state-space reduction proposed in the section). Fig. 4.7 shows the average verification time taken for varying input noise. As the noise bounds/range increase, the size of the resulting Kripke structure also increases. This accounts for the large timing overhead for the verification. Moreover, for the precise analysis of DNN's robustness 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, it can be observed from the case study in Fig. 4.7 that running FANNet for a small time duration like 5 minutes does not provide a large database of counterexamples for precise robustness bias and input node sensitivity analysis. These factors limit the scalability while increasing the timing overhead of FANNet.



Average Verification Time for Range of Noise, for Binary Classifier trained on Heart Disease dataset
Avg. Time Taken - T (sec)
90000
80000
70000
60000
50000
40000
30000
20000
10000
0
1 2 3 4 5 6 7 8 9 10
Noise Range - Δx (%)
For Correctly Classified Inputs
|η| = 21ⁿ, C = 2, n = 4
N_s = 388,963 , N_t = 1.52e11
T ≈ 87,800 sec
|η| = 5ⁿ, C = 2 , n = 4
N_s = 1,251, N_t = 1.56e6
T ≈ 13,770 sec
|η| = 3ⁿ, C = 2 , n = 4
N_s = 163, N_t = 2.64e4
T ≈ 430 sec
Large state-transition system
N_s = 3, T ≈ 15 sec
N_s = 3, T ≈ 4.0 sec
N_s = 3, T ≈ 3.4 sec
FANNet+
C : Number of output classes
n : Number of Noisy Input Nodes
|η| : Number of noise options
E : Number of counterexamples collected
Number of states (N_s) = 1 + nC
Number of transistions (N_t) = nC(nC+1)
Timeout used: 5 minutes
For Misclassified Classified Inputs
Small number of counterexamples collected, for training and input node sensitivity analysis
E = 15
E = 26
E = 117
E = 100
E = 706
E = 2,181
FANNet+
Proposed optimizations in FANNet+
Smaller Kripke structure,
Reduced verification time,
Larger counterexample database, using same timeout (= 5 minutes)

Figure 4.7: Formal analysis of a binary classifier trained on heart disease dataset using FANNet. The framework delineates limited scalability due to: (a) large size of the network's Kripke structure, (b) large verification time, and (c) small number of counterexamples collected within 5 minutes timeout. The proposed optimizations for FANNet+ improve scalability of DNN analysis by addressing all the aforementioned limitations.

Completeness – The above state-space reduction is enabled by the use SMC, i.e., via a use of symbolic representation of model states with constraints on the (noise) symbols. This preserves the completeness of verification since the formal model implicitly contains the exact details of the actual network. Note that, unlike the state merging resulting from the abstraction or overapproximation of the input domain, the state merging resulting from SMC do not lead to false positives [KKBC12].

The Kripke structure has a completeness threshold[BCC$^+$09] of 2, i.e., absence of counterexample on a path of up to 2 states proves that the specification holds for the network model. This is because we restrict our research to formal models for feedforward neural networks, for which all possible output states are reachable with a path of length 2. Hence, the uses of bounded model checking with bound 2 ensures completeness of the model checking for the above network model.

4.2.2 Input Domain Segmentation

The analysis focused on using FANNet made 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. This again requires a large Kripke structure, likely leading to state-space explosion. This paper proposes two approaches to resolve this problem: *coarse-grain verification* and *random input segmentation*.

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 sampling rate/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.

It must be noted that the use of SMC does not aid in the verification of systems with a large input domain, for highly non-linear systems like DNNs. This is because, even though symbolic representation of input domain reduces the size of the generated Kripke structure, the addition of input bounds to the existing formal specification may make the specification too large to be verified by the model checker.

Random Input Segmentation

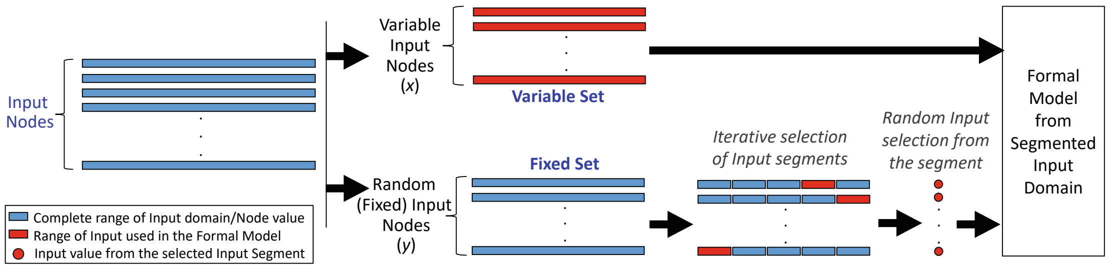

Input Nodes
Variable Input Nodes (x)
Variable Set
Random (Fixed) Input Nodes (y)
Fixed Set
Iterative selection of Input segments
Random Input selection from the segment
Formal Model from Segmented Input Domain
Complete range of Input domain/Node value
Range of Input used in the Formal Model
Input value from the selected Input Segment

Figure 4.8: Overview of Random Input Segmentation: the input nodes are split into *variable* and *fixed* sets, which are then fed to the formal DNN model.

To address the challenge of dealing with a large input domain, we propose Random Input Segmentation (RIS), as shown in Fig. 4.8. 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 *fix* set represented as constants in the model. The details for the approach, also highlighted in Algorithm 2, are as follows:

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 2. 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 2 illustrates the selection of a single input node i for the *variable* set, while remaining nodes $Btemp$ form the *fixed* set. However,

Algorithm 2 Random Input Segmentation

```
Input:      Input Domain Bounds ($I$), Network Parameters ($w$, $b$, $L$, $N$),
            Normalization Parameters ($\mu$, $\varsigma$), Bins per Input Node ($X$), Specification ($\Phi$)
Output:     Counterexample ($CE$)
Initialize: $CE \leftarrow [\,]$

    //Creating bins to split ranges of each input node
 1: for $i = 1$:Size$(I, 2)$ do                                          ▷ For each input node
 2:     for $j = 1$:$X(i)$ do                                            ▷ For each input segment
 3:         $B[1][j][i] \leftarrow (\frac{I(2,i)-I(1,i)}{X(i)} \times (j-1)) + I(1,i)$        ▷ Step 1)
 4:         $B[2][j][i] \leftarrow (\frac{I(2,i)-I(1,i)}{X(i)} \times j) + I(1,i)$            ▷ Step 1)
 5:     end for
 6: end for
    //Segmentation
 7: for $i = 1$:Size$(I, 2)$ do
 8:     $I' \leftarrow I$
 9:     $Btemp \leftarrow B \setminus B\{:, :, i\}$                                  ▷ Step 2)
10:     $k \leftarrow$Size$(I, 2) - 1$
11:     for $j_1 = 1$:Size$(Btemp\{:, :, 1\}, 2)$ do                          ▷ Step 3)
12:         ...
13:         for $j_k = 1$:Size$(Btemp\{:, :, k\}, 2)$ do
14:             $temp[1] \leftarrow rand(Btemp[1, j_1, 1], Btemp[2, j_1, 1])$
15:             ...
16:             $temp[k] \leftarrow rand(Btemp[1, j_k, k], Btemp[2, j_k, k])$
17:             for $m = 1 : k$ do                                       ▷ Step 4)
18:                 if $i \leq m$ then
19:                     $I'[m+1] \leftarrow temp[m]$
20:                 else
21:                     $I'[m] \leftarrow temp[m]$
22:                 end if
23:             end for
24:             $temp \leftarrow [\,]$
25:             $CE \leftarrow$ FANNet+$(I', w, b, L, N, \mu, \varsigma, \Phi)$              ▷ Step 5)
26:             if $CE \neq [\,]$ then return 1                              ▷ Step 6)
27:             end if
28:         end for
29:         ...
30:     end for
31: end for
```

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 2.

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 2.

5. The model checking is then performed with this updated input domain, shown by Line 25 of Algorithm 2.

6. If a counterexample corresponding to a property violation is obtained (as shown in Line 26 of Algorithm 2), 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 2. (Back to step 2)

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 [WOZ+20]. However, by using RIS, each 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 combination of nodes from *fixed* and *variable* sets using a different computation core. This, in turn, reduces the timing-overhead of the analysis.

Soundness – As shown in Fig. 4.8 and Lines $17-23$ of the Algorithm 2, the input domain 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 $\mathcal{X}'$ is the proper subset of the original valid input domain $\mathcal{X}$ of the DNN, i.e., $\mathcal{X}' \subset \mathcal{X}$. This preserves soundness of the framework.

Completeness – Since the updated input domain $\mathcal{X}'$ is the proper subset of domain $\mathcal{X}$, the model checking using the domain entails incompleteness. This means, in case the specifications hold (UNSAT result returned by the model checker) for the input domain $\mathcal{X}'$, there remains the possibility of the specification being violated for the input domain $\mathcal{X} \setminus \mathcal{X}'$. This is the direct result of the Lines $14-16$ of Algorithm 2, 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 from the incompleteness often observed in DNN analysis literature [SGM+18, SGPV19, TPL+21] 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.

4.2.3 FANNet+: An Integrated and Optimized Framework for Formal DNN Analysis

The proposed optimizations reduce the size of network's Kripke structure as well as split the input domain into more manageable sub-domains. This improves the scalability and timing-efficiency of the framework, also allowing the analysis of DNN properties beyond those addressed in FANNet. Fig. 4.9 summarizes our overall proposed framework FANNet+ for formal DNN analysis.

Algorithm 3 Network Formalization and Property Checking

```
Input:      Input ($I$), Network Parameters ($w$, $b$, $L$, $N$),
            Normalization Parameters ($\mu$, $\varsigma$), Specification ($\Phi$)
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($\Phi, I, Out$)                         ▷ Checking DNN specification
16:     return $CE$
17: end function
```

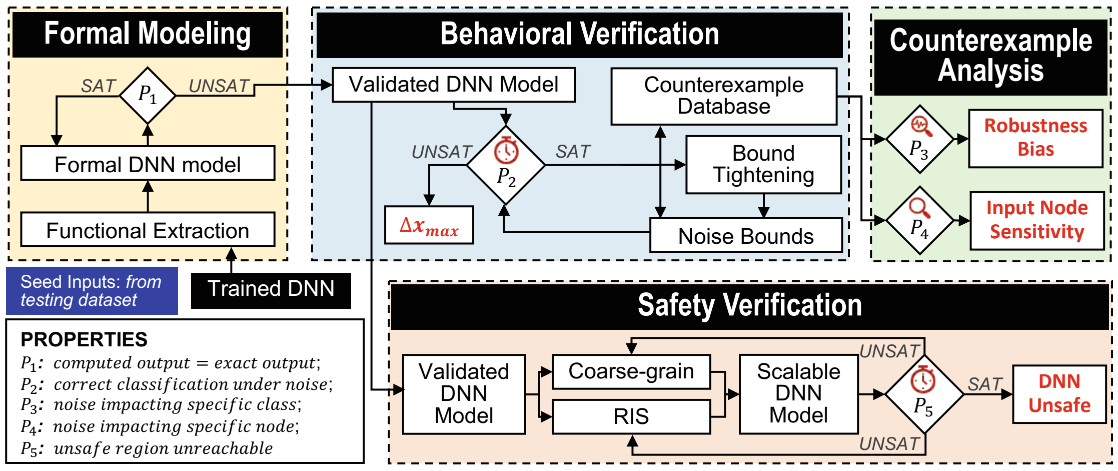


Figure 4.9: FANNet+: scalable SMT-based bounded model checking framework for complete and incomplete DNN analysis.

Initially, the formal model of the DNN is defined in the appropriate syntax of the model checker, as indicated by the yellow box in Fig. 4.9 and Algorithm 3. 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 (as shown in Line 2 of Algorithm 3). Some DNNs may in turn also use inverse-normalization for the DNN output (as shown in Line 14 of Algorithm 3). To validate the functional correctness of the model, the output of the model is checked for the known (testing) inputs (P_1 in Fig. 4.9).

The behavioral verification of the validated formal model is carried out using seed inputs from the testing dataset and noise bounds, as explained in Section 4.2.1. This is expressed in the blue box in Fig. 4.9. Given the noise bounds Δx, the noise applied to seed inputs η is taken to be a percentage of original input node values, i.e., $x * (\eta/100)$ (i.e., relative noise). The DNN specification is updated iteratively with the obtained counterexamples (see P_2 in Fig. 4.9) until a pre-defined timeout is reached. This allows the generation of a large counterexample database. In case the property holds before reaching the timeout, the model checking is immediately terminated. The noise bounds are iteratively reduced until the noise tolerance Δx_{max} of the given network is obtained.

The counterexample database is then used in an empirical counterexample analysis to check the sensitivity of individual input nodes and detect any underlying robustness bias. The is shown by the green box in Fig. 4.9. 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 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. 4.9. Here, either coarse-grain verification or verification with RIS can be opted. As mentioned earlier, the sampling rate (in coarse-grain verification) and the size of input segments (in RIS) is chosen on the basis of the size of original DNN, input domain and the computational resources available to the model checker. The larger the network, input domain or computation requirements, the higher the sampling rate (or smaller the input segment) needs to be.

4.3 Qualitative DNN Analysis: The Experiments

In addition to leukemia diagnosis case study described earlier in Section 4.1.2, we use two additional datasets (and networks) for the qualitative DNN analysis using FANNet+, i.e., heart disease diagnosis and aircraft collision avoidance.

4.3.1 Heart Disease Prognosis

The heart disease dataset [DG17a] provides the records for 13 attributes of the patients: among these, 4 attributes are represented by continuous values while remaining represent discrete attributes. The outputs indicate if the patients' blood vessels are narrowing, hinting to a heart disease. The dataset is split into training and testing datasets with

261 and 42 inputs, respectively. Approximately 55% of the samples from training dataset correspond to the case with narrowing blood vessels, while the remaining represent the samples with no significant narrowing of the blood vessels.

A feed-forward fully-connected network with 3 hidden layers, and a total of 622 network parameters was trained for this dataset. Adaptive learning rate (*Adam optimizer* [KB15]) was used for 1500 epochs of training, to achieve the training and testing accuracies of the networks were 90% and 86%, respectively.

To imitate real-world case scenarios, where the noise is much 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 robustness bias.

4.3.2 Airborne Collision Avoidance System (ACAS Xu)

Airborne Collision Avoidance System (ACAS Xu) DNNs belongs to the family of ACAS X systems that make use of the trajectories of ownship and intruder to ensure safety while maneuvering the ownship. The system consists of 45 DNNs, which correspond to approximately different parts of the input domain. 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. Then the safety properties relevant to the DNNs, which are also well-studied in the literature, 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.

$$(i_1 \geq 55947.691) \ \wedge (i_4 \geq 1145) \wedge (i_5 \leq 60) \implies (o_1 \leq 1500)$$

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.

$$\begin{aligned}&(i_1 \geq 55947.691) \wedge (i_4 \geq 1145) \wedge (i_5 \leq 60)\\ &\qquad \implies (o_2 > o_1) \vee \ (o_3 > o_1) \vee (o_4 > o_1) \vee \ (o_5 > o_1)\end{aligned}$$

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.

$$\begin{aligned}&(1500 \leq i_1 \leq 1800) \wedge (-0.06 \leq i_2 \leq 0.06) \wedge (i_3 \geq 3.10) \wedge i_4 \geq 980)\\ &\qquad \wedge (i_5 \geq 960) \implies (o_2 < o_1) \vee (o_3 < o_1) \vee (o_4 < o_1) \vee (o_5 < o_1)\end{aligned}$$

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.

$$(1500 \leq i_1 \leq 1800) \wedge (-0.06 \leq i_2 \leq 0.06) \wedge (i_3 = 0) \wedge \ (i_4 \geq 1000) \\ \wedge (700 \leq i_5 \leq 800) \implies (o_2 < o_1) \vee (o_3 < o_1) \vee (o_4 < o_1) \vee (o_5 < o_1)$$

As indicated in Section 4.2.2, the analysis of safety properties involves a formal model with large input domains. Hence, coarse-grain verification and random input segmentation are used for the analysis of ACAS Xu DNNs.

4.3.3 Results and Analysis

We use the newtorks highlighted in the previous subsections 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 [Bea16] back end. The tools Reluplex [KBD+17] and Marabou [KHI+19] are implemented on virtualbox running Ubuntu 18.04, for comparison. The timeout used for networks trained on leukemia and heart disease datasets is 5 minutes, while a timeout of 2 hours is used for ACAS Xu DNNs.

In addition to the formal behavioral analysis using FANNet+, the timing performance of FANNet and FANNet+ are also compared to indicate the performance improvements. The safety verification results for ACAS Xu DNNs are then presented, while comparing these results to those obtained from popular DNN verification tools Reluplex and Marabou.

Behavioral (Robustness) Verification and Noise Tolerance Determination

As explained in Section 4.1, the Kripke structure model for networks 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 stat-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. Tables 4.1 and 4.2 summarise the the robustness of the seed inputs from the leukemia and heart disease datasets, respectively, under the incidence of increasing noise bounds.

FANNet versus FANNet+. We run both frameworks for the networks trained on the leukemia and the heart disease datasets. Figs. 4.10 shows the average execution time for verifying networks for both datasets, for seed inputs robust to the applied noise. Identical noise bounds for seed inputs were used for both frameworks. Both frameworks use iterative noise reduction, leading to noise tolerance for the networks.

As observed in Fig. 4.11(a) and Fig.4.13 for network trained on leukemia dataset, both frameworks lead to the same noise tolerance i.e., 11% (also observe Table 4.1). The same

Table 4.1: Robustness of network, with seed inputs from leukemia dataset under varying noise bounds: UNSAT (✓), SAT(✗). The noise tolerance of the network is 11%.

In-put	Result with Noise: $\leq 11\%$	20%	30%	40%	In-put	Result with Noise: $\leq 11\%$	20%	30%	40%
1	✓	✓	✓	✓	17	✓	✓	✓	✓
2	✓	✓	✓	✓	18	✓	✓	✓	✓
3	✓	✓	✓	✓	19	✓	✓	✓	✓
4	✓	✓	✓	✓	20	✓	✗	✗	✗
5	✓	✓	✓	✓	21	✓	✓	✓	✓
6	✓	✓	✓	✓	22	✓	✓	✓	✓
7	✓	✓	✓	✓	23	✓	✓	✓	✓
8	✓	✓	✓	✓	24	✓	✓	✓	✗
9	✓	✓	✓	✓	25	✓	✓	✗	✗
10	✓	✓	✓	✓	26	✓	✓	✓	✓
11	✓	✓	✓	✓	27	✓	✗	✗	✗
12	✓	✓	✓	✓	28	✓	✓	✓	✓
13	✓	✓	✓	✓	29	✓	✓	✗	✗
14	✓	✓	✓	✓	30	✓	✗	✗	✗
15	✓	✓	✓	✗	31	✓	✓	✓	✓
16	✓	✓	✓	✓	32	✓	✓	✓	✗

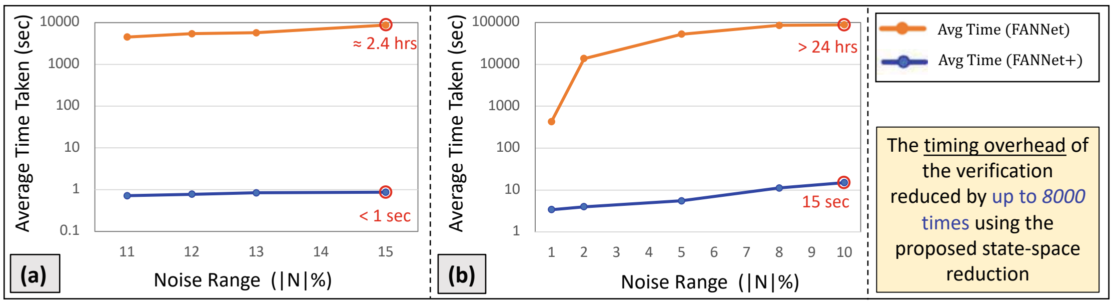


Figure 4.10: Comparison of Timing Overhead using FANNet and FANNet+: (a) plots the results obtained from the network trained on the leukemia dataset, while (b) plots results from the network trained on the heart disease dataset.

is observed with network trained on the heart disease dataset, having a noise tolerance of $< 1\%$. However, the execution time for property verification is significantly larger for FANNet, as shown in Fig. 4.10. For the given networks, 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.

Table 4.2: Robustness of network, with seed inputs from heart disease dataset under varying noise bounds: UNSAT (✓), SAT(✗).

Input	**Result with Noise:** 2%	5%	8%	10%	**Input**	**Result with Noise:** 2%	5%	8%	10%
1	✓	✓	✓	✓	19	✓	✗	✗	✗
2	✓	✓	✓	✓	20	✓	✓	✓	✓
3	✓	✓	✓	✓	21	✓	✓	✓	✓
4	✓	✓	✗	✗	22	✗	✗	✗	✗
5	✓	✓	✓	✓	23	✗	✗	✗	✗
6	✓	✓	✓	✓	24	✓	✓	✓	✓
7	✓	✓	✓	✓	25	✓	✓	✓	✓
8	✓	✓	✓	✓	26	✓	✓	✓	✓
9	✓	✗	✗	✗	27	✗	✗	✗	✗
10	✓	✗	✗	✗	28	✓	✓	✓	✓
11	✓	✓	✓	✓	29	✓	✓	✓	✗
12	✓	✓	✓	✓	30	✓	✓	✓	✓
13	✓	✓	✓	✗	31	✓	✓	✓	✓
14	✓	✓	✓	✓	32	✓	✓	✓	✓
15	✗	✗	✗	✗	33	✓	✓	✓	✓
16	✓	✓	✓	✓	34	✓	✓	✓	✓
17	✓	✓	✗	✗	35	✗	✗	✗	✗
18	✓	✓	✓	✗	36	✓	✓	✓	✓

Counterexample Analysis for detecting Network's Robustness 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 network outputs for these counterexamples provide insights regarding robustness bias and input node sensitivity, as shown in Figs. 4.11 and 4.12.

In Fig. 4.11(a), Label 1 corresponds to ALL leukemia while Label 0 corresponds to AML leukemia. On the other hand, Label 1 and 0 correspond to the cases with and without blood vessels narrowing, respectively, in Fig. 4.11(b). As observed in the figures, for the network trained on leukemia dataset, even when the large noise is applied to inputs, ALL inputs are rarely misclassified to AML. As discussed earlier, the leukemia dataset has long-tail distribution, with approximately 70% inputs belonging to ALL. Hence, the obtained results indicate a strong bias in the resulting trained network likely due to the imbalance in dataset.

On the contrary, for the network 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. It contains inputs from the two input classes Labels 1 and 0 with a ratio of 55 : 45. This likely accounts for the relatively moderate bias observed in the network.

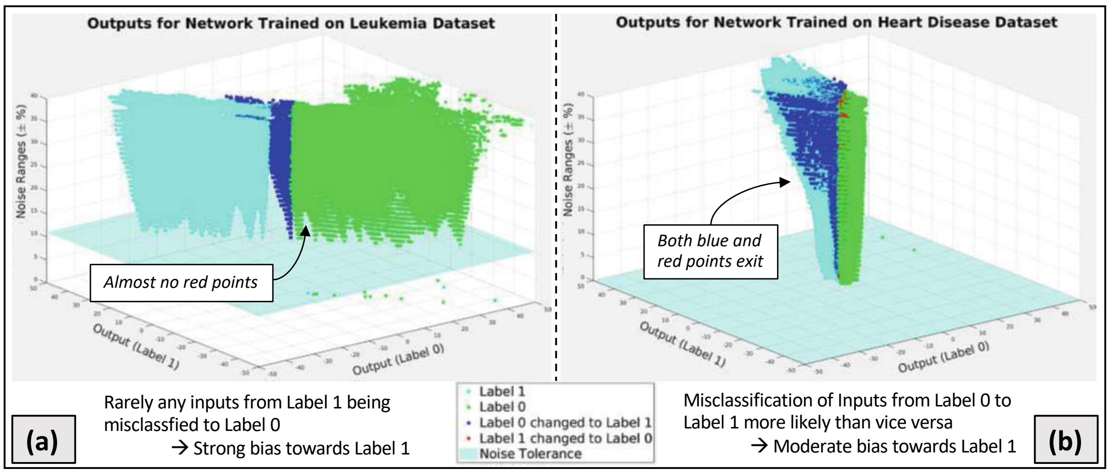


Figure 4.11: Output classification of networks trained on (a) Leukemia dataset and (b) Heart disease dataset, mapped with respect to the noise applied to seed inputs, using FANNet+. An unequal number of red and blue points indicates a bias in trained networks.

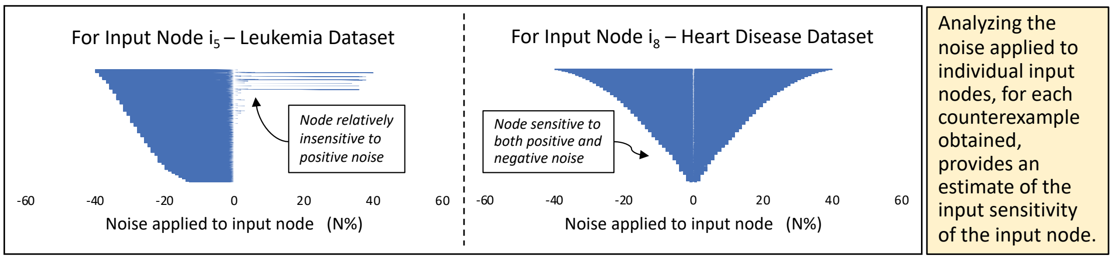


Figure 4.12: Plots of noise applied to individual input nodes (using FANNet+), that lead to misclassification.

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 network trained on the leukemia dataset, shown in Fig. 4.12, 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 network. However, this was not the case for any of the input nodes for the network trained on heart disease dataset. For instance, node i_8 in Fig. 4.12 gives an example of a typical input node that is not sensitive to any specific input noise.

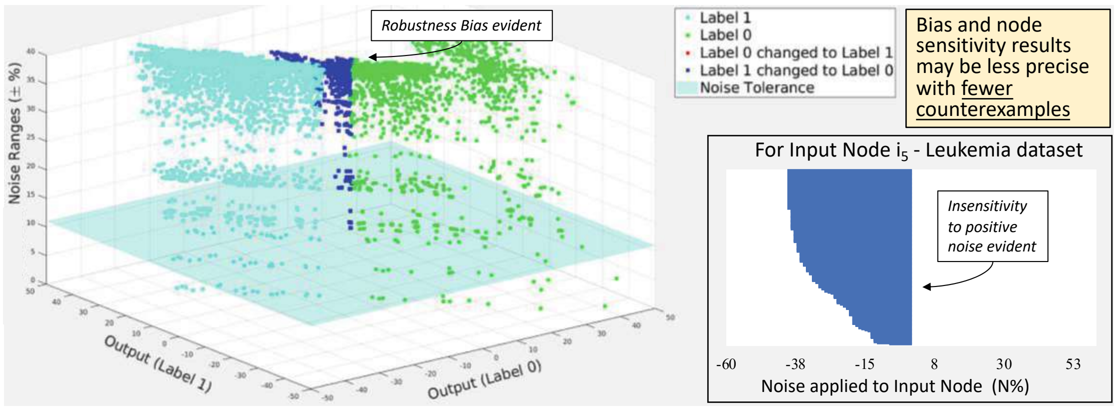


Figure 4.13: Robustness bias and input node sensitivity of the network trained on leukemia dataset, analyzed using the counterexample database obtained using FANNet.

FANNet versus FANNet+. We performed the same experiment for network trained on leukemia using FANNet, as shown in Fig. 4.13. As discussed earlier, due to the large timing overhead of FANNet, the counterexamples obtained compose a smaller counterexample database. This is evident in the sparse 3D plot in Fig. 4.13. Hence, even though both FANNet and FANNet+ indicate the network 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. 4.13 indicate node i_5 to be completely insensitive to any positive noise. However, FANNet+ (as shown in Fig. 4.12(left)) predicts the node to be relatively insensitive to positive noise, but may still lead to input misclassification with certain noise patterns. Hence, the larger counterexample database with FANNet+ is able to provide more precise results for robustness bias and input node sensitivity, as compared to FANNet.

Table 4.3: Input step sizes used for coarse-grain verification of ACAS Xu safety properties.

Input	**Step size for sampling input for each safety property**			
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

Safety Verification

For DNN properties, like safety, which involve large input domain, coarse-grain verification with input step sizes for ACAS Xu DNNs as shown in Table 4.3, is first used. The verification for each property, for each DNN, takes only a few seconds to complete.

Table 4.4: Number of segments of input domain corresponding to every input node, for each property under analysis.

Input Node	P1	P2	P3	P4
i_1	3	3	2	2
i_2	4	4	3	3
i_3	4	4	2	1
i_4	2	2	4	4
i_5	2	2	4	4

As mentioned earlier in Sec. 4.2.2, coarse-grain verification is suitable for DNNs and properties where large segments of input domain violate the property. This does not hold true for ACAS Xu DNNs. Hence, no counterexamples to property violation were found.

Next, random input segmentation is deployed for the safety verification of ACAS Xu properties. The number of segments for each input node for the safety properties is given in Table 4.4. 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 [SGM+18, XTJ18, SGPV19, TPL+21], 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. These are summarized in Fig. 4.14, where the success of FANNet+ in identifying unsafe networks is comparable to those of the state of the art. 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`.

FANNet versus FANNet+. As opposed to FANNet+, which leverages coarse-grain verification and random input segmentation to split input domain prior to verification, FANNet relies on bounds of entire input domain. Hence, the verification of safety

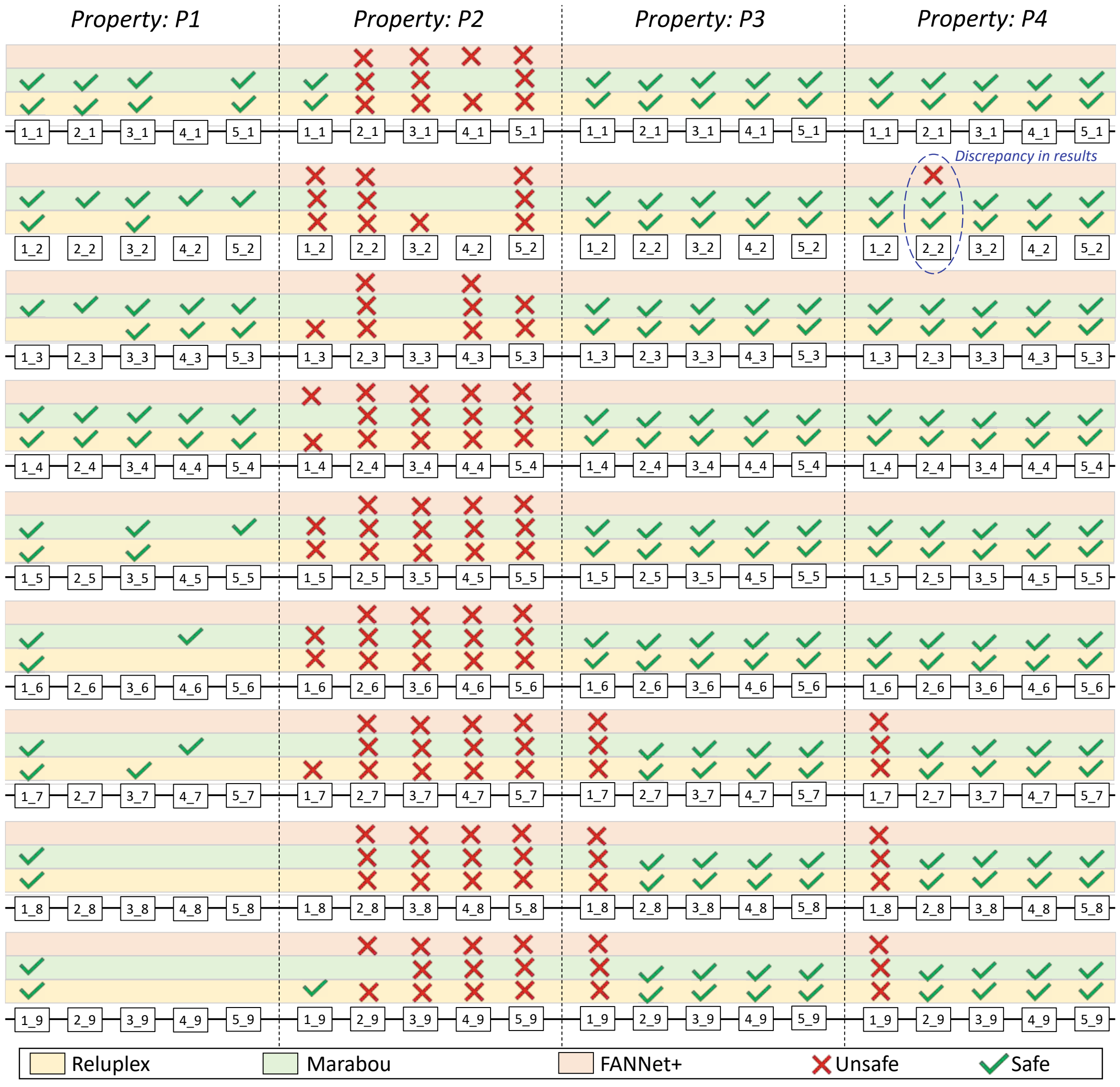


Figure 4.14: Safety verification using Reluplex, Marabou and FANNet+: the frameworks either identify whether the bounded input domain is safe or unsafe for desired network safety properties, or remain inconclusive. In case of discrepancy in result, FANNet+ is found to provide correct result as opposed to the the other frameworks.

properties of ACAS Xu DNNs was infeasible with FANNet, even with a timeout of 24 hours.

4.4 Summary of Qualitative DNN Analysis

Formally analyzing DNN is an actively sought research domain. This chapter presented FANNet and FANNet+, a model checking-based frameworks to verify the robustness of trained DNNs under constrained noise, and also analyze DNNs' noise tolerance, input node sensitivity, underlying robustness bias, and safety specifications. The aim of this work was to explore not only novel formal analysis methodologies for trained DNNs but also challenge the limits of model checking for DNN analysis. The scalability challenge was tackled via state-space reduction and input segmentation. Additionally, a diverse range of DNN properties were qualitatively analyzed. However, as indicated in earlier chapters, qualitative analysis provide only binary results rather than the extent of DNNs following the desired behavioral properties. The next chapter deals with the challenge of DNN analysis quantitatively to answer the aforementioned limitation.

CHAPTER 5

Quantitative Neural Network Analysis

As mentioned in the previous chapters, the qualitative analysis provides merely a binary result, i.e., either the specification holds for the formal model or it does not. From an extensive focus of the existing literature (discussed in Chap. 2) on the identification of DNN vulnerabilities, it can be observed that the desired specifications rarely hold for the DNNs. Hence, to address the aforementioned problem, this chapter focuses on the quantitative analysis of trained DNNs to identify the *extent of DNN vulnerability*. This is also summarized in Fig. 5.1. In particular, the novel contributions of this research, in regards to quantitative analysis, are as follows:

1. Extend the *qualitative* model checking-based DNN analysis framework (explained in Chap. 4) to provide *quantitative* DNN behavioral guarantees.
2. Use realistic and practical noise mode noise model, i.e., the relative noise model (described in Sec. 3.3.2) for DNN analysis.
3. Provide concrete evidence of the limitations of the popular L^p norm-based noise models for DNN analysis.
4. Focus on a diverse range of properties namely, input node sensitivity, noise tolerance, reachability, robustness, robustness bias and safety.
5. Develop a statistical framework, QuanDA, to provide quantitative DNN analysis along with a confidence interval of the estimated results.
6. Propose an intricate sampling-based approach to improve scalability and ensure good coverage of the analysis.

7. Propagate realistic probability estimates through DNN layers, while avoiding any unrealistic assumptions on the node probability distributions.

8. Introduce GPU implementation of the framework to reduce timing overhead for DNN quantitative analysis.

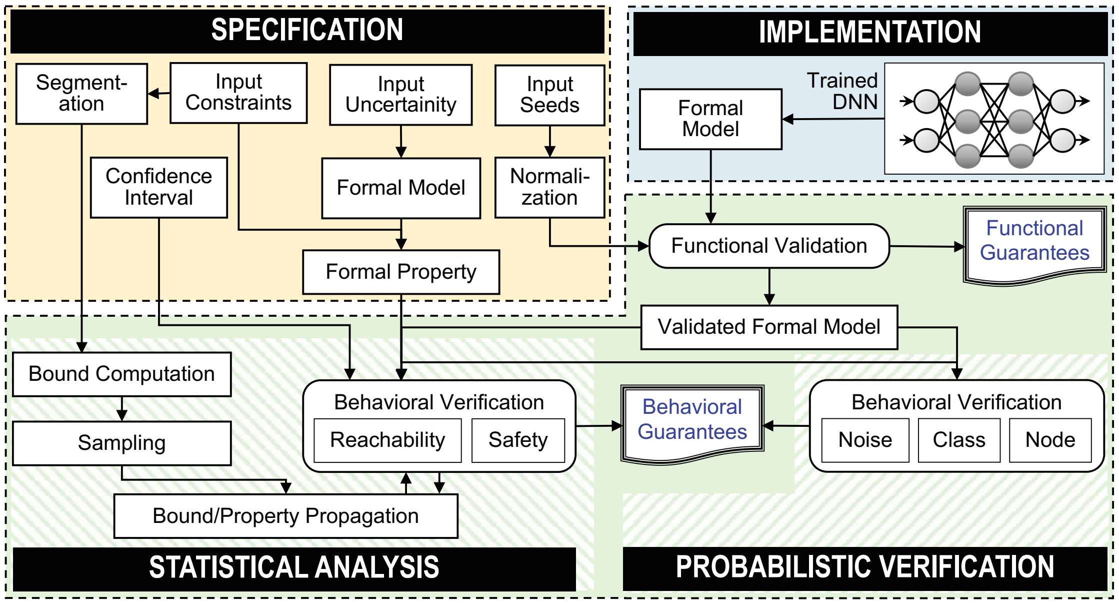


Figure 5.1: The overall quantitative analysis methodology presented in this chapter.

5.1 Probabilistic Analysis

Noise is a ubiquitous component of the physical environment. Whether it be due to atmospheric conditions like fog and pollution, or perturbation at input sensors during data acquisition, it is unlikely to have a system deployed in the real-world that is completely immune to noise [StT+20]. Even though the magnitude of the noise is often considerably small compared to the magnitude of the input, as shown by the orange and blue bars, respectively, in Fig. 5.2 for a DNN trained on MNIST dataset, it is capable of making the DNNs delineate unexpected behavior.

©2022 ACM. Parts reproduced, with permission, from I. T. Bhatti, M. Naseer, M. Shafique, O. Hasan: A formal approach to identifying the impact of noise on neural networks, *Communications of the ACM (CACM 2022)*, pp. 70-73. [DOI:`doi.org/10.1145/3550492`].

©2023 Springer International Publishing. Parts reproduced, with permission, from M. Naseer, I. T. Bhatti, O. Hasan, M. Shafique: Considering the Impact of Noise on Machine Learning Accuracy, *Embedded Machine Learning for Cyber-Physical, IoT, and Edge Computing: Hardware Architectures*, pp. xxx-xxx. [DOI:`doi.org/xxxx`].

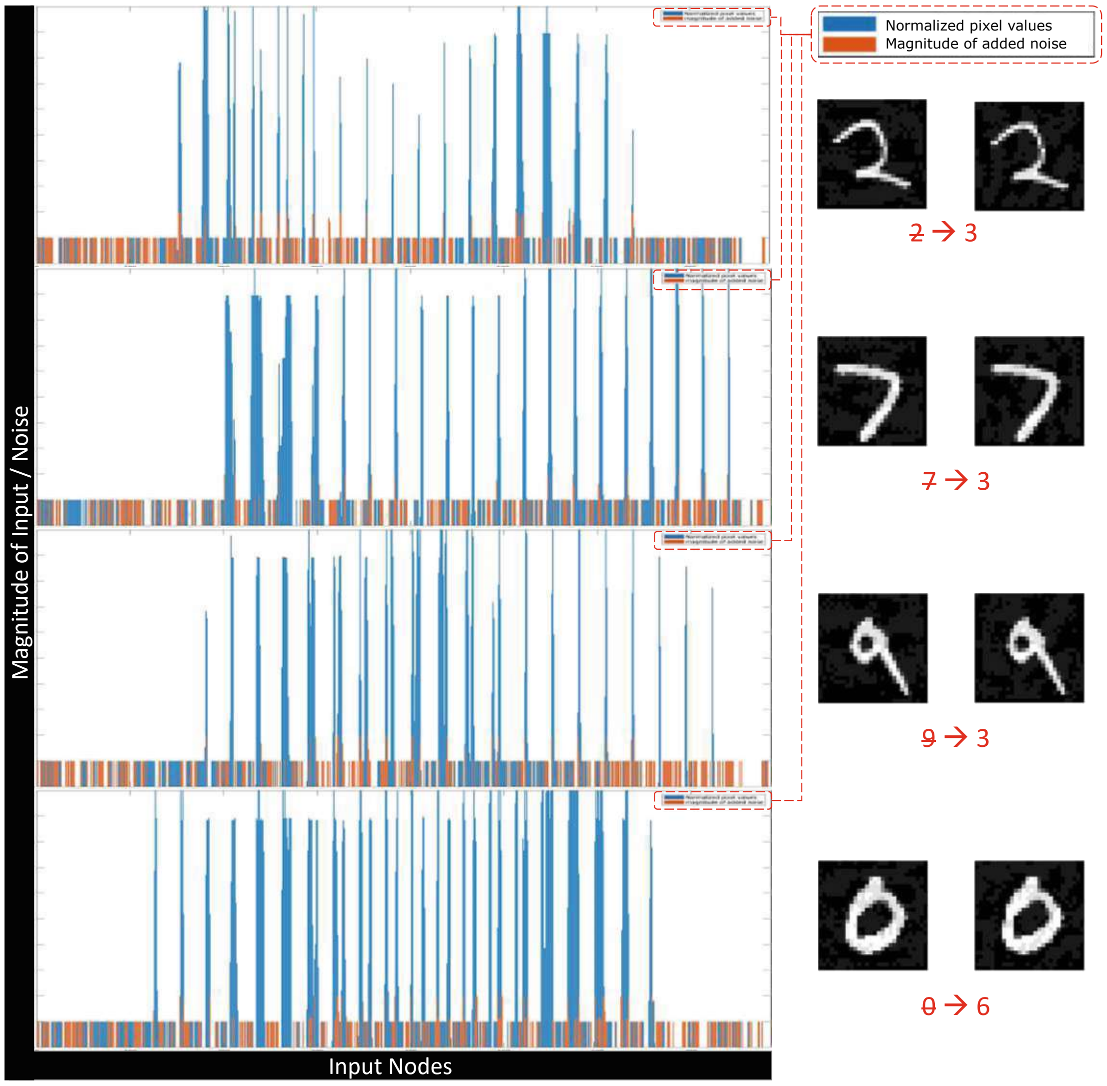


Figure 5.2: The magnitude of noise (shown in orange) is often small in comparison to the input magnitude (shown in blue). Hence, the resulting change in input is too minute to be perceptible, while still making the DNN misclassify the (noisy) input.

Given the completeness of the formal methods [HKWW17, tMK+20, WOZ+20], they are of particular interest to provide reliable guarantees for the behavior of DNNs under such noisy conditions [SZS+13, MMS+22, WOZ+20]. To achieve this, we already proposed a *qualitative* DNN analysis framework, as elaborated in the previous chapter. In this section, we adapt our qualitative analysis framework for the probabilistic analysis of trained DNNs. This is achieved by swapping the model checker used backend by the framework to a probabilistic model checker. Additionally, we provide a detailed comparison of the analysis using the various noise models provided in Chap. 3.3, establishing the practicality and usefulness of the relative noise modeling.

5.1.1 Experimental Setup

To demonstrate the impact of noise on a real-world dataset quantitatively, we trained a single hidden layer, ReLU-based fully-connected binary classifier on leukemia dataset [GST+99] to the training and testing accuracy of 100% and 94.12%, respectively. The Storm model checker [DJKV17] was used backend for the analysis, on AMDRyzen Threadripper 2990WX processors running Ubuntu 18.04 LTS operating system. For the experiments using the relative noise model, the precision of noise was chosen to be 1%, while for the experiments using the L^p norm model, the precision was 0.01.

5.1.2 Results and Analysis

From prior qualitative analysis (Chap. 4), it was observed qualitatively that the increase in noise reduces the classification accuracy of the DNN, while aggravating the bias. This is summarized in Fig. 5.3. Note that these results correspond to the use of relative noise model for analysis. The following provides quantitative results for the network using

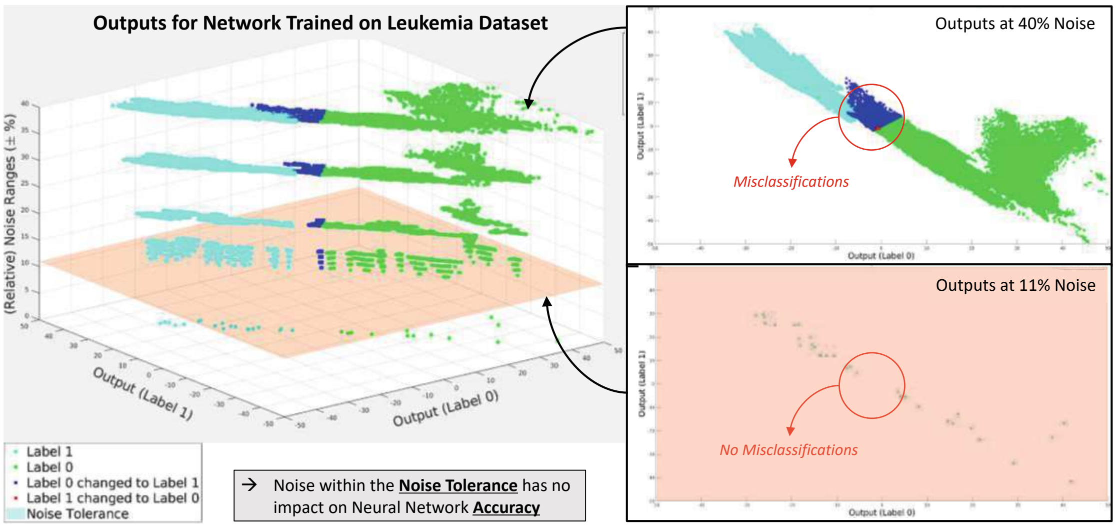


Figure 5.3: Impact of increasing (relative) noise on the output classification of the trained network, as observed using nuXmv-based FANNet implementation.

relative noise as well as L^P norm-based noise models, for individual DNN vulnerabilities.

Robustness and Tolerance. As expected, the probability of correct classification reduces with the increase in the magnitude of noise, as shown in Fig. 5.4. For all noise less than the noise tolerance of the network (also shown in Fig. 5.3), the DNN provides correct output classification with a probability of 1.0, even in the presence of noise in the input. For the given network, this tolerance is found to be 11% in the case of the relative noise. For the network under L^p norm-based noise, the robustness was significantly low, with the noise tolerance less than the precision of the analysis.

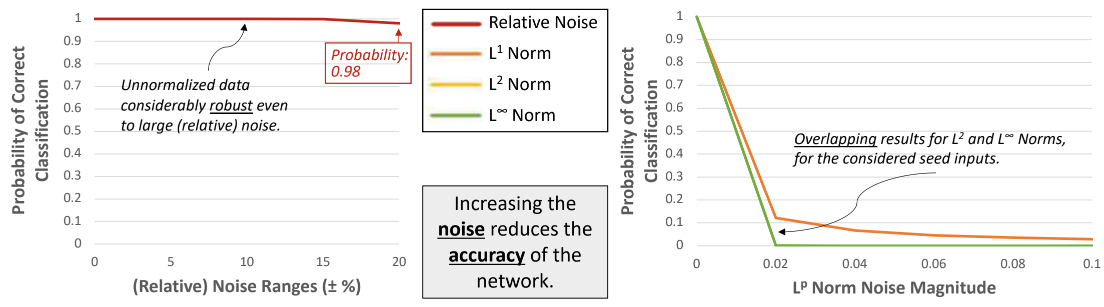


Figure 5.4: Increasing noise versus the probability of correct output classification: the decreasing robustness of DNN beyond noise tolerance is observable in the case of the relative noise model (i.e., the graph on the left).

Nevertheless, the decreasing robustness of trained DNN model under the impact of increasing noise is evident for all noise models, as shown in Fig. 5.4. Noise tolerance, which is an essential (constant) attribute of a trained DNN that can aid during system design by providing an acceptable noise level to ensure the robustness of the DNN, is also found to be extremely low for the DNNs with L^P norm-based models.

Robustness Bias. Robustness bias is a stealthy vulnerability of the DNN, which may aggravate under noisy input and may lead DNN to provide incorrect response in real-world applications. As indicated earlier, the DNN is trained on a dataset with a significantly larger proportion of inputs from patients having ALL (henceforth referred to as Label 1), as compared to those having AML (henceforth referred to as Label 0). This is likely to result in a biased DNN, as observed with the relative noise model (Fig. 5.5 - left). For inputs classified correctly in the absence of noise, i.e., inputs having a correct classification probability of 1.0, the input noise has a more adverse impact on the inputs belonging to Label 0, as compared to vice versa. Observing the qualitative analysis from Fig. 5.3 supports the same conclusion. However, the bias is not observable under L^p norm-based noise model Fig. 5.5 - right), likely due to the low robustness of the DNN under that model.

We believe that, owing to the larger proportion of inputs from Label 1 in the training dataset, the decision boundary learned by the DNN better encapsulates the inputs from Label 1. The inputs from Label 0, on other hand, are presumably closer to the decision boundary, and hence more likely to be misclassified under the application of noise.

Input Node Sensitivity. As discussed in the previous chapter, different input nodes of a trained DNN may have a different sensitivity to the applied noise. Again, this impact of noise is observable only with the relative noise model, for the DNN trained on the

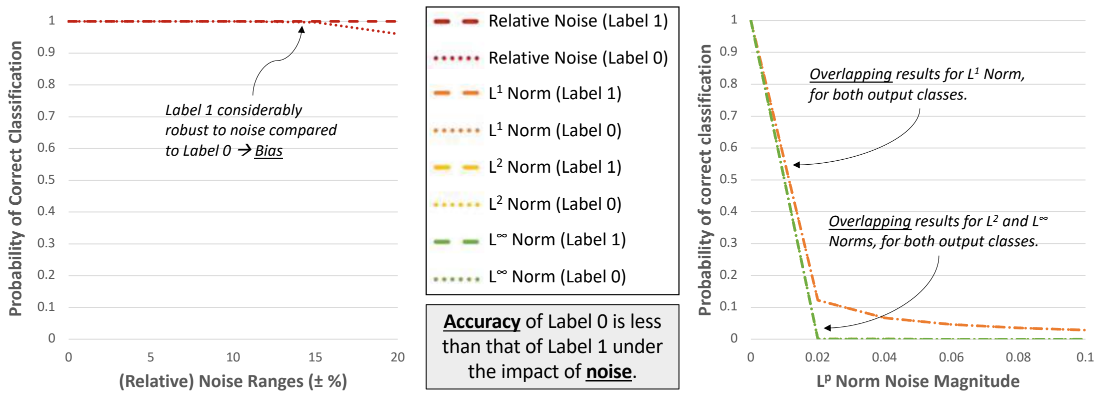


Figure 5.5: The bias is visible through the analysis under relative noise model, where probability of correct classification reduces only for single output class. However, the impact is not observable with L^p norm noise model.

leukemia dataset, as shown in Fig. 5.6. It can also be observed that certain input nodes may be more sensitive to either positive (for instance, node x_3) or negative (for instance, node x_5) noise.

5.1.3 Discussion

As elaborated in Chap. 3.3, unlike the relative noise, the L^p norm noise is added to the normalized inputs, i.e., the inputs in the range of 0 to 1. For the analyzed network, the raw, unnormalized input values range on the scale of hundreds to thousands. Assuming an input node value to be $10,000$, the addition of 0.01 units of noise to the input implies the addition of a noise of magnitude 100. Such a large noise may or may not be very realistic for the noise analysis for a DNN to be deployed in a practical setting. This could be a possible reason for the inadequacy of the aforementioned noise model for analyzing the impacts of noise beyond robustness, for the given DNN.

At the same time, it is possible to have another node with an input value of 100. Here, the application of the same noise (i.e., 0.01) implies a change of only a unit difference in the magnitude of the input node. This is a very likely change in the input of DNN deployed in real-world. Hence, the noise 0.01 may result in realistic noise for some input nodes, while unrealistic for others, making the noise model inept for DNNs with inputs having different input ranges.

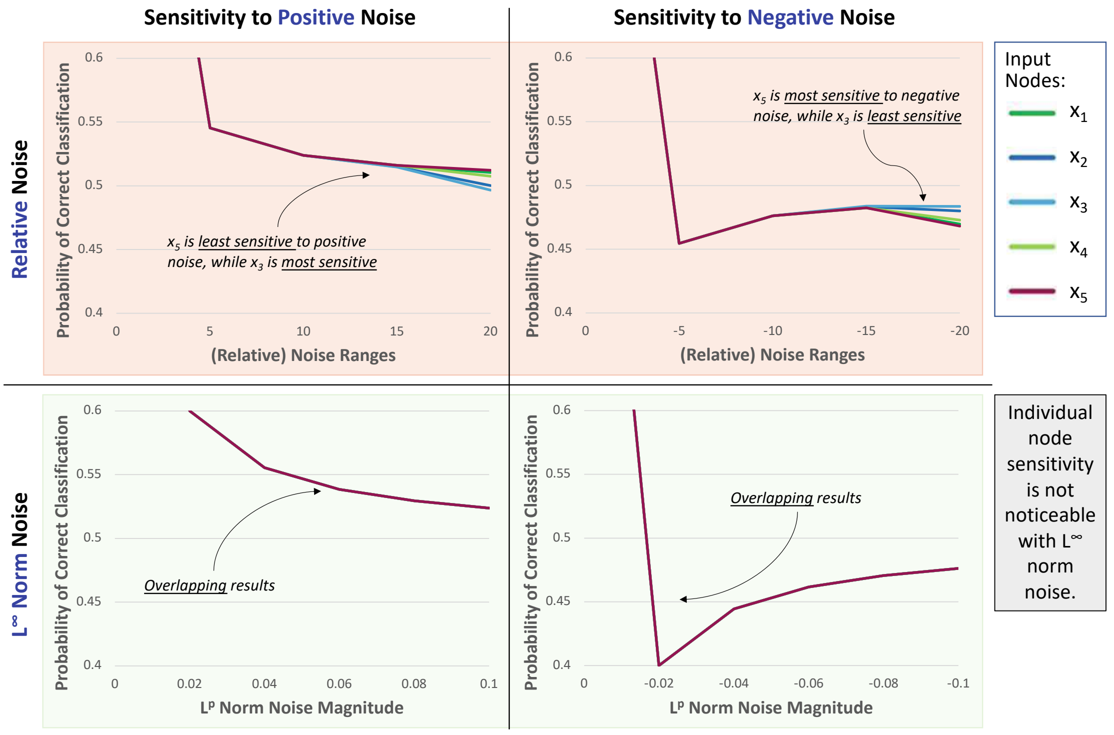


Figure 5.6: The sensitivity of individual input nodes, to positive and negative noise, as observed under the relative and L^∞ norm-based noise models.

5.2 Statistical Analysis

To address the challenge of *quantitative* DNN analysis, recent works [CFGP20, YLL+21b] attempt to focus on the estimated probability of properties like the robustness of the trained network in the presence of small noise around seed inputs. The notion of probability in these works is considered as the ratio between the volume of reachable outputs to the volume of the entire valid output domain. However, such a notion of probability implicitly assumes that all regions of the output domain are equally likely, i.e., the network nodes entail uniform distribution for their output values. As will be shown with our detailed analysis and discussion in later subsections, this is a simplistic notion of probability, which does not hold for practical DNNs.

In contrast, we propose a novel quantitative DNN analysis framework QuanDA, which does not make any assumptions on the probability distribution of the output at DNN nodes. However, this is not a straightforward task: even if the probability distribution is assumed to be uniform at the input nodes, the same (i.e., uniform probability distribution)

©2023 ACM. Parts reproduced, with permission, from M. Naseer, O. Hasan, M. Shafique: QuanDA: GPU Accelerated Quantitative Deep Neural Network Analysis, *Transactions on Design Automation of Electronic Systems (TODAES 2023)*, pp. xxx-xxx. [DOI:`doi.org/xxxx`].

does not hold for the hidden network nodes due to the computations involved in the DNN. To the best of our knowledge, this is the first work estimating realistic probability distribution for all DNN nodes, without any a priori assumptions regarding the probability distribution of the hidden nodes. Apart from the computational challenge of probability distribution estimation, the propagation of estimated distribution along DNN layers poses the additional challenge of high timing overhead. This is tackled by QuanDA via group formation and parallelization leveraging efficient use of both CPU and GPU capabilities. This ultimately provides the precise probability of correct, safe, or robust DNN outputs under bounded input and/or noise. In summary, the contributions of this work (also provided in Fig. 5.7) are as follows:

1. Providing a novel framework, QuanDA, for quantitatively analyzing DNN properties.
2. Computing exact reachability bounds for all DNN nodes, at all network layers, hence identifying the subset of real domain involved in network computations and making the DNN analysis tractable.
3. Adapting stratified sampling and weighted sum models to ensure better precision and coverage of input domain, for realistic probability estimates.
4. Using statistical methods for realistically propagating and estimating output probabilities of node values at each network layer, without using any assumptions about the probability distributions at the hidden nodes.
5. Leveraging Hoeffding's inequality to provide the deviation from exact probabilities and precise confidence level for the probability estimates obtained at each DNN layer.
6. Utilizing QuanDA to check the reachability property of the nodes in the benchmark ACAS Xu networks, and also demonstrating the potential application of the framework for the analysis of the safety properties of these networks.
7. Demonstrating, using the benchmark networks, that the outputs of hidden nodes in the DNNs do not follow any univariate probability distribution, including uniform distribution.

5.2.1 QuanDA: Quantitative Deep Neural Network Analysis

In general, quantitative analysis is a branch of mathematics leveraging statistical methods to collect, evaluate and analyze numeric data. For DNNs, this could provide insight into the behavior of trained networks under varying inputs, as well as their precise correctness, safety and robustness estimates. This section systematically explains our quantitative DNN analysis framework, QuanDA, also shown in Fig. 5.8. The framework accepts a trained DNN and input/noise bounds (dictated by the desired network properties under study) as inputs. These are used for exact bound computations for all DNN nodes across

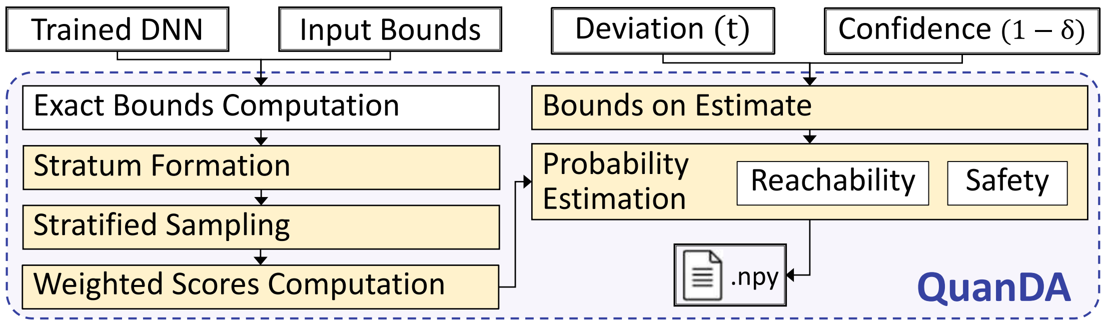


Figure 5.7: Overview of QuanDA, along with novel contributions of the work (highlighted in yellow).

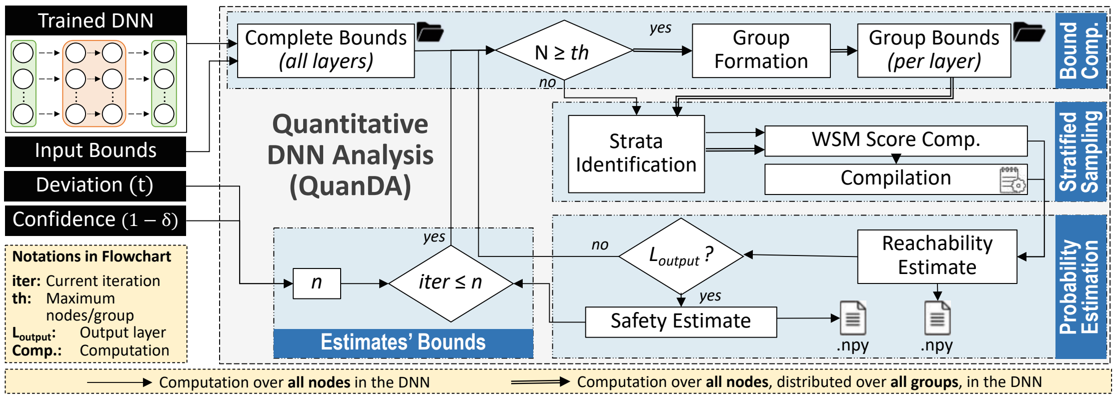


Figure 5.8: Detailed overview of QuanDA: the probability estimates (namely reachability and safety estimates) for the trained network are determined quantitatively.

all network layers. The bounds are divided into non-overlapping strata for stratified sampling. Efficient group formation of nodes, for DNN layers with a large number of nodes, is also deployed to reduce the timing overhead of the analysis.

This is followed by weighted score computation using Weighted Sum Models (WSMs) for each DNN node. This aids realistic probability estimation using the framework. The probability estimates are propagated sequentially over all network layers. The efficient design of QuanDA enables the parallelization of computations in each layer of the DNN, further improving the timing efficiency. The current version of the framework focuses on reachability and safety properties for the trained DNNs.

Moreover, the user-defined confidence interval and the maximum deviation between exact and estimated probability values provided at the input ensure precise probability estimates. The final result from the framework is probability estimates for the reachability property at all network nodes, and the safety properties at the network's output layer. These are available to users as `.npy` files at the output.

The details of the methodology comprising QuanDA is provided as follows.

Bound Computation

The input bounds refer to the extrema of the valid input domain of the given DNN. Fundamentally, these are dictated either by the span of data available to the DNN at its input layer or the subset of the input domain of interest for the desired DNN property. For instance, the valid input bounds for a pixel in a grayscale image is $[0, 255]$ and the subset of the distance of interest between two aircraft could be $[0, 1000]$ft. These are used to compute the output bounds for the layer, which form the input for the subsequent network layer. Without the determination of *precise* node bounds at each DNN layer, the node values could lie anywhere in the real domain, making the DNN analysis intractable. QuanDA leverages the laws of interval arithmetic [HJVE01] to propagate node bounds, starting at the input nodes, to obtain the exact bounds of the output for each node of the trained DNN. This is a useful step in the DNN analysis since it provides the boundaries within which all node values of interest lie.

Consider DNN nodes x_1,x_2 such that $[\underline{x_1}, \overline{x_1}] = \{x_1 \in \mathbb{R} | \underline{x_1} \leq x_1 \leq \overline{x_1}\}$ and $[\underline{x_2}, \overline{x_2}] = \{x_2 \in \mathbb{R} | \underline{x_2} \leq x_2 \leq \overline{x_2}\}$. The affine transformation (discussed earlier in Eq. 3.1) involves the multiplication of the node values with scalar (weight w). Depending on whether the scalar is positive or negative, the bounds of the resulting output $[\underline{x'_1}, \overline{x'_1}] = w.[\underline{x_1}, \overline{x_1}]$ can be obtained as follows:

$$\begin{aligned} w.[\underline{x_1}, \overline{x_1}] &= (w < 0) \implies [w.\overline{x_1}, w.\underline{x_1}]) \\ &\wedge (w \geq 0) \implies [w.\underline{x_1}, w.\overline{x_1}]) \end{aligned} \tag{5.1}$$

The summation of the bounds of the resulting products involves the addition of individual lower and upper bounds:

$$[\underline{y_1}, \overline{y_1}] = [\underline{x'_1}, \overline{x'_1}] + [\underline{x'_2}, \overline{x'_2}] = [\underline{x'_1} + \underline{x'_2}, \overline{x'_1} + \overline{x'_2}] \tag{5.2}$$

The addition of bias value (scalar) b then translates the entire bounds by the value b:

$$[\underline{y'_1}, \overline{y'_1}] = [\underline{y_1}, \overline{y_1}] + b = [\underline{y_1} + b, \overline{y_1} + b] \tag{5.3}$$

As stated in Eq. 3.2, the ReLU activation only alters negative input values, while providing only identity mapping for the positive inputs. Hence, the resulting bounds after ReLU activation can be calculated as follows:

$$max(0, [\underline{y'_1}, \overline{y'_1}]) = [max(0, \underline{y'_1}), max(0, \overline{y'_1})] \tag{5.4}$$

The bounds computed above are precise and exact, and correspond to the valid output domain for each of the network nodes. (Note that this does not hold for non-piecewise linear activations - the application of linear arithmetic to which would require approximation of the function [SGPV19]. This would lead to imprecise, but *not* inaccurate bounds.)

Stratified Sampling

The bounds computed from the previous step are used to form $\mathcal{S}$ non-overlapping *strata*, to enable sampling. For instance, consider again the neuron x_1 with bounds $[\underline{x_1}, \overline{x_1}]$. The bounds of the stratum for the node x_1 are as follows:

$$[\underline{x_1}, \overline{x_1}] = \bigcup_{i=0}^{\mathcal{S}-1} \{[\frac{\overline{x_1} - \underline{x_1}}{\mathcal{S}} i, \frac{\overline{x_1} - \underline{x_1}}{\mathcal{S}}(i+1)]\} \tag{5.5}$$

The combination of samples taken from all unique combinations of strata of different input nodes is, in turn, used to compute the output of the nodes in the following layer. This approach of selecting random samples from the individual stratum instead of the entire input nodes' bounds, known as the *stratified sampling*, ensures that the response of output to a wide range of input samples is considered in the analysis. Hence, the sampling provides a "wider coverage" of input bounds, and in turn, provides "better precision" in calculated outputs. The result is the *joint score* of the input-output node stratum, which will be elaborated further later in the section.

However, to consider the combination of inputs from all strata belonging to all input nodes entail a time complexity of $O(\mathcal{S}^N)$, where $\mathcal{S}$ is the number of strata for each of the N nodes in a DNN layer. Therefore, QuanDA uses an apt *group formation* to reduce this timing complexity.

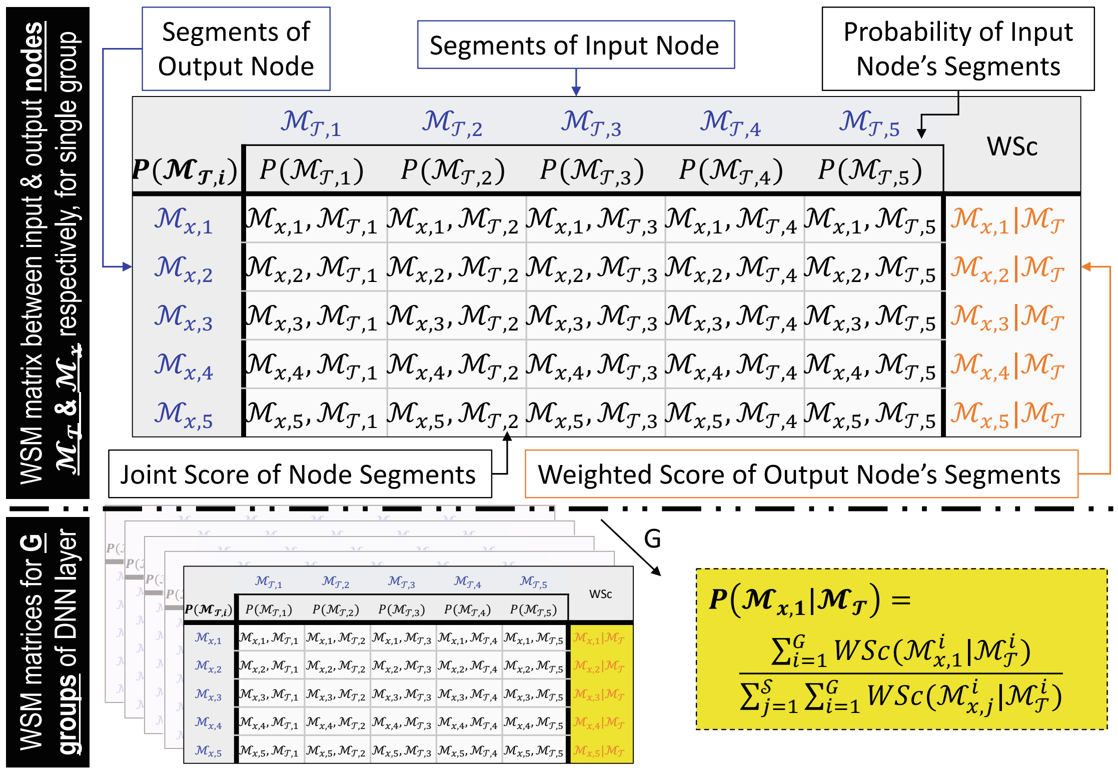
Group Formation. For DNN layers with a large number of nodes (i.e., $N \geq th$), the nodes are distributed into G groups comprising of N_G nodes each. Naturally, this calls for the recomputation of the bounds of output nodes i.e., nodes of the subsequent layer, with respect to the input nodes in the group ($\in N_G$). The affine transformation is hence split into two stages. The first stage involves only the multiplication of nodes in individual groups to their respective weights (i.e., $x'_{j*} = \sum_{k \in N_G} w_{kj}.x_k$). The second stage adds the output results from all groups together, along with the corresponding bias value, to obtain the final output for the node (i.e., $x'_j = b_j + \sum_G x'_{j*}$). This reduces the time complexity of the sampling to $O(G.\mathcal{S}^{N_G} + (N_G)^G)$.

The sampling approach described so far entails that samples selected from each stratum have an equal probability, hence indicating a uniform distribution. To overrule such an assumption, QuanDA takes inspiration from the WSM, to incorporate the precise probability of reaching each stratum into the analysis to ensure realistic probability estimates. In general, WSM [And64] is a statistical approach to determine the likelihood of individual responses (i.e., outputs) to the various criteria (i.e., input). Each criterion, in turn, has a certain weight of occurrence associated with it.

In QuanDA, we use inspiration from WSMs to obtain the weighted score WSc for each stratum of the output node, given the probabilities of strata of the input node. For instance, let $\mathcal{M}_x^{\mathcal{G}}$ and $\mathcal{M}_\tau^{\mathcal{G}}$ be the output and input nodes, respectively, for group $\mathcal{G}$ of the DNN layer. The WSc for the strata s of $\mathcal{M}_x^{\mathcal{G}}$, given $\mathcal{M}_{\mathcal{T}}^{\mathcal{G}}$, can then be determined as:

$$WSc(\mathcal{M}_{x,s}^{\mathcal{G}}|\mathcal{M}_{\mathcal{T}}^{\mathcal{G}}) = \sum_{i=1}^{\mathcal{S}} Score(\mathcal{M}_{x,s}^{\mathcal{G}}, \mathcal{M}_{\mathcal{T},i}^{\mathcal{G}}).P(\mathcal{M}_{\mathcal{T},i}^{\mathcal{G}}) \tag{5.6}$$

This is also depicted pictorially, using the WSM matrix, in Fig. 5.9. Each cell of the WSM matrix gives the score for reaching the specific output stratum given individual input stratum, i.e., the *joint score* of input-output node stratum. The probabilities of the individual input node stratum are used as weights for computing WSc of each stratum of the output node, as shown in the last column of the WSM matrix (in Fig. 5.9). This computation of WSc (and the subsequent probability estimation described in the next subsection) are carried out layer-wise, with probabilities of nodes from each layer used as weights to compute WSc at the next layer.



WSM matrix between input & output nodes $\mathcal{M}_{\mathcal{T}}$ & $\mathcal{M}_x$ respectively, for single group
Segments of Output Node
Segments of Input Node
Probability of Input Node's Segments
$\mathcal{M}_{\mathcal{T},1}$ $\mathcal{M}_{\mathcal{T},2}$ $\mathcal{M}_{\mathcal{T},3}$ $\mathcal{M}_{\mathcal{T},4}$ $\mathcal{M}_{\mathcal{T},5}$
WSc
$P(\mathcal{M}_{\mathcal{T},i})$ $P(\mathcal{M}_{\mathcal{T},1})$ $P(\mathcal{M}_{\mathcal{T},2})$ $P(\mathcal{M}_{\mathcal{T},3})$ $P(\mathcal{M}_{\mathcal{T},4})$ $P(\mathcal{M}_{\mathcal{T},5})$
$\mathcal{M}_{x,1}$ $\mathcal{M}_{x,1},\mathcal{M}_{\mathcal{T},1}$ $\mathcal{M}_{x,1},\mathcal{M}_{\mathcal{T},2}$ $\mathcal{M}_{x,1},\mathcal{M}_{\mathcal{T},3}$ $\mathcal{M}_{x,1},\mathcal{M}_{\mathcal{T},4}$ $\mathcal{M}_{x,1},\mathcal{M}_{\mathcal{T},5}$ $\mathcal{M}_{x,1}|\mathcal{M}_{\mathcal{T}}$
$\mathcal{M}_{x,2}$ $\mathcal{M}_{x,2},\mathcal{M}_{\mathcal{T},1}$ $\mathcal{M}_{x,2},\mathcal{M}_{\mathcal{T},2}$ $\mathcal{M}_{x,2},\mathcal{M}_{\mathcal{T},3}$ $\mathcal{M}_{x,2},\mathcal{M}_{\mathcal{T},4}$ $\mathcal{M}_{x,2},\mathcal{M}_{\mathcal{T},5}$ $\mathcal{M}_{x,2}|\mathcal{M}_{\mathcal{T}}$
$\mathcal{M}_{x,3}$ $\mathcal{M}_{x,3},\mathcal{M}_{\mathcal{T},1}$ $\mathcal{M}_{x,3},\mathcal{M}_{\mathcal{T},2}$ $\mathcal{M}_{x,3},\mathcal{M}_{\mathcal{T},3}$ $\mathcal{M}_{x,3},\mathcal{M}_{\mathcal{T},4}$ $\mathcal{M}_{x,3},\mathcal{M}_{\mathcal{T},5}$ $\mathcal{M}_{x,3}|\mathcal{M}_{\mathcal{T}}$
$\mathcal{M}_{x,4}$ $\mathcal{M}_{x,4},\mathcal{M}_{\mathcal{T},1}$ $\mathcal{M}_{x,4},\mathcal{M}_{\mathcal{T},2}$ $\mathcal{M}_{x,4},\mathcal{M}_{\mathcal{T},3}$ $\mathcal{M}_{x,4},\mathcal{M}_{\mathcal{T},4}$ $\mathcal{M}_{x,4},\mathcal{M}_{\mathcal{T},5}$ $\mathcal{M}_{x,4}|\mathcal{M}_{\mathcal{T}}$
$\mathcal{M}_{x,5}$ $\mathcal{M}_{x,5},\mathcal{M}_{\mathcal{T},1}$ $\mathcal{M}_{x,5},\mathcal{M}_{\mathcal{T},2}$ $\mathcal{M}_{x,5},\mathcal{M}_{\mathcal{T},3}$ $\mathcal{M}_{x,5},\mathcal{M}_{\mathcal{T},4}$ $\mathcal{M}_{x,5},\mathcal{M}_{\mathcal{T},5}$ $\mathcal{M}_{x,5}|\mathcal{M}_{\mathcal{T}}$
Joint Score of Node Segments
Weighted Score of Output Node's Segments
WSM matrices for G groups of DNN layer
G
$P(\mathcal{M}_{x,1}|\mathcal{M}_{\mathcal{T}}) = \frac{\sum_{i=1}^{G} WSc(\mathcal{M}_{x,1}^{i}|\mathcal{M}_{\mathcal{T}}^{i})}{\sum_{j=1}^{\mathcal{S}}\sum_{i=1}^{G} WSc(\mathcal{M}_{x,j}^{i}|\mathcal{M}_{\mathcal{T}}^{i})}$

Figure 5.9: (Top): WSM matrix between nodes $\mathcal{M}_{\mathcal{T}}$ and $\mathcal{M}_x$; (Bottom): Computation of probability estimate, using WSM matrices of all groups in the DNN layer.

Parallelization. It must be noted that sampling and WSc computation for an output node are independent of similar computations for all other output nodes in the same network layer. QuanDA leverages this independence in computation by computing results for all output nodes in a layer in parallel. This parallelism is greatly aided by using the parallel processing capabilities of the GPU, hence accelerating the DNN analysis.

Probability Estimation

Depending on whether the nodes in a layer were distributed over G groups or not, each DNN layer computes one or more WSM matrices for each input-output node pair. The probability of the strata s of output node N_x can then be estimated (also shown in Fig. 5.9) as:

$$\begin{aligned} P(\mathcal{M}_{x,s}|\mathcal{M}_\tau) &= \frac{\sum_{i=1}^{G} WSc(\mathcal{M}^i_{x,s}|\mathcal{M}^i_\tau)}{\sum_{j=1}^{\mathcal{S}}\sum_{i=1}^{G} WSc(\mathcal{M}^i_{x,j}|\mathcal{M}^i_\tau)} \\ P(\mathcal{M}_{x,s}) &= \prod_{i=1}^{|\tau|} P(\mathcal{M}_{x,s}|\mathcal{M}_i) \end{aligned} \tag{5.7}$$

where $|\tau|$ is the total number of input nodes contributing to the computation of $\mathcal{M}_x$.

Correctness Criteria. The correct computation of the probability estimates, as described above, entails the probabilities of the strata of each DNN node must sum to 1:

$$\sum_{i=1}^{\mathcal{S}} P(\mathcal{M}_{x,i}) = 1 \tag{5.8}$$

The current framework focuses on mainly two DNN specifications, i.e., *reachability* and *safety*. Probabilistically, reachability can be expressed as the probability of reaching a subset (or strata) $\mathcal{Y}_s$ of output, given the input $[\underline{x}, \overline{x}] \in \chi$, i.e., $P(\mathcal{Y}_s|[\underline{x}, \overline{x}])$. Hence, the probability estimation approach explained earlier provides the probabilistic reachability estimates without any extra computations. Additionally, the layer-wise computation in QuanDA allows not only to obtain reachability results at the output layer but also for all nodes in the hidden layer(s).

For the safety analysis, QuanDA obtains the probability estimate by modifying the WSM matrices at the DNN's *output layer* such that each output node comprises of two strata, i.e., the one corresponding to the desired (safe) output region and the other corresponding to the unsafe region. The rest of the probability estimation proceeds similarly to the previously explained procedure. The final result is the probability estimate $P(\mathcal{Y}_s|[\underline{x}, \overline{x}])$, which is the precise estimate for the output to lie in the (safe) desired output region given the input bounds $[\underline{x}, \overline{x}]$.

Bounds on Estimate

Similar to available quantitative analysis frameworks for DNNs [CFGP20, YLL⁺21b], QuanDA also provides only an estimate for the actual probability of the properties holding true for the trained DNNs. However, unlike the prior efforts, QuanDA tries to ensure that the probability estimates obtained lie within certain pre-defined (acceptable) bounds. This is to allow more precise and accurate probability estimation. Towards this end, QuanDA leverages upon Hoeffding's inequality[1] [Hoe94] to identify the minimum number of iterations (i.e., n) the probability estimation needs to be repeated to ensure estimated probabilities have converged towards the exact probabilities. The estimates from these n iterations are averaged out to obtain the calculated mean M_n, while $E(M_n)$ represent the (exact) expected mean of the probabilities:

$$P(|M_n - E(M_n)| \geq t) \leq 2e^{-\frac{2n^2t^2}{\sum_{i=1}^{n}(b_i-a_i)^2}} < \delta \tag{5.9}$$

Based on the above inequality, the maximum deviation between the estimated and exact mean of probabilities remains below t. δ represents the significance level, while $1-\delta$ is the confidence interval ensuring that the deviation between the exact and estimated probabilities stays less than t. b_i and a_i represent the upper and lower bounds of the estimates, respectively. Since QuanDA deals with the probability estimates, these bounds can be equated to 1 and 0, respectively, reducing the above inequality to:

$$P(|M_n - E(M_n)| \geq t) \leq 2e^{-2nt^2} < \delta \tag{5.10}$$

As indicated earlier, the parameters t and δ are user-defined, and can be modified to vary the accuracy and precision of probability estimates.

5.2.2 Experimental Evaluation

We implemented QuanDA on AMD Ryzen Threadripper 2990WX CPUs and NVIDIA GeForce RTX 2080 Ti GPUs. Each CPU has 32 cores with 64 threads, with the maximum boost clock rate of 4.2GHz. Each GPU, on the other hand, hosts 4352 CUDA cores at 1635MHz GPU boost clock rate, providing it an overall 14.2TFLOPS. This means, the GPU is capable of handling 14.2 trillion floating point computations per second. The GPU hosts 11GB GDDR6 device memory with 616GB/s peak memory bandwidth. The systems use CUDA 11.6 toolkit and run on Ubuntu 18.04LTS.

The framework is written in Python and uses Numba [LPS15] for CUDA GPU programming support. The current version of QuanDA accepts trained DNNs and input bounds in `.nnet` and `.npy` formats, respectively. The probability estimates at the output are stored in the `.npy` format as well. Fig. 5.10 summarizes the various operations (described in Section 5.2.1) and flow of data throughout QuanDA, distinguishing the operations carried out by the CPU and the GPU.

[1]Hoeffding's inequality applies to independent random variables - this holds reachability probability estimates of the network nodes feedforward networks since each computation of the probability is independent of the other probability estimates. Note that the relation between: (1) the deviation t between exact M_n and estimated $E(M_n)$ of probabilities, (2) confidence interval $(1-\delta)$ of the deviation, and the number of iterations n required to achieve the confidence interval, established by Hoeffding's inequality, is of the accuracy of the individual probability estimates $E(M_n)$.

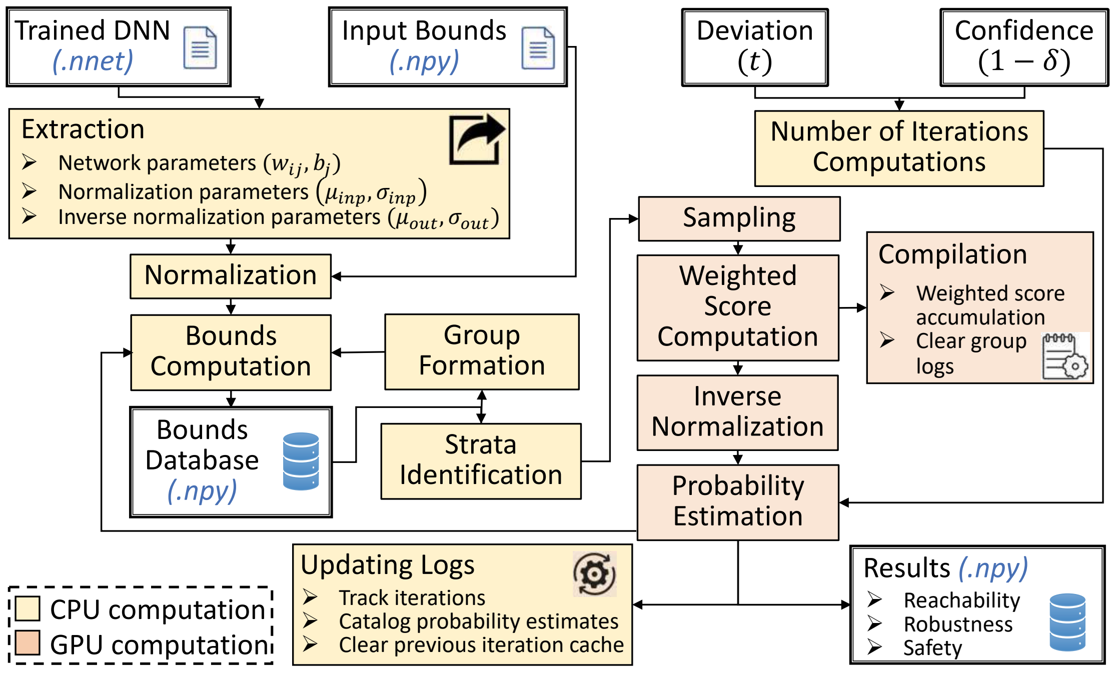


Figure 5.10: Overview of the python-based QuanDA toolchain, using trained DNN and input bounds, to generate the probability estimates for the desired DNN property.

Experimental Setup

We use the aircraft collision avoidance system, ACAS Xu neural networks [KBD+17] to provide the quantitative analysis using QuanDA. This is a well-known benchmark opted for both qualitative and quantitative DNN analysis [KHI+19, CFGP20]. The benchmark comprises of 45 fully-connected feed-forward DNNs, with 6 hidden layers each consisting of 50 nodes. The networks use ReLU activation for all hidden layers. The output layers instead use the minpool activation, i.e., the output node with the minimal value is chosen as the DNNs' decision. Each network accepts five inputs (i_{1-5}), i.e., the distance between ownship and intruder, and the directions and speeds of the aircraft. Likewise, the output layer provides five possible maneuvering decisions (o_{1-5}) by the networks, i.e., clear-of-conflict (COC), weak left/right and strong left/right.

The inputs are normalized prior to being fed to the DNNs, and inverse normalization is deployed to the output prior to decision-making. Additionally, we take all input values to be equally likely. However, as stated earlier, we use no assumption for the probability distribution of node values in any of the following DNN layers. In all our experiments, we use 5 strata to distribute the bounds of all DNN nodes. Hence, the analysis of reachability property (discussed in Section 5.2.1) provides the probability estimates $P(\mathcal{Y}_s|[\underline{x}, \overline{x}])$, where $s \in \mathbb{Z}^+ : s \leq 5$. Additionally, we use s^N samples for the computation of probability estimates for each node. This is the minimum number of samples required to ensure the maximum coverage during probability estimation, i.e., a combination of input samples from all strata of each node are considered during computation.

We also show the potential application of QuanDA to estimate the probability of the following safety properties:

Property 1 (φ_1). If the intruder is far away and flying slower than the ownship (i.e., $i_1 \geq 55947.691 \wedge i_4 \geq 1145 \wedge i_5 \leq 60$), the COC output stays within a certain threshold (i.e., $o_1 \leq 1500$).

Property 2 (φ_2). If the intruder is far away and flying slower than the ownship (i.e., $i_1 \geq 55947.691 \wedge i_4 \geq 1145 \wedge i_5 \leq 60$), the COC output is not maximal (i.e., $max_{i \in \mathbb{Z}^+ : i \leq 5}(o_i) \neq o_1$).

Property 3 (φ_3). If the intruder is directly ahead and flying slower than the ownship (i.e., $1500 \leq i_1 \leq 1800 \wedge -0.06 \leq i_2 \leq 0.06 \wedge i_3 = 0 \wedge i_4 \geq 1000 \wedge 700 \leq i_5 \leq 800$), the COC is not minimal (i.e., $min_{i \in \mathbb{Z}^+ : i \leq 5}(o_i) \neq o_1$).

We use the maximum deviation between exact and estimated probabilities (t) to be 0.05, and a confidence interval ($1 - \delta$) of 87% for all our experiments.

5.2.3 Results and Discussion

A single iteration of probability estimate of the safety properties (checked at the output nodes) takes 4.5min. The reachability probability estimates of the hidden layer nodes naturally have a smaller timing overhead. Compared to a complete CPU implementation (i.e., single-threaded execution without using GPU), this amounts to roughly a 3600 times speed up[2].

Fig. 5.11 shows the box and whisker plot of the output from randomly selected DNNs nodes from the first hidden layer of two of the ACAS Xu networks. The output of the nodes using different properties (and hence, different input bounds) are shown. The graphs provide the *reachability* probability estimates $P(\mathcal{Y}_s | [\underline{x}, \overline{x}])$ for the randomly selected nodes. The reachability results from the remaining nodes show a similar diversity of probability estimates over the different node strata.

It is interesting to note that despite all inputs being equally likely (i.e., uniformly distributed inputs), the DNN transformations of even a single layer lead the nodes in the hidden layer to lose the uniformity of the input distribution. This invalidates the uniform distribution assumption used for DNN nodes in the existing literature [CFGP20, YLL$^+$21b]. In fact, the output nodes do not follow *any* univariate probability distribution. This could be partially attributed to the affine transformation, where (scaled)

[2]It must be noted that the execution time speedup between complete CPU implementation and CPU-GPU was checked only for the first hidden layer of the DNNs. This is because the timing overhead of the CPU implementation is too large to be of practical advantage for *deep* networks.

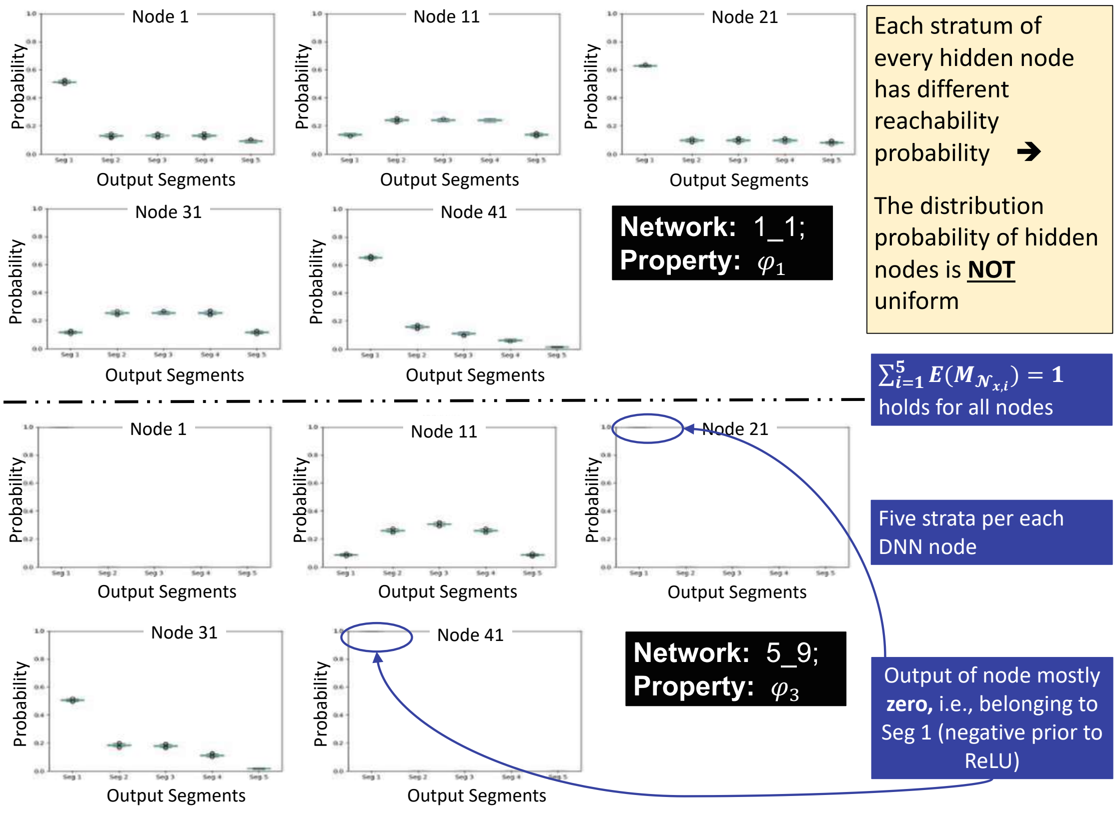


Figure 5.11: The probability estimates for 5 randomly selected DNN nodes from the first hidden layer of different DNNs, using input bounds from different properties: as expected, the nodes do not follow any univariate probability distribution.

discrete uniform distribution of the multiple input nodes, spanning over varying input bounds, combine to result in a multivariate distribution. Additionally, the consequent application of ReLU activation maps all negative results to zero. The effect of this can be observed clearly for nodes 1, 21 and 41 of the network 5_9 (in the bottom row of graphs in Fig. 5.11), where the strata containing output zero have probabilities of ~ 1.0.

Our observations from the reachability hence confirm that even for applications where the input is uniformly distributed, it is very unlikely for nodes in consequent layers to follow a uniform distribution. The realistic probability estimation of DNN properties hence requires careful consideration and propagation of the probability distributions of the hidden DNN nodes.

We additionally observe the probability estimates for the ACAS Xu safety properties φ_{1-3} using QuanDA. The probability estimates for the aforementioned properties to hold for different ACAS Xu networks are indicated in Table 5.1. The additional results for the convergence of probability estimates with increasing confidence are provided in

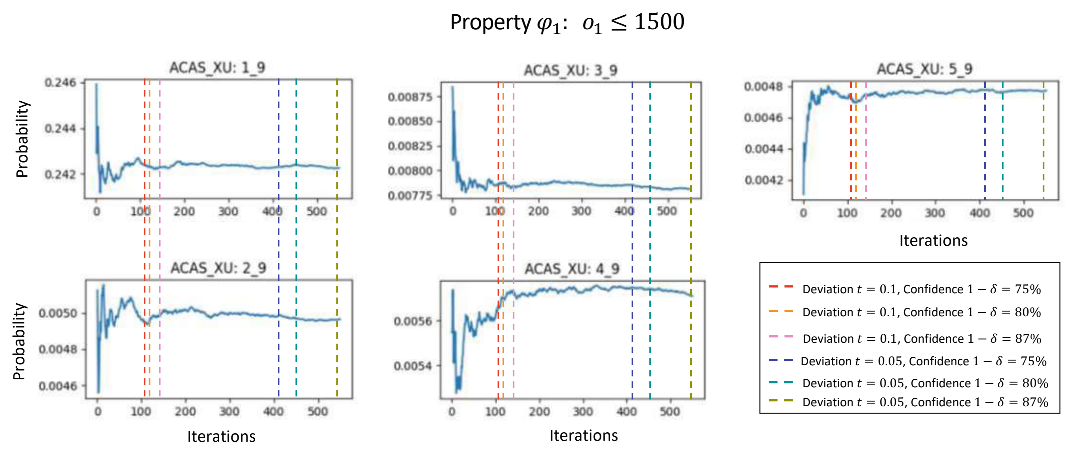


Figure 5.12: Running average of the estimated probability for the ACAS Xu networks, for φ_1, indicating probability estimates at different deviation and confidence levels. The plots for the remaining networks follow a similar pattern.

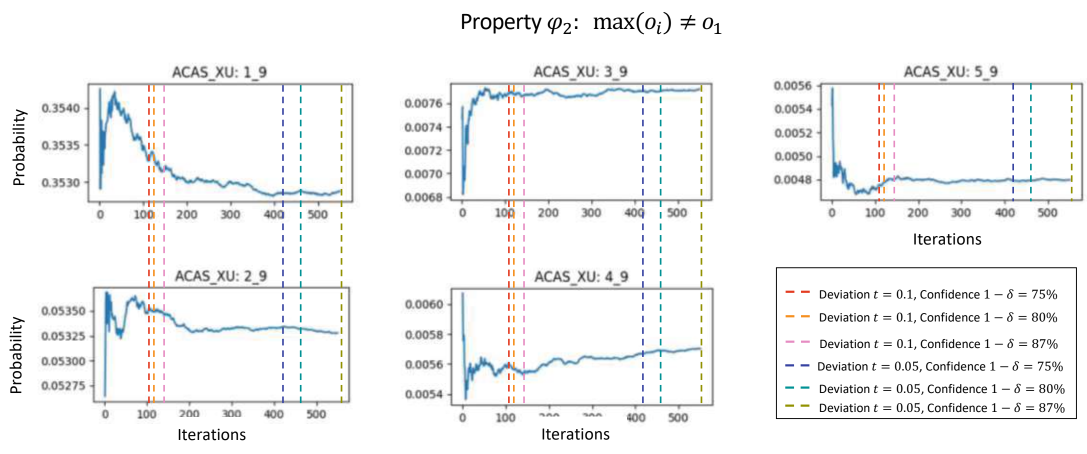


Figure 5.13: Running average of the estimated probability for the ACAS Xu networks, for φ_2, indicating probability estimates at different deviation and confidence levels. The plots for the remaining networks follow a similar pattern.

Figs. 5.12-5.14. Even though the bounds on estimates are not available for the final DNN output (the bounds provided by Hoeffding's inequality hold for individual layers), QuanDA still provides quantitative probability estimates. This highlights the potential of the framework for estimating the probability of a diverse range of DNN properties.

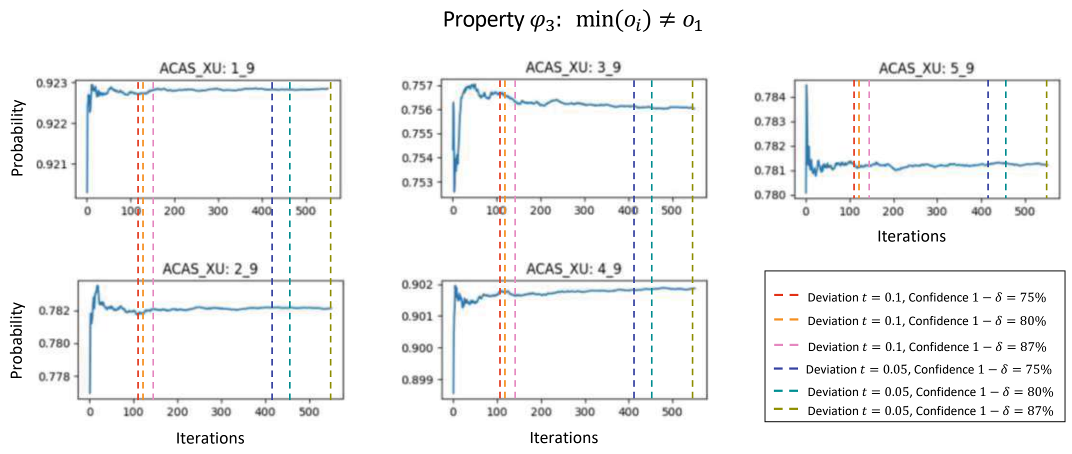


Figure 5.14: Running average of the estimated probability for the ACAS Xu networks, for φ_3, indicating probability estimates at different deviation and confidence levels. The plots for the remaining networks follow a similar pattern.

5.3 Summary of Quantitative DNN Analysis

The choice of the best-suited noise model, along with a more broad-spectrum noise analysis are essential tools for ensuring the high accuracy of ML-based systems deploying DNNs in noisy, real-world environments. While L^p norm-based noise models are often a popular choice for the DNN robustness analysis, our probabilistic analysis of trained DNN emphasizes its inadequacy for analyzing impacts of noise beyond robustness. Particularly, these models are not ideal for DNNs where the different input nodes have different ranges of values.

This chapter also introduced our novel quantitative neural network analysis framework QuanDA, which estimates the precise probabilities for the outputs of DNN nodes, without relying on any assumption on the probability distribution of network nodes. It makes use of efficient CPU and GPU processing, to compute the probability estimates layer-wise with user-defined deviation and confidence intervals, and efficiently parallelize the computations to ensure fast analysis. The framework is used to provide reachability probability estimates of the nodes in the ACAS Xu networks benchmark. We also show the potential application of QuanDA for three safety properties of the indicated networks. Hoeffding's inequality allows a trade-off between confidence/deviation between exact and estimated probabilities, and the number of repeated experiments. This could be leveraged to improve the scalability of the framework under restrictive timing constraints.

Up till now, we established the need for formal DNN analysis, and introduced qualitative and quantitative analysis frameworks to provide formal behavioral guarantees for these trained networks. These corresponded to the prime research goals of this research. The next chapter now explores ways to use the knowledge obtained from the formal DNN

Table 5.1: Probability estimates for the safety properties φ_{1-3}, using deviation $t = 0.05$ and confidence $1 - \delta = 87\%$.

DNN	φ_1	φ_2	φ_3	**DNN**	φ_1	φ_2	φ_3	**DNN**	φ_1	φ_2	φ_3
1_1	0.01	0.63	0.6	2_7	0.01	0.01	0.78	4_4	0.01	0.29	0.76
1_2	0.01	0.62	0.63	2_8	0.01	0.01	0.78	4_5	0.01	0.09	0.76
1_3	0.66	0.92	0.6	2_9	0	0.05	0.78	4_6	0	0	0.99
1_4	0.26	0.66	0.56	3_1	0	0.62	0.4	4_7	0	0	0.78
1_5	0.01	0.6	0.62	3_2	0	0.52	0.56	4_8	0.01	0.01	0.78
1_6	0.01	0.57	0.76	3_3	0.01	0.42	0.52	4_9	0.01	0.01	0.9
1_7	0.01	0.53	0.79	3_4	0.01	0.34	0.65	5_1	0	0.32	0.66
1_8	0.76	0.59	0.94	3_5	0	0	0.93	5_2	0.01	0.38	0.57
1_9	0.24	0.35	0.92	3_6	0.01	0.01	0.77	5_3	0.01	0.17	0.65
2_1	0.01	0.35	0.65	3_7	0.01	0.01	0.77	5_4	0.01	0.38	0.57
2_2	0	0.41	0.57	3_8	0.01	0.01	0.79	5_5	0.01	0.07	0.77
2_3	0.01	0.33	0.62	3_9	0.01	0.01	0.76	5_6	0.01	0.01	0.77
2_4	0.01	0.74	0.66	4_1	0.01	0.32	0.69	5_7	0.01	0.01	0.8
2_5	0.01	0.04	0.77	4_2	0.01	0.26	0.6	5_8	0.01	0.01	0.78
2_6	0.01	0.01	0.78	4_3	0.01	0.32	0.61	5_9	0	0	0.78

analysis to potentially strengthen the behavioral performance of these DNNs.

CHAPTER 6

Robustifying Neural Networks

An essential aim for providing behavioral guarantees for trained DNNs is to amend the vulnerabilities of these DNNs to ultimately obtain reliable networks, which in turn could enable the design of reliable ML-based systems. The existing literature proposes multiple directions to provide such *robustification* of trained DNNs, either implicitly or explicitly, by leveraging the analysis results. For instance, patching has been proposed for DNNs, which updates the network parameters to provide safer classification boundaries for the networks [ST19, STT23]. Adding verification objectives (i.e., the specification for desired DNN behavior) to the training loop has also been explored in an attempt to eliminate DNN vulnerabilities during the training process itself [APDC19].

On the other end, an adaptation of the overall formal analysis approach to overcome the challenges of the existing works is also gaining more attention. Notable efforts in this direction include incremental DNN verification [UBMS23], where instead of verifying the DNN from scratch after each parameter update, only partial verification is required. Analysis of relatively newer network architectures, like quantized DNNs and Spiking Neural Networks (SNNs) are also being explored [HLŽ21, MSB22, BGBM23] due to the increasing popularity of these architectures. Additionally, the possibility of noise affecting the network parameters is also considered in recent literature [Mey23] to broaden the horizon of DNN analysis to include parameter perturbation (instead of simply input perturbations).

Hence, there appear to be two natural directions to pursue after successful DNN analysis: (1) adaptation of existing DNN analysis frameworks to possibly discover new DNN vulnerabilities, and (2) provide methodologies to potentially improve the reliability of the DNNs. We work in both directions (see chapter summary in Fig. 6.1). In particular the novel contributions of this research, focused in this chapter, include:

1. The exploration of node level robustness bias, similar to the class level robustness bias identified in previous chapters.

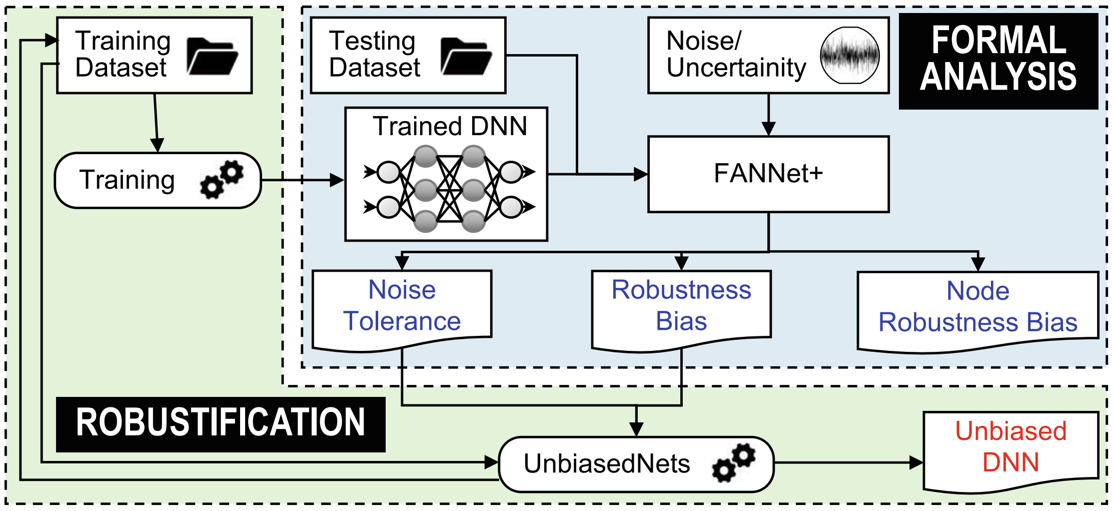


Figure 6.1: Summary to DNN analysis and robustification explored in the chapter.

2. Define the notion of the balanced dataset, which is essential to understanding and addressing the robustness bias.

3. Explore the effectiveness of existing dataset-centric approaches to analyze their impact on minimizing robustness bias.

4. Introduce a systematic data diversification framework, UnbiasedNets, to provide balanced datasets and potentially minimize the robustness bias.

The rest of the chapter is organized as follows. Sec. 6.1 details the exploration of node robustness bias using our established frameworks provided in previous chapters. Sec. 6.2 provides potential direction for the minimization of robustness bias from training datasets via the development of UnbiasedNets framework. The works are accompanied by case studies, while Sec. 6.3, finally summarize our robustification efforts.

6.1 Link between bias and node sensitivity

Long-tail distribution is a widely studied challenge in DNN research community, since it is often associated with a varying classification performance of the network for *head* and *tail* classes [Lea18, Fea22]. The concerns surrounding long-tail distribution are not ill-founded. Numerous available datasets, in fact, comprise of long-tail distribution. The MIT-BIH Arrhythmia dataset [Mea01] contains a significant proportion of normal ECG samples (as opposed to ECG samples indicating arrhythmia). The IMDB-WIKI dataset [RTVG18] comprises of a significantly larger proportion of Caucasian faces. Wafer map training dataset [Wea14] comprises of a bigger proportion of fault-free wafers (as opposed to faulty wafers). Such discrepancy in the number of inputs across different output classes is not always surprising, since the tail classes often present rare events of the real-world.

It is not also surprising that the DNNs trained on long-tail distribution learn the patterns in head classes better than those in the tail classes due to the availability of ample input samples. It has also been observed that such networks also delineate a robustness bias under the influence of noise, i.e., the network is likely to correctly classify even noisy inputs from the head class(es), while the robustness of the tail class(es) against noise is only negligible [tMK$^+$20, NDS$^+$21]. Numerous pre-training, training, fine-tuning and transfer-learning approaches have been proposed to resolve this discrepancy and ensure the overall classification performance stays consistent for all classes [Zea21, Sea21, Wea17, Cea02, Yea19, Lea20].

Orthogonally, the sensitivity of input nodes has also been found to vary [BtSH22]. While this variation comes in handy while determining the relevant input nodes for the designated task of the trained DNN [ZMC94, CRF$^+$20], it may also pose itself as a concern for applications where a revelation of the sensitive attributes (nodes) may lead to a privacy infringement [ZL19, MFVRT20].

However, there is another aspect of concern for DNNs, which is inadvertently linked to those indicated above - i.e., the (robustness) bias of the individual input nodes - which remains unrecognized in the existing literature. This section deals with such node bias, indicating its stealthy existence and the non-triviality of understanding its causes. To summarize, the novel contributions of this section are as follows:

1. Highlighting the link between robustness bias, node sensitivity, and node bias.
2. Identifying the existence of node bias and empirically analyzing it in a network trained on a real-world leukemia dataset [GST$^+$99].
3. Discussing the severity of node bias with respect to long-tail distribution of the training dataset.
4. Elucidating the open challenges pertaining to node bias, in trained DNNs.

6.1.1 Proposed Methodology for the Study of Node Robustness Bias

Fig. 6.2 provides an overview of our proposed analysis framework to study the node (robustness) bias of the trained networks. The architecture and parameter details of the trained DNN are initially used to construct the formal model of the network [BtSH22]. The formal model is validated using inputs from the testing dataset (i.e., the correct and

©2023 IEEE. Parts reproduced, with permission, from M. Naseer, M. Shafique: Poster: Link between Bias, Node Sensitivity and Long-Tail Distribution in trained DNNs, *Conference on Software Testing, Verification and Validation (ICST)*, Dublin, Ireland, 16-20 April, 2023, pp. 474-477. [DOI:doi.org/10.1109/ICST57152.2023.00054].

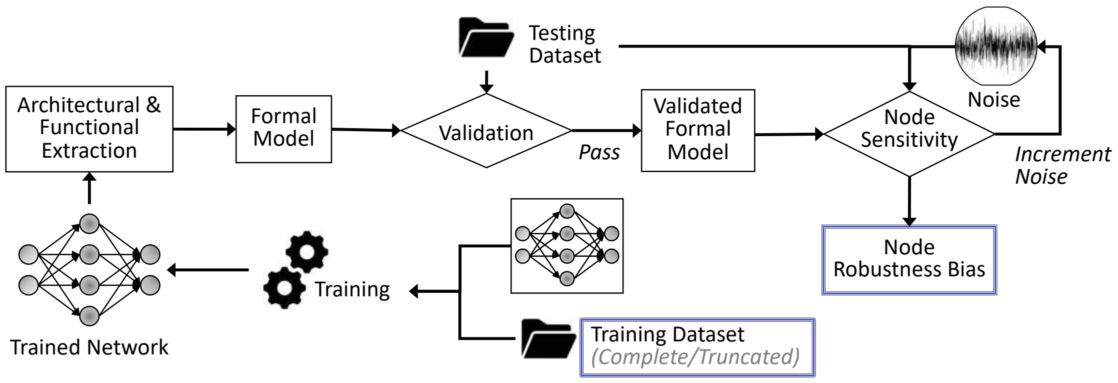


Figure 6.2: Overview of our proposed framework for node (robustness) bias analysis. The training dataset could be complete or truncated to obtain a dataset to avoid long-tail distribution.

model's computed output classification of the testing inputs are compared for consistency of results). The sensitivity of the input nodes is then analyzed using a probabilistic model checker as follows:

$$\mathbb{P}_{=?}[\mathbb{F}(\mathcal{N}(x) = \mathcal{N}(x \setminus x^i, x^i + \eta)) \wedge (\eta \leq N)],$$

where i is the node under sensitivity analysis and $\mathbb{F}(\mathcal{N}(x) = \mathcal{N}(x \setminus x^i, x^i + \eta))$ indicates that the network eventually provides correct output classification for input x. This is repeated iteratively, while gradually increasing the incident noise applied to the testing inputs. The exact node sensitivity results are then analyzed for individual input nodes to understand the node (robustness) bias. This is achieved initially using network trained on the complete dataset[1].

As indicated in earlier chapters, the classification performance of the trained network may vary for networks trained on training datasets with long-tail distribution. Intuitively, this suggests that a network trained on dataset with an equal number of inputs from each class might address the discrepancy in classification performance across different classes [Lea18] and ensure that node (robustness) bias holds for the network. To test the hypothesis, we truncate the training dataset by deleting inputs from the head class(es), and repeat the analysis on this new dataset (which no longer has a long-tail distribution).

6.1.2 Case Study

This section provides a case study to highlight the node (robustness) bias in a DNN trained on real dataset. This is followed by a discussion of the results and analysis.

[1]This corresponds to our probabilistic analysis setup using FANNet elaborated in Chap. 5.1

Experimental Setup

We again train single-hidden layer ReLU-based fully-connected neural networks on the top-5 relevant features of leukemia dataset [Kea18]. The training dataset comprises of 38 inputs, with the head class (i.e., ALL Leukemia) constituting approximately 70% of the dataset, while the tail class (i.e., AML Leukemia) constitutes the remaining dataset.

The experiments were repeated 10 times, while noting the results for the networks trained on the complete dataset. Similarly, 10 networks were also trained on a truncated dataset via deleting randomly selected subset of inputs from the head class, before each training. This ensures an equal number of inputs from each class. Storm model checker was used for the quantitative verification of node sensitivity. All experiments were run on AMDRyzen Threadripper 2990WX processors running Ubuntu 18.04 LTS operating system.

Results and Discussion

As indicated earlier, the training dataset of the leukemia dataset composes a long-tail distribution. Hence, the networks trained on it delineate robustness bias, with the increase in incident noise gradually decreasing the probability of correct classification of AML (tail class), but not for ALL (head class). This is presented by the blue lines in Fig. 6.3. The truncation of ALL inputs from the training dataset, in turn, generates networks that appear unbiased for at low incident noise (see the orange lines in Fig. 6.3). However, for large noise, the classification probability of ALL starts to decrease whereas the AML is correctly classified with a probability of ~ 1.0. This suggests that log-tail distribution is only a component of a much more complicated problem, leading to robustness bias. Hence, while avoiding long-tail distribution addresses robustness bias for smaller noise, the strategy alone may not be a sufficient bias reduction strategy for inputs exposed to larger noise.

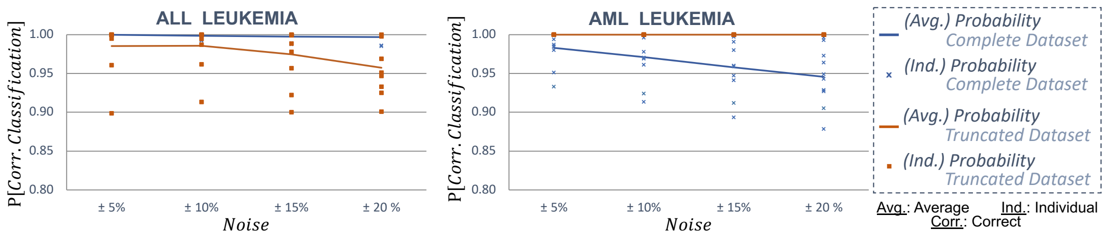


Figure 6.3: Network trained on long-tail distribution indicate robustness bias for AML (i.e., tail class). Truncation of training dataset prior to training shifts the robustness bias towards ALL instead, but only for large noise.

Similar trends are observed from the results of node sensitivity for negative noise, as shown in Fig. 6.4. Truncation of training dataset leads to network's input nodes having approximately equal classification probability for small noise. However, the probability of correct classification gradually decreases for ALL at higher noise. It can also be observed for the networks trained on the original dataset that the sensitivity of different nodes is visibly different, as observed by the corresponding gradients of the blue lines for AML.

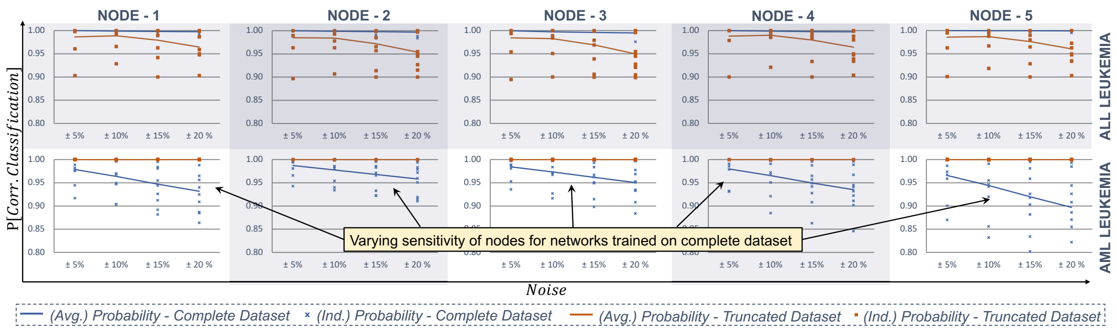


Figure 6.4: The node sensitivity against negative noise for ALL and AML Leukemia. The lines present the average probability results while points present result from each (of the 10) experiments. The sensitivity varies for networks trained on complete and truncated datasets.

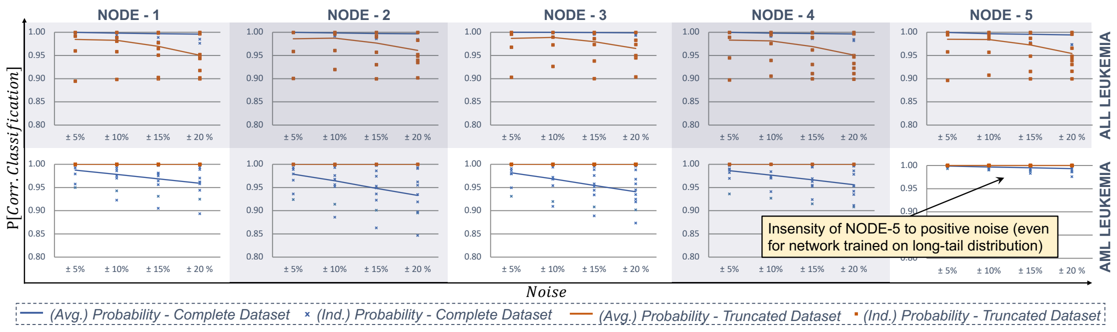


Figure 6.5: The node sensitivity against positive noise for ALL and AML leukemia. The lines present the average probability results while points present result from each (of the 10) experiments. The sensitivity varies for networks trained on complete and truncated datasets.

The observations for the analysis with positive incident noise provide similar results in case of ALL (see Fig. 6.5). However, a stark difference can be observed for the sensitivity results of NODE−5, for AML. Where with negative noise, NODE−5 was observed to be most sensitive to noise, in case of positive noise, the node is observed to be very insensitive. This indicates a significant node (robustness) bias for networks trained on the original dataset. It is also interesting to note that the similar node (robustness) bias is not observed for the *same input node* for inputs belonging to ALL leukemia, for DNNs trained on both the original and truncated datasets.

Such sensitivity of NODE−5 suggests a biased learning of the node for AML. It can also be observed through Table 6.1 that NODE−5 has the largest variance among all nodes, for AML. This could be a possible reason for the strange sensitivity of the node, and the subsequent node (robustness) bias. However, given a similar condition for NODE−3 for ALL (i.e., having the largest variance for ALL), a similar discrepancy for node (robustness) bias, between positive and negative noise, is not observed.

Table 6.1: Variance of input node values in training datasets. The extrema of the variances, for each class, are underlined.

	Training Dataset	
	ALL Variance (x10^3)	AML Variance (x10^3)
NODE - 1	114.27	129.72
NODE - 2	81.21	$\underline{11.71}$
NODE - 3	$\underline{5531.62}$	231.77
NODE - 4	$\underline{45.24}$	284.02
NODE - 5	156.40	$\underline{2271.00}$

6.2 UnbiasedNets

Typically, DNNs are trained on large datasets, with tens to hundreds of thousands of input samples, using various supervised training algorithms. Testing accuracy is often the most commonly (and possibly the only) used metric to analyze the performance of these DNNs. This spotlights two major limitations: (a) there is a notable reliance on large, labeled datasets, obtaining which is a significant challenge for the ML community, especially for new use-cases, and (b) the trained DNN may experience problems like robustness bias, i.e., the robustness of DNN to noise is not the same across all output classes, which accentuate in the presence of noisy real-world data. Even when large datasets are available, they may contain a significantly large number of samples from one output/decision class, i.e., the long-tail distribution.

Motivating Example

As shown earlier in Fig. 6.3 (for DNNs trained on leukemia dataset), mere truncation of samples from head class is insufficient to avoid robustness bias in trained network. This can be further observed in Fig. 6.6. The plot on the left shows the classification performance of DNN trained on long-tail distribution, under the application of varying noise. Not surprisingly, the trained DNN is more likely to misclassify inputs from the tail class. For the repeated experiments, with random deletion of inputs from the head class (Fig. 6.6 (right)), the robustness bias persists.

©2023 Springer. Parts reproduced, with permission, from M. Naseer, B. S. Prabakaran, O. Hasan, M. Shafique: UnbiasedNets: a dataset diversification framework for robustness bias alleviation in neural networks, *Machine Learning (2023)*, pp. 1-28. [DOI:`doi.org/10.1007/s10994-023-06314-z`].

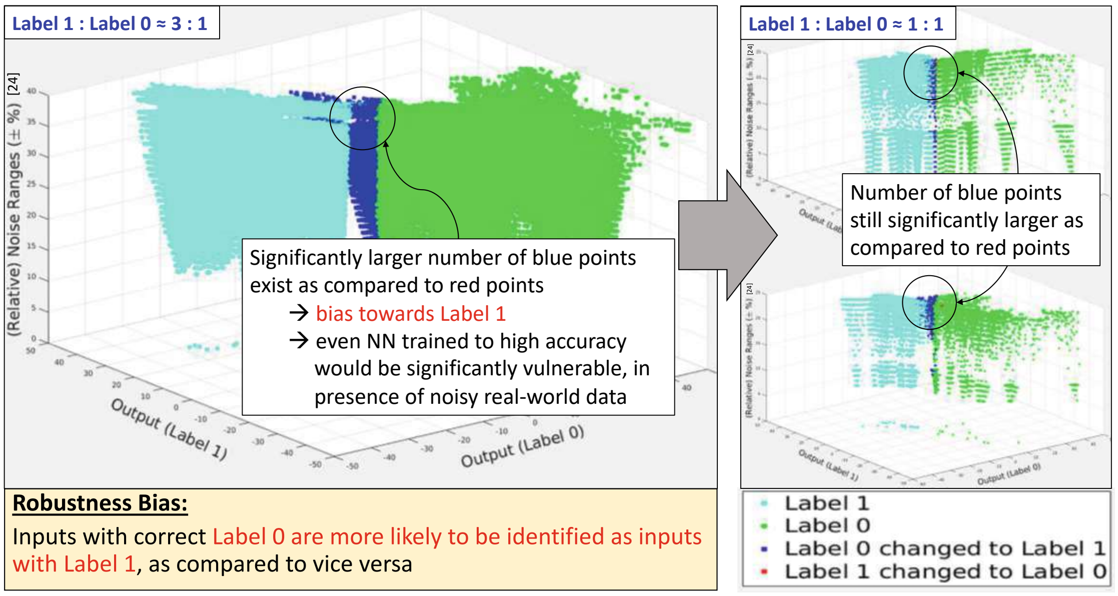


Figure 6.6: Networks trained on unequal (left) and equal (right) number of inputs from the classes: *Label* 0 and *Label* 1. All networks used the same network architecture and training hyper-parameters, and all indicate a higher likelihood of *Label* 0 being misclassified as compared to *Label* 1.

It must also be noted that the bias becomes apparent only in the presence of noise, since the trained DNNs do not indicate misclassifications in the absence of noise. Hence, the robustness bias in a trained DNN may go undetected before the deployment of the DNN in a real-world application if testing accuracy is the only analysis the DNN goes through before deployment in real-world. This gravitates the need to address robustness bias and calls for the better description and acquisition of *balanced datasets* that may enable training unbiased DNNs. However, obtaining such datasets is not a straightforward task.

The existing works dealing with bias alleviation either aim to improve the training algorithms to ensure unbiased training, or manipulate training data to obtain datasets that favor minimal DNN bias. The bias alleviation approaches can be broadly classified into two major categories: (1) unbiased training algorithms (i.e., algorithm-centric (AC) approaches), and (2) bias reduction via dataset manipulation (i.e., data-centric (DC) approaches). This is sumarrized in Table 6.2

Algorithm-Centric (AC) Approaches for Bias Alleviation

Training unbiased DNN via AC approaches often involves splitting the network model into two separate but connected networks [KKK$^+$19, AZN18, NCA$^+$20]. The first network aims at either identifying key input features or amplifying the bias present in the dataset. The second network, in turn, uses these features or accentuated bias to unlearn the bias from the network. Learning features at deeper DNN layers during training for

Table 6.2: Comparison of the state-of-the-art bias alleviation approaches with our proposed UnbiasedNets framework.

Recent Work	Small Dataset	Non-Visual Dataset	App-roach	Dataset Aug./Del.	Lever-ages Δx_{max}	X_{new} Validation
[AZN18]	✗	✗	AC	N/A	✗	N/A
[GSSH20]	✗	✗	AC	N/A	✗	N/A
[KKK+19]	✗	✗	AC	N/A	✗	N/A
[LV19]	✗	✗	AC	Del.	✗	N/A
[NCA+20]	✗	✗	AC	N/A	✗	N/A
[SWBR20]	✗	✓	AC	N/A	✗	N/A
[SWG20]	✗	✗	AC	N/A	✗	N/A
[ZWY+17]	✗	✗	AC	N/A	✗	N/A
[XLL+21]	✗	✗	AC	N/A	✗	N/A
[BZKK21]	✗	✗	AC	N/A	✗	N/A
[ZWL+19]	✗	✗	AC+DC	Aug.	✗	✗
[Cea02]	✓	✓	DC	Aug.	✗	✓
[Hea08]	✓	✓	DC	Aug.	✗	✓
[LNA17]	✓	✓	DC	Aug./Del.	✗	✓
[LBSB+20]	✗	✓	DC	Del.	N/A	N/A
[LLV18]	✗	✗	DC	Del.	N/A	N/A
UnbiasedNets	✓	✓	DC	Aug.+Del.	✓	✓

Aug.: Input Augmentation Del.: Input Deletion N/A: technique not applicable for scenario
Δx_{max}: Noise Tolerance X_{new}: Synthetic Data

data augmentation [ZWL+19] has also been shown to aid unbiased training. In addition, knowledge of known biases in the dataset and a DNN trained using standard cross-entropy loss has also been leveraged to develop a more robust DNN [SWBR20]. Other AC bias reduction approaches include the incorporation of additional constraints during training to guide the DNN in order to avoid learning unwanted correlations in data [ZWY+17].

For biases specific to multi-modal datasets (like colored MNIST [KKK+19], where the dataset contains two kinds of information: the colors and the numerals), the use of a training algorithm based on functional entropy is shown to perform better [GSSH20]. A recent work [LV19] also explores inputs in the dataset to identify the weights[2] that the inputs must be encoded with before training, to successfully reduce the bias. The determination of invariants in inputs has also been proposed [ABGLP19] to enable unbiased training of a DNN. In addition, recent work [SWG20] also explores algorithms where instead of training an unbiased network from scratch, a trained DNN and dataset

[2]Note that the weights for encoding inputs in [LV19] are not same as the parametric weights of DNN layers.

(not used during training) are used to fine-tune the network to be devoid of biases specific to a certain application.

However, as indicated earlier, these works are tailored for minimizing data and representation biases, generally for large datasets. The biases are often explored in visual datasets. In contrast, DNNs deployed in the real-world often also deal with non-visual inputs, like patient's medical data, where the existence of a bias (even the data and representation biases) may not always be easy to detect and hence may go unnoticed. Hence, bias alleviation poses a challenge in cases where the detection of bias is beyond visual perception. Moreover, the exploration of robustness bias is a fairly new research direction, and hence, the success of these AC approaches for minimizing robustness bias remains largely unexplored.

Data-Centric (DC) Approaches for Bias Alleviation

The orthogonal direction to minimize bias is by manipulating the training dataset via DC approaches, to potentially eliminate the bias at its core. Among the simplest and most popular DC bias alleviation approaches are random oversampling (ROS), i.e., random replication of inputs from the class with less number of input samples, or random undersampling (RUS), i.e., random deletion of inputs from the class with a significantly larger portion of available inputs [Lea18, BL21]. The idea is to obtain a dataset with an equal number of inputs from each class. However, RUS is known to reduce the number of input samples available for DNN to learn, while ROS may lead to overfitting the training data.

The synthetic minority oversampling (SMOTE) [Cea02] and adaptive synthetic sampling (ADASYN) [Hea08] techniques provide an improvement over ROS by synthesizing new points in the class with less number of samples using the available inputs as reference for the synthesis of new input samples [LNA17]. However, the general assumption in these works is that having an equal number of inputs for each of the classes ensures a balanced dataset, and in turn ensures an absence of bias [PHJ$^+$19, BL21]. As such, the approaches deploy data manipulation for the output class with a smaller number of inputs only. As observed in the motivating example earlier, this assumption provides a limited notion of balanced datasets. In addition, neither do these works have the means to ensure if the new inputs generated in fact belong to the minority class (i.e., output class with less number of inputs), nor the sophistication to analyze the number of inputs required to be added to the class to alleviate bias.

Other works explore heuristics to identify the inputs that must be removed from the training dataset [LLV18, LBSB$^+$20] for obtaining an unbiased DNN. However, for most real-world applications, large labeled datasets may not always be available, except to a few tech giants. This leaves limited scope for tasks relying on limited dataset for bias alleviation.

In summary, the DC approaches again focus on alleviating *representation* and *data bias*, i.e., the biases pertaining to faulty data acquisition and lack of data generalizing well to all output classes. Alleviation of *robustness bias* remains an unexplored research direction in the existing works. The notion of a balanced dataset often used in these works is too naive. For the approaches relying on the deletion of inputs from the training dataset, the approaches are ideal only for large datasets to ensure sufficient inputs remain for DNN training. For the augmentation approaches (like ROS, SMOTE and ADASYN), i.e., the approaches where synthetic inputs are added to the training dataset (henceforth referred to as *data augmentation*), the location for the new inputs is chosen to be in the close proximity around existing "randomly" selected inputs. The new inputs may or may not be realistic for the real-world input domain. The validation of these generated synthetic inputs relies solely on them being a part of DNN training, and how well the trained DNN works with the testing dataset.

However, most of these works [NCA$^+$20, GSSH20, LBSB$^+$20] encounter the following limitations, making robustness bias alleviation a challenging task:

1. Most works [LV19, LLV18, ZWY$^+$17] focus on either the dataset bias, i.e., the lack of generalization of the available dataset to real-world data, or representation bias, i.e., flaws in the dataset acquired during its collection process. However, they rarely focus on biases like robustness bias, which generally becomes evident only during DNN deployment, where noisy inputs are common.

2. A limited notion of balanced dataset is often used in literature [LNA17, BL21], i.e., a balanced dataset is the one that contains an equal number of inputs from all output classes. However, as seen from our motivational example, such a dataset does not necessarily aid in the alleviation of robustness bias.

3. They primarily focus on large datasets [KKK$^+$19, NCA$^+$20, GSSH20, ZWL$^+$19, LBSB$^+$20], which provide a large pool of training samples to learn the input features from as well as to handpick a subset of inputs that favor an unbiased DNN. However, such large datasets may not always be available.

4. Some works focus on adding new input samples to the training dataset or at deeper network layers [ZWL$^+$19]. However, the heuristics for adding new inputs do not always favor a balanced dataset.

5. The addition and deletion of input samples [BL21] may also lead to overfitting or reduction of the training dataset, respectively.

6. The works also often focus on visual datasets, like colored MNIST or the IMDB dataset, where the existence of bias is perceptually easy to detect and comprehend [ZWY$^+$17, WQK$^+$20]. However, the robustness bias problem may stretch beyond visual datasets, albeit often being difficult to (perceptually) detect in non-visual datasets.

To address the aforementioned limitations and challenges, this research proposes the UnbiasedNets framework, which facilitates the detection and reduction (ideally elimination) of bias in a trained DNN by addressing the bias at the root level, i.e., by reducing the bias within the training data, rather than relying on training algorithms to unlearn biases. Our framework is generic and hence can be implemented along with any training algorithm, using any programming language (including MATLAB, Python, C++, etc.). The novel contributions of the work are as follows:

1. This work deals with robustness bias, which results from having an imbalanced dataset (which may in turn be a consequence of either dataset bias or representation bias or both), to alleviate bias from datasets where the bias may not always be apparent in the absence of noisy inputs.
2. We redefine the notion of balanced dataset to provide a more precise explanation of the extent to which the number of inputs from each output class is, or is not, essential for training unbiased DNNs.
3. Unlike the state-of-the-art approaches, UnbiasedNets can work efficiently to diversify the dataset even in the absence of a large dataset using K-means clustering and the noise tolerance of a DNN previously trained on the dataset.
4. Our novel framework can identify the practical bounds for generating synthetic input samples using clusters of input features obtained via K-means and the noise tolerance bounds of the trained network. To the best of our knowledge, UnbiasedNets is the only framework exploiting noise tolerance to obtain realistic bounds for synthetic inputs. We also make use of feature correlation from real-world inputs to ensure that the synthesized inputs are realistic.
5. UnbiasedNets combines synthetic input generation with redundancy minimization to diversify and generate potentially balanced and equally-represented datasets, with not necessarily an equal number of inputs from all output classes.
6. The framework is applicable in diverse application scenarios. We demonstrate this using UnbiasedNets on two real-world datasets, where the bias in the dataset is not always visually detectable, and hence may not be straightforward to address.

6.2.1 UnbiasedNets: Framework for Bias Alleviation

We categorize UnbiasedNets into two major tasks: *bias detection* using a trained DNN to identify the existence of robustness bias followed by *bias alleviation* to diversify the training dataset to eliminate the bias at its core. Fig. 6.7 provides an overview of our proposed methodology.

Bias Detection

The first step here is the application of noise η, bounded by the small noise bounds Δx to the inputs present in the testing dataset $x \in X$ (shown as Block 0 in Fig. 6.7) to

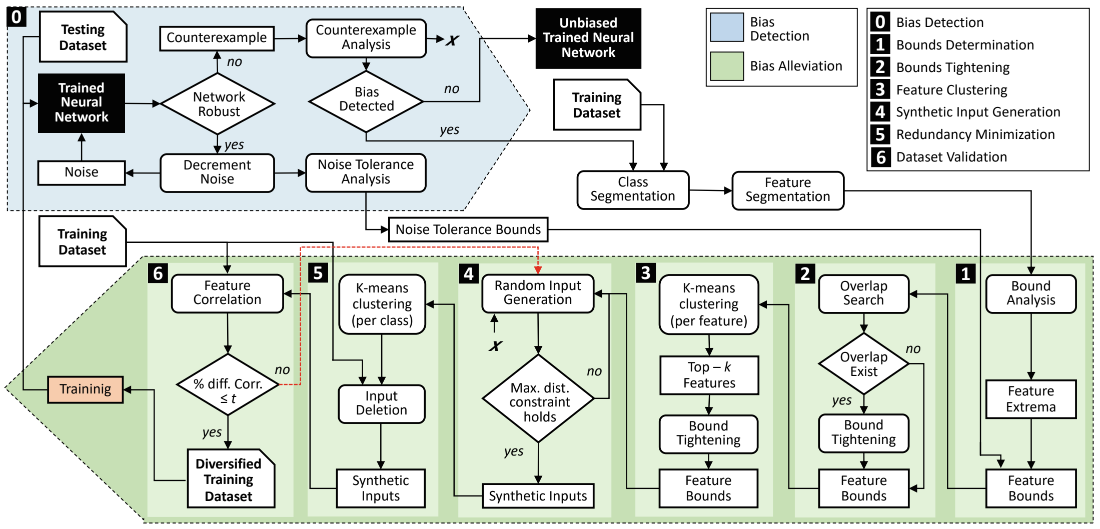


Figure 6.7: UnbiasedNets: methodological approach to training dataset diversification, starting with a trained DNN undergoing bias detection, followed by bias alleviation, ultimately leading to a diversified dataset and potentially unbiased trained DNN.

obtain the noisy inputs x_n.

$$x_n = x + \eta \quad s.t. \quad \eta \leq \Delta x \tag{6.1}$$

The noisy inputs are then supplied to the trained DNN, and their output classifications are compared to the classifications of inputs in the absence of noise. For the network to be robust, the DNN's classification must not change under the influence of noise. The noise is then iteratively increased, beyond the maximum noise at which the DNN does not misclassify the inputs, i.e., beyond the DNN's noise tolerance. Such iterative increment of noise provides the noise tolerance bounds of the network.

The application of noise larger than the noise tolerance bounds of the DNN entails that the DNN misclassifies some or all the noisy inputs. These misclassifying noise patterns (i.e., the counterexamples) act as inputs for the counterexample analysis. These noise patterns can be collected either using a formal framework (such as the ones based on model checking used by [tMK+20] and [BtSH22]) or an empirical approach (like the Fast Gradient Sign Method (FGSM) attack [GSS15]).

During counterexample analysis, the obtained counterexamples are used to quantify the bias. We define a metric $\mathcal{B}_R$ to achieve this bias quantification. Let R_i be the ratio of misclassified to correctly classified inputs from class i, which defines the average tendency of inputs from output class i to be misclassified. The robustness bias estimate $\mathcal{B}_R$ can then be represented as follows:

$$\mathcal{B}_R = max(abs(R_i - \frac{\sum_{j \in L \setminus i} R_j}{\mid L \mid -1}))$$

where L is the set of all output classes. This not only detects the presence, but also provides the severity of robustness bias in the trained DNN, i.e., having a $\mathcal{B}_R$ of zero indicates an equal R_i across all output classes, and therefore an unbiased DNN. Conversely, a non-zero $\mathcal{B}_R$ implies a robustness bias in the network.

Additionally, the number of misclassified inputs from each class is also used to determine the number of synthetic inputs required in the training dataset to alleviate the bias.

Bias Alleviation

Using the noise tolerance available from the bias detection and the feature extremum of the inputs from the training dataset, we provide the step-by-step bias alleviation methodology. The aim of the methodology is to identify the valid input domain for the generation of synthetic data and provide a diversified training dataset for the training of a potentially unbiased DNN. The details of each step in the methodology are as follows:

1. **Bounds Determination.** For each input feature in every output class, the feature extremum, i.e., the maximum and minimum value of the feature as per the available training data, is first identified (as shown in Block 1 of Fig. 6.7). As discussed earlier, the inputs with noise, less than the allowed noise tolerance, are still likely to be correctly classified by a trained DNN. Hence, the feature bounds are relaxed using Δx_{max}, to provide a larger input space for the diversified inputs (also shown in Fig. 6.8(a)), as follows:

Theorem 1 (Bound Relaxation using Noise Tolerance)**.** *For input domain X, let $[\underline{x_i}, \overline{x_i}]$ represent the bounds of inputs belonging to X_i (where $X_i \subset X$) and Δx_{max} be the noise tolerance of the network. From the definition of noise tolerance of noise tolerance, we know that the application of noise within the tolerance of the network does not change the output classification. Hence, more realistic input bounds $[\underline{x_i'}, \overline{x_i'}]$ can be obtained using the laws of interval arithmetic as:*

$$\underline{x_i'} = min((\underline{x_i} - \Delta x_{max}), (\underline{x_i} + \Delta x_{max}), (\overline{x_i} - \Delta x_{max}), (\overline{x_i} + \Delta x_{max})), \tag{6.2}$$

$$\overline{x_i'} = max((\underline{x_i} - \Delta x_{max}), (\underline{x_i} + \Delta x_{max}), (\overline{x_i} - \Delta x_{max}), (\overline{x_i} + \Delta x_{max})) \tag{6.3}$$

It must be noted that due to the scalability of underlying bias detection framework (for instance [tMK+20]), where the application of large noise to DNN inputs may lead to very large formal models, not suitable for analysis, noise tolerance may not always be available for bound relaxation. A similar challenge is encountered for DNNs with a very low noise tolerance. Consider the example of a DNN trained on an image dataset, where the addition of noise leading to a magnitude change of even 1.0 in the pixel value of an image may still lead to misclassification [MNG+21]. This indicates a very low noise tolerance. Under these conditions, UnbiasedNets assumes the noise tolerance to be zero, and proceeds with feature extremum as the feature bounds obtained during bound determination.

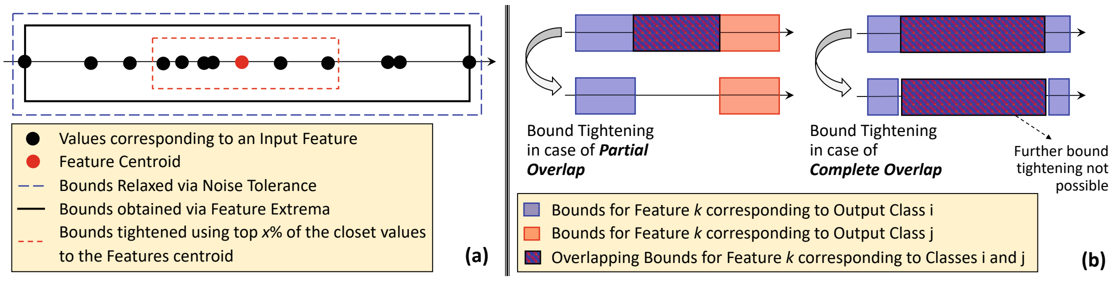

● Values corresponding to an Input Feature
● Feature Centroid
Bounds Relaxed via Noise Tolerance
Bounds obtained via Feature Extrema
Bounds tightened using top x% of the closet values to the Features centroid
(a)
Bound Tightening in case of Partial Overlap
Bound Tightening in case of Complete Overlap
Further bound tightening not possible
▨ Bounds for Feature k corresponding to Output Class i
▨ Bounds for Feature k corresponding to Output Class j
▨ Overlapping Bounds for Feature k corresponding to Classes i and j
(b)

Figure 6.8: (a) Realistic bounds determination for individual feature bounds using available training inputs, K-means clustering and noise tolerance, (b) Bound tightening to eliminate/reduce bound overlap for synthetic input generation.

2. **Bound Tightening.** Bounds obtained from the previous step identify the regions in the input space where real inputs from the training dataset exist, and hence provide an estimate for the generation of valid synthetic data. However, it is possible for the feature bounds for different output classes to overlap, as shown in Fig. 6.8(b). The overlap can be either *partial* or *complete*. This provides a means for tightening the feature bounds (shown as Block 2 in Fig. 6.7), hence leading to smaller, yet realistic, input space for the generation of synthetic data. This in turn ensures that a lesser number of iterations are required for realistic synthetic input generation in the later steps of the framework. The generation of tighter feature bounds in the case of partial feature can be seen as follows:

Theorem 2 (Bound Tightening in case of Partial Overlap)**.** *Given the bounds of input feature a for inputs belonging to class i and j to be* $[\underline{x_i^a}, \overline{x_i^a}]$ *and* $[\underline{x_j^a}, \overline{x_j^a}]$*, respectively, the bounds can be tightened to* $[\underline{x_i^a}, \underline{x_j^a}]$ *and* $[\overline{x_i^a}, \overline{x_j^a}]$ *provided that* $\underline{x_i^a} < \underline{x_j^a}$ *and* $\overline{x_i^a} < \overline{x_j^a}$ *(i.e., the bounds overlap partially). Then, any input belonging to the new bounds also belongs to the original feature bounds as well.*

$$\begin{aligned} \forall i,j.(([\underline{x_i^a}, \overline{x_i^a}] \in X_i^a \wedge [\underline{x_j^a}, \overline{x_j^a}] \in X_j^a) \implies ([\underline{x_i^a}, \underline{x_j^a}] \in X_i^a \wedge [\overline{x_i^a}, \overline{x_j^a}] \in X_j^a)) \\ s.t.\ \ \underline{x_i^a} < \underline{x_j^a} < \overline{x_i^a} < \overline{x_j^a} \end{aligned} \tag{6.4}$$

However, the same cannot be generalized for complete overlap since the bounds of one label form a subset of the other. As such, tightening is possible for a single label only.

Theorem 3 (Bound Tightening in case of Complete Overlap)**.** *Given the bounds of input feature a for inputs belonging to class i and j to be* $[\underline{x_i^a}, \overline{x_i^a}]$ *and* $[\underline{x_j^a}, \overline{x_j^a}]$*, respectively, the bounds for feature a of class i,* X_i^a*, can be tightened to* $[\underline{x_i^a}, \underline{x_j^a}]$ *and* $[\overline{x_j^a}, \overline{x_i^a}]$ *provided that* $\underline{x_i^a} < \underline{x_j^a}$ *and* $\overline{x_j^a} < \overline{x_i^a}$*. Then, any input belonging to the new bounds for* X_i^a *also belongs to the original feature bounds as well.*

$$\begin{aligned} \forall i,j.(([\underline{x_i^a}, \overline{x_i^a}] \in X_i^a \wedge [\underline{x_j^a}, \overline{x_j^a}] \in X_j^a) \implies ([\underline{x_i^a}, \underline{x_j^a}] \in X_i^a \wedge [\overline{x_j^a}, \overline{x_i^a}] \in X_i^a)) \\ s.t.\ \ \underline{x_i^a} < \underline{x_j^a} < \overline{x_j^a} < \overline{x_i^a} \end{aligned} \tag{6.5}$$

Motivating Example. Consider an arbitrary feature a with valid input values in the range $[0, 10]$. Let the inputs from class i have the bounds $[2, 8]$ and those from class j have the bounds $[7, 10]$, for the feature a. Without bound tightening, any input $7 < x^a < 8$ can belong to either class i or j (but not both). On the contrary, bound tightening reduces the bounds of the feature a for classes i and j to $[0, 7]$ and $[8, 10]$, respectively. This reduces the valid input domain for feature a such that it is impossible to pick a sample for feature a that may belong to more than a single output class, hence simplifying the task of generating realistic synthetic input samples.

3. **Feature Clustering.** The previous steps in the framework make use of the entire training dataset to obtain realistic feature bounds. But intuitively, real-world inputs often contain outliers that may be part of the training dataset, which do not occur frequently in practical case scenarios. To subsume this characteristic into the synthetic inputs generated, further bound tightening is carried out (shown as Block 3 in Fig. 6.7) on the *top-k* input features, i.e., the k features with the smallest distance from cluster centroid to the farthest input.

4. **Synthetic Input Generation.** Using the feature bounds obtained from the previous step, the random input values are chosen within the available bounds (shown as Block 4 in Fig. 6.7). The number of inputs to be added to each output class $\mathcal{Q}_i$ is determined on the basis of the ratio of percentage of misclassified inputs from class i (i.e., α_i) and the percentage of misclassified inputs from the class with minimum misclassifications (i.e., $min(\alpha_L)$) using counterexamples recorded during the bias detection. Hence, the class with higher α_i gets the most synthetic inputs added to the dataset.

Algorithm 4 outlines the entire synthetic data generation process, starting from the training dataset and noise tolerance bounds. Function `classSegment` (Line: 3) splits the dataset into non-overlapping subsets of inputs belonging to each class, `globalExt` (Line: 5) provides feature bounds using feature extremum, `nonOverlapping` (Line: 8) performs bound tightening on basis of Theorems 2 and 3, `minDist` (Line: 10) identifies the *top-k* features based on k-means clustering, `boundsFinal` (Line: 12) performs further bound tightening based on the top features, and `randInp` (Line: 15) finally generates the synthetic inputs for each output class.

It must be noted that the above input generation assumes an implicit hyperrectangular distribution of the input domain. This means, each input feature may take any input value (from within the defined input bounds), with equal likelihood. However, it is also possible for the input features to have non-rectangular distributions. Assuming these distributions to be known a priori, the random input generation could be modified to select input values, from within the input bounds, according to their probability of occurrence in their exact input distributions, i.e., with the more probable values having higher likelihood of selection and vice versa.

Algorithm 4 Synthetic Data Generation

Input: Training Inputs (X), Number of Output Classes ($\mathcal{C}$), Noise Tolerance (Δx_{max}), Number of top Features to use for Bound Tightening k, Number of Inputs to add to each Class ($\mathcal{Q}$)
Output: Augmented Input Matrix (X'), Vector of Output Classes (L')

```
1:  function SynthGen($X$, $\mathcal{C}$, $\Delta x_{max}$, $\mathcal{Q}$)
2:      $n$ = size($X$,2)                                              ▷ Number of Input Features
3:      $(X_1,...,X_{\mathcal{C}})$ = classSegment($X$,$\mathcal{C}$)
4:      for $i$ = 1:$\mathcal{C}$ do
5:          $(f\_min_i, f\_max_i)$ = globalExt($\Delta x_{max}$,$X_i$)                 ▷ Block 1 in Fig. 6.7
6:      end for
7:      for $j$ = 1:$n$ do
8:          $(f'\_min^j, f'\_max^j)$ = nonOverlappping($f\_min^j$ ,...,$f\_max^j$)      ▷ Block 2 in Fig. 6.7
9:      end for
10:     $(T_1,...,T_k)$ = minDist($X$)
11:     for $m = 1 : k$ do
12:         $(f''\_min^{T_m},...,f''\_max^{T_m})$ = boundsFinal($f'\_min^{T_m}$,$f'\_max^{T_m}$)   ▷ Block 3 in Fig. 6.7
13:     end for
14:     for $i$ = 1:$\mathcal{C}$ do
15:         $X_{new}$ = randInp($f''\_min_i$,...,$f''\_max_i$,$\mathcal{Q}_i$)           ▷ Block 4 in Fig. 6.7
16:     end for
17: end function
```

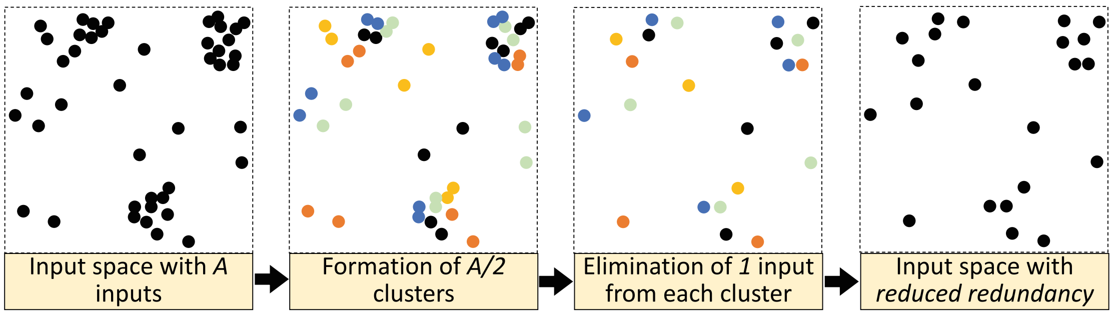

Input space with *A* inputs
Formation of *A/2* clusters
Elimination of *1* input from each cluster
Input space with *reduced redundancy*

Figure 6.9: Redundancy minimization by 50% in a two-dimensional input space.

5. **Redundancy Minimization** Oversampling may lead an DNN to overfit to the training samples. Moreover, the existence of similar inputs, after the addition of synthetic inputs, does not add to the diversity of the dataset. Existing works also indicate that training the DNNs on smaller datasets – for instance, those obtained by eliminating input instances leveraging different distance matrices – may reduce the timing overhead for training while providing comparable classification accuracy [KKP$^+$06, WTY09, Fua22].

Hence, x% closely resembling inputs from each class are removed to minimize the redundancy in the diversified training dataset (shown as Block 5 in Fig. 6.7). This is done by generating $\frac{1}{x}$ clusters for each output class and then retaining a single input from each cluster. The result is a dataset with input samples covering diverse input space, without densely populating any specific region of the input space (as realized in Fig. 6.9).

6. **Dataset Validation** Up until the previous step, UnbiasedNets used real-world inputs to identify valid input space within which the inputs exist, used knowledge of the percentage of misclassified inputs from each output class to identify the number of synthetic inputs to generate, and minimized the redundancy in the generated input samples to obtain a diversified dataset. However, features in the real-world data may be correlated, and the synthetic input features, despite lying in the valid input domain, may not follow the correlation of real-world data. Hence, this step aims to validate the synthetic inputs by comparing their feature correlation with that of the original training data. If the percentage difference between the correlation coefficients is within $t\%$, the new inputs are deemed suitable for training a potentially unbiased DNN. Otherwise, the process of synthetic data generation is repeated until the feature correlation of the synthetic inputs resembles that of the original training dataset (shown as Block 6 in Fig. 6.7).

The choice of t is made on the basis of the percentage difference between the correlation coefficients of training and testing datasets. However, if this difference is too large, the features may simply be independent, or obtaining appropriate correlations may require some input pre-processing [ZHG06]. The use of *only* simple Pearson correlation coefficient, on such raw data, may not be an appropriate statistical measure to ensure the synthetic inputs to be realistic here.

6.2.2 Experiments

This section describes our experimental setup, and details of DNNs and datasets used in our experiments.

Experimental Setup

All experiments were carried out on CentOS-7 system running on a 3.1GHz 6 core Intel i5-8600. Our UnbiasedNets framework was implemented on MATLAB. The DNN training was carried out using Keras.

However, the setup did not make use of any special libraries and, hence, can be easily re-implemented using any programming language(s). Bias detection (and counterexample generation) was carried out using SMV models with applied noise in the range of $1-40\%$ of the actual input values, using a timeout of 5 minutes for each input.

6.2.3 Datasets and Neural Network Architecture

We experimented on the long-tailed leukemia dataset [GST+99] composed of the genetic attributes of leukemia patients classified between Acute Lymphoblast Leukemia (ALL) and Acute Myeloid Leukemia (AML). The training dataset consists of 38 input samples (with 27 and 11 inputs indicating ALL and AML, respectively), while the testing dataset contains 34 inputs (with 20 and 14 ALL and AML inputs, respectively). We trained a single hidden layer (20 nodes), fully-connected ReLU-based DNN, using the top-5 most essential genetic features from the dataset extracted using Minimum Redundancy and Maximum Relevance (mRMR) feature selection technique [Kea18]. A learning rate of 0.5 for 40 epochs followed by another 40 epochs with a learning rate of 0.2 were used during training.

We also experimented on the Iris dataset [Fis36, DG17b], which is a multi-label dataset, with characteristics of three iris plant categories as input features. The dataset has an equal number of inputs from all output classes. We split the dataset into training and testing datasets, with 120 and 30 inputs, respectively, while ensuring an equal number of inputs from all classes in each dataset. A fully-connected ReLU-based two-hidden layer (15 nodes each) DNN was trained with a learning rate of 0.001 for 80 epochs, using a training to validation split of $4 : 1$.

Since UnbiasedNets is a data-centric bias alleviation framework, we compare the framework to well-acknowledged open-source state-of-the-art data-centric approaches: RUS, ROS, SMOTE [Cea02] and ADASYN [Hea08]. The Python toolbox `imbalanced-learn` implements all of the aforementioned techniques, except RUS, and was used for the generation of testing datasets. Since these approaches require the number of inputs to be different in each class, 50% of the inputs from the Iris dataset were randomly selected to create a sub-dataset with an unequal number of inputs for the classes. RUS was implemented on MATLAB, removing inputs from class with more inputs to ensure both classes have the same number of inputs in the case of the leukemia dataset and removing 25% samples from each class in the case of the Iris dataset. To avoid overfitting during retraining of DNNs using augmented datasets, the number of training epochs was reduced proportionally to the increase in the size of datasets.

All DNNs considered in the experiments were trained to the training and testing accuracies of over 90%. In addition, the experiments for each bias alleviation approach were repeated 10 times to ensure conformity.

6.2.4 Results and Analysis

This section elaborates on the empirical results obtained from our experiments followed by comparison and analysis of UnbiasedNets to the data-centric bias alleviation approaches.

6.2.5 Observations

As the number of output classes increases, ensuring an unbiased DNN becomes a more challenging task. This was clearly observed in our experiments (Table 6.3), wherein the multi-label classifiers had a higher bias and at the same time, their bias reduction was substantially less effective in all bias alleviation approaches. (Note that the table represents the bias of the network trained on original dataset is given in bold, and that of the network trained diversified dataset is given in bold italics.)

As discussed earlier, lower $\mathcal{B}_R$ indicates that the difference in the ratio of misclassified to correctly classified inputs is low, implying that the DNN is less biased towards any output

Table 6.3: Comparison of $\mathcal{B}_R$ values (average ± standard deviation) obtained for the DNNs trained on original and diversified datasets, using open-source state-of-the-art approaches and UnbiasedNets.

Datasets / Approach	Robustness Bias ($\mathcal{B}_R$)	
	Leukemia Dataset	Iris Dataset
Original	**0.2228**	**0.4732**
RUS	0.1710 ± 0.07	0.5042 ± 0.11
ROS	0.2213 ± 0.07	0.8059 ± 0.36
SMOTE	0.1452 ± 0.08	0.7709 ± 0.72
ADASYN	0.2434 ± 0.06	*ADASYN not suited for dataset*
UnbiasedNets	**0.1236 ± 0.05**	**0.4906 ± 0.15**

class. As summarized in Table 6.3, our UnbiasedNets framework outperformed all the DC bias alleviation techniques while obtaining optimum $\mathcal{B}_R$ values for both binary and multi-label datasets. Moreover, in the case of the Iris dataset, using classical data-centric approaches to generate dataset with an equal number of inputs from each class seems to exacerbate the robustness bias. Although UnbiasedNets may not always reduce the robustness bias, the data diversification ensures that the dataset remains balanced.

This success of biased can also be seen in Fig. 6.10, which shows the variation in $\mathcal{B}_R$ values over the repeated experiments. It is clearly evident that the individual experiments leading to a decrease in average robustness bias are far more compared to vice versa. Hence, we advocate executing several instances of experiments in order to obtain dataset instances that offer the best bias alleviation.

Additionally, it can be seen from the box plots that DNNs trained using the UnbiasedNets datasets demonstrate considerably low interquartile ranges and the lowest average $\mathcal{B}_R$ values. Even though RUS illustrates competitive $\mathcal{B}_R$ values, the use of RUS is not appropriate for small datasets, since the approach involves the deletion of real input samples and may hence diminish the learning capability of the DNN. The remaining approaches, i.e., ROS, SMOTE, and ADASYN, present a large variation in $\mathcal{B}_R$ results, deeming the approaches less effective for alleviation of robustness bias.

Analysis

Our work focuses on robustness bias, which is exhibited by a trained DNN in the presence of inputs having higher robustness to noise for certain output classes as compared to others. From our experiments, we confirm the hypothesis that having an equal number of inputs (as in the case of Iris dataset) is in fact *insufficient* to ensure an unbiased network.

In the case of the datasets where the number of inputs in each class is different, the known approaches like RUS, ROS, SMOTE, and ADASYN may reduce the bias. But

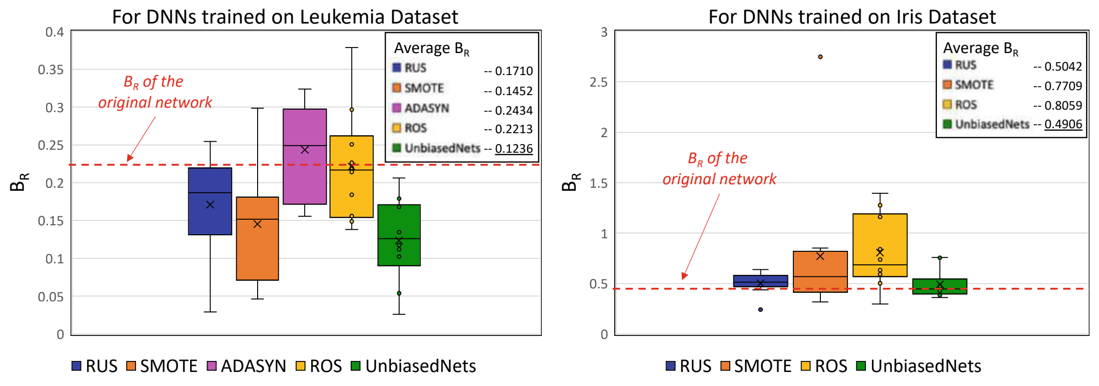


Figure 6.10: Variation in $\mathcal{B}_R$ results for DNNs trained on RUS, SMOTE, ADASYN, ROS, and the diversified UnbiasedNets datasets.

for most datasets, they may be inadequate for robustness bias alleviation mainly for two reasons: (1) they rely on the naive definition of balanced datasets and only ensure the number of inputs for each class is equal, which overlooks the requirement of each class to be *equally-represented* in the input[3], and (2) during data augmentation, new inputs are only added in between the existing inputs, which neither diversifies the dataset sufficiently nor ensures that the new inputs are valid candidates for the augmented dataset. UnbiasedNets, on other hand, uses counterexample analysis from the bias detection stage to obtain the required number of inputs in each class for a potentially *equally-represented* dataset. It also uses noise tolerance, which allows us to diversify the data beyond the bounds of the existing training dataset, which is subsequently validated by leveraging feature correlations, to alleviate bias in DNN.

In the case of the Iris dataset, ROS and SMOTE were observed to significantly worsen robustness bias. This may be partially due to the deletion of inputs from the dataset to create an unequal number of inputs in the classes, which reduces the data available for DNN training. However, RUS retained the $\mathcal{B}_R$ value close to the original dataset, even though the approach also employs input deletion. This suggests that the data augmentation by ROS and SMOTE may actually contribute to an exacerbation of bias rather than alleviation. In the case of UnbiasedNets, even though the improvement in bias is often small, the results clearly suggest that diversifying the training dataset by adding realistic synthetic inputs and reducing redundancy in dataset is a potential direction to alleviate bias in DNNs, unlike the other approaches.

6.2.6 Discussion

UnbiasedNets aims to diversify the dataset so as to (potentially) achieve a balanced dataset. While the diversification goal for obtaining a completely unbiased network

[3]A dataset with each class equally-represented potentially leads a balanced dataset (as described in Chap: 2).

may not always be achieved, UnbiasedNets rarely aggravates the bias due to its precise perception of balanced datasets, unlike existing DC techniques. This section discusses the various aspects of DNNs, which contribute to the challenge of data diversification and ultimately the persisting bias in trained networks.

Input Resemblance

As seen from Table 6.3, the greater the number of output classes, the higher the robustness bias in the DNN. This implies that the higher the number of output classes, the more likely is the dataset imbalanced, and the more unlikely it is to obtain a trained DNN that is equally robust for all output classes. A likely explanation for this could in fact be a close resemblance of inputs from the different classes, for datasets with a higher number of output classes.

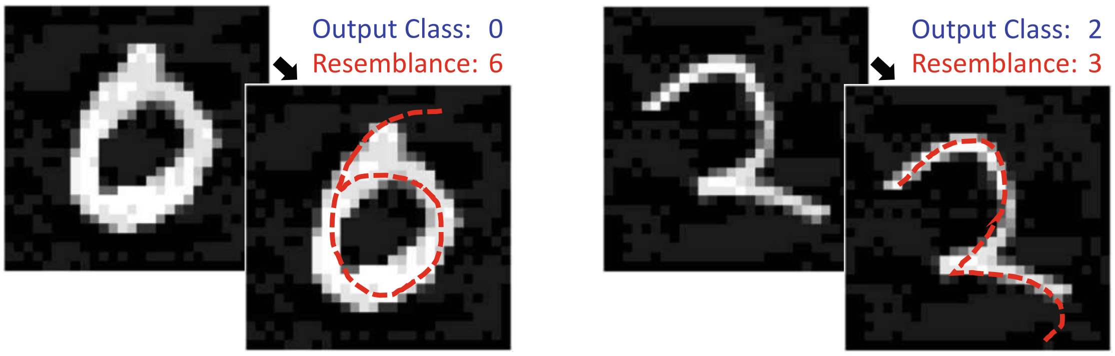


Figure 6.11: Inputs from one output class may resemble inputs from other classes, as observed in the MNIST dataset.

For instance, consider the case of hand-written digits (from the MNIST dataset), which comprises of 10 output classes. As shown in Fig. 6.11, it is possible for inputs from some classes to closely resemble inputs from other classes – for example, digit 0 may resemble a 6, and digit 2 may resemble a 3. With inputs having likely resemblance to multiple classes, it is challenging to generate realistic synthetic inputs, and hence obtain successful data diversification for reducing the bias.

A more careful study of the example provided above also reveals that the difference between the closely resembling inputs blur when their semantic distance is smaller [Ken19], as shown in Fig. 6.11. Yet, the syntactic rules for output classification stay intact even for these closely resembling inputs. For instance, a single loop forms the digit 0, while an arc of a length comparable to half the circumference of the loop is required in addition to the loop to syntactically define the digit 6. Hence, the addition of such syntactic rules for the generation of synthetic inputs (similar to the approach taken in neuro-symbolic learning [SZEH21]) may improve the data diversification.

Curse of Dimensionality

Another challenge to data diversification is the large number of input nodes comprising the DNN inputs – a challenge often referred to as the "curse of dimensionality" in the DNN analysis literature [WOZ+20]. This implies that as the number of input nodes for the DNN increase, the computational requirements for its analysis increase exponentially.

To understand this from the perspective of data diversification, let us consider the example of an image dataset. Data diversification determines input feature bounds directly from the raw input data to generate inputs such that the synthesized inputs x belong to the valid input $\mathcal{X}$, i.e., $x \in \mathcal{X}$. However, various transformations, like affine, homographic and photometric transforms associated with image inputs may tremendously change the inputs, while still keeping the inputs realistic [PCYJ17]. Hence, for a practical image dataset, inputs belonging to even a single output class will have individual inputs that have undergone different transformations. As a result, the bounds of each input feature obtained from the inputs, for such a dataset, will be very large. This hinders the generation of synthetic data using these bounds, in turn making the data diversification halt at the data validation step since the search input space is too large for the randomly generated inputs to be realistic.

Towards this end, appropriate input pre-processing and the use of feature correlation knowledge to determine the bounds of the correlated input features (rather than raw input features) could potentially extend the applicability of UnbiasedNets framework to a larger variety of datasets.

6.3 Summary of Efforts towards DNN Robustification

The overall performance of DNNs, particularly those relying on supervised training algorithms, is largely dependent on the training data available. However, the data used to train these DNNs may often be biased towards specific output class(es) or have different sensitivity for the input nodes for individual output class(es). This may propagate not only as varying classification performance across different output classes, but also robustness bias and node robustness bias in the trained DNN. As highlighted by the works in this chapter, identifying and correcting such vulnerabilities in DNNs is not a straightforward task.

Towards this, we leverage our (qualitative) formal analysis framework to explore possible link between the variance of node values in the training dataset and the impact of removing samples of head class(es) from the training dataset prior to training. We also proposed a novel bias alleviation framework UnbiasedNets, which initially detects and quantifies the extent of bias in a trained DNN and then uses a methodological approach to diversify the training datasets by leveraging the DNN's noise tolerance and K-means clustering. To the best of our knowledge, this is the first framework specifically addressing the *robustness bias* problem.

The analysis and robustification attempts are supported by experiments using binary and multi-label classifiers. We also discuss the challenges in robustness bias alleviation and elaborate on the potential future research direction for addressing the vulnerabilities resulting from long-tail distribution in trained DNNs. The next chapter proposes other future directions to leverage formal analysis frameworks to ensure better, more robust DNNs for practical applications.

CHAPTER 7

Conclusion and Future Outlook

7.1 Thesis Summary

Machine Learning (ML) has been finding more and more use cases in real-world applications for the past several decades. However, deep neural networks (DNNs), which often form the core component of such ML-based systems are data-driven models. Hence, at best, the performance of these models can only be as good as the data fed to them for training. This is reflected in the numerous works focusing on identifying the vulnerabilities of trained DNNs where DNNs are shown to delineate incorrect behavior under different input scenarios. With this increasing reliance of ML-based system on trained DNNs, and the vulnerabilities of such models, there is a need to provide behavioral guarantees for DNNs to ensure reliable ML operations.

Among the efforts to identify the vulnerabilities include the *empirical analysis*. These had been fairly successful in identifying DNN vulnerabilities, particularly that to the adversarial noise. However, fundamentally, these approaches can merely provide evidence to DNN vulnerability, but not provide any guarantees in case such evidence is not found. Hence, *formal verification* forms the alternative research direction to overcome this limitation. However, formal verification comes with it's own challenge, i.e., the scalability. Added to this challenge is the limited scope of current formal verification of DNNs, both in terms of the DNN vulnerabilities addressed and the types of verification strategies explored. Toward this important goal of providing behavioral guarantees for trained DNNs, this thesis focuses on *formal analysis*, which combines the strengths of both empirical analysis and formal verification. The research spans to provide both *qualitative* and *quantitative* behavioral guarantees, while focusing on a diverse range of DNN vulnerabilities.

On the qualitative end, we propose the novel model checking-based framework FANNet+, which challenges the limits of model checking for DNN analysis while catering to a diverse range of DNN properties namely, robustness, noise tolerance, input node sensitivity, robustness bias, and safety. To address the scalability challenge, numerous model reduction techniques are used without compromising the completeness of DNN formal models. We also develop a novel random input segmentation (RIS) approach to scale the valid input domain for analysis. Although RIS introduces incompleteness in the formal analysis, unlike the incompleteness often observed in existing literature, this incompleteness of the input domain does not lead to false positives in analysis results.

Towards the quantitative front, we develop QuanDA, a novel sampling-based approach to estimate precise probabilities of DNNs' behavior. QuanDA allows the propagation of probability distribution across DNN layers, hence avoiding the use of unrealistic assumptions on the probability distributions of network nodes. Hoeffding's inequality is also leveraged by the framework to provide a confidence interval for the probability estimates, while focusing on DNN reachability and safety properties. This leads to quantitative DNN analysis that outperforms the existing state of the art in terms of both scalability and providing realistic analysis.

For both frameworks, we show the efficacy of analysis using DNNs trained on real-world data. The resulting DNNs correspond to critical applications namely, leukemia diagnosis, heart disease prognosis, and aircraft collision avoidance.

Once the DNN vulnerabilities are identified, the next natural direction is to rectify the vulnerabilities. Although this was not the prime focus of the present research, we still provide some preliminary research directions toward developing more robust DNNs. Our efforts are directed specifically to deal with vulnerabilities resulting from long-tail distribution in the training dataset. Towards this, we develop UnbiasedNets to diversify the training dataset, and potentially minimize the robustness bias. The framework shows some success compared to available data-centric approaches for addressing the robustness bias.

7.2 Future Outlook

This research focused on two directions for DNN analysis: model checking for the qualitative analysis and sampling-based framework for the quantitative analysis. While both directions cater to important research goals, there remain open challenges for the future research. These include the ongoing scalability challenge, particularly for the model checking-based framework. While the model reduction and input segmentation approaches developed in this research stretch the limits for model checking for DNN analysis, the model checking-based analysis for practically-sized networks remains an open challenge. Specific to the quantitative analysis is the open challenge of propagating the confidence interval over layers. While our framework provides intricate methodology for the propagation of probability distribution over network layers, the theoretical confidence interval is currently available only on a per-layer basis.

Additionally, while the present research focused on the development of DNN analysis methodologies catering a diverse range of DNN behavioral specifications, it did not explore the implementation of proposed frameworks on DNN architectures beyond fully-connected neural networks. The analysis of DNN architectures like convolutional and residual networks would be interesting future directions for the research community. We also identified node robustness bias in trained DNNs, i.e., the individual input nodes of the DNN may have different robustness for the different output classes. This opens potential research direction towards the interpretability of trained DNNs at the node level.

It is interesting to note that most of the DNN vulnerabilities explored in this research as well as existing literature likely result from the data-dependent nature of the DNN models. While DNN behavioral analysis is an important step to ensure its reliable operation in real world, it might also be fruitful to address DNN behavioral issues at an earlier stage in the DNN design cycle. For instance, the addition of behavioral specifications commonly checked post-training (explored in Chap. 3) could potentially be included in the DNN training loop as network constraints or additional cost function, for robust DNN training.

This page is intentionally left blank.

List of Figures

List of Tables

List of Algorithms

Glossary

complete	a proof system where there exists a formal proof for every property expressed in the system.. 13
Completeness	a proof system is said to be complete if there exists a formal proof for every true property expressed in the proof system [HR04].. 39, 42
fully-connected neural network	neural network where all nodes in each network layer are connected to all nodes of the adjacent layer. 34
long-tail distribution	distribution with significantly larger portion of samples belonging to head class(es) as opposed to the tail class(es). 4
quantized neural network	a neural network with fixed bit-width parameter sizes.. 13
shallow (network)	neural network comprising of a single (or few) hidden layers. 34
Soundness	a proof system is said to be sound if everything that is provable in the system is in fact true [HR04].. 42

Acronyms

ADASYN	adaptive synthetic sampling. 84, 85, 93, 94
AP	Atomic Proposition. 32, 37
BaB	branch and bound. 18
BDD	Binary Decision Diagram. 20
BNN	Binary Neural Network. 6, 15
CEGAR	Counterexample Guided Abstraction Refinement. 16
CNF	Conjunctive Normal Form. 15, 30
DNN	Deep Neural Network. 3, 5, 11, 13, 23, 24, 29, 55, 75, 97
LP	Linear Programming. 6, 17, 29
ML	Machine Learning. 3–5, 15, 75
OBDD	Ordered Binary Decision Diagram. 17, 20
ReLU	Rectified Linear Unit. 16, 17, 25
RIS	Random Input Segmentation. 40
ROS	random oversampling. 84, 85, 93–95
RUS	random undersampling. 84, 93–95
SAT	satisfiable. 15, 17, 20, 38
SAT	Satisfiability. 6, 15, 29, 30
SMC	Symbolic Model Checking. 37, 39, 40

SMOTE	synthetic minority oversampling. 84, 85, 93–95
SMT	Satisfiability Modulo Theories. 6, 15
SNN	Spiking Neural Network. 75
UNSAT	unsatisfiable. 15, 17, 20, 38
WSM	Weighted Sum Model. 63, 65

Bibliography

[ABC+02] Gilles Audemard, Piergiorgio Bertoli, Alessandro Cimatti, Artur Korniłowicz, and Roberto Sebastiani. A sat based approach for solving formulas over boolean and linear mathematical propositions. In *International Conference on Automated Deduction*, pages 195–210. Springer, 2002.

[ABGLP19] Martin Arjovsky, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz. Invariant risk minimization. *arXiv preprint arXiv:1907.02893*, 2019.

[AHKM20] Pranav Ashok, Vahid Hashemi, Jan Křetínskỳ, and Stefanie Mohr. Deepabstract: neural network abstraction for accelerating verification. In *International Symposium on Automated Technology for Verification and Analysis*, pages 92–107. Springer, 2020.

[ALMP18] Michael Akintunde, Alessio Lomuscio, Lalit Maganti, and Edoardo Pirovano. Reachability analysis for neural agent-environment systems. In *International Conference on Principles of Knowledge Representation and Reasoning*, 2018.

[And64] Norman H Anderson. Note on weighted sum and linear operator models. *Psychonomic Science*, 1(1):189–190, 1964.

[APDC19] Greg Anderson, Shankara Pailoor, Isil Dillig, and Swarat Chaudhuri. Optimization and abstraction: a synergistic approach for analyzing neural network robustness. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 731–744, 2019.

[AZN18] Mohsan Alvi, Andrew Zisserman, and Christoffer Nellåker. Turning a blind eye: Explicit removal of biases and variation from deep neural network embeddings. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, pages 0–0, 2018.

[B18] Wieland Brendel *, Jonas Rauber *, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. In *International Conference on Learning Representations*, 2018.

[BA21] Pedro RAS Bassi and Romis Attux. A deep convolutional neural network for covid-19 detection using chest X-rays. *Res. on Biomed. Engineering*, pages 1–10, 2021.

[BCC$^+$09] Armin Biere, Alessandro Cimatti, Edmund M Clarke, Ofer Strichman, and Yunshan Zhu. Bounded model checking. *Handbook of satisfiability*, 185(99):457–481, 2009.

[BCMS21] Teodora Baluta, Zheng Leong Chua, Kuldeep S Meel, and Prateek Saxena. Scalable quantitative verification for deep neural networks. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 312–323. IEEE, 2021.

[BDTD$^+$16] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[Bea16] Marco Bozzano et al. *nuXmv 1.1. 1 User Manual*, 2016.

[BGBM23] Soham Banerjee, Sumana Ghosh, Ansuman Banerjee, and Swarup K Mohalik. Smt-based modeling and verification of spiking neural networks: A case study. In *Verification, Model Checking, and Abstract Interpretation: 24th International Conference, VMCAI 2023, Boston, MA, USA, January 16–17, 2023, Proceedings*, pages 25–43. Springer, 2023.

[BIL$^+$16] Osbert Bastani, Yani Ioannou, Leonidas Lampropoulos, Dimitrios Vytiniotis, Aditya Nori, and Antonio Criminisi. Measuring neural net robustness with constraints. In *Advances in neural information processing systems*, pages 2613–2621, 2016.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT press, 2008.

[BKK$^+$20] Elena Botoeva, Panagiotis Kouvaros, Jan Kronqvist, Alessio Lomuscio, and Ruth Misener. Efficient Verification of ReLU-based Neural Networks via Dependency Analysis. In *AAAI Conference on Artificial Intelligence*, 2020.

[BL21] Sikha Bagui and Kunqi Li. Resampling imbalanced data for network intrusion detection datasets. *Journal of Big Data*, 8(1):1–41, 2021.

[BLT$^+$20] Rudy Bunel, Jingyue Lu, Ilker Turkaslan, P Kohli, P Torr, and P Mudigonda. Branch and bound for piecewise linear neural network verification. *Journal of Machine Learning Research*, 21(2020), 2020.

[BMB$^+$23] Christopher Brix, Mark Niklas Müller, Stanley Bak, Taylor T Johnson, and Changliu Liu. First three years of the international verification of neural networks competition (vnn-comp). *arXiv preprint arXiv:2301.05815*, 2023.

[BPV17] Igor Buzhinsky, Antti Pakonen, and Valeriy Vyatkin. Explicit-state and symbolic model checking of nuclear i&c systems: A comparison. In *Proc. IECON*, pages 5439–5446. IEEE, 2017.

[Bra93] Daniel Brand. Verification of large synthesized designs. In *Proceedings of 1993 International Conference on Computer Aided Design (ICCAD)*, pages 534–537. IEEE, 1993.

[Bro19] J Browniee. 14 different types of learning in machine learning. `https://machinelearningmastery.com/types-of-learning-in-machine-learning/`, 2019. Accessed: 2023-07-24.

[BS23] Achim D Brucker and Amy Stell. Verifying feedforward neural networks for classification in isabelle/hol. *Formal Methods (FM 2023)*, 2023.

[BSS$^+$19] Teodora Baluta, Shiqi Shen, Shweta Shinde, Kuldeep S Meel, and Prateek Saxena. Quantitative verification of neural networks and its security applications. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1249–1264, 2019.

[BtSH22] Iram Tariq Bhatti, **Mahum Naseer**, Muhammad Shafique, and Osman Hasan. A Formal Approach to Identifying the Impact of Noise on Neural Networks. *Communications of the ACM*, 65(11):70–73, 2022. *[Iram Tariq Bhatti and Mahum Naseer contributed equally to the work]*.

[BTT$^+$18] Rudy R Bunel, Ilker Turkaslan, Philip Torr, Pushmeet Kohli, and Pawan K Mudigonda. A unified view of piecewise linear neural network verification. In *Advances in Neural Information Processing Systems*, pages 4790–4799, 2018.

[BZKK21] Philipp Benz, Chaoning Zhang, Adil Karjauv, and In So Kweon. Robustness may be at odds with fairness: An empirical study on class-wise accuracy. In *NeurIPS 2020 Workshop on Pre-registration in Machine Learning*, pages 325–342. PMLR, 2021.

[BZSL19] Osbert Bastani, Xin Zhang, and Armando Solar-Lezama. Probabilistic verification of fairness properties via concentration. *Proceedings of the ACM on Programming Languages*, 3(OOPSLA):1–27, 2019.

[Car00] Neal L Carothers. *Real analysis.* Cambridge University Press, 2000.

[Cea02] N. V. Chawla et al. Smote: synthetic minority over-sampling technique. ***JAIR***, 16:321–357, 2002.

[CFGP20] Hayes Converse, Antonio Filieri, Divya Gopinath, and Corina S Păsăreanu. Probabilistic symbolic analysis of neural networks. In *2020 IEEE 31st*

International Symposium on Software Reliability Engineering (ISSRE), pages 148–159. IEEE, 2020.

[CGJ+03] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement for symbolic model checking. *Journal of the ACM (JACM)*, 50(5):752–794, 2003.

[CKS05] Byron Cook, Daniel Kroening, and Natasha Sharygina. Symbolic model checking for asynchronous boolean programs. In *Proc. SPIN*, pages 75–90. Springer, 2005.

[CNHR18] Chih-Hong Cheng, Georg Nührenberg, Chung-Hao Huang, and Harald Ruess. Verification of binarized neural networks via inter-neuron factoring. In *Working Conference on Verified Software: Theories, Tools, and Experiments*, pages 279–290. Springer, 2018.

[cpl] IBM ILOG CPLEX Optimization Studio. `https://www.ibm.com/analytics/cplex-optimizer`.

[CRF+20] Si Chen, Yaxing Ren, Daniel Friedrich, Zhibin Yu, and James Yu. Sensitivity analysis to reduce duplicated features in ann training for district heat demand prediction. *Energy and AI*, 2:100028, 2020.

[CSSD17] Arthur Choi, Weijia Shi, Andy Shih, and Adnan Darwiche. Compiling neural networks into tractable boolean circuits. *intelligence*, 2017.

[CW96] Edmund M Clarke and Jeannette M Wing. Formal methods: State of the art and future directions. *Computing Surveys (CSUR)*, 28(4):626–643, 1996.

[CW17] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *Symposium on Security and Privacy (SP)*, pages 39–57. IEEE, 2017.

[DG17a] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[DG17b] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[DJKV17] Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. A storm is coming: A modern probabilistic model checker. In *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II 30*, pages 592–600. Springer, 2017.

[DJST18] Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. Output range analysis for deep feedforward neural networks. In *NASA Formal Methods Symposium*, pages 121–138. Springer, 2018.

[EGK20] Yizhak Yisrael Elboher, Justin Gottschlich, and Guy Katz. An abstraction-based framework for neural network verification. In *International Conference on Computer Aided Verification*, pages 43–65. Springer, 2020.

[Ehl17] Ruediger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *International Symposium on Automated Technology for Verification and Analysis*, pages 269–286. Springer, 2017.

[EP02] Cindy Eisner and Doron Peled. Comparing symbolic and explicit model checking of a software system. In *Proc. SPIN*, pages 230–239. Springer, 2002.

[ERR$^+$19] Andre Esteva, Alexandre Robicquet, Bharath Ramsundar, Volodymyr Kuleshov, Mark DePristo, Katherine Chou, Claire Cui, Greg Corrado, Sebastian Thrun, and Jeff Dean. A guide to deep learning in healthcare. *Nature Medicine*, 25(1):24, 2019.

[Fea22] Y. Fu et al. Long-tailed visual recognition with deep models: A methodological survey and evaluation. ***Neurocomputing***, 2022.

[FH07] Martin Fränzle and Christian Herde. Hysat: An efficient proof engine for bounded model checking of hybrid systems. *Formal Methods in System Design*, 30(3):179–198, 2007.

[Fis36] Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188, 1936.

[FJ18] Matteo Fischetti and Jason Jo. Deep neural networks and mixed integer linear optimization. *Constraints*, 23(3):296–309, 2018.

[FLES19] Maximilian Fink, Ying Liu, Armin Engstle, and Stefan-Alexander Schneider. Deep learning-based multi-scale multi-object detection and classification for autonomous driving. In *Fahrerassistenzsysteme 2018*, pages 233–242. Springer, 2019.

[FMP19] Mahyar Fazlyab, Manfred Morari, and George J Pappas. Probabilistic verification and reachability analysis of neural networks via semidefinite programming. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pages 2726–2731. IEEE, 2019.

[FMP20] Mahyar Fazlyab, Manfred Morari, and George J Pappas. Safety verification and robustness analysis of neural networks via quadratic constraints and semidefinite programming. *IEEE Transactions on Automatic Control*, 67(1):1–15, 2020.

[Fua22] Piyabute Fuangkhon. Effect of the distance functions on the distance-based instance selection for the feed-forward neural network. *Evolutionary Intelligence*, 15(3):1991–2015, 2022.

[GBM21] Bishwamittra Ghosh, Debabrota Basu, and Kuldeep S Meel. Justicia: a stochastic sat approach to formally verify fairness. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 7554–7563, 2021.

[GKPB18] Divya Gopinath, Guy Katz, Corina S Păsăreanu, and Clark Barrett. Deepsafe: A data-driven approach for assessing robustness of neural networks. In *International symposium on automated technology for verification and analysis*, pages 3–19. Springer, 2018.

[GLDGG19] Tianyu Gu, Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. BadNets: Evaluating Backdooring Attacks on Deep Neural Networks. *IEEE Access*, 7:47230–47244, 2019.

[GMDC$^+$18] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation. In *Symposium on Security and Privacy (SP)*, pages 3–18. IEEE, 2018.

[GP18] Yuan Gong and Christian Poellabauer. An overview of vulnerabilities of voice controlled systems. *arXiv preprint arXiv:1803.09156*, 2018.

[Gro02] Ignacio E Grossmann. Review of nonlinear mixed-integer and disjunctive programming techniques. *Optimization and engineering*, 3(3):227–252, 2002.

[GSK$^+$11] Vehbi C Gungor, Dilan Sahin, Taskin Kocak, Salih Ergut, Concettina Buccella, Carlo Cecati, and Gerhard P Hancke. Smart grid technologies: Communication technologies and standards. *Transactions on Industrial informatics*, 7(4):529–539, 2011.

[GSS15] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and Harnessing Adversarial Examples. In *International Conference on Learning Representations (ICLR)*, 2015.

[GSSH20] Itai Gat, Idan Schwartz, Alexander Schwing, and Tamir Hazan. Removing bias in multi-modal classifiers: Regularization by maximizing functional entropies. In *Advances in Neural Information Processing Systems*, volume 33, pages 3197–3208, 2020.

[GST$^+$99] Todd R Golub, Donna K Slonim, Pablo Tamayo, Christine Huard, Michelle Gaasenbeek, Jill P Mesirov, Hilary Coller, Mignon L Loh, James R Downing, Mark A Caligiuri, et al. Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. *science*, 286(5439):531–537, 1999.

[Gur] Gurobi Optimizer. `https://www.gurobi.com/`.

[HCS+16] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. *Advances in neural information processing systems*, 29, 2016.

[HDB20] Thomas Hellström, Virginia Dignum, and Suna Bensch. Bias in machine learning–what is it good for? *arXiv preprint arXiv:2004.00686*, 2020.

[HDY+12] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Brian Kingsbury, et al. Deep neural networks for acoustic modeling in speech recognition. *Signal processing magazine*, 29, 2012.

[Hea08] H. He et al. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In ***Proc. IJCNN***, pages 1322–1328. IEEE, 2008.

[HJVE01] Timothy Hickey, Qun Ju, and Maarten H Van Emden. Interval arithmetic: From principles to implementation. *Journal of the ACM (JACM)*, 48(5):1038–1068, 2001.

[HKHF00] Andrew Hunter, Lee Kennedy, Jenny Henry, and Ian Ferguson. Application of neural networks and sensitivity analysis to improved prediction of trauma survival. *Computer methods and programs in biomedicine*, 62(1):11–19, 2000.

[HKWW17] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *International Conference on Computer Aided Verification*, pages 3–29. Springer, 2017.

[HLŽ21] Thomas A Henzinger, Mathias Lechner, and Đorđe Žikelić. Scalable verification of quantized neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 3787–3795, 2021.

[Hoe94] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. In *The collected works of Wassily Hoeffding*, pages 409–426. Springer, 1994.

[HR04] Michael Huth and Mark Ryan. *Logic in Computer Science: Modelling and reasoning about systems.* Cambridge university press, 2004.

[HYY+15] Guosheng Hu, Yongxin Yang, Dong Yi, Josef Kittler, William Christmas, Stan Z Li, and Timothy Hospedales. When face recognition meets with deep learning: an evaluation of convolutional neural networks for face recognition. In *International conference on computer vision workshops*, pages 142–150. IEEE, 2015.

[KAH^{+}19] Faiq Khalid, Hassan Ali, Muhammad Abdullah Hanif, Semeen Rehman, Rehan Ahmed, and Muhammad Shafique. RED-Attack: Resource Efficient Decision based Attack for Machine Learning. *arXiv preprint arXiv:1901.10258*, 2019.

[KB15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015*, 2015.

[KBD^{+}17] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 97–117. Springer, 2017.

[Kea18] Shujaat Khan et al. A novel fractional gradient-based learning algorithm for recurrent neural networks. ***CSSP***, 37(2):593–612, 2018.

[Ken19] Yoed N Kenett. What can quantitative measures of semantic distance tell us about creativity? *Current Opinion in Behavioral Sciences*, 27:11–16, 2019.

[KFS21] Haitham Khedr, James Ferlez, and Yasser Shoukry. Peregrinn: Penalized-relaxation greedy neural network verifier. In *International Conference on Computer Aided Verification*, pages 287–300. Springer, 2021.

[KGB17a] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. In *International Conference on Learning Representations, ICLR*, pages 1–14, 2017.

[KGB17b] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. In *International Conference on Learning Representations, ICLR*, pages 1–17, 2017.

[KHI^{+}19] Guy Katz, Derek A Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, et al. The marabou framework for verification and analysis of deep neural networks. In *International Conference on Computer Aided Verification*, pages 443–452. Springer, 2019.

[KHR^{+}19] Faiq Khalid, Muhammad Abdullah Hanif, Semeen Rehman, Rehan Ahmed, and Muhammad Shafique. TrISec: Training Data-Unaware Imperceptible Security Attacks on Deep Neural Networks. In *On-Line Testing and Robust System Design (IOLTS)*. IEEE/ACM, 2019.

[KKBC12] Volodymyr Kuznetsov, Johannes Kinder, Stefan Bucur, and George Candea. Efficient state merging in symbolic execution. *Acm Sigplan Notices*, 47(6):193–204, 2012.

[KKK$^+$19] Byungju Kim, Hyunwoo Kim, Kyungsu Kim, Sungjin Kim, and Junmo Kim. Learning not to learn: Training deep neural networks with biased data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9012–9020, 2019.

[KKP$^+$06] Sotiris Kotsiantis, Dimitris Kanellopoulos, Panayiotis Pintelas, et al. Handling imbalanced datasets: A review. *GESTS international transactions on computer science and engineering*, 30(1):25–36, 2006.

[KNP18] Marta Kwiatkowska, Gethin Norman, and David Parker. Probabilistic model checking: Advances and applications. *Formal System Verification: State-of the-Art and Future Trends*, pages 73–121, 2018.

[LBSB$^+$20] Ronan Le Bras, Swabha Swayamdipta, Chandra Bhagavatula, Rowan Zellers, Matthew Peters, Ashish Sabharwal, and Yejin Choi. Adversarial filters of dataset biases. In *International Conference on Machine Learning*, pages 1078–1088. PMLR, 2020.

[Lea18] J. L. Leevy et al. A survey on addressing high-class imbalance in big data. ***J. Big Data***, 5(1):1–30, 2018.

[Lea20] J. Liu et al. Deep representation learning on long-tailed data: A learnable embedding augmentation perspective. In ***Proc. CVPR***, pages 2970–2979, 2020.

[LFZ21] Yicheng Luo, Antonio Filieri, and Yuan Zhou. Symbolic parallel adaptive importance sampling for probabilistic program analysis. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1166–1177, 2021.

[LLT$^+$19] Axel Legay, Anna Lukina, Louis Marie Traonouez, Junxing Yang, Scott A. Smolka, and Radu Grosu. *Statistical Model Checking*, pages 478–504. Springer International Publishing, 2019.

[LLV18] Yingwei Li, Yi Li, and Nuno Vasconcelos. Resound: Towards action recognition without representation bias. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 513–528, 2018.

[LLY$^+$19] Jianlin Li, Jiangchao Liu, Pengfei Yang, Liqian Chen, Xiaowei Huang, and Lijun Zhang. Analyzing deep neural networks with symbolic propagation: Towards higher precision and faster verification. In *International static analysis symposium*, pages 296–319. Springer, 2019.

[LM17] Alessio Lomuscio and Lalit Maganti. An approach to reachability analysis for feed-forward ReLU neural networks. *arXiv preprint arXiv:1706.07351*, 2017.

[LNA17] Guillaume Lemaître, Fernando Nogueira, and Christos K Aridas. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *The Journal of Machine Learning Research*, 18(1):559–563, 2017.

[LPS15] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: A llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, pages 1–6, 2015.

[LV19] Yi Li and Nuno Vasconcelos. Repair: Removing representation bias by dataset resampling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9572–9581, 2019.

[LYC$^+$19] Wang Lin, Zhengfeng Yang, Xin Chen, Qingye Zhao, Xiangkun Li, Zhiming Liu, and Jifeng He. Robustness verification of classification deep neural networks via linear programming. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11418–11427, 2019.

[MDFFF17] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *Proc. CVPR*, pages 1765–1773, 2017.

[Mea01] G. B. Moody et al. The impact of the mit-bih arrhythmia database. ***Eng. Med. Biol. Mag.***, 20(3):45–50, 2001.

[Mey23] Pierre-Jean Meyer. Reachability analysis of neural networks with uncertain parameters. *arXiv preprint arXiv:2303.07917*, 2023.

[MFVRT20] Aythami Morales, Julian Fierrez, Ruben Vera-Rodriguez, and Ruben Tolosana. Sensitivenets: Learning agnostic representations with application to face images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(6):2158–2164, 2020.

[MGS17] Charlie Murphy, Patrick Gray, and Gordon Stewart. Verified perceptron convergence theorem. In *SIGPLAN International Workshop on Machine Learning and Programming Languages*, pages 43–50. ACM, 2017.

[Mil82] R. Milner. *A Calculus of Communicating Systems*. Springer-Verlag, Berlin, Heidelberg, 1982.

[MMS$^+$21] Ninareh Mehrabi, Fred Morstatter, Nripsuta Saxena, Kristina Lerman, and Aram Galstyan. A survey on bias and fairness in machine learning. *ACM Computing Surveys (CSUR)*, 54(6):1–35, 2021.

[MMS$^+$22] Mark Niklas Müller, Gleb Makarchuk, Gagandeep Singh, Markus Püschel, and Martin T Vechev. Prima: general and precise neural network certification via scalable convex hull approximations. *Proc. ACM Program. Lang.*, 6(POPL):1–33, 2022.

[MNG$^+$21] Xingjun Ma, Yuhao Niu, Lin Gu, Yisen Wang, Yitian Zhao, James Bailey, and Feng Lu. Understanding adversarial attacks on deep learning based medical image analysis systems. *Pattern Recognition*, 110:107332, 2021.

[MSB22] Samvid Mistry, Indranil Saha, and Swarnendu Biswas. An milp encoding for efficient verification of quantized deep neural networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(11):4445–4456, 2022.

[NCA$^+$20] Junhyun Nam, Hyuntak Cha, Sungsoo Ahn, Jaeho Lee, and Jinwoo Shin. Learning from failure: De-biasing classifier from biased classifier. In *Advances in Neural Information Processing Systems*, volume 33, pages 20673–20684, 2020.

[NDS$^+$21] Vedant Nanda, Samuel Dooley, Sahil Singla, Soheil Feizi, and John P Dickerson. Fairness Through Robustness: Investigating Robustness Disparity in Deep Learning. In ***FAccT***, pages 466–477, 2021.

[NH10] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning (ICML)*, pages 807–814, 2010.

[NKR$^+$18] Nina Narodytska, Shiva Kasiviswanathan, Leonid Ryzhyk, Mooly Sagiv, and Toby Walsh. Verifying properties of binarized deep neural networks. In *AAAI Conference on Artificial Intelligence*, 2018.

[NSM$^+$19] Nina Narodytska, Aditya Shrotri, Kuldeep S Meel, Alexey Ignatiev, and Joao Marques-Silva. Assessing heuristic machine learning explanations with model counting. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 267–278. Springer, 2019.

[OBK22] Matan Ostrovsky, Clark Barrett, and Guy Katz. An abstraction-refinement approach to verifying convolutional neural networks. In *Automated Technology for Verification and Analysis: 20th International Symposium, ATVA 2022, Virtual Event, October 25–28, 2022, Proceedings*, pages 391–396. Springer, 2022.

[PCFG20] Corina Păsăreanu, Hayes Converse, Antonio Filieri, and Divya Gopinath. On the probabilistic analysis of neural networks. In *Proceedings of the IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 5–8, 2020.

[PCYJ17] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Deepxplore: Automated whitebox testing of deep learning systems. In *Symposium on Operating Systems Principles*, pages 1–18. ACM, 2017.

[PHJ$^+$19] Stjepan Picek, Annelie Heuser, Alan Jovic, Shivam Bhasin, and Francesco Regazzoni. The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(1):1–29, 2019.

[PRA19] Pavithra Prabhakar and Zahra Rahimi Afzal. Abstraction based output range analysis for neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

[PT10] Luca Pulina and Armando Tacchella. An abstraction-refinement approach to verification of artificial neural networks. In *International Conference on Computer Aided Verification*, pages 243–257. Springer, 2010.

[PT12] Luca Pulina and Armando Tacchella. Challenging SMT solvers to verify neural networks. *Ai Communications*, 25(2):117–135, 2012.

[PWW20] Brandon Paulsen, Jingbo Wang, and Chao Wang. Reludiff: Differential verification of deep neural networks. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 714–726. IEEE, 2020.

[Ric69] Daniel Richardson. Some undecidable problems involving elementary functions of a real variable. *The Journal of Symbolic Logic*, 33(4):514–520, 1969.

[Ros57] F Rosenblatt. The perceptron–a perceiving and recognizing automation. *Report 85-460-1 Cornell Aeronautical Laboratory, Ithaca, Tech. Rep.*, 1957.

[Ros61] Frank Rosenblatt. Principles of neurodynamics: Perceptrons and the theory of brain mechanisms. Technical report, Cornell Aeronautical Lab Inc Buffalo NY, 1961.

[RRCST19] Vicenc Rubies-Royo, Roberto Calandra, Dusan M Stipanovic, and Claire Tomlin. Fast neural network verification via shadow prices. *arXiv preprint arXiv:1902.07247*, 2019.

[RTVG18] Rasmus Rothe, Radu Timofte, and Luc Van Gool. Deep expectation of real and apparent age from a single image without facial landmarks. *International Journal of Computer Vision*, 126(2-4):144–157, 2018.

[SBG$^+$19] Luiz H Sena, Iury V Bessa, Mikhail R Gadelha, Lucas C Cordeiro, and Edjard Mota. Incremental bounded model checking of artificial neural networks in cuda. In *2019 IX Brazilian Symposium on Computing Systems Engineering (SBESC)*, pages 1–8. IEEE, 2019.

[SDC19a] Andy Shih, Adnan Darwiche, and Arthur Choi. Verifying binarized neural networks by angluin-style learning. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 354–370. Springer, 2019.

[SDC19b] Andy Shih, Adnan Darwiche, and Arthur Choi. Verifying binarized neural networks by local automaton learning. In *AAAI Spring Symposium on Verification of Neural Networks (VNN)*, 2019.

[Sea21] D. Samuel et al. Distributional robustness loss for long-tail learning. In ***Proc. ICCV***, pages 9495–9504, 2021.

[SGM$^+$18] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. Fast and Effective Robustness Certification. In *Advances in Neural Information Processing Systems*, pages 10802–10813, 2018.

[SGPV19] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An Abstract Domain for Certifying Neural Networks. *Proceedings of the ACM on Programming Languages*, 3(POPL):41, 2019.

[SMS$^+$21] Xidan Song, Edoardo Manino, Luiz Sena, Erickson Alves, Iury Bessa, Mikel Lujan, Lucas Cordeiro, et al. Qnnverifier: A tool for verifying neural networks using smt-based model checking. *arXiv preprint arXiv:2111.13110*, 2021.

[SSS$^+$19] Md Amran Siddiqui, Jack W Stokes, Christian Seifert, Evan Argyle, Robert McCann, Joshua Neil, and Justin Carroll. Detecting Cyber Attacks Using Anomaly Detection with Explanations and Expert Feedback. In *Proc. ICASSP*, pages 2872–2876. IEEE, 2019.

[ST19] Matthew Sotoudeh and Aditya V Thakur. Correcting deep neural networks with small, generalizing patches. In *Workshop on safety and robustness in decision making*, 2019.

[StT$^+$20] Muhammad Shafique, **Mahum Naseer**, Theocharis Theocharides, Christos Kyrkou, Onur Mutlu, Lois Orosa, and Jungwook Choi. Robust Machine Learning Systems: Challenges, current Trends, Perspectives, and the Road ahead. *IEEE Design & Test*, 37(2):30–57, 2020. *[Mahum Naseer contributed to the Formal Verification and Empirical Analysis aspects of the work.]*

[STT23] Matthew Sotoudeh, Zhe Tao, and Aditya V Thakur. Syrenn: A tool for analyzing deep neural networks. *International Journal on Software Tools for Technology Transfer*, pages 1–21, 2023.

[SWBR20] Victor Sanh, Thomas Wolf, Yonatan Belinkov, and Alexander M Rush. Learning from others' mistakes: Avoiding dataset biases without modeling them. *arXiv preprint arXiv:2012.01300*, 2020.

[SWG20] Yash Savani, Colin White, and Naveen Sundar Govindarajulu. Intra-processing methods for debiasing neural networks. *Advances in Neural Information Processing Systems*, 33, 2020.

[SWWB15] Karsten Scheibler, Leonore Winterer, Ralf Wimmer, and Bernd Becker. Towards Verification of Artificial Neural Networks. In *MBMV*, pages 30–40, 2015.

[SZEH21] Md Kamruzzaman Sarker, Lu Zhou, Aaron Eberhart, and Pascal Hitzler. Neuro-symbolic artificial intelligence: Current trends. *arXiv preprint arXiv:2105.05330*, 2021.

[SZS+13] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[TDG09] Ninad Thakoor, Venkat Devarajan, and Jean Gao. Computation complexity of branch-and-bound model selection. In *International Conference on Computer Vision*, pages 1895–1900. IEEE, 2009.

[TKJ+21] Qi Tian, Kun Kuang, Kelu Jiang, Fei Wu, and Yisen Wang. Analysis and applications of class-wise robustness in adversarial training. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1561–1570, 2021.

[TLM+19] Hoang-Dung Tran, Diago Manzanas Lopez, Patrick Musau, Xiaodong Yang, Luan Viet Nguyen, Weiming Xiang, and Taylor T Johnson. Star-Based Reachability Analysis of Deep Neural Networks. In *International Symposium on Formal Methods*, pages 670–686. Springer, 2019.

[tMK+20] **Mahum Naseer**, Mishal Fatima Minhas, Faiq Khalid, Muhammad Abdullah Hanif, Osman Hasan, and Muhammad Shafique. FANNet: Formal Analysis of Noise Tolerance, Training Bias and Input Sensitivity in Neural Networks. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 666–669. IEEE, 2020.

[TPL+21] Hoang-Dung Tran, Neelanjana Pal, Diego Manzanas Lopez, Patrick Musau, Xiaodong Yang, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, and Taylor T Johnson. Verification of piecewise deep neural networks: a star set approach with zonotope pre-filter. *Formal Aspects of Computing*, 33(4):519–545, 2021.

[TPM+21] Hoang-Dung Tran, Neelanjana Pal, Patrick Musau, Diego Manzanas Lopez, Nathaniel Hamilton, Xiaodong Yang, Stanley Bak, and Taylor T Johnson. Robustness verification of semantic segmentation neural networks using relaxed reachability. In *International Conference on Computer Aided Verification*, pages 263–286. Springer, 2021.

[TXT19] Vincent Tjeng, Kai Y Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. In *International Conference on Learning Representations (ICLR)*, 2019.

[TYML$^+$20] Hoang-Dung Tran, Xiaodong Yang, Diego Manzanas Lopez, Patrick Musau, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, and Taylor T Johnson. Nnv: the neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In *International Conference on Computer Aided Verification*, pages 3–17. Springer, 2020.

[UBMS23] Shubham Ugare, Debangshu Banerjee, Sasa Misailovic, and Gagandeep Singh. Incremental verification of neural networks. *arXiv preprint arXiv:2304.01874*, 2023.

[Wea14] M.-Ju Wu et al. Wafer map failure pattern recognition and similarity ranking for large-scale data sets. ***Trans. Semicond. Manuf.***, 28(1):1–12, 2014.

[Wea17] Y.-Xiong Wang et al. Learning to model the tail. In et al. I. Guyon, editor, ***Proc. NeurIPS***, volume 30. Curran Associates, Inc., 2017.

[WOZ$^+$20] Haoze Wu, Alex Ozdemir, Aleksandar Zeljić, Kyle Julian, Ahmed Irfan, Divya Gopinath, Sadjad Fouladi, Guy Katz, Corina Pasareanu, and Clark Barrett. Parallelization techniques for verifying neural networks. In *Proc. FMCAD*, pages 128–137, 2020.

[WPW$^+$18a] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Efficient formal safety analysis of neural networks. In *Advances in Neural Information Processing Systems*, pages 6367–6377, 2018.

[WPW$^+$18b] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Formal security analysis of neural networks using symbolic intervals. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 1599–1614, 2018.

[WQK$^+$20] Zeyu Wang, Klint Qinami, Ioannis Christos Karakozis, Kyle Genova, Prem Nair, Kenji Hata, and Olga Russakovsky. Towards fairness in visual recognition: Effective strategies for bias mitigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8919–8928, 2020.

[WRTK19] Stefan Webb, Tom Rainforth, Yee Whye Teh, and M. Pawan Kumar. Statistical verification of neural networks. In *International Conference on Learning Representations*, 2019.

[WTY09] Shuo Wang, Ke Tang, and Xin Yao. Diversity exploration and negative correlation learning on imbalanced data sets. In *2009 International joint conference on neural networks*, pages 3259–3266. IEEE, 2009.

[WX18] Rey Wiyatno and Anqi Xu. Maximal Jacobian-based Saliency Map Attack. *arXiv preprint arXiv:1808.07945*, 2018.

[XLL+21] Han Xu, Xiaorui Liu, Yaxin Li, Anil Jain, and Jiliang Tang. To be robust or to be fair: Towards fairness in adversarial training. In *International Conference on Machine Learning*, pages 11492–11501. PMLR, 2021.

[XTJ17] Weiming Xiang, Hoang-Dung Tran, and Taylor T Johnson. Reachable set computation and safety verification for neural networks with ReLU activations. *arXiv preprint arXiv:1712.08163*, 2017.

[XTJ18] Weiming Xiang, Hoang-Dung Tran, and Taylor T Johnson. Output Reachable Set Estimation and Verification for Multilayer Neural Networks. *IEEE transactions on neural networks and learning systems*, 29(11):5777–5783, 2018.

[Yea19] X. Yin et al. Feature transfer learning for face recognition with under-represented data. In ***Proc. CVPR***, pages 5704–5713, 2019.

[YLL+21a] Pengfei Yang, Jianlin Li, Jiangchao Liu, Cheng-Chao Huang, Renjue Li, Liqian Chen, Xiaowei Huang, and Lijun Zhang. Enhancing robustness verification for deep neural networks via symbolic propagation. *Formal Aspects of Computing*, 33(3):407–435, 2021.

[YLL+21b] Pengfei Yang, Renjue Li, Jianlin Li, Cheng-Chao Huang, Jingyi Wang, Jun Sun, Bai Xue, and Lijun Zhang. Improving neural network verification through spurious region guided refinement. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 389–408. Springer, 2021.

[Zea21] Y. Zhang et al. Bag of tricks for long-tailed visual recognition with deep convolutional neural networks. In ***Proc. AAAI***, volume 35, pages 3447–3455, 2021.

[ZHG06] Feng Zhao, Qingming Huang, and Wen Gao. Image matching by normalized cross-correlation. In *International Conference on Acoustics Speech and Signal Processing Proceedings*, volume 2, pages II–II. IEEE, 2006.

[ZL19] Ziqi Zhang and Lei Luo. Hate speech detection: A solved problem? the challenging case of long tail on twitter. *Semantic Web*, 10(5):925–945, 2019.

[ZMC94] Jacek M Zurada, Aleksander Malinowski, and Ian Cloete. Sensitivity analysis for minimization of input data dimension for feedforward neural network. In *Proceedings of IEEE International Symposium on Circuits and Systems*, volume 6, pages 447–450. IEEE, 1994.

[ZRM20] Erfan Zangeneh, Mohammad Rahmati, and Yalda Mohsenzadeh. Low resolution face recognition using a two-branch deep convolutional neural network architecture. *Expert Syst. with Applications*, 139:112854, 2020.

[ZSAT19] Ameema Zainab, Dabeeruddin Syed, and Dena Al-Thani. Deployment of deep learning models to mobile devices for spam classification. In *2019 IEEE First International Conference on Cognitive Machine Intelligence (CogMI)*, pages 112–117. IEEE, 2019.

[ZWL+19] Yundong Zhang, Hang Wu, Huiye Liu, Li Tong, and May D Wang. Improve model generalization and robustness to dataset bias with bias-regularized learning and domain-guided augmentation. *arXiv preprint arXiv:1910.06745*, 2019.

[ZWY+17] Jieyu Zhao, Tianlu Wang, Mark Yatskar, Vicente Ordonez, and Kai-Wei Chang. Men also like shopping: Reducing gender bias amplification using corpus-level constraints. In *Conference on Empirical Methods in Natural Language Processing*, pages 2979–2989, Copenhagen, Denmark, 2017. Association for Computational Linguistics.

[ZZC+21] Yedi Zhang, Zhe Zhao, Guangke Chen, Fu Song, and Taolue Chen. Bdd4bnn: a bdd-based quantitative analysis framework for binarized neural networks. In *International Conference on Computer Aided Verification*, pages 175–200. Springer, 2021.